Lecture 7 – Web Security

University of Illinois

ECE 422/CS 461

Some content adapted from materials by Raluca Ada Popa

Goals

- By the end of this lecture you should:
  - Understand web session management
  - Articulate the two main attacks unique to web: CSRF and XSS
  - Understand common defenses to CSRF and XSS

Recall Cookies

- A way for websites to store states on clients
  - Browser maintains all cookies it receives (cookie-jar)
  - Browser **automatically** attaches all cookies **in scope** in subsequent HTTP requests to the website

Web Sessions

- A sequence of user interactions with a website

- Session management
  - Authenticate user once, give user a secret token
    - Called session token, implemented by a cookie
  - User (browser) submits the secret token (cookie) with every subsequent HTTP request

A Web Session w/o Authentication

H1Y3fHC7I69v9W3oC2	Guest

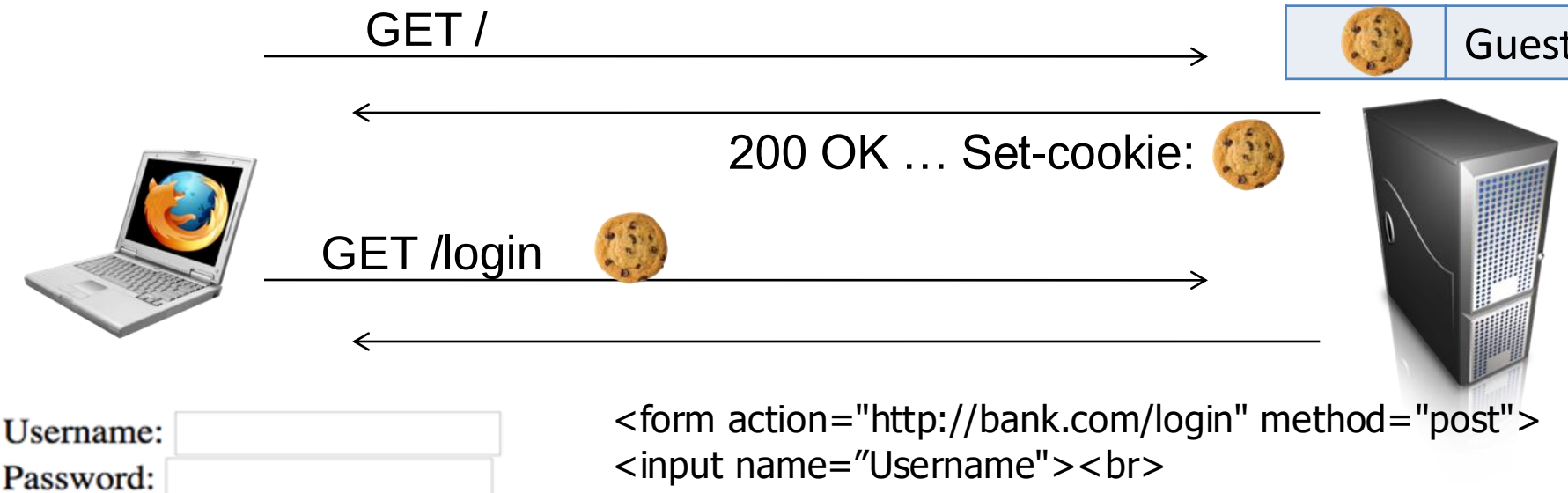
GET /

HTTP response:
Set-cookie: sessionID =
H1Y3fHC7I69v9W3oC2

GET /item123
Cookie: sessionID = H1Y3fHC7I69v9W3oC2

POST /cart
Cookie: sessionID = H1Y3fHC7I69v9W3oC2

A Web Session with Authentication



A Web Session with Authentication

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

200 OK … 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

Web Sessions

- A sequence of user interactions with a website
- Session management
  - Authenticate user once, give user a secret token
  - User (browser) submits the secret token (cookie) with every HTTP subsequent request
- Must protect the  session token! It gives attacker full access to user's account.

Cookie Attributes

Expiration/TTL	How long the key-value cookie should be kept
Domain	What domain (host) this cookie belongs to
Path	What path this cookie belongs to
SameSite=None	Send this cookie on a cross-site request
SameSite=Strict	Do NOT send this cookie on a cross-site request
SameSite=Lax	Send this cookie on a cross-site request only for top-level GET
HttpOnly	JavaScript can NOT access this cookie
Secure	Only send this cookie in HTTPS requests – not HTTP

Cookie Policy

- Each cookie has a domain and a path attributes
  - E.g., cs.illlinois.edu/courses/cs461
- Cookie policy: browser-enforced rules about
  - Can the server set a cookie for this domain/path?
  - Should this cookie (given its domain and path) be attached to an HTTP request to this server?
  - Confusingly, very different from same-origin policy

Cookie Policy: Setting a Cookie

- No restrictions on path
- Server with domain X can set a cookie whose domain attribute is Y, if Y is a suffix of X

  … and Y is not a top-level domain
  - cs.illinois.edu can set a cookie for cs.illinois.edu
  - cs.illinois.edu can set a cookie for illinois.edu
  - illinois.edu can not set a cookie for cs.illinois.edu

  - neither can set a cookie for edu

Cookie Policy: Attaching a Cookie

- A browser sends a cookie on an HTTP request if cookie's domain is a suffix of server's domain and cookie's path is a prefix of server's path

  – A cookie with illlinois.edu/courses will be sent to cs.illinois.edu/courses/cs461

  – A cookie with cs.illlinois.edu/courses/cs461 will NOT be sent to illinois.edu/courses/cs461 or cs.illinois.edu/courses/

Cookie Sending Policy Intuition

- Domain and path define the cookie's **scope**
  - A domain suffix is larger scope
  - A path prefix is a larger scope

- Cookie is sent if it is **in scope**
  - i.e., cookie's scope ≥ HTTP request's scope

Back to Session Management

- Cookie policy ensures a session token/cookie will not be sent to a different website

- Is same-origin policy still important here?

Cross Site Request Forgery (CSRF)

Cross Site Request Forgery (CSRF)

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


Logged in Alice

200 OK … Set-cookie:

Click Here Free iPhone !!!

JavaScript of this page issues a HTTP request

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

(cookie automatically attached by browser)

Does same-origin policy prevent this attack?

Cross Site Request Forgery (CSRF)

- Client is logged into a legit website
- Attacker tricks the user into clicking on its malicious website
- Attacker website's JavaScript makes an HTTP request to the legit website
  - Browser will automatically attach the legit website's **session cookie**
  - Essentially, attacker made an HTTP request to the legit website that appeared to come from the user

CSRF Defenses

- CSRF token
  - Embed a secret token in HTML forms
  - Attacker cannot predict token value when crafting CSRF attempt

CSRF Token

form served:
8d642fed

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

200 OK … 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

200 OK … Set-cookie:

Click Here Free iPhone !!!

JavaScript of this page
issues a HTTP request

(cookie automatically
attached by browser)

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

CSRF Defenses

- CSRF token
  - Embed a secret token in HTML forms
  - Attacker cannot cannot predict token when crafting CSRF attempt
- SameSite cookie
  - Let browser attach cookie only if the request originates from the same site (exceptions exist)

Cookie Attributes

Expiration/TTL	How long the key-value cookie should be kept
Domain	What domain (host) this cookie belongs to
Path	What path this cookie belongs to
SameSite=None	Send this cookie on a cross-site request
SameSite=Strict	Do NOT send this cookie on a cross-site request
SameSite=Lax	Send this cookie on a cross-site request only for top-level GET
HttpOnly	JavaScript can NOT access this cookie
Secure	Only send this cookie in HTTPS requests – not HTTP

SameSite Attribute of Cookies

- Set-cookie: sessionID=H1Y3fH, Expires=1 Oct 2020, Domain=bank.com, **SameSite=**…
  - **None**: Cookies will be sent in all contexts (both first-party and cross-site requests)
  - **Lax**: Cookies not sent for cross-site requests, except when a user is *navigating* to the site (top-level GET request, i.e., changes page)
  - **Strict**: Cookies not sent for cross-site requests
    - Will affect user experience when following a benign link from another website

SameSite Cookie

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

200 OK … Set-cookie:
SameSite=…

Click Here Free iPhone !!!

JavaScript of this page
issues a HTTP request

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

~~(cookie automatically attached by browser)~~

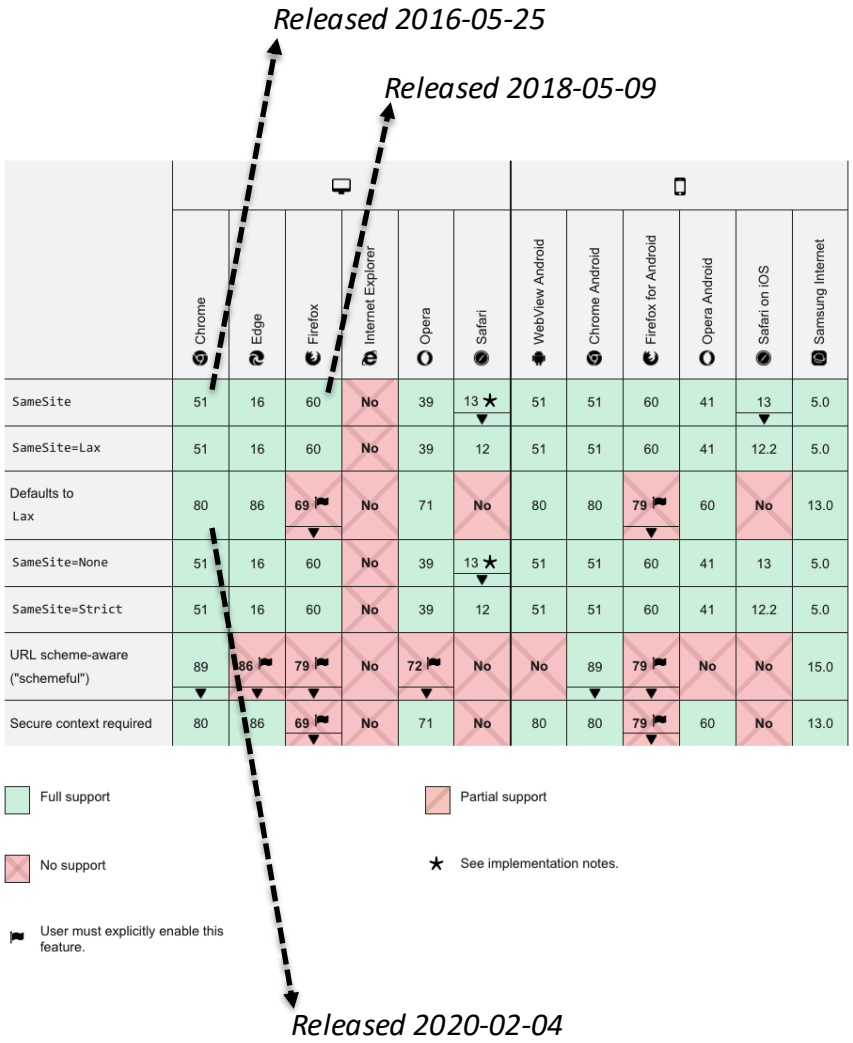
Which SameSite values prevent the attack here?

GET vs. POST

- Good practice: GET for viewing and POST for changing states
  - Then SameSite=Lax prevents unauthorized "writes" to server by restricting when cookie is sent
- Bad practice: use GET to change states
  - E.g., GET /transfer?recipient=bob&amount=10
  - Then, CSRF attack will succeed with SameSite=Lax, but will be prevented with SameSite=Strict

CSRF Defenses

- SameSite cookie is a relatively new defense
  - Not supported in older versions of browsers
- Combine CSRF token and SameSite cookie for “defense in depth”



Cross-Site Scripting (XSS)

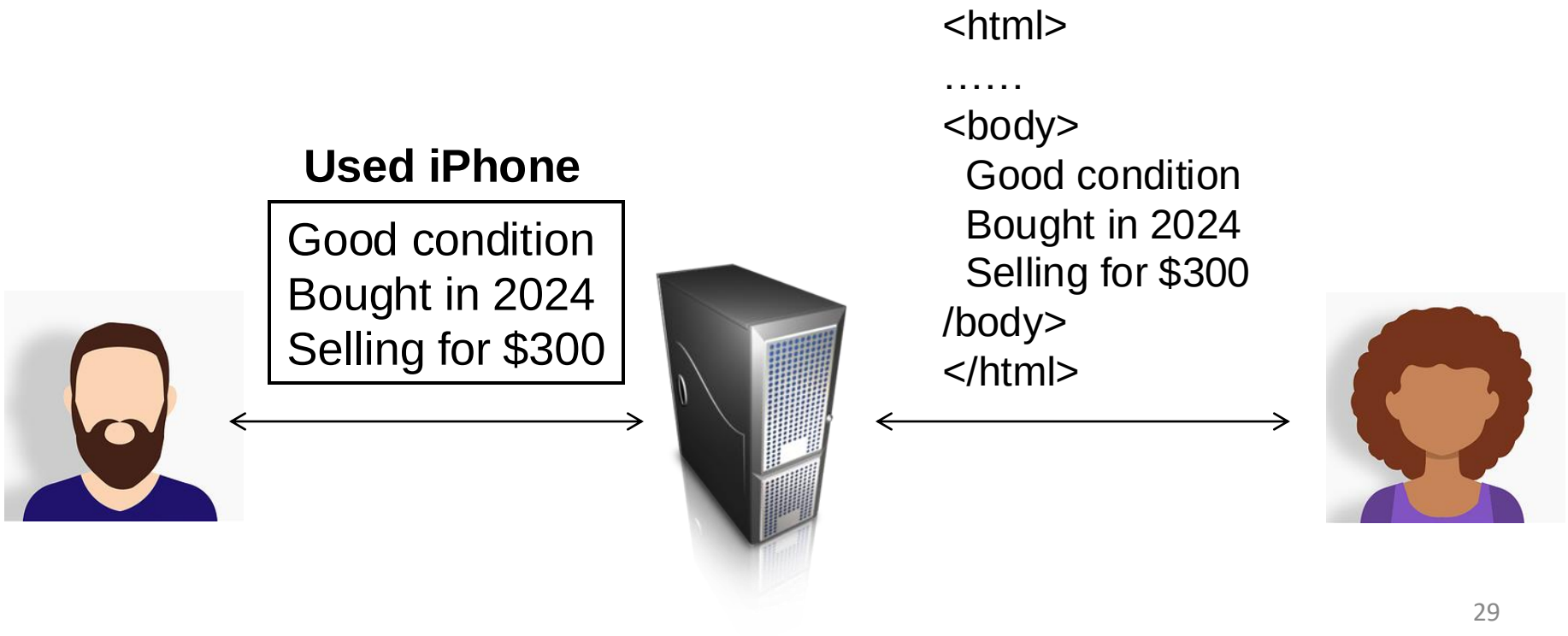
Cross-Site Scripting (XSS)

- Attacker takes advantage of a vulnerability to trick a website (e.g., bank.com) to send a user the attacker's malicious JavaScript code
  - Subverts same origin policy (**Why?**)
  - But does not necessarily involve another website
  - Possibly better name: JavaScript injection

- Two types: stored XSS and reflected XSS

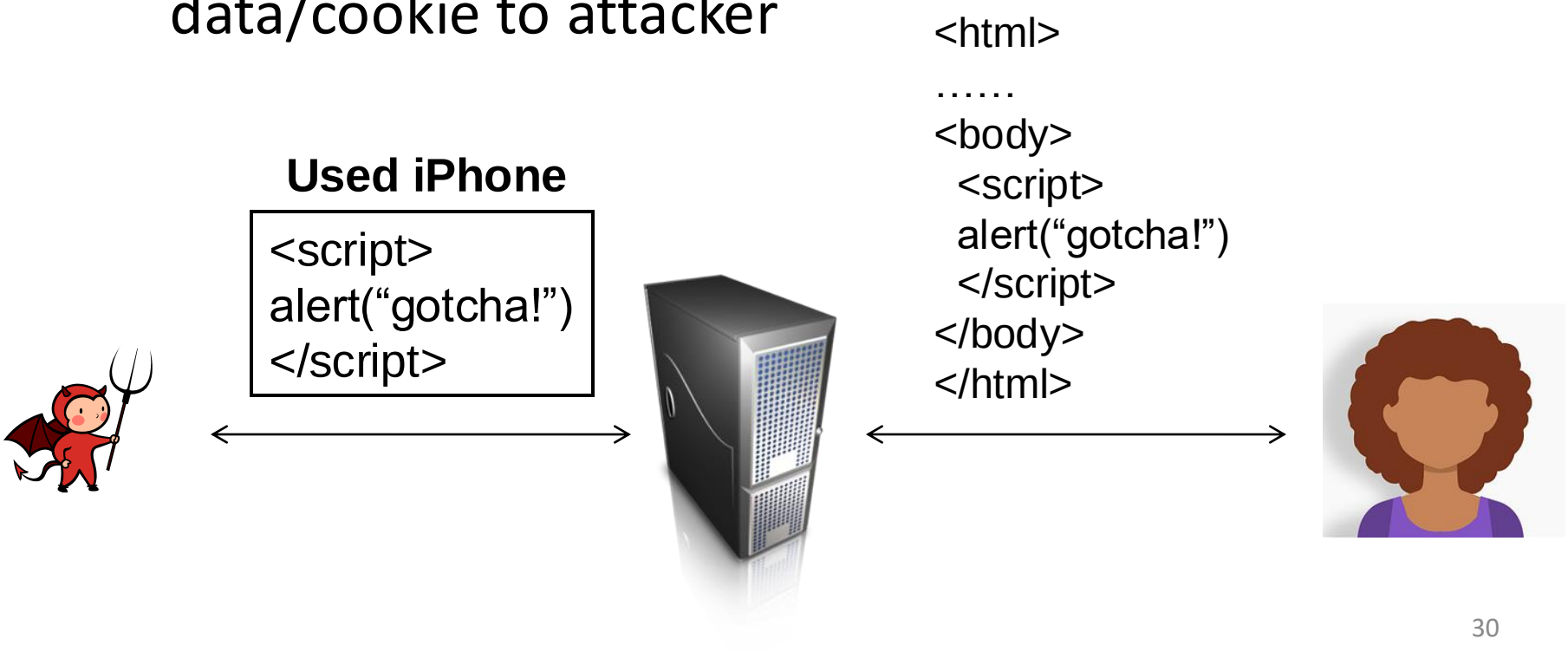
Stored XSS

- Imagine a website where users create and view postings



Stored XSS

- The injected JavaScript is sent by the 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

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

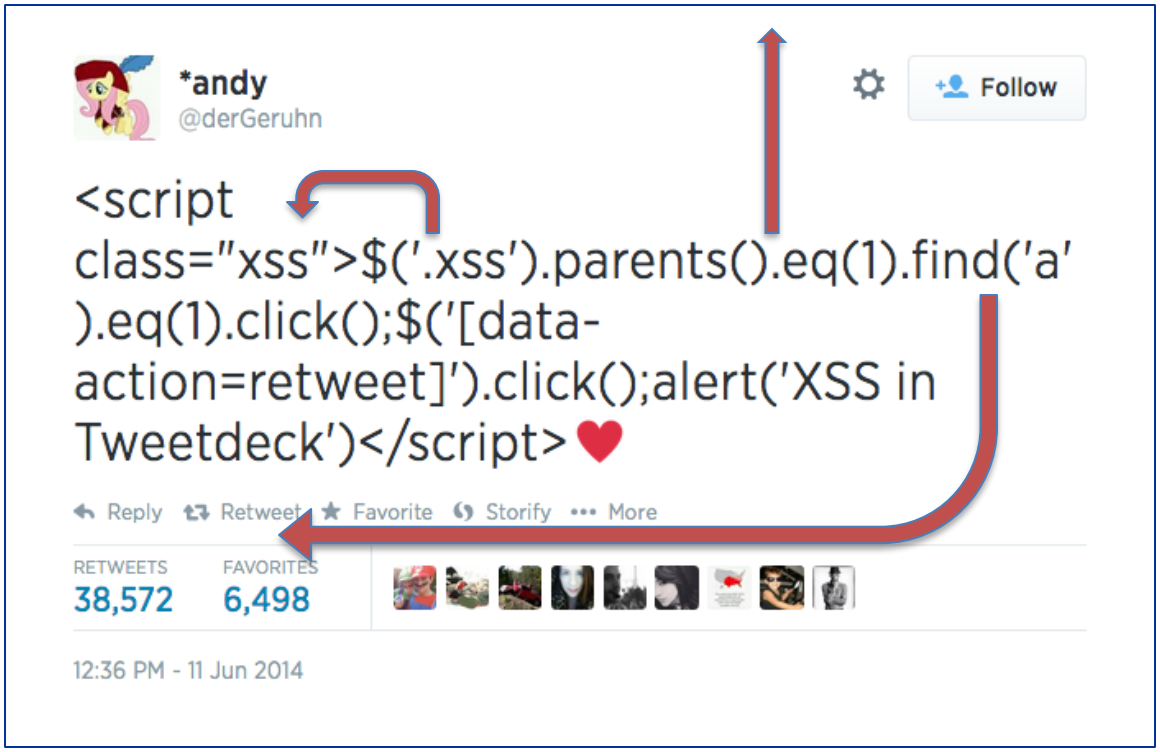
```
<html>
……
<body>
Search results for <script>
alert(“HiFromAttacker”)
</script>
.....
</body>
</html>
```

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 website → Website echoes injected JavaScript into its response to user

TweetDeck XSS Vulnerability (2014)

- Some users constructed a tweet that automatically re-tweeted itself on the TweetDeck platform



https://blog.sucuri.net/2014/06/serious-cross-site-scripting-vulnerability-in-tweetdeck-twitter.html

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
  - If user input should not contain special characters, (e.g., usernames, tracking number), enforce that!
  - If users need to input special characters, escape them
  - Does not fix all XSS vulnerabilities (recall different ways of including JavaScript code in HTML)

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:

```
Content-Security-Policy: script-src 'self' https://apis.google.com
```

  - Must use <script src="trustedScript.js"></script>
  - Inline scripts will be ignored by browser

Cookie Attributes

Expiration/TTL	How long the key-value cookie should be kept
Domain	What domain (host) this cookie belongs to
Path	What path this cookie belongs to
SameSite=None	Send this cookie on a cross-site request
SameSite=Strict	Do NOT send this cookie on a cross-site request
SameSite=Lax	Send this cookie on a cross-site request only for top-level GET
HttpOnly	JavaScript can NOT access this cookie
Secure	Only send this cookie in HTTPS requests – not HTTP

*HTTPS is HTTP augmented with cryptography to defend against network attackers

HttpOnly Cookies

- Orthogonal mitigation
  - Prevent JavaScript from accessing the cookie
  - Set-cookie: sessionID=H1Y3fH, Expires=1 Oct 2020, Domain=bank.com, **HttpOnly,** SameSite=…
- Prevent XSS from stealing cookie, but does not prevent XSS from doing other harm

Summary

- CSRF arises because browser automatically sends cookies (used for session management)
  - Defense: CSRF token, SameSite cookies
- XSS: JavaScript injection
  - Defense: validate user input, content-security policy