



Ulrich Büttner

Martin Bergann



Sicherheitsrisiko PHP

Rund 80% aller Angriffe von außen auf Webserver finden über unsichere PHP-Konfiguration und unsichere PHP-Scripte statt.



Inhalt

- Was ist PHP?
- Ziele der Angreifer
- Angriffsvektoren
- Gegenmaßnahmen
- Beispiele --> Live-Hacking
- Fazit
- Links
- Lokale Diskussions-Plattformen



Was ist PHP?

- serverseitige Scriptsprache
- zur Erstellung dynamischer Websites, Grafiken,...
- erste Version 1995 von Rasmus Lerdorf (*Grönland*) entwickelt
(Lerdorf arbeitet heute bei yahoo)
- anfangs Sammlung von Perl-Scripten, später (und auch noch heute) in C geschrieben
- ursprünglich: „**personal homepage**“
- später: rekursives Acronym - **PHP: hypertext preprocessor**
- November 1997: PHP/FI 2.0
- Juni '98: PHP 3, neu geschrieben von Zeev Suraski und Andi Gutmans (*Israel*) , in Kooperation mit Rasmus Lerdorf (Entwicklung von PHP/FI wurde eingestellt)

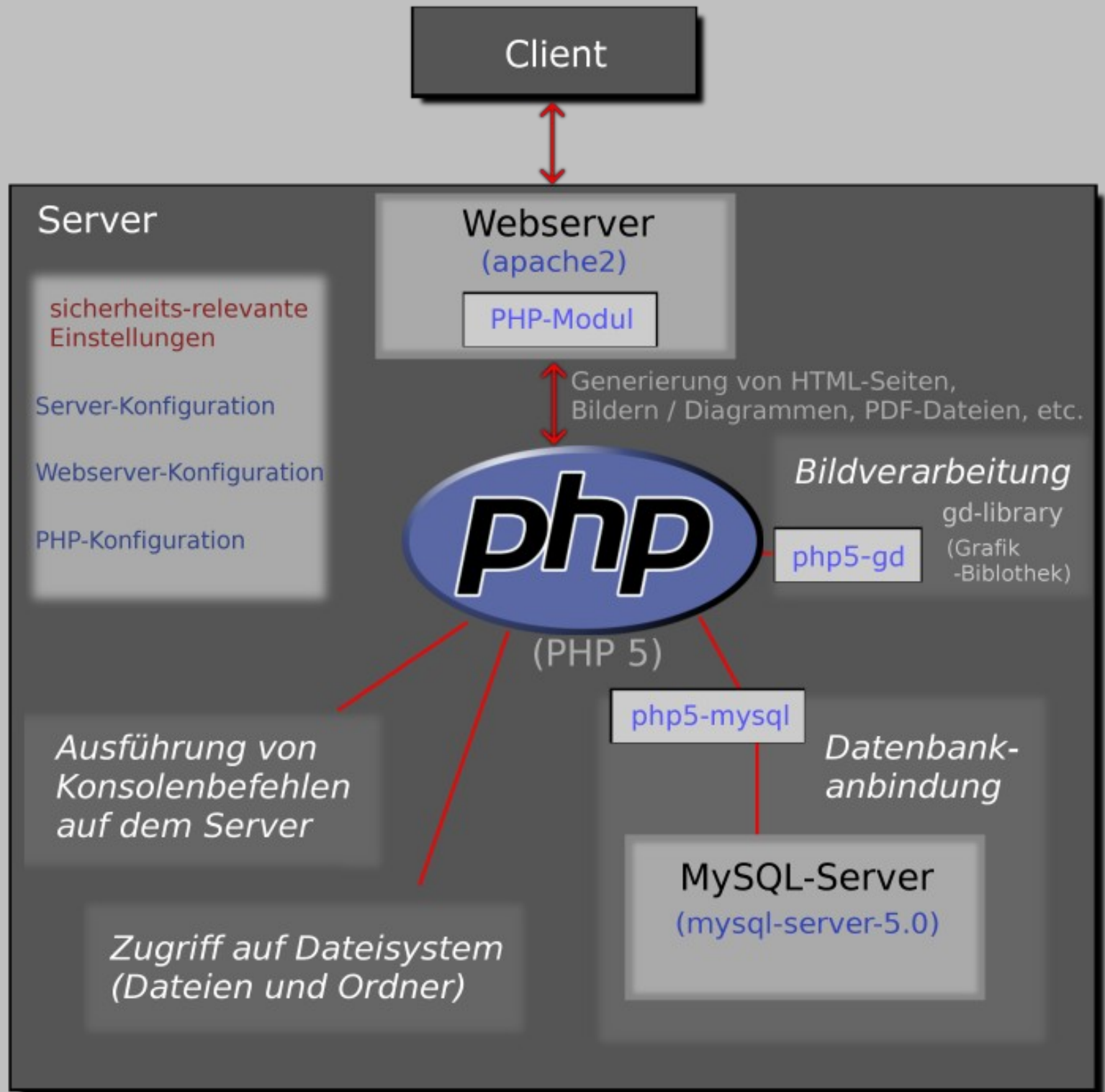


Was ist PHP?

- Syntax leicht zu lernen
 - an C angelehnt
 - keine expliziten Datentypen (automatisches Type-Casting)
 - Variablen müssen nicht deklariert werden
 - Variablen außerhalb ihres Gültigkeitsbereiches (wie man ihn von C her kennt) verfügbar, durch includes auch über verschiedene Dateien hinaus
- durch modularen Aufbau für viele Einsatzgebiete geeignet
- Fazit: sehr umfangreich und mächtig, gleichzeitig ein großes Sicherheits-Risiko

Zitat von metax. auf hackerboard.de:

„PHP zu Programmieren ist, wie auf einem Drahtseil zu balancieren, während man eine Stange Dynamit mit brennender Lunte in der Hand hält.“





Ziele der Angreifer

- Seite defacen / ownen ("verschandeln" / "übernehmen")
- Informationsbeschaffung
 - Spionage (Zugangsdaten oder sonstige sensible Daten)
 - Phishing
- anonyme Informations- und Daten-Verbreitung
 - illegale Downloads auf zu eigen gemachten Servern hosten
 - Spam-Mails verschicken
- Erschaffung und Erweiterung von Bot-Netzen (z.B. IRC-Bots)
 - DoS-/DDoS-Attacken
 - Spam-Verteiler



Angriffsvektoren

schlecht geschriebene Scripte	↔	Unsichere Server-Konfiguration	Schlechte Server-Wartung
LFI (Local File Inclusion)			Exploits
RFI (Remote File Inclusion)			
SQL-Injection			
XSS (CrossSiteScripting)			



Gegenmaßnahmen

im Script

- „White-Listing“ bei Includen oder Einlesen/Verarbeiten von Dateien verwenden (nur Seiten includen oder verarbeiten, die wirklich zur Seite gehören)
- Variablen mit festem Wert initialisieren
- **Never trust an user-input**
 - alle Daten, die irgendwie vom User manipuliert werden könnten, auf gültigen Inhalt überprüfen (GET, POST, COOKIE, Daten aus zu verarbeitenden Dateien)
 - Variablen überprüfen, ob sie von dem Datentyp sind, von dem sie sein sollten (z.B. Postleitzahl = 5 Ziffern von 0-9)



Gegenmaßnahmen

in PHP.ini

- **register_globals = Off**

bewirkt, dass man auf Variablen, die mittels POST-Methode oder GET-Methode gesendet wurden, nur noch via `$_POST[„varname“]` bzw. `$_GET[„varname“]` bzw. mit `$_REQUEST[„varname“]` zugreifen kann - nicht mehr direkt mit `$varname`. Somit wird ein mögliches Manipulieren von Seiten-Inhalten erschwert.

(ab PHP6 wird es diese Option nicht mehr geben, dann wird `register_globals` generell abgeschaltet sein.)

- **open_basedir = „/var/www:/tmp:/usr/share/phpmyadmin“**

legt Verzeichnisse fest, auf welche PHP-Scripte zugreifen dürfen (incl. Unterverzeichnisse) - alle anderen Verzeichnisse sind für PHP-Scripte gesperrt

- **allow_url_include = Off**

bewirkt, dass URLs (`http://www.example.com/file.txt`) nicht mehr includet werden können. Somit wird das Einschleusen von Schad-Code erschwert.

- **disable_functions = shell_exec, exec, chown, chmod**

hiermit lassen sich bestimmte Funktionen (z.B. Funktionen zum Ausführen von Konsolen-Programmen, Funktionen zur Bearbeitung von Dateien und Ordnern, etc.) deaktivieren.



Gegenmaßnahmen

in PHP.ini

- **disable_classes**

hiermit lässt sich die Benutzung bestimmter Klassen deaktivieren.

- **magic_quotes_gpc = On**

bewirkt, dass Anführungszeichen escapet werden (aus " wird \")

Somit werden XSS-Angriffe und SQL-Injections erschwert.

(wird in PHP6 ebenfalls abgeschafft)

- **max_execution_time = 30**

legt die maximale Ausführungs-Zeit von PHP-Scripten fest

--> verhindert DoS durch Endlosschleifen

- **safe_mode-Optionen**

schränken einige Rechte ziemlich hart ein - gerade bei shared hosting nicht

empfehlenswert *(wird ab PHP6 ganz abgeschafft)*

Weitere Optionen: alle Optionen in der PHP.ini sind per default mit Kommentaren versehen, welche die Funktion der entsprechenden Option beschreiben



Gegenmaßnahmen

Server-Konfiguration und -Wartung allgemein

- aktuelle stabile Software verwenden (und regelmäßig updaten)
- Webserver und PHP niemals unter Benutzer „root“ laufen lassen, sondern einen eigenen Benutzer mit weitestgehend eingeschränkten Rechten dafür verwenden (z.B. „www-data“)
- Versionsnummern von Apache und PHP ausblenden
Warnung: „security through obscurity“ kann zwar helfen, den „Script-Kiddies“ das Hacken des Servers zu erschweren, sollte jedoch niemals als alleinige Maßnahme gesehen werden.
- PHP nicht via mod_php einbinden, sondern z.B. via fastcgi
Vorteile fastcgi:
 - jede Domain bekommt eigene PHP.ini (oder auch einzelne Kunden gar kein PHP)
 - PHP läuft für jeden Kunden unter dessen User-Account
System-Account - somit wird sichergestellt, dass Kunde A keine Rechte hat, bei Kunde B die Dateien auszuspionieren...



Gegenmaßnahmen

Server-Konfiguration und -Wartung allgemein

- zusätzliche Sicherheits-Software und -Patches verwenden
 - IDS (Intrusion-Detection-System) (z.B. snort)
 - Firewall (z.B. iptables) + IP-Sperre (fail2ban)
 - Webserver zusätzlich absichern, z.B. Apache2 mit libapache2-mod-security
 - zusätzliche PHP-Patches (php-hardend / suhosin)



Demonstration

- Schutz vor Local File Inclusion und Remote File Inclusion

durch PHP-Konfiguration

durch sicherere Programmierung

- wenn noch Zeit:
 - simples XSS-Beispiel
 - Beispiele aus dem Netz...



Fazit

- möglichst auf sichere Server setzen
- Scripte gegen jegliche Art von Angriffen absichern
 - never trust an user-input
 - Variablen – wenn bei PHP auch nicht explizit notwendig – vor Gebrauch initialisieren
- bei größeren Projekten, insbes. wenn schon Sicherheitslücken vermutet werden, an entsprechende Firmen wenden
 - z.B.:
 - SektionEins (spezialisiert auf PHP-Sicherheit)
 - SySS GmbH (allgemein Sicherheit im Bereich Netzwerk, Datenübertragung, etc.)



Fazit

- gutes PHP-Script sollte
 - auf einem stark geschützten Server immernoch laufen
 - auf einem unsicheren Server trotzdem keine signifikanten Sicherheits-Mängel aufweisen



Links

- PHP Handbuch und Download
 - <http://www.php.net/>
- Foren für Hilfe bei PHP-Problemen
 - <http://phpforum.de/>
 - <http://www.php.de/>
 - <http://www.php-resource.de/>
 - <http://entwickler-forum.de/>
 - <http://www.hackerboard.de/>
- Firmen für Security-Audits
 - SektionEins (spezialisiert auf PHP-Sicherheit) <http://www.sektioneins.de/>
 - SySS GmbH (allgemein Sicherheit im Bereich Netzwerk, Datenübertragung, etc.)
<http://www.syss.de/>



Links

- aktuelle Informationen zum Thema Security
 - Allgemein
 - <http://www.heise.de/> / <http://www.heise.de/security/>
 - speziell PHP
 - <http://www.php-security.org> / <http://blog.php-security.org/>
 - Info's über Bugs / Bugs melden: <http://bugs.php.net/>
- PHPSecInfo (Script, welches PHP-Einstellungen auf Server überprüft)
 - <http://phpsec.org/projects/phpsecinfo/>

dieses Script und die Quellcodes der LiveHacking-Demonstration
<http://studium.cs-bergann.de>



Diskussions-Plattformen

- Lokales Angebot an Diskussions-Plattformen (Jena und Umgebung)
 - WebMontag in Jena : <http://www.webmontag.de/doku.php?id=jena>
 - PHP UserGroup Thüringen : <http://blog.phpug-thueringen.de/>
 - für Typo3-User: Typo3-UserGroup Erfurt



Fragen?

