CSE543 - Computer and Network Security

Module: Web Security

Professor Trent Jaeger
What is the web?

- A collection of application-layer services used to distribute content
  - Web content (HTML)
  - Multimedia
  - Email
  - Instant messaging

- Many applications
  - News outlets, entertainment, education, research and technology, …
  - Commercial, consumer and B2B
Web security: the high bits

• The largest distributed system in existence
  ‣ threats are as diverse as applications and users
  ‣ But need to be thought out carefully …

• The stakeholders are …
  ‣ Consumers (users, businesses, *agents*, …)
  ‣ Providers (web-servers, IM services, …)

• 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
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
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. use digest authentication to login user
  2. set cookie containing hashed username
  3. 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. use digest authentication to login user
  2. set cookie containing encrypted username
  3. check cookie for encrypted 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)

\[ E\{k_s, "host_ip : timestamp : username"\} \]
Web Transport Security: SSL

• Secure socket Layer (SSL/TLS)
• Used to authenticate servers
  › Uses certificates, “root” CAs
• Can authenticate clients
• Inclusive security protocol
• Security at the socket layer
  › Transport Layer Security (TLS)
  › Provides
    • authentication
    • confidentiality
    • integrity
SSL Handshake
(1) Client Hello (algorithms,...)
(1) **Client Hello (algorithms,…)**

(2) **Server Hello (alg. selection,…)**
SSL Handshake

(1) Client Hello (algorithms,…)
(2) Server Hello (alg. selection,…)
(3) Server Certificate
SSL Handshake

1. Client Hello (algorithms, …)
2. Server Hello (alg. selection, …)
3. Server Certificate
SSL Handshake

1. Client Hello (algorithms, ...)
2. Server Hello (alg. selection, ...)
3. Server Certificate
4. ClientKeyRequest
SSL Handshake

1. Client Hello (algorithms, ...)
2. Server Hello (alg. selection, ...)
3. Server Certificate
4. ClientKeyRequest
5. ChangeCipherSuite
SSL Handshake

1. Client Hello (algorithms, …)
2. Server Hello (alg. selection, …)
3. Server Certificate
4. ClientKeyRequest
5. ChangeCipherSuite
6. ChangeCipherSuite

Client → Server

Server → Client
SSL Handshake

(1) Client Hello (algorithms,...) →
(2) Server Hello (alg. selection,...) →
(3) Server Certificate ←
(4) ClientKeyRequest ←
(5) ChangeCipherSuite ←
(6) ChangeCipherSuite ←
(7) Finished ←
SSL Handshake

(1) Client Hello (algorithms,...)
(2) Server Hello (alg. selection,...)
(3) Server Certificate
(4) ClientKeyRequest
(5) ChangeCipherSuite
(6) ChangeCipherSuite
(7) Finished
(8) Finished
Simplified Protocol Detail

**Participants:** Alice/A (client) and Bob/B (server)

**Crypto Elements:** Random R, Certificate C, $k_i^+$ Public Key (of i)

**Crypto Functions:** Hash function $H(x)$, Encryption $E(k, d)$, Decryption $D(k, d)$, Keyed MAC $HMAC(k, d)$

1. Alice → Bob $R_A$

2. Bob → Alice $R_B, C_B$
   - Alice pick pre-master secret $S$
   - Alice calculate master secret $K = H(S, R_A, R_B)$

3. Alice → Bob $E(k_B^+, S), HMAC(K', CLNT' + [#1, #2])$
   - Bob recover pre-master secret $S = D(k_B^-, E(k_B^+, S))$
   - Bob calculate master secret $K = H(S, R_A, R_B)$

4. Bob → Alice $HMAC(K', SRV R' + [#1, #2])$

**Note:** Alice and Bob: IV Keys, Encryption Keys, and Integrity Keys 6 keys, where each key $k_i = g_i(K, R_A, R_B)$, and $g_i$ is key generator function.
SSL Tradeoffs

• Pros
  ‣ Server authentication*
  ‣ GUI clues for users
  ‣ Built into every browser
  ‣ Easy to configure on the server
  ‣ Protocol has been analyzed like crazy

• Cons
  ‣ Users don’t check certificates
  ‣ Too easy to obtain certificates
  ‣ Too many roots in the browsers
  ‣ Some settings are terrible
Dynamic Content: CGI

• Common Gateway Interface (CGI)
  ‣ Generic way to call external applications on the server
  ‣ Passes URL to external program (e.g., form)
  ‣ Result is captured and return to requestor

• Historically
  ‣ “shell” scripts used to generate content
    • Very, very dangerous

• **NOTE**: server extensions are no better (e.g., servlets)
DC: Embedded Scripting

- Program placed directly in content, run on server upon request and output returned in content
  - MS active server pages (ASP)
  - PHP
  - mod_perl
  - server-side JavaScript
  - python, ...

- Nice at generating output
  - Dangerous if tied to user input
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

• A kind of plug-in …
  ‣ (1997) David.exe
  ‣ “Free pornography …”

• Moral: beware of plugins
JavaScript

• Scripting Language used to improve the quality/experience
  ‣ Create dialogs, forms, graphs, …
  ‣ Built upon API functions (lots of different flavors)
  ‣ No ability to read local files, open connections …

• Security: No ability to read local files, open connections, but …
  ‣ DOS – the “infinite popup” script
    • Often could not “break out” with restarting computer
  ‣ Spoofing – easy to create “password” dialogs
Drive by downloads

- Using a deceptive means to get someone to install something on their own (spyware/adware)

- Once you have one, then it starts downloading lots of others, their friends, …

- A personal favorite: extortion-ware -- pay us 40$ for our popup blocker, etc ….
  - The real gambit is that they demand 40$ for the uninstall option

- Answer: go get adaware and install it (its free)!
Spyware

• Definition: hidden software that uses local host to transmit user secrets
  ‣ e.g., browsing habits, forms data
• Typically found in “free” software
  ‣ Gnutella, game tools, demo software, MP3 tools …
  ‣ Implemented using spyware “engines” - **gator**
• Imbeds in local host to
  ‣ Adds shared libraries (.dlls), adds to startup as TSR programs
  ‣ Often difficult or impossible to remove
    • You are never really sure it is gone (advice: reinstall)
• Gets installed by user action or via some of IEs ability to “help” the user via tools such as **Active-X**
Malicious content injection
Malicious content injection

- Currently, two central infection vectors
Malicious content injection

• Currently, two central infection vectors
  1. Website compromise (and insert IFRAMEs)
Malicious content injection

- Currently, two central infection vectors
  1. Website compromise (and insert IFRAMEs)
  2. Advertising: the abuse of Ad syndication (malverts)
Malicious IFrame(s)

• An IFRAME is a HTML tag that creates an embedded frame in the content of another page.
  ‣ This is the attack vector de jour for adversaries attempting to delivery content that exploits browser vulnerabilities.
  ‣ E.g., deliver crafted .jpg or malicious scripting

• The attack occurs when the adversary breaks into a webserver and places a IFRAME in legitimate content
  ‣ e.g., by sniffing passwords, recursively adding IFRAMEs

<iframe src=http://[REMOVED].info/counter style=display:none></iframe>
Active X

• ActiveX is a MS windows technology
  ‣ Really, just a way to run arbitrary code
  ‣ Called controls (.OCX), just programs
  ‣ Conforms to MS APIs to interact with web

• Extends user experience in lots of nice ways
  ‣ Microsoft upgrade service
  ‣ BIOS Upgrades
  ‣ Lookup services

• **Massive** security hole ....
Is there a concern?

• Initially, MS thought that users would have no problem with ActiveX controls
  ‣ Hey, you run programs you buy, right?
  ‣ With traditional applications
    • You (generally) know who the software comes from
    • You (generally) have some recourse
  ‣ On the Internet …
    • Neither of the above may be true
    • User not actually be involved/aware in execution
Authenticode

• **Problem**: I need to run an application code on my machine, but I worry about security

• **Solution**: Make sure code only comes from people that you trust.

• Authenticode
  ‣ Sign download content
  ‣ Check that signer is “trusted”
  ‣ Used for all Win* content
  ‣ Problem: Jan 2001
    • Verisign issued two bad MS
**Problem**: I need to run an application code on my machine, but I worry about security

**Solution**: Make sure code only comes from people that you trust.

**Authenticode**

- Sign download content
- Check that signer is “trusted”
- Used for all Win* content
- Problem: Jan 2001
  - Verisign issued two bad MS
ActiveX Cautionary Tales

- Exploder (Win95)
  - 1996, Fred McLain
  - Acquired Verisign cert
  - Signed Exploder
    - 10 second countdown
    - … shutdown
- MS/Verisign upset

- Microsoft Access
  - 2000, Guninski
  - ActiveX related control
  - Allowed a website to load and execute a spreadsheet
    - ..Which can contain any command …
  - … which means …
  - A website can run any command on the user machine.
Java

• Platform and language for writing applets
  ‣ Sun Microsystems platform for set-top boxes
  ‣ Applets embedded in web pages (or native)
  ‣ Language loosely resembling C++
  ‣ Runs in a Java Virtual Machine (JVM)
    • Every platform has JVM
    • Platform runs arbitrary code (bytecode)
    • Hence: one application runs on a bunch of platforms
    • Great way to take advantage of the web
    • Slow for data/processing intensive applications
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 ...
The new web-page

- Rendered elements from many sources containing scripts, images, and stylized by cascading style sheets (CSS)

- A browser may be compromised by any of these
Web-server APIs

- Web-servers often provide application extension APIs to which developers can build ...
  - ISSAPI
  - Apache API
- Act as kinds of “kernel modules” for web-server
  - Web-server processes received inputs (URL, fields, etc.)
  - Passes result to custom code (typically, C code)
Application Frameworks

• Application frameworks are software stacks that implement web application
  ‣ Programmer adds domain-specific programming
  ‣ Handle request handling and rendering
  ‣ Quickly implement web apps without dealing the the nasty details of HTTP/HTML

• For example, the Zend framework implements a web application by processing incoming URLs
  ‣ E.g., http://base/module/function
  ‣ Zend accepts returned framework objects and renders them via internal API
  ‣ Modify documents on the fly using AJAX scripts such as JavaScript
AJAX

- **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)
Attacks on web systems

- Web systems have replaced custom organization, enterprise and customer applications..
- ... this move is has led to many new attacks ...
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…
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)
      
        $INPUT = “Alice\;mail($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
    - e.g., change temporary filename input to ~/.profile

<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 ...

```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

• Use the SQL/perl prevent libraries

• Before

```php
$sql = "select * from some_table where some_col = $input";
$sth = $dbh->prepare( $sql );
$sth->execute;
```

• After

```php
$sql = "select * from some_table where some_col = ?";
$sth = $dbh->prepare( $sql );
$sth->execute( $input );
```

• 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.
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
  - Example, 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.
    
    http://www.mybank.com/loggedin?sessionid=11
  
  - ...note that if the user was logged in, then the attacker has full control over that account.
  - *Countermeasure*: randomized, confidential session IDs that are tied to individual host address (see cookies)
Preventing Web System Attacks

• Largely just applications
  ‣ In as much as applications 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
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/