CSE543 - Computer and Network Security

Module: Web Security

Professor Trent Jaeger
Network vs. Web Security
Web Vulnerabilities

- Web vulnerabilities surpassed OS vulnerabilities around 2005
  - The “new” buffer overflow

![Graph showing Web (XSS) and Buffer Overflow trends from 2001 to 2006]
Components of the Web

- Multiple interacting components

Clients (Browsers) → HTTP Servers → Web Applications → Backend
Components of the Web

- Multiple interacting components

Clients (Browsers) → Multiple interacting components → HTTP Servers → Backend → Web Applications
Components of the Web

- Multiple interacting components

Clients (Browsers) → Web Applications → Backend

- HTTP Servers
- Apache
- MySQL
Components of the Web

- Multiple interacting components

Clients (Browsers) → HTTP Servers → Backend

Web Applications

Clients (Browsers) → Web Applications

Clients (Browsers) → HTTP Servers

Web Applications → Backend
Web security: the high bits

- The largest distributed system in existence
- Multiple sources of threats, varied threat models
  - Users
  - Servers
  - Web Applications
  - Network infrastructure
- We shall examine various threat models, attacks, and defenses
- Another way of seeing web security is
  - Securing the web infrastructure such that the integrity, confidentiality, and availability of content and user information is maintained
Early Web Systems

- Early web systems provided a click-render-click cycle of acquiring web content.
  - Web content consisted of static content with little user interaction.
Adding State to the Web: Cookies

• Cookies were designed to offload server state to browsers
  ‣ Not initially part of web tools (Netscape)
  ‣ Allows users to have cohesive experience
  ‣ E.g., flow from page to page,
• Someone made a design choice
  ‣ Use cookies to authenticate and authorize users
  ‣ E.g. Amazon.com shopping cart, WSJ.com
• Q: What is the threat model?
Cookie Issues ...

• New design choice means
  ‣ Cookies must be protected
    • Against forgery (integrity)
    • Against disclosure (confidentiality)
• Cookies not robust against web designer mistakes, committed attackers
  ‣ Were never intended to be
  ‣ Need the same scrutiny as any other tech.

Many security problems arise out of a technology built for one thing incorrectly applied to something else.
Cookie Design 1: mygorilla.com

- Requirement: authenticate users on site mygorilla.com

- Design:
  1. set cookie containing hashed username
  2. check cookie for hashed username

- Q: Is there anything wrong with this design?
Cookie Design 2: mygorilla.com

• Requirement: authenticate users on site mygorilla.com

• Design:
  1. set cookie containing encrypted username
  2. check cookie for encrypted username

• Q: Is there anything wrong with this design?
Cookie Design 2: mygorilla.com

- Requirement: authenticate users on site mygorilla.com

- Design:
  1. set cookie containing encrypted + HMAC’d username
  2. check cookie for encrypted + HMAC’d username

- Q: Is there anything wrong with this design?
Exercise: Cookie Design

- Design a secure cookie for mygorilla.com that meets the following requirements
  - Requirements
    - Users must be authenticated (assume digest completed)
    - Time limited (to 24 hours)
    - Unforgeable (only server can create)
    - Privacy-protected (username not exposed)
    - Location safe (cannot be replayed by another host)
Exercise: Cookie Design

• Design a secure cookie for mygorilla.com that meets the following requirements
• Requirements
  ‣ Users must be authenticated (assume digest completed)
  ‣ Time limited (to 24 hours)
  ‣ Unforgeable (only server can create)
  ‣ Privacy-protected (username not exposed)
  ‣ Location safe (cannot be replayed by another host)

\[
E\{k_s, "host_ip : timestamp : username" \}
\]
Exercise: Cookie Design

• Design a secure cookie for mygorilla.com that meets the following requirements

• Requirements
  ▸ Users must be authenticated (assume digest completed)
  ▸ Time limited (to 24 hours)
  ▸ Unforgeable (only server can create)
  ▸ Privacy-protected (username not exposed)
  ▸ Location safe (cannot be replayed by another host)

\[ \text{User} \leftrightarrow \text{Server} \]

\[ E\{k_s, "host_ip : timestamp : username"\} + \text{HMAC}\{k_s, "..."\} \]
Content from Multiple Sites

• Browser stores cookies from multiple websites
  ‣ Tabs, mashups, ...

• Q. What is the threat model?

• More generally, browser stores content from multiple websites
  ‣ HTML pages
  ‣ Cookies
  ‣ Flash
  ‣ Java applets
  ‣ JavaScript

• How do we isolate content from multiple sites?
Same-Origin Policy

• A set of policies for isolating content across different sites (origins)

• What is an origin?
  ‣ site1.com vs site2.com?
    • Different hosts are different origins
  ‣ http://site.com vs https://site.com?
    • Different protocols are different origins
    • Different ports are different origins
  ‣ http://site1.com vs http://a.site1.com?
    • Establishes a hierarchy of origins

• Origin: host:protocol:port
Same-Origin Policy

- **Principle:** Any active code from an origin can read only information stored in the browser that is from the same origin
  - Active code: Javascript, VBScript,…
  - Information: cookies, HTML responses, ...

![Diagram showing Same-Origin Policy](image)
Document Domain

- Scripts from **two origins in the same domain** may wish to interact
  - www.example.com and program.example.com
- Any web page may set `document.domain` to a
  - “right-hand, fully-qualified fragment of its current host name” (example.com, but not ample.com)
- Then, **all scripts** in that domain may share access
  - All or nothing
- NOTE: Applies “null” for port, so does not actually share with normal example.com:80
SOP Weaknesses

- Complete and partial bypasses exist
  - Browser bugs
  - Corner cases
  - Functionality often requires SOP bypass!
    - Many advertisement companies hire people to find and exploit SOP browser bugs for cross-domain communication
    - E.g., JSON with padding (JSONP)
- Cross-site scripting
  - Execute scripts from one origin in the context of another
Cross-Site Scripting

- Assume the following is posted to a message board on your favorite website:

  Hello message board.

  `<SCRIPT>malicious code</SCRIPT>
   This is the end of my message.`

- Now a reasonable ASP (or some other dynamic content generator) uses the input to create a webpage (e.g., blogger nonsense).

- Now a malicious script is now running
  - Applet, ActiveX control, JavaScript…
Cross-Site Scripting

• Script from attacker is executed in the victim origin’s context
  ‣ Enabled by inadequate filtering on server-side

• Three types
  ‣ Reflected
  ‣ Stored
  ‣ DOM Injection
Reflected XSS

```php
<?php
$name = $_GET['name'];
echo "Welcome $name<br>";
?>

<form method="get" action="index.php">
    Name: <input type="text" name="name" /><br />
    <input type="submit" value="submit" />
</form>
```

`index.php?name=guest<script>alert('hi');</script>`
Web Systems Evolve ...

• The web has evolved from a *document retrieval* and rendering to sophisticated *distributed application platform* providing:
  ‣ dynamic content
  ‣ user-driven content
  ‣ interactive interfaces
  ‣ multi-site content
  ‣ ....

• With new interfaces comes new vulnerabilities ...
AJAX / “Web 2.0”

- **AJAX**: asynchronous JavaScript and XML
  - A collection of approaches to implementing web applications
  - Changes the click-render-click web interface to allow webpages to be interactive, change, etc.
  - Examples: Google Gmail/Calendar, Facebook, ...
  - Hidden requests that replace document elements (DOM)
  - DOM XSS caused by JavaScript modifying DOM elements without sanitizing input
Cross-site Request Forgery

- An XSS attack exploits the trust the browser has in the server to filter input properly
- A CSRF attack exploits the trust the server has in a browser
  - Authorized user submits unintended request
    - Attacker Maria notices weak bank URL
    - Crafts a malicious URL
    - Exploits social engineering to get Bob to click the URL
  - Can make attacks not obvious

Defense: Referer header
- Bank does not accept request unless referred to (linked from) the bank’s own webpage
- Disadvantage: privacy issues
HTTP Response Splitting

• Again, due to insufficient server-side filtering
  ‣ Cookies can be set to arbitrary values to split HTTP response

```java
String author = request.getParameter(AUTHOR_PARAM);
...
Cookie cookie = new Cookie("author", author);
cookie.setMaxAge(cookieExpiration);
response.addCookie(cookie);
```

- Can be used for page hijacking through proxy server

HTTP/1.1 200 OK
...
Set-Cookie: author=Jane Smith
...
HTTP/1.1 200 OK
...
Set-Cookie: author=Wiley Hacker
HTTP/1.1 200 OK
...
Session Hijacking

- Virtual sessions are implemented in many ways
  - session ID in cookies, URLs
  - If I can *guess, infer, or steal* the session ID, game over
  - Login page using HTTPS, but subsequent communication is not! Cookies sent in cleartext
  - If your bank encodes the session ID in the url, then a malicious attacker can simply keep trying session IDs until gets a good one.
  - ... note that if the user was logged in, then the attacker has full control over that account.
  - Countermeasure: HTTPS, secure cookie design
Privacy

• Have you ever …
  ‣ Searched for a product on some website
  ‣ ...Advertisement for the same product shows up on another website?
  ‣ **Reason:** Tracking! Profile users for targeted advertisement

• Study by WSJ found (2012)
  ‣ 75% of top 1000 sites feature social networking plugins
    • Can match users’ identities with web-browsing activities

• abine and UC Berkeley found
  ‣ Online tracking is 25% of browser traffic
    • 20.28% google analytics
    • 18.84% facebook

http://www.abine.com/
Privacy

- Tracking is done when one site embeds content in another

- “Tracker” code is from
  - Social networking sites
  - Analytics
  - Advertisement agencies
  - ...

Protecting Browser State from Web Privacy Attacks: Jackson et al.
Privacy

• Objective of tracking code is to maintain state of users across multiple sites
  ‣ Build profile of sites visited
• Semi-cooperative tracking done by
  ‣ Javascript
    • e.g., Cached redirect URLs
  ‣ Web bugs
    • 1x1 images
    • Ever wondered why email clients have “Display images”?
  ‣ IFrames
  ‣ Cookies
    • Traditional, flash, HTML5 LocalStorage, …
• Tasks: (1) get your tracking code running; (2) store state; (3) send to server
Third-Party Cookies

• A third-party cookie is a cookie from a website different from the website being viewed
• Browsers can block third-party cookies
  ‣ Different browsers have different variations
    • Some completely block
    • “Do Not Track” - except Chrome
• Limitation
  ‣ Other ways exist to store state
    • HTML5 LocalStorage
    • Redirect caching
    • ETags - https://lucb1e.com/rp/cookielesscookies/
Third-Party Cookies

- A third-party cookie is a cookie from a website different from the website being viewed.
- Browsers can block third-party cookies.
  - Different browsers have different variations.
    - Some completely block.
    - “Do Not Track” - except Chrome.
- Limitation.
  - Other ways exist to store state (more).
    - Canvas fingerprinting.
    - Evercookies.
    - “Cookie syncing”.
- OpenWPM - [https://github.com/citp/OpenWPM](https://github.com/citp/OpenWPM).
Unintended Tracking

• “Data” access not all governed by same-origin policy
  ‣ Specified: HTML DOM, cookies
  ‣ What about
    • Web caches?
      ‣ Tracking notes time to fetch URL
      ‣ If URL in cache, served faster
    • Visited links?
      ‣ Mostly fixed in current browsers
  • **Take-away**: Difficult to prevent tracking if any browser state is stored

• To mitigate tracking
  ‣ Reset browser regularly, store no state, visit random sites!
Browsers

• Browsers are the new operating systems
• Huge, complex systems that support
  ‣ Many document types, structures, e.g., HTML, XML, ...
  ‣ Complex rendering, e.g., CSS, CSS 2.0
  ‣ Many “program/scripting” languages, e.g., JavaScript
  ‣ Dynamic content, e.g., AJAX
  ‣ Native code execution, e.g., ActiveX

• Virtualized computers in a single program ...
Browser Security

- We don’t have the ability to control this much complexity, so we have to try other things...
  - Restricting functionality, e.g., NoScript
  - Process Isolation, e.g., OP, Chrome
    - Read: http://www.google.com/googlebooks/chrome/
OP Browser

• What did they do to build a more secure browser?
• (1) Decompose the browser into multiple processes
  • Called “Privilege Separation”
• What are the permissions of a set of processes forked from the same parent?
OP Browser

- What did they do to build a more secure browser?
- (1) Decompose the browser into multiple processes
  - Called “Privilege Separation”
- What are the permissions of a set of processes forked from the same parent? Same as parent
- (2) Need different policy for each process
  - Multiple subjects in the access control policy
- What browser processes are trusted to manage the permissions?
OP Browser

- What did they do to build a more secure browser?
  - (1) Decompose the browser into multiple processes
    - Called “Privilege Separation”
  - What are the permissions of a set of processes forked from the same parent? Same as parent
  - (2) Need different policy for each process
    - Multiple subjects in the access control policy
  - What browser processes are trusted to manage the permissions? None
  - (3) Need mandatory access control
    - Subjects cannot escape confined “protection domain”
OP Browser

- How do you determine what parts of the browser should be a “subject” and identify the permissions to be assigned to that subject?
  - One subject (client)
    - Code that requires the same permissions to run
    - E.g., a particular web page
  - Another subject (server)
    - Code that manages the same permissions
    - E.g., UI, network, and storage subsystems
- How do we determine the permission assignments?
OP Browser

• How do you determine what parts of the browser should be a “subject” and identify the permissions to be assigned to that subject?

• One subject (client)
  • Code that requires the same permissions to run
  • E.g., a particular web page

• Another subject (server)
  • Code that manages the same permissions
  • E.g., UI, network, and storage subsystems

• How do we determine the permission assignments?
  • Least privilege
  • Information flow
JavaScript

• Scripting Language used to improve the quality/experience
  ‣ Create dialogs, forms, graphs, …
  ‣ Built upon API functions (lots of different flavors)

• Security: No ability to read local files, open connections, but …
  ‣ Spoofing – easy to create “password” dialogs
  ‣ Eval - Can inject data to be executed
  ‣ Difficult to write secure JavaScript code - XSS, XSRF, Request Splitting, etc.
Applications/Plugins

• A plugin is a simply a program used by a browser to process content
  ‣ MIME type maps content to plugin
  ‣ Like any old application (e.g., RealAudio)
  ‣ Newer browsers have autoinstall features

• Plugins are sandboxed, but have been circumvented in various ways
  ‣ Interesting design point - Google Chrome allows “native” plugins but still preserves (some) security!
    • Native Client sandbox for running compiled C/C++ code

• Moral: beware of plugins
Social Engineering

• Attacks another weak point -- users!
• Phishing
  ‣ Lure users using bait (fishing) to steal valuable information
  ‣ Common technique: mimic original site and use similar URL
    • www.aol.com vs www.ao1.com
    • Combine with other techniques e.g., turn off address bar
Drive by downloads

- Using a deceptive means to get someone to install something on their own (spyware/adware)
  - Often appears as an error message on the browser
  - Sometimes, user does not click anything at all!
  - **Concern**: *extortion-ware* -- pay us $ to unencrypt your data
    - Used to demand $ for uninstall of annoying software
  - “biggest cybersecurity threat” - Kaspersky
- **Answer**: Back up stuff externally that you really want!
Content Security Policies

- Recent computer security standard to prevent (May 2016)
  - XSS, clickjacking, and other code injection attacks
- Invent as “Content Restrictions” in 2004 for Firefox

- If “Content-Security-Policy” header is present in a server response, a compliant client enforces the declarative whitelist policy
  - Which means several features are disabled by default
    - Inline JavaScript (script tags), Inline CSS (style tags), Dynamic JavaScript (eval), Dynamic CSS
- Unfortunately, researchers are already finding these whitelists to be sources of errors, permitting exploits
Web Applications: Injection

- Attacker that can inject arbitrary inputs into the system can control it in subtle ways
  - *interpreter injection* - if you can get PHP to “eval” your input, then you can run arbitrary code on the browser ...
  - e.g., leak cookies to remote site (e.g., session hijacking)

```php
$INPUT = "Alice\;mail($to, $subject, $body);"
```

- *filename injection* - if you can control what a filename is in application, then you can manipulate the host
  - Poorly constructed applications build filename based on user input or input URLs, e.g., hidden POST fields
    - Examples: Directory traversal, PHP file inclusion
  - e.g., change temporary filename input to ~/.profile

```html
<FORM METHOD=POST ACTION="../cgi-bin/mycgi.pl">
<INPUT TYPE="hidden" VALUE="~/.profile" NAME="LOGFILE">
</FORM>
```
SQL Injection

• An injection that exploits the fact that many inputs to web applications are
  ‣ under control of the user
  ‣ used directly in SQL queries against back-end databases
• Bad form inserts escaped code into the input ...
  
  xUserId = getRequestString("UserId");
  txtSQL = "SELECT * FROM Users WHERE UserId = " + xUserId;

• This vulnerability became one of the most widely exploited and costly in web history.
  ‣ Industry reported as many as 16% of websites were vulnerable to SQL injection in 2007
  ‣ This may be inflated, but clearly an ongoing problem.
**SQL Injection**

- An injection that exploits the fact that many inputs to web applications are
  - under control of the user
  - used directly in SQL queries against back-end databases
- Bad form inserts escaped code into the input ...

```sql
SELECT email, login, last_name
FROM user_table
WHERE email = 'x'; DROP TABLE members; --';
```

- This vulnerability became one of the most widely exploited and costly in web history.
  - Industry reported as many as 16% of websites were vulnerable to SQL injection in 2007
  - This may be inflated, but clearly an ongoing problem.
Preventing SQL injection

• From Unsafe SQL

```java
String query = "SELECT account_balance FROM user_data WHERE user_name = " + request.getParameter("customerName");

try {
    Statement statement = connection.createStatement( ... );
    ResultSet results = statement.executeQuery( query );
}
```

• Prepared SQL statements

```java
String custname = request.getParameter("customerName"); // REALLY be validated too
// perform input validation to detect attacks
String query = "SELECT account_balance FROM user_data WHERE user_name = ? ";

PreparedStatement pstmt = connection.prepareStatement( query );
pstmt.setString( 1, custname);
ResultSet results = pstmt.executeQuery( );
```

• **Other approaches:** have built (static analysis) tools for finding unsafe input code and (dynamic tools) to track the use of inputs within the web application lifetime.
Preventing Web System Attacks

• Largely just applications
  ‣ In as much as application are secure
  ‣ Command shells, interpreters, are dangerous
• Broad Approaches
  ‣ Validate input (also called input sanitization)
  ‣ Limit program functionality
    • Don’t leave open ended-functionality
  ‣ Execute with limited privileges
  ‣ Input tracking, e.g., taint tracking
  ‣ Source code analysis, e.g., c-cured
Conclusion

• Web security has to consider threat models involving several parties
  ‣ Web browsers
  ‣ Web servers
  ‣ Web applications
  ‣ Users
  ‣ Third-party sites
  ‣ Other users

• Security is so difficult in the web because it was largely *retrofitted*