Secure Coding and Code Review

Berlin : 2012
Outline

- Overview of top vulnerabilities
- Code review practice
- Secure design / writing secure code
- Write some secure code
- Review a volunteer's code
Top Problems

- Cross-Site Scriptng (xss)
- Cross-Site Request Forgery (csrf)
- Register Globals
- SQL Injection
XSS

- An attacker is able to inject client-side scripting into a web page, executed by others. May, or may not, be cross-domain.

- Can result in:
  - Authenticated Requests by victim
  - Session hijacking
  - Click jacking
  - Propagation of Script (xss worm)
  - Internal network access / portscanning
Reflected XSS

• Javascript in the request is written to the page

• `<input type="text" name="search_term" value="<? echo $_GET['search_term']; ?>" />

• And if someone sends you a link: “page.php?search_term=”><script>alert()</script> <!--”?
2\textsuperscript{nd} Order (Stored) XSS

- Attacker-controlled data is stored on the website, and executable scripts are displayed to the viewer (victim)
3rd Order (Dom-based) XSS

- Attacker controls existing DOM manipulations in a way that generates attacker-influenced execution of scripts
Cross-Site Script Inclusion (xssi)

- (or “Javascript Hijacking”)
- A script with sensitive data is included and manipulated from another domain
Preventing XSS

- Validate your input
- Escape your output

Mediawiki Tools:
- Html, Xml, Sanitizer classes
- jQuery elements
- Jsonp api runs as anonymous
Additional Reading

- For the theories behind XSS, and why certain filters should be applied, read: https://www.owasp.org/index.php/XSS_(Cross_Site_Scripting)_Prevention_Cheat_Sheet
- Quick reference of how to escape data in different HTML/document contexts: https://www.owasp.org/index.php/Abridged_XSS_Prevention_Cheat_Sheet
Additional Reading on SOP

- Understanding cross-domain aspects of XSS requires knowledge of the Same Origin Policy (SOP) you are dealing with
  - https://www.owasp.org/index.php/File:SameOriginPolicy.ppt
  - http://code.google.com/p/browsersec/wiki/Part2
- The SOP for javascript, XHR, and Flash are different!
Cross-Site Request Forgery (csrf)

- If a user has an authenticated session established to a secure site, a remote site can reference resources on that site, which will be requested with the authority of the logged-in user.

- A page on funnykitties.com can call the “image”:

  ```html
  ```
Preventing CSRF

- Tokens written into form just prior to editing, and checked when form is received
  - This is in *addition* to authentication / authorization checks
  - Tokens must be difficult to predict
- Mediawiki uses editToken
Register Globals

- If register_globals is on, then an attacker can set variables in your script
- If an attacker can control variables in your script, there is potential for
  - Remote File Inclusion
  - Altering code execution path
Register Global Vulnerabilities

- include($lang.".php");

- <?php
  //MyScript.php
  if ( authenticate( $_POST['username'], $_POST['pass'] ) ) {
    $authenticated = true;
  }
  if ( $authenticated ) {
    ...
  }
Register Global Protections

- Don't use globals in script paths
- Ensure your script is called in the correct context
  - if ( !defined( 'MEDIAWIKI' ) ) die( 'Invalid entry point.' );
- Sanitize defined globals before use
- Define security-critical variables before use as 'false' or 'null'
SQL Injection

- Poorly validated data received from the user is used as part of a database (SQL) statement.

- Can result in:
  - Authentication Bypass
  - Data Corruption
  - System Compromise
Preventing SQLi

- Use MediaWiki built-in database classes and pass key=>value pairs to the functions
- `select()`, `selectRow()`, `insert()`, `insertSelect()`, `update()`, `delete()`, `deleteJoin()`, `buildLike()`
Additional Top Web Vulnerabilities

- OWASP Top 10
Secure Design Principles

- Simplicity
- Secure by Default
- Secure the Weakest Link
- Least Privilege
Secure Coding Checklist

- Avoid eval, create_function
- Regex'es
  - Don't use /e
  - Escape with preg_quote()
- Filter / Validate your Inputs
  - intval(), getInt(), etc
  - Use a whitelist of expected values when possible
Secure Coding Checklist

- Use HTMLForm class, or include/check $wgUser->editToken
- Defend against register-globals
- Use Html and Xml helper classes
- Use Sanitizer::checkCss to use user's css
- Use database wrapper functions
- **Write clean, clearly commented code!**
Write Some Secure Code

- Create a Special Page that allows searching, and showing results
- Assume you have a database of important text data:
  - `CREATE table `myData` (`id` INT, `name` varchar(80), `body` TEXT);
- Present a search box
- Search the database for matches in `name` or `body`, display matches to user
Review a Volunteer's Code
Secure Coding and Code Review

Berlin : 2012
Outline

- Overview of top vulnerabilities
- Code review practice
- Secure design / writing secure code
- Write some secure code
- Review a volunteer's code
Top Problems

- Cross-Site Scriptng (xss)
- Cross-Site Request Forgery (csrf)
- Register Globals
- SQL Injection
XSS

- An attacker is able to inject client-side scripting into a web page, executed by others. May, or may not, be cross-domain.
- Can result in:
  - Authenticated Requests by victim
  - Session hijacking
  - Click jacking
  - Propagation of Script (xss worm)
  - Internal network access / portscanning
Reflected XSS

- Javascript in the request is written to the page
  
  `<input type="text" name="search_term" value="<? echo $_GET['search_term']; ?>" />

- And if someone sends you a link:
  
  "page.php?search_term="<script>alert()</script></script>"?"
2\textsuperscript{nd} Order (Stored) XSS

- Attacker-controlled data is stored on the website, and executable scripts are displayed to the viewer (victim)
- 
-
3rd Order (Dom-based) XSS

- Attacker controls existing DOM manipulations in a way that generates attacker-influenced execution of scripts
Cross-Site Script Inclusion (xssi)

- (or “Javascript Hijacking”)
- A script with sensitive data is included and manipulated from another domain
Preventing XSS

- Validate your input
- Escape your output

Mediawiki Tools:
  - Html, Xml, Sanitizer classes
  - jQuery elements
  - Jsonp api runs as anonymous
Additional Reading

- For the theories behind XSS, and why certain filter should be applied, read: https://www.owasp.org/index.php/XSS_(Cross_S
- Quick reference of how to escape data in different html/document contexts: https://www.owasp.org/index.php/Abridged_XSS.
Additional Reading on SOP

- Understanding cross-domain aspects of XSS requires knowledge of the Same Origin Policy (SOP) you are dealing with
  - http://code.google.com/p/browsersec/wiki/Part2
- The SOP for JavaScript, XHR, and Flash are different!
Cross-Site Request Forgery (csrf)

- If a user has an authenticated session established to a secure site, a remote site can reference resources on that site, which will be requested with the authority of the logged-in user.

- A page on funnykitties.com can call the “image”:
Preventing CSRF

- Tokens written into form just prior to editing, and checked when form is received
  - This is in **addition** to authentication / authorization checks
  - Tokens must be difficult to predict
- Mediawiki uses editToken
Register Globals

- If register_globals is on, then an attacker can set variables in your script
- If an attacker can control variables in your script, there is potential for
  - Remote File Inclusion
  - Altering code execution path
Register Global Vulnerabilities

- include($lang.".php");

- <?php
  //MyScript.php
  if ( authenticate( $_POST['username'], $_POST['pass'] ) ) {
    $authenticated = true;
  }
  if ( $authenticated ) {
    ...
  }
}
Register Global Protections

- Don't use globals in script paths
- Ensure your script is called in the correct context
  - if ( !defined( 'MEDIAWIKI' ) ) die( 'Invalid entry point.' );
- Sanitize defined globals before use
- Define security-critical variables before use as 'false' or 'null'
**SQL Injection**

- Poorly validated data received from the user is used as part of a database (SQL) statement
- Can result in:
  - Authentication Bypass
  - Data Corruption
  - System Compromise
Preventing SQLi

- Use MediaWiki built-in database classes and pass key=>value pairs to the functions
- `select()`, `selectRow()`, `insert()`, `insertSelect()`, `update()`, `delete()`, `deleteJoin()`, `buildLike()`
Additional Top Web Vulnerabilities

- OWASP Top 10
Secure Design Principles

- Simplicity
- Secure by Default
- Secure the Weakest Link
- Least Privilege
Secure Coding Checklist

- Avoid eval, create_function
- Regex’es
  - Don't use /e
  - Escape with preg_quote()
- Filter / Validate your Inputs
  - intval(), getInt(), etc
  - Use a whitelist of expected values when possible
Secure Coding Checklist

- Use HTMLForm class, or include/check $wgUser->editToken
- Defend against register-globals
- Use Html and Xml helper classes
- Use Sanitizer::checkCss to use user's css
- Use database wrapper functions
- **Write clean, clearly commented code!**
Write Some Secure Code

- Create a Special Page that allows searching, and showing results
- Assume you have a database of important text data:
  - `CREATE table `myData` (`id` INT, `name` varchar(80), `body` TEXT);
- Present a search box
- Search the database for matches in `name` or `body`, display matches to user
Review a Volunteer's Code