CSE227 – Graduate Computer Security

Web Fundamentals

UC San Diego

Housekeeping

General course things to know

- Everyone should’ve received some initial feedback on course projects
- Overall vibe is good, lots of really exciting projects, but general advice
  - Figure out what success means to your team (the more concrete the better)
  - Be ambitious, but also acknowledge you only have now ~6 – 7 weeks (last week is presentations!)

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?

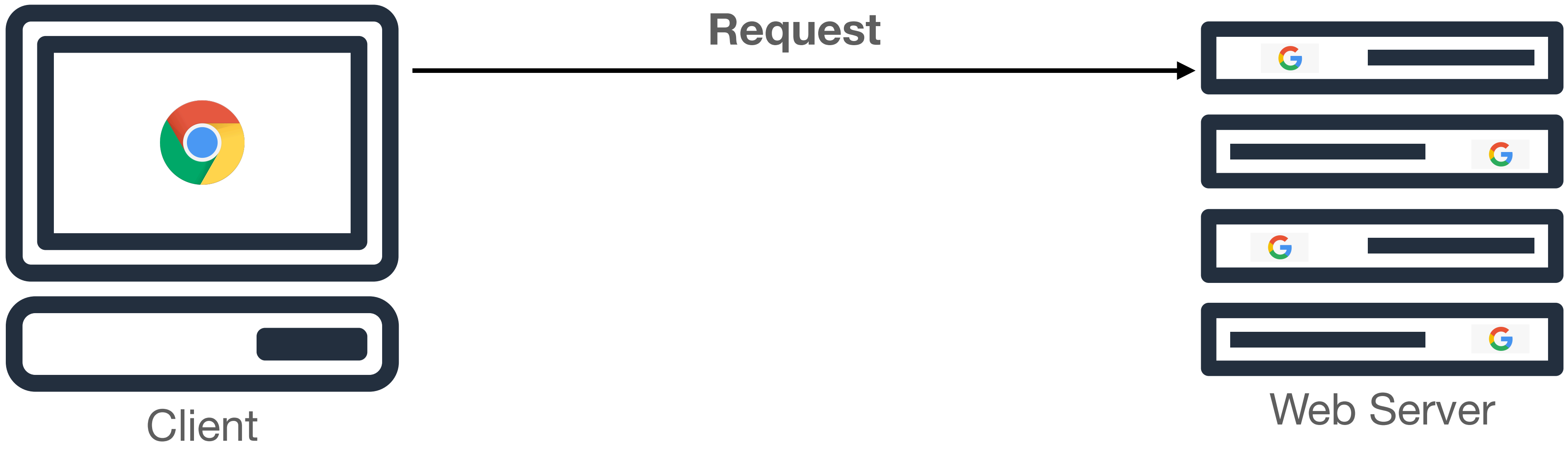
Interfacing with the Web

Client / Server Model

Client
Web Server

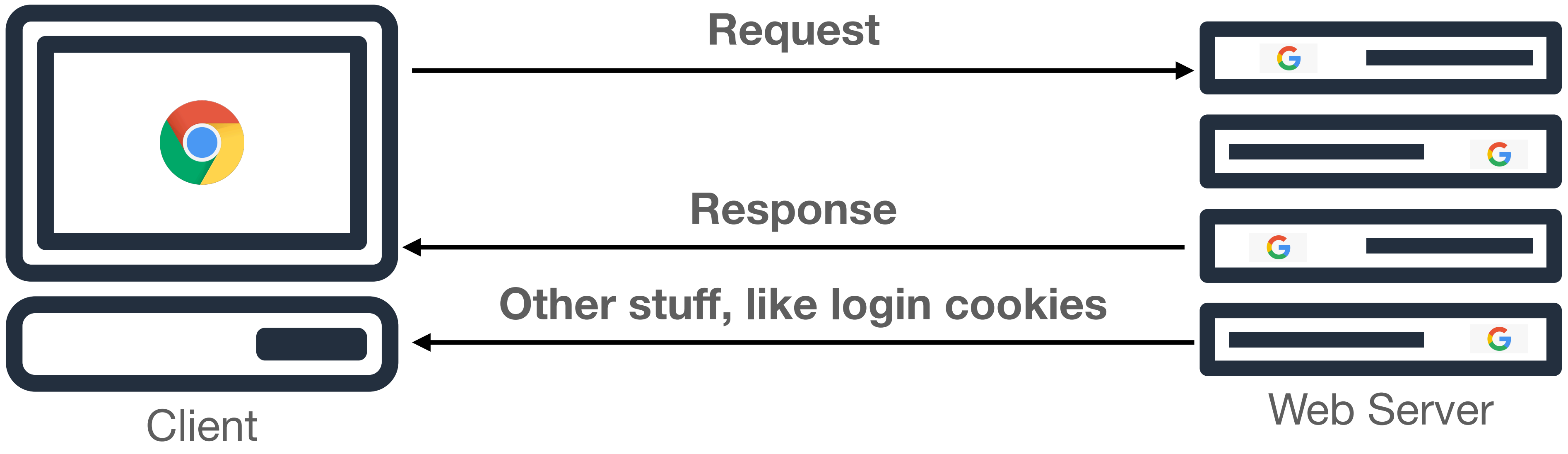
Interfacing with the Web

Client / Server Model



Interfacing with the Web

Client / Server Model



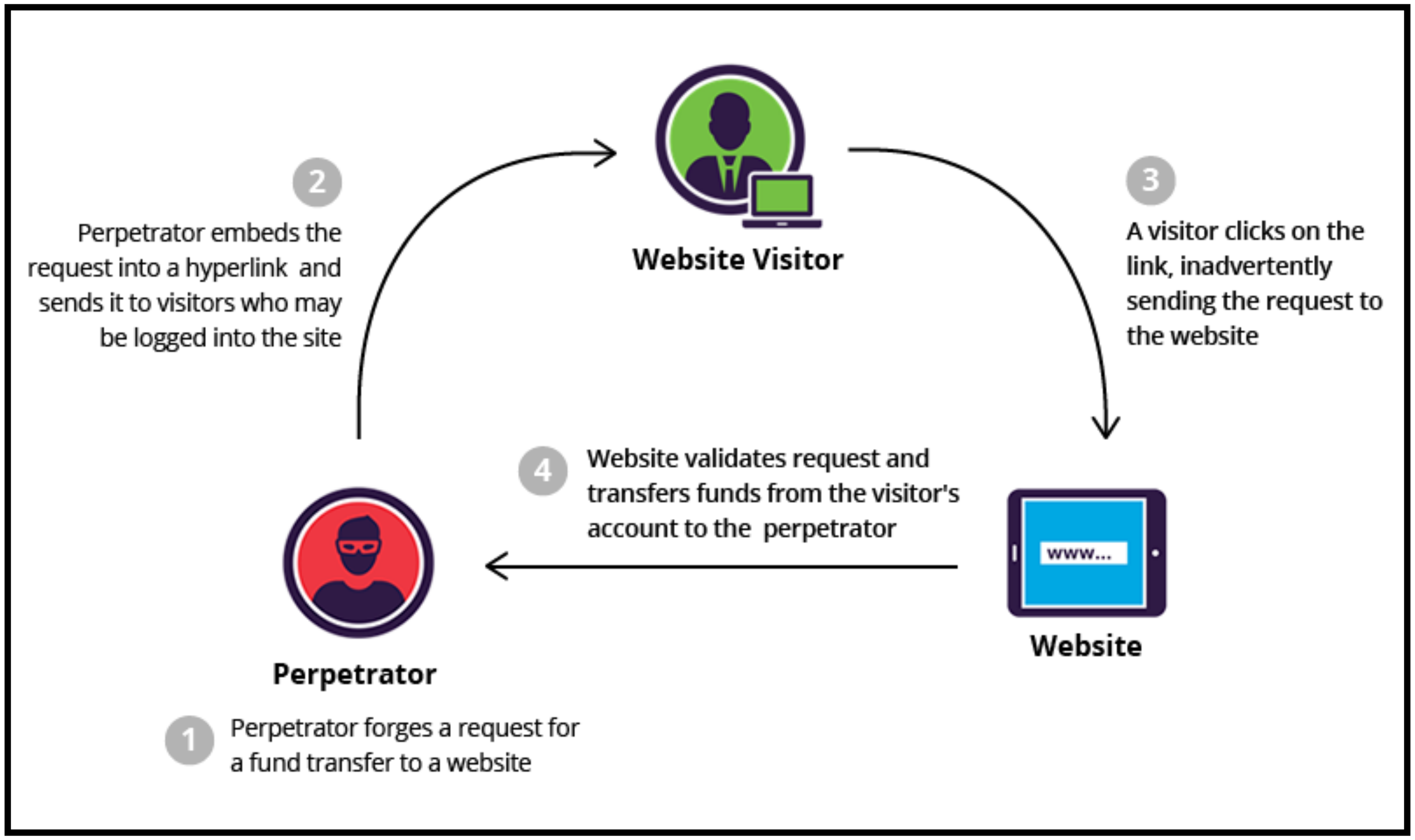
Robust Defenses for Cross-Site Request Forgery

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

What is Cross-Site Request Forgery?


Website Visitor
2
Perpetrator embeds the request into a hyperlink and sends it to visitors who may be logged into the site
3
A visitor clicks on the link, inadvertently sending the request to the website
4
Website validates request and transfers funds from the visitor's account to the perpetrator
www...
Website
Perpetrator
1
Perpetrator forges a request for a fund transfer to a website

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?

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)

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?

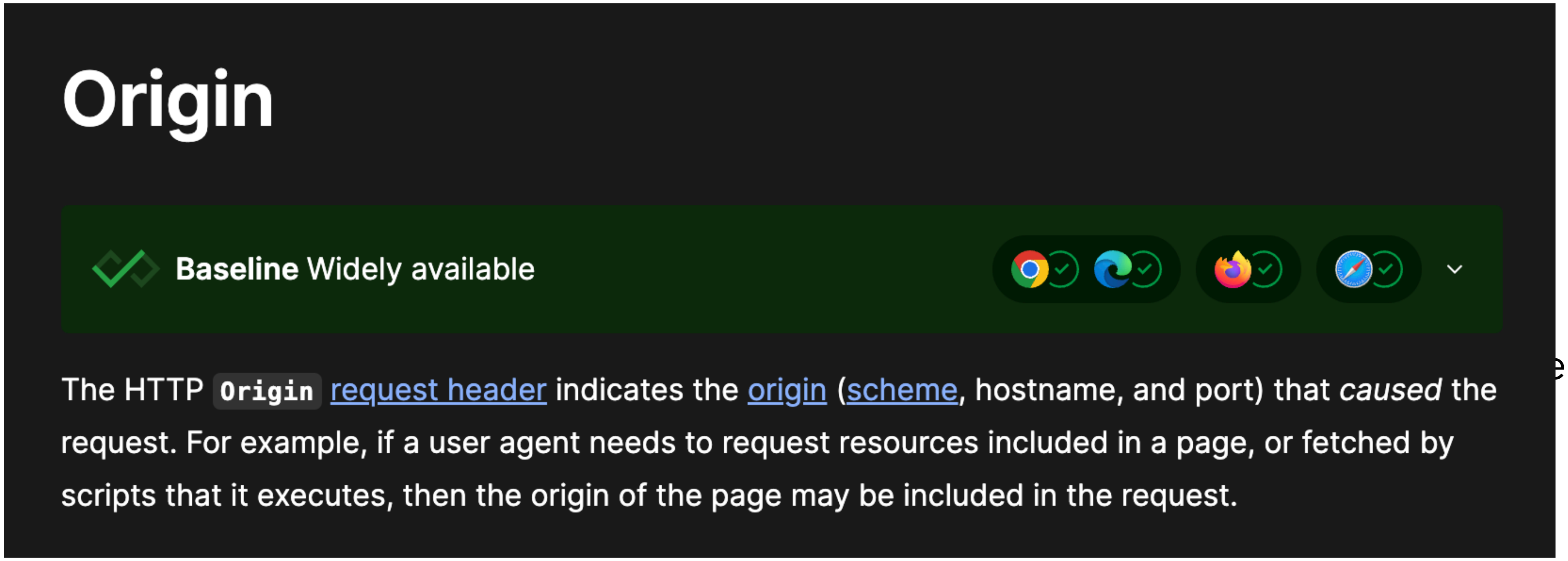
Defeating CSRF with the Referer header

- 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



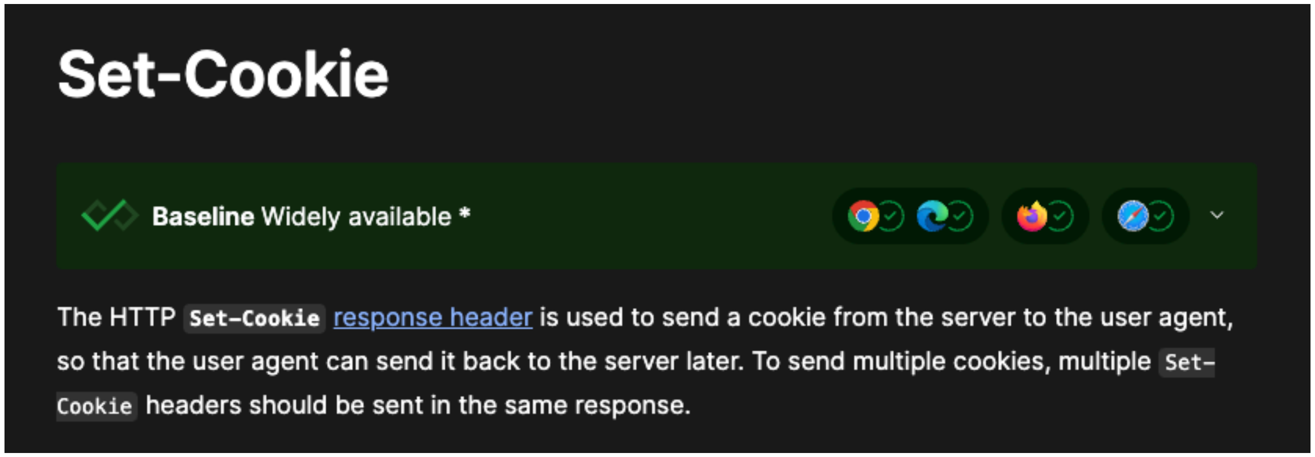
Origin

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.

e

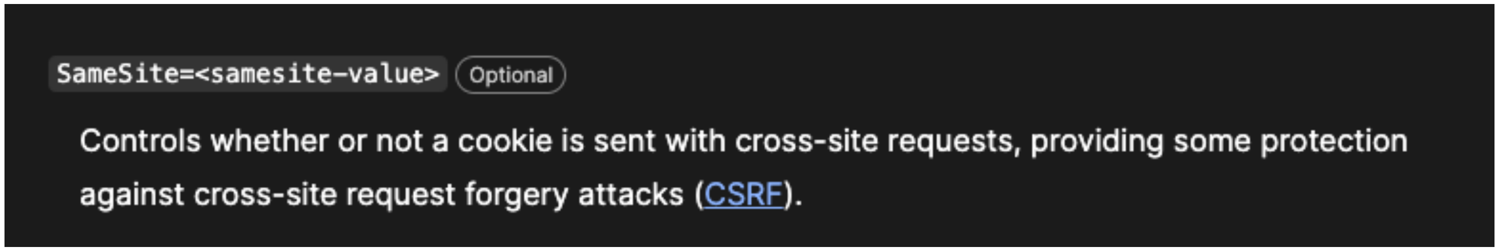
Today’s Defenses: SameSite Cookies



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).

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?

Break Time + Attendance

https://tinyurl.com/cse227-attend

Codeword:
See-Surf

Parse Me Baby One More Time: Bypassing HTML Sanitizer via Parsing Differentials

What is HTML?

What is HTML?

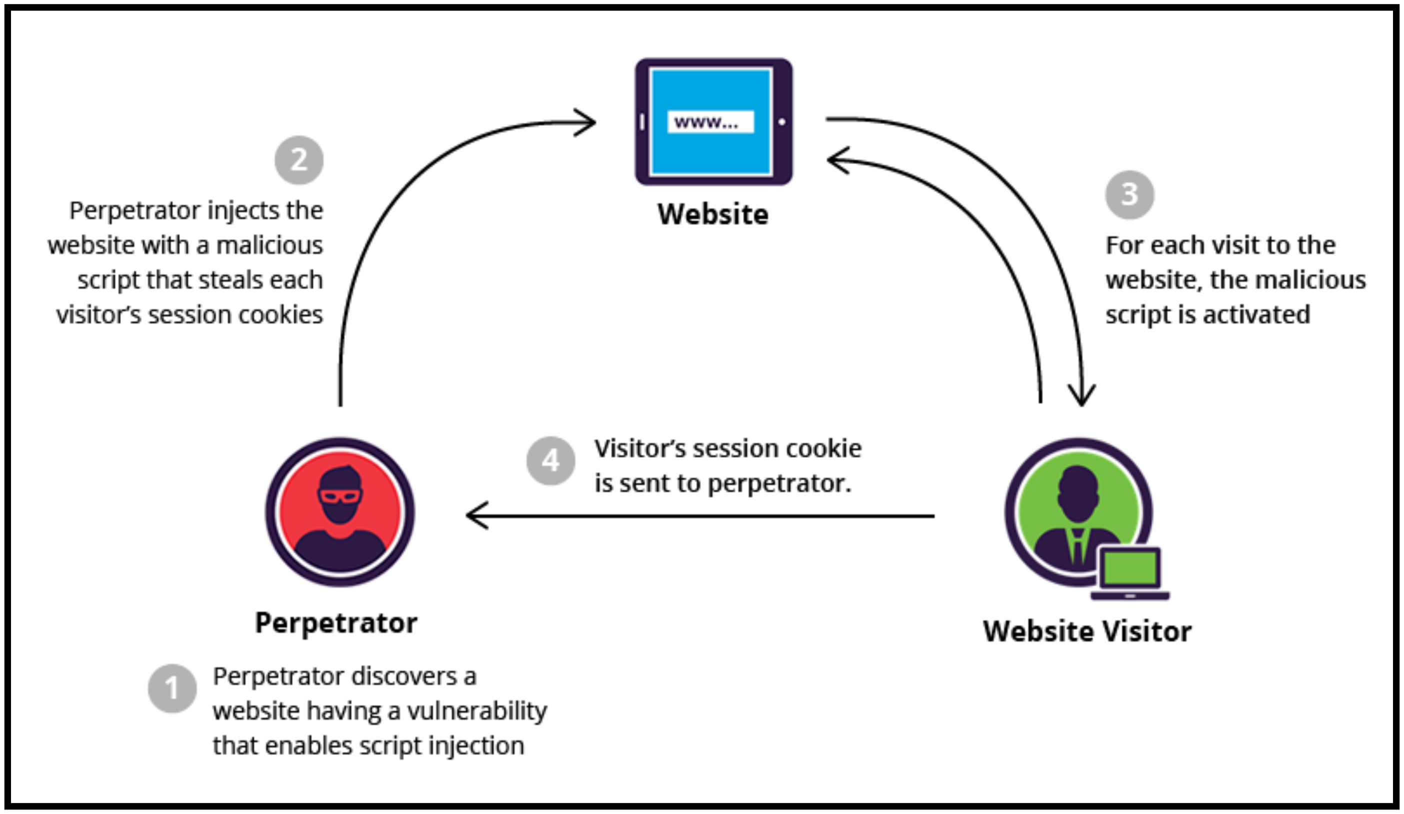
Hypertext Markup Language: The **structure** of how we embed web content into web pages.

What is Cross-Site Scripting (XSS)?

What is Cross-Site Scripting (XSS)?

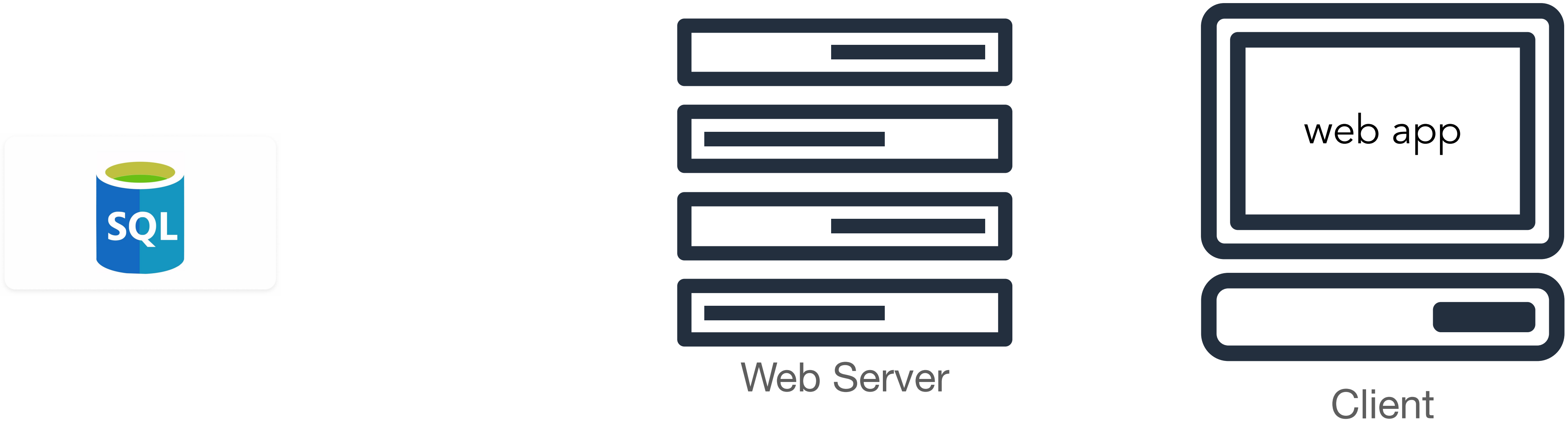
“Cross-Site Scripting (XSS) attacks are a type of injection, in which malicious scripts are injected into otherwise benign and trusted websites” – OWASP

How does XSS work?

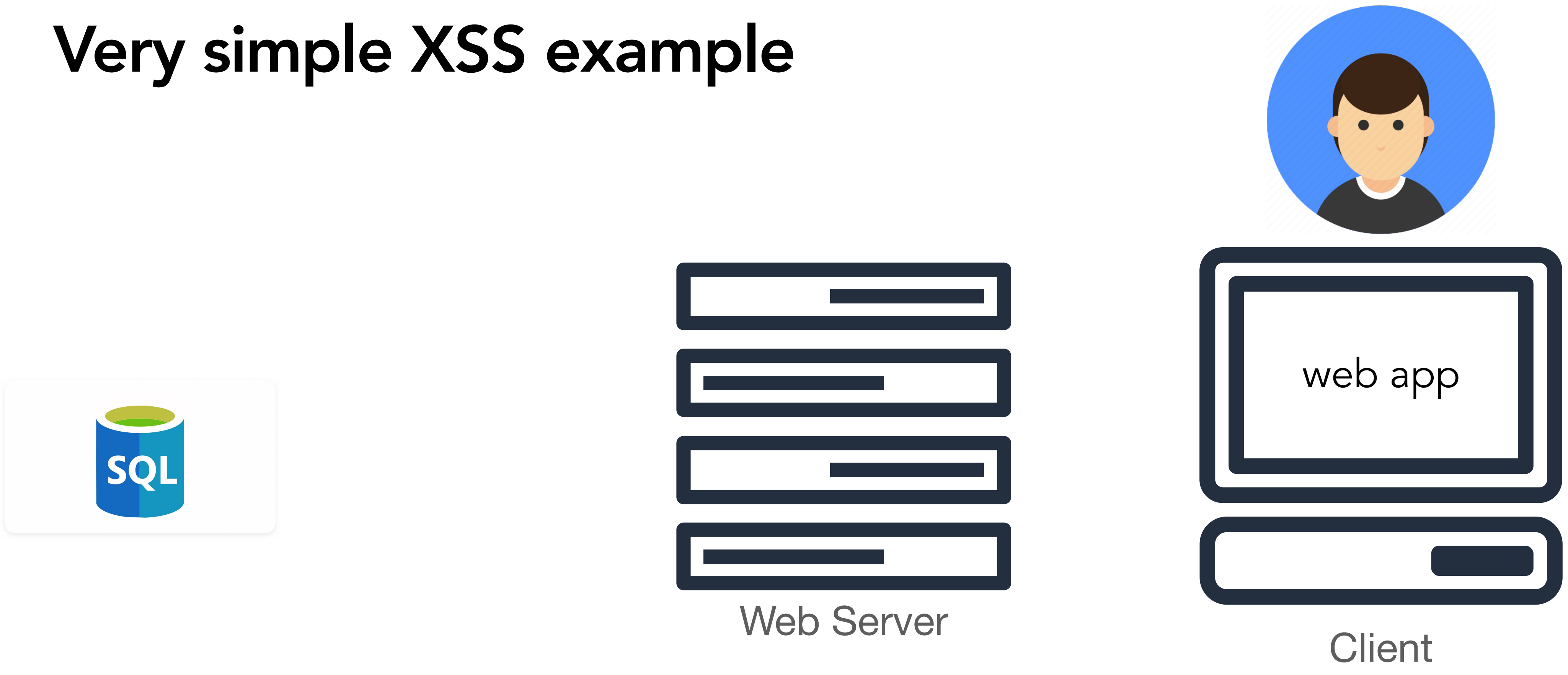

www...
Website
2
Perpetrator injects the website with a malicious script that steals each visitor's session cookies
3
For each visit to the website, the malicious script is activated
4
Visitor's session cookie is sent to perpetrator.
Perpetrator
Website Visitor
1
Perpetrator discovers a website having a vulnerability that enables script injection

Imperva

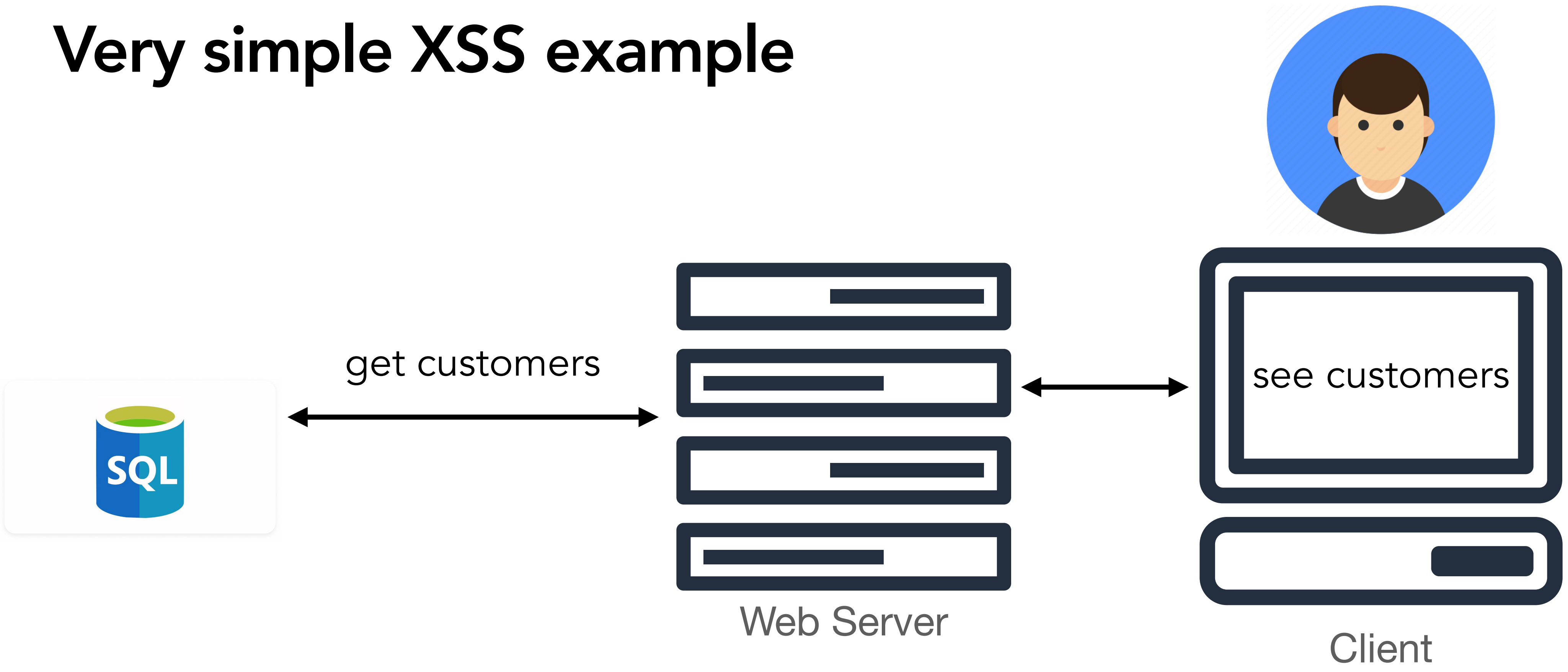
Very simple XSS example



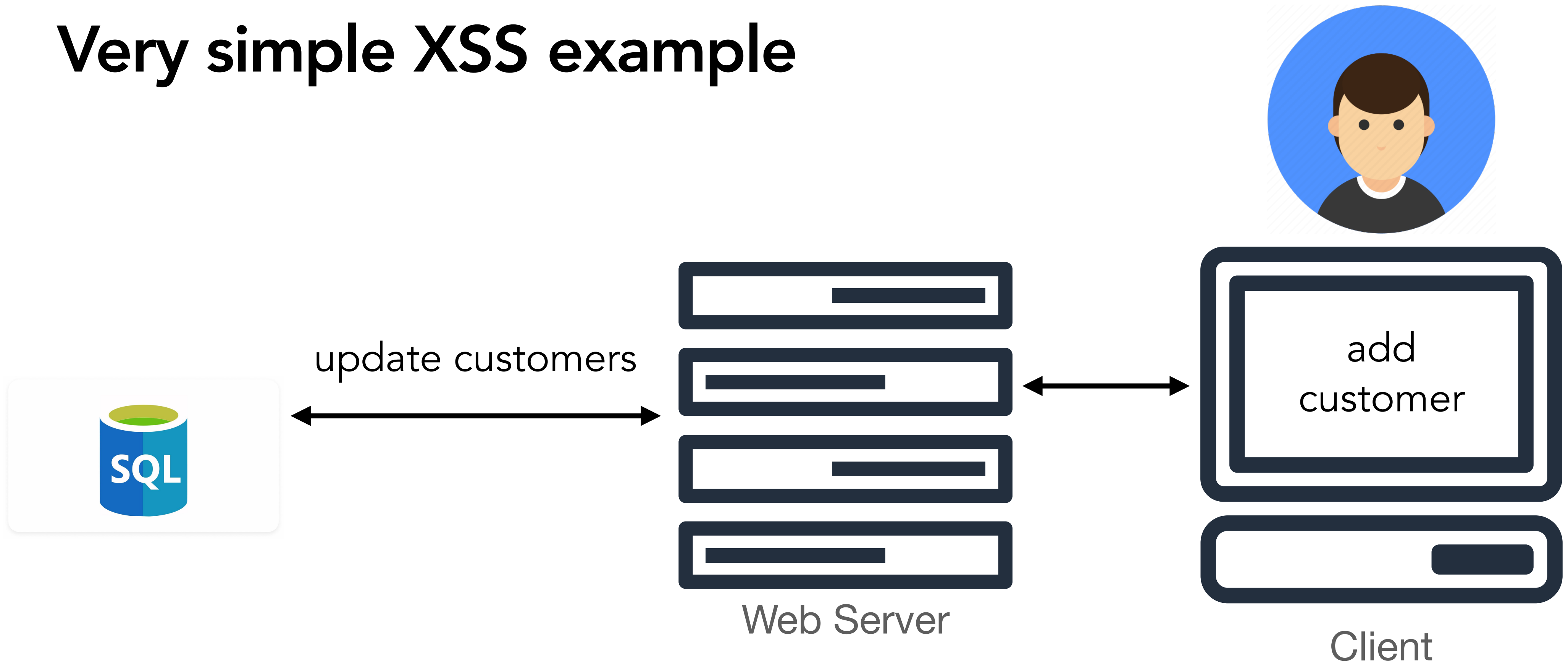
Very simple XSS example



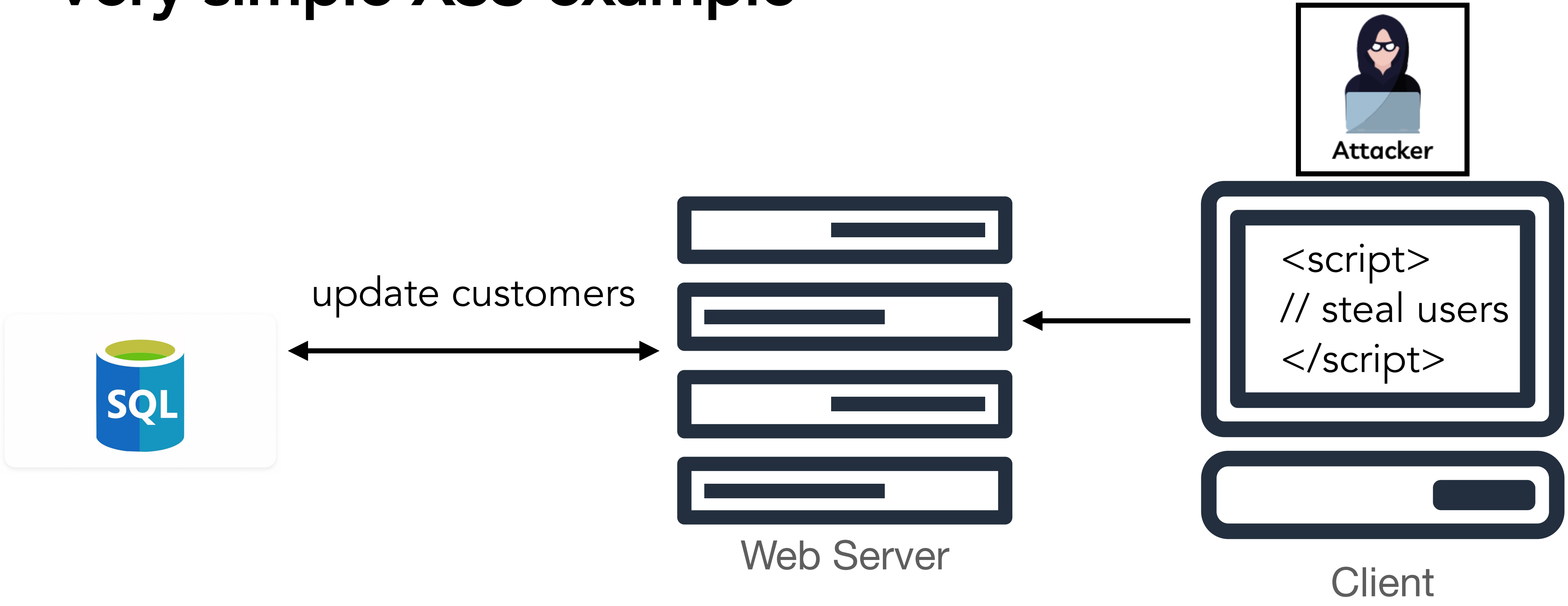
Very simple XSS example



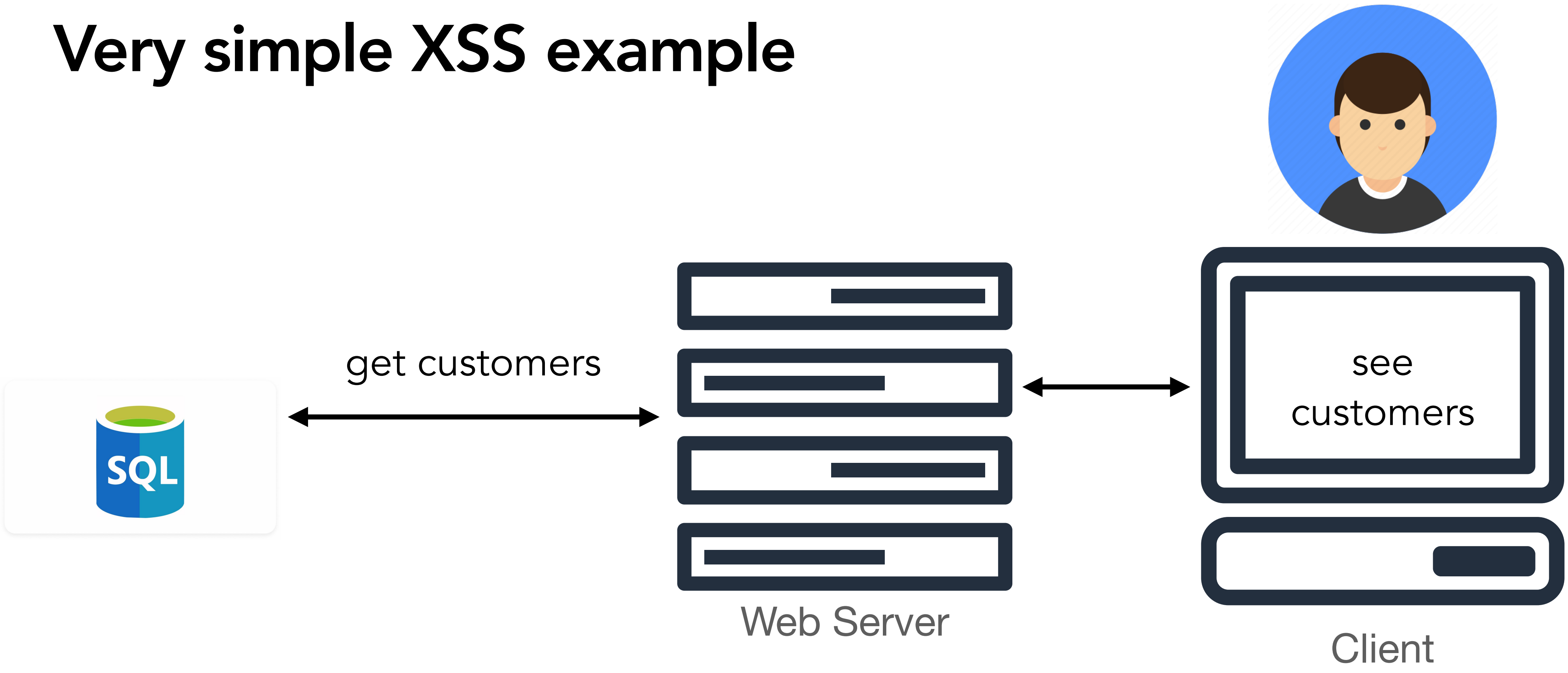
Very simple XSS example



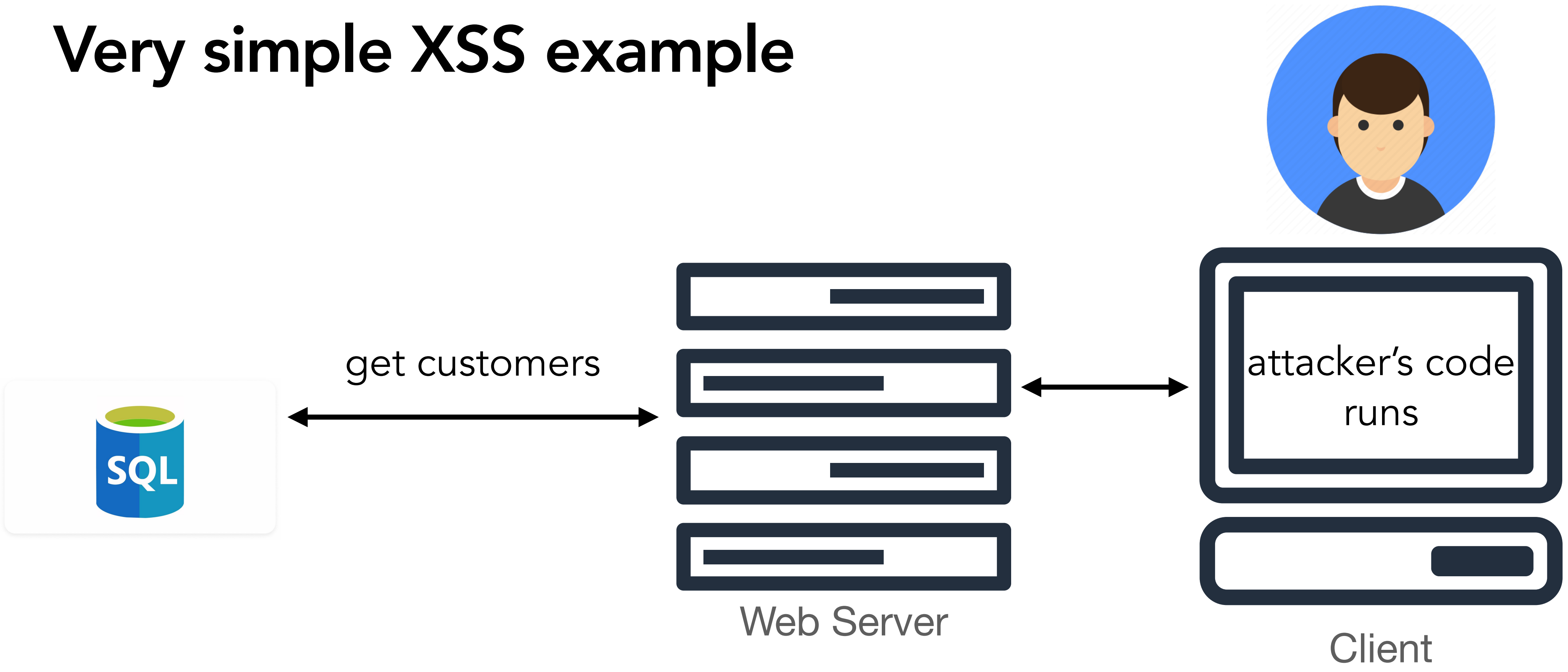
Very simple XSS example



Very simple XSS example



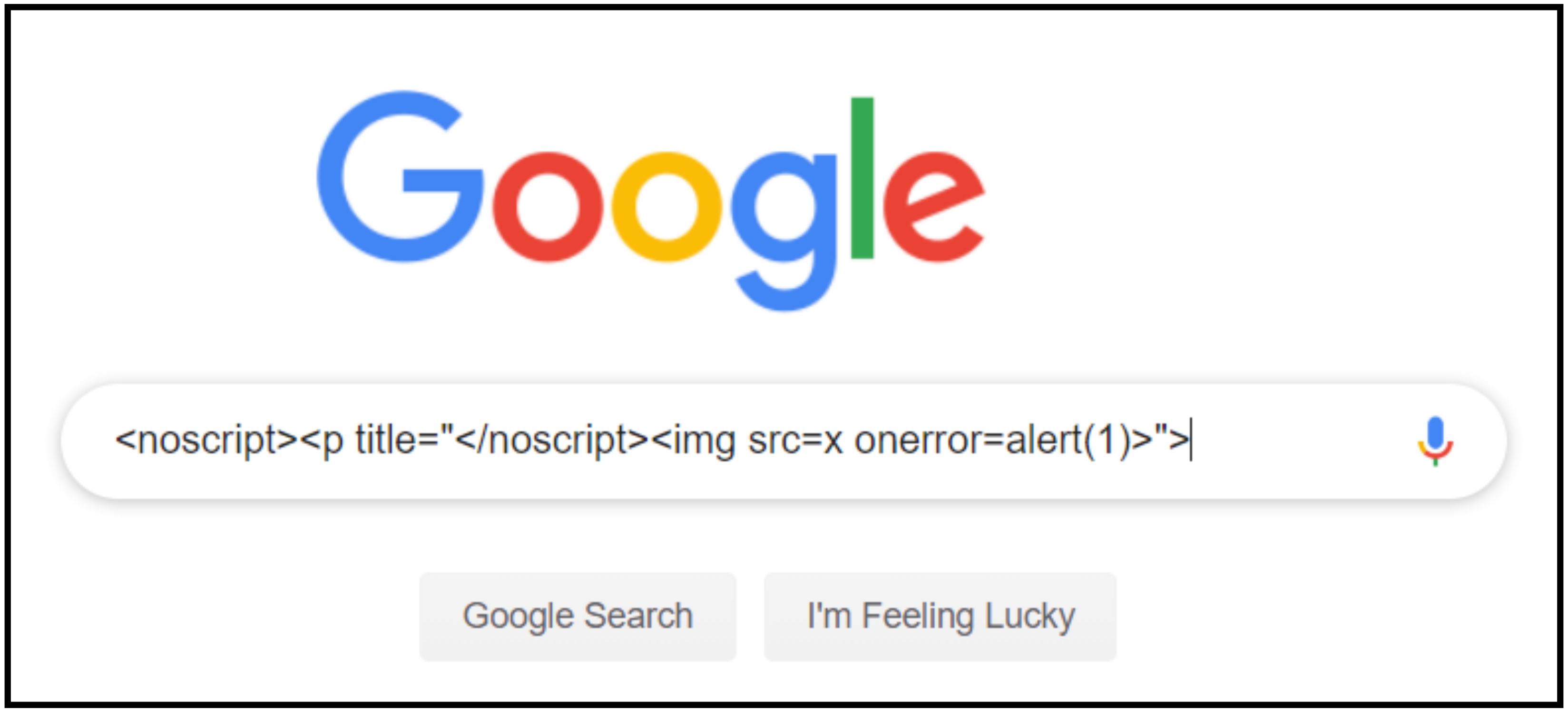
Very simple XSS example



What is Mutation Cross-Site Scripting (mXSS)?

“Such a vulnerability occurs if an HTML fragment is parsed, serialized, and yields a different result upon being parsed again.”

mXSS example: Google Search in 2019



```
<noscript><p title="</noscript><img src=x onerror=alert(1)>">
```

Common XSS defenses

- How do we defend against XSS attacks?

Common XSS defenses

- How do we defend against XSS attacks?
- What is input sanitization?

Common XSS defenses

- How do we defend against XSS attacks?
- What is input sanitization?
- Where does input sanitization happen? On the client side or server side?

Issues with server-side sanitization

- Why is accurate HTML sanitization quite hard for servers to do?

Issues with server-side sanitization

- Why is accurate HTML sanitization quite hard for servers to do?
  - Context dependent
  - Requires understanding how the browser is going to interpret the HTML, which turns out is not easy!

This paper asks two questions

1. Is server-side sanitization even feasible (does not ruin benign content) and is secure?
2. How do popular open-source libraries fare in parsing and sanitizing HTML content for XSS attacks?

Their setup for evaluating parsing differentials

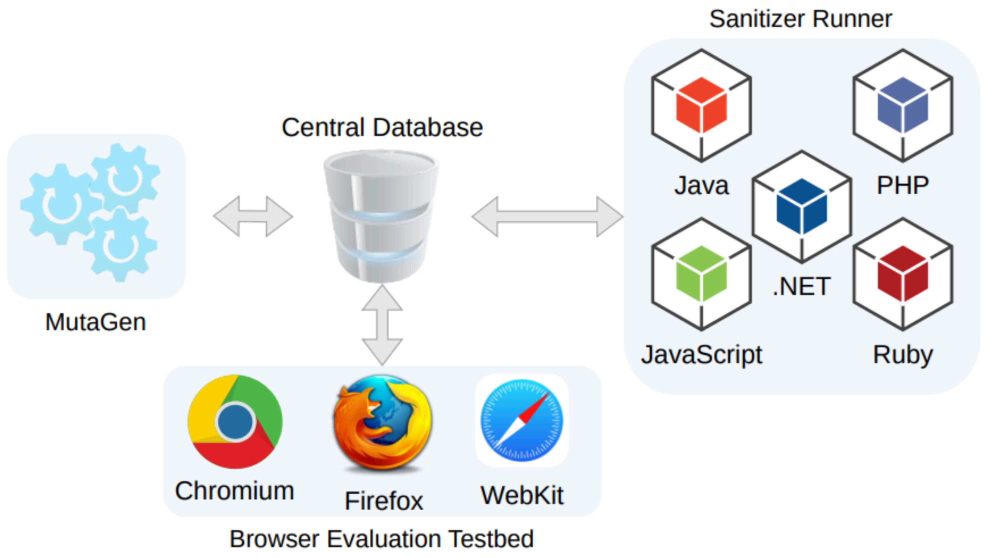


Figure 3: Sanitizer Evaluation Setup

Mutagen: Generating HTML Fragments

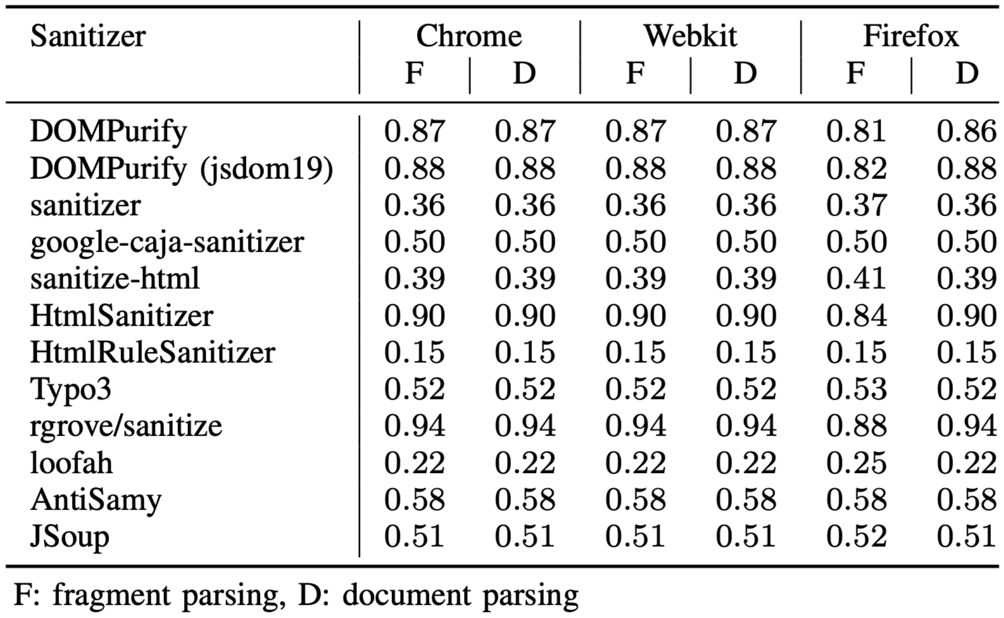
- Not going to get into the details here (there are many fine points) – general gist is as follows:
  - Start with a payload *P* that you're sure works
  - Make some transformations to *P* you think might be tricky for a browser
  - Test your set of transformations and keep the ones you think work
  - Repeat with new P

Parsing differential strategy

- Tested 11 (really 12) different parsers (common libraries), throwing all the generated mutations into each one, and saw how they compared
- How did the authors evaluate if the parsing was as they expected it to be?

Results

- What did the authors find as their top-line results?
- Did every browser interpret HTML identically? Which browsers didn't?
- What do these results tell us about HTML parsing?



Sanitizer	Chrome		Webkit		Firefox	
	F	D	F	D	F	D
DOMPurify	0.87	0.87	0.87	0.87	0.81	0.86
DOMPurify (jsdom19)	0.88	0.88	0.88	0.88	0.82	0.88
sanitizer	0.36	0.36	0.36	0.36	0.37	0.36
google-caja-sanitizer	0.50	0.50	0.50	0.50	0.50	0.50
sanitize-html	0.39	0.39	0.39	0.39	0.41	0.39
HtmlSanitizer	0.90	0.90	0.90	0.90	0.84	0.90
HtmlRuleSanitizer	0.15	0.15	0.15	0.15	0.15	0.15
Typo3	0.52	0.52	0.52	0.52	0.53	0.52
rgrove/sanitize	0.94	0.94	0.94	0.94	0.88	0.94
loofah	0.22	0.22	0.22	0.22	0.25	0.22
AntiSamy	0.58	0.58	0.58	0.58	0.58	0.58
JSoup	0.51	0.51	0.51	0.51	0.52	0.51

F: fragment parsing, D: document parsing

Combining our two papers…

- How does XSS related to CSRF?
- Do CSRF defenses protect against XSS?
- What is the relationship between XSS and CSRF?
- What would you say is a **fundamental issue** that enables a XSS attack?

Combining our two papers…

- How does XSS related to CSRF?
- Do CSRF defenses protect against XSS?
- What is the relationship between XSS and CSRF?
- What would you say is a **fundamental issue** that enables a XSS attack?
    - Mixing code and data!

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?

Discussion

What about these attacks *surprised* you?

What do these attacks teach us about *trust?*

Next time…

- Talking less about web attacks and more about the web *ecosystem* – i.e., web tracking