CSE 484 / CSE M 584

Computer Security: Web Security

TA: Franzi Roesner
franzi@cs.washington.edu

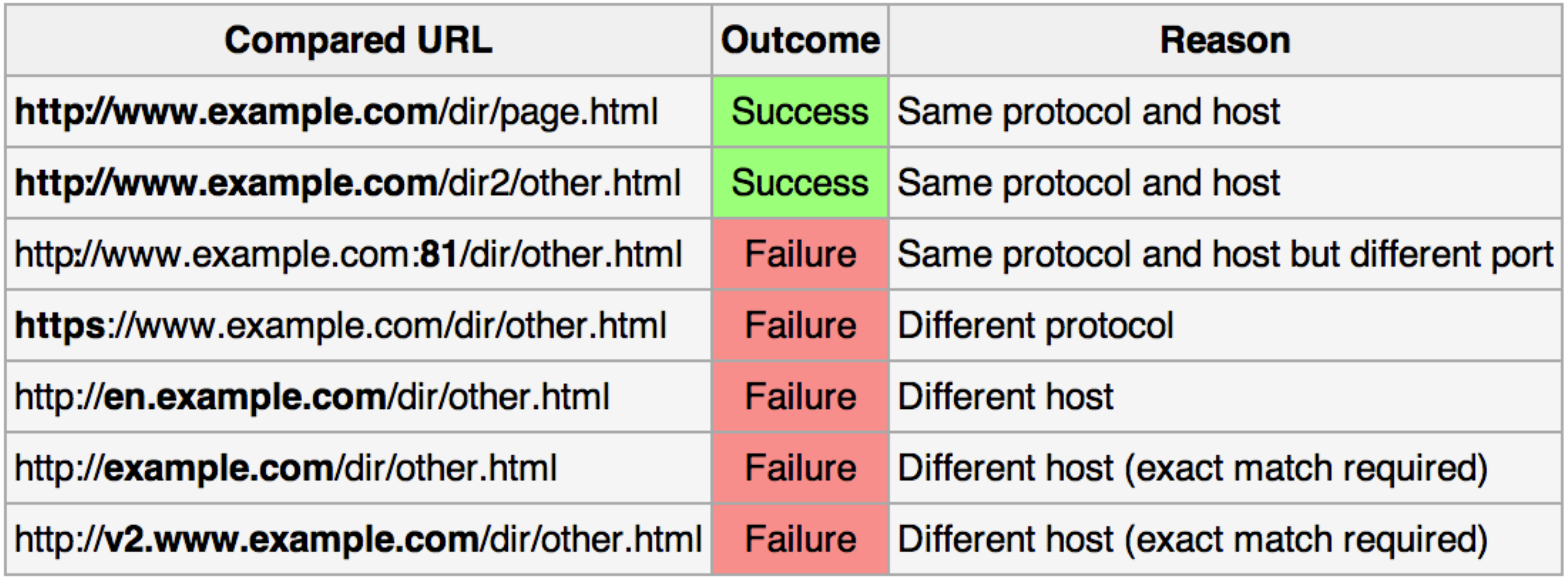
Logistics

- Homework #2 (crypto) due 2/22 5pm.
- Lab #2 (web security) due 2/27 5pm.

- Lab #1 looks AWESOME! ☺

Same-Origin Policy

Website origin = (scheme, domain, port)

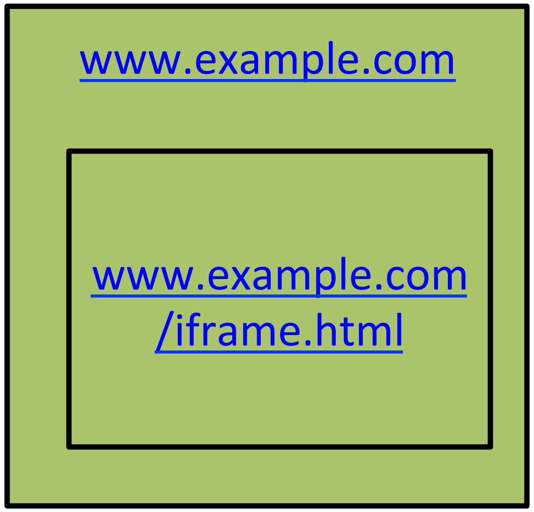


Compared URL	Outcome	Reason
http://www.example.com/dir/page.html	Success	Same protocol and host
http://www.example.com/dir2/other.html	Success	Same protocol and host
http://www.example.com:**81**/dir/other.html	Failure	Same protocol and host but different port
https://www.example.com/dir/other.html	Failure	Different protocol
http://**en.example.com**/dir/other.html	Failure	Different host
http://**example.com**/dir/other.html	Failure	Different host (exact match required)
http://**v2.www.example.com**/dir/other.html	Failure	Different host (exact match required)

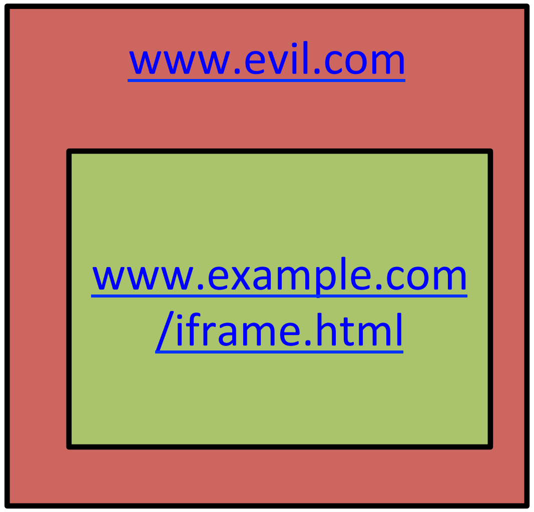
[Example thanks to Wikipedia.]

Same-Origin Policy (DOM)

- Only code from same origin can access HTML elements on another site (or in an iframe).



www.example.com (the parent) **can** access HTML elements in the iframe (and vice versa).



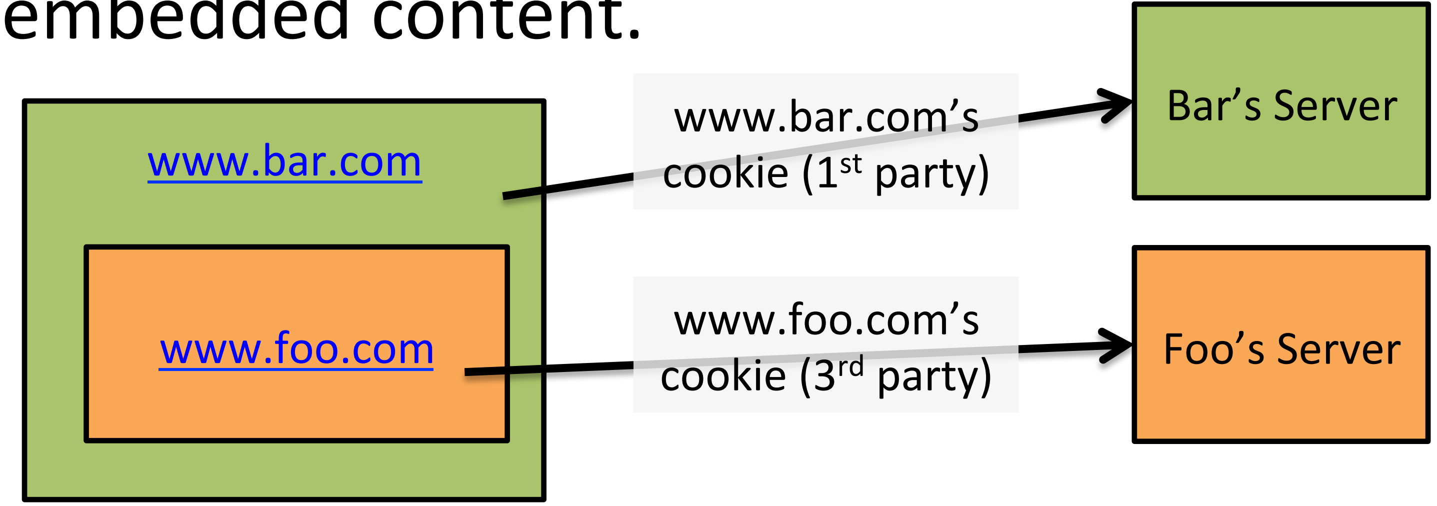
www.evil.com (the parent) **cannot** access HTML elements in the iframe (and vice versa).

Same-Origin Policy (Cookies)

- **For cookies:** Only code from same origin can read/write cookies associated with an origin.
  - Can be set via Javascript (`document.cookie=…`) or via `Set-Cookie` header in HTTP response.
  - Can narrow to subdomain/path (e.g., http://example.com can set cookie scoped to http://account.example.com/login.)
  - Secure cookie: send only via HTTPS.
  - HttpOnly cookie: can’t access using JavaScript.

Same-Origin Policy (Cookies)

- Browsers automatically include cookies with HTTP requests.
- **First-party cookie:** belongs to top-level domain.
- **Third-party cookie:** belongs to domain of embedded content.


www.bar.com
www.foo.com
www.bar.com's cookie (1st party)
www.foo.com's cookie (3rd party)
Bar's Server
Foo's Server

Same-Origin Policy (Scripts)

- When a website **includes a script**, that script runs in the context of the embedding website.

www.example.com

```
<head>
<script src="http://
otherdomain.com/
library.js"></script>
</head>
```

The code from http://otherdomain.com **can** access HTML elements and cookies on www.example.com.

- If code in the script sets a cookie, under what origin will it be set?

XSS: Cross-Site Scripting

- **Idea:** Place user-provided data in the page.
  - Makes page more interactive and personal.
- **Threat:** Improperly used data can be interpreted as code.
- Demo…
- **Solutions?**
  - Sanitize/validate input. (e.g., `htmlspecialchars()`)
  - Browser detection/prevention.

XSSI: Cross-Site Script Inclusion

- **Idea:** Include scripts (e.g., libraries) to run in context of current domain.

Example:

```
<head> <script src="//ajax.googleapis.com/ajax/libs/
jquery/1.9.1/jquery.min.js"></script> </head>
```

- **Threat:** Attacker provides malicious library, can execute code in your domain's context.
- **Solution:** Make sure included code comes from trusted site.

XSRF: Cross-Site Request Forgery

- **Idea:** Protect sensitive actions (e.g., Amazon purchase) by authenticating users w/ cookies.
- **Threat:** Attacker tricks user's browser into visiting sensitive URL. For example:

```
http://amazon.com/purchase.php?
oneclick=true&item=523586
```

- Why does this work?
  – Browsers automatically attach cookies to requests.

XSRF Defense

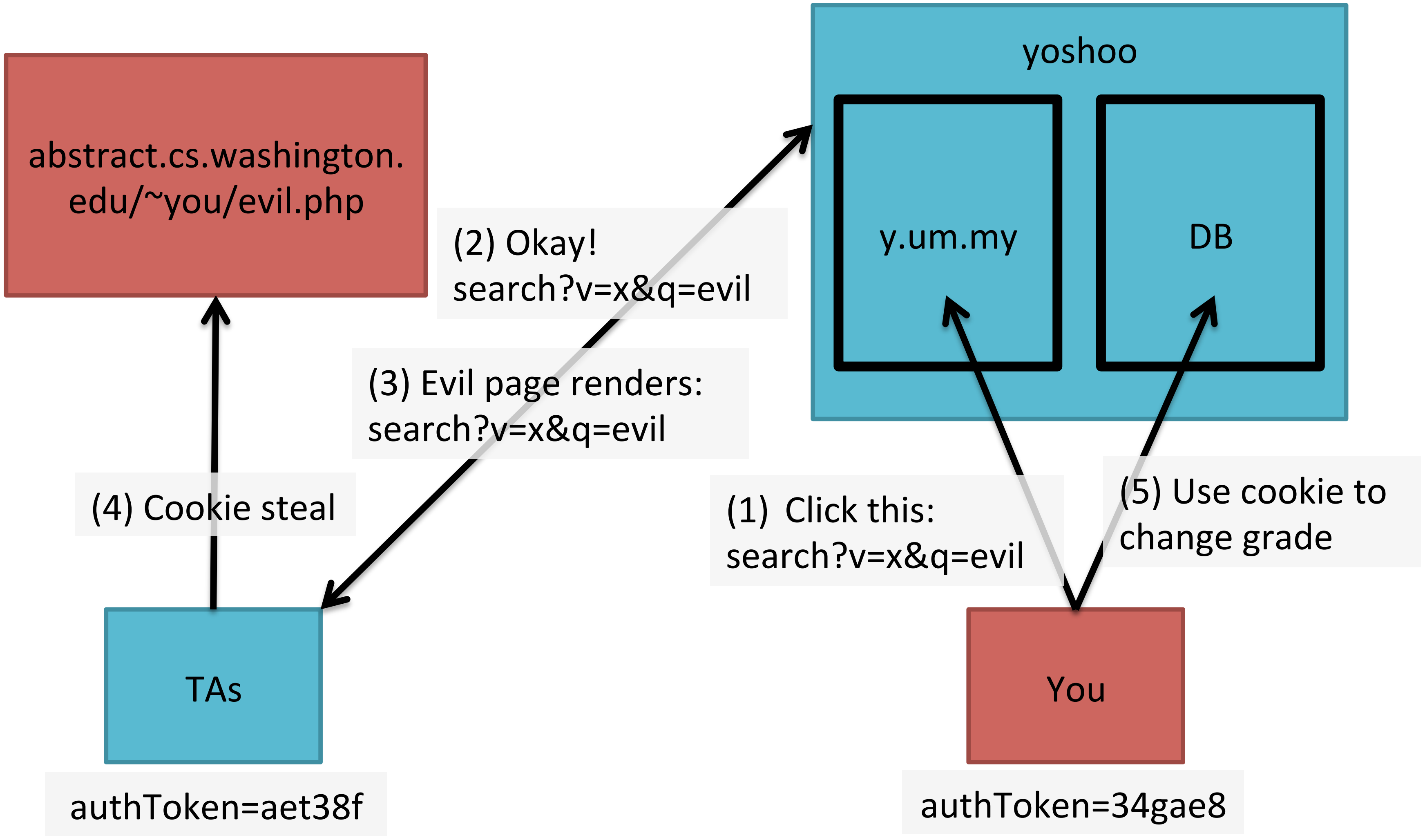
Include XSRF token (e.g., based on user session):

```
<form action=”purchase.php" method="post">
  <input type="hidden" name="csrf"
  value="<?php echo $key; ?>" />
  <input type="submit" value=”One-Click
  Purchase">
</form>
```

Why does this work?

Attacker can’t read token due to same-origin policy.

Lab #2 Explained

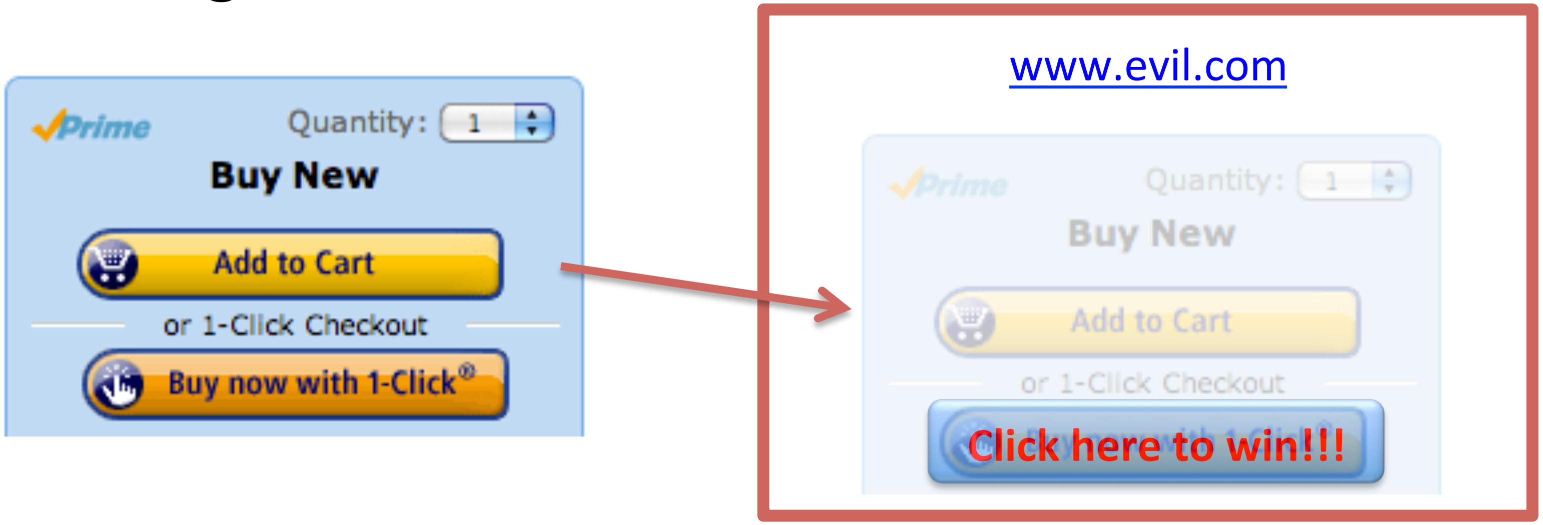

abstract.cs.washington.edu/~you/evil.php
yoshoo
y.um.my
DB
(2) Okay! search?v=x&q=evil
(3) Evil page renders: search?v=x&q=evil
(4) Cookie steal
(1) Click this: search?v=x&q=evil
(5) Use cookie to change grade
TAs
You
authToken=aet38f
authToken=34gae8

Lab #2 Guidelines

- Email me your group members, group name, and desired password.
- Your script must run on abstract.cs.washington.edu.
- Some versions of some browsers provide XSS protections, so testing might fail. (Try Firefox.)
- Make sure exploits work locally before submitting links to y.um.my.
- See lab FAQ for links to add-ons to modify cookies.
- Extra credit is hard/unexpected, based on real bug from previous TAs (don't waste your time).

Clickjacking

- Trick users into interacting with sensitive user interfaces in another domain.
  – Using invisible iframes:



  – Exploit predictable user timing: http://lcamtuf.coredump.cx/ffgeo2/