Web Application Hacking

How to Make and Break Security on the Web

By Wesley Aptekar-Cassels
SQL Injection - Example

The user inputs both **NAME** and **PASSWORD** directly into the SQL statement.

```
SELECT * FROM `users` WHERE name = 'NAME' AND password = 'PASSWORD';
```

```
SELECT * FROM `users` WHERE name = 'NAME' AND password = '' OR 'x'='x';
```
SQL Injection - Protection

`mysqli_real_escape_string()`

```php
echo mysql_real_escape_string("' OR 'x'='x");
\' OR \'x\'=\'x
```

Appends backslashes before unsafe characters.

Escape everything that might ever touch a SQL statement.
Broken Authentication & Session Management

- **Passwords**
  - Hashed and salted
  - HTTPS

- **Session IDs**
  - Don’t use URL rewriting
  - Set reasonable session timeouts
  - Invalidate session after logout
  - HTTPS
Cross Site Scripting (XSS) - Example

Input:
<script>alert('HACKED');</script>

Output:
Cross Site Scripting - Protection

`htmlentities()`

echo htmlentities("<script>alert('HACKED')</script>");
&lt;script&gt;alert('HACKED')&lt;/script&gt;
Insecure Direct Object References - Example

GET http://example.com/delete_post.php?id=5

Delete link is only shown on users own posts, but works for any post. (More often used when talking about accessing data, but applies to functions as well)
Insecure Direct Object References - Protection

```php
if (user_has_permission($user)) {
    do_action();
} else {
    echo 'GTFO';
}
```

It’s that simple.
Security Misconfiguration

- Out of date software
- Unnecessary ports/services/features
- Default passwords
- Error handling displayed to user in production
- Extra security settings
- And more…
Sensitive Data Exposure

- Sensitive data in plaintext
- Data transmitted in plaintext
- Old cryptography used (md5, etc.)
- Private key management
- And more...
Missing Function Level Access Control

Same as Insecure Direct Object References, but refers more to functions and user actions.
Cross Site Request Forgery - Example

Browser sends GET request, and gets invalid data back (HTML). This can trick any user into sending a request to any page. (Javascript required for post requests on the same domain).

The important bit is that www.evilsite.com can send a request to www.example.com, and the user that sees it will have the privileges they do on www.example.com.

Can also be done on same domain as the request is being sent to.
Add a CSRF token. Random string that is generated by the server and MUST be sent by the browser for the request to go through.

Origin header is not a solution, as it is broken in firefox. ([Bug ID 446344](https://bugzilla.mozilla.org/show_bug.cgi?id=446344))
Using Components with Known Vulnerabilities

- Try to use components that you write
- Monitor CVE, NVD, mailing lists, etc.
- Use up to date software
- chroot and use users with few privileges
- Remember that all software is or will be broken
Unvalidated Redirects and Forwards - Example


Not very dangerous, but evilsite can be phishing, malware, etc.

Can be a security feature if it prompts to leave the site ("You are leaving example.com! Continue? Y/N")
if ($_SERVER['HTTP_HOST'] == parse_url($_GET['url'])) {
    // Go ahead!
} else {
    // NOPE!
}
And much more…

- Directory traversal
- Poison null byte
- Buffer Overflow (yes, even in webapps)
- etc.
Thank you!

Any questions?