Sicherheit für Websites

Aus eigener Erfahrung weiß ich, dass selbst in hochwertigen Anwendungen immer wieder sehr "billige" Sicherheitslücken eingebaut sind, da die Programmierer nicht die Zeit haben, sich mit den Sicherheitsaspekten von PHP zu beschäftigen. Dieser Artikel soll helfen, das Sicherheitsbewusstsein zu stärken.

Vorbereitung

Worum gehts?

Der Untertitel dieses Artikels könnte lauten: "PHP/MySQL-Anwendungen hacken", da im Folgenden typische Angriffe auf eben solche Applikationen beschrieben werden. Mein Hauptaugenmerk möchte ich aber auf deren Verhinderung legen. Aus eigener Erfahrung weiß ich, dass selbst in Business-Anwendungen immer wieder sehr "billige" Sicherheitslücken eingebaut sind, da die Programmierer nicht die Zeit haben, sich mit den Sicherheitsaspekten von PHP zu beschäftigen. Aber auch in dem Code erfahrener Entwickler tauchen zwischen schönen Templatesystem und übersichtlichen aber funktionalen Klassenstrukturen immer wieder böse Zeilen

auf, die einem Angreifer Tür und Tor öffnen.

Einführung

Die beschriebenen Attacken lassen sich - bei einer fehlerhaften Implementierung - auf diversen Systemkonfigurationen ausführen. In den Ausführungen wird von PHP in Verbindung mit MySQL ausgegangen, eine häufig anzutreffende Konfiguration.

Generell ein guter Anlaufspunkt bei PHP-Fragen und auch bei Sicherheitsfragen im Speziellen ist die PHP-FAQ.

GET und POST

Diese beiden sog. Methoden stellen Wege zur Verfügung, auf denen der Browser diverse Daten an eine Website übermitteln kann. Vom Programmierer werden hier meist wenige Werte vorgesehen, in Wahrheit steht es dem Browser und damit dem Benutzer, der ihn kontrolliert, aber völlig frei, welche Daten er übermittelt. Zur Manipulation der gesendeten Daten reicht es bei der GET-Methode, die URL zu verändern; aus `http://www.example.com/index.php?name=Peter&nummer=15` kann ein Benutzer ohne besondere Fertigkeiten innerhalb von Sekunden `http://www.example.com/index.php?name=&nummer=TEXT` machen und damit evtl. schon eine Sicherheitslücke ausnutzen.

Es ist wichtig zu wissen, dass in einer URL nicht alle Zeichen erlaubt sind. Wir gehen daher davon aus, dass der verwendete Browser Sonderzeichen - sofern nötig - automatisch in ihren Zeichencode umwandelt. Um die Lesbarkeit zu verbessern werde ich daher im Folgenden auf diese Zeichencodes verzichten. Bei Bedarf kann man unter Stichworten wie "URL encoding" bei Google entsprechende Informationen finden.

JavaScript-Witze

Bilderschutz

Eigentlich gehört das, was jetzt kommt, nicht wirklich hier hin. Es soll mehr als warnendes Beispiel dienen, wie wenig der Webseitenersteller den Browser eines Benutzers wirklich unter Kontrolle hat und verdeutlichen, dass JavaScript sich nicht für Sicherheitsrelevante Dinge einsetzen lässt.

Es gibt Seiten im Netz, auf denen darf man keinen Rechtsklick einsetzen. Entweder der Browser reagiert einfach nicht auf den Klick oder es kommt ein Fenster was soviel sagt wie "Rechtsklick verboten!", "Alle Bilder (c) 2022 by ME!" oder ähnliches. Damit will der Ersteller der Seite verhindern, dass man Bilder mit Rechtsklick -> "Speichern unter" auf der lokalen Festplatte abspeichert.

Was dabei jedoch übersehen wird: In dem Moment, in dem ich das Bild sehe, ist es bereits auf meiner Festplatte. Nämlich im Cache-Verzeichnis des Browsers. Außerdem muss das Bild ja nach wie vor abrufbar sein, es muss eine URL haben und damit ist es auch direkt erreichbar; Meist reicht es, das Bild einfach in die Adressleiste des Browsers zu ziehen und es erscheint komplett ungeschützt im Fenster. Auch das liesse sich verhindern, aber in allen Browser kann man auch JavaScript ausschalten.

Im Firefox gibt es eine ganz besonders einfache Möglichkeit, alle Bilder bequem abzuspeichern: Extras, Seiteninformationen, Medien. Schwupps sieht der Benutzer alle Bilder und kann sie bequem abspeichern.

> Wer an ein Bild ran will, kriegt es auch!

Diese Art von Pseudo-Schutz ist nur ein Ärgernis für Benutzer, die mit dem Rechtsklick navigieren (entweder Rechtsklick -> zurück oder mit Mausgesten). Sie zeugt von Inkompetenz seitens des Erbauers der Page.

Formularschutz

Oftmals werden Formulare im Web von JavaScript auf deren Validität, also das korrekte Ausfüllen durch den Benutzer geprüft. Das ist sinnvoll, weil man so sofort nach Klick auf den Absenden-Button ein Feedback erhält, ob man nichts vergessen hat.

> Niemals auf JavaScript vertrauen!

Aber diese Prüfung darf keinesfalls die einzige sein, bevor die Daten z.B. von PHP-Code verarbeitet werden. Es muss immer zusätzlich in PHP oder mit vergleichbaren Mitteln *serverseitig* geprüft werden! Denn wie oben schon gilt: JavaScript kann ausgeschaltet sein oder man kann auf Browser treffen, die gar kein JavaScript können.

JavaScript-Zugriffsschutz

Du willst alleine mit JavaScript eine Passwortabfrage für eine Seite erstellen? Vergiss es, keine Chance. Auch wenn das richtige Passwort noch so kompliziert "verschlüsselt" im Code auftaucht, es ist leicht rauszufinden. Nur .htaccess oder sichere Lösungen in serverseitigen Sprachen erfüllen hier ihren Zweck. Alles andere ist sicherheitstechnische Homöopathie.

Mailformulare missbrauchen

Die zweitbekannteste Anwendung des Internets ist nach dem World Wide Web wohl E-Mail. Und beide tauchen dort gerne zusammen auf, wo der Seitenersteller sich z.B. ein Feedback von seinen Benutzern erhofft oder Dienste wie "Seite weiterempfehlen" angeboten werden. (Meiner Meinung nach eine der überflüssigsten Funktionen überhaupt, aber egal.)

Im Grunde stellen diese Formulare damit meistens nur eine einfache Schnittstelle zum mail-Befehl von PHP dar. Um die dabei oft entstehende Sicherheitslücke zu kennen, muss man ein bisschen über Header bei E-Mails wissen:

E-Mail-Header bestehen, wie man leicht z.B. im Thunderbird E-Mail-Client mit Strg+U oder in Outlook-Express mit Strg+F3 nachschauen kann, aus mehreren Zeilen, die jeweils mit einem Schlüsselwort anfangen und durch Absatzzeichen getrennt sind, z.B.

```
To: frosch@see.de
From: info@example.com
Subject: Ich bin eine Betreffzeile
```

Die Formulare bieten in der Regel eine Möglichkeit, dass der Absender seine eigene Adresse eintippt und die Formularinhalte unter seinem Namen irgendwohin geschickt werden.

In PHP sieht das dann z.B. so aus:

```
mail($to, $subject, $text, "From: ".$from);
```

Dabei kommen $to, $subject, $text und $from direkt aus dem Formular. Im mail-Befehl kann man als viertes Argument beliebige Header angeben. Der Entwickler hat also an dieser Stelle einen From-Header, gefüllt mit der Eingabe des Benutzers, vorgsehen.

Ein Angreifer könnte sich dies folgendermaßen zunutze machen: Er sendet für $from den Wert (ohne Anführungszeichen) "test@example.com\n\rBcc: spam-empfaenger@example.com". Wobei \n und \r hier natürlich für die Zeichen mit den ASCII-Codes 13 und 10 stehen. Zusammen ergeben sie einen Zeilenumbruch (nach Windows-Art).

Damit sorgt er dafür, dass an der Stelle, an der eigentlich nur ein From-Header stehen sollte, nun auch ein Bcc-Header steht. Mit diesem kann er den Inhalt der Mail an viele, beliebig wählbare Empfänger schicken - und damit die Server-Resourcen missbrauchen und dem Inhaber evtl. auch Ärger einbringen.

Um das zu verhindern, genügt es im Grunde, aus sämtlichen Eingaben die Absatzzeichen auszufiltern, z.B. mit

```
$checkedFrom = str_replace(array("\n", "\r"), '', $from);
```

register_globals und was es bewirkt

Es gibt in der PHP.INI die Einstellung register_globals. Sie bewirkt, wie es in älteren PHP-Versionen auch Standard war, den Import sämtlicher externer Daten in den globalen Namensraum des Skripts. Alle Daten aus Cookies, Get, Post, Umgebung (Environment) und Sessions werden also als Variablen der Form $name einfach so bereitgestellt. Üblicherweise sollte der Zugriff auf diese Variablen immer über $_COOKIE, $_GET, $_POST(, $_REQUEST) und $_SERVER erfolgen!

Abschalten - 1

Diese Einstellung sollte dringen auf "off" stehen. Warum? Weil sonst - bei unsauberer Programmierung - dein Skript beeinflußt werden könnte an Stellen wo dies nicht gewünscht oder sogar gefährlich ist. Beispiel:

```
<?php
if ($passwort == "okay")
{
    $loggedIn = true;
}
.
.
.
if ($loggedIn == true)
{
    ...
```

```
?>
```

In diesem Beispiel wird also eine Variable $loggedIn auf true gesetzt, sobald das Passwort stimmt. Stimmt es nicht, wird die Variable nicht gesetzt. Weiter unten wird dann geprüft ob die Variable auf true gesetzt ist - und dann davon ausgegangen, der User sei eingeloggt.

Bei ordentlicher Programmierung wurde am Anfang des Skriptes error_reporting auf E_ALL gesetzt - damit käme bei Ausführung des unteren Teils eine Notice-Meldung, wenn der Benutzer nicht eingeloggt ist. Denn es wird dann ja eine Variable geprüft, die nicht vorher gesetzt wurde. Und dies könnte sich ein Hacker zunutze machen, indem er die PHP-Datei so aufruft:

```
index.php?loggedIn=1
```

- und schon wäre er drin. Also:

> error_reporting(E_ALL) nervt, aber wirkt!

Stets Variablen vor der Benutzung einen definierten Wert zuweisen und vor allem

> register_globals immer auf "off" stellen!

Gewöhnt man sich beides an, verbessert sich auch automatisch der Programmierstil :-)

Abschalten - 2

Aus einem ähnlichen Grund sollte man nie Session-Daten über eine mit register_globals registrierte Variable prüfen. Denn man weißt nicht sicher, ob die Variable wirklich Daten aus der Session enthält (denen man trauen kann) oder ob sie nicht doch eine verkappte GET- oder POST-Variable ist und damit evtl. manipulierte Inhalte hat.

Cookie-Daten-Integrität

Nur weil der normale Benutzer selten bis nie etwas von den Cookies einer Seite mitbekommt, heisst das für den Programmierer nicht, dass er den dort abgelegten Daten trauen darf.

> Traue niemandem! Prüfe alle Eingaben!

(Diese Warnung ist zugegebenermaßen geklaut aus der PHP-FAQ, aber sehr eingängig!)

Es ist für den Benutzer ein leichtes (z.B. mit entsprechenden FireFox-Plugins) die Daten in den Cookies zu verändern. Daher sollte man dort nie z.B. den Login-Status, die eingeloggte User-ID oder Passwörter speichern! Dafür gibt es Sessions, die erledigen das auf alle Fälle für dich.

eval is evil!

Das folgende Code-Fragment hätte ich nie zu sehen bekommen, wenn es nicht genau dort gewesen wäre wo es war. Gleichzeitig ist es eines der fatalsten Code-Fragmente in Bezug auf PHP-Sicherheit:

```
<?php
foreach($HTTP_GET_VARS as $key => $value) {
    eval("$$key = \"$value\";");
}
?>
```

Was genau der Autor damit anstellen wollte, wird wohl ewig im Dunkeln bleiben. Fakt ist aber dass er mich damit dazu eingeladen hat, beliebige PHP-Befehle direkt aus der Adressleiste auszuführen. Als Beispiel stelle man sich vor, $value enthielte `";php_info();//` womit man schon mal eine schöne Übersicht über die PHP-Installation bekommt. Das lässt sich dann mit ein wenig Kreativität und ein paar PHP-Funktionen ausbauen zu einer Dateiübersicht, der Anzeige von Dateien (z.B. der Datei, die eben diese Lücke enthält), dem Löschen von Dateien, Spam-Versand... etc.pp.. Von daher:

> eval vermeiden wo immer möglich

(wie z.B. in dem gezeigten Fall, wo das völlig überflüssig eingesetzt wurde) und vor allem

> NIE NICHT NIE ungeprüfte Eingaben in eval's packen!

denn

> EVAL is EVIL!

und es geht eben meist auch ohne...

MySQL-Injections

...sind eigentlich ganz einfach, eine entsprechend nachlässige Programmierung vorausgesetzt.

Für Anfänger...

Ein Beispiel: Angnommen wir haben ein ganz normales Login-Formular, also so mit Benutzername und Passwort und so. Damit erhält der Benutzer, der eine richtige Kombination aus Passwort und Benutzername eintippt Zugang zu einem geschützten Bereich. Der Programmierer prüft das folgendermaßen: (`$_REQUEST['passwort']` und `$_REQUEST['benutzername']` enthalten die Daten aus dem Login-Formular)

```
mysql_query('SELECT * FROM users WHERE passwort = "'.$_REQUEST['passwort'].'" AND username =
"'.$_REQUEST['benutzername'].'"');
```

Hier wird geprüft, ob eine Kombination aus Benutzername und Passwort in der Datenbank sind. Auch wenn man mal davon absieht, dass Passwörter nie in Klartext in eine Datenbank gehören wird hier eine Einladung für Angreifer ausgesprochen. Als Beispiel für einen Hack sei die Standard-MySQL-Injection für Benutzername und Passwort genannt: `" OR 1 OR "` in beide Felder eintippen und man ist drin. Geht mit erstaunlich vielen Seiten! Alternativ natürlich noch einfache Anführungszeichen statt doppelten.

Netterweise ist der Account, der dabei SELECTet wird, meistens auch noch ein Administrator-Account...

...und Fortgeschrittene.

Etwas mehr Kreativität seitens des Eindringlings erfordert folgendes Beispiel, welches im Ernstfall aber auch wesentlich größere Konsequenzen hat.

Ausgegangen wird von einem Formular in dem ein Benutzer seine ICQ-Nummer ändern kann. `$_REQUEST['icq']` enthalte daher seine Eingabe, für $username sei bereits sichergestellt dass der richtige Benutzername enthalten ist.

```
mysql_query('UPDATE users SET icqNummer = "'.$_REQUEST['icq'].'" WHERE username = "'.$username.'"');
```

Damit hat der Programmierer einem Angreifer bereits alle Möglichkeiten in die Hand gegeben, die Daten in der Datenbank beliebig zu verändern. Ein Beispiel? Bitte:

```
", is_admin="1
```

eingeben und schon hat man Admin-Rechte, vorausgesetzt der Programmierer hat eine Spalte is_admin vorgesehen, die angibt ob ein Benutzer Administratorrechte hat. Da allerdings, wenn überhaupt solche Spalten vorhanden sind, diese meist nicht all zu kreative Namen haben, kann man hier mit ein wenig Herumraten schon oft etwas erreichen.

Will ein Eindringling nur etwas Schaden anrichten, wird er sich daran erinnern, dass man mit # in MySQL Kommentare anfängt. Daher eignet sich folgende Eingabe von ihm, um allen Benutzern Admin-Rechte zu geben:

```
", is_admin="1" #
```

Auch schön, oder?

Will er großen Schaden anrichten, macht er das Ganze in einem DELETE-Statement.

Vorbeugen!

Um solche Angriffe zu verhindern, muss man sich vor allem an

> Traue niemandem! Prüfe alle Eingaben!

erinnern. Also: Numerische Werte z.B. mit (int) zu einem Integer casten, um sicherzustellen dass wirklich keine unerlaubten Zeichen enthalten sind. Z.B. so:

```
$checkedValue = (int)$_REQUEST['value'];
```

Wie man beispielsweise String-Werte und die Rückgabewerte aus Select-Boxen prüft, steht in der PHP-FAQ.

Auf der sicheren Seite ist man insbesondere bei Eingabestrings, wenn man mysql_escape_string() bzw. das neuere mysql_real_escape_string() anwendet. Die oben gezeigten Abfragen in der sicheren Version lauten also:

```
mysql_query('SELECT * FROM users WHERE passwort = "'.mysql_real_escape_string($_REQUEST['passwort']).'" AND username =
"'.mysql_real_escape_string($_REQUEST['benutzername']).'"');
mysql_query('UPDATE users SET icqNummer = "'.mysql_real_escape_string($_REQUEST['icq']).'" WHERE username =
"'.$username.'"');
```

(Wobei man das natürlich alles noch schöner schreiben kann.)

Zombie-Includes

Mein persönlicher Favorit unter den Sicherheitslücken sind aber amoklaufende Includes, die wieder mal nur funktionieren, weil jemand die Eingabedaten nicht richtig prüft.

Finden...

Auffällig werden Seiten mit solchen Schwachstellen in der URL:

```
http://www.example.com/index.php?include=content/readme
```

Sieht "content/readme" nicht verdächtig nach einem Dateinamen aus? Nur ohne Dateiendung. Der Programmierer wird hier also eine Datei includen, die vom aktuellen Ordner aus gesehen im Ordner content liegt und mindestens mit readme anfängt. Es wird wohl eine feste Dateiendung geben, die er immer an den angegebenen Pfad dranhängt. Also könnte obiges Konstrukt z.B. auf /content/readme.txt verweisen.

Sein Code könnte z.B. so aussehen:

```
include($_REQUEST['include'].'.txt');
```

Machen wir ein paar Versuche!

```
http://www.example.com/index.php?include=content/readmeBLABLA
```

sollte eine Datei sein, die nicht existiert. Wenn wir Glück haben, wird eine entsprechende Fehlermeldung ausgegeben.

```
http://www.example.com/index.php?include=index.php%00
```

Könnte, mit etwas Glück, die index.php in sich selber einbinden. Denn: Das %00 in einer URL sendet PHP das ASCII-Zeichen mit Code 0 - und das schneidet Strings ab. Daher wird die Dateiendung, die der Autor hinzufügt, auch abgeschnitten. Was wir nicht wissen ist, ob er vielleicht noch einen Text vorne an den angegebenen Namen anfügt, dann sähe sein Code so aus:

```
include('inc/'.$_REQUEST['include'].'.txt');
```

Dann gehen oben beschrieben Attacken natürlich nicht mehr. Stattdessen müsste man sich in diesem Fall mit .. durch die Verzeichnisstruktur hangeln um so an andere Dateien zu kommen.

Noch viel gefährlicher ist die Tatsache, dass man unter Umständen bei der Version ohne Verzeichnisangabe Code von externen Quellen ausführen kann - damit stehen einem Angreifer endgültig Tür und Tor offen.

...und vermeiden!

Wieder bedeutet die Lösung für dieses Problem nicht viel Aufwand: mit str_replace werden sämtliche Slashes aus der Eingabe abgefangen, zur Sicherheit auch Backslashes und Punkte:

```
$safeInclude = str_replace(array('/', '\/', '.'), $_REQUEST['include']);
```

Natürlich muss die Seite so gebaut sein, dass sie eben ohne diese Zeichen in der übergebenen Variablen auskommt!

Auf Nummer Sicher geht man aber, wenn man statt des Dateinamens nur ein Kürzel übergibt, das mittels eines Arrays den Dateinamen ermittelt.

File-Uploads

Eine weitere Möglichkeit für Hacker, eigenen Code auf fremden Systemen auszuführen bilden immer wieder Dateiuploads. Welche Gefahr fremde Daten auf dem eigenen Server darstellen scheint vielen Programmierern nicht bewusst zu sein. In der Regel werden die Uploads z.B. für eine Bilder-Galerie vorgesehen und der Autor rechnet nicht damit, dass man nicht nur Bilder hochladen kann.

Für einen erfolgreichen Upload eigener Skripte müssen einige Voraussetzungen erfüllt sein:

- Der Autor nimmt in seinem Skript keine Prüfung des Dateiinhalts vor. Dies würde er z.B. durch

Bestimmung der Bildgröße der hochgeladenen Datei sehr sicher erreichen (in PHP getimagesize).

- Der Autor muss fahrlässigerweise den Dateinamen verwenden, der ihm vom Client mitgesendet wird. Dies ist grundsätzlich ein Fehler. Abgesehen davon, dass es alles andere als verlässlich ist, ermöglicht es wie in diesem Falle dem Angreifer umfangreiche Manipulationsmöglichkeiten.

- Der Angreifer muss das Ziel kennen, wohin die Datei nach dem Upload verschoben wird. Außerdem muss dieses unterhalb des document root liegen, also in einem aus dem Web zugänglichen Bereich.

Dann kann der Hacker also eine Datei nach seinen Wünschen hochladen und, da er ihr ja eine individuelle Endung verpassen kann, diese auch ausführen. Denkbar ist auch, dass wenn Punkt 2 und die Einschränkung von Punkt 3 nicht zutreffen, er mittels eines "offen stehenden" includes (Siehe vorangegangenen Abschnitt) eine Datei ausführt, die nicht im document root liegt und die evtl. noch eine Endung wie .gif trägt. Daher ist bei Dateiuploads immer höchste Vorsicht geboten!

Abwehrstrategien

Man kann auch vorsorglich einige Maßnahmen treffen, um es gar nicht erst zum Versuch eines Angriffs kommen zu lassen. Bots erkennen dann nicht mehr, an welcher Stelle sie angreifen müssten und Menschen vergeht schnell die Lust am Herumexperimentieren wenn kein Erfolg zu sehen ist.

Verwirre deinen Gegner!

Das Apache-Modul mod_rewrite ermöglicht es, Aufrufe wie www.example.com/df_maoam auf eine URL wie www.example.com/index.php?f=maoam "zeigen zu lassen". (Vielleicht wirft der ein oder andere an diese Stelle zurecht einen Blick in die Adresszeile seines Browsers.) Bei jedem Aufruf einer Seite wird also der Webserver eine URL, die mit df_ beginnt in einen Aufruf der index.php umwandeln. Für den Browser des Besuchers und den Besucher selbst geschieht das transparent, d.h. sie bekommen nicht mit, dass hier in echt eine andere Seite aufgerufen wird.

Dieses Mapping wird z.B. in einer .htaccess-Datei festgelegt und sieht für das Beispiel so aus:

```
RewriteEngine On
RewriteRule ^df_$ index.php
RewriteRule ^df_([^\.]+)$ index.php?f=$1
```

Der Kenner sieht natürlich, dass es sich bei dem Zeichenwirrwar in der zweiten und dritten Zeile um einen regulären Ausdruck handelt - und wie die gebildet werden findet man nebenan im Regex-Tutorial.

Mit mod_rewrite kann man lustige Sachen machen, aber man verliert auch genau so schnell mal gerne den Überblick, was man da eigentlich gemacht hat. Eine komplette Anleitung gibt es natürlich auch, und die Zitate am Anfang sind ernst zu nehmen!

Täuschen und Tarnen

Ein weiteres Mittel ist das Verstecken von PHP. Nur mit HTML gestaltete Sites sind sicherlich ein wesentlich unattraktiveres Angriffsziel als interaktive. Um PHP zu verstecken sind im Grunde nur zwei Schritte nötig:

- Umstellung der Standard-Dateierweiterung für PHP: Sag deinem Webserver, er solle ab sofort auch z.B. .html-Dateien mit PHP bearbeiten. Wie das genau geht, findet sich z.b. in der PHP-Installationsanleitung. Damit erreichst du, dass deine PHP-Dateien auch .html heissen können und damit nicht mehr interessant aussehen. Natürlich verlangsamt das deinen Server, wenn du wirklich einige HTML-Seiten hast, die gar kein PHP enthalten.

- PHP verstecken: Standardmäßig fügt PHP einen Header mit der verwendeten PHP-Version in die Ausgabe mit ein. Diesen kannst du mit der Einstellung

  ```
  expose_php = On
  ```

  in der PHP.INI deaktivieren.

Beide Maßnahmen sind allerdings nur als kleine zusätzliche Vorkehrungen zu verstehen und bieten nicht all zu viel Sicherheit. Vor allem endet ihre Wirksamkeit dort, wo GET-Parameter in der URL auftauchen, ganz offensichtlich dynamische Inhalte vorliegen oder PHP-Fehlermeldungen angezeigt werden.

Wichtige Einstellungen der PHP.INI

In der Konfigurationsdatei von PHP, der PHP.INI findest du einige Einstellungen, die wesentlich zur Sicherheit beitragen können:

- **register_globals = Off** Wie oben schon mehrmals gezeigt: Diese Einstellung sollte unbedingt so und nicht anders konfiguriert sein.
- **display_errors = Off** Verhindert dass Fehlermeldungen am Bildschirm zusammen mit der Ausgabe des Skriptes angezeigt werden. Diese könnten Pfade, Datenbank(zugangs)daten und ähnliches beinhalten, was für einen Hack nützlich sein könnte. Selbstverständlich sollte diese Einstellung auf deinem *Entwicklungs*system nicht gesetzt sein.
- **error_reporting = E_ALL** auf dem Entwicklungssystem zeigt dir beim Schreiben eines Skriptes auf, wo du evtl. uninitialisierte Variablen benutzt. Weiter oben wird beschrieben, wie diese sich für einen Hack ausnutzen lassen.

Meldung erstatten!

An dieser Stelle wirds dann schon leicht paranoid. Hast du mit Angriffen zu rechnen, lasse dir fehlgeschlagene MySQL-Queries, Parameter die durch die Eingangsprüfung am Anfang des Skriptes durchgefallen sind und fehlgeschlagene Logins direkt per E-Mail zusammen mit allen Daten, die du vom Verursache zusammenkratzen kannst (IP-Adresse, Browser etc.pp.) zusenden.

Klingt gut, hat aber zwei Design-Schwächen: Erstens kann ein Angreifer seine Identität locker verstecken. Zweitens könnte er genau diese Form von "Error Reporting" für einen Denial-of-Service Angriff nutzen indem er dein Postfach mit Mails zumüllt und deinen Server durch das massenhafte Versenden von Mails lahmlegt.

Wenn man auf das Live-Feeling verzichten kann, ist es daher besser, solche Benachrichtigungen in einem Logfile abzuspeichern.

Bitte beachte, dass ich aus Zeitgründen keine Fragen zu Web-Sicherheit in den Kommentaren beantworten kann.
Kommentare (neuste zuerst)

- **Alphamann schreibt am 14.09.12 um 19:39 Uhr:** Tolle Seite - Schön gemacht !
- **Robert schreibt am 14.09.12 um 16:07 Uhr:** Danke - Super Artikel
- **Andreas schreibt am 13.09.12 um 12:13 Uhr:** Eine vernünftige SSL Verschlüsselung halte ich für gesonderte Bereiche wie Login definitiv empfehlenswert.
- **Bruce schreibt am 26.03.12 um 14:33 Uhr:** Danke - super Tipps, besonders die Abwehrstrategien!
- **Nur ich schreibt am 20.03.12 um 10:58 Uhr:** Was vieleicht auch noch ein wichtiger Punkt wäre: Wenn eine Seite schon via GET gesteuert wird (zB.mypage.de?seite=home) sollte mann, wenn auch absolut Googleunfreundlich den kompletten String "?seite=home&datum=SESSIONDATUM aufbereiten, als base64 in den Links übergeben und aus der GET Variable wieder im Script dekodieren. SESSIONDATUM wäre dann der ZEITSTEMPEL der dem USER beim ersten Aufruf der Seite zugewiesen wird. Bei der dekodierung prüft das Script dann, ob der ZEITSTEMPEL übereinstimmt und arbeitet erst weiter, wenn true zurückgegeben wird. Vorteil: Keine GET Manipulation mehr möglich. Nachteil: Man kann keine bestimmten Seiten mehr bei Facebook posten.
- **ggg schreibt am 02.02.12 um 12:18 Uhr:** <script>alert(1)</script>
- **Druckerpatronen schreibt am 19.12.11 um 11:56 Uhr:** Wenn man vielleicht auch nicht selbst in der Lage ist, diese Richtlinien umzusetzen, so gibt einem der Artikel doch genügend Informationen an die Hand, um im Gespräch mit den Entwicklern seine Wünsche korrekt zu formulieren. Immerhin "hat man schonmal davon gehört". Danke
- **romi schreibt am 14.12.11 um 15:22 Uhr:** Interesanter Artikel. Und nicht zuletzt lehrreich.
- **Rainer schreibt am 30.09.11 um 05:16 Uhr:** Wie @Someonelse bereits sagte... auch für alte Hasen sehr informativ - sehr guter Artikel - wie auch der über reguläre Expressionen! ;-)

- **Prob schreibt am 21.09.11 um 17:03 Uhr:** "Links" schau mal wie das aussieht.
- **lernen schreibt am 20.09.11 um 23:32 Uhr:** wollte nur diesen formular probieren
- **Motte schreibt am 07.08.11 um 19:23 Uhr:** Danke, sehr guter Artikel. Speziell was die Absicherung von Datenbankabfragen angeht, habe ich grade einiges dazugelernt.
- **Someonelse schreibt am 25.07.11 um 13:56 Uhr:** Auch für einen alten Hasen sehr informativ! Und damit für einen Anfänger ein MUSS. Das hier ist bestimmt die beste deutsche Zusammenfassung im Netz!
- **Christian schreibt am 18.07.11 um 01:26 Uhr:** Wirklich eine gute und hilfreiche Zusammenfassung. Danke und auf jeden Fall empfehlenswert!
- **Jürgen schreibt am 17.07.11 um 21:11 Uhr:** Danke für diese Zusammenfassung.
- **Holy schreibt am 19.06.11 um 01:31 Uhr:** Vielen Dank für die Zeit und Mühe die du aufgewendet hast, um diesen SUPER Artikel zu schreiben. Ich bin noch Neuling im Webdesign und wusste bisher nicht wie wichtig die Sicherheitsaspekte sind, bzw. wie einfach es ist Sicherheitslücken zu benutzen, da muss man ja echt kein Profi sein..
- **nelson Montz schreibt am 25.05.11 um 14:27 Uhr:** HAW HAW HAW HAW
- **xxx schreibt am 25.05.11 um 14:25 Uhr:** Mann das ist ja ne Seite Da bleibt einem ja die spucke weg
- **rsphi schreibt am 25.05.11 um 14:24 Uhr:** Toller Artikel
- **El Golfo schreibt am 23.04.11 um 14:26 Uhr:** Hui, man kann so viele Fehler machen. Danke für Deine Übersicht.
- **treaki schreibt am 09.04.11 um 21:38 Uhr:** zum Thema Meldung erstatten: wie mach ich dass, gibt es da ein Tutorial? gruß treaki
- **websecurity > all schreibt am 24.03.11 um 17:23 Uhr:** @ Frager (27.01.11 um 18:48 Uhr): Das ^ negiert nur in Charakterklassen, an dieser Stelle bezeichnet es den Anfang des überprüften Strings. Es ist also das Gegenteil des $'s, welches das Ende markiert ;)
- **oli schreibt am 24.02.11 um 01:36 Uhr:** Tolle Seite, hab garnicht gewusst, dass man mit MySQL so einfach fehler einbauen kann, hab meine seiten aber nochmal ueberprueft. Aber ich lese eh immer das passwort aus der datenbank aus und ueberpruefe dann mit php ob die beiden ueberein stimmen. habs also sowieso besser gemacht... hoffe da gibts nicht auch ne luecke. danke nochmal.
- **Frager schreibt am 27.01.11 um 18:48 Uhr:** was bewirkt das ^ in: "RewriteRule ^df_$ index.php" es sollte doch normalerweise bedeuten dass das darauffolgende zeichenkette negiert wird oder??
- **noob_webdesigner schreibt am 27.01.11 um 06:26 Uhr:** Ich bin gerade dabei mir eine Website zu machen und stolberte über diese Seite, als ich auf der suche nach einer sicheren Möglichkeit um Kontaktformulareingaben (zusätzlich zum js mit PHP) zu prüfen und zu senden. Ich finde es super, dass hier einige Sachen zusammengetragen sind, aber für mich ist das fast komplett unverständlich - ich muss leider fast alles googlen bis wirklich weiss, was damit gemeint ist, auswirkungen usw.. Aber nach der hälfte hab ich es aufgegeben - daher würde z.B. ich mich doch mehr über ein wenig Ausführung freuen. Ansonsten aber dennoch Danke schön für die Infos und Mühe.
- **Daniel Fett schreibt am 17.01.11 um 21:23 Uhr:** 10 Minuten? Da bist du aber langsam. Es geht nicht darum, den engagierten Einzeltäter abzuhalten. Aber kaum ein Spambot schafft es derzeit, den zu knacken, einfach weil er individuell ist. Und wenn doch, gibts halt schnell was neues. Know your enemy!

- **sagichned schreibt am 17.01.11 um 19:44 Uhr:** gute seite, gute erklärung hab ich auch gleich meine seiten nochmal überprüft aber über den spamschutz würde ich nochmal nachdenken, brauch 10 minuten den zu knacken(mit vb .net)
- **Tina schreibt am 21.12.10 um 13:38 Uhr:** Danke für die gute und vor allem gut verständliche Zusammenfassung!
- **A. Hoban schreibt am 11.12.10 um 00:26 Uhr:** Coole zusammenfassung. Allerdings müsste es expose_php = off heissen ;-)
- **onip schreibt am 26.11.10 um 16:25 Uhr:** danke für deine mühen. sehr gut und verständlich geschrieben. hab die seite über google mit "mysql sicherheit" auf platz gefunden .
- **Neil schreibt am 15.11.10 um 19:11 Uhr:** genialer text, hat mir sehr geholfen ;) die schreibfehler hab ich übesehen :)
- **Christoph schreibt am 22.09.10 um 14:28 Uhr:** Gute Zusammenstellung. Vor allem gut, weil es nicht zu sehr in die Tiefe geht, dadurch aber einen guten Überblick verschafft. Sicher gibt es noch einige Sicherheitsmaßnahmen mehr, aber drei vier Punkte konnte ich noch finden. Besonders die Funktion mysql_escape_string scheint mir nützlich, habe bisher aber noch nichts von gehört / gelesen. Schöne Grüße
- **encud schreibt am 13.08.10 um 14:58 Uhr:** Hey! Super Artikel, wird mir sicher helfen bei meinen Projekten. Hab aber nen kleinen fehler entdeckt unter zombie includes muss es nicht $safeInclude = str_replace(array('/', '\/', '.'), $_REQUEST['include']); sondern $safeInclude = str_replace(array('/', '\/', '.'), '', $_REQUEST['include']); heißen... hast den 2. Parameter vergessen, aber das ist ja eigentlich trivial, die hauptsache ist man versteht was du meinst :)
- **philip schreibt am 01.07.10 um 01:12 Uhr:** Guter Artikel, danke dir!
- **Cujo schreibt am 21.05.10 um 20:51 Uhr:** Vielen Dank für die Tipps zum Thema PHP-Sicherheit :)
- **Peter schreibt am 18.05.10 um 09:33 Uhr:** Vielen Dank für die tollen Tipps und Infos!
- **Matthias Griesbach schreibt am 17.05.10 um 16:16 Uhr:** Auch wenn mir das meiste schon bekannt ist, gabe es doch wieder neue Infos dazu. OR 1 OR war mit neu.
- **Dirk schreibt am 16.03.10 um 11:06 Uhr:** Sehr gute Tipps, die man auch befolgen sollte! Hatte selber schon ein Problem mit meiner WEB-Seite. Ich habe über ein Menü verschiedene Texte mit include in meine Seite eingebunden. Alles ohne die Eingabe zu überprüfen! Eines Tages wurde meine Seite von verschiedenen Stellen (z.B. Google) gesperrt und mein Provider teilte mir mit, dass Malware gefunden wurde! Bei der Überprüfung stellte ich schnell fest, dass jemand diese Lücke ausgenutzt hatte um seine eigenen Daten bei mir auf zuspielen. War eine Menge Arbeit um alles wieder zu bereinigen. Ich habe leider erst danach, mich mit der Sicherheit beschäftigt. Das war schon Blauäugig! Ich übersende die Parameter immer noch in Klartext, aber lasse nur noch die Parameter durch, die es auch gibt und überprüfe die Länge des Strings (um lange Zeichenketten auszuschließen). Bei einem Versuch andere Zeichen ein zu geben, kommt keine Fehlermeldung sondern man landet auf der Startseite. Ich hoffe damit, der Sicherheit auf meiner Seite genüge getan zu haben. Ist für mich auch nur ein wenig Spaß am programmieren – Teile der Welt ja nicht wirklich wichtige Sachen mit. Brauche meine Domäne mehr für meine Mail-Adressen. Dirk
- **Analphbet schreibt am 18.01.10 um 14:32 Uhr:** wollte nur mal testen ob du deine Ratschläge auch selber befolgst... :)
- **" OR 1 OR " schreibt am 18.01.10 um 14:30 Uhr:** " OR 1 OR "
- **Eknoes schreibt am 02.01.10 um 01:55 Uhr:** Danke :) Ist echt super, hat mir sehr geholfen! lg

- **Peper schreibt am 13.12.09 um 16:59 Uhr:** Danke sehr guter Artikel gefällt mir sehr sehr gut. :)
- **Raphi schreibt am 11.12.09 um 13:01 Uhr:** Danke, toller Artikel!

Kommentar schreiben

- **Bugs in AGTL** bitte hier melden.
  Please report any **bugs in AGTL** here.
- Wer eine **Antwort** erwartet, sollte eine echte E-Mail-Adresse angeben.
  If you expect a **reply from me**, please provide a working e-mail address.
- Ich kann **keine individuelle Problemlösung** zu Websicherheit oder regulären Ausdrücken geben!
-

Name:

Website: http:// optional

E-Mail: wird nicht veröffentlich

Spam-Check: 20 plus 4 ist gleich

absenden

22 · Startseite · Impressum · Kontakt