

Secure Web App Programming

Akash Mahajan

Talk at NULL Bangalore Meeting

8th August 2009

Cross Site Scripting - XSS

- Injecting HTML/JS into the site.
 - Non-persistent/Reflected/First Order
 - Script is taken from the request and displayed in the browser directly
 - [example.com/search?q=<script>alert\('hi'\);</script>](http://example.com/search?q=<script>alert('hi');</script>)
 - Example.com/index.php?lang=path to php shell
 - Persistent/Stored/Second Order
 - First name of a registration form is vuln and the value is stored in the database
 - Hello <iframe src=http://f1y.in/0.js></iframe>
 - DOM Based
 - No example, mentioned by Amit Klien in his paper XSS of the Third Kind

XSS mitigation in PHP

- Sanitize all globals (`$_GET`, `$_POST`, `$_COOKIE`)
 - Use `strip_tags()`
 - Use `inspekt` library code.google.com/p/inspekt
- Escape everything before displaying
 - `htmlspecialchars()`, `htmlspecialchars()`
- Client headers like user agent can be malicious as well.
- Thumb rule, if its not your data consider it bad. If you can verify it, consider it trusted good data.
- White listing helps in verifying good data more than black listing.
- See examples at xssed.com

SQL Injection

- Allowing SQL to be injected in the database query.
- Most common attack point is the search of any dynamic website and registration forms. These two will be definitely talking to the database.
- `$sql = "SELECT * FROM table WHERE id = " . $_REQUEST['id'] . """;`
- `id = ' OR 1 UNION ALL SELECT * FROM table;`
- Excellent examples
<http://google.com/search?q=site:slideshare.net sql injection>

SQL Injection - Mitigation

- `mysql_real_escape_string()`
 - `$dbquery = sprintf("SELECT name FROM user WHERE id='%s'", mysql_real_escape_string('id'));`
- Parameterized queries
 - `$res = $query("SELECT name FROM user WHERE id=?", $id);`
 - Standard mysql module in PHP doesn't allow for parameterized queries. You need `mysqli`
- Stored Procedures
 - See a kickass example of stored proc used to hack more than hundred thousand websites
 - <http://www.breach.com/resources/breach-security-labs/alerts/mass-sql-injection-attack-evolutio>

File Uploads

- Web apps add a directory in document root for storing file uploads and give write access.
- They don't randomize filenames. So a specially crafted image file which has PHP code written in it gets saved there.
- The malicious user is now free to call it using a GET request and it gets executed.
- <http://www.scanit.be/uploads/php-file-upload.pdf>

File Uploads - Mitigation

- The usual use case is uploading of image files.
- Use `getimageinfo()` to get the correct mime type of the file from the file header.
- Generate a random file name
 - `$rand = time() . substr(md5(microtime()), 0, rand(5, 12));`
 - Return `$rand` and append file extension
- Ideally `noexec` permission should be set on the directory where files are copied to.

Endgame

- At this point you have reasonable ensured that your PHP web application is not compromised.
- But the user connecting to your website are vulnerable to session hijacking, CSRF from your site etc.
- There are work around to the standard PHP functions like this one for `mysql_real_escape_strings()`
 - <http://shiflett.org/blog/2006/jan/addslashes-versus-mysql-real-escape-string>