Chapter 09 – Web Security

University of Illinois

ECE 422/CS 461

Some content adapted from materials by Raluca Ada

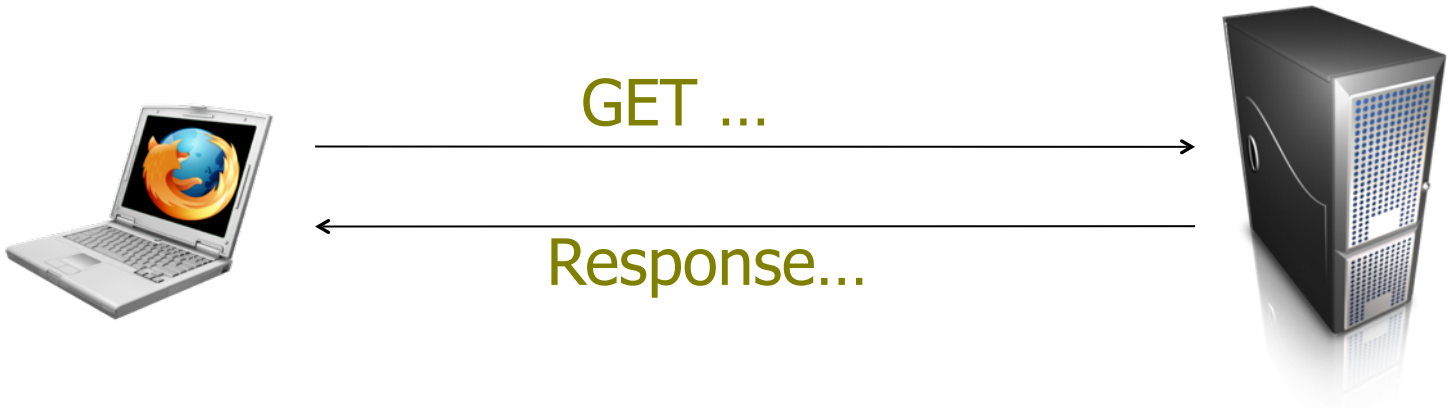
Goals

- By the end of this chapter you should:
  - Understand the threat model underlying the Web
  - Define the same origin policy
  - Articulate the two main attacks unique to the web: CSRF and XSS
  - Illustrate common defenses to CSRF and XSS

WEB BACKGROUND

What is the Web?

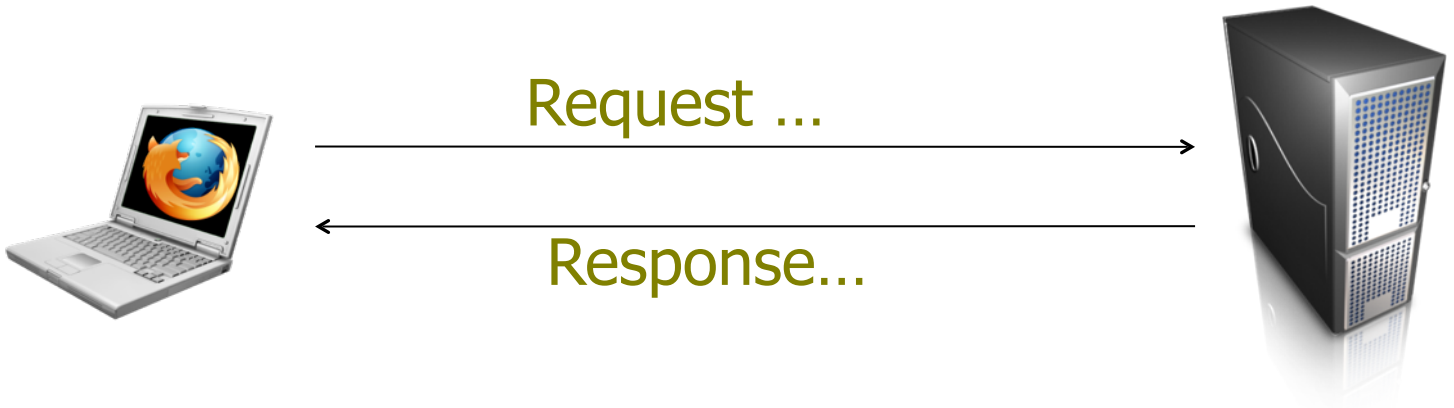
- Application layer on top of TCP/UDP that follows a **client-server mode**



What is the Web?

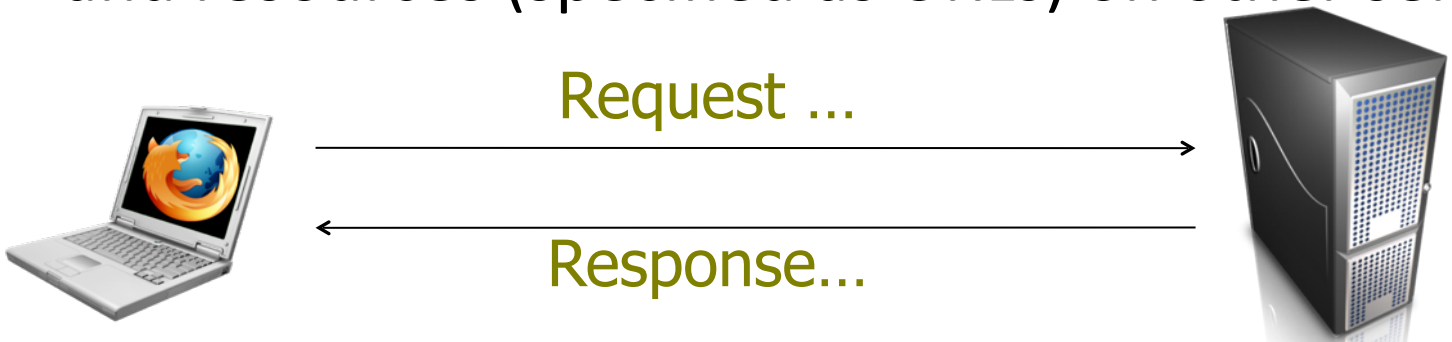
- Application layer on top of TCP/UDP that follows a **client-server model**
  - Web resources are identified by Uniform Resource Locators (URLs)

Request URL:	https://www.google.com/search?q=uiuc+ece
Request Method:	GET
Status Code:	200 OK
Remote Address:	142.250.190.36:443
Referrer Policy:	strict-origin-when-cross-origin



What is the Web?

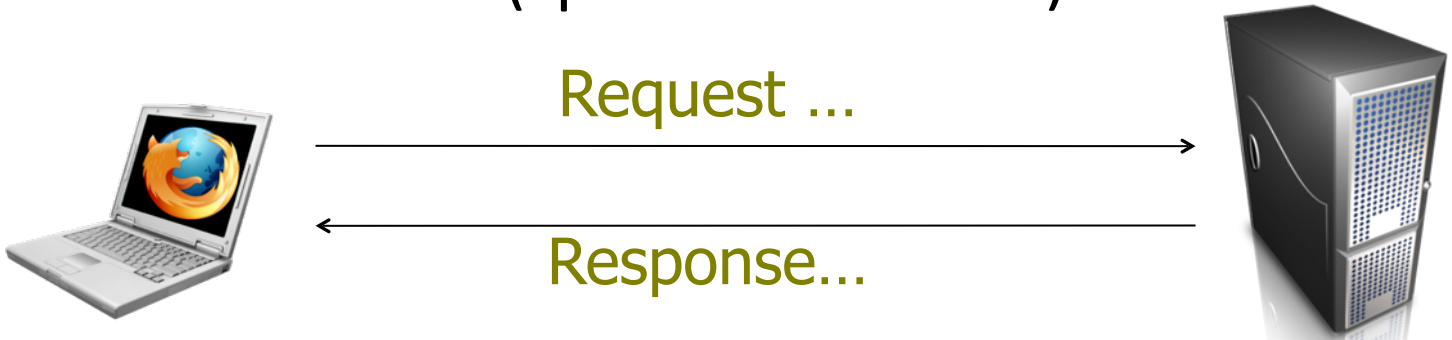
- Application layer on top of TCP/UDP that follows a **client-server model**
  - Web resources are identified by Uniform Resource Locators (URLs) and transferred via the Hypertext Transfer Protocol (HTTP)
  - Web pages formatted using Hypertext Markup Language (HTML) and include **links** to other pages and resources (specified as URLs) on other servers



What is the Web?

```
<html itemscope="" itemtype="http://schema.org/SearchResultsPage" lang="en">
    <head>
        <meta charset="UTF-8">
        <meta content="origin" name="referrer">
        <meta content="Anm+hhtuh7NJguqSnXHEAIqqMaV+GXCks8WYXHJKF7l6AeYMj+wO+fi9OdDqFnJTg9t0492DykVxx4jpvFbxnA8AAABseyJvcmlnaW4iOiJodHRwczovL2dvb2dsZS5j
        <meta content="/images/branding/googleg/1x/googleg_standard_color_128dp.png" itemprop="image">
        <title>uiuc ece - Google Search</title>
        <script nonce="YhFRxdlGO7e1viIlwX1Ccw">
            window._hst = Date.now();
            performance && performance.mark && performance.mark("SearchHeadStart");
        </script>
        <script nonce="YhFRxdlGO7e1viIlwX1Ccw">
            (function() {
                var b = window.addEventListener;
                window.addEventListener = function(a, c, d) {
                    a !== "unload" && b(a, c, d)
                }
                ;
```

- Web pages formatted using Hypertext Markup Language (HTML) and include **links** to other pages and resources (specified as URLs) on other servers



HTML

```
<html itemscope="" itemtype="http://schema.org/SearchResultsPage" lang="en">
    <head>
        <meta charset="UTF-8">
        <meta content="origin" name="referrer">
        <meta content="Anm+hhtuh7NJguqSnXHEAIqqMaV+GXCks8WYXHJKF7l6AeYMj+wO+fi9OdDqFnJTg9t0492DykVxx4jpvFbxnA8AAABseyJvcmlna
        <meta content="/images/branding/googleg/1x/googleg_standard_color_128dp.png" itemprop="image">
        <title>uiuc ece - Google Search</title>
        <script nonce="YhFRxdlGO7e1viIlwX1Ccw">
            window._hst = Date.now();
            performance && performance.mark && performance.mark("SearchHeadStart");
        </script>
        <script nonce="YhFRxdlGO7e1viIlwX1Ccw">
            (function() {
                var b = window.addEventListener;
                window.addEventListener = function(a, c, d) {
                    a !== "unload" && b(a, c, d)
                }
                ;
```

Javascript

From Web Pages to Web Applications

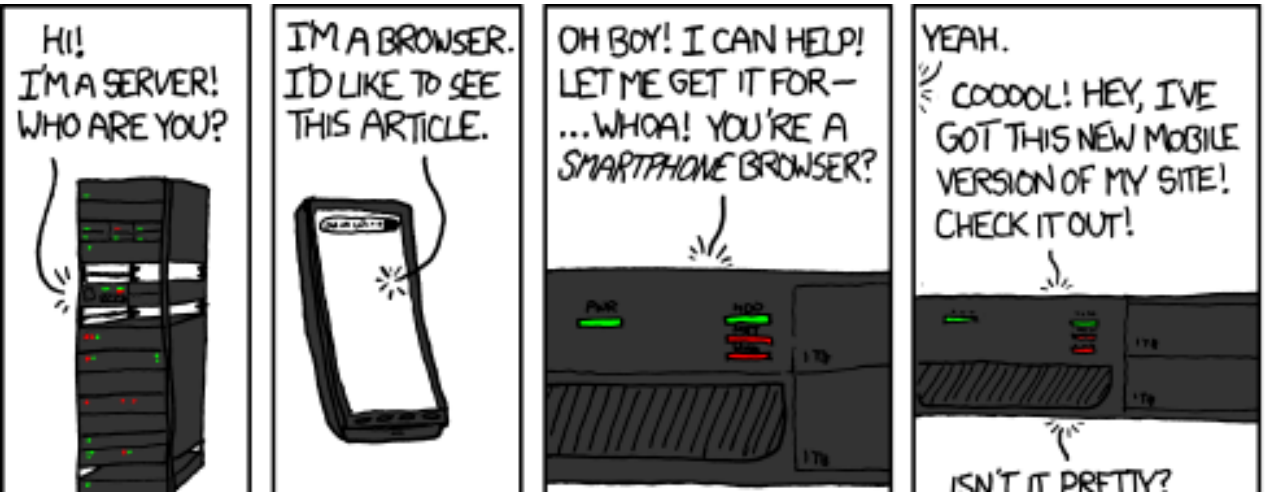
- Initial web pages were static text
  - Developed to meet the demand for information sharing
- New applications had *interactive* functionality
  - Games
  - Message boards
  - Banking
  - …
- Needed to track *state* across HTTP requests
  - HTTP is **stateless**

From Web Pages to Web Applications

- Initial web pages were static text
  - Developed to meet the demand for information sharing
- New applications had *interactive* functionality
  - Games
  - Message boards
  - Banking
  - …
- Needed to track *state* across HTTP requests
  - HTTP is **stateless**

From Web Pages to Web Applications

- Initial web pages were static text
  - Developed to meet the demand for information sharing
- New applications had *interactive* functionality
  - Games
  - Message boards
  - Banking
  - …
- Needed to track *state* across HTTP requests
  - HTTP is **stateless**



From Web Pages to Web Applications

- Initial web pages were static text
  - Developed to meet the demand for information sharing
- New applications had *interactive* functionality
  - Games
  - Message boards
  - Banking
  - …
- Needed to track *state* across HTTP requests
  - HTTP is **stateless**

From Web Pages to Web Applications

- Initial web pages were static text
  - Developed to meet the demand for information sharing
- New applications had *interactive* functionality
  - Games
  - Message boards
  - Banking
  - …

- Needed to track *state* across HTTP requests
  - HTTP is **stateless**

Cookies

- A way for websites to store state on clients

Cookies

- A way for websites to store state on clients
  - Browser maintains all cookies it receives

Cookies

- A way for websites to store state on clients
  - Browser maintains all cookies it receives
  - Browser **automatically** attaches all cookies **in scope** in subsequent requests to the website

Web Sessions

Web Sessions

- A sequence of user interactions with a website

Web Sessions

- A sequence of user interactions with a website
  - High security applications: 15 minutes
  - Medium security applications: 30 minutes
  - Low security applications: 1 hou

Web Sessions

- A sequence of user interactions with a website
  - High security applications: 15 minutes
  - Medium security applications: 30 minutes
  - Low security applications: 1 hou
- Session management
  - Authenticate user once, give user a secret token
  - User (browser) submits the secret token with every subsequent request

Logged in cookies

_hp2_props.300103...	%7B%22Bas...	.illinois.edu	/	202...	60		✓	None
_hp2_ses_props.300...	%7B%22ts%...	.illinois.edu	/	202...	125		✓	None
_legacy_normandy_s...	IWl14cjx49-...	canvas.illinois.edu	/	Ses...	744	✓	✓	
canvas_session	IWl14cjx49-...	canvas.illinois.edu	/	Ses...	734	✓	✓	None
dpUseLegacy	false	canvas.illinois.edu	/	202...	16			
inst-fs-session	eyJpZGVud...	.inst-fs-iad-prod.i...	/	202...	143	✓	✓	None
inst-fs-session.sig	XKF0FRNF4...	.inst-fs-iad-prod.i...	/	202...	46	✓	✓	None
log_session_id	c721d626ba...	canvas.illinois.edu	/	Ses...	46	✓	✓	

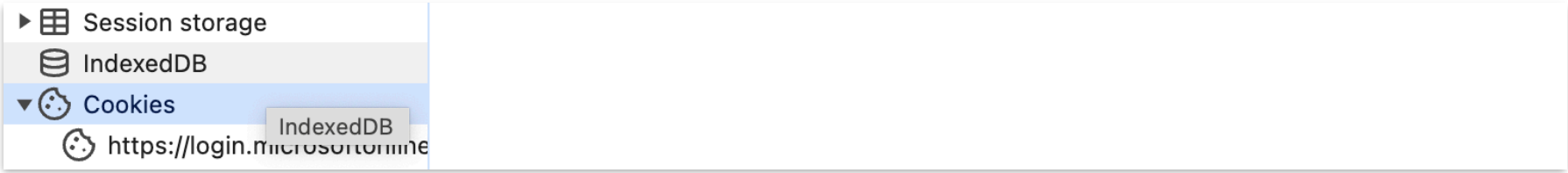
When Request

Headers | Preview | Response | Initiator | Timing | Cookies

Request Cookies ☐ show filtered out request cookies

Name	Value	Domain	Path	Expires / Max-...	Size	Ht...	Se...	SameSite	Parti...	Cros...	Prior...
OptanonAlertBoxClosed	2024-09-24T...	.illinois.edu	/	2024-12-23T...	45			Lax			Medi...
OptanonConsent	isGpcEnabled...	.illinois.edu	/	2024-12-23T...	260			Lax			Medi...
_csrf_token	n65xVqnpUx...	canvas.illinois.edu	/	Session	113		✓				Medi...
_ga	GA1.1.191417...	.illinois.edu	/	2025-10-29T...	29						Medi...
_ga_71JGWHBFGH	GS1.1.172714...	.illinois.edu	/	2025-10-29T...	52						Medi...
_hp2_id.3001039959	%7B%22userI...	.illinois.edu	/	2025-10-23T1...	372		✓	None			Medi...
_hp2_props.3001039959	%7B%22Base...	.illinois.edu	/	2025-10-23T1...	60		✓	None			Medi...
_hp2_ses_props.3001039...	%7B%22ts%...	.illinois.edu	/	2024-09-24T...	125		✓	None			Medi...
_legacy_normandy_session	IWl14cjx49-Li...	canvas.illinois.edu	/	Session	744	✓	✓				Medi...
canvas_session	IWl14cjx49-Li...	canvas.illinois.edu	/	Session	734	✓	✓	None			Medi...
dpUseLegacy	false	canvas.illinois.edu	/	2024-12-31T0...	16						Medi...
log_session_id	c721d626bac...	canvas.illinois.edu	/	Session	46	✓	✓				Medi...

After logging out



Web Sessions

- A sequence of user interactions with a website
  - Authenticate user once, set a cookie with session ID
  - User (browser) submits the session cookie with every request

Web Sessions

- A sequence of user interactions with a website
  - Authenticate user once, set a cookie with session ID
  - User (browser) submits the session cookie with every request
- Need to protect the cookie! If stolen, it gives attacker full access to the user's account

Web Sessions

- A sequence of user interactions with a website
  - Authenticate user once, set a cookie with session ID
  - User (browser) submits the session cookie with every request
- Need to protect the cookie! If stolen, it gives attacker full access to the user's account
- Impersonation attacks can happen even without stealing the cookie (CSRF)

WEB SECURITY

Web Security History

- The web is an example of “bolted-on security”
- Originally, the web was invented to allow scientists to share their research papers
  - Only textual web pages + links to other pages; no threat model to speak of

Web Security History

- The web is an example of “bolted-on security”
- Originally, the web was invented to allow scientists to share their research papers
  - Only textual web pages + links to other pages; no threat model to speak of
- Then, it got more and more complex
  - Images, videos, frames, Javascript, …
- Web security is a challenge!

Web Security Risks

- What are we defending?

Web Security Risks

- What are we defending?
  - Confidentiality, integrity and availability
- From whom?

Web Security Risks

- What are we defending?
  - Confidentiality, integrity and availability
- From whom?
  - Anyone can be malicious

Web Security Risks

- What are we defending? From whom?
- Risk #1: malicious client steals/modifies data on a web server, or takes control of server

Code Red worm

```
GET /default.ida?
NNNNNNNNNNNNNNNNNNNNNNNNNNNNNNNNNNNNNNNNNNNNNNNNNNNNNNNNNNNN
NNNNNNNNNNNNNNNNNNNNNNNNNNNNNNNNNNNNNNNNNNNNNNNNNNNNNNNNNNNN
NNNNNNNNNNNNNNNNNNNNNNNNNNNNNNNNNNNNNNNNNNNNNNNNNNNNNNNNNNNN
NNNNNNNNNNNNNNNNNNNNNNNNNNNNNNNNNNNNNNNNNNNNNNNNNNNNNNNNNNNN
NNNNNNNNNNNN%u9090%u6858%ucbd3%u7801%u9090%u6858%ucbd
3%u7801%u9090%u6858%ucbd3%u7801%u9090%u9090%u8190%u00
c3%u0003%u8b00%u531b%u53ff%u0078%u0000%u00=a HTTP/1.0
```

A really bad sever app

```
<?php

echo system(“ls ” . $_GET[“path”]);
```

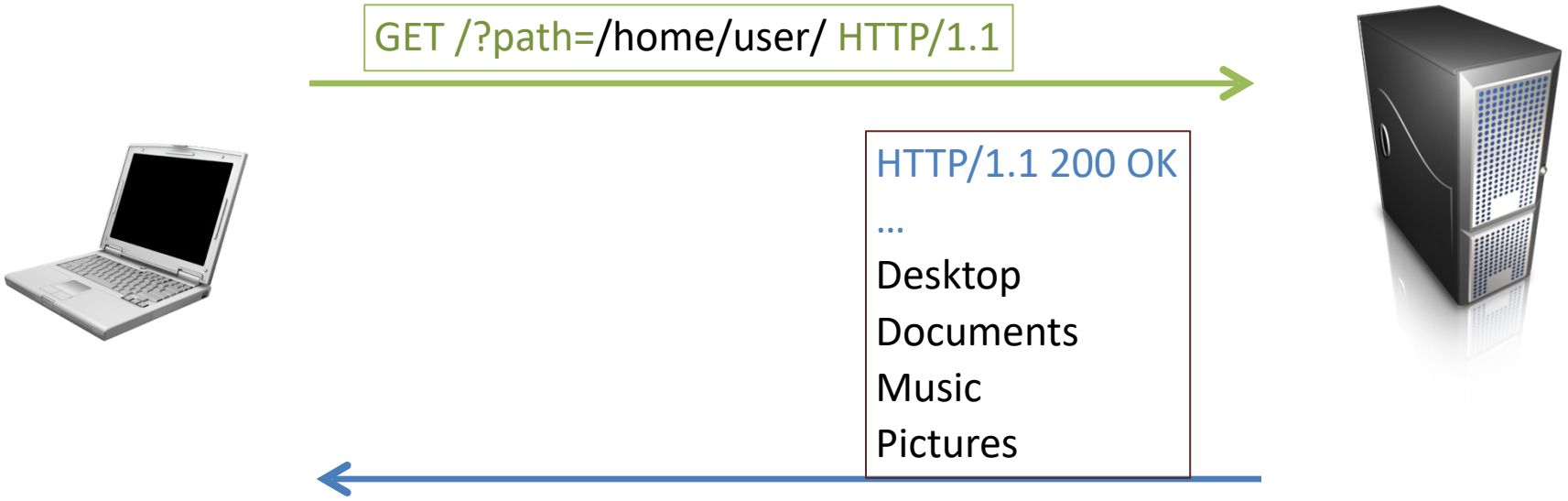
A really bad sever app

```
<?php

echo system(“ls ” . $_GET[“path”]);
```

GET /?path=/home/user/ HTTP/1.1

A really bad sever app

```
<?php

echo system(“ls ” . $_GET[“path”]);
```



A really bad sever app

```
<?php

echo system(“ls ” . $_GET[“path”]);
```

A really bad sever app

```
<?php

echo system(“ls ” . $_GET[“path”]);
```

GET /?path=$(rm –rf /) HTTP/1.1

A really bad sever app

```
<?php

echo system(“ls ” . $_GET[“path”]);
```

GET /?path=$(rm –rf /) HTTP/1.1

```
<?php

echo system(“ls $(rm –rf /)”);
```

A really bad sever app

```
<?php

echo system(“ls ” . $_GET[“path”]);
```

GET /?path=$(rm –rf /) HTTP/1.1

```
<?php

echo system(“ls $(rm –rf /)”);
```

A really bad sever app

```
<?php

echo system(“ls ” . $_GET[“path”]);
```

GET /?path=$(rm –rf /) HTTP/1.1


```
<?php

echo system(“ls $(rm –rf /)”);
```

Aside: Code Injection

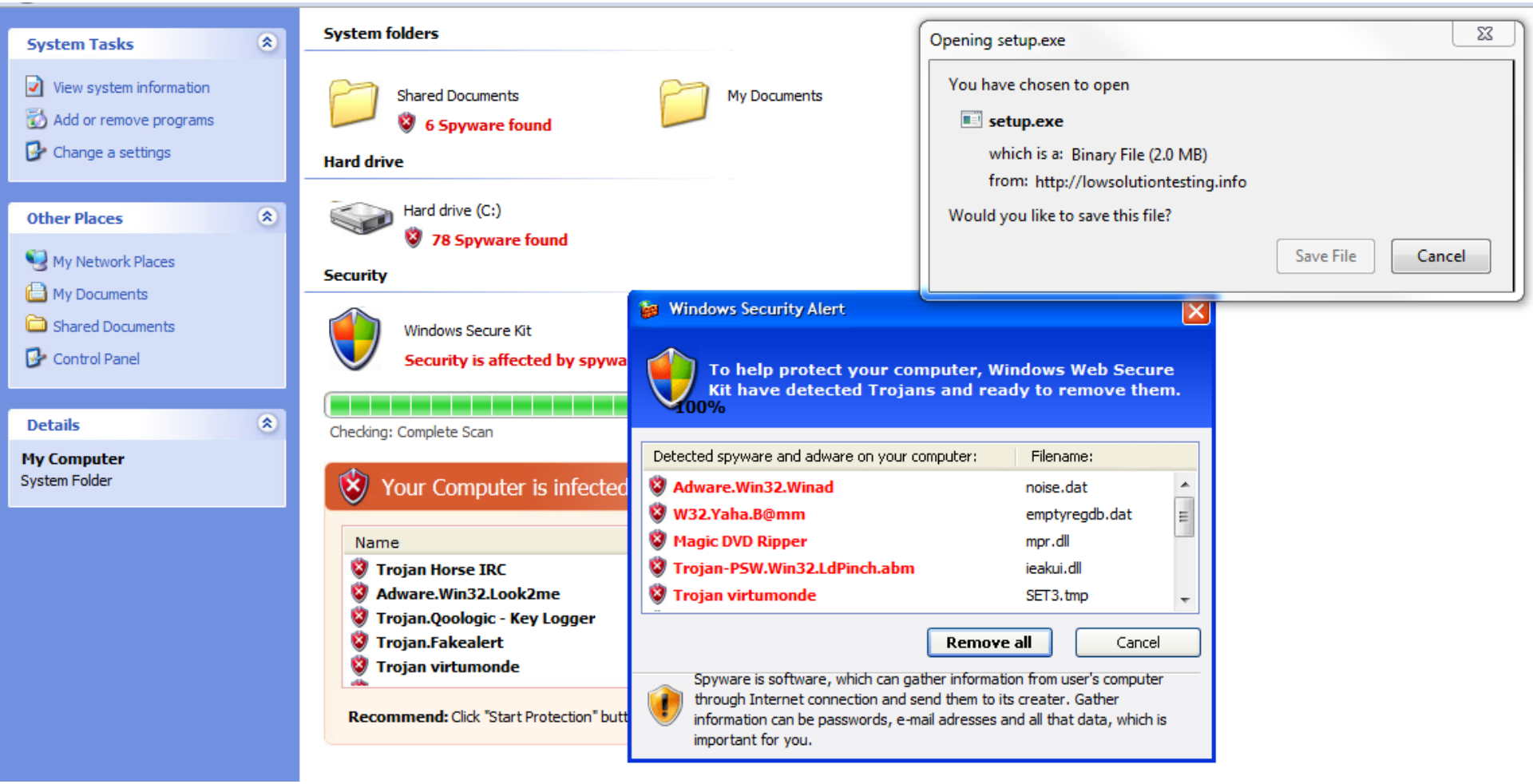
```
<?php

echo system("ls $(rm –rf /)");
```

- Confusing **Data** and **Code**
  - Programmer thought user would supply data,
    but instead got (and unintentionally executed) code
- Common and dangerous class of vulnerabilities
  - Saw it before
    - Control-flow Hijacking (Buffer overflows)
  - Will see it today
    - Cross-Site Scripting (XSS)
  - Will see it next time
    - SQL Injection

Web Security Risks

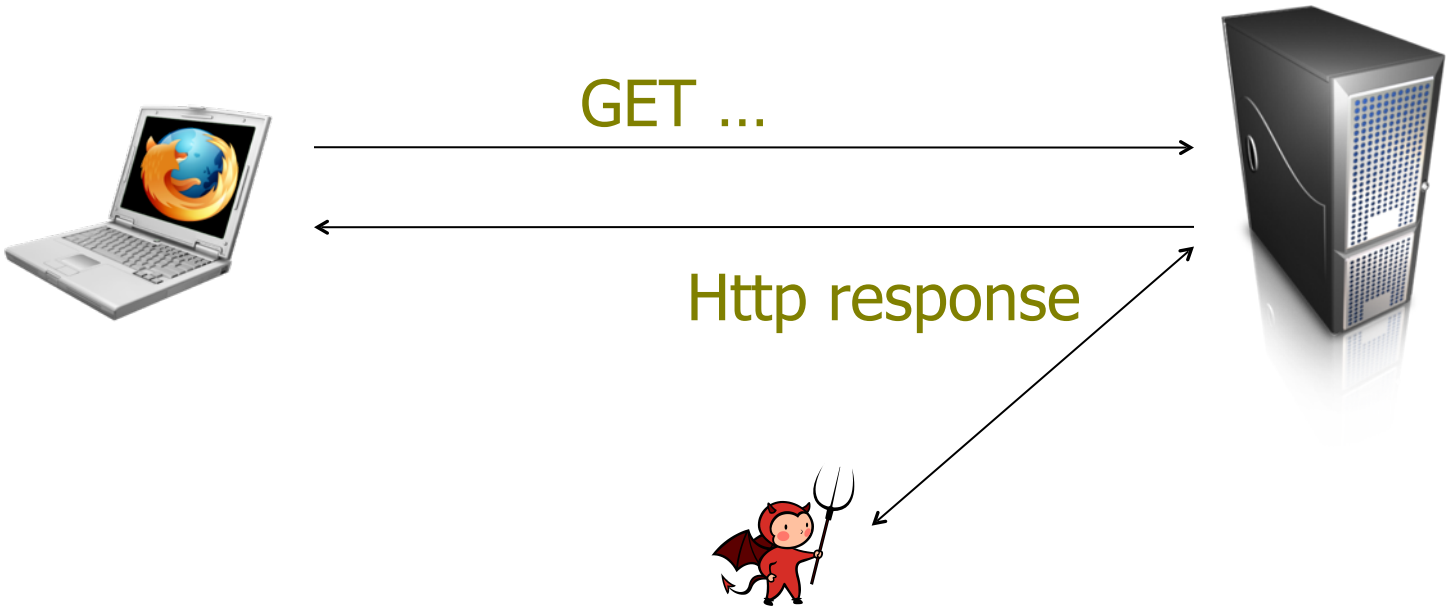
- What are we defending? From whom?
- Risk #2: malicious website steals/trashes files on clients, or infects clients with malware

Example: FakeAV



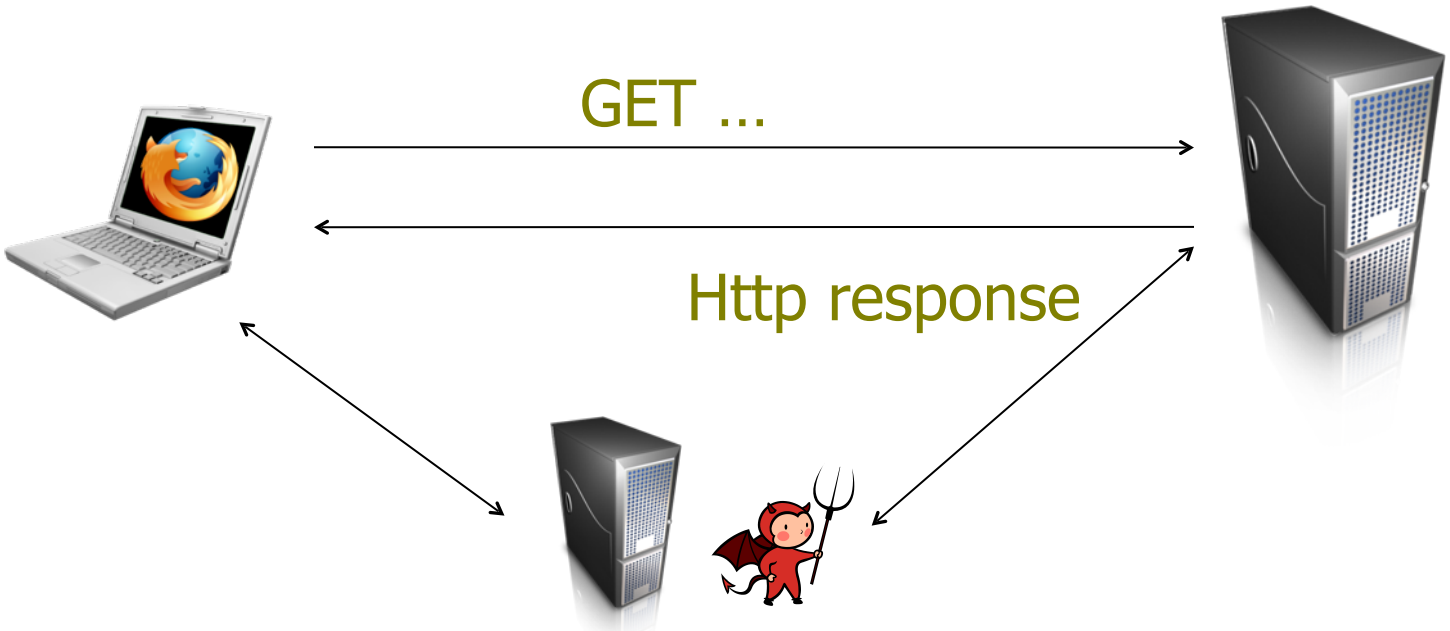
Web Security Risks

- What are we defending? From whom?
- Risk #3: an attacker spies on or tampers with a client's interaction with a website



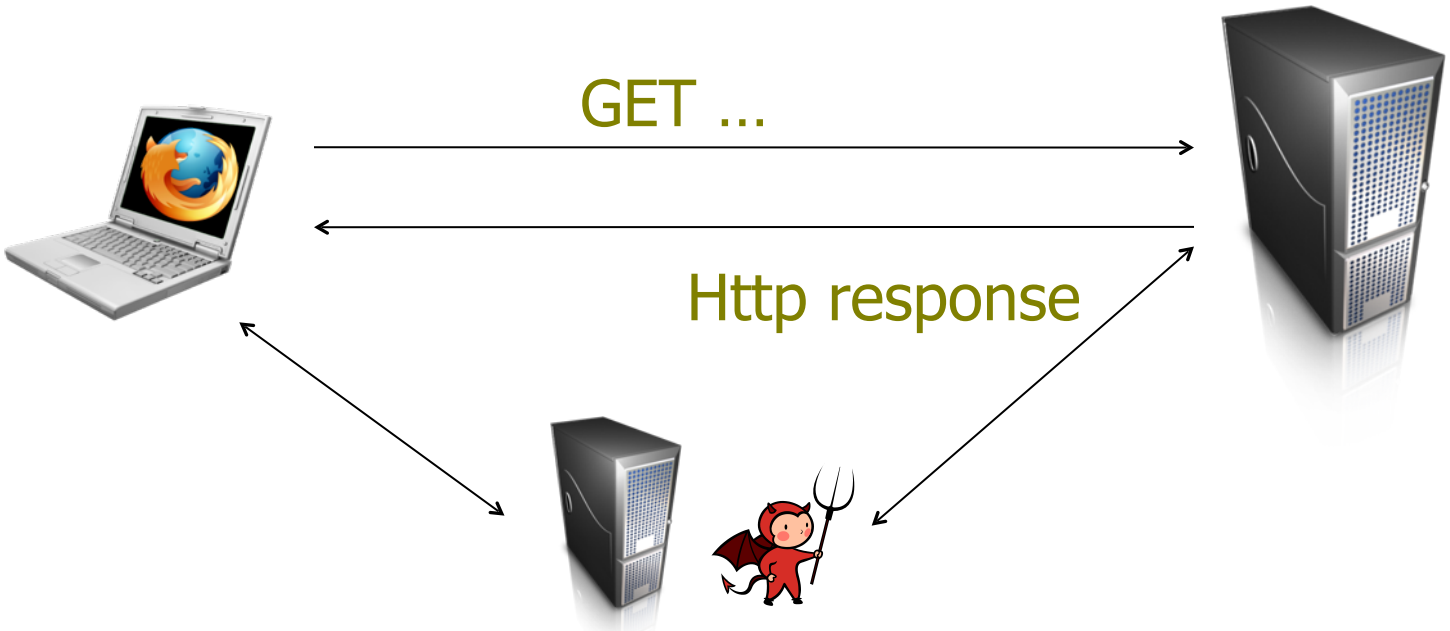
Web Security Risks

- What are we defending? From whom?
- Risk #3: an attacker spies on or tampers with a client's interaction with a website
  - Possibly by baiting the client to visit its own site



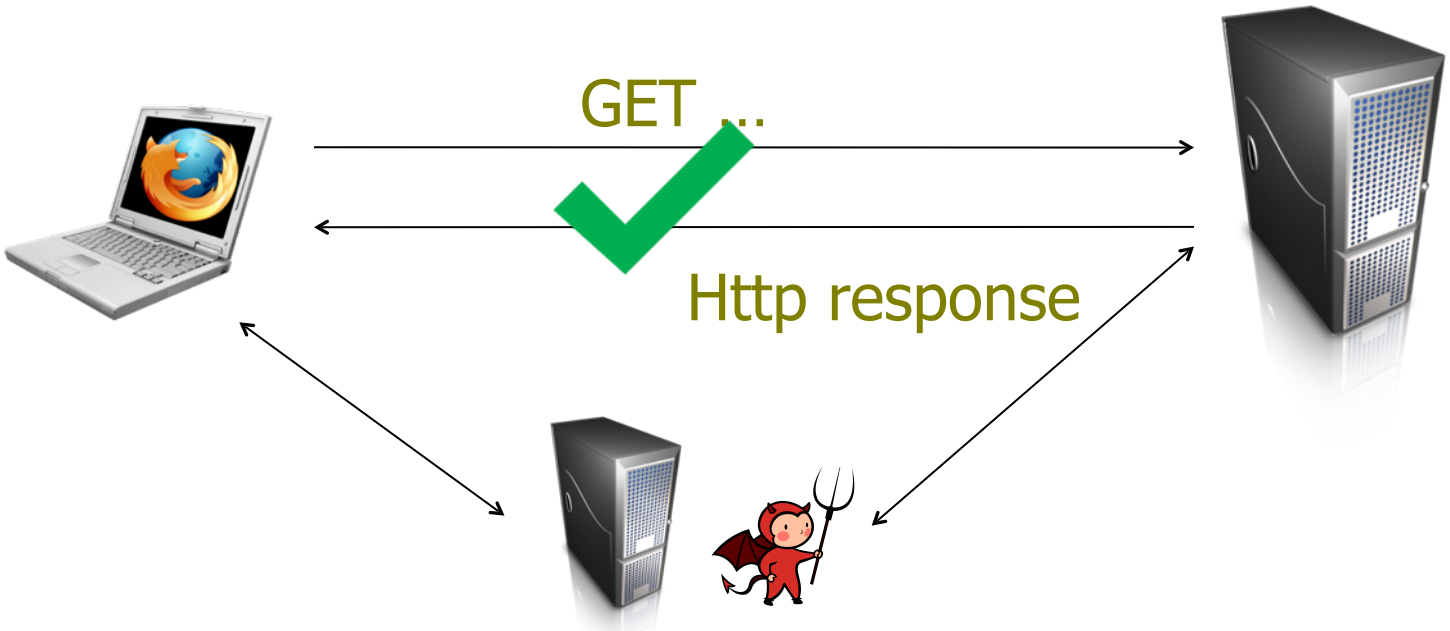
Web Security Risks

- Will focus on risk #3 (more unique to web)
  - An on-path adversary is a concern, but we will defer it to crypto and network security; assume communication channel is **trusted** for now



Web Security Risks

- Will focus on risk #3 (more unique to web)
  - An on-path adversary is a concern, but we will defer it to crypto and network security; assume communication channel is **trusted** for now



WEB SECURITY

Review

Review

- Who are we defending from in Web security?

Review

- Who are we defending from in Web security?
  - Malicious clients, servers, and third parties

Review

- Who are we defending from in Web security?
  - Malicious clients, servers, and third parties
- How does a server track authenticated sessions?

Review

- Who are we defending from in Web security?
  - Malicious clients, servers, and third parties
- How does a server track authenticated sessions?
  - Cookies

Review

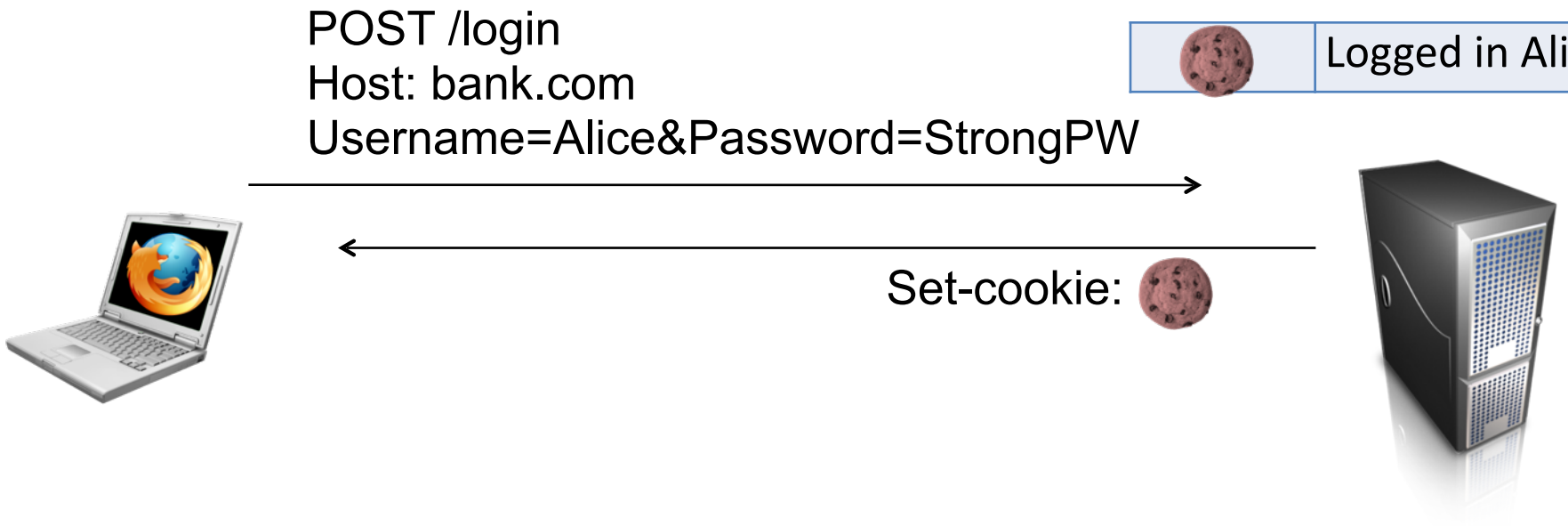
- Who are we defending from in Web security?
  - Malicious clients, servers, and third parties
- How does a server track authenticated sessions?
  - Cookies
- What is an important property of authentication cookies?

Review

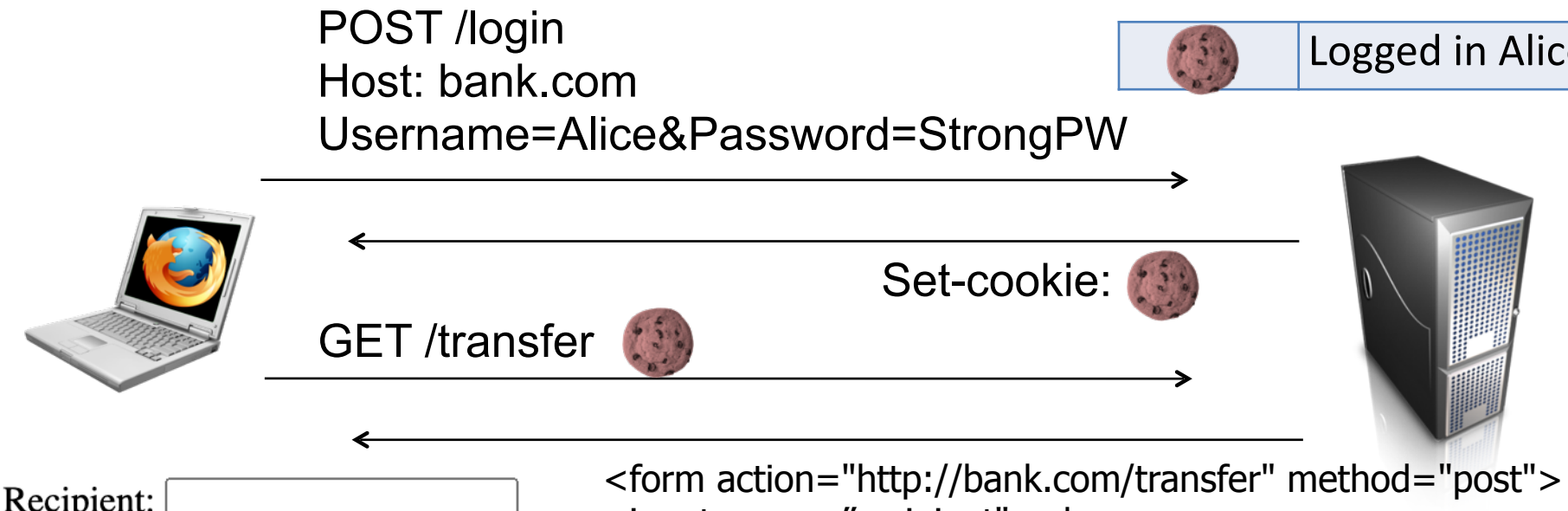
- Who are we defending from in Web security?
  - Malicious clients, servers, and third parties
- How does a server track authenticated sessions?
  - Cookies
- What is an important property of authentication cookies?
  - Must be secret

Cross Site Request Forgery (CSRF)

A Web Session

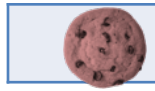


A Web Session



A Web Session

POST /login
Host: bank.com
Username=Alice&Password=StrongPW



Logged in Alice

Set-cookie:

GET /transfer

Recipient:
Amount:
Submit

```
<form action="http://bank.com/transfer" method="post">
<input name=”recipient"><br>
<input name=”amount"><br>
<input type="submit" value="Submit">
</form>
```

POST /transfer
Host: bank.com
recipient=Carol&amount=10

Think Like an Attacker

Think Like an Attacker

1. Bank will execute transfer if:

Think Like an Attacker

1. Bank will execute transfer if:
   A. It receives a POST request

Think Like an Attacker

1. Bank will execute transfer if:
   A. It receives a POST request
   B. With the right parameters (recipient, amount)

Think Like an Attacker

1. Bank will execute transfer if:
   A. It receives a POST request
   B. With the right parameters (recipient, amount)
   C. And correct authentication cookie

Think Like an Attacker

1. Bank will execute transfer if:
   A. It receives a POST request
   B. With the right parameters (recipient, amount)
   C. And correct authentication cookie
2. User's browser will send authenticated cookie with every request to bank.com

Think Like an Attacker

1. Bank will execute transfer if:
   A. It receives a POST request
   B. With the right parameters (recipient, amount)
   C. And correct authentication cookie
2. User's browser will send authenticated cookie with every request to bank.com
3. User's browser will send a POST request to bank.com when user submits a form with action="http://bank.com/" on any site

Think Like an Attacker

1. Bank will execute transfer if:
   A. It receives a POST request
   B. With the right parameters (recipient, amount)
   C. And correct authentication cookie
2. User's browser will send authenticated cookie with every request to bank.com
3. User's browser will send a POST request to bank.com when user submits a form with action="http://bank.com/" on any site
4. Users will submit the request when promised free iPhones

CSRF attack (GET)

CSRF attack (GET)

CSRF attack (GET)

Click on
“Get free iPhone”

CSRF attack (GET)

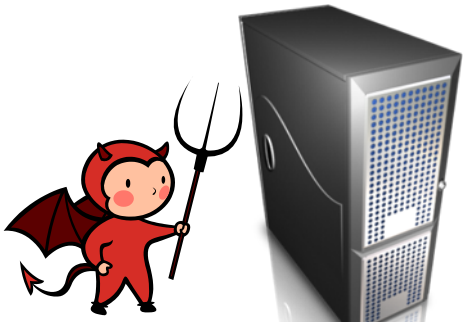
bank server

Click on
“Get free iPhone”

CSRF attack (GET)

bank server

Click on
“Get free iPhone”



attack server

CSRF attack (GET)

bank server

Click on
“Get free iPhone”

attack server

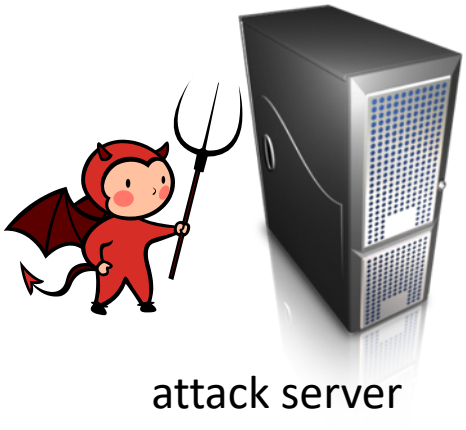
```
<html>
<title>Free iPhone!</title>
<img: src=“http://bank.com/transfer/recipient=attacker&amount=100”>
</html>
```

CSRF attack (GET)

Click on
“Get free iPhone”

bank server

GET /transfer//transfer/recipient=attacker&amount=100



attack server

```
<html>
<title>Free iPhone!</title>
<img: src=“http://bank.com/transfer/recipient=attacker&amount=100”>
</html>
```

CSRF attack (POST)

CSRF attack (POST)

CSRF attack (POST)

bank
server

CSRF attack (POST)

bank server

attack server

CSRF attack (POST)

bank server

attack server

```
<html>
<title>Free iPhone!</title>

<form method="post"
action="http://bank.com/transfer/">
<input type="submit" value="Get free iPhone!">
```

CSRF attack (POST)

Click on
“Get free iPhone”

bank
server

attack server

```
<html>
<title>Free iPhone!</title>

<form method=“post”
action=“http://bank.com/transfer/”>
<input type=“submit” value=“Get free iPhone!”>
```

CSRF attack (POST)

Click on
“Get free iPhone”

POST /transfer

bank
server

attack server

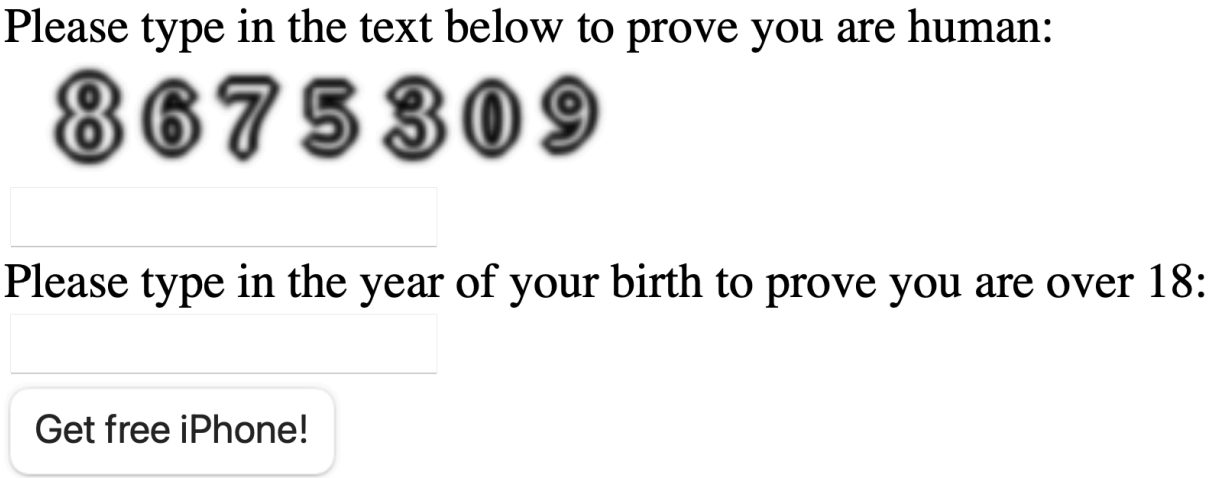
```
<html>
<title>Free iPhone!</title>

<form method=“post”
action=“http://bank.com/transfer/”>
<input type=“submit” value=“Get free iPhone!”>
```

Correct Arguments

How to supply correct arguments to post request?

Correct Arguments

How to supply correct arguments to post request?

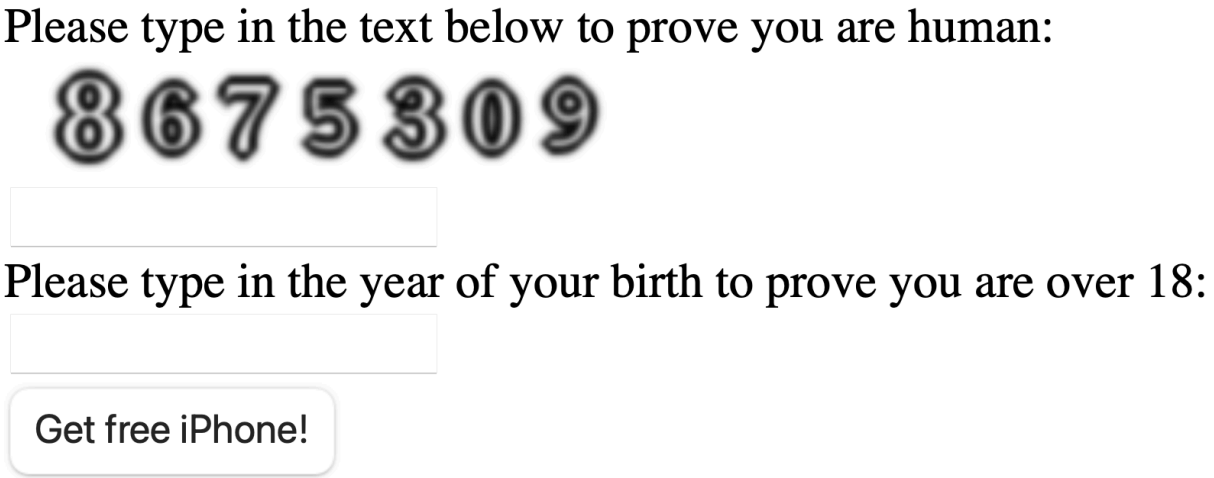


Correct Arguments

How to supply correct arguments to post request?

- Hidden parameters

```
<input type="hidden" name="recipient"
value="8675309">
<input type="hidden" name="amount" value="1000">
```



Think Like a Defender

1. Bank will execute transfer if:
   A. It receives a POST request
   B. With the right parameters (recipient, amount)
   C. And correct authentication cookie
2. User's browser will send authenticated cookie with every request to bank.com
3. User's browser will send a POST request to bank.com when user submits a form with action="http://bank.com/" on any site
4. Users will submit forms when promised free iPhones

Which of these can we change to stop attack?

CSRF Defenses

- SameSite cookie
  - Let browser attach cookie only if the request originates from the same site (exceptions exist)

Name	Value	Domain	Pa...	Expires / Max-...	Size	Ht...	Se...	SameSite	P
OptanonAlertBoxClosed	2024-09-24T02:37:...	.illinois.edu	/	2024-12-23T...	45			Lax	
OptanonConsent	isGpcEnabled=0&dat...	.illinois.edu	/	2024-12-23T...	260			Lax	
_csrf_token	%2BKp%2Bpa1jRoy0...	canvas.illino...	/	Session	113		✓		
_ga	GA1.1.191417252.172...	.illinois.edu	/	2025-10-29T...	29				
_ga_71JGWHBFGH	GS1.1.1727142530.8....	.illinois.edu	/	2025-10-29T...	52				
_hp2_id.3001039959	%7B%22userId%22...	.illinois.edu	/	2025-10-23T...	371		✓	None	
_hp2_props.30010399...	%7B%22Base.appNa...	.illinois.edu	/	2025-10-23T...	60		✓	None	
_hp2_ses_props.3001...	%7B%22ts%22%3A1...	.illinois.edu	/	2024-09-24T...	125		✓	None	
_legacy_normandy_se...	hUbD3JvvYn3X0g0jtI...	canvas.illino...	/	Session	744	✓	✓		
canvas_session	hUbD3JvvYn3X0g0jtI...	canvas.illino...	/	Session	734	✓	✓	None	
dpUseLegacy	false	canvas.illino...	/	2024-12-31T...	16				
inst-fs-session	eyJpZGVudGl0aWVzI...	.inst-fs-iad-...	/	2024-09-25T...	143	✓	✓	None	
inst-fs-session.sig	heHEIO47hOc8FetH...	.inst-fs-iad-...	/	2024-09-25T...	46	✓	✓	None	
log_session_id	c0b240bf4a0f20386...	canvas.illino...	/	Session	46	✓	✓		

SameSite Cookie

- SameSite=None: always sent
- SameSite=Strict: not sent for cross-site requests
  - Will affect user experience when following a benign link from another website

SameSite Cookie

- SameSite=None: always sent
- SameSite=Strict: not sent for cross-site requests
  - Will affect user experience when following a benign link from another website
- SameSite=Lax: not sent for cross-site requests except for top-level GET requests
  - I.e., navigate to a new website

GET vs. POST

- GET for viewing and POST for changing states

GET vs. POST

- GET for viewing and POST for changing states
  - Same Origin Policy protects viewing (coming soon)

GET vs. POST

- GET for viewing and POST for changing states
  - Same Origin Policy protects viewing (coming soon)

- Bad practice: GET /transfer?recipient=bob&amount=10

GET vs. POST

- GET for viewing and POST for changing states
  - Same Origin Policy protects viewing (coming soon)

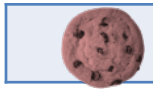
- Bad practice: GET /transfer?recipient=bob&amount=10
  - CSRF attack will succeed with SameSite=Lax, but will be prevented with SameSite=Strict

CSRF Defenses

- SameSite cookie is a relatively new defense, proposed in 2016
  - Not supported in old versions of browsers
- CSRF token is the recommended defense
- Can be combined for “defense in depth”

CSRF Token

POST /login
Host: bank.com
Username=Alice&Password=StrongPW



Logged in Alice

Set-cookie:

GET /transfer

Recipient:

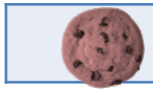
Amount:

Submit

POST /transfer
Host: bank.com
recipient=Carol&amount=10

CSRF Token

POST /login
Host: bank.com
Username=Alice&Password=StrongPW



Logged in Alice

Set-cookie:

GET /transfer

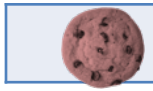
Recipient:
Amount:
Submit

```
<form action="http://bank.com/transfer" method="post">
<input name=”recipient"><br>
<input name=”amount"><br>
<input type=“hidden” name=“CSRFToken” value=“8d642fed”>
<input type="submit" value="Submit">
</form>
```

POST /transfer
Host: bank.com
recipient=Carol&amount=10

CSRF Token

form served:
8d642fed


Logged in Alice

POST /login
Host: bank.com
Username=Alice&Password=StrongPW

Set-cookie: 

GET /transfer 

Recipient:
Amount:
Submit

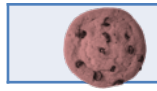
```
<form action="http://bank.com/transfer" method="post">
<input name=”recipient"><br>
<input name=”amount"><br>
<input type=“hidden” name=“CSRFToken” value=“8d642fed”>
<input type="submit" value="Submit">
</form>
```

POST /transfer
Host: bank.com
recipient=Carol&amount=10

CSRF Token

form served:
8d642fed

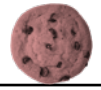
POST /login
Host: bank.com
Username=Alice&Password=StrongPW



Logged in Alice

Set-cookie:

GET /transfer



Recipient:
Amount:
Submit

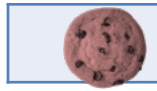
```
<form action="http://bank.com/transfer" method="post">
<input name=”recipient"><br>
<input name=”amount"><br>
<input type=“hidden” name=“CSRFToken” value=“8d642fed”>
<input type="submit" value="Submit">
</form>
```

POST /transfer
Host: bank.com

recipient=Carol&amount=10 &CSRFToken=8d642fed

CSRF Token

form served:
8d642fed


Logged in Alice

POST /login
Host: bank.com
Username=Alice&Password=StrongPW

Set-cookie: 

Click Here Free iPhone !!!

(cookie automatically attached by browser)

POST /transfer
Host: bank.com
recipient=attacker&amount=100 

JavaScript Sandbox

- JavaScript is an *interpreted* language running in a *sandbox*
- Highly limited access to system
  - Can't e.g., read/write your files
- Instead, focus on implementing interactive browser functionality
  - Take input from user (text, clicks)
  - Update web page
  - Make new requests
  - Read cookies

Same-Origin Policy (SOP)

Why We Need SOP

Why We Need SOP

- Javascript is powerful; it can
  - Alter page contents
  - Track events (mouse clicks, motion, keystrokes)
  - Issue web requests & read replies

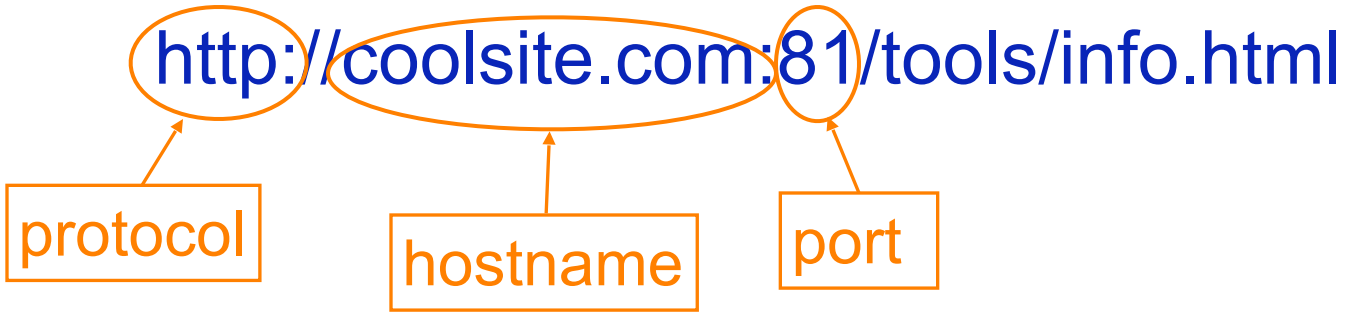
- Same-origin policy ensures a page's elements can be accessed only by its own Javascript

Same-Origin Policy

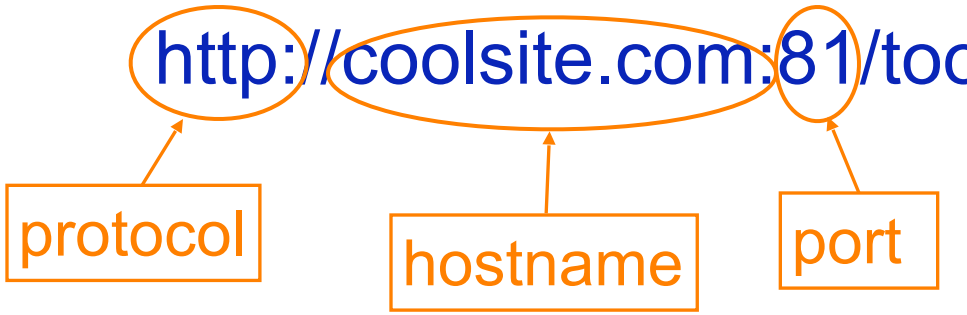
Same-Origin Policy

http://coolsite.com:81/tools/info.html

Same-Origin Policy



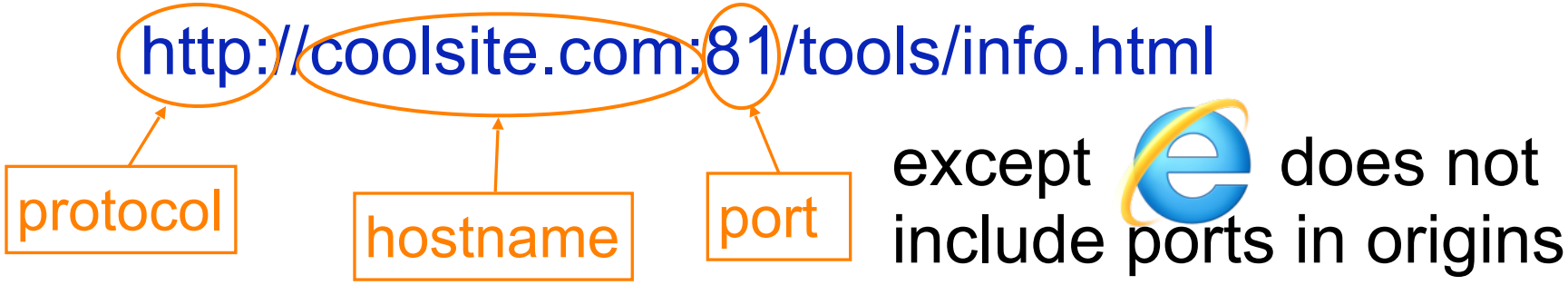
Same-Origin Policy



except  does not include ports in origins

Same-Origin Policy

- Granularity of protection: the *origin*
- Origin = (protocol, hostname, port)



- It is **string matching**! Given two URLs, if these match, they have the same origin, else they do not (even though logically they may)

Exercises: Same origin?

Originating document	Accessed document
http://wikipedia.org/a/	http://wikipedia.org/b/
http://wikipedia.org/	http://www.wikipedia.org/
http://wikipedia.org/	https://wikipedia.org/
http://wikipedia.org/a/	http://wikipedia.org:80/b/
http://wikipedia.org:81/	http://wikipedia.org/

Exercises: Same origin?

Originating document	Accessed document
http://wikipedia.org/a/	http://wikipedia.org/b/
http://wikipedia.org/	http://www.wikipedia.org/
http://wikipedia.org/	https://wikipedia.org/
http://wikipedia.org/a/	http://wikipedia.org:80/b/
http://wikipedia.org:81/	http://wikipedia.org/

Exercises: Same origin?

Originating document	Accessed document
http://wikipedia.org/a/	http://wikipedia.org/b/
http://wikipedia.org/	http://www.wikipedia.org/
http://wikipedia.org/	https://wikipedia.org/
http://wikipedia.org/a/	http://wikipedia.org:80/b/
http://wikipedia.org:81/	http://wikipedia.org/

Exercises: Same origin?

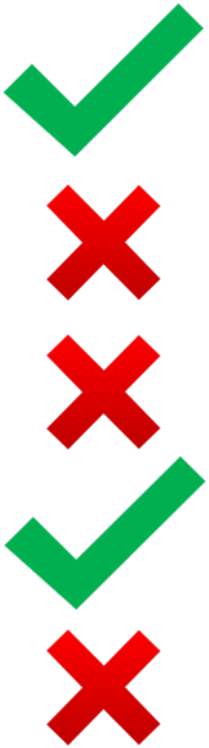
Originating document	Accessed document
http://wikipedia.org/a/	http://wikipedia.org/b/
http://wikipedia.org/	http://www.wikipedia.org/
http://wikipedia.org/	https://wikipedia.org/
http://wikipedia.org/a/	http://wikipedia.org:80/b/
http://wikipedia.org:81/	http://wikipedia.org/

Exercises: Same origin?

Originating document	Accessed document
http://wikipedia.org/a/	http://wikipedia.org/b/
http://wikipedia.org/	http://www.wikipedia.org/
http://wikipedia.org/	https://wikipedia.org/
http://wikipedia.org/a/	http://wikipedia.org:80/b/
http://wikipedia.org:81/	http://wikipedia.org/

Exercises: Same origin?

Originating document	Accessed document
http://wikipedia.org/a/	http://wikipedia.org/b/
http://wikipedia.org/	http://www.wikipedia.org/
http://wikipedia.org/	https://wikipedia.org/
http://wikipedia.org/a/	http://wikipedia.org:80/b/
http://wikipedia.org:81/	http://wikipedia.org/



Why We Need SOP

Why We Need SOP

- Javascript is powerful; it can
  - Alter page contents
  - Track events (mouse clicks, motion, keystrokes)
  - Issue web requests & read replies
- Same-origin policy ensures a page's elements can be accessed only by its own Javascript
- Demo

Cross-Site Scripting (XSS)

Cross-Site Scripting (XSS)

- Attacker takes advantage of a vulnerability to trick a website (e.g., bank.com) to send its user attacker's Javascript code

Cross-Site Scripting (XSS)

- Attacker takes advantage of a vulnerability to trick a website (e.g., bank.com) to send its user attacker's Javascript code
  - Subvert same origin policy

Cross-Site Scripting (XSS)

- Attacker takes advantage of a vulnerability to trick a website (e.g., bank.com) to send its user attacker's Javascript code
  - Subvert same origin policy
  - But does not necessarily involve another website

Cross-Site Scripting (XSS)

- Attacker takes advantage of a vulnerability to trick a website (e.g., bank.com) to send its user attacker's Javascript code
  - Subvert same origin policy
  - But does not necessarily involve another website
  - Possibly better name: Javascript injection

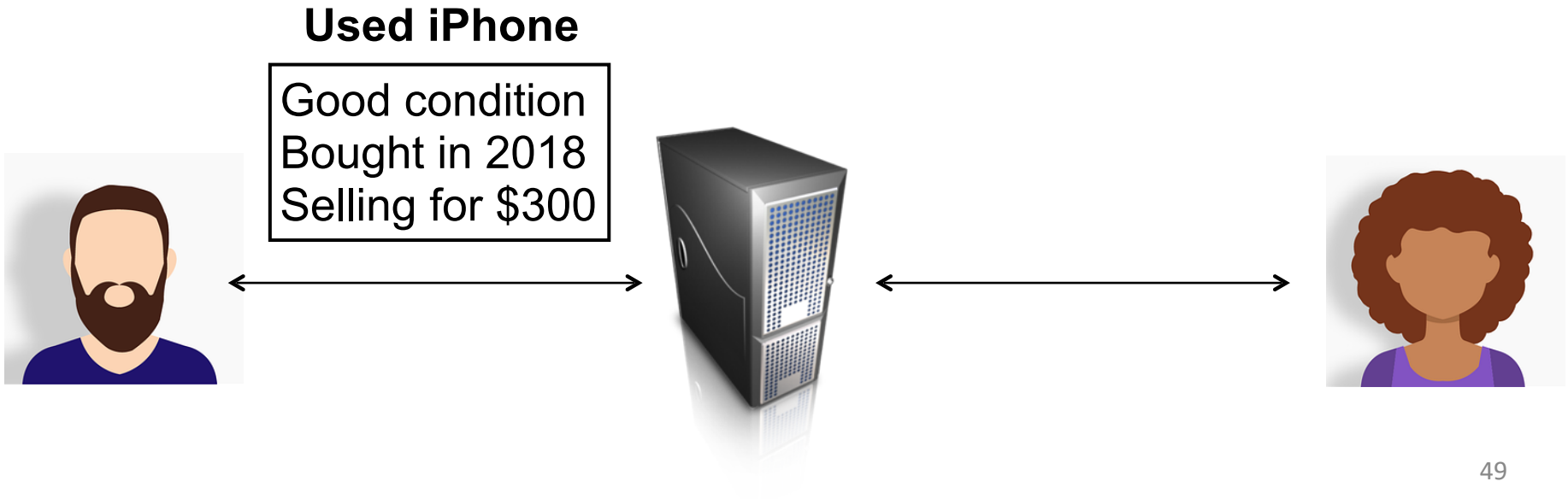
Cross-Site Scripting (XSS)

- Attacker takes advantage of a vulnerability to trick a website (e.g., bank.com) to send its user attacker's Javascript code
  - Subvert same origin policy
  - But does not necessarily involve another website
  - Possibly better name: Javascript injection

- Two types: stored XSS and reflected XSS

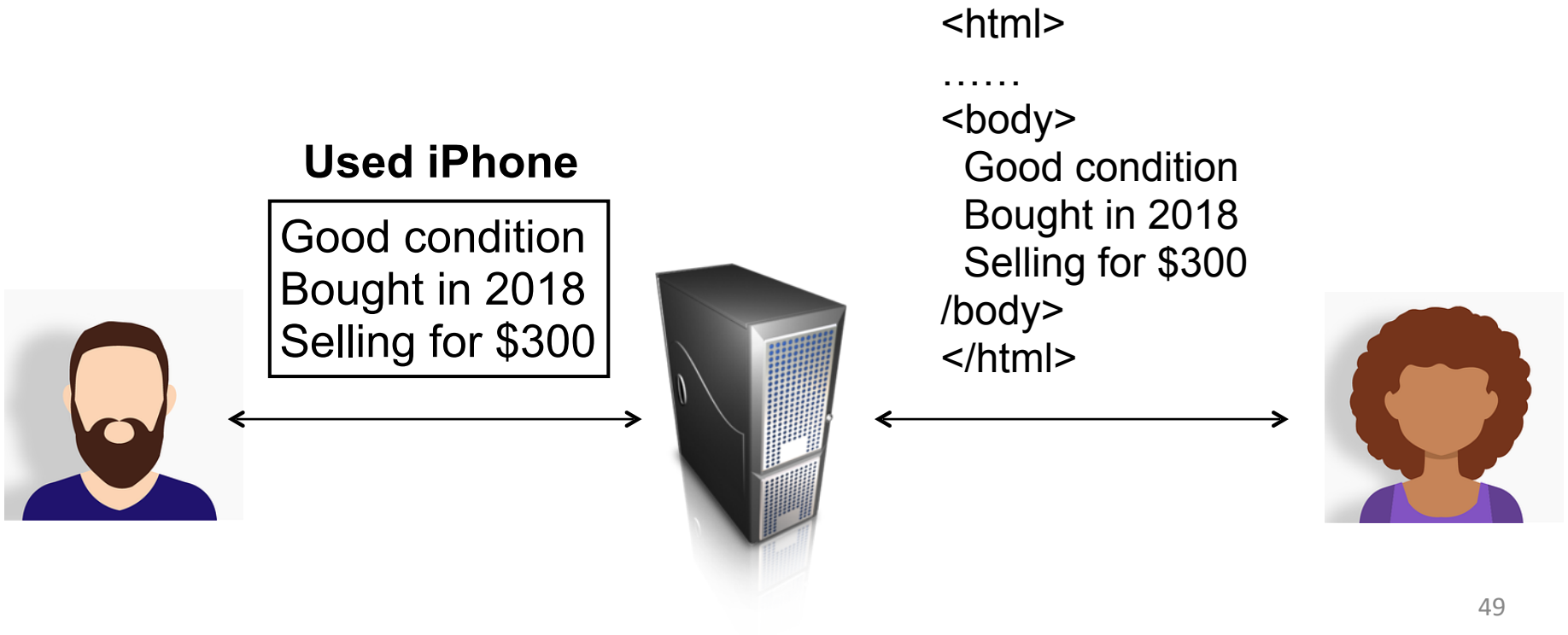
Stored XSS

- Imagine a website where users create and view postings

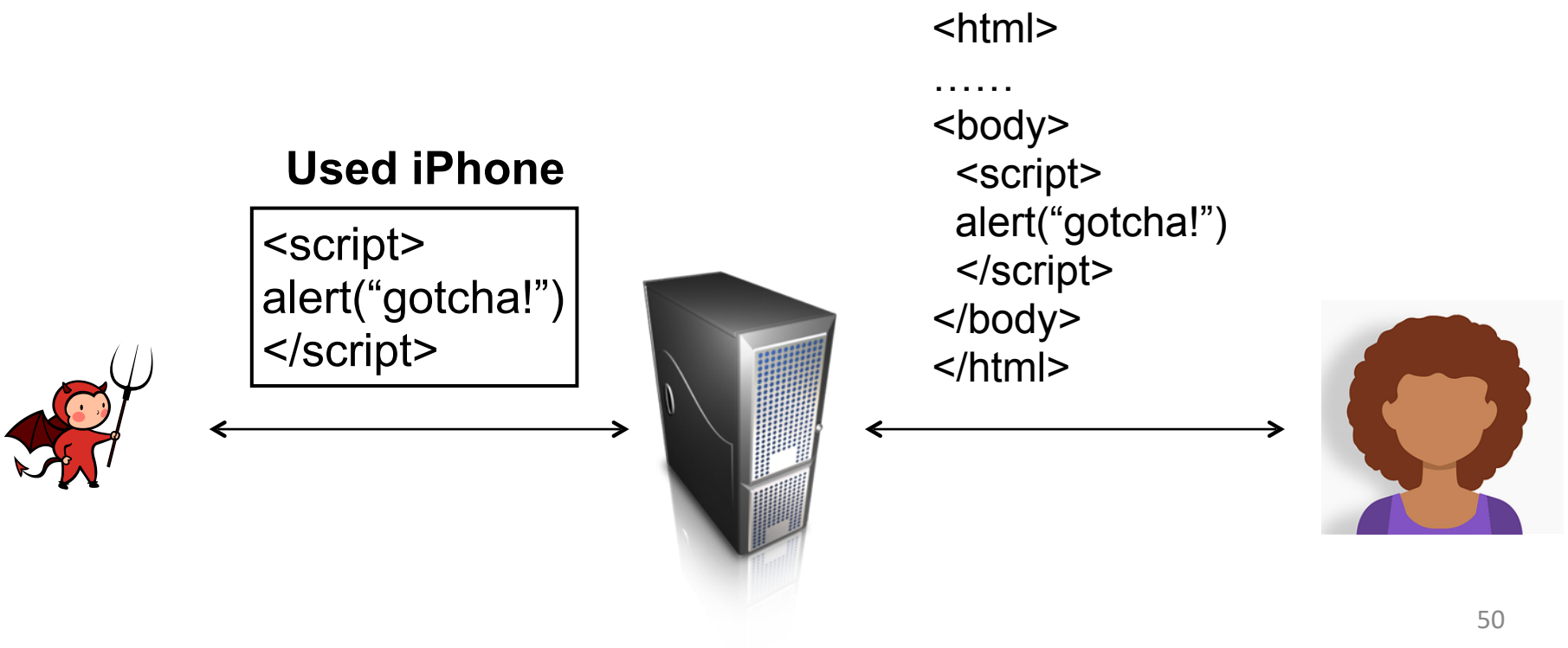


Stored XSS

- Imagine a website where users create and view postings

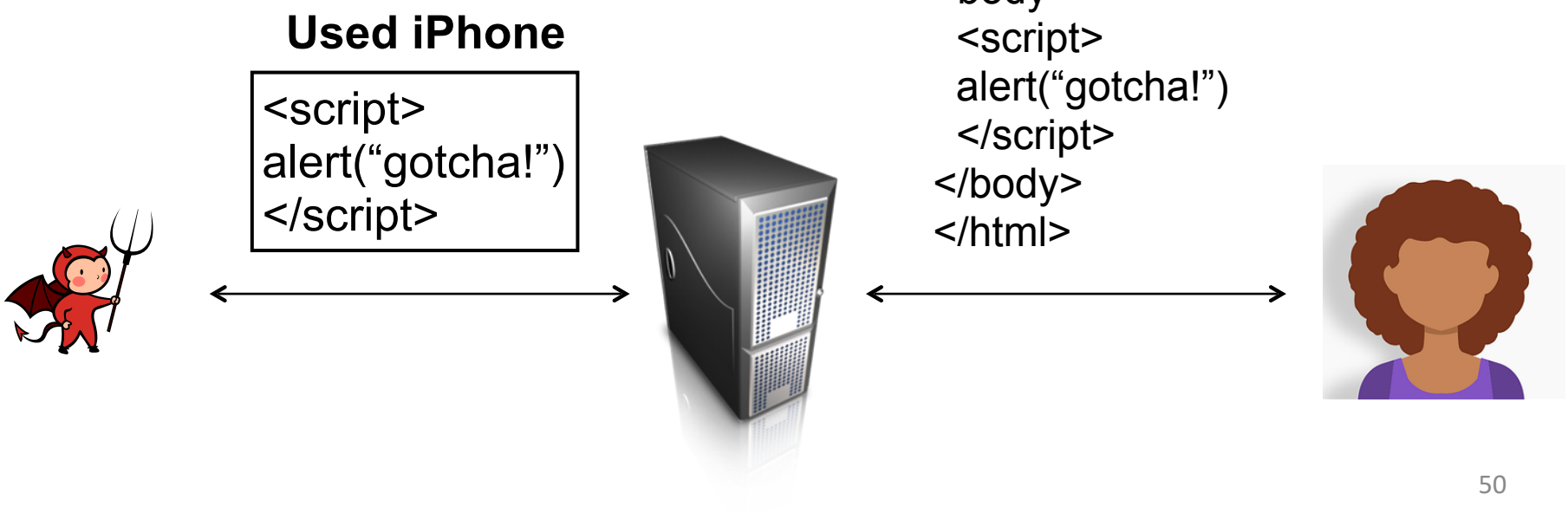


Stored XSS


Used iPhone
<script>
alert("gotcha!")
</script>
<html>
……
<body>
<script>
alert("gotcha!")
</script>
</body>
</html>

Stored XSS

- The injected Javascript code is from the victim website (same-origin)
  - Can take actions on user's account or send user data/cookie to attacker



Reflected XSS

- User input echoed back in HTTP response

Reflected XSS

- User input echoed back in HTTP response

Search results for Cool stuff:

……

Reflected XSS

- User input echoed back in HTTP response

Search results for Cool stuff:
……

```
<html>
……
<body>
Search results for Cool stuff
.....
</body>
</html>
```

Reflected XSS

- User input echoed back in HTTP response


```
<html>
……
<body>
Search results for <script>
alert(“gotcha”) </script>
.....
</body>
</html>
```

Reflected XSS

- User input echoed back in HTTP response
- Why is this a problem? The user is just injecting Javascript to itself …


```
<html>
……
<body>
Search results for <script>
alert("gotcha") </script>
.....
</body>
</html>
```

Reflected XSS

- User input echoed back in HTTP response
- Why is this a problem?

Click Here Free iPhone !!!

→ http://G00g1o.com/?search=<script>alert(“HiFromAttacker”)</script>

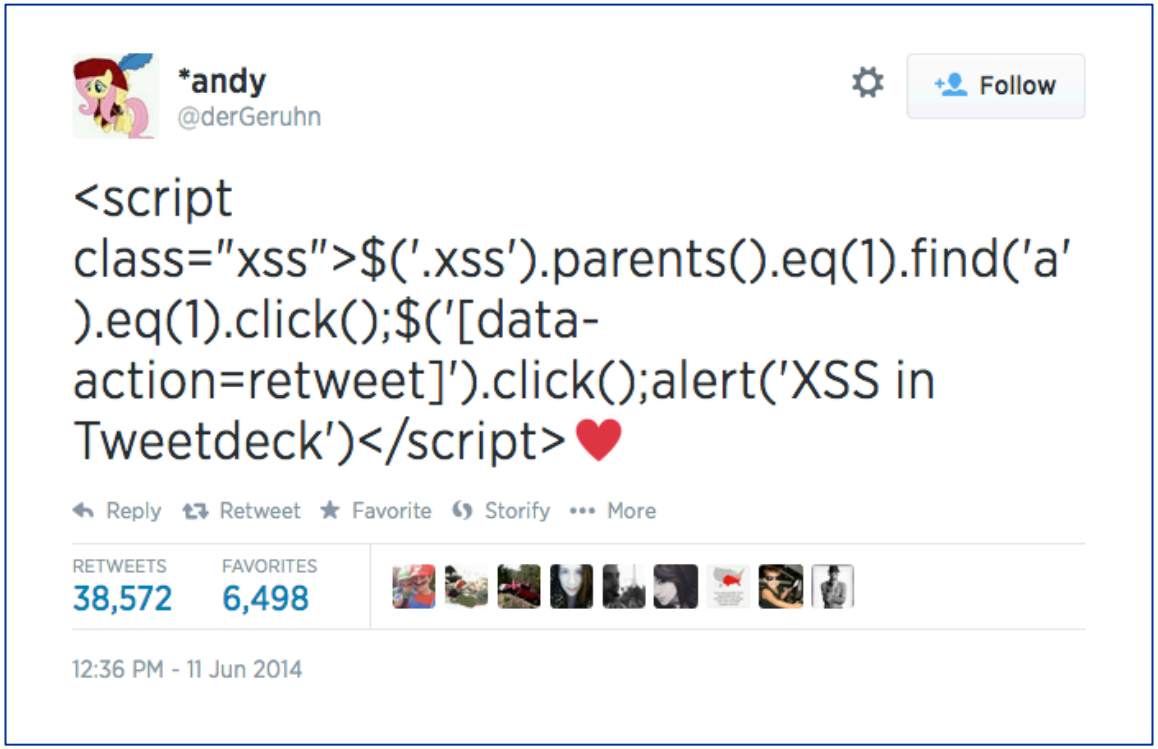
```
<html>
……
<body>
Search results for <script>
alert(“HiFromAttacker”) </
script>
.....
</body>
</html>
```

XSS Recap

- Goal: inject malicious Javascript code into a website's HTTP response to its user
  - Injected script has the website's origin
- Stored XSS
  - Leave content on the website
- Reflected XSS
  - Trick user to click on a malicious link → Javascript injected into a request to the vulnerable website → Vulnerable website echoes injected Javascript into its response to the user

Twitter XSS Vulnerability

- Some users constructed a tweet that were automatically retweeted



XSS Defenses

- Core issue: confusion between data and code

XSS Defenses

- Core issue: confusion between data and code
- Validate and escape user input

XSS Defenses

- Core issue: confusion between data and code
- Validate and escape user input
  - If user input should not contain special characters, (e.g., usernames, tracking number), enforce that!

XSS Defenses

- Core issue: confusion between data and code
- Validate and escape user input
  - If user input should not contain special characters, (e.g., usernames, tracking number), enforce that!
  - If users need to input special characters, escape them

<b> → HTML tag
<b> → <b> on screen

Character	Escape sequence
<	<
>	>
&	&amp
“	"
‘	'

XSS Defenses

- Core issue: confusion between data and code
- Validate and escape user input
- Content-Security-Policy (CSP): website specifies an allowlist of trusted scripts in HTTP header

XSS Defenses

- Core issue: confusion between data and code
- Validate and escape user input
- Content-Security-Policy (CSP): website specifies an allowlist of trusted scripts in HTTP header
  - Must use <script src=”trustedScript.js"></script>
  - Inline scripts will be ignored by browser

Summary

- Cookie is used for web session management
- Same-origin policy (SOP) isolates different websites on the client side (browser enforced)
- CSRF arises because browser automatically sends cookies
  - Defense: CSRF token, SameSite cookie
- XSS: Javascript injection
  - Defense: validate user input, CSP

To Learn More …

- Books
  - Pfleeger and Pfleeger, Chapter 4
  - Goodrich and Tamassia, Chapter 7
  - Anderson, Chapter 23
  - Du, Chapter 11
- Papers
  - Robust Defenses for Cross-Site Request Forgery - Barth
  - BLUEPRINT: Robust Prevention of Cross-site Scripting Attacks for Existing Browsers - Louw
  - Cross Site Scripting Explained - Klein
  - Securing Frame Communication in Browsers - Barth
  - Beware of Finer-Grained Origins – Jackson