Obvious Threats

- Telnet
- FTP
- HTTP
- MySQL
- ...
suPHP

http://www.suphp.org/
Session Hijacking (scenarios)

- Physical Access
- Packet Sniffing
- XSS
Session Hijacking (defenses)

- Hard-to-guess session keys?
- Rekey session?
- Check IP address?
- Encryption?
SSL

Image from godaddy.com.
Public-Key Cryptography

Diagram showing the process of public-key cryptography:
- **Public Key**
  - Used for encryption
  - Can be shared publicly
- **Private Key**
  - Used for decryption
  - Must be kept secret

**Plaintext** → **Encryption** → **Ciphertext** → **Decryption** → **Plaintext**

Image from http://ww.nuitari.de/crypto.html.
Diffie-Hellman (DLP)

Alice
choose random A

Bob
choose random B

agree on g,p

\[ T_A = g^A \mod p \]

\[ T_B = g^B \mod p \]

compute \( T_B^A \)

compute \( T_A^B \)

agree on \( g^{AB} \mod p \)
$result = mysql_query(sprintf(" SELECT uid FROM users
WHERE username='%s' AND password='%s' ",
$_POST["username"], $_POST["password"]));
SQL Injection Attacks

SELECT uid FROM users
WHERE username='jharvard'
AND password='12345' OR '1' = '1'
SQL Injection Attacks

```php
$result = mysql_query(sprintf(" SELECT uid FROM users
WHERE username='%s' AND password='%s' ",
mysql_real_escape_string($_POST['username']),
mysql_real_escape_string($_POST['password'])));
```

![Login Form with SQL Injection Vulnerability](image)
SQL Injection Attacks

```
SELECT uid FROM users
WHERE username='jharvard'
AND password='12345\' OR \'1\' = \'1'
```
The Same-Origin Policy

“The same origin policy prevents document or script loaded from one origin from getting or setting properties of a document from a different origin. ... Mozilla considers two pages to have the same origin if the protocol, port (if given), and host are the same for both pages. To illustrate, this table gives examples of origin comparisons to the URL http://store.company.com/dir/page.html.”

<table>
<thead>
<tr>
<th>URL</th>
<th>Outcome</th>
<th>Reason</th>
</tr>
</thead>
<tbody>
<tr>
<td><a href="http://store.company.com/dir2/other.html">http://store.company.com/dir2/other.html</a></td>
<td>Success</td>
<td></td>
</tr>
<tr>
<td><a href="http://store.company.com/dir/inner/another.html">http://store.company.com/dir/inner/another.html</a></td>
<td>Success</td>
<td></td>
</tr>
<tr>
<td><a href="https://store.company.com/secure.html">https://store.company.com/secure.html</a></td>
<td>Failure</td>
<td>Different protocol</td>
</tr>
<tr>
<td><a href="http://store.company.com:81/dir/etc.html">http://store.company.com:81/dir/etc.html</a></td>
<td>Failure</td>
<td>Different port</td>
</tr>
<tr>
<td><a href="http://news.company.com/dir/other.html">http://news.company.com/dir/other.html</a></td>
<td>Failure</td>
<td>Different host</td>
</tr>
</tbody>
</table>

The Same-Origin Policy Affects…

- Windows
- Frames
- Embedded Objects
- Cookies
- XmlHttpRequest
Attacks

- Cross-Site Request Forgery (CSRF/XSRF)
- Cross-Site Scripting (XSS)
- ...
CSRF/XSRF (scenario)

1. You log into project1.domain.tld.
2. You then visit a bad guy’s site.
3. Bad guy’s site contains a link to http://project1.domain.tld/buy.php?symbol=INFX.PK
4. You unwittingly buy the penny stock!
CSRF/XSRF (implementations)

- `<img src="http://project1.domain.tld/buy.php?symbol=INFX.PK">`
- `<script src="http://project1.domain.tld/buy.php?symbol=INFX.PK"></script>`
- `<iframe src="http://project1.domain.tld/buy.php?symbol=INFX.PK">`
- `<script>
    var img = new Image();
</script>`
- `...`
CSRF/XSRF (defenses)

- Use POST for sensitive actions?
- Use HTTP_REFERER?
- Append session tokens to URLs?
- Expire sessions quickly?
- CAPTCHAs?
- Prompt user to re-login?
XSS (scenario)

1. You click a link like
   or, really,

2. vulnerable.com makes the mistake of writing value of foo to its body

3. badguy.com gets your cookies!
XSS (defenses)

- Don’t click links?
- Don’t trust user input?
- Encode all user input?
Computer Science S-75
Building Dynamic Websites

Harvard Summer School
https://www.cs75.net/

Lecture 8: Security

David J. Malan
malan@harvard.edu