Chapter 1: Security

Security on the web has multiple facets

- Client side
- Server side

High value targets

- Private information
- Financial information

What is your website suppose to do?

Is your website suppose to:

- Be a distributor of malware?
- Vouch for the identity of frausters?
- Run arbitrary code?
- Distribute pirated software/media?
- Host pornography
- Do anything you didn't want it to?

Web Content Takeover (XSS)

Imagine you ask a user for a username. They provide this:

```
<iframe width="100%" height="100%
↪  src="http://cnn.com/"></iframe>
```

I assume this wasn't your intent, you just wanted to show a username. This happened because you didn't properly encoded the output such that it escapes as HTML.

Common XSS

Often values are printed as URIs or attributes:

- e.g., `<a href="http://example.com/user/%s">`
- The simplest XSS exploit is to pass a " and wreck that tag like `name="><script>alert("xss");</script><`
- For provide the color for your username like `color=FFFFFF`

Solution

- Never print out anything from the user (easier said than done)
- Validate all values you embed in HTML
- Appropriately encode all values
  - URI Encode URIs, don't just concatenate
  - HTML Escape HTML entities
- Use a templater that will automatically escape everything for you
- Don't use innerHTML in Javascript. Use .html and .text in Jquery or new Text(text) in Javascript.

XSS is a big deal because if the website trusts the user and the attacker can inject content, they can inject javascript or other tags and execute commands on the website.

Cross-Site Request Forgery (CSRF)

- Trick a user or user agent in executing unintended requests.
- Hijack weak authentication measures - Cookies and sessions
- Repeat actions unnecessarily

Solution

- Enforce referrer headers (still not perfect)
- Request tokens - don't allow repeated requests
- Make GET/HEAD/OPTIONS safe - /logout should not be a GET
- Avoid any chance of XSS
- Don't rely on cookies, rely on full HTTP auth
- Don't allow users to provide URLs that get embedded!
- Rely on matching cookies
- Origin header - Make sure it comes from a trusted source
- Challenge Response - Make the client provide extra information like re-login, password, captcha, two factor, etc

Improper Limitation of a Pathname to a Restricted Directory

Access files and URIs that weren't supposed to be exposed!

Solution

Often the solution many people do is inadequate. You should try get absolute path and check if the path is within the safe parent Directory then allow the request otherwise deny. Or if you detect path traversal, maybe you should just deny them access?

Inclusion of Functionality from Untrusted Control Sphere

- Lots of services want you to include iframes and embeddings from them.

    - They make you trust them not to ruin your site.
    - Lots of advertisement networks expect the same from you.
- When included untrusted content there can be consequences, whether by iframe or actual values.

SQL Injection

When a user make a valid SQL operation from an input field and makes unauthorized SQL operations.

SQL Injection Patterns

- Breaking quotes
- Returning all values with 1 or 1=1
- Making multiple statements
- Selecting ALL passwords from the database
- Vandalism: dropping tables

Solution

SQL Quote all values,

- Use the SQL execute statement
- NEVER craft a SQL query purely from input strings
- ESCAPE ESCAPE ESCAPE

Poorly Encrypted Cookies/Tokens

- The web is stateless! Why not rely on the user to hold the state?
- So let's set application state in the user's cookie so we don't need to use a database to store their session

  The dangers for

Shell Injection

DOS and DDOS