

CS 253: Web Security

Same Origin Policy

```
<Cthon98> hey, if you type in your pw, it will show as stars
<Cthon98> ***** see!
<AzureDiamond> hunter2
<AzureDiamond> doesnt look like stars to me
<Cthon98> <AzureDiamond> ****
<Cthon98> thats what I see
<AzureDiamond> oh, really?
<Cthon98> Absolutely
<AzureDiamond> you can go hunter2 my hunter2-ing hunter2
<AzureDiamond> haha, does that look funny to you?
<Cthon98> lol, yes. See, when YOU type hunter2, it shows to us as *****
<AzureDiamond> thats neat, I didnt know IRC did that
<Cthon98> yep, no matter how many times you type hunter2, it will show to us as *****
<AzureDiamond> awesome!
<AzureDiamond> wait, how do you know my pw?
<Cthon98> er, I just copy pasted YOUR *****'s and it appears to YOU as hunter2 cause its your pw
<AzureDiamond> oh, ok.
```

What should be allowed?

- Should site A be able to **link** to site B?
- Should site A be able to **embed** site B?
- Should site A be able to **embed** site B and **modify** its contents?
- Should site A be able to **submit a form** to site B?
- Should site A be able to **embed images** from site B?
- Should site A be able to **embed scripts** from site B?
- Should site A be able to **read data** from site B?

Same Origin Policy

- This is the fundamental security model of the web
- **If you remember one thing from this class, this is it:**
 - Two pages from different sources should not be allowed to interfere with each other

The web is an operating system

- An **origin** is analogous to an OS **process**
- The **web browser** itself is analogous to an OS **kernel**
- Sites rely on the browser to enforce all the system's security rules
 - Just like in OSes, if there's a bug in the browser itself then all these rules go out the window

The basic rule

- Given **two separate JavaScript execution contexts**, one should be able to access the other only if the **protocols, hostnames, and port numbers** associated with their host documents **match exactly**.
- This "**protocol-host-port tuple**" is called an "**origin**".

https://example.com:4000/a/b.html?user=Alice&year=2019#p2

Protocol

Hostname

Port

Path

Query

Fragment

https://example.com:4000

Protocol

Hostname

Port

Same origin policy

```
function isSameOrigin (url1, url2) {  
  return url1.protocol === url2.protocol &&  
    url1.hostname === url2.hostname &&  
    url1.port === url2.port  
}
```

What should be allowed?

- Which actions should be subject to security checks?
- Where does one document begin or end?
- What is an origin?
- How much interaction should be allowed between non-cooperating origins?

Demo: Same origin policy

Recall

From <https://web.stanford.edu/class/cs106a/>:

```
document.cookie = 'sessionId=1234; Path=/class/cs106a/'
```

From <https://web.stanford.edu/class/cs253/>:

```
const iframe = document.createElement('iframe')
iframe.src = 'https://web.stanford.edu/class/cs106a/'
document.body.appendChild(iframe)
console.log(iframe.contentDocument.cookie)
```

Demo: Same origin policy + iframes

From <https://web.stanford.edu/class/cs253/>:

```
const iframe = document.createElement('iframe')
iframe.src = 'https://crypto.stanford.edu'
document.body.append(iframe)

console.log(iframe.contentDocument.cookie) // Not allowed!

iframe.src = 'https://example.com' // Allowed! Surprised?
```

Demo: Is cross-origin fetch allowed?

From <https://web.stanford.edu/class/cs253/>:

```
const res = await fetch('https://axess.stanford.edu')
const data = await res.body.text()
console.log(data)
```

- No! Would be a huge violation of Same Origin Policy.
- Any site in the world could read your grades if you're logged into Axess in another tab!

Same origin or not?

- **https://example.com/a/ → https://example.com/b/**
 - Yes!
- **https://example.com/a/ → https://www.example.com/b/**
 - No! Hostname mismatch!
- **https://example.com/ → http://example.com/**
 - No! Protocol mismatch!
- **https://example.com/ → https://example.com:81/**
 - No! Port mismatch!
- **https://example.com/ → https://example.com:80/**
 - Yes!

Problems

- Sometimes policy is too **narrow**: Difficult to get **login.stanford.edu** and **axess.stanford.edu** to exchange data.
- Sometimes policy is too **broad**: No way to isolate **https://web.stanford.edu/class/cs106a/** from **https://web.stanford.edu/class/cs253/** ...much to CS 106A staff's disappointment! 😊
- Policy is not enforced for certain web features!
 - You need to know which ones!

document.domain

- Idea: Need a way around Same Origin Policy to allow two different origins to communicate
- Two cooperating sites can agree that for the purpose of Same Origin Policy checks, they want to be considered equivalent.
- Sites must share a common top-level domain.
- Example: both **login.stanford.edu** and **axess.stanford.edu** may perform the following assignment:

```
document.domain = 'stanford.edu'
```

`document.domain` requires opt-in

- Both origins must explicitly opt-in to this feature
- So, if **attacker.stanford.edu** runs:

```
document.domain = 'stanford.edu'
```

- Then **attacker.stanford.edu** still cannot access content on **stanford.edu**!
- **stanford.edu** also needs to run the same code to opt-in to this behavior:

```
document.domain = 'stanford.edu'
```

- This is not a no-op, despite how it looks!

Originating URL	document.domain	Accessed URL	document.domain	Allowed?
http:// www.example.com/	example.com	http:// payments.example .com/	example.com	?
http:// www.example.com/	example.com	https:// payments.example .com/	example.com	?
http:// payments.example .com/	example.com	http:// example.com/	(not set)	?
http:// www.example.com/	(not set)	http:// www.example.com/	example.com	?

Originating URL	document.domain	Accessed URL	document.domain	Allowed?
http:// www.example.com/	example.com	http:// payments.example .com/	example.com	Yes
http:// www.example.com/	example.com	https:// payments.example .com/	example.com	No
http:// payments.example .com/	example.com	http:// example.com/	(not set)	No
http:// www.example.com/	(not set)	http:// www.example.com/	example.com	No

document.domain is a bad idea

- In order for **login.stanford.edu** and **axess.stanford.edu** to communicate, they must set:

```
document.domain = 'stanford.edu'
```

- This allows anyone on **stanford.edu** to join the party
 - Example: **attacker.stanford.edu** can also set **document.domain** to **stanford.edu** to become same origin with the others

Send messages from a parent page to a child iframe

- Idea: Need a way around Same Origin Policy to allow two different origins to communicate
- What if we encoded data in URL fragment identifiers?
 - Gap in same origin policy!
 - Parent is allowed to navigate child iframes
 - Child can poll for changes to the fragment identifier

https://example.com:4000/a/b.html?user=Alice&year=2019#p2

Protocol

Hostname

Port

Path

Query

Fragment

Demo: Fragment identifier cross-origin communication

Demo: Fragment identifier cross-origin communication

parent.html:

```
<h1>localhost:4000</h1>
<input name='val' />
<br /><br />
<iframe src='http://localhost:4001/child.html'></iframe>
<script>
  const input = document.querySelector('input')
  const iframe = document.querySelector('iframe')
  input.addEventListener('input', () => {
    iframe.src = `http://localhost:4001/child.html#${encodeURIComponent(input.value)}`})
</script>
```

child.html:

```
<h1>localhost:4001</h1>
<div></div>
<script>
  const div = document.querySelector('div')
  setInterval(() => {
    div.textContent = decodeURIComponent(window.location.hash).slice(1)
  }, 100)
</script>
```

The postMessage API

- Secure cross-origin communications between cooperating origins
- Send strings and arbitrarily complicated data cross-origin
- Useful features:
 - "Structured clone" algorithm used for complicated objects.
Handles cycles. Can't handle object instances, functions, DOM nodes.
 - "Transferrable objects" allows transferring ownership of an object.
It becomes unusable (neutered) in the context it was sent from.

Demo: postMessage cross-origin communication

parent.html:

```
<h1>localhost:4000</h1>
<input name='val' />
<br /><br />
<iframe src='http://localhost:4001/child.html'></iframe>
<script>
  const input = document.querySelector('input')
  const iframe = document.querySelector('iframe')
  input.addEventListener('input', () => {
    iframe.contentWindow.postMessage(input.value, 'http://localhost:4001')
  })
</script>
```

child.html:

```
<h1>localhost:4001</h1>
<div></div>
<script>
  const div = document.querySelector('div')
  window.addEventListener('message', event => {
    if (event.origin !== 'http://localhost:4000') return
    div.textContent = event.data
  })
</script>
```

More realistic example

- **axess.stanford.edu** wants to display name of logged in user, so it registers a listener for messages:

```
window.addEventListener('message', event => {
  setCurrentUser(event.data.name)
})
```

- Then it embeds an iframe to **login.stanford.edu** which runs:

```
const data = { name: 'Feross Aboukhadijeh' }
window.parent.postMessage(data, '*')
```

- This is insecure! Why?

axess.stanford.edu

axess.stanford.edu

login.stanford.edu

axess.stanford.edu

{ name: 'Feross Aboukhadijeh' }

login.stanford.edu





attacker.com

attacker.com

login.stanford.edu

attacker.com

{ name: 'Feross Aboukhadijeh' }

login.stanford.edu



Need to validate destination of messages!

- If an attacker embeds **login.stanford.edu**, they can listen to it's message which reveals the name of the logged in user!
- Solution: **login.stanford.edu** should specify intended recipient origin. Browser will enforce this.

```
const data = { name: 'Feross Aboukhadijeh' }
window.parent.postMessage(data, 'https://axess.stanford.edu')
```



attacker.com

attacker.com

axess.stanford.edu

attacker.com

axess.stanford.edu

login.stanford.edu

attacker.com

axess.stanford.edu

{ name: 'Marc Tessier-Lavigne' }

login.stanford.edu



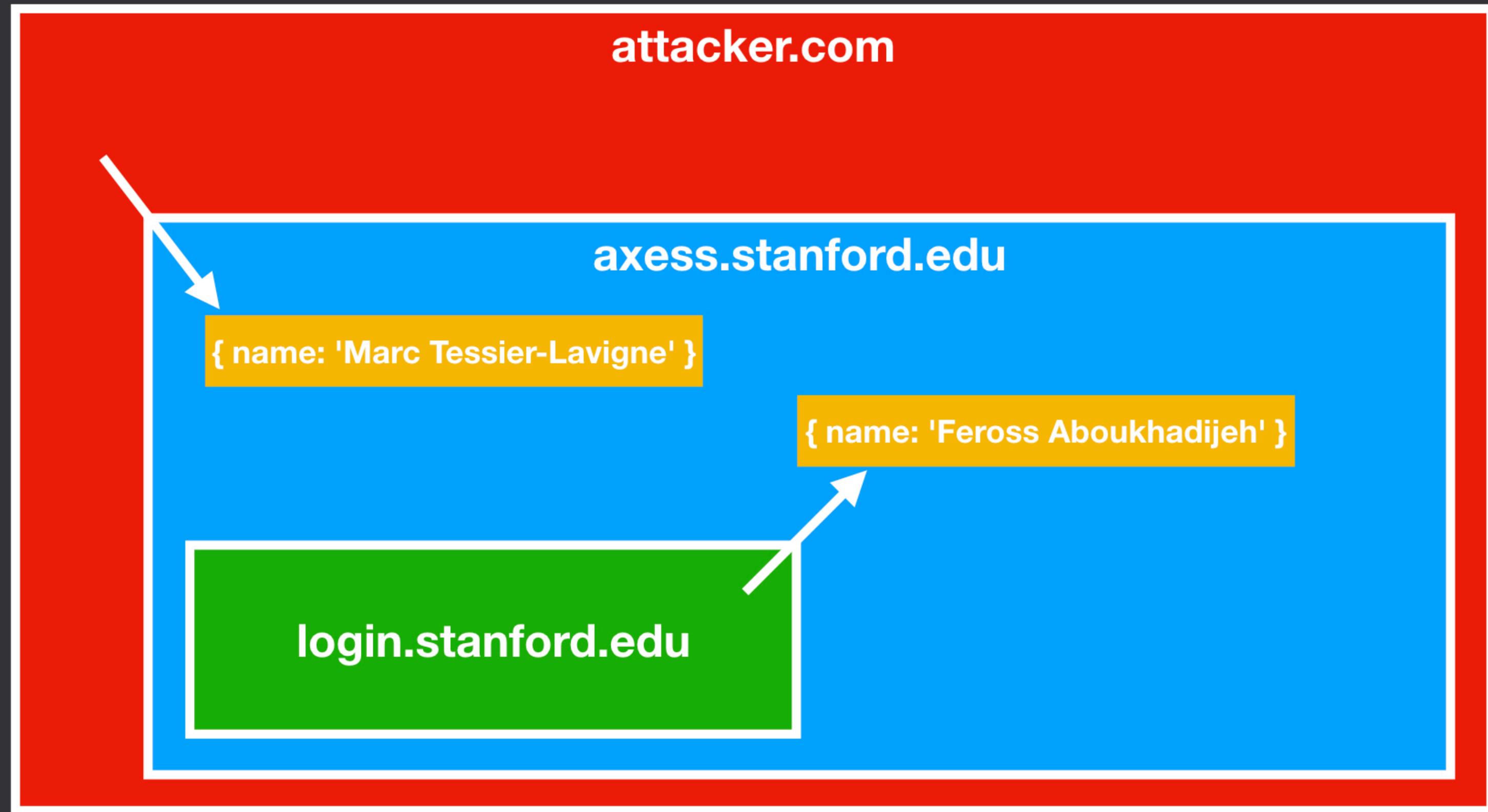
attacker.com

axess.stanford.edu

{ name: 'Marc Tessier-Lavigne' }

{ name: 'Feross Aboukhadijeh' }

login.stanford.edu



attacker.com

axess.stanford.edu

{ name: 'Marc Tessier-Lavigne' }

login.stanford.edu

{ name: 'Feross Aboukhadijeh' }

Need to validate source of messages!

- If an attacker has a reference to a **axess.stanford.edu** window (by e.g. embedding it in an iframe), they can send a message to it to trick it!
- Solution: **axess.stanford.edu** should verify source origin of message!

```
window.addEventListener('message', event => {
  if (event.origin !== 'https://login.stanford.edu') return
  setCurrentUser(event.data.name)
})
```

Integrity of postMessage

- Allows sender to specify what origins are permitted to receive the message
 - In case the URL of the target window has changed
- Provides recipient with the identity of the sender
 - In case another window is sending the message
- **Important:** Always specify intended recipient or expected sender!

Same origin policy exceptions

- There are **explicit opt-out** mechanisms like `document.domain`, fragment identifier communication, and the `postMessage` API
- There are also **automatic exceptions**
 - Need to be aware of these!
 - Source of many security issues!

Same origin policy exceptions

- Which of these requests from **example.com** are allowed?

```
<!doctype html>
<html lang='en'>
  <head>
    <meta charset='utf-8' />
    <link rel='stylesheet' href='https://other1.com/style.css' />
  </head>
  <body>
    <img src='https://other2.com/image.png' />
    <script src='https://other3.com/script.js'></script>
  </body>
</html>
```

Same origin policy exceptions

- Answer: All of them!
- Embedded static resources can come from another origin
 - Images (e.g. hotlinking to memes)
 - Scripts (e.g. Facebook like button, ads, tracking scripts)
 - Styles (e.g. Google Fonts)
- Why was it designed this way?

Same origin policy exceptions + ambient authority

- Remember: Ambient authority comes from cookies!
- Access to the raw data is not allowed
 - Similar to how we could **embed** iframes, but not read or write to their DOM
- Consequence: **attacker.com** can embed user's real avatar from **target.com**:

```
<h1>Welcome to your account!</h1>
<img src='https://target.com/avatar.png' />
```

Solution: SameSite cookies

- Use **SameSite** cookie attribute to prevent cookie from being sent with requests initiated by other sites

From **target.com**:

```
GET /avatar.png HTTP/1.1
```

```
Origin: target.com
```

```
Cookie: sessionId=1234
```

From **attacker.com**:

```
GET /avatar.png HTTP/1.1
```

```
Origin: attacker.com
```

Solution: Origin header

- Inspect the **Origin** HTTP header
- Reject any requests from origins not on an "allowlist"
- One gotcha: Watch out for HTTP caches!



Client

Server

Client

GET /avatar.png HTTP/1.1
Origin: target.com



Server

Client

GET /avatar.png HTTP/1.1
Origin: target.com

Server

Origin allowed?

Client

GET /avatar.png HTTP/1.1
Origin: target.com

Server

Origin allowed?

OK!

Client

GET /avatar.png HTTP/1.1
Origin: target.com



Origin allowed?

OK!

HTTP/1.1 200 OK
Cache-Control: public, max-age=31536000



Server

Client

GET /avatar.png HTTP/1.1
Origin: target.com



Origin allowed?

OK!

HTTP/1.1 200 OK
Cache-Control: public, max-age=31536000



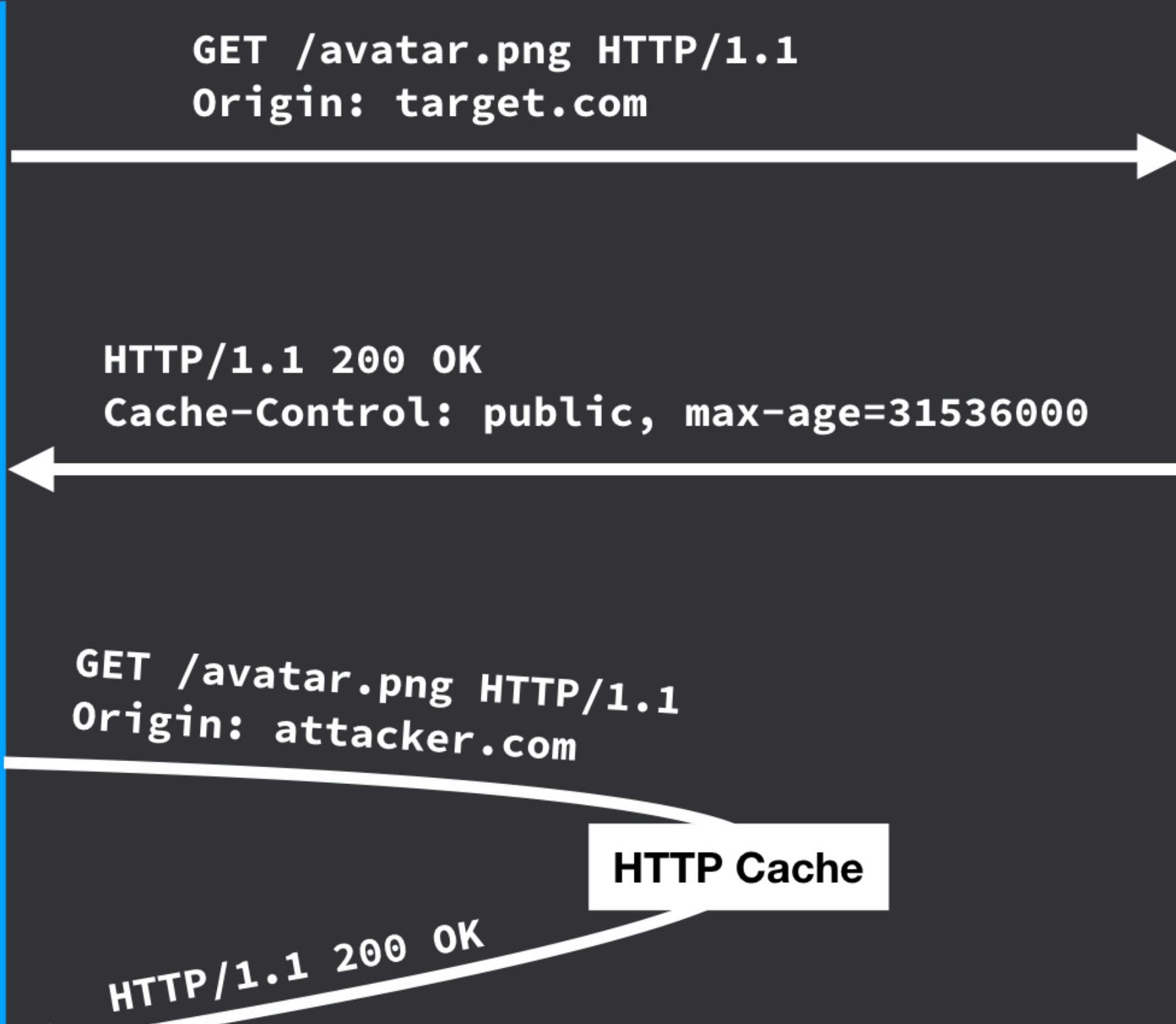
Server

GET /avatar.png HTTP/1.1
Origin: attacker.com

?

Client

GET /avatar.png HTTP/1.1
Origin: target.com



Server

Origin allowed?

OK!

Solution: Origin header

- Inspect the **Origin** HTTP header
- Reject any requests from origins not on an "allowlist"
- One gotcha: Watch out for HTTP caches!
 - Add a **Vary: Origin** header
 - Or, add a **Cache-Control: no-store** header

Same origin policy exceptions + ambient authority

- Remember: Forms are allowed to post to another origin!

```
<form method='POST' action='http://localhost:4000/transfer'>
  <input name='amount' value='100' />
  <input name='to' value='alice' />
  <input type='submit' value='Send' />
</form>
<script>
  document.forms[0].submit()
</script>
```

Cookies don't obey Same Origin Policy

- Cookies were created before Same Origin Policy so have different security model
- Cookies are **more specific** than Same Origin Policy
 - **Path** is ineffective because same origin pages can access each other's DOMs
- Cookies are **less specific** than Same Origin Policy
 - Different origins can mess with each others cookies (e.g. **attacker.stanford.edu** can set cookies for **stanford.edu**)
 - This is why Stanford login is **login.stanford.edu** and not **stanford.edu/login**

Cookies + "legitimate" DNS hijacking

The screenshot shows a search results page from [lookup.t-mobile.com](http://lookup.t-mobile.com/index.php?origURL=http%3A//questionablecontent.net/&r=&bc=). The page features the T-Mobile logo and a search bar. The search term is "QuestionableContent". The results are categorized under "Web Results".

Related Searches:

- [Accstation](#)
- [Qc](#)
- [All Comic Book](#)
- [X Man](#)
- [Marvel Comic](#)
- [Marvels](#)
- [C.c Cosplay](#)
- [Comic Cartoon](#)
- [Comic Book Price Guide](#)
- [The Batman](#)
- [Find Answers Fast](#)
- [Search for Information Here Look Up Quick Results Now!](#)
- [wow.com/Fast-Answers](#)
- [EVINE Live - Once ShopHQ](#)
- [Huge Selection of Jewelry, Watches, Apparel, Beauty, and Home Decor.](#)
- [www.evine.com](#)
- [Compare Prices](#)
- [Now up to 75% off Compare prices and save up to 75%](#)
- [Compare.salebounty.com](#)
- [Q-C - Cheap Prices](#)
- [See Hot Bargains for Q-C! Update Your Home for Less.](#)
- [Sponsored by: www.NexTag.com/Home-and-Garden](#)
- [Questionable Content](#)
- [Centers around an average frustrated 20-something music nerd, his PC and Faye. Includes archive, FAQ and overview.](#)
- [questionablecontent.net](#)
- [Questionable Content - definition of Questionable Content by ...](#)
- [By its very nature, it is open sourced -- meaning any one can edit its contents, providing questionable content that is then taken to be the definitive information ...](#)
- [www.thefreedictionary.com/Questionable+Content](#)
- [Questionable Content - Questionable Content Wiki](#)
- [Overview Edit. Questionable Content is a slice-of-life webcomic popular for its combination of believable and engaging characters, flights of fancy, and banter. It is ...](#)
- [questionablecontent.wikia.com/wiki/Questionable_Content](#)

Cookies + "legitimate" DNS hijacking

- If advertising page wants, it can steal cookies
 - **nonexistent.example.com** is different origin than **example.com**, yet can access cookies
- If advertising page contains a malicious third-party script, the script can steal cookies
- If advertising page contains a cross-site scripting issue (but **example.com** doesn't), then **anyone** can steal cookies
 - Attacker causes user to visit **nonexistent.example.com/<some-attack-code>**
 - DNS is hijacked by advertising page which includes **<some-attack-code>** in page
 - As before, **nonexistent.example.com** can access **example.com** cookies, even though it's another origin

What is allowed?

- Is site A allowed to **link to** site B? **Yes!**
- Is site A allowed to **embed** site B? **Yes!**
- Is site A allowed to **embed** site B and **modify** its contents? **No!**
- Is site A allowed to **submit a form** to site B? **Yes!**
- Is site A allowed to **embed images** from site B? **Yes!**
- Is site A allowed to **embed scripts** from site B? **Yes!**
- Is site A allowed to **read data** from site B? **No!**

END

Credits: Michal Zalewski. "The Tangled Web."