

PHP Security

Kevin Schroeder
Zend Technologies

Disclaimer

- **Do not use anything you learn here for nefarious purposes**

Why Program Securely?

- **Your job/reputation depends on it**
- **You may provide access to your own private data**
- **You may provide access to others private data**
- **You may allow someone to impersonate another (identity theft)**
- **You may take the blame for another person's attack (remote code injection)**
- **You may be prone to service attacks**

Why's the web so dangerous?

- **It's open**
 - Lots of bad code out there
 - There are lots of bad people out there
 - Many servers set up by inexperienced sys admins
 - Or someone simply forgot to filter a variable
- **Many people think they are immune/not a target**
 - Security not taken seriously
 - Insufficient time or resources to take security into consideration
 - Stored information not considered important enough to secure

What are the rules?

- **Always use multiple methods of security**
 - Validating a login is not enough
- **The principle of least privileges**
- **Initialize all variables**
- **Cast variables when appropriate**
- **Don't store sensitive data in the web tree**
- **Filter all data**
- **Don't rely on hidden form variables.**

What are the rules?

- **And last, but not least. No matter how much they cry. No matter how much they beg...**
- **Never, ever, trust your users.**

What are the rules?

- **They could be this...**



What are the rules?

- **But you have no way of being absolutely sure that they aren't this...**



What are the rules?

- **So what does that mean?**
 - Always filter data coming from the user or leaving your program (such as a database or system call)
 - Always escape output
 - Use a whitelist approach to data filtering whenever possible

Basic types of attacks

- **SQL Injection**
- **Cross Site Scripting (XSS)**
- **Cross Site Request Forgery (XSRF)**
- **Command Injection**
- **Remote Code Injection**
- **Session Hijacking**
- **Session Fixation**
- **Cookie Forging**

SQL Injection

- **Used to insert data into SQL statements**

SQL Injection

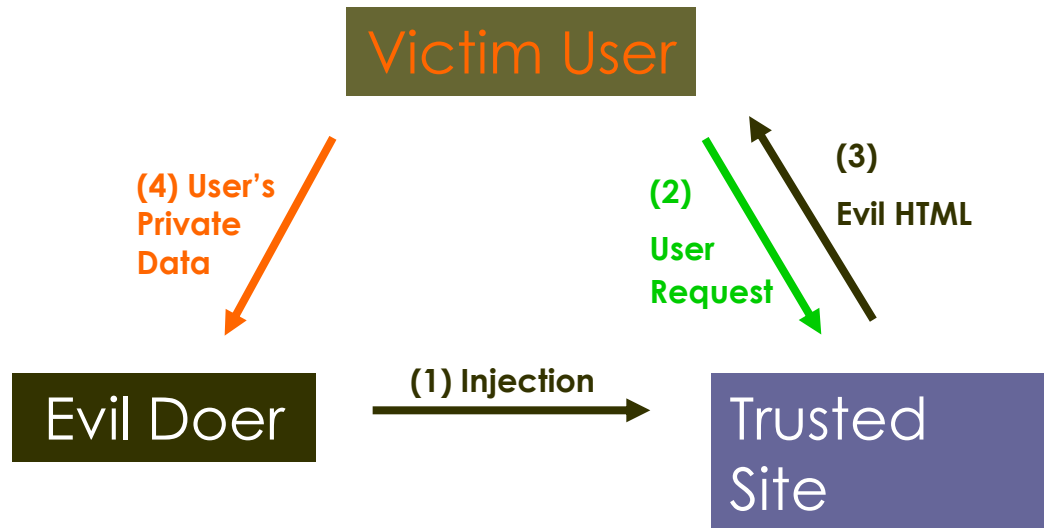
- **Code time**

SQL Injection

- **Cast to (int) whenever possible**
- **Use prepared statements if possible**
- **If prepared statements are not available escape everything using database-specific escaping functions**
- **Always validate data (ctype_*, preg_*, Zend_Filter_*)**
- **Only give your database user the permissions it needs.**
- **Never trust your users**

Cross Site Scripting (XSS)

- Used to trick the user



Cross Site Scripting (XSS)

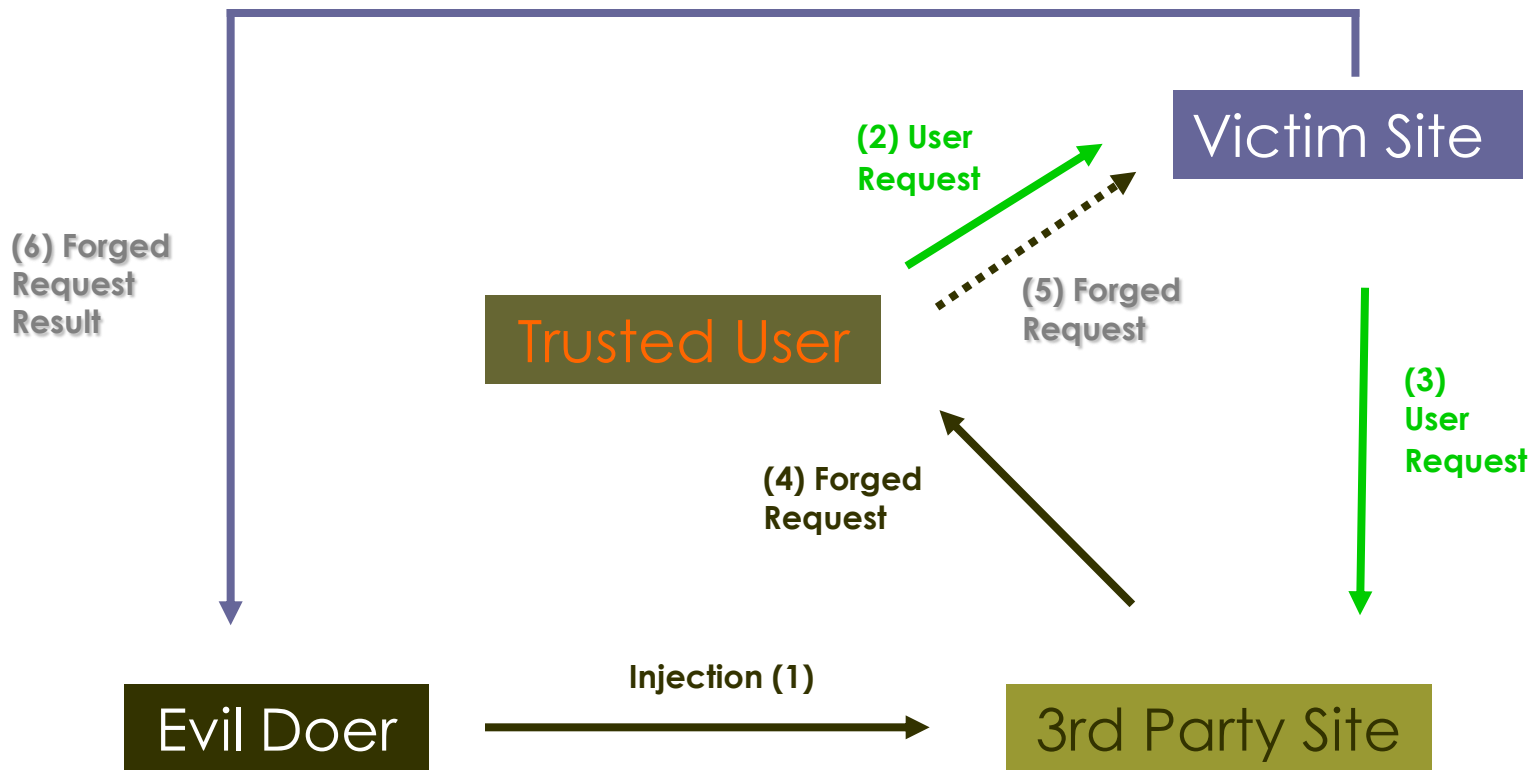
- **Code Time**

Cross Site Scripting (XSS)

- **Always escape user data (htmlspecialchars, htmlspecialchars, strip_tags)**
- **Employ a whitelist for places where HTML input is required (!)**
- **Use ctype_digit and ctype_alnum for simple fields like names or phone numbers**
- **Don't limit validation to Javascript**
- **Never trust your users**

Cross Site Request Forgery (XSRF)

- Used to trick the server



Cross Site Request Forgery (XSRF)

- **Use a site that relies on a user's identity**
- **Exploit the website's trust in that user**
- **Trick the user's browser into sending HTTP requests**
 - Cause the user's browser to execute an action or retrieve data on your behalf on that site

Cross Site Request Forgery (XSRF)

- **Code Time**

Cross Site Request Forgery (XSRF)

- **Use tokens that expire during sensitive operations**
- **Force session timeouts**
- **Never trust your users**

Command Injection

- **Used to execute programs on your server**

Command Injection

- **Code Time**

Command Injection

- **Don't use `exec`, `system`, `popen`, `shell_exec`, etc. in your program**
- **If you need to use those functions use hard coded values. Do not trust a variable or anything defined in another file**
- **If you need to have user input always use `escapeshellargs` and `escapeshellcmd`**
- **Never trust your users**

Remote Code Injection

- **Used to run an attacker's PHP code on your system**

Remote Code Injection

- **Code Time**

Remote Code Injection

- **Never use unchecked/unfiltered data in require|include(|_once)**
- **Set allow_url_include to false**
- **If you must make remote requests always filter any user provided data**
- **Use eval() judiciously**
- **Never trust your users**

Session Hijacking

- **Often used in conjunction with XSS**
- **Attacker retrieves a user's session ID and uses it as their own**
- **Can be used in conjunction with document.cookie**

Session Hijacking

- **Code Time**

Session Hijacking

- **Use `htmlspecialchars`, `htmlspecialchars` or `strip_tags` to disable JavaScript-based attacks**
- **Use `session_regenerate_id(true)`**
- **Validate a session against an IP address**
 - Note that this should be used to generate an alert, not restrict a user's access

Session Fixation

- **Used to piggy back on someone's session by setting the user's session to the attackers**

Session Fixation

- **Code Time**

Session Fixation

- **Difficult to guard against**
- **Use `session_regenerate_id(true)`**
 - before logging in
 - periodically in a user's session
 - if the domain in the `HTTP_REFERER` doesn't match the current domain
 - None of these are foolproof, but they limit the ability of an attacker to fixate a session
- **Disable the use of the session ID in the URL**
 - Still able to change the session ID using JavaScript, though

Cookie Forging

- **Forging cookies that are used to determine permissions or access**

Cookie Forging

- **Code Time**

Cookie Forging

- **Don't use cookies. Use the session handler**
- **If you must use cookies, encrypt the contents**
 - Do not hash the contents
 - `$_COOKIE['isAdmin'] = md5(1);`
 - Same for hacker as it is for you

Miscellaneous good ideas

- **Turn `display_errors` off**
- **Do not use `register_globals`**
- **Keep as much code and data out of the public code tree (`htdocs/wwwroot`) as possible**
- **Use a whitelist approach when dealing with HTML**

What about buffer overflows and such

- **Very few of those weaknesses occur in PHP**
- **When they do they are usually in extension interfaces or the extensions themselves, not PHP**
- **Disable all unused streams, extensions, filters, etc.**

Concluding Thoughts

- **Never trust your users**
- **Always filter your data**
- **Security through obscurity IS useful, just don't bet the farm on it**

- **For the full Zend security course check out**
 - http://www.zend.com/education/php_training_courses

Obligatory Plug

- **Zendcon – September 15-18**
 - Theme – High Impact PHP
 - One of the worlds largest gathering of PHP people

Contact me

Kevin Schroeder

kevin@zend.com