CS 253: Web Security

Same Origin Policy

Admin

- Assignment 0 is due Wednesday, September 29 at 5:00pm

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
  - If there's a bug in the browser itself then all these rules go out the window (just like in an OS)

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?

- Where does one **document** begin and another end?
- How much **interaction** should be allowed between non-cooperating origins?
- Which **actions** should be subject to security checks?

Demo: Same origin policy

Demo: Same origin policy

From **https://web.stanford.edu/class/cs253/:**

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

// Same origin, so everything is allowed
iframe.contentDocument.body.style.backgroundColor = 'red'

// Even this!
let i = 0
const imgs = iframe.contentDocument.body.querySelectorAll('img')
setInterval(() => {
  imgs.forEach(img => img.style.transform = `rotate(${i}deg)`)
  i += 1
}, 50)
```

Demo: Same origin policy (pt 2)

From **https://web.stanford.edu/class/cs253/:**

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

// Is this allowed?
document.body.append(iframe) // Allowed!

// Is this allowed?
iframe.contentDocument.body.style.backgroundColor = 'red' // Not allowed!

// Navigate to Dan's homepage, then...

// Is this 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'
```

Any subdomain can join the `document.domain` party!

- So, if **axess.stanford.edu** and **attacker.stanford.edu** both run:

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

- Then **attacker.stanford.edu** can access content on **axess.stanford.edu**! Bad!

Demo:

`document.domain` on crypto.stanford.edu

Demo: `document.domain` on crypto.stanford.edu

From **https://crypto.stanford.edu:**

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

From **https://web.stanford.edu/class/cs253/:**

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

// Is this allowed?
iframe.contentDocument.body.style.backgroundColor = 'red' // Allowed!
```

`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** can NOT access content on **stanford.edu**!
- **stanford.edu** would also need to run the same code to opt-in to this behavior:

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

- The above 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

site-a.com:

```
<h1>Parent: http://site-a.com:8000</h1>
<input name='val' />
<br /><br />
<iframe src='http://site-b.com:8001' width='100%' height='400px'></iframe>
<script>
  const input = document.querySelector('input')
  const iframe = document.querySelector('iframe')
  input.addEventListener('input', () => {
    iframe.src = `http://site-b.com:8001#${encodeURIComponent(input.value)}`
  })
</script>
```

site-b.com:

```
<h1>Child: http://site-b.com:8001</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

site-a.com:

```
<h1>Parent: http://site-a.com:8000</h1>
<input name='val' />
<br /><br />
<iframe src='http://site-b.com:8001' width='100%' height='400px'></iframe>
<script>
  const input = document.querySelector('input')
  const iframe = document.querySelector('iframe')
  input.addEventListener('input', () => {
    iframe.contentWindow.postMessage(input.value, 'http://site-b.com:8001')
  })
</script>
```

site-b.com:

```
<h1>Child: http://site-b.com:8001</h1>
<div></div>
<script>
  const div = document.querySelector('div')
  window.addEventListener('message', event => {
    if (event.origin !== 'http://site-a.com:8000') 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)
})
```

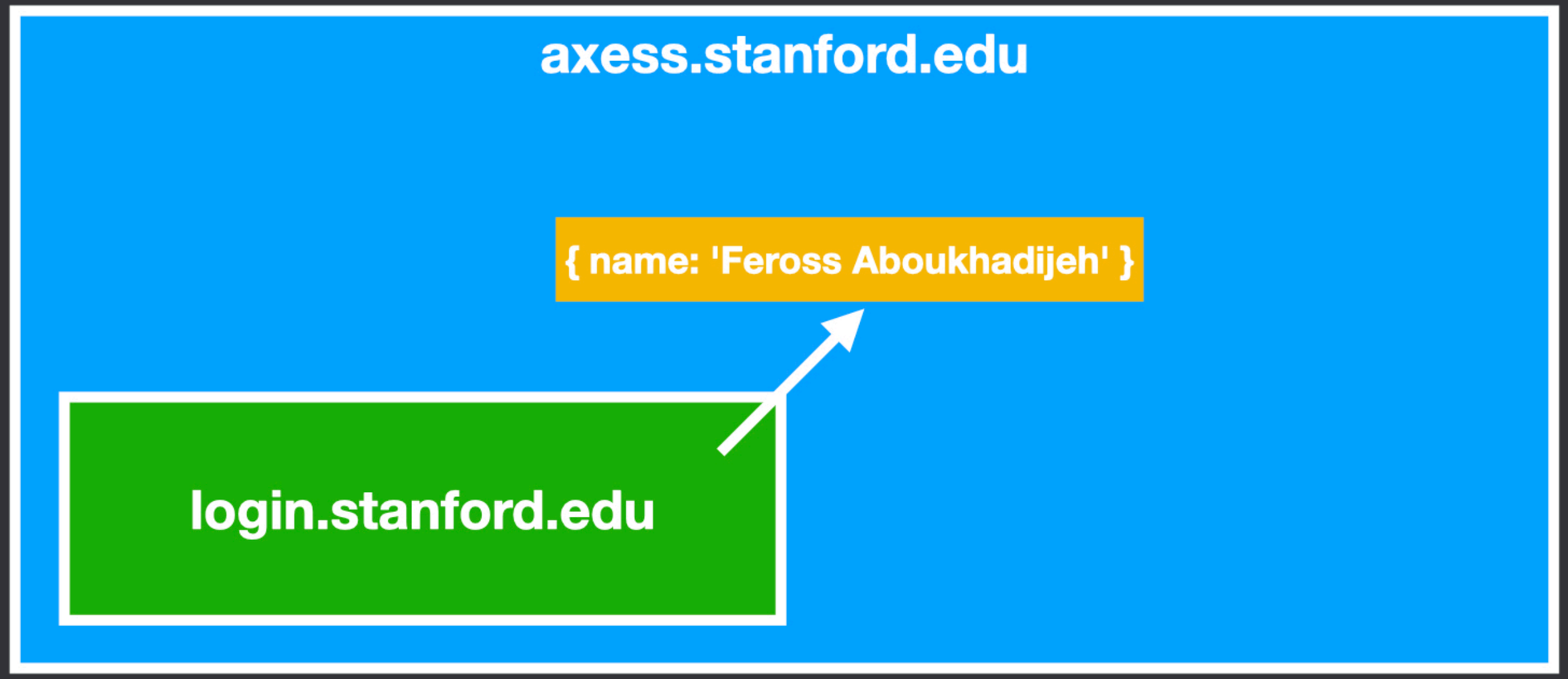
- Then it embeds an iframe to **login.stanford.edu** which runs:

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

- This is insecure! Why?

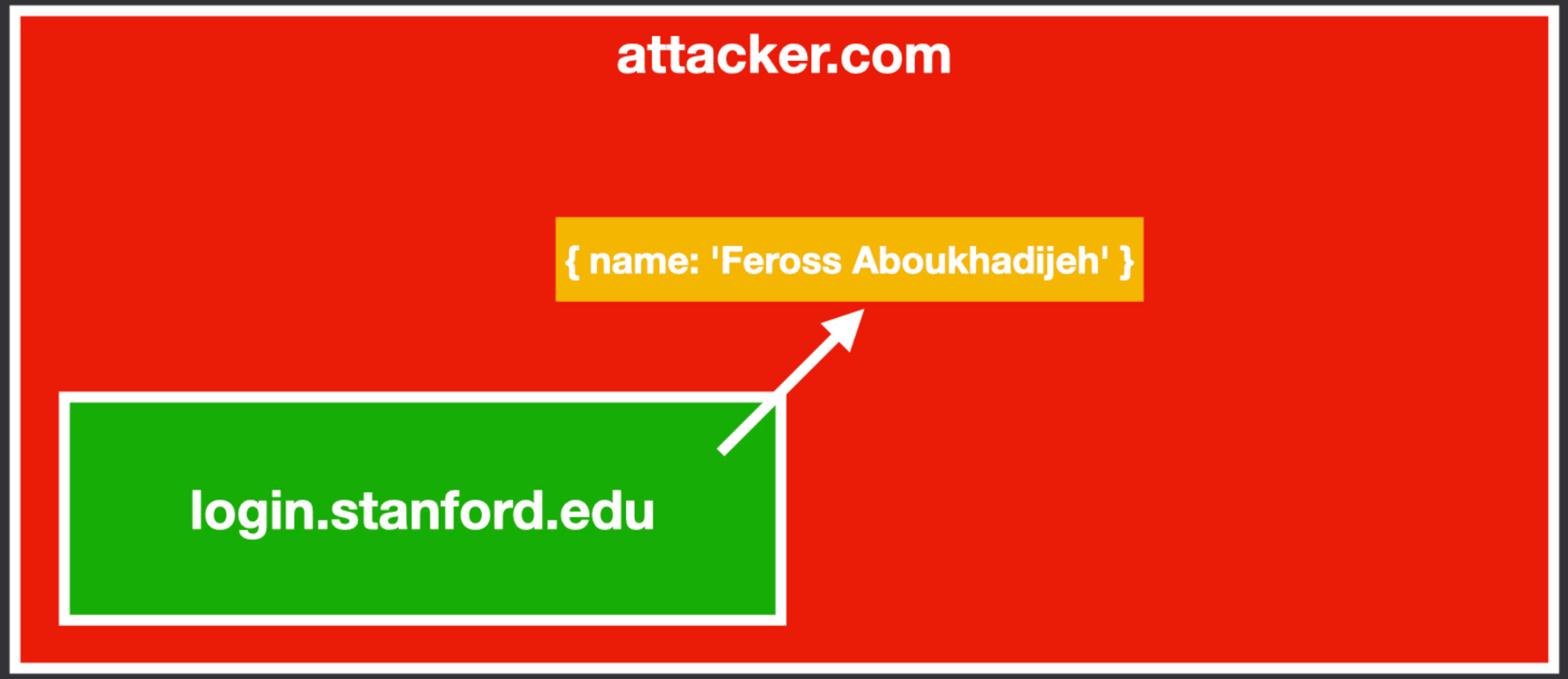
axess.stanford.edu


axess.stanford.edu
login.stanford.edu


axess.stanford.edu
{ name: 'Feross Aboukhadijeh' }
login.stanford.edu


attacker.com


attacker.com
login.stanford.edu

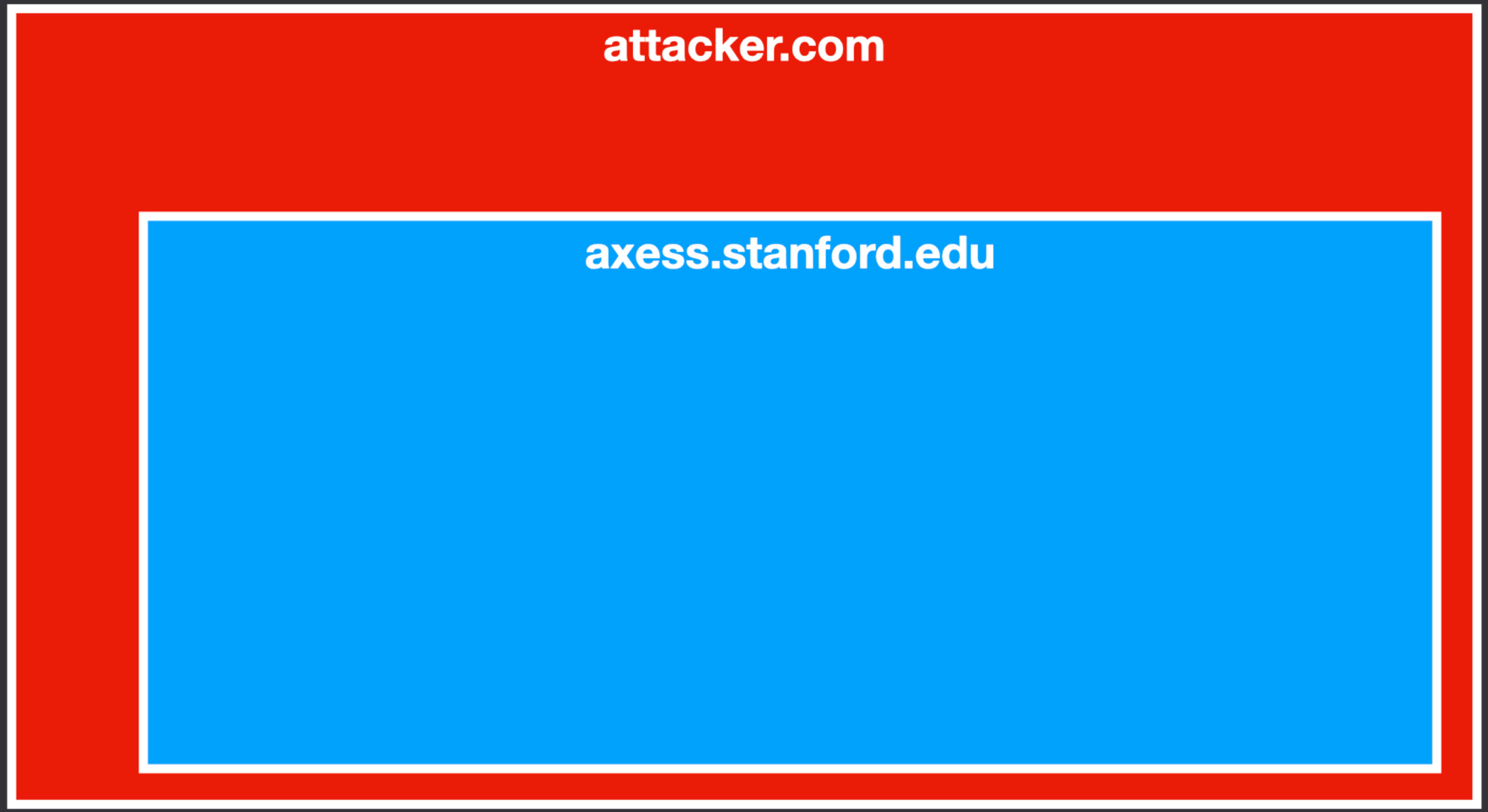

attacker.com
{ name: 'Feross Aboukhadijeh' }
login.stanford.edu

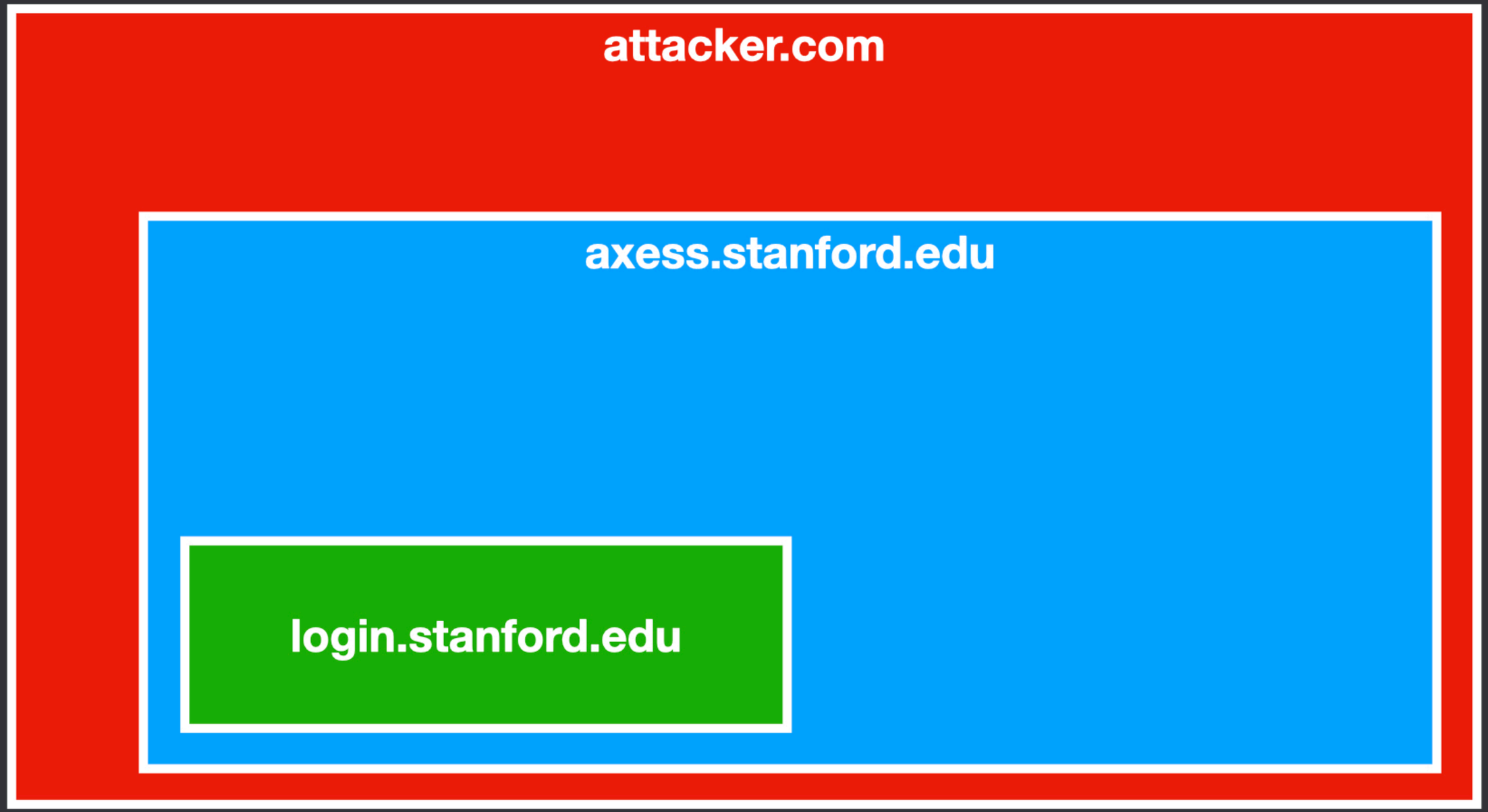
Need to validate destination of messages!

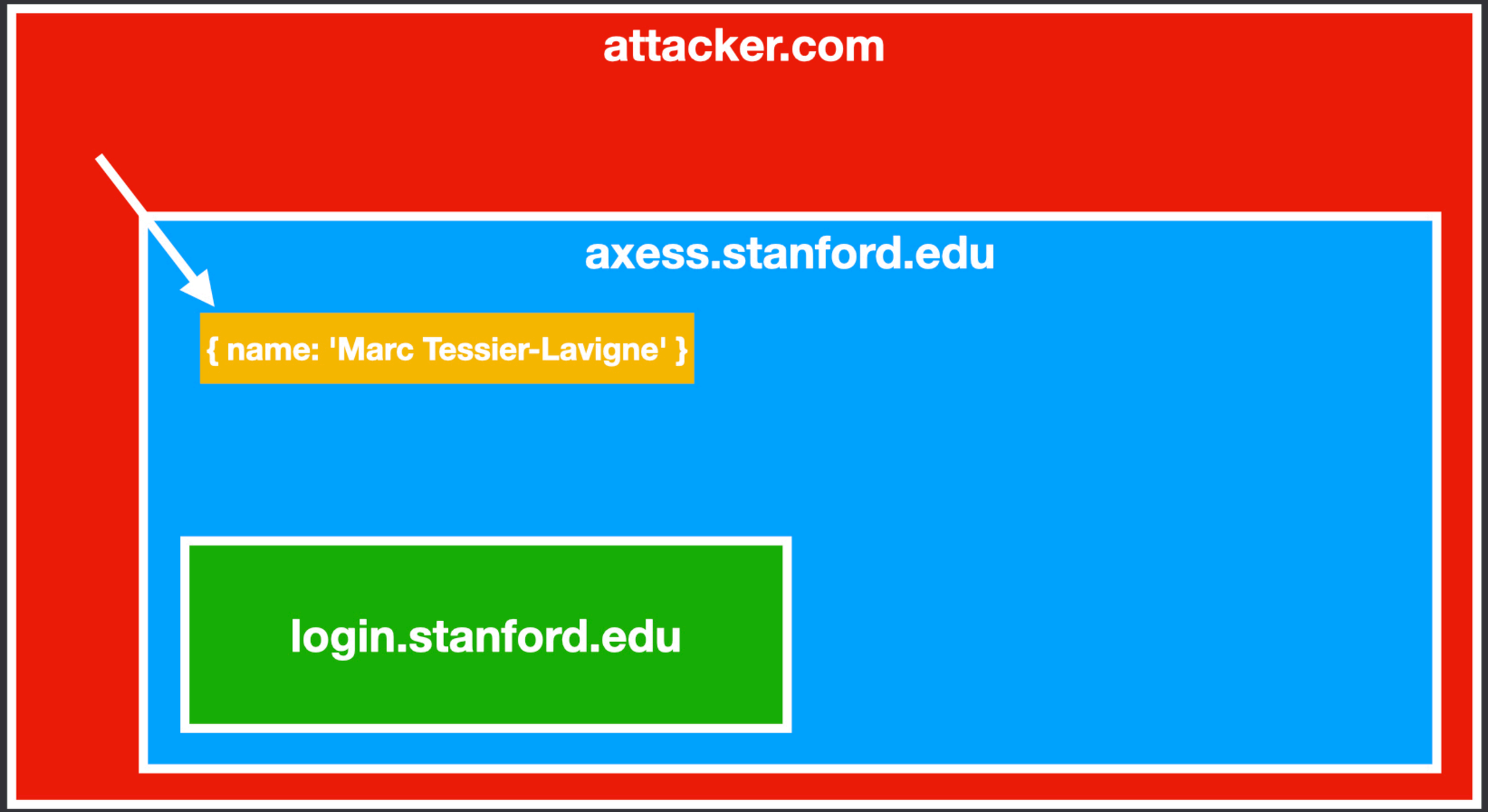
- If an attacker embeds **login.stanford.edu**, they can listen to it's message which reveals the name of the logged in user!
- Solution: **login.stanford.edu** should specify intended recipient origin. Browser will enforce this.

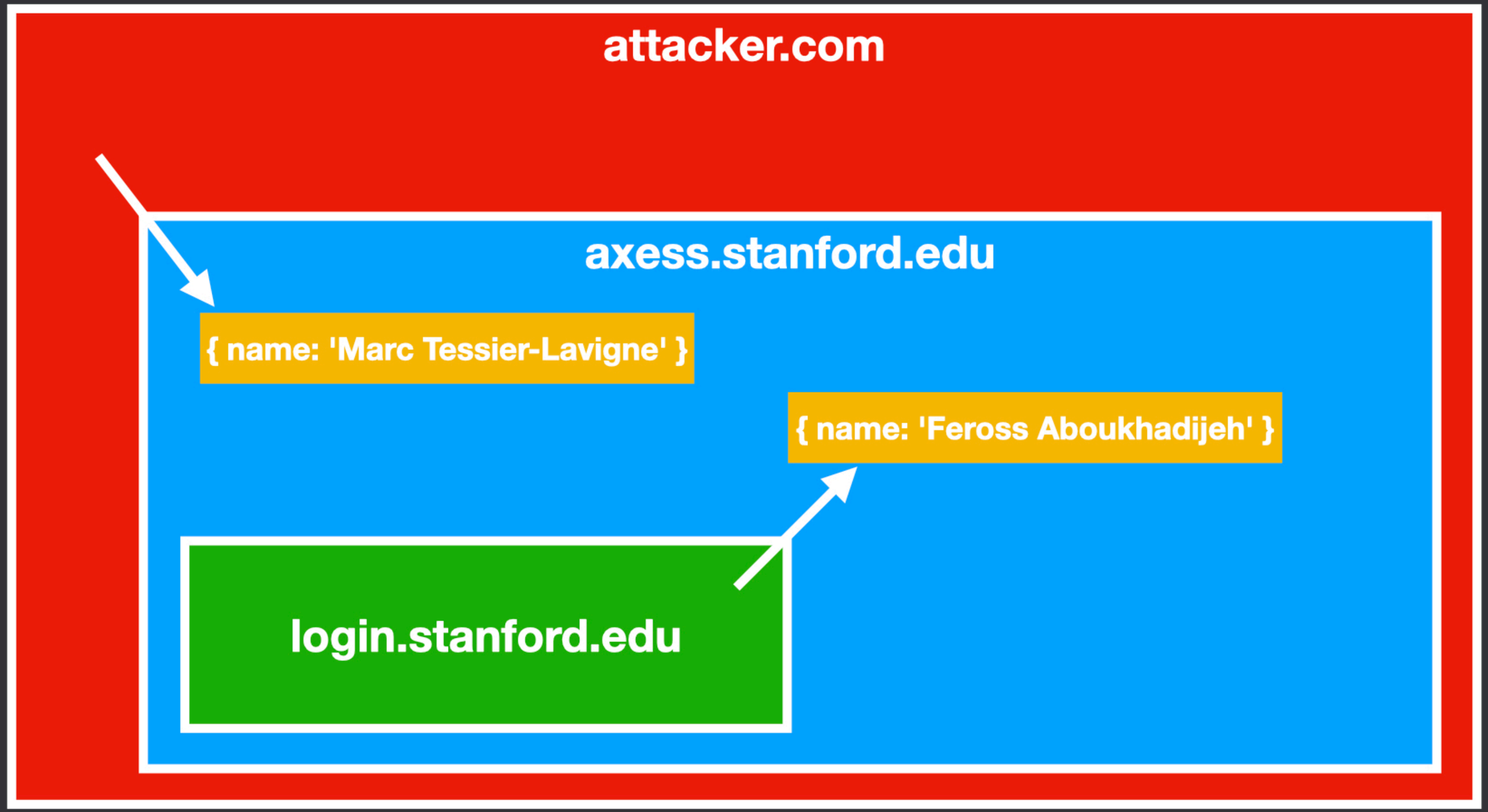
```
const data = { name: 'Feross Aboukhadijeh' }
window.parent.postMessage(data, 'https://axess.stanford.edu')
```

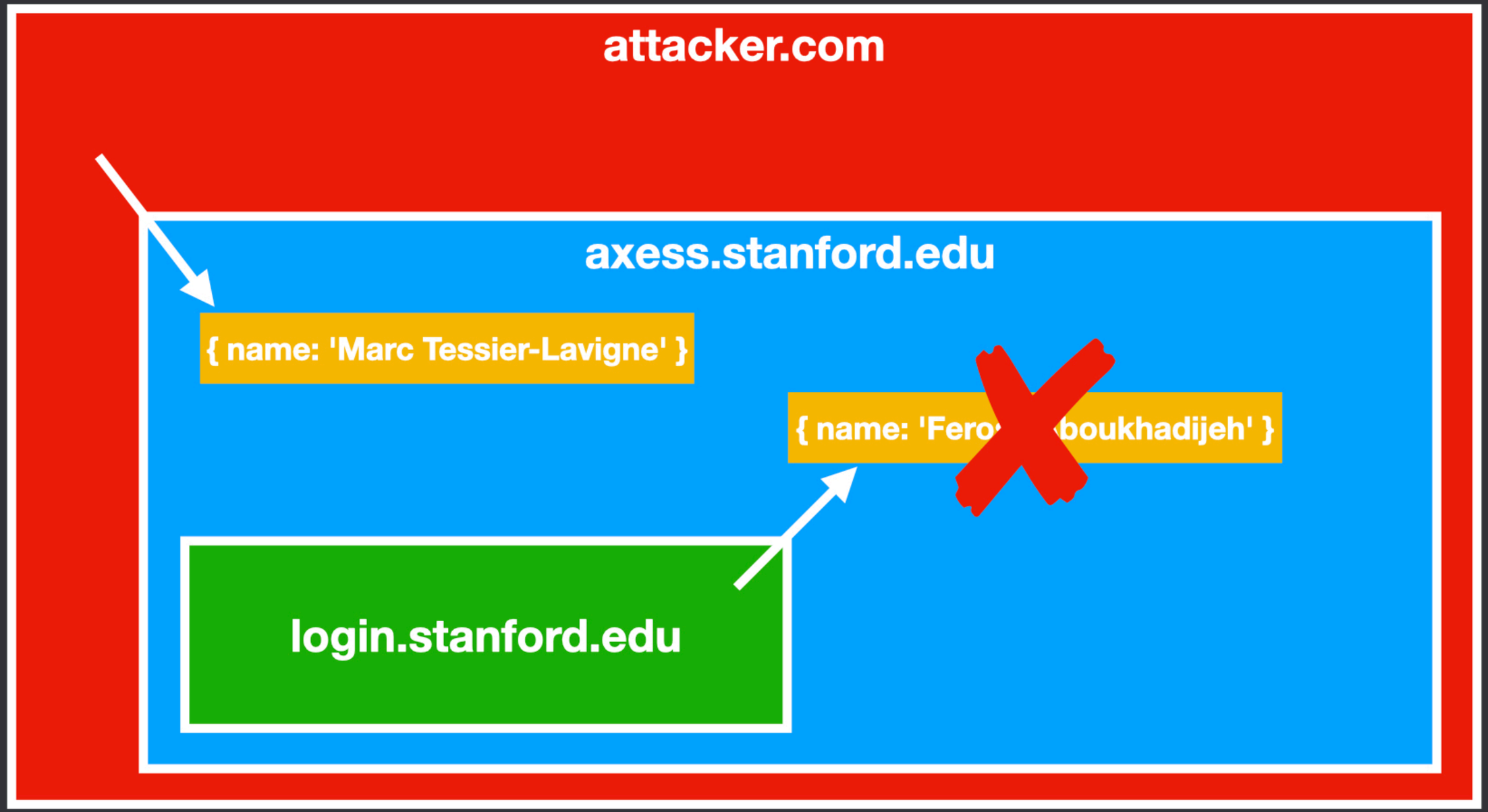
attacker.com


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' }
{ name: 'Fero boukhadijeh' }
login.stanford.edu

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`

- **Sender** must specify origin which is permitted to receive message
  - In case the URL of the target window has changed
- **Recipient** must validate the identity of the sender
  - In case some other window is sending the message
- ***Remember:*** *Always specify intended recipient or expected sender!*

Same origin policy exceptions

- **Summary:** 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 - automatic 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?

END