

root@localhost CTF Write Up

Used workshop recording, ChatGPT, Youtube, and various CTF write-ups available online to solve all the challenges!

▼ misc

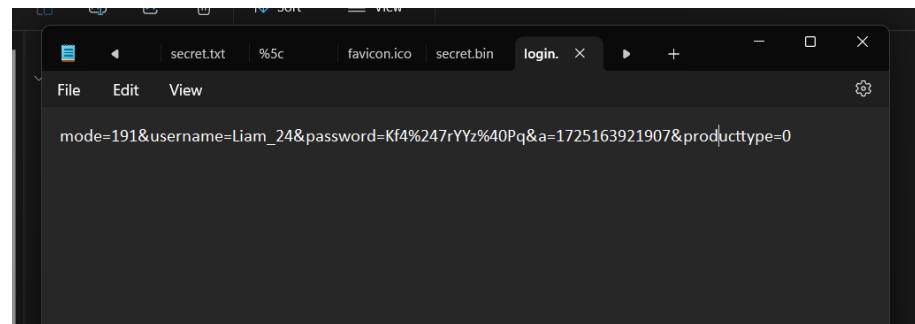
▼ Welcome

The flag was present in announcements of discord

```
root@localhost:{W3lc0m3_T0_r00t@10c4lh3l1!}
```

▼ The Great Login Heist

I used wireshark to extract the files, found the username inside login.xml



▼ Silent Courier

I downloaded the pcapng file and used a-packets website to analyze it. I was able to download the zip file.

After using John the ripper, I cracked the password.

```
root@localhost{you.....}
```

▼ play with QR

I sorted the images by date modified and found one qr had different time. Scanned the qr and got the flag

Name	Type	Compressed size	Password p-	Size	Ratio	Date modified
fake_qr_669	PNG File	1 KB	No	1 KB	1%	10/9/2024 2:29 PM
fake_qr_664	PNG File	1 KB	No	1 KB	0%	10/9/2024 2:19 PM
fake_qr_665	PNG File	1 KB	No	1 KB	0%	10/9/2024 2:19 PM
fake_qr_666	PNG File	1 KB	No	1 KB	0%	10/9/2024 2:19 PM
fake_qr_667	PNG File	1 KB	No	1 KB	0%	10/9/2024 2:19 PM
fake_qr_668	PNG File	1 KB	No	1 KB	0%	10/9/2024 2:19 PM
fake_qr_670	PNG File	1 KB	No	1 KB	0%	10/9/2024 2:19 PM
fake_qr_671	PNG File	1 KB	No	1 KB	0%	10/9/2024 2:19 PM
fake_qr_672	PNG File	1 KB	No	1 KB	0%	10/9/2024 2:19 PM
fake_qr_673	PNG File	1 KB	No	1 KB	0%	10/9/2024 2:19 PM
fake_qr_674	PNG File	1 KB	No	1 KB	0%	10/9/2024 2:19 PM
fake_qr_675	PNG File	1 KB	No	1 KB	0%	10/9/2024 2:19 PM
fake_qr_676	PNG File	1 KB	No	1 KB	0%	10/9/2024 2:19 PM
411e...c37	PNG File	1 KB	No	1 KB	0%	10/9/2024 2:19 PM

```
root@localhost{7h3_q6_!s_fun}
```

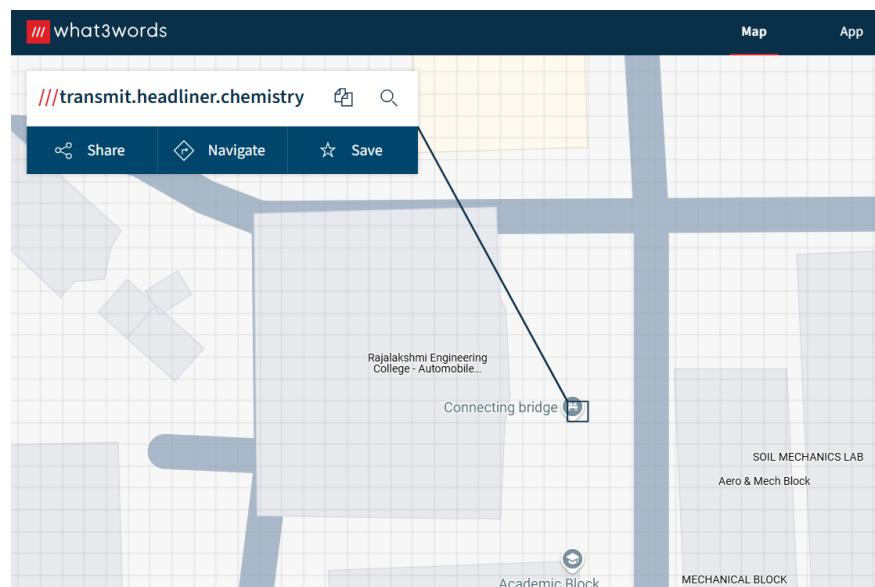
▼ Osint

▼ Weak

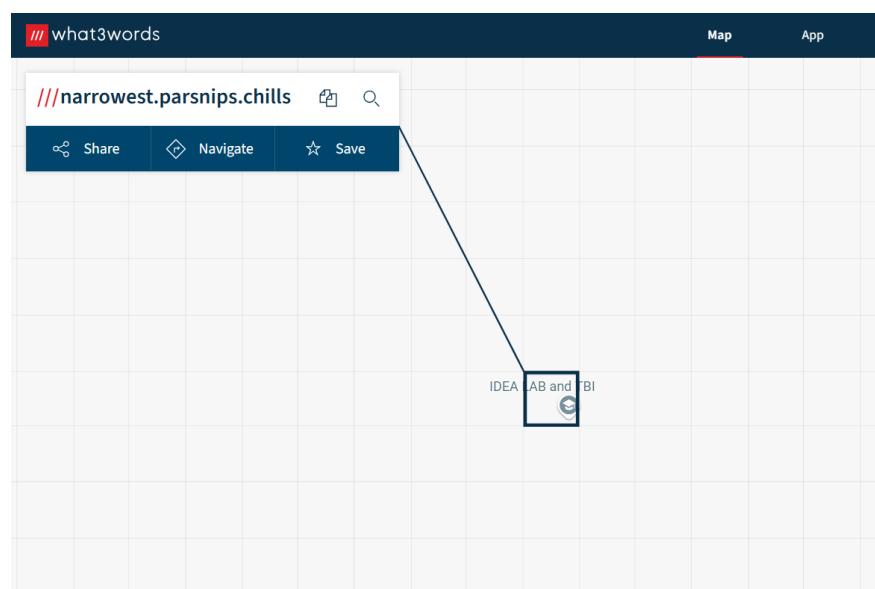
Used chatgpt to get the most commonly used weak passwords

12345678

▼ Locate the Bridge



▼ Find the Lab

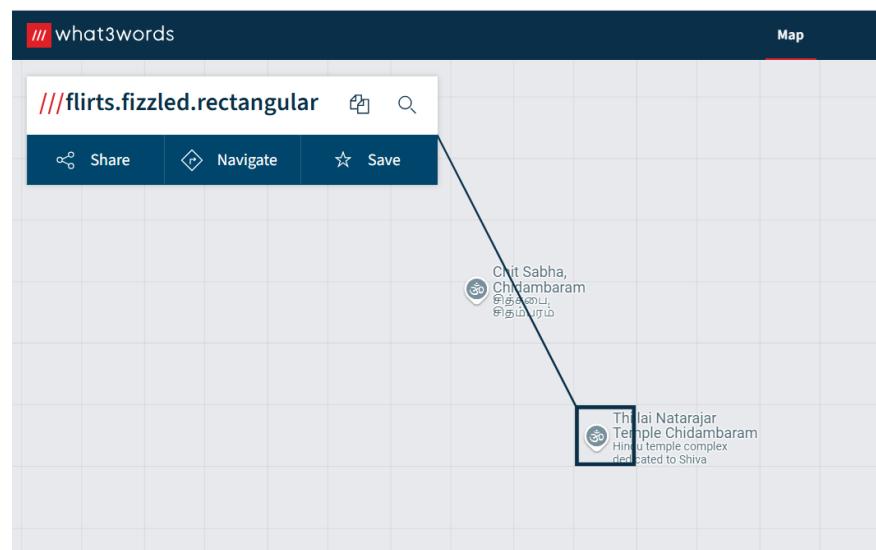


▼ The Magnetic Epicenter

I found the place of magnetic epicenter in TamilNadu using ChatGPT and located it in the what3words website!

A certain point in Tamil Nadu is often considered to align closely with the Earth's magnetic equator. Your task is to locate this point and retrieve its what3words address

Searched 4 sites ▾
The Chidambaram Nataraja Temple in Tamil Nadu is often associated with claims of being aligned closely with the Earth's magnetic equator. While this is a widely circulated belief, there is no scientific confirmation that the temple is precisely on the magnetic equator. The temple is renowned for its architectural beauty and spiritual significance, with a history tracing back to the Chola dynasty
OBSERVER DAWN BEHIND EVERY TEMPLE .
Unfortunately, I could not locate a precise what3words address for this point. However, if you'd like to explore further or pinpoint a location in Chidambaram using what3words, you can visit what3words.com. Let me know if you need assistance finding specific coordinates!
🕒 🗺️ 🚻 🕵️ 🌐



▼ Find the ranch

I used exiftool to extract the metadata of photo.jpg

I then converted it to degrees and found the location in Google Maps

Big_River_Ranch

```

Exposure Time : 1/14
F Number : 2.2
Exposure Program : Program AE
ISO : 800
Exif Version : 0220
Date/Time Original : 2021:06:11 21:33:04
Create Date : 2021:06:11 21:33:04
Offset Time : -05:00
Offset Time Original : -05:00
Shutter Speed Value : 1
Aperture Value : 2.2
Exposure Compensation : 0
Max Aperture Value : 2.4
Metering Mode : Center-weighted average
Flash : No Flash
Focal Length : 1.8 mm
Color Space : sRGB
Exif Image Width : 4608
Exif Image Height : 3456
Exposure Mode : Auto
White Balance : Auto
Digital Zoom Ratio : 1
Focal Length In 35mm Format : 13 mm
Scene Capture Type : Standard
Image Unique ID : L12XLLD00SM
GPS Latitude Ref : North
GPS Longitude Ref : West
Image Width : 4608
Image Height : 3456
Encoding Process : Baseline DCT, Huffman coding
Bits Per Sample : 8
Color Components : 3
YCbCr Sub Sampling : YCbCr4:4:4 (1 1)
Aperture : 2.2
Image Size : 4608x3456
Megapixels : 15.9
Scale Factor To 35 mm Equivalent: 7.2
Shutter Speed : 1/14
Date/Time Original : 2021:06:11 21:33:04-05:00
Modify Date : 2021:06:11 21:33:04-05:00
GPS Latitude : 39 deg 12' 22.41" N
GPS Longitude : 93 deg 49' 34.69" W
Circle Of Confusion : 0.004 mm
Field Of View : 108.3 deg
Focal Length : 1.8 mm (35 mm equivalent: 13.0 mm)
GPS Position : 39 deg 12' 22.41" N, 93 deg 49' 34.69" W
Hyperfocal Distance : 0.35 m
Light Value : 3.1

```

(kali㉿kali)-[~/Downloads]

▼ The Cyber Sentinels Hunt

I found the bits and pieces of encoded flag in the comment sections of LinkedIn, Instagram

r00t@localhost [OSInT_Cha1n3d_To_Th3_W3b]

Posts



Comments

Reply

1

_sam_sk 8w • ❤️

Reply

cybersentinels_rec 8w • Author @__sam_sk 🔥

Reply

cybersentinels_rec 4d • Author @__sam_sk Flag vro cJAwdBsb2NhbgHvc3R7T1MxbI Q=

Reply

Hide Replies

Nice 🎉, Can't wait to join the CTF challenge 😊

Like | Reply

Harish S • 2nd Student at Rajalakshmi Engineering Col... 1d

Worth attending

Like | Reply

Thoufeeq Mohammed M • 2nd 1d

Super 🔥

Like | Reply

JOTHIPRASAD D • 2nd Student at Rajalakshmi Engineering Col... 1d

Great 🌟 Ready for the Challenge 🎉

Like | Reply 1

Cyber Sentinels • Author 81 followers 1d

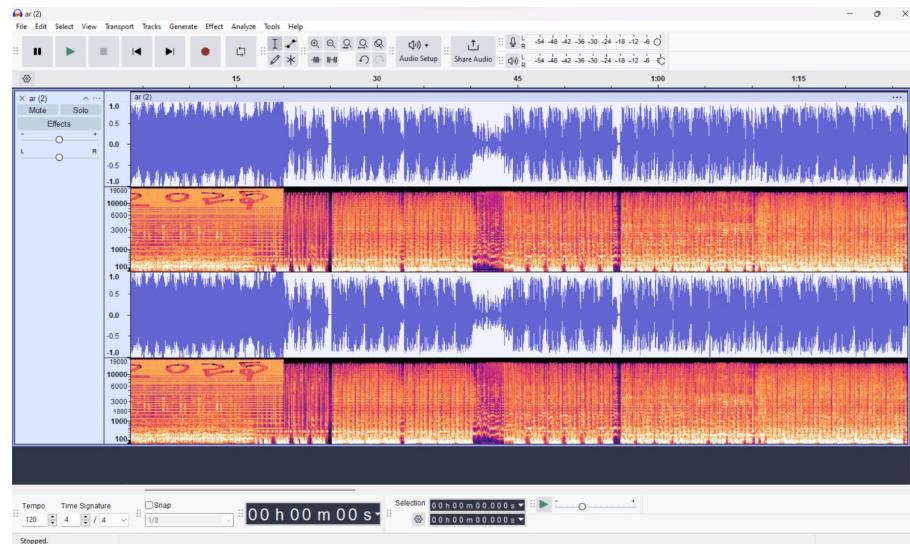
X0NoYTFuM2RFVG8=

Like 2 | Reply

▼ Steganography

▼ Echo of Time

Found the year with the help of a tool called Audacity. The year was visible in the Multi-View form. Thanks to Siva bro who taught us about this tool during the workshop.



▼ Hidden Truth

After extracting the metadata using the online exiftool, I found that the title contained base64 encoding. I then entered that into Cyberchef to find the flag.

Image EXIF data:

ExifTool Version Number:	12.12
File Name:	exif7-3-c303aa9000.png
Directory:	
File Size:	2.0 MB
File Modification Date/Time:	2024/12/08 18:13:14+01:00
File Access Date/Time:	2024/12/08 18:13:14+01:00
File Inode Change Date/Time:	2024/12/08 18:13:14+01:00
File Flags:	0
File Type:	PNG
File Type Extension:	png
MIME Type:	image/png
Image Width:	1280
Image Height:	720
Bit Depth:	8
Color Type:	RGB
Compression:	Deflate/Inflate
Interlace:	None
Pixels Per Unit X:	2700
Pixels Per Unit Y:	2700
Pic Units:	meters
XMP Toolkit:	ImageExifTool 12.16
Adm Toolset:	00020cf-df76-4bae-8ddc-9ea2ed20526
Adm Fd Id:	E2520591479590
Adm File Type:	
Title:	cverhfbld2N8MjQwv3R7QzBzZ1AaCRWtB1X0YwvWSk1RqAfV9aXNMj5J5dwe20h
Image Size:	1280x720
Megapixels:	0.922

▼ Pixel Secrets

I wrote a bash script to brute for the passwords given in password.txt

```
while read -r password; do
    steghide extract -sf steg1.jpg -p "$password" && break
done < password.txt
```

After executing this script, the script will automatically create a separate file, I was able to extract the information.

```
root@localhost{H1dd3n_M3ss4g3_F0und}
```

▼ Secret Stash

I wrote the same bash script with little modifications.

```
while read -r password; do
    steghide extract -sf steg2.jpg -p "$password" && break
done < steg2_pass.txt
```

After executing this script, I was able to access secret.zip which asked me for a password.

To crack the password, I used John the Ripper.

```
zip2john secret.zip > local.txt
```

This cmd will convert zip to txt file.

```
john --wordlist=/usr/share/wordlists/rockyou.txt local.txt
```

I cracked the password and opened the flag.txt file.

```
root@localhost{SecureByDesign!2024}
```

▼ Web

▼ Easy-Web_challenge

I found the username and password of the application in the script.js of the source code!

```
root@localhost[The_web_chall_is_easy]
```

▼ Mini Vulnerable Compiler

I used ChatGPT to generate code with the most common file name for storing the flag!

```

ChatGPT < Memory Full >
• Console: Look at the browser console for JavaScript or server-side errors. Any issues in rendering the page could show up there.
4. Modify Your Code
If your attempt to exploit the system or read files resulted in no visible output, you could try simpler approaches. For example, instead of executing commands directly, try loading files step-by-step using basic Python code.
python
try:
    with open('flag.txt', 'r') as file:
        print(file.read())
except Exception as e:
    print(f"Error: {e}")
5. Review Server Logs
If the online compiler allows you to access log ↓ error messages from the server, reviewing them might give you insight into why your code isn't working as expected.

```

r00t@localhost[mini_compiler_pwn]

▼ iDoor: The Secret Portal

I understood that the value of camera in the URL is encoded to generate the corresponding customer ID 20 using ChatGPT

Then tried replacing the camera value by encoding different numbers, and finally found the right ID as 0.

iDoor
Customer ID: 20
Camera Status: Inactive
Flag: root@localhost{the_is_fake_flag}

Zoom In Zoom Out Capture Record Full Screen Pan Right

SHA256 Encrypt/Decrypt

Encrypter Decrypter

SHA256 Hash:
f5eca8f748a1d5eaft26b8a42fb575c5c7ff1864a81435017
82de13da2d9202b

Text:
0

Elapsed Time: 0.377s Trial Count: 17K

SHA256 Encrypt/Decrypt

Encrypter Decrypter

Text:
0

SHA256 Hash:
f5eca8f748a1d5eaft26b8a42fb575c5c7ff1864a81435017
82de13da2d9202b

iDoor
Customer ID: 0
Camera Status: Active
Flag: root@localhost{the_is_fake_flag}

r00t@localhost[770a058a80a9bca0a87c3e2ebe]

▼ XSS vulnerability

Used ChatGPT to generate code that bypasses the sanitized input field, revealing the web app's XSS vulnerability!

Give some malicious cmd
You learn about how_cross_site_scripting

Give me something:

Submit

xss-jdin.onrender.com says
root@localhost:~/ptfs_Sanitizer.t23)

Your lower cased input is :

▼ jwt

I used ChatGPT to crack the hash for the jwt key in the demo user account of the application. I found that the hash was "lol" instead of "your_secret_key." I replaced "lol" with the generated key, logged into the system, and got the flag.

STUDENT CARE FINDER

LOGIN

Username

Password

```

Page Workspace > index > comput.get.default.export >
top
web2-k7a3.onrender.com
(index)
<html>
  <head>
    <title>STUDENT CARE FINDER</title>
    <link href="https://fonts.googleapis.com/css2?family=Open+Sans:wght@400;600;700;800;900&display=block" rel="stylesheet">
    <link href="https://cdn.jsdelivr.net/npm/@material-ui/core@4.11.3/dist/material-ui.min.css" rel="stylesheet">
    <script src="https://unpkg.com/react@17.0.2/umd/react.development.js"></script>
    <script src="https://unpkg.com/react-dom@17.0.2/umd/react-dom.development.js"></script>
    <script src="https://unpkg.com/@material-ui/core@4.11.3/umd/material-ui.development.js"></script>
  </head>
  <body>
    <div class="container">
      <div>
        <h1>STUDENT CARE FINDER</h1>
        <h2>LOG IN</h2>
        <form>
          <div>
            <label>Username</label>
            <input type="text" placeholder="Enter Username" name="username" value="Enter Username" />
          </div>
          <div>
            <label>Password</label>
            <input type="password" placeholder="Enter Password" name="password" value="Enter Password" />
          </div>
          <div>
            <button type="submit">LOG IN</button>
          </div>
        </form>
      </div>
    </div>
  </body>
</html>

```

SYSTEM DASHBOARD

YOU ARE LOGGED IN AS USER DEMO.

System Status:

Welcome demo user! In the real app, you'd see your confidential system info here!

Name	Value	Domain	Path	Exp.	Size	Htt.	Sec.	Sa...	Par...	Cro...	Pr...
token	eyJhbGciOiJIUzI1NiIsInR5cCI6IkpXVCJ9.eyJcI2VyIjoiZGVtbyJ9.41kHonJ4bofdcJG8VnrnzGL9KKUQwknSTccjAj-0.M	web2-k7a3.o...	/	Ses...	106						

HEADER: ALGORITHM & TOKEN TYPE

```
{
  "alg": "HS256",
  "typ": "JWT"
}
```

PAYOUT: DATA

```
{
  "user": "demo"
}
```

VERIFY SIGNATURE

```

HMACSHA256(
  base64UrlEncode(header) + "." +
  base64UrlEncode(payload),
  your-256-bit-secret
) □ secret base64 encoded

```

HEADER: ALGORITHM & TOKEN TYPE

```
{
  "alg": "HS256",
  "typ": "JWT"
}
```

PAYOUT: DATA

```
"user": "root"
```

VERIFY SIGNATURE

```

HMACSHA256(
  base64UrlEncode(header) + "." +
  base64UrlEncode(payload),
  your-256-bit-secret
) □ secret base64 encoded

```

```
echo "eyJhbGciOiJIUzI1NiIsInR5cCI6IkpXVCJ9.eyJlc2VyIjoiZGVtbyJ9.4lkHonJ4bofdcJG8VnrnzC"
hashcat -a 0 -m 16500 jwt_signature.txt /usr/share/wordlists/rockyou.txt
```

HEADER: ALGORITHM & TOKEN TYPE

```
{
  "alg": "HS256",
  "typ": "JWT"
}
```

PAYOUT: DATA

```
"user": "root"
```

VERIFY SIGNATURE

```
HMACSHA256(
  base64UrlEncode(header) + "." +
  base64UrlEncode(payload),
  key
) □ secret base64 encoded
```

▼ Crypto

▼ Decode The Hex Value

I used Cyberchef for decoding the hex value:

[https://gchq.github.io/CyberChef/#recipe=From_Hex\('Space'\)&input=NzIgNmYgNmYgNzQgNDAgNmMgNmYgNj](https://gchq.github.io/CyberChef/#recipe=From_Hex('Space')&input=NzIgNmYgNmYgNzQgNDAgNmMgNmYgNj)

```
root@localhost{The_Hex_value_is_3433f}
```

▼ DH - 9000

Used chatgpt to find the shared secret.

```

mod = int(input("Public mod: "))
base = int(input("Public base: "))

a_secret = int(input("A secret: "))
b_secret = int(input("B secret: "))

a_public = (base ** a_secret) % mod
b_public = (base ** b_secret) % mod

print("====")
print("A public = " + str(a_public)) DH - 9000
print("B public = " + str(b_public))

The robots just introduced this supposedly unbreakable crypto
scheme that allows them to share secrets over insecure channels. DH-
9000. I'm pretty sure this isn't anything new though, so we should still
be able to find their shared secret.

p = 8089
q = 823
A = 7608
B = 5796

Note: submit the shared secret wrapped in root@localhost]
print("A public = " + str(a_public))
print("B public = " + str(b_public))

a_shared = (b_public ** a_secret) % mod
b_shared = (a_public ** b_secret) % mod

print("A shared secret = " + str(a_shared))
print("B shared secret = " + str(b_shared))

```

④ This scenario describes the Diffie-Hellman key exchange scheme, where two parties exchange public values to compute a shared secret. The task involves deriving the shared secret from given values.

Provided Information:

- Prime modulus (p): 8089
- Base (g): 823
- Public key of A (A): 7608
- Public key of B (B): 5796

You need to find the shared secret using the formula:

$$\text{Shared secret} = (B^a \mod p) = (A^b \mod p)$$

Steps to Solve:

1. Recover the private keys (a and b). The private key of A or B is unknown, so brute-forcing a and b is necessary. Since the Diffie-Hellman scheme relies on modular exponentiation, finding the private key involves solving the discrete logarithm problem:

$$A \equiv g^a \pmod{p}$$

2. Compute the shared secret: Once a or b is found, the shared secret can be computed using

Maxence ChatGPT

-
2. Compute the shared secret: Once a or b is found, the shared secret can be computed using $A^b \bmod p$ or $B^a \bmod p$.
-

Python Implementation:

Here's how to solve it programmatically using brute force to find the private keys:

```
python
# Given values
p = 8089
g = 823
A = 7608
B = 5796

# Brute force to find a_secret
a_secret = None
for i in range(p):
    if pow(g, i, p) == A:
        a_secret = i
        break

# Brute force to find b_secret
b_secret = None
for i in range(p):
    if pow(g, i, p) == B:
        b_secret = i
        break

# Calculate shared secret
if a_secret is not None and b_secret is not None:
    shared_secret = pow(B, a_secret, p) # or pow(A, b_secret, p)
    print(f"Shared secret: root@localhost{{shared_secret}}")
else:
    print("Failed to find secrets.")
```

Output:

When executed, the script will:

1. Compute the private key(s) a and b .
2. Use the private key(s) to calculate the shared secret.
3. Format the result as `root@localhost{shared_secret}` for submission.



[find the flag](#)



The flag is:

```
scss
Copy code
root@localhost{2293}
```

▼ Route 47

used: <https://www.dcode.fr/rot-47-cipher>

```
root@localhost{You_got_the_flag_using_rot47}
```

▼ The Rail Conductor's Secret

Using the question, confirmed I had to use a rail fence decoder with a height of 24. Then tried multiple combinations of the available options, and finally found a match with the flag!

Challenge 30 Solves X

The Rail Conductor's Secret

82

As a rail conductor for the ancient Conclave, you've stumbled upon a mysterious train schedule, encrypted to protect the ultimate secret. The clue is simple: "To reach your destination, follow the rails on Track 24 leads to the secret!"

Decode the given string to reveal the flag:

| li_4WR4_y3sh_TL{et4sdo_hTl0a_cT0hI3@_thl030rrt|

0/10 attempts

Flag Submit

r00t@localhost{Th3_R4il_W4ys_Le4d_T0_Th3_

▼ Byte Buster

I used cipher identifier to find all the possible decoders. Got the flag in the first attempt itself by using brainfuck decoder!

▼ Feedback

▼ Feed back

Filled out the Google form shared on WhatsApp and received the flag.

You've already responded

Thanks for filing out the form! Take the reward:

d0@graham-thomas.com?subject=I%20filled%20out%20this%20form

You can fill out this form only once.

Try contacting the owner of the form if you think this is a mistake.

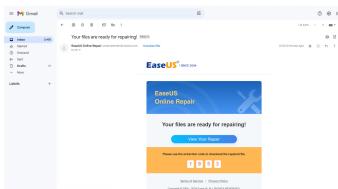
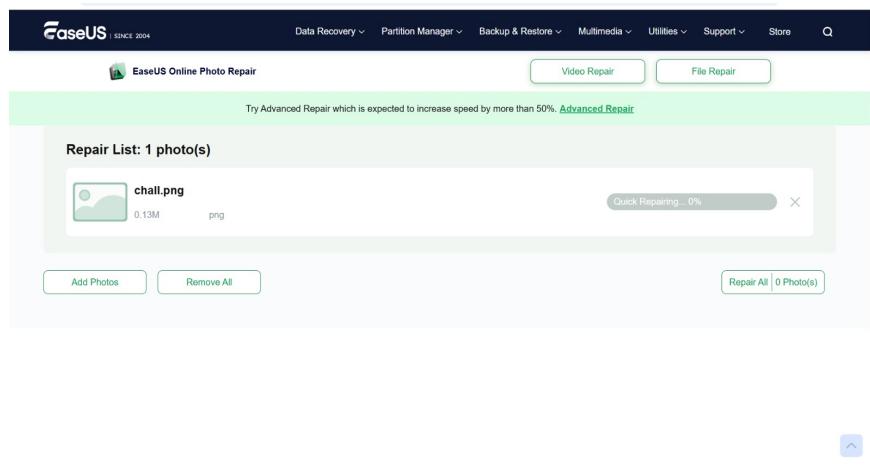
This form was created inside of [Simplenote](#).
Does this form look suspicious? [Report](#)

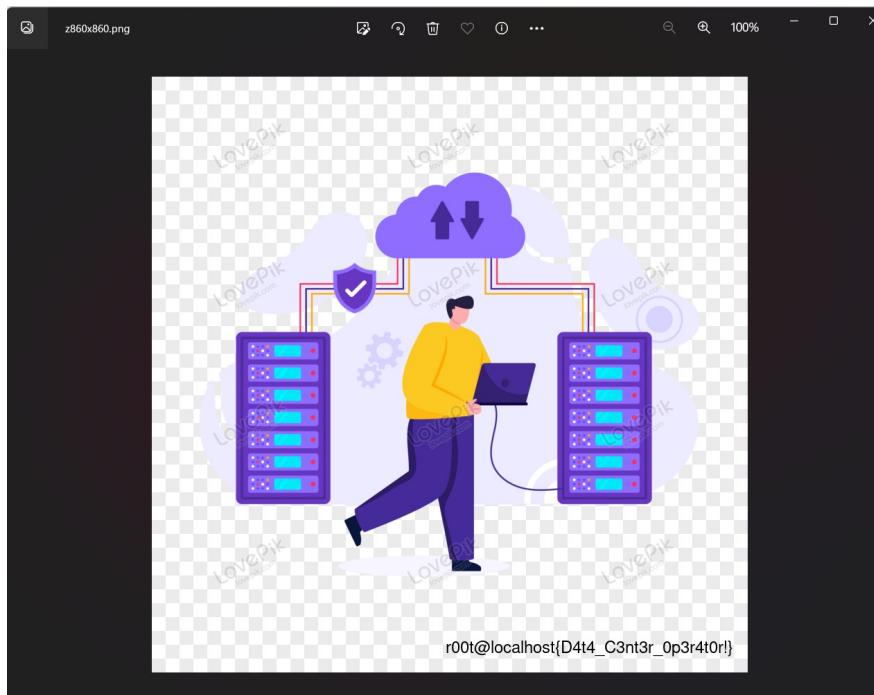
▼ Forensic

▼ Decrypting the Ransom: Malicious DOCM Analysis

▼ EDIT

Since the image wasn't in the correct format, I looked into various tools to correct that, and eventually, I found a for the useful hints given by the organizers.





▼ Avengers

This question was already posted as a blog on a website - <http://www.andynoel.xyz/?p=518> So, I used that to find the flag.

◀ ▶ 🔍 ⌂ Not secure andynoel.xyz/?p=518

甚时跃马归来，
倚栏，迎门，轻笑。
看我红妆，与天不老！

2024 年 12 月

日	一	二	三	四	五	六
1	2	3	4	5	6	7
8	9	10	11	12	13	14
15	16	17	18	19	20	21
22	23	24	25	26	27	28
29	30	31				

+ 11 月

搜索...

近期文章

How did I exploit your desires to implant a Trojan shell?
2024-11-28

2024-11-28

Penetration (9)

- WEB (26)
- WP (23)
- 二进制安全 (2)
- 公考 (1)
- 随笔 (12)

其他操作

- 登录
- 条目 feed
- 评论 feed

Challenge Description

My friend John is an "environmental" activist and a humanitarian. He hated the ideology of Thanos from the Avengers: Infinity War. He sucks at programming. He used too many variables while writing any program. One day, John gave me a memory dump and asked me to find out what he was doing while he took the dump. Can you figure it out for me?

我的朋友约翰是一位“环保”活动家和人道主义者。他讨厌复仇者联盟中灭霸的观点：无限战争。他编程很烂。他在编写任何程序时使用了太多变量。有一天，约翰给了我一个内存转储，并让我找出他在转储时在做什么。你能帮我弄清楚吗？

Progress

整体下来就是一个常规取证思路，先imageinfo看一下：

```
[root@localhost ~]# ./volatility --f=f:\MemLabs\Lab0\Challenge.raw --imagedir=.
Volatility Foundation Volatility Framework 2.6
INFO : volatility.debug = True
INFO : volatility.config = Determining config. Based on 100G search...
Suggested AS file(s):
    1. f:\MemLabs\Lab0\Challenge.raw (Kernel AS)
AS Layer1: IA32PagedMemoryPax (Kernel AS)
AS Layer2: FileAddressSpace (f:\MemLabs\Lab0\Challenge.raw)
PAC Enabled
DTB : 0x185000L
```

b = b + chr(ord(j) ^ i)
print b

```
[kali㉿DESKTOP-F30A7VG:~]# /mnt/f/MemLabs/Lab0
$ python2 xor.py
3]6o]`1v1p
2]7n\@8w@Bq-
1_4m_b3tt3r]
0*51*c2uu2s|
7Y2Kyd5r5t{
6X3jXe4s4uz
5[0i[f7pp7vY
4Z1hZg6qg6wx
:UgUh9-9xw
:T?Et188vv
```

flag{you_are_good_but1_4m_b3tt3r}

▼ Rev

▼ Enigma Unveiled

I opened the rev1 file in Ghidra

root@localhost CTF Write Up

14

I noticed there was a base64 text.

ZjByX3kwdXJfNWVjMG5kX2x1NTVvb191bmJhc2U2NF80bGxfN2gzXzdoMW5nNQ==

I decoded that base64 text and found the flag.

```
root@localhost{f0r_y0ur_5ec0nd_le550n_unbase64_4ll_7h3_7h1ng5}
```

▼ Reverse

I opened chall2 file in Ghidra.

I noticed that every character in a function is in an if condition. So, I entered those characters as passwords and found the flag.

root@localhost{AB12CD34EF56GH7}

▼ Cloud Security

▼ Misconfigured Bucket

bucket name: **ctf-flag-bucket**

file name present in: <https://ctf-flag-bucket.s3.amazonaws.com/>

This XML file does not appear to have any style information associated with it. The document tree is shown below.

```
<ListBucketResult xmlns="http://s3.amazonaws.com/doc/2006-03-01/">
  <Name>ctf-flag-bucket</Name>
  <Prefix/>
  <Marker/>
  <MaxKeys>1000</MaxKeys>
  <IsTruncated>false</IsTruncated>
  <Contents>
    <Key>adkdkjsadlafjlfjdsdkjds1kfjdsgljfjdskjg1kfjdjglkfjdglkjfghjghfbrehgkjrehgjfehgjrehgrhkjhrgfjgn.txt</Key>
    <LastModified>2024-12-07T04:12:37.000Z</LastModified>
    <ETag>"fcfd1d204a22be6b2a4ec92598cd75"</ETag>
    <Size>36</Size>
  </Contents>
</Owner>
<ID>fa3b3417e09697dd3e0552ff0b0684222c5c790ccdf9f3afad819b3a4bf97</ID>
<DisplayName>211001070</DisplayName>
</Owner>
<StorageClass>STANDARD</StorageClass>
</Contents>
</ListBucketResult>
```

<https://ctf-flag-bucket.s3.amazonaws.com/sdskdjsadlafljfljdslkfjdslkfjdslgfjdlskjgkfdjglkfjdglkjfghighfkbrehgkjrehgjfehgjrehgjrehgj>

r00t@localhost{wh0_st0le_my_c00kies}

▼ S3crets

<https://rootatlocalhost.s3.us-east-1.amazonaws.com/flag.txt>

r00t@localhost{s3_bucket_leaked_data}

▼ Cloud Infiltration

The instance is not running now, but I used ChatGPT to connect Kali to the instance earlier.

```
(kali㉿kali)-[~/Downloads]
└─$ aws configure
AWS Access Key ID [*****3NET]:
AWS Secret Access Key [*****BKhz]:
Default region name [us-east-1]:
Default output format [json]:

(kali㉿kali)-[~/Downloads]
└─$ aws iam list-user-policies --user-name ctf --query "PolicyNames[]" --output text
Ec2Access      IAMAccess      S3Access      SSMAccessss

(kali㉿kali)-[~/Downloads]
└─$ aws iam get-user-policy --user-name ctf --policy-name EC2Access
{
    "UserName": "ctf",
    "PolicyName": "EC2Access",
    "PolicyDocument": {
        "Version": "2012-10-17",
        "Statement": [
            {
                "Sid": "VisualEditor0",
                "Effect": "Allow",
                "Action": "ec2:StartInstances",
                "Resource": "arn:aws:ec2:us-east-1:381492101404:instance/i-01664eeeea278b8c48"
            },
            {
                "Sid": "VisualEditor1",
                "Effect": "Allow",
                "Action": [
                    "ec2:DescribeInstances",
                    "ec2:DescribeRegions",
                    "ec2:DescribeInstanceStatus"
                ],
                "Resource": "*"
            }
        ]
    }
}
```

```
[...]
<[MSILBASIC] > /Downloads>
  <#> and describe-instance->id = $4000002700c4 - query [System.Diagnostics.Process]::GetProcessesByName
  <#> and describe-instance->id = $4000002700c4 - query [System.Diagnostics.Process]::GetProcessesByName

  Describereferences
  PrivateSet | PublicIP
  372.35.0.104 | 3.0.0.237.234

<[MSILBASIC] > /Downloads>
  <#> and describe-instance->id = $4000002700c4 - query [System.Diagnostics.Process]::GetProcessesByName
  <#> and describe-instance->id = $4000002700c4 - query [System.Diagnostics.Process]::GetProcessesByName

  An error occurred (AccessDenied) when calling the ListProcessesByName operation: user: arcmin3610:18052018100545
  or its ancestor failed to obtain client/bucket on resource: /arcmin3610/reddispatcher as explicit access is denied.
```

```
Convert Public IP to DNS:  
AWS provides a default DNS name for instances, which is  
formatted like this:  
  
CSS Copy code  
ec2-[PublicIP].compute-1.amazonaws.com  
  
Replace the dots in the Public IP ( 3.95.217.234 ) with dashes,  
and you get:  
  
CSS Copy code  
ec2-3-95-217-234.compute-1.amazonaws.com
```

Alright, now that we've got the address `ec2-3-95-217-234.compute-1.amazonaws.com`, the next thing to try is connecting to it. If you have the private key file, like `remote.pem`, you can use it to attempt an SSH connection. The command would look like this:

```
bash Copy code  
ssh -i "remote.pem" ubuntu@ec2-3-95-217-234.compute-1.amazonaws.com
```

If that works, you'll get into the EC2 instance, and from there, we can start looking for the flag.

If you don't have the key file, we might need to check if there's another way to access the instance. Sometimes, the default usernames like `ubuntu`, `ec2-user`, or `root` might work depending on the setup, or we could explore any credentials shared in the environment.

If the connection doesn't go through, it's worth checking that the key file has the right permissions. Use this command to make it secure:

```
bash Copy code  
chmod 400 remote.pem
```

We should also confirm that the instance is running. You can use:

Alright, once you're logged into the EC2 instance, let's look around to find the flag in the system. Start by listing the files in the current directory:

```
bash Copy code  
ls
```

Found flag.txt inside the instance with the flag!!