Browser Cross Site Science

Using Browsers for Evil

Amy Burnett

about:welcome

About Me

- RPISEC President 2016-2017
- Co-Founder - RET2 Systems
  - Focus on Security Education and Training
  - RET2 WarGames
    - In-Browser Exploitation Training Platform
- PWN2OWN 2018
  - Single Click Safari Exploit Escalating to Root

about:about

- Why Browsers?
- What Even is Cross Site
- SOP Interlude
- Cross-Site Requests
- Poking Holes in SOP
- XS-Leaking and Side Channels
- Browser Bugs
- Into the Future

Why Browsers?

- Extremely complex applications
- Ubiquitous use
- All your important stuff is in them
- Who even knows how they work??

So let's talk about Cross Site Attacks!

What Even is Cross Site??

~~Cr[illegible]ite~~ Scripting?

What Even is Cross Site??

"Cross Site" Scripting

- Attacker injects scripts into target site
- Can now control the page as if they were the user

ONLY INVOLVES ONE SITE

Cross Site Attacks

- Attacker uses malicious site to access target site
- Can potentially read or access the site from the browser

SOP Interlude

Same-Origin Policy (SOP)

- Origins can only access themselves (kinda)
- An origin is the location of a site
  - [twitter.com] is an origin
  - [mail.google.com] is a different origin
  - They shouldn’t be able to directly access each other

Cross Site Requests

Cross Site Requests

Cross Site Request Forgery (CSRF)

```
<img src="https://target.com/send_all_my_bitcoin?to=bob">
```

```
GET /send_all_my_bitcoin?to=bob
Host: target.com
Cookie: users_session=cookie
```

Cross Site POST Requests?

```
<form action="http://target.com/send_bitcoin">
    <input name="to" value="bob">
    <input name="amount" value="1337">
    <input type="submit" value="CLICK ME!">
</form>
<script>document.forms[0].submit()</script>
```

```
POST /send_bitcoin
Host: target.com
Cookie: users_session=cookie

to=bob&amount=1337
```

Preventing Cross Site Requests

Hard to prevent them... instead detect them

- Give users a secret token
  - Sent as a header
    - `X-CSRF-Token: 2ba9dcc3daaa87a1`
  - Inserted into a form
    - `<input type="hidden" value="2ba9dccdaaa87a1">`
  - `NOT A COOKIE!! Cookies get sent in XS-Requests`
- Use other HTTP methods
  - XS-Requests can only send `GET` and `POST`
  - Use `PATCH, PUT, DELETE`

What About XS-JSON??

"My site takes JSON as input, so I don't have to worry"

- Site Admin (before their account was compromised by clicking on a meme)

XS-Requests can only send:

```
application/x-www-form-urlencoded
multipart/form-data
text/plain
```

You better actually be checking the content type!

What About XS-JSON??

"My site takes JSON as input, so I don't have to worry"

- Site Admin (before their account was compromised by clicking on a meme)

XS-Requests can only send:

```
application/x-www-form-urlencoded
multipart/form-data
text/plain
```

You better actually be checking the content type!

Cross Site POST Requests?

```
fetch('https://target.com/send_bitcoin', {
    method: 'POST',
    credentials: 'include',
    headers: {'Content-Type': 'text/plain'},
    body: '{"to":"bob", "value": 1337}'
});
```

```
POST /send_bitcoin
Host: target.com
Content-Type: text/plain
Cookie: users_session=cookie

{"to":"bob", "value": 1337}
```

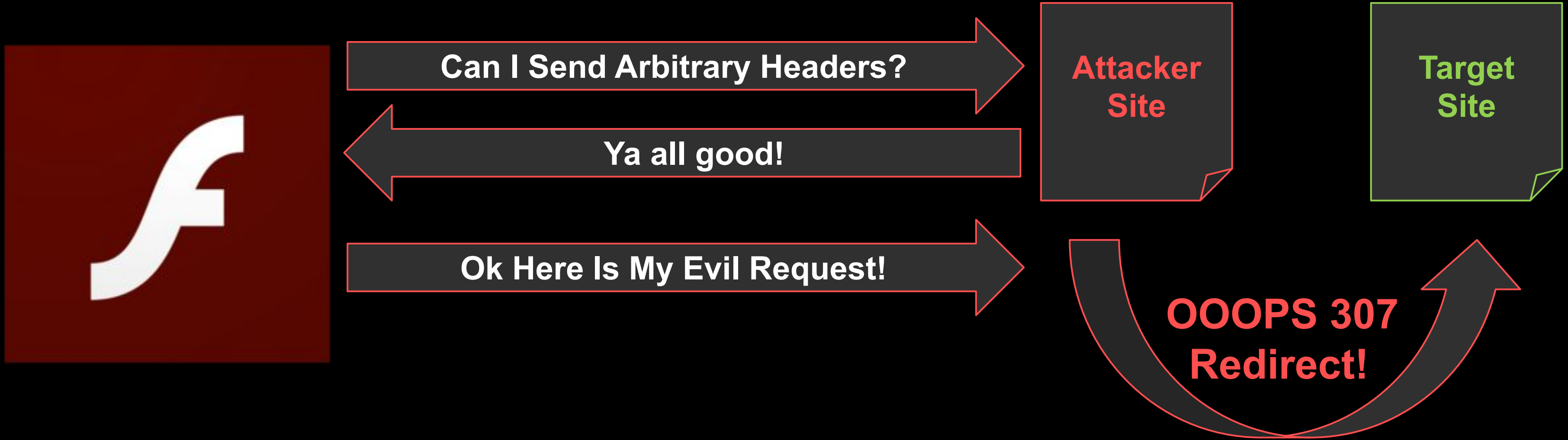
Ok I Check Content Type.. Good?

NOPE!! (well kinda)

Ok I Check Content Type.. Good?

NOPE!! (well kinda)

Issue with Flash XS-Requests -> Can Set Arbitrary Headers...



Ok, But Flash...........

Just convince someone to play your flash games

do people still do this???

Poking Holes In SOP

Poking Holes in SOP

Making Requests

- GET via Media, Scripts, CSS
- GET via iframes and `window.open(...)`
- Limited POST via forms or XHR requests (and fetch)
- Bodyless POST via the ping attribute `<a ping="...">`

Reading Responses

Making requests is not too bad

Reading responses is much harder…

Browser uses Cross-Origin Resource Sharing (CORS) to determine response blocking

CORS Interlude

Cross-Origin Resource Sharing (CORS)

- Method for sites to share data cross origins
- Site can set CORS headers in a response
- Gives permission for some other site to read the response

Normal sites will not have these options set for sensitive data

=

The browser will block the response data from being read

CORS Exceptions

Some XS-Response data can actually be used:

```
<img src="...">

<script src="...">

<link rel="stylesheet" href="...">
```

(JSONP actually abuses `<script>` loading XS Data)

Leaking with `<script>`?

Contents of `secret.html`:

```
<html><body>secret is hunter2</body></html>
```

```
<script src="https://target.com/secret.html">
```

```
Uncaught SyntaxError: Unexpected token <
    at <anonymous>:1:1
```

Leaking with <script>?

Contents of `secret.json`:

```
{"user":"alice", "secret": "hunter2"}
```

```
<script src="https://target.com/secret.json">
```

```
Uncaught SyntaxError: Unexpected token :
    at <anonymous>:1:1
```

Leaking with `<script>`?

Contents of `secret.json`:

```
["hunter1", "hunter2", "hunter3", "hunter4"]
```

```
<script>function Array(){alert(this)}</script>
<script src="https://target.com/secret.json">
```

Leaking with <script>?

Contents of se t.json:

```
["hunter1        2", "h              nter4"]
```

```
<scrip                                  cript>
<scrip                                  son">
```

This page says

hunter1,hunter2

Leaking with `<script>`?

Contents of `secret.json`:

```
while(1);["hunter1", "hunter2", "hunter3"]
```

```
<script>function Array(){alert(this)}</script>
<script src="https://target.com/secret.json">
```

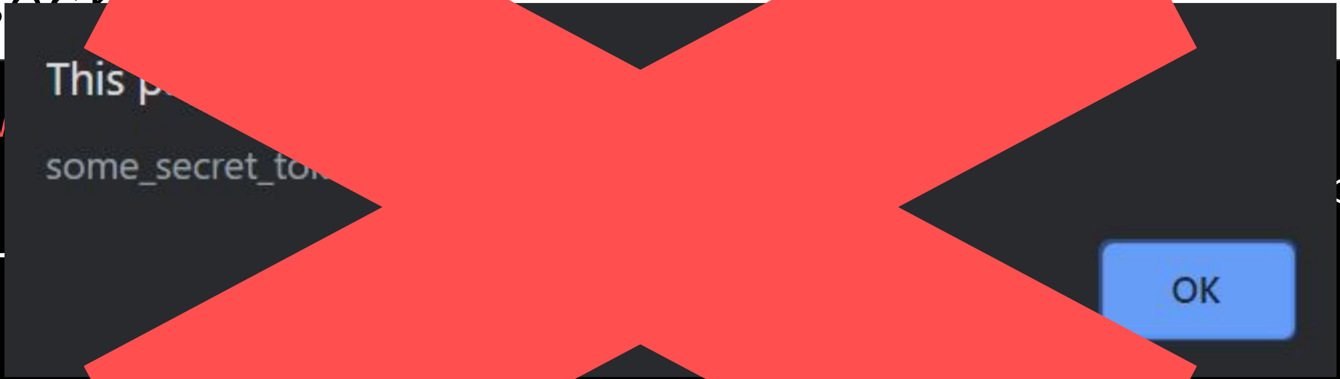
...

Leaking with `<script>`?

Contents of se`ret.txt:`

```
some_secr
```

```
<script>w                                                    {
    has:                                                   pt>
<scrip                                                    t">
```


This p

some_secret_to

OK

Leaking with `<script>`?

Contents of `secret.txt`:

```
some_secret_token
```

```
<script>window.__proto__ = new Proxy(window.__proto__, {
   has: function (target, name) {alert(name)}});</script>
<script src="https://target.com/secret.txt">
```

Sniffing Javascript

Browsers “Sniff” data to guess it it could be used

`X-Content-Type-Options: nosniff`

Error unless `application/javascript`

Cross Site Leaking

XS-Leaking

Side Channels!

- Perform some XS-Request
- Leak some bit of information about it
- Repeat…

Look for actions that can access target site's information

Detecting XS-Errors

```
<script src="https://target.com/endpoint">
```

200 HTTP response = JS execution attempt -> JS syntax error

Non 200 HTTP response = No JS execution

Request blocked (ie. nosniff) = No JS execution

We can catch the error to tell if the request succeeded

Detecting XS-Errors

```
<script>
var no_error = false;
window.onerror = function() {
  no_error = true;
}
setTimeout(()=>{
  if (no_error)
    alert("No HTTP error and not blocked");
  else
    alert("HTTP error or blocked");
}, 500)
</script>
<script src="http://target.com/endpoint"></script>
```

Detecting XS-Errors

Can detect if Chrome page errored while loading

- Set event hooks on iframe
- Check how many are called (different number for error)

We can abuse the Chrome XSS Auditor to cause errors

- Normally blocks XSS injected in page
- Attacker can selectively block page if element present
- Chrome recently disabled blocking by default

XS-Cache APIs

Browsers provide APIs to inspect cache profiling

- We can check the size of other site's cache
- Cause XS-Request to return large amount of data ->

Large change in cache size for positive result

Chrome recently fixed a bug which led to byte size resolution

- Still exploitable, padded random number of megabytes

Other Side-Channels

- Leak page changes via history
  - `window.history.length`
- Leak number of frames on page
  - `targetWindow.frames.length`
- Timing side-channel on blending CSS options
  - Measure how long translucent blend over iframe takes
  - Leaks data at a pixel resolution from target (but slow)

XS-Search

Using XS-Leaks to get a binary result from a search query

New vulnerability class, potentially lots of sites vulnerable

- Google issue tracker vulnerable via cache apis
- Twitter protected tweets vulnerable via history length

Will probably be a major issue for many sites to come!

Get those bug bounties ;)

Browser Bugs

Universal XSS (UXSS)

- Vulnerability in browser that allows XSS on ANY site
- Not the web-app's fault: can't be prevented

Arbitrary Code Execution

- Browser process enforces SOP and CORS
- Exploit browser process -> Bypass SOP + UXSS

Into the Future

Browsers are adding mitigations to limit leaks!

Chrome Site-Isolation

- Each origin has a separate browser process
- Broker uses IPC to limit what data is sent between sites
- Limits damage of Arbitrary Code Execution

Cross-Origin Request Blocking (CORB)

- Try to guess if request makes sense (ie html to a <script>)
- If it doesn’t make sense block right away!