Web Security

Web Programming

Uta Priss
ZELL, Ostfalia University

2013

Outline

PHP-security information on the web

Quotes from an on-line forum:

PHP-security information on the web

Quotes from an on-line forum:

"Perl/CGI scripts are very insecure."
"A PHP programmer does not have to worry about security".

PHP-security information on the web

Quotes from an on-line forum:

"Perl/CGI scripts are very insecure."
"A PHP programmer does not have to worry about security".

"At first my remote connection to Mysql did not work, but then I discovered I only had to stop my firewall and it worked fine."

PHP-security information on the web

Quotes from an on-line forum:

"Perl/CGI scripts are very insecure."
"A PHP programmer does not have to worry about security".

"At first my remote connection to Mysql did not work, but then I discovered I only had to stop my firewall and it worked fine."

"You can use HTTP_REFERER to make sure that your site can only be accessed from your web form."

All you need to connect to a database with PHP is something like this:

```
<?php
$db = pg_pconnect("host=localhost,dbname=a,user=b");
pg_exec($db,"select * from $table";);
?>
```

To send an email with PHP back to a user, you'll need something like this:

```
<?php
$body = "Hi, How are you?";
mail($user, "Subject", $body)
?>
```

Apache error log:

66.147.118.70-[7/7/06] "GET /phpadmin/main.php HTTP/1.1" 404
66.147.118.70-[7/7/06] "GET /phpmyadmin1/main.php HTTP/1.1" 404
66.147.118.70-[7/7/06] "GET /phpAdmin-2/main.php HTTP/1.1" 404

Debian Security Advisory - phpmyadmin (DSA 1207-2)

```
-----BEGIN PGP SIGNED MESSAGE-----
Hash: SHA1

- ------------------------------------------------
Debian Security Advisory DSA 1207-2
http://www.debian.org/security/
November 19th, 2006                        http://www
- ------------------------------------------------

Package         : phpmyadmin
Vulnerability   : several
Problem-Type    : remote
Debian-specific: no
CVE ID          : CVE-2006-1678 CVE-2006-2418 CVE-2
CVE-2006-5116
Debian Bug      : 339437 340438 362567 368082 39109

The phpmyadmin update in DSA 1207 introduced a reg
corrects this flaw. For completeness, the original
```

Scripting languages are different

If you search the Web for information on PHP:
you may find useless information.

Scripting languages are different

If you search the Web for information on PHP:
you may find useless information.

If you search the Web for information on Haskell:
you'll find accurate information.

Why?

Scripting languages are different

If you search the Web for information on PHP:
you may find useless information.

If you search the Web for information on Haskell:
you'll find accurate information.

Why?

→ For PHP: only trust the official manual!!!

Server-side applications

- Client-server separation.
- End-users and content providers have different security requirements.
- Browser security versus server security.
- Web space is often hosted externally and shared with other users.
- Most HTTP traffic is plain text (non encrypted).
- Limitations of the HTTP-protocol.

...

Server-side applications (continued)

- Even with sessions: security checks are required for each data transfer, i.e. more than once per session.
- Security problems apply equally to all scripting languages (Perl, PHP, ASP, etc). But some languages are more open about the problems while others keep more details hidden from the programmers.

Security Strategies

- Prevention
  security guidelines, advisories, common sense
- Detection
  monitor webserver logs, system activity, detection software
- Response
  script-level, webserver, institutional policies

Software testing

Traditional approaches for software testing
(functional testing, user testing, ...)
are useless for security validation.

Security validation:

- no “debugging”, no immediate feedback
- no clear testing protocols
- different types of problems are possible:
  requires lateral thinking

Security Engineering

see “patterns & practices Security Engineering Index” (msdn.microsoft.com)

- Security objectives
- Threat modelling
- Security design guidelines
- Security architecture and design reviews
- Security code reviews
- Security testing
- Security deployment reviews

General security risks

- Physical security
- Social engineering and human error (e.g. insecure passwords)
- Eavesdropping, “man-in-the-middle” attacks
- Software flaws (buffer overflows)
- Software vulnerabilities (check security advisories)
- Outdated software
- Installation of malicious software:
  Trojan horses, back doors, viruses, worms
- Denial of service (DoS) attacks

Webserver security

Web space is often hosted externally and shared with other users. Files with permission 755 or 644 can be read by other users on the same host.

- Disallow server-side includes.
- Disallow indexes.
- Only store files in the public_html directory if they really need to be there.
- Security through obscurity.
- Set usage limits for script CPU time and storage space.

Webserver security (continued)

Apache's mod_security

- Place Apache in a chroot directory.
- POST filtering based on headers, values, IP addresses.
- POST payload analysis.
- Restrict the use of certain HTML tags (e.g. <script>).
- Prevent SQL injection ("delete", "insert").
- Prevent SHELL commands.
- etc

Of course, the server will run slower and use more memory.

Other server functions

- Email: protect against spam and phishing
- Install email server on different machine from webserver.
- Don't allow the www user to send email.
- HTACCESS
  Useful for group-based restriction to part of site,
  not very useful for login/registration of users.
- Database
  DB security and script security need to be integrated.

Client-side threats

- **Content spoofing**: tricks a user into believing content is from a different website (e.g. using redirection).
- **Cross-site scripting (XSS)**: forces a website to display malicious code in a user's browser.
- **Phishing**: masquerading as a trustworthy website in order to obtain user's passwords, bank details etc.
- **Spam**: unwanted email. Hacker abuses script that sends emails to unchecked addresses entered via a web form.

These are server threats, if your server or script is the one that is being abused by a hacker in this manner.

General server-side threats

- **Trojan horses**: code downloaded from somewhere and integrated into a script without being carefully checked.
- **Predictable resource location**: hackers scan for predictable names of scripts and files that could be abused.
- **Directory indexing** and **path traversal**: list directory contents, if "index.html" does not exist.

These can all lead to **information leakage**.

Hacker pretends to be another user

- **Authentication**: insufficient authentication or obtaining passwords by brute force or by recovering another user's password.
- **Authorisation**: insufficient authorisation or hijacking another user's session.
- **Cross-site request forgery**: forces one user's browser to make a HTTP request pretending to be another user.
- **Session hijacking**: hacker predicts session ID or captures it from another user's cookie or HTTP request by cross-site request forgery.

Sessions and cookies cannot be trusted to come from the correct user.

Unchecked user input

- **HTML injection**: user enters Javascript etc into textfields of a web form.
- **Defacing**: user enters <img> etc into a guestbook or forum, which then displays the image to other users. (Phishing and XSS can be achieved in this manner.)
- **Spoofed form submission or spoofed HTTP request**:
  - User guesses hidden variables or filenames used in a script and uses them to disclose information or opening files.
  - User edits the query string or POST data.
  - User edits environment variables: HTTP_REFERER, HTTP_USER_AGENT, etc can never be trusted.
- **Unencrypted cookies** can be read and modified by user.

Solution: remove all HTML tags and special characters from form data. Check all form variables. Make no assumptions about form variables, environment variables or cookies. Encrypt cookies.

Unchecked user input sent to other processes

- **SQL injection**: user data from a web form is directly entered into an SQL statement without careful checking. (E.g. user enters "0; additional query; - -".)
  Similar: **LDAP injection**, **SSI injection**, etc.
- **Shell access**: user could try to start a new process or to open shell from database.
- **Information leakage:** disclose hidden information, for example, user enters "somebody@somewhere</etc/passwd;" as an email address.

Solution: remove special characters (e.g. semicolon) from user data. Check all user input. Apply some heuristics for checking email addresses or check them manually.

How paranoid does one need to be?

- If your site doesn't store sensitive data, then you probably mainly need to worry about information leakage, shell access and general server threats. You don't need to worry about sessions and cookies.
- Defacing, phishing, and XSS mostly apply to websites that permanently display external data (guestbooks, forums, wikis, hosted blogs).
- The less a script interacts with other resources on the server (email, database, opening files), the more secure it is.
- If you are storing sensitive customer data (addresses, credit card numbers, etc), then you must encrypt the whole session using SSL and you must check for all possible threats.

PHP security

- Read the security part in the PHP manual.
- Use the latest version of PHP (which is more secure).
- Use appropriate functions:
  htmlspecialchars(); strip_tags(); add_slashes();
  mysql_real_escape_string(); etc.
- Encrypt cookies: setcookie(base64_encode($cookie)) .
- Apply "hardening" patch or Suhosin to PHP before installing.
- PHP safe_mode: no longer available in PHP 6.0 because it is misleading.
- open_basedir: only files in specified directory-tree can be opened.

Perl security

- Read about Perl and CGI security in the W3 Security FAQ.
- Use tainted mode (Perl -T).
- Be extremely careful with system calls, backticks, open(), etc.
- Check user input. A few lines of regular expression code should do the trick.