CSE227 – Graduate Computer Security

Web Fundamentals

UC San Diego

Housekeeping

General course things to know

- Course projects
  - I will provide some initial thoughts and feedback on each of your project ideas by **this Friday** via email — really enjoyed them so far
  - Start meeting with your teams, ideating, and reaching out to me if you have things you want to chat about!

Today's lecture

Learning Objectives

- Talk about the web, understand its fundamentals, and the ways in which the design of the web makes security hard
- Discuss the CSRF paper
- Discuss the HTML sanitization paper

Preliminaries

Polling the room

- How many people have built a website before?
- How many people have built a web app before?
- How many people have *deployed* a web app before?
  - Where?

What is the web?

What is the web?

Information system that runs on the Internet that allows *documents* to be connected to other *documents*, increasingly enabled through *scripting* and *server-side logic*

Web Fundamentals

- What is a web server?

Web Fundamentals

- What is a web server?
- What is a web client?

Web Fundamentals

- What is a web server?
- What is a web client?
- What are some examples of web clients?

Web Fundamentals

- What is a web server?
- What is a web client?
- What are some examples of web clients?
- What is an HTTP request?

Web Fundamentals

- What is a web server?
- What is a web client?
- What are some examples of web clients?
- What is an HTTP request?
- What is the client-server architecture?

Web fundamentals 2

- How do websites keep track of if you've logged in already?

Web fundamentals 2

- How do websites keep track of if you've logged in already?
- When are cookies set? Who sets the cookies?

Web fundamentals 2

- How do websites keep track of if you've logged in already?
- When are cookies set? Who sets the cookies?
- When are cookies sent? Who sends the cookies?

Interfacing with the Web

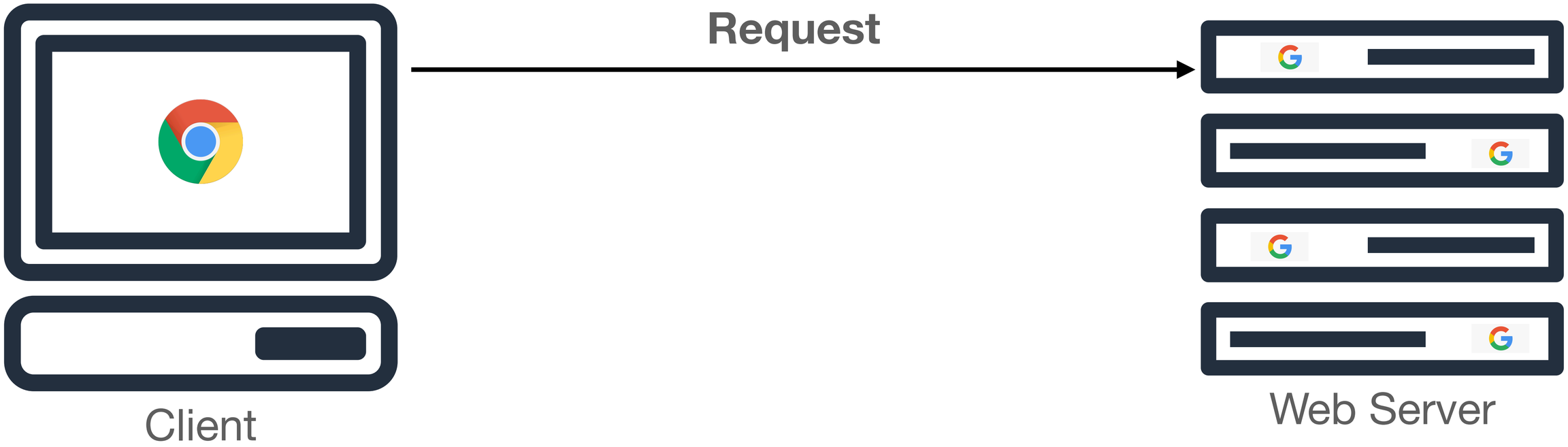
Client / Server Model

Client

Web Server

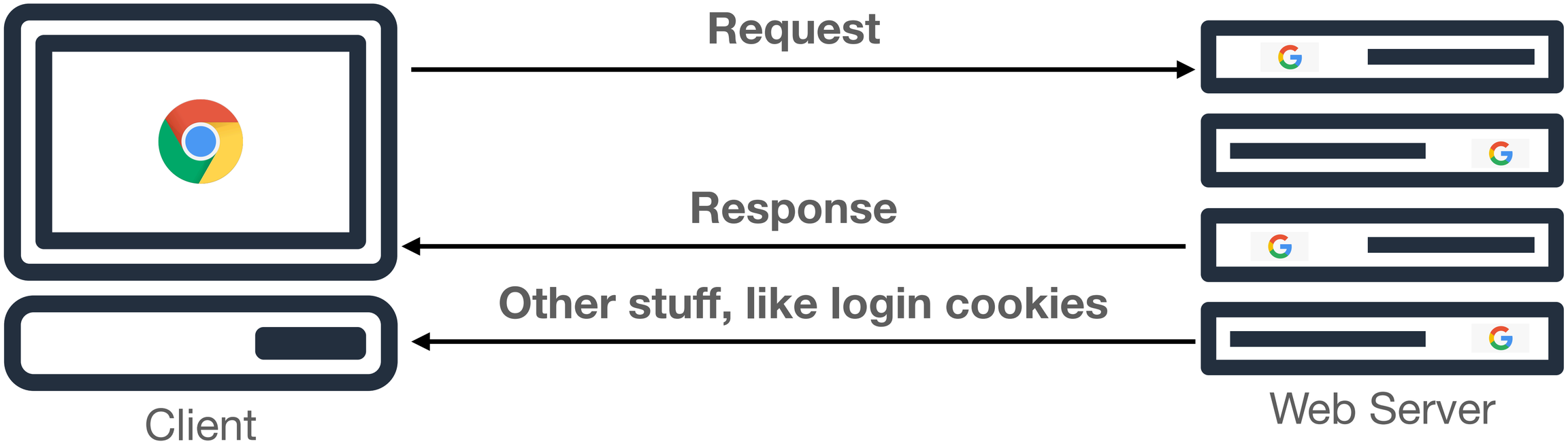
Interfacing with the Web

Client / Server Model



Interfacing with the Web

Client / Server Model



Robust Defenses for Cross-Site Request Forgery

What is Cross-Site Request Forgery?

What is Cross-Site Request Forgery?

“CSRF is an attack that forces an end user to execute unwanted actions on a web application in which they’re currently authenticated.” – OWASP

Cross-Site Request Forgery

- Recall: Browsers send cookies with requests all the time. How?

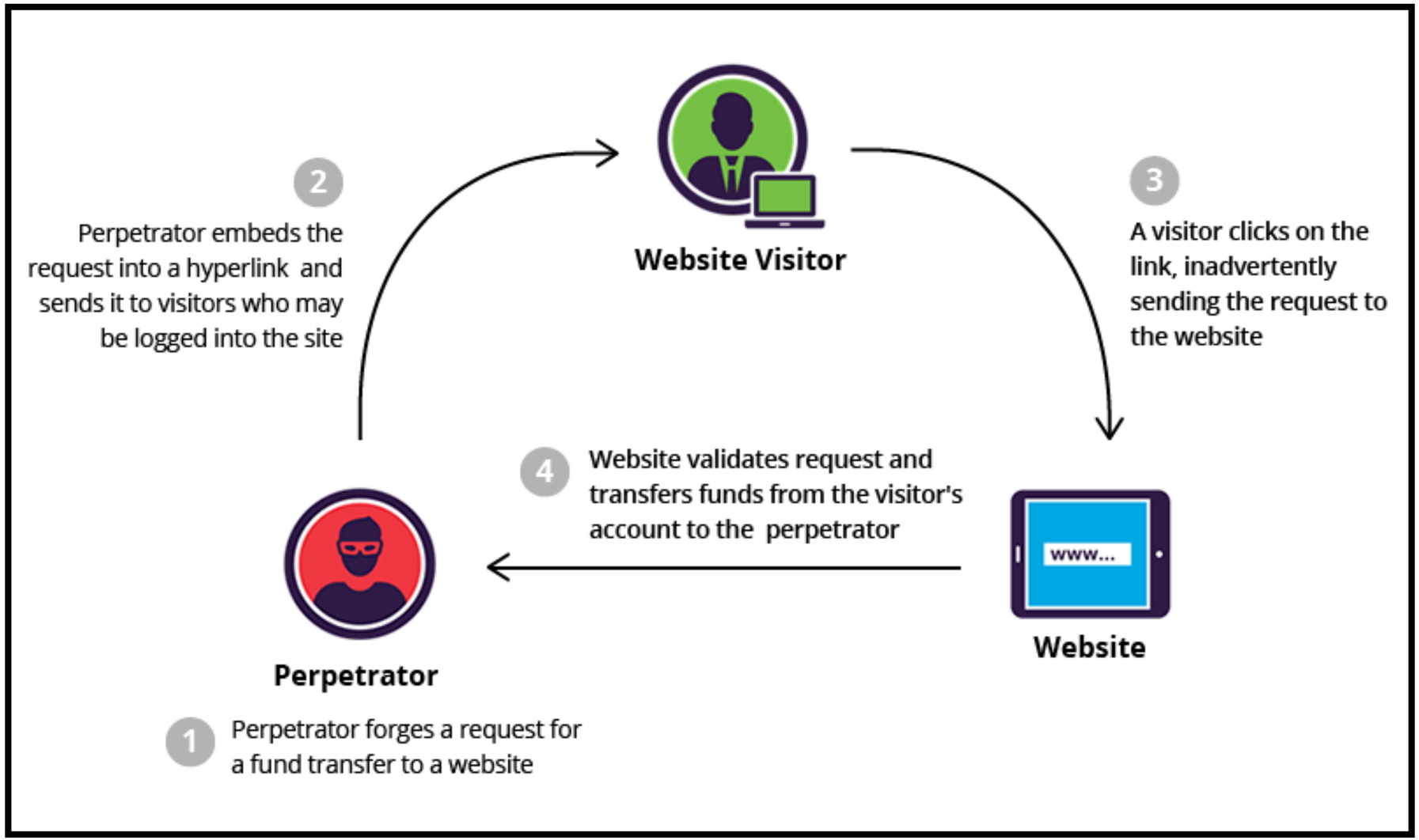
Cross-Site Request Forgery

- Recall: Browsers send cookies with requests all the time. How?
  - If a user clicked a link (on a website, or even in email)
  - If another page embedded the target page in an iframe
  - If a client-side script issued the request
- Doesn't matter where the request comes from, only thing that matters is the ***target*** of the request
  - Where might there be a problem?

Cross-Site Request Forgery

- Recall: Browsers send cookies with requests all the time. How?
  - If a user clicked a link (on a website, or even in email)
  - If another page embedded the target page in an iframe
  - If a client-side script issued the request
- Doesn't matter where the request comes from, only thing that matters is the ***target*** of the request
  - Where might there be a problem?
- Target doesn't know if the request was **intended** or **authorized** by the user

What is Cross-Site Request Forgery?



Imperva

Wait, how the heck is CSRF allowed?!

- Websites are ***allowed*** to send arbitrary HTTP requests to any other website by default. **Why?**

Wait, how the heck is CSRF allowed?!

- Websites are ***allowed*** to send arbitrary HTTP requests to any other website by default. **Why?**
- What is the Same-Origin Policy?

Wait, how the heck is CSRF allowed?!

- Websites are ***allowed*** to send arbitrary HTTP requests to any other website by default. **Why?**
- What is the Same-Origin Policy?
  - Restricts the **reading** of content from different *origins*, but sites can still POST data

Weird Web Carveouts

- Can a website read an image from another website?
- Can a website read a script from another website?
- Can a website load another website?
- Can a website load *content* from another website?

Typical Authentication Pattern

chase.com

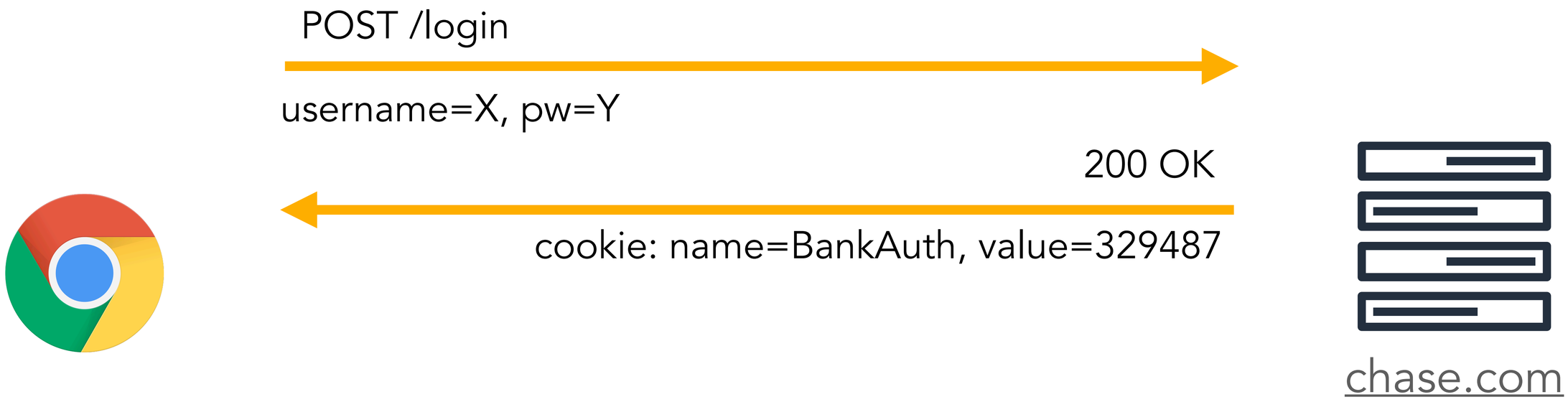
Typical Authentication Pattern

POST /login

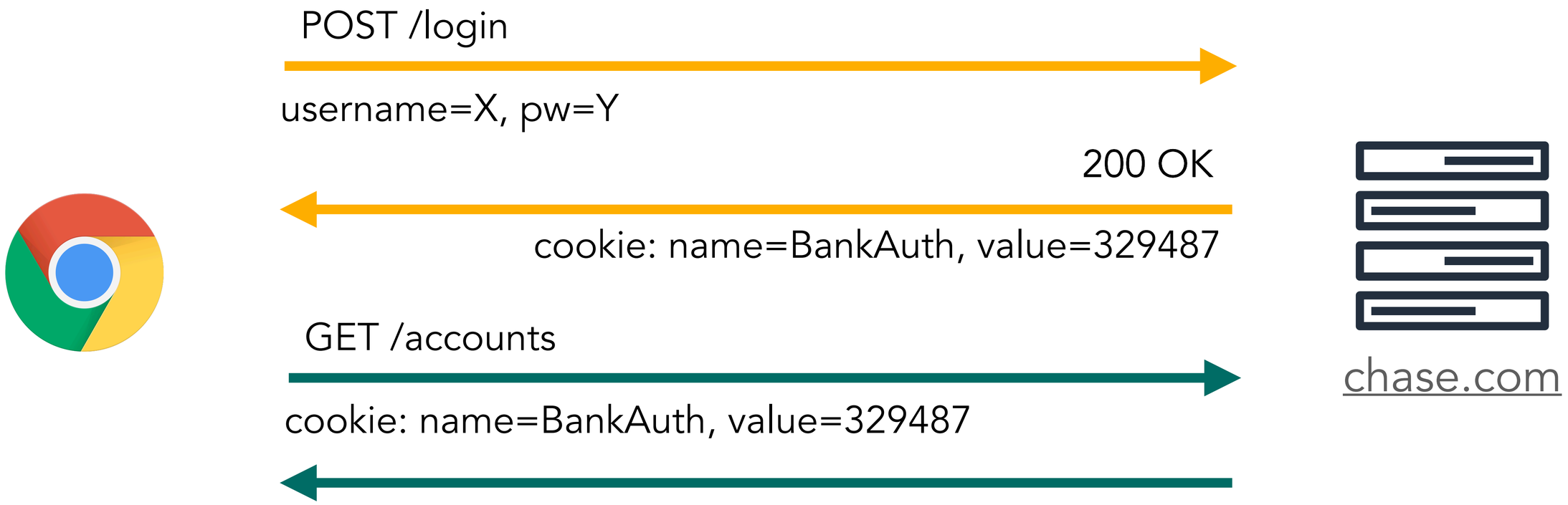
username=X, pw=Y

chase.com

Typical Authentication Pattern



Typical Authentication Pattern



Typical Authentication Pattern

POST /login

username=X, pw=Y

200 OK

cookie: name=BankAuth, value=329487

chase.com

GET /accounts

cookie: name=BankAuth, value=329487

200 OK

POST /transfer

cookie: name=BankAuth, value=329487

200 OK

CSRF Scenario

- User is signed into chase.com
  - Cookie **remains** in the browser's state
- User then visits a malicious website, containing the following:

```
<form name=BillPayForm action="https://chase.com/transfer">
<input name=recipient value=badguy>
<input amount=10000000>
<script> document.BillPayForm.submit(); </script>
```

What does the code above do?

CSRF Scenario

- User is signed into chase.com
  - Cookie **remains** in the browser's state
- User then visits a malicious website, containing the following:

```
<form name=BillPayForm action="https://chase.com/transfer">
<input name=recipient value=badguy>
<input amount=10000000>
<script> document.BillPayForm.submit(); </script>
```

Will the attacker be able to read the HTTP response?

CSRF Scenario

- User is signed into chase.com
  - Cookie **remains** in the browser's state
- User then visits a malicious website, containing the following:

```
<form name=BillPayForm action="https://chase.com/transfer">
<input name=recipient value=badguy>
<input amount=10000000>
<script> document.BillPayForm.submit(); </script>
```

- Code executes an HTTP Post to chase.com
  - Good news, attacker.com can't see the result of POST request thanks to SOP💪

CSRF Scenario

- User is signed into chase.com
  - Cookie **remains** in the browser's state
- User then visits a malicious website, containing the following:

```
<form name=BillPayForm action="https://chase.com/transfer">
<input name=recipient value=badguy>
<input amount=10000000>
<script> document.BillPayForm.submit(); </script>
```

- Code executes an HTTP Post to chase.com
  - Good news, attacker.com can't see the result of POST request thanks to SOP💪
  - Bad news, all your money is gone! 😭

CSRF Patterns

Currently logged into chase.com

evil.com

chase.com

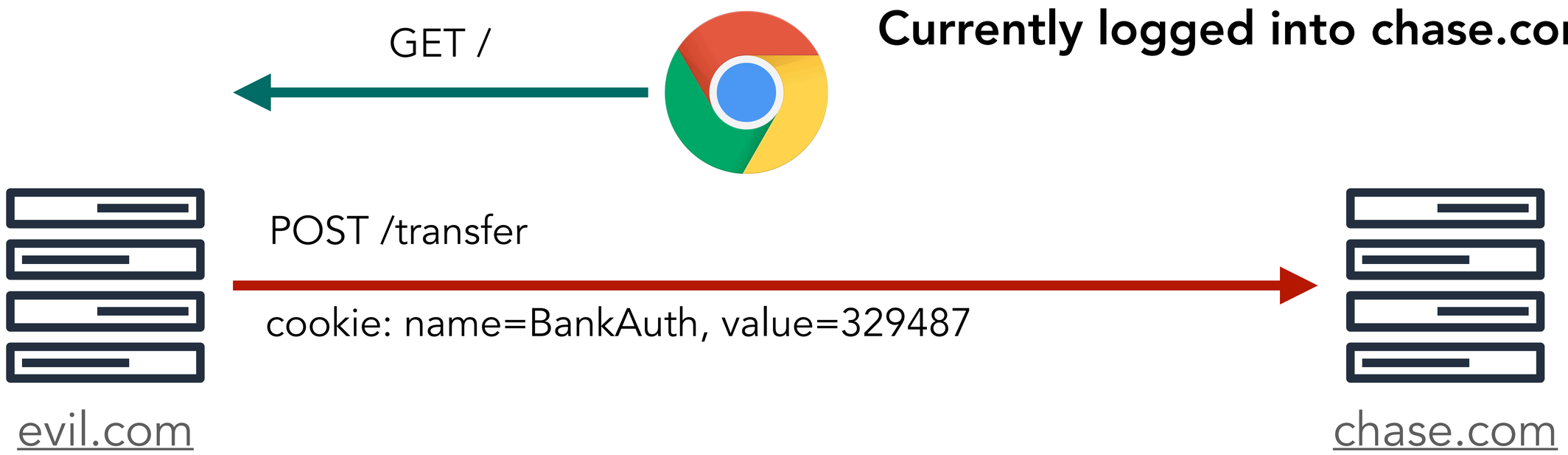
CSRF Patterns

Currently logged into chase.com

evil.com

chase.com

CSRF Patterns



Common CSRF defenses

- What is a CSRF token? How does it work?
- What is the `Referer` header? How does it work?
- What is an XMLHttpRequest and how does the CSRF defense work?

Common CSRF defense fails

- What's wrong with CSRF tokens?
- What's wrong with the `Referer` header?
- What's wrong with the XMLHttpRequest strategy?

Login CSRF

- What is login CSRF?
  - Attacker signs in *as themselves*, unbeknownst to the user
- What can you do with login CSRF?
  - What is the “search history” attack?
  - What is the “malicious merchant” attack? (e.g., PayPal)

Login CSRF (Special Case)

Currently not logged into google.com

evil.com

google.com

Login CSRF (Special Case)

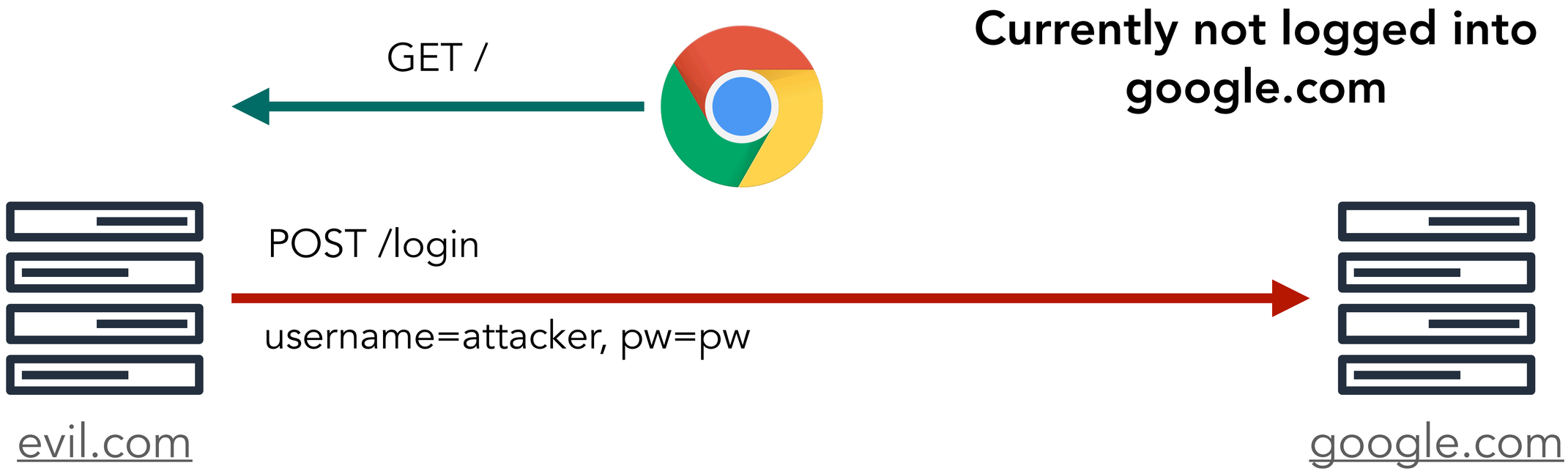
GET /

evil.com

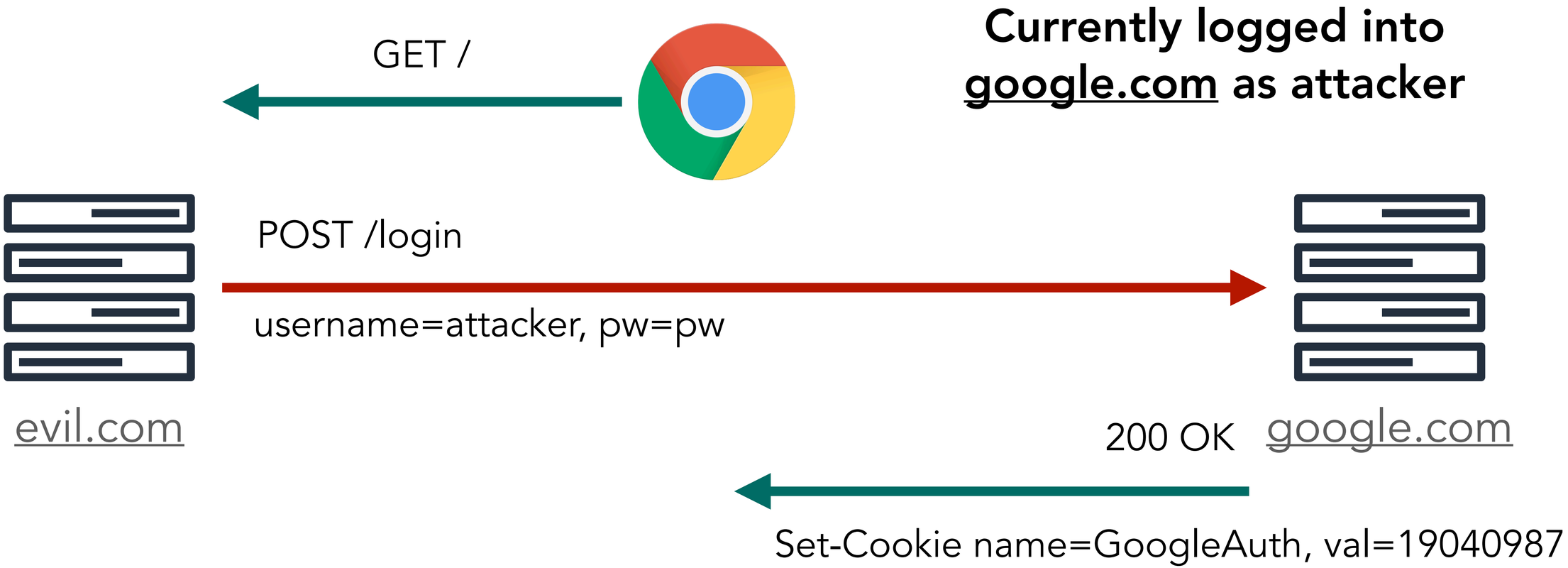
Currently not logged into google.com

google.com

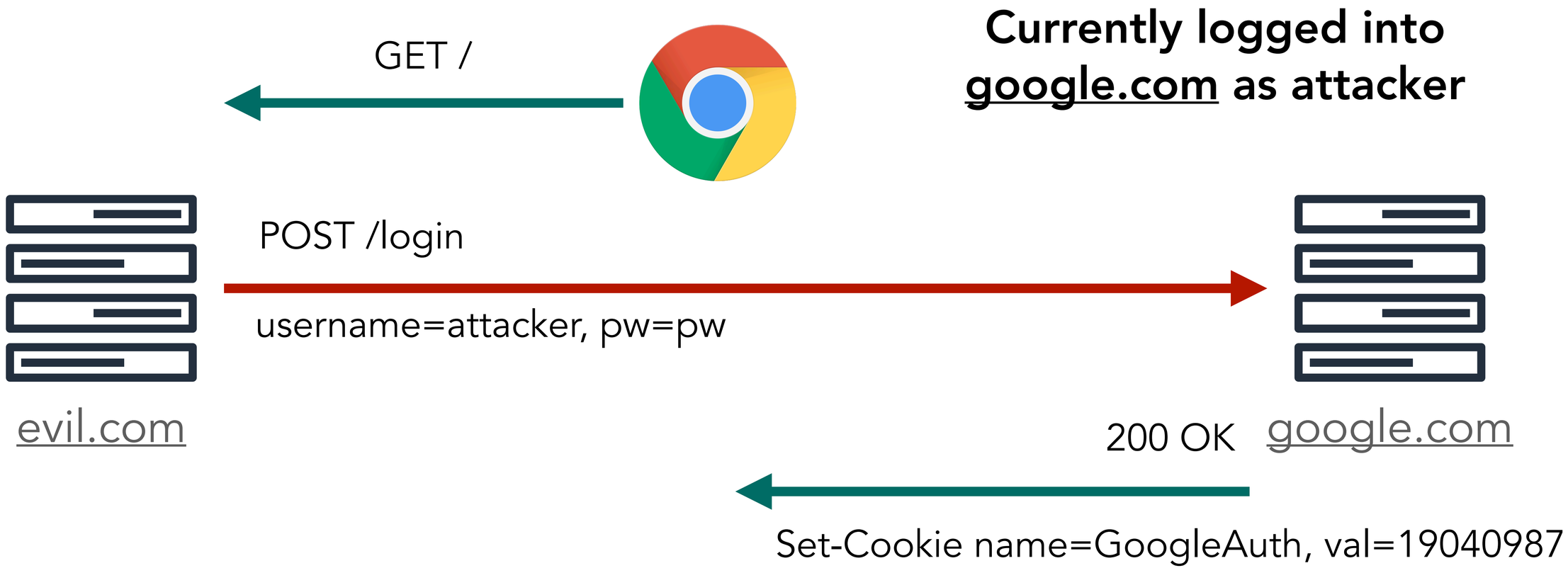
Login CSRF (Special Case)



Login CSRF (Special Case)

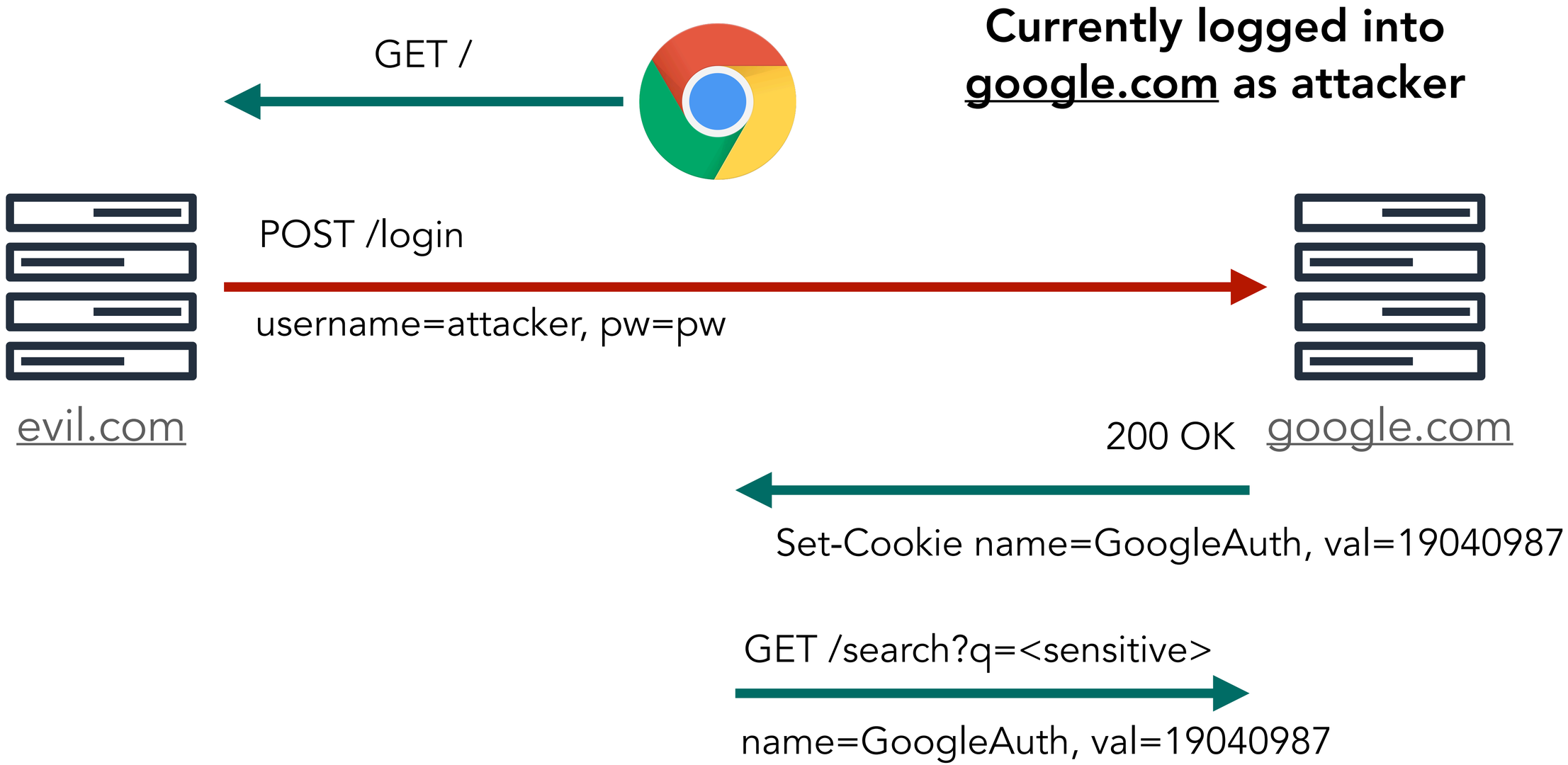


Login CSRF (Special Case)

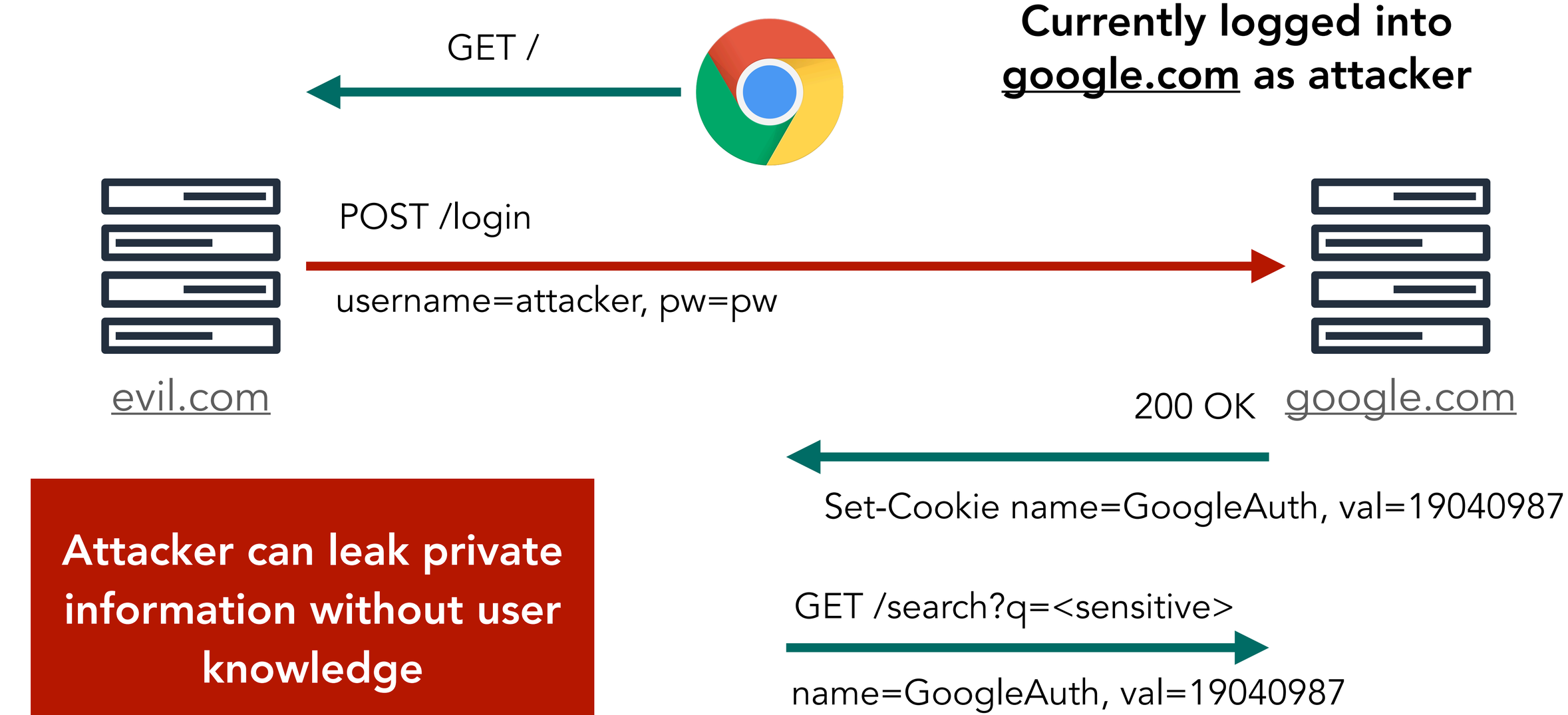


Why might an attacker want to do this?

Login CSRF (Special Case)



Login CSRF (Special Case)



Attacker can leak private information without user knowledge

Defeating CSRF with the Referer header

- By default (usually), when the browser makes an HTTP request, it contains the *Referer*, aka the URL of the webpage that is making the request
  - Validation of the Referer header could easily defend against CSRF attacks
- People decry Referers because of privacy concerns. What part of the Referer contains these privacy issues?

Defeating CSRF with the Referer header

- By default (usually), when the browser makes an HTTP request, it contains the *Referer*, aka the URL of the webpage that is making the request
  - Validation of the Referer header could easily defend against CSRF attacks
- People decry Referers because of privacy concerns. What part of the Referer contains these privacy issues?
- Why does validation with the Referer header **not** work all the time?

Defeating CSRF with the Referer header

- By default (usually), when the browser makes an HTTP request, it contains the *Referer*, aka the URL of the webpage that is making the request
  - Validation of the Referer header could easily defend against CSRF attacks
- People decry Referers because of privacy concerns. What part of the Referer contains these privacy issues?
- Why does validation with the Referer header **not** work all the time?
  - Fail-open: Allow requests where there is no Referer header
  - Fail-closed: Block requests where there is no Referer header

The Defense: Origin header

- What is the Origin header proposal in this paper?
  - Why does it help with the privacy concerns brought up before?
- What happens when the browser does not add an Origin header?
- Why do they think the Origin header will fix CSRF? Why do they think it'll be adopted?

The Defense: Origin header

Origin

Baseline Widely available

The HTTP `Origin` request header indicates the origin (scheme, hostname, and port) that *caused* the request. For example, if a user agent needs to request resources included in a page, or fetched by scripts that it executes, then the origin of the page may be included in the request.

Today's Defenses: SameSite Cookies

Set-Cookie

Baseline Widely available *

The HTTP `Set-Cookie` response header is used to send a cookie from the server to the user agent, so that the user agent can send it back to the server later. To send multiple cookies, multiple `Set-Cookie` headers should be sent in the same response.

`SameSite=<samesite-value>` Optional

Controls whether or not a cookie is sent with cross-site requests, providing some protection against cross-site request forgery attacks (CSRF).

SameSite Cookies

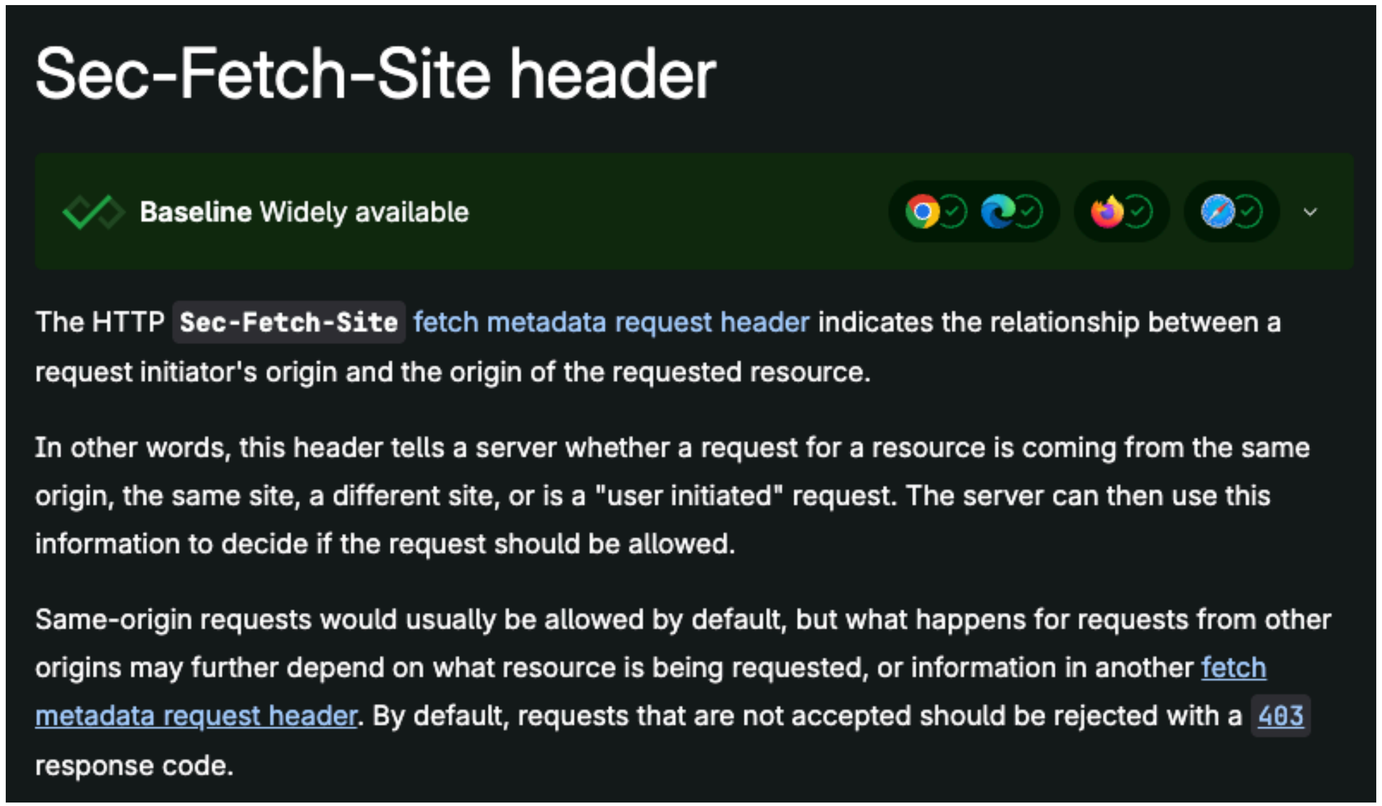
- Cookie option that prevents browser from sending a cookie along with cross-site requests
- **SameSite = Strict** Never send a cookie in a cross-site browsing context, even when following a regular link
- **SameSite = Lax** Session cookie is allowed when following a navigation link but blocks it in CSRF-prone request methods, like POST (default)
- **SameSite = None** Send cookies from any context
- Why might these strategies not always work?

SameSite Cookies

- Cookie option that prevents browser from sending a cookie along with cross-site requests
- **SameSite = Strict** Never send a cookie in a cross-site browsing context, even when following a regular link
- **SameSite = Lax** Session cookie is allowed when following a navigation link but blocks it in CSRF-prone request methods, like POST (default)
- **SameSite = None** Send cookies from any context
- Why might this not always work?
  - Server has to trust browser to implement correctly. And they might not.

CSRF Defenses Today

- Defense in depth — usually some combination of all three defenses
- New paradigm: Fetch metadata



https://developer.mozilla.org/en-US/docs/Web/HTTP/Reference/Headers/Sec-Fetch-Site

CSRF meta-questions

- How feasible is a CSRF attack? Will it work in practice?
- What software does the "Origin" proposal require you to *trust*?
  - Is this assumption always going to be true?
- How would you defend against a CSRF attack today? Is it that different from 2007, when this paper was written?
- What would you say is a **fundamental issue** that enables a CSRF attack?

CSRF meta-questions

- How feasible is a CSRF attack? Will it work in practice?
- What software does the "Origin" proposal require you to *trust*?
  - Is this assumption always going to be true?
- How would you defend against a CSRF attack today? Is it that different from 2007, when this paper was written?
- What would you say is a **fundamental issue** that enables a CSRF attack?
  - Side-effects in the interface between the web server and web browser
  - *Feature*, not a bug

Paper meta-questions

- What did we think about the paper?
  - You can comment on the organization, the writing, the experiments, etc.
- What do you think about the solution presented in the paper?
- Why do you think this paper was so successful?