

Mastering PHP Security

Chris Shiflett

Brain Bulb
The PHP Consultancy

chris@brainbulb.com

Who Am I? (Why Listen to Me?)

- ▶ Author of PHP Security (O'Reilly) and HTTP Developer's Handbook (Sams)
- ▶ Author of Security Corner (php|architect) and Guru Speak (PHP Magazine)
- ▶ Founder of PHP Security Consortium
- ▶ Member of Zend Advisory Board and an author of the Zend PHP Certification
- ▶ Founder and President of Brain Bulb, The PHP Consultancy

Talk Outline

- ▶ Introduction
- ▶ Two Best Programming Practices
- ▶ Two Most Common Vulnerabilities
- ▶ Lightning Attacks
- ▶ PHP Security Audit HOWTO (abridged)
- ▶ More Information
- ▶ Questions and Answers

Two Best Practices (The Least You Can Do)

- ▶ Filter Input
- ▶ Escape Output

Filter Input: What Is Input?

- ▶ Most input is obvious - form data (`$_GET` and `$_POST`), cookies (`$_COOKIES`), RSS feeds, etc.
- ▶ Some data is harder to identify - `$_SERVER`, data from databases, etc.
- ▶ Some data is frequently misunderstood - `$_SESSION`, etc.
- ▶ The key is to identify the origin of data. If it originates from any external source, it is input and must be filtered.

Filter Input: What Is Filtering?

- ▶ Filtering is the process by which you inspect data to prove its validity.
- ▶ When possible, use a whitelist approach - assume data to be invalid unless you can prove otherwise.
- ▶ Filtering is useless if you can't keep up with what has been filtered and what hasn't.
- ▶ Employ a strict naming convention that lets you easily and reliably distinguish between filtered and tainted data.

Filter Input: Show Me the Code!

```
<?php
$clean = array();

switch($_POST['color'])
{
    case 'red':
    case 'green':
    case 'blue':
        $clean['color'] = $_POST['color'];
        break;
}

?>
```

Filter Input: Show Me the Code!

```
<?php
$clean = array();

switch($_POST['color'])
{
    case 'red':
    case 'green':
    case 'blue':
        $clean['color'] = $_POST['color'];
        break;
}

?>
```

Filter Input: Show Me the Code!

```
<?php  
  
$clean = array();  
  
switch($_POST['color'])  
{  
    case 'red':  
    case 'green':  
    case 'blue':  
        $clean['color'] = $_POST['color'];  
        break;  
}  
  
?>
```

Filter Input: Show Me the Code!

```
<?php
$clean = array();

switch($_POST['color'])
{
    case 'red':
    case 'green':
    case 'blue':
        $clean['color'] = $_POST['color'];
        break;
}

?>
```

Filter Input: Show Me the Code!

```
<?php
$clean = array();

switch($_POST['color'])
{
    case 'red':
    case 'green':
    case 'blue':
        $clean['color'] = $_POST['color'];
        break;
}

?>
```

Filter Input: Show Me the Code!

```
<?php
$clean = array();

switch($_POST['color'])
{
    case 'red':
    case 'green':
    case 'blue':
        $clean['color'] = $_POST['color'];
        break;
}

?>
```

Filter Input: Show Me More Code!

```
<?php
$clean = array();

if (ctype_alnum($_POST['username']))
{
    $clean['username'] = $_POST['username'];
}

?>
```

Filter Input: Show Me More Code!

```
<?php
$clean = array();

if (ctype_alnum($_POST['username']))
{
    $clean['username'] = $_POST['username'];
}

?>
```

Filter Input: Show Me More Code!

```
<?php
$clean = array();

if (ctype_alnum($_POST['username']))
{
    $clean['username'] = $_POST['username'];
}

?>
```

Filter Input: Show Me More Code!

```
<?php
$clean = array();

if (ctype_alnum($_POST['username']))
{
    $clean['username'] = $_POST['username'];
}

?>
```

Filter Input: Show Me More Code!

```
<?php
$clean = array();

if (ctype_alnum($_POST['username']))
{
    $clean['username'] = $_POST['username'];
}

?>
```

Escape Output: What Is Output?

- ▶ Most output is obvious (anything sent to the client is output) - HTML, JavaScript, etc.
- ▶ The client isn't the only external destination - databases, session data stores, RSS feeds, etc.
- ▶ The key is to identify the destination of data. If it is being sent to any external source, it is output and must be escaped.

Escape Output: What Is Escaping?

- ▶ Escaping is the process by which you escape any character that has a special meaning in the external system for which it is destined.
- ▶ Unless you're sending data somewhere unusual, there is probably a function that performs the escaping for you.
- ▶ The two most common destinations are the client (use `htmlspecialchars()`) and MySQL (use `mysql_real_escape_string()`).
- ▶ If you must write your own, make sure you're exhaustive - find a reliable and complete list of all characters with special meaning.

Escape Output: Show Me the Code!

```
<?php
$html = array();

$html['username'] = htmlentities($clean['username'],
    ENT_QUOTES, 'UTF-8');

echo "<p>Welcome back, {$html['username']}</p>";

?>
```

Escape Output: Show Me the Code!

```
<?php
$html = array();

$html['username'] = htmlentities($clean['username'],
    ENT_QUOTES, 'UTF-8');

echo "<p>Welcome back, {$html['username']}</p>";

?>
```

Escape Output: Show Me the Code!

```
<?php
$html = array();

$html['username'] = htmlentities($clean['username'],
    ENT_QUOTES, 'UTF-8');

echo "<p>Welcome back, {$html['username']}</p>";

?>
```

Escape Output: Show Me the Code!

```
<?php
$html = array();

$html['username'] = htmlentities($clean['username'],
    ENT_QUOTES, 'UTF-8');

echo "<p>Welcome back, {$html['username']}</p>";

?>
```

Escape Output: Show Me the Code!

```
<?php
$html = array();

$html['username'] = htmlentities($clean['username'],
    ENT_QUOTES, 'UTF-8');

echo "<p>Welcome back, {$html['username']}</p>";

?>
```

Escape Output: Show Me More Code!

```
<?php
$mysql = array();

$mysql['username'] =
    mysql_real_escape_string($clean['username']);

$sql = "SELECT *
        FROM   profile
        WHERE  username = '{$mysql['username']}'";

$result = mysql_query($sql);

?>
```

Escape Output: Show Me More Code!

```
<?php
mysql = array();

mysql['username'] =
    mysql_real_escape_string($clean['username']);

$sql = "SELECT *
        FROM   profile
        WHERE  username = '{$mysql['username']}'";

$result = mysql_query($sql);

?>
```

Escape Output: Show Me More Code!

```
<?php  
  
$mysql = array();  
  
$mysql['username'] =  
    mysql_real_escape_string($clean['username']);  
  
$sql = "SELECT *  
        FROM   profile  
        WHERE  username = '{$mysql['username']}'";  
  
$result = mysql_query($sql);  
  
?>
```

Escape Output: Show Me More Code!

```
<?php
$mysql = array();

$mysql['username'] =
    mysql_real_escape_string($clean['username']);

$sql = "SELECT *
        FROM   profile
        WHERE  username = '{ $mysql['username'] }'";

$result = mysql_query($sql);

?>
```

Escape Output: Show Me More Code!

```
<?php  
  
$mysql = array();  
  
$mysql['username'] =  
    mysql_real_escape_string($clean['username']);  
  
$sql = "SELECT *  
        FROM   profile  
        WHERE  username = '{$mysql['username']}'";  
  
$result = mysql_query($sql);  
  
?>
```

Escape Output: Show Me More Code!

```
<?php
$mysql = array();

$mysql['username'] =
    mysql_real_escape_string($clean['username']);

$sql = "SELECT *
        FROM   profile
        WHERE  username = '{$mysql['username']}'";

$result = mysql_query($sql);

?>
```

Two Most Common Vulnerabilities

- ▶ SQL Injection
- ▶ Cross-Site Scripting

SQL Injection: What's the Problem?

```
<?php
$query = "SELECT *
        FROM   profile
        WHERE  username = '{$_POST['username']}'";

$result = mysql_query($query);

?>
```

SQL Injection: What's the Problem?

```
<?php
$query = "SELECT *
        FROM   profile
        WHERE  username = 'myname' OR 'foo' = 'foo'";

$result = mysql_query($query);

?>
```

SQL Injection: What's the Solution?

- ▶ Filter input.
- ▶ Escape output.
- ▶ Use an escaping function native to your database (`mysql_real_escape_string()` for MySQL)
- ▶ If there isn't one, `addslashes()` is a good last resort.
- ▶ Never enable magic quotes.

Cross-Site Scripting: What's the Problem?

```
<?php  
echo "<p>Welcome back, {$_GET['username']}</p>";  
?>
```

Cross-Site Scripting: What's the Problem?

```
<?php  
echo "<p>Welcome back, <script> ... </script>.</p>";  
?>
```

Cross-Site Scripting: What's the Solution?

- ▶ Filter input.
- ▶ Escape output.
- ▶ Use `htmlspecialchars()` for the escaping.
- ▶ If you want to allow some HTML to be interpreted, you can convert specific entities back to HTML (whitelist approach).
- ▶ BBCode offers no protection.

Lightning Attacks: Exposed Source Code

- ▶ The Problem:
Includes named `foo.inc` are displayed in plain text in the browser, exposing source code.
- ▶ The Solution:
Don't store includes under document root. The only resources you should store under document root are those that must be accessible via URL. Making anything else available to the public is an unnecessary risk.

Lightning Attacks: Session Fixation

- ▶ The Problem:

PHP uses any session identifier provided by the client. An attacker can take advantage of this by providing links to your application with an embedded session identifier.

- ▶ The Solution:

Use `session_regenerate_id()` whenever there is a change in the level of privilege.

Lightning Attacks: Session Hijacking

- ▶ The Problem:

An attacker can impersonate another user if that user's session identifier is known by the attacker.

- ▶ The Solution:

Protect the session identifier from exposure. Use SSL and propagate it in a cookie. For a defense in depth approach, propagate an authentication token to strengthen the identity of the client.

Lightning Attacks: Spoofed Form Submissions

- ▶ The Problem:
A user can use a form other than the one you provide using a number of different methods.
- ▶ The Solution:
Don't worry about it. As long as a user abides by your rules, it doesn't matter.
- ▶ The Caveat:
Cross-Site Request Forgeries are an exception. Make sure to protect yourself from these.

Lightning Attacks: Spoofed HTTP Requests

- ▶ The Problem:

A user can send arbitrary HTTP requests to your server using a number of different methods.

- ▶ The Solution:

Don't worry about it. As long as a user abides by your rules, it doesn't matter.

Lightning Attacks: Cross-Site Request Forgeries

- ▶ The Problem:

An attacker can send arbitrary HTTP requests from a victim. Because the requests originate from the victim, they can bypass traditional safeguards, including firewalls and access control.

- ▶ The Solution:

Use a unique token in every form that you send to the user. Whenever you receive a request from the user that represents a form submission, check for this unique token.

Lightning Attacks: Command Injection

- ▶ The Problem:

If you use tainted data in a command (using `exec()`, `system()`, etc.), an attacker can potentially execute arbitrary commands.

- ▶ The Solution:

As always, filter input and escape output. In this case, escaping output means using `escapeshellcmd()`. The `escapeshellarg()` function can also be helpful.

PHP Security Audits: Overview

- ▶ Focus on failures to filter input or escape output.
- ▶ This is accomplished primarily by tracking data.
- ▶ If tracking data is difficult for you, then it's also difficult for the developers.
- ▶ If distinguishing between tainted and filtered data is difficult for you, then it's also difficult for the developers.

PHP Security Audits: Guidelines

- ▶ Ask questions. No one knows an application as well as the developer(s).
- ▶ Have the design explained to you, and identify design problems before you review the code.
- ▶ Swallow your pride. Your purpose is not to impress anyone.
- ▶ Provide specific references to code. Generic recommendations are not a security audit.
- ▶ Educate yourself. Web application security is a highly specialized discipline. The more you know, the better the audit.

More Information

- ▶ PHP Security Consortium
<http://phpsec.org/>
- ▶ PHP Security Consortium Library
<http://phpsec.org/library/>
- ▶ PHP Security Guide
<http://phpsec.org/projects/guide/>
- ▶ My Personal Web Site
<http://shiflett.org/>
- ▶ My Business Web Site
<http://brainbulb.com/>

Questions and Answers

- ▶ Now it's your chance to really get your money's worth!
(Hopefully there is plenty of time left.)
- ▶ Feel free to ask me questions anytime.
- ▶ Visit brainbulb.com for PHP security audits, training, etc.
- ▶ Thanks for listening!