WEB SECURITY I

COURSE REVIEW

- Threat Model
  - Identify and assess potential threats, their motivations, and their capabilities
- Security Principles & policies
  - Recognize and reason about principles, policies, and enforcement in a given scenario
  - Identify a Trusting Computing Base and its requirements
- Software Security
  - Preconditions and postconditions for memory safety
  - Software vulnerabilities
  - Techniques for building secure systems
  - Buffer overflow attacks and defenses

WHAT IS “THE WEB” ???

World Wide Web

HTTP

Hypertext Transfer Protocol (HTTP)

Hypertext Transfer Protocol (HTTP)

WEB DEVELOPMENT IS EASY! IT’S ALL TEXT!

HTML

CSS

JavaScript

SQL

HTTP

URL

Web development is easy! It’s all text!...
HTTP OVER TELNET

$ telnet www.ucsc.edu 80
Trying 128.114.109.5...
Connected to wcms-ucsc.aws-wcms.ucsc.edu.
Escape character is '^]'.

GET / HTTP/1.1
host: www.ucsc.edu

HTTP/1.1 301 Moved Permanently
Date: Mon, 22 Jan 2018 22:52:54 GMT
Server: Apache
X-Frame-Options: SAMEORIGIN
Location: https://www.ucsc.edu/
Content-Length: 330
Connection: close
Content-Type: text/html; charset=iso-8859-1

<!DOCTYPE HTML PUBLIC "-//IETF//DTD HTML 2.0//EN">
<html><head>
title=301 Moved Permanently
</head><body>
<h1>Moved Permanently</h1>
<p>The document has moved <a href="https://www.ucsc.edu/">here</a>.</p>
<hr>
<address>Apache Server at www.ucsc.edu (mailto:webtools@ucsc.edu) Port 80</address>

Connection closed by foreign host.
**WEB SECURITY I**

**HYPERTEXT MARKUP LANGUAGE (HTML)**
- A language for creating structured, interconnected documents
- Easy to embed media objects, create forms

```html
<html>
<body>
  <div>
    <a href="http://google.com">Go to Google!</a>
  </div>
</body>
</html>
```

**CASCADING STYLE SHEETS (CSS)**
- Language that describes how to present a document
- Separates the content of a document from its formatting

```css
p.serif { font-family: "Times New Roman", Times, serif; }
.p.sansserif { font-family: Arial, Helvetica, sans-serif; }
```

**JAVASCRIPT**
- A scripting language for manipulating web pages
- High-level, dynamically typed, interpreted language
- A language for creating web applications
- Today's JS engines are very efficient!

- Powerful and flexible – possible to write sophisticated applications that run entirely in the browser

```javascript
function myFunction()
{
  document.getElementById("motd").innerHTML = "Would you like to play a nice game of chess?";
}
```

**PAGE RENDERING**
- Platform-independent model for representing and interacting with HTML components in a document

**WHAT CAN JAVASCRIPT DO TO A WEB PAGE?**
- Almost anything: basically full access to modify the DOM
- If an attacker gets your browser to load their script into an otherwise safe page, bad things can happen...
- Javascript can:
  - Change HTML content, images, style
  - Hide (or unhide) elements
  - Change the cursor
  - Read or modify cookies
WEB SECURITY I

FRAMES
- Embed a page within a page
- Good for:
  - Modularity
  - Client-side aggregation, servers are unaware
  - Delegation
  - Frame can only draw within its own rectangle
  - Isolation
  - Outer page cannot change contents of inner page (only sizing and placement)
  - Inner page cannot change contents of outer page

WEB SECURITY I

THINKING ABOUT WEB SECURITY

Confidentiality
- Websites should not be able to learn confidential information from our computers or other web sites
- Websites should not be able to spy on us or our online activities

Integrity
- Web sites should not be able to tamper with integrity of our computers or our information on other web sites

Availability
- Third parties should not be able to keep us from accessing our web resources

WEB SECURITY GOALS

WEB SECURITY RISK #1
- A malicious site might trash the files or programs on our computer
- Example:
  - Browsing to sketchyWebsiteFromALinkIShouldntHaveClickedOn.com should not infect our computers with malware, read or write files on our computer, etc.
- Defenses:
  - Sandbox Javascript
  - Privilege separation
  - Or just avoid bugs in browser code...

WEB SECURITY RISK #2
- A malicious site might be able to spy on, tamper with, or corrupt our interactions with other websites
- Example:
  - Browsing to evil.com should not let evil.com spy on our Gmail or buy limited-edition Pez dispensers with our Amazon account.
- Defenses:
  - The same origin policy
  - An ad-hoc policy enforced in the browser
  - Details to come :)

WEB SECURITY RISK #3
- Data stored on a web server might be viewed, deleted, or modified by unauthorized third parties
- Example:
  - Our safebank.com account details can be viewed by a malicious third party if they compromise a query URL when connected anonymously.
  - Our safebank.com account details can be viewed by an otherwise authorized third party if they accidentally enter special characters into a form
- Defenses:
  - Server-side security
WEB SECURITY I
SAME ORIGIN POLICY
- Each site in the browser is isolated from all others.

SAME ORIGIN POLICY
- Multiple pages from the same site are NOT isolated.

WEB SECURITY I
WHAT'S AN ORIGIN?
- Granularity of protection for same origin policy – principal
- Origin = protocol + hostname + port

- http://safebank.com:81/accounts

- If strings match, it is considered the same origin.
- Otherwise they have different origins.
- Even if they are logically the same: e.g., www.safebank.com and safebank.com.

WEB SECURITY I
THE SAME ORIGIN POLICY AND ITS EXCEPTIONS
- Origin of a page is derived from the URL it was loaded from.


- Javascript runs with the origin of the page that loaded it.


- Javascript runs with the origin of the page that loaded it.

- http://upload.wikimedia.org

- Images loaded via <src> don't count!

- http://www.google-analytics.com

- Neither does Javascript!
WEB SECURITY I

THE SAME ORIGIN POLICY: ASSESSMENT

<table>
<thead>
<tr>
<th>Originating document</th>
<th>Accessed document</th>
</tr>
</thead>
</table>

WEB SECURITY I

THE SAME ORIGIN POLICY: WEAKNESSES

▸ Does not prevent tracking users via embedded <script> or <src> objects
▸ Pages can pass sensitive information to embedded <iframe> via URL
▸ Ambiguous specification, doesn’t cover all corner cases (e.g., file:, data:)
▸ Too restrictive – can’t communicate among subdomains etc, can’t create legitimate mash-ups
▸ Approaches for weakening SOP: CORS, Websockets, JSONP

WEB SECURITY I

SERVER SIDE THREATS

COMMAND INJECTION

▸ Suppose a user fills in web form to query a local phone book
▸ Javascript generates a regular expression from the form data
▸ Server is accessed with a URL like:
   http://phonebook.com/search?regex=<pattern>
▸ Example:
   http://phonebook.com/search?regex=Alice%20*Smith*
▸ Searches for all “Alice” with family names including “Smith”...
   Returns “Alice Smithson” “Alice Poncenby-Smith”, etc.

FUNCTION

```c
void find(char *regex){
    char cmd[512];
    sprintf(cmd, sizeof cmd, 
        "grep %s phonebook.txt", regex);
    system(cmd);
}
```

COMMAND INJECTION DEFENSES

▸ Poor: Input sanitization
  ▸ Look for “bad things” in the input and neutralize them
  ▸ Tricky to get right and brittle!
  ▸ Goes against “failsafe defaults” principle:
    ▸ Input is considered okay unless a bad thing is found
▸ Better: Constrain the API
  ▸ Keep it simple + defensive programming
  ▸ http://phonebook.com/search?first=Alice&last=Smith
  ▸ void find(char *first, char *last);
From the looks of it, however, one ought to suspect an SQL injection, in which the Web site, Markovich also questions, not noticed the hack for six months, 45
Suppose the web server runs the following PHP:

```
$user = $_POST['user'];
$sql = "SELECT AcctNum FROM Customer
WHERE Balance > 1000000 AND
Username='\$user';
$result = $db->executeQuery($sql);
```

- Query returns the user's account number if they are a "valued" customer
- Web server will send value of $sql variable to database server to get account numbers from a database

```html
http://safebank.com/account.php?user=alice
SELECT AcctNum FROM Customer
WHERE Balance > 1000000 AND
Username='alice'
```

Operator precedence in SQL turns this into:

```sql
SELECT AcctNum FROM Customer
WHERE Balance > 1000000 AND (Username='alice' OR 1=1')
```

Since 1=1 is always true, this is effectively:

```sql
SELECT AcctNum FROM Customer
WHERE Balance > 1000000
```

### SQL Injection Target
- SQL Injection Attacks
  - Syntax error!
  - SQL comment

### Next Time
- Web Security II
  - More on SQL Injection
  - Cookies
  - Cross Site Request Forgery
  - Cross Site Scripting
  - Lab 2 Preview