CERT-SE-2024 CTF

Update: I got a email from MSB a couple days after the CTF finished saying I got 8/9 flags. The flag missed will be specified at that point in the writeup.

This CTF was hosted by CERT-SE (MSB) in October 2024. It is available at the following link: https://www.cert.se/2024/09/cert-se-ctf2024.html.

From the description we have:

```
<scenario>
A fictional organization has been affected by a ransomware attack. It has been
successful in setting up an emergency channel for communication and has access to
parts of its infrastructure.

Can you find all the flags?
</scenario>
```

There are a total of 9 flags, eight having the format "CTF[FLAG]" and one just in the format "[FLAG]", appending CTF gives you the full flag.

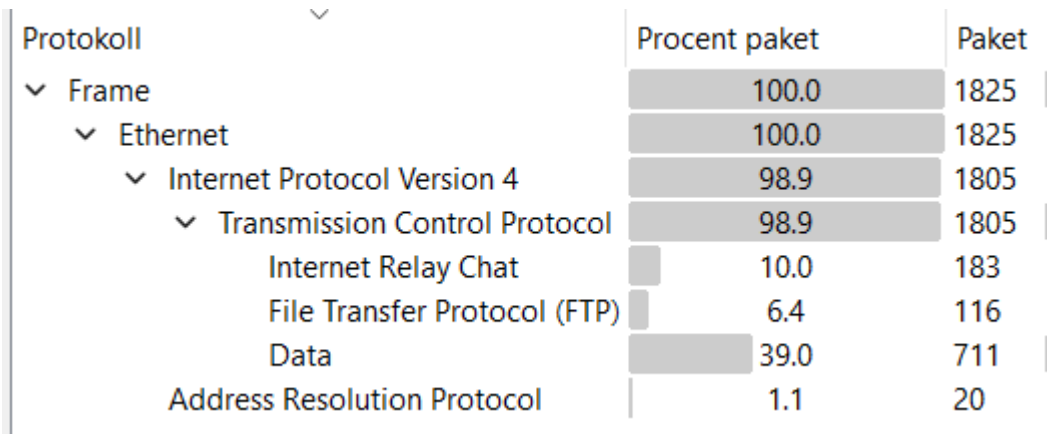
The following tools and programs was mainly used to complete the challenge:

- Wireshark / tshark
- Autopsy (not needed)
- Cyberchef
- Binwalk
- Hashcat

CERT-SE_CTF2024.pcap

Extracting the initial .zip file we are greeted with a pcap file. We open this into wireshark to start analyzing it and looking for clues.

Going through the protocol hierarchy we can see that there appears the be an IRC conversation in place along with that files being transferred using FTP:



Protokoll	Procent paket	Paket
Frame	100.0	1825
Ethernet	100.0	1825
Internet Protocol Version 4	98.9	1805
Transmission Control Protocol	98.9	1805
Internet Relay Chat	10.0	183
File Transfer Protocol (FTP)	6.4	116
Data	39.0	711
Address Resolution Protocol	1.1	20

Lets start by looking over the IRC conversation for any interesting hints.

IRC

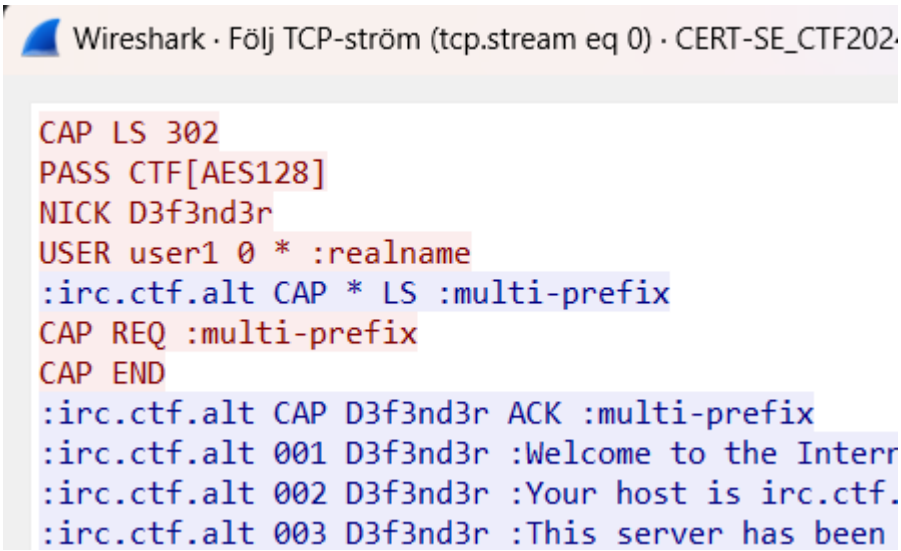
By following the TCP-stream that corresponds to the IRC conversation we can easily examine the entire conversation.

From the IRC conversation we can conclude the following:

- Two users are in the #emergency chanel, that being An4lys3r! and D3f3nd3r.
- The ransomware note has been found on the infected system and sent over IRC as `RANSOM_NOTE.gz`.
- There was two `.pcap` files uploaded to the FTP server containing captured traffic from weeks / days before the attack happened.
- There is one disk file being disused that is also uploaded to the FTP server from one of their clients
- A wordlist was able to be recovered that was created by scraping their public website. It is also uploaded to the FTP server
- Two helping indicators for who was behind the attack was given.
    - First that traffic recorded from a windows workstation `CTF-PC01` was being discussed on a closed forum.
    - Second, the ip-address `195.200.72.82` was involved in C2 and exfiltration activities

This gives us plenty of leads to follow and analyze. Before we continue, in the IRC conversation the first two flags can be found.

The first one can be found at the very start of the IRC conversations TCP-stream, that being `CTF[AES128]`:



Continuing down the conversation we can see a message discussing a "strange string" that their were handed by someone at allsafe. This is the second flag `CTF[E65D46AD10F92508F500944B53168930]`:

Wireshark · Följ TCP-ström (tcp.stream eq 0) · CERT-SE_CTF2024.pcap

```
massive task. You don't happen to have the contact info to Allsafe, I think our rep is called Elliot?
PRIVMSG #emergency :Agree, I already called them in. He is here now sharing some interesting stuff.
PRIVMSG #emergency :They picked up some info on a closed forum regarding our situation, someone posted a
public website. And they where ranting about recording network traffic from a windows workstation called
to the ftp.
WHO #emergency
:irc.ctf.alt 352 D3f3nd3r #emergency ~user1 10.0.0.10 irc.ctf.alt An4lys3r H :0 realname
:irc.ctf.alt 352 D3f3nd3r #emergency ~user1 10.0.0.20 irc.ctf.alt D3f3nd3r H@ :0 realname
:irc.ctf.alt 315 D3f3nd3r #emergency :End of WHO list
PING LAG1727122561261
:irc.ctf.alt PONG irc.ctf.alt :LAG1727122561261
PRIVMSG #emergency :He also handed over a strange looking string CTF[E65D46AD10F92508F500944B53168930],
PING LAG1727122591261
:irc.ctf.alt PONG irc.ctf.alt :LAG1727122591261
:An4lys3r!~user1@10.0.0.10 PRIVMSG #emergency :not really, but why don't you ask john?
```

> MISSED FLAG
>
> The flag in question here is supposed to be decrypted. This was the part I missed when doing the CTF.
>
> The IRC conversations suggest they should ask "John" about it, a direct reference to JohnTheRipper tool. Saving the hash to a file and running john on it while get the flag (or running hashcat with the correct mode).
>
> I'll leave this up for own solving.

With the two flags out of the way lets starts extracting the different files from the traffic.

We start of with the ransom note sent over IRC. To begin we find the packet were it began transferring in the IRC conversation, that being packet nr 157. Going a few packets down we can see a large amount of just pure TCP packets. By following this stream and viewing it in hex, we can see that the first hex characters, `1f 8b`, match the magic bytes for a gzip:

Wireshark · Följ TCP-ström (tcp.stream eq 2) · CERT-SE_CTF2024.pcap

```
00000000  1f 8b 08 08 30 4b d0 66  00 03 77 68 79 5f 6e 6f   ....0K.f ..why_no
00000010  74 00 44 dd 5b 8e 64 39  ae 44 d1 c1 1d 40 bf 04   t.D.[.d9 .D...@..
00000020  04 fe 09 39 ff 69 24 62  19 25 bf ed 38 8d be c5   ...9.i$b .%..8...
00000030  aa 4a 8b 70 37 97 b8 f9  f8 be f7 7f ed b9 3c 8f   .J.p7... ......<.
00000040  67 79 ee df 5f 6c ff d5  62 5a 4c 8b e9 fd fb 5b   gy.._l.. bZL....[
00000050  97 98 e5 7f 2c 31 4b cc  da bf 7f f0 11 73 c4 1c   ....,1K. .....s..
00000060  ff af 23 e6 ec df bf b6  c4 94 98 12 53 fe 42 ed   ..#..... ....S.B.
00000070  f7 87 fa b6 98 2d 66 8b  d9 62 f6 fe fd 91 fd 77   .....-f. .b.....w
```

To save this, we simply show the data in the "RAW" format (binary) and save the file as `RANSOM_NOTE.gz`.

With that, we have gotten everything from the IRC conversation and can move on to analyze the FTP protocol.

FTP

We know that at least 3-4 four files have been uploaded to a FTP server, so lets get them.

By following the TCP streams in Wireshark and incrementing through them we eventually get to the first one, displaying FTP commands:

Wireshark · Följ TCP-ström (tcp.stream eq 3) · CERT-SE_CTF2024.pcap

```
220 INetSim FTP Service ready.
USER anonymous
331 Please specify the password.
PASS NcFTP@
230 Login successful.
PWD
257 "/"
FEAT
500 Unknown command.
HELP SITE
214-The following commands are recognized.
 ABOR CDUP CWD  DELE HELP LIST MKD  MODE
 NLST NOOP PASS PORT PWD  QUIT RETR RMD
 STAT STOR STRU SYST TYPE USER
214 Help OK.
CLNT NcFTPPut 3.2.6 linux-x86_64-libc5
500 Unknown command.
CWD /
250 Directory successfully changed.
TYPE I
200 Switching to BINARY mode.
SIZE corp_net1.pcap
500 Unknown command.
PORT 10,0,0,20,143,63
200 PORT command successful.
STOR corp_net1.pcap
150 Ok to send data.
226 File receive OK.
MDTM 20240904045628 corp_net1.pcap
500 Unknown command.
QUIT
221 Goodbye.
```

At the bottom of the stream we can see the name of the file being uploaded (in this case `corp_net1.pcap`) and that it was successfully transferred. Looking at the next TCP stream (nr 4) in hex view, we can see that the first hex characters, `d4 c3`, matches that of the magic bytes of a .pcap files. This indicates that this is the file uploaded:

Wireshark · Följ TCP-ström (tcp.stream eq 4) · CERT-SE_CTF2024.pcap

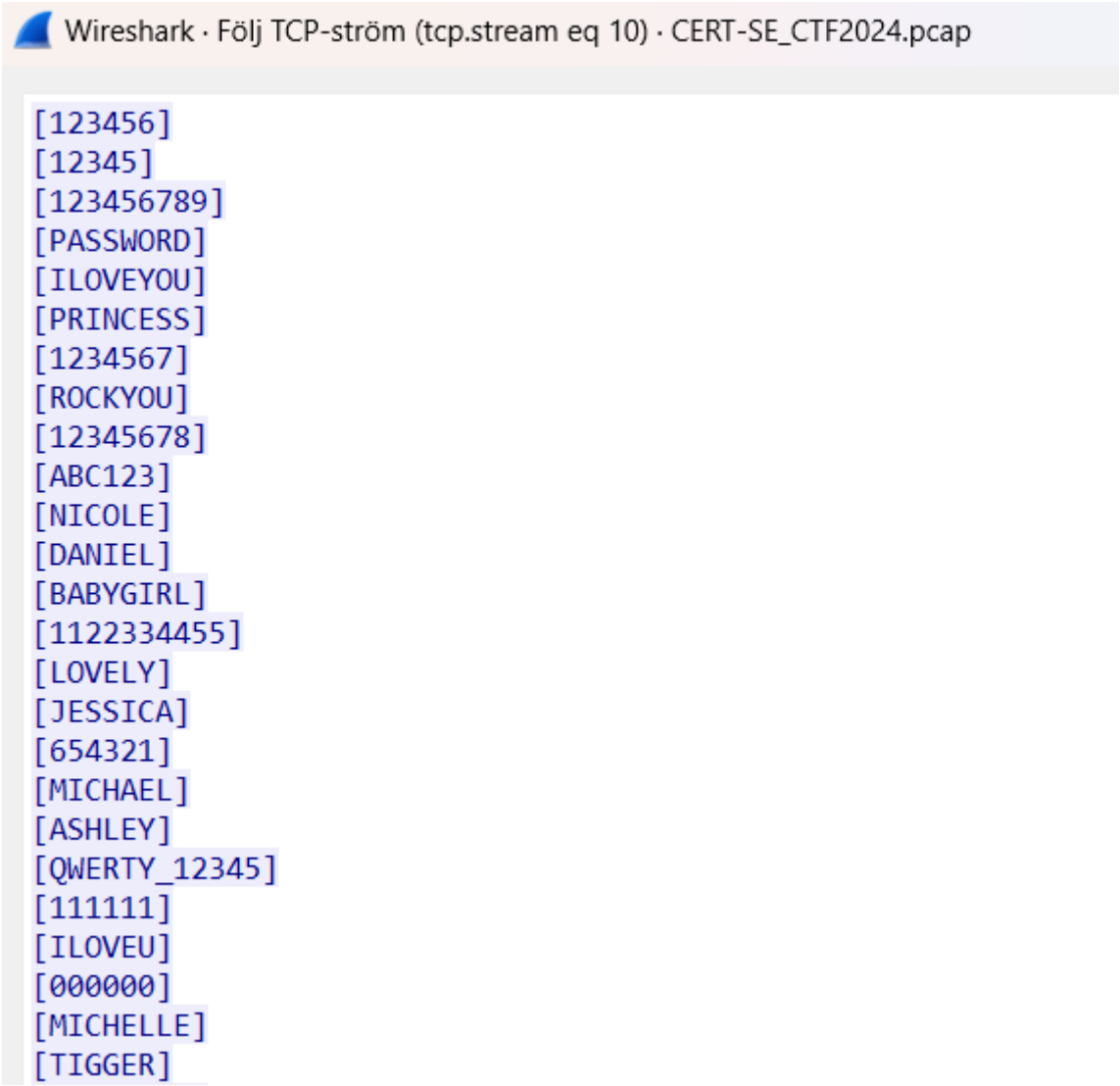
```
00000000  d4 c3 b2 a1 02 00 04 00  00 00 00 00 00 00 00 00   ........ ........
00000010  00 00 04 00 01 00 00 00  cb 2b d7 66 20 82 00 00   ........ .+.f ...
00000020  51 00 00 00 51 00 00 00  52 54 00 1a 9f 00 52 54   Q...Q... RT....RT
00000030  00 11 19 a7 08 00 45 00  00 43 f4 11 40 00 40 11   ......E. .C..@.@.
00000040  b3 28 c0 a8 89 1c c0 a8  89 02 94 cf 00 35 00 2f   .(...... .....5./
00000050  93 b0 09 ff 01 00 00 01  00 00 00 00 00 01 03 77   ........ .......w
00000060  77 77 03 66 72 61 02 73  65 00 00 01 00 01 00 00   ww.fra.s e.......
```

To extract it we do the same method as we did with the ransome note (view as RAW and save as with the corresponding extension).

Navigating through all the remaining TCP streams related to FTP and repeating the same method as show above we end up with the following files:

- corp_net1.pcap
- corp_net2.pcap
- disk.img.gz
- WORDLIST.txt

For clarities sake the WORDLIST.txt file contains strings in the following format `[STRING]`, the exact same as one of the flags:

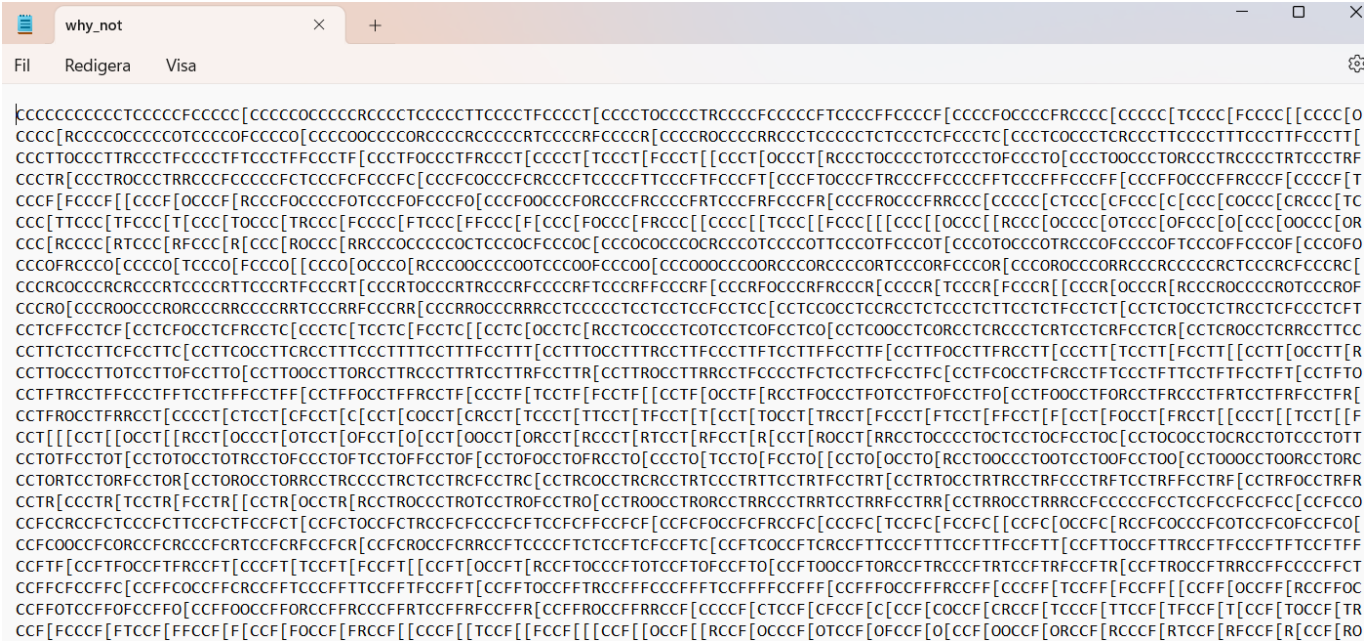


This is most likely going to be used in the future to get the correct flag from this file.

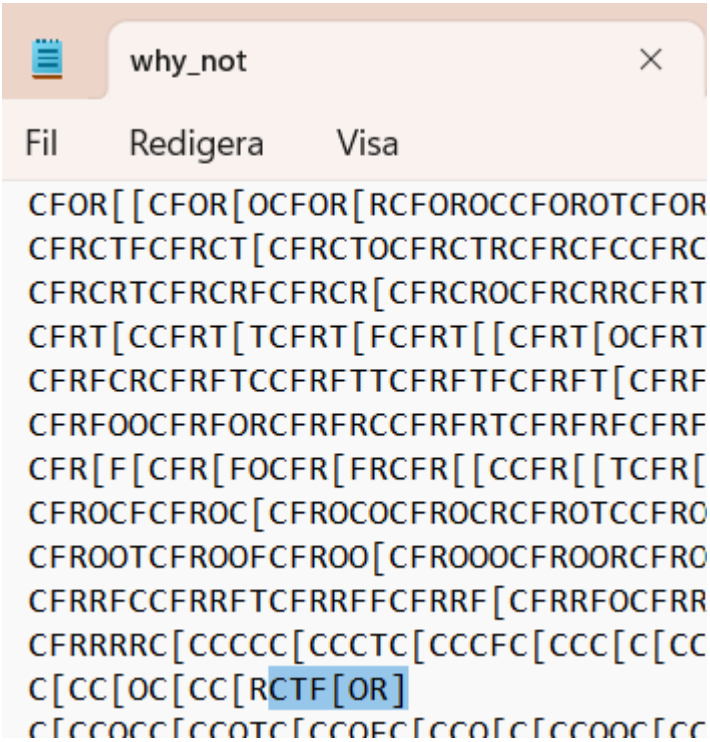
With that we have extracted everything of note and completed the analysis of the initial file. The remaining file can be tackled in any order.

RANSOM_NOTE.gz

To start of we extract the archive to get the contents, inside we can find a single file named `why_not`. Opening it in a text editor gives the following:



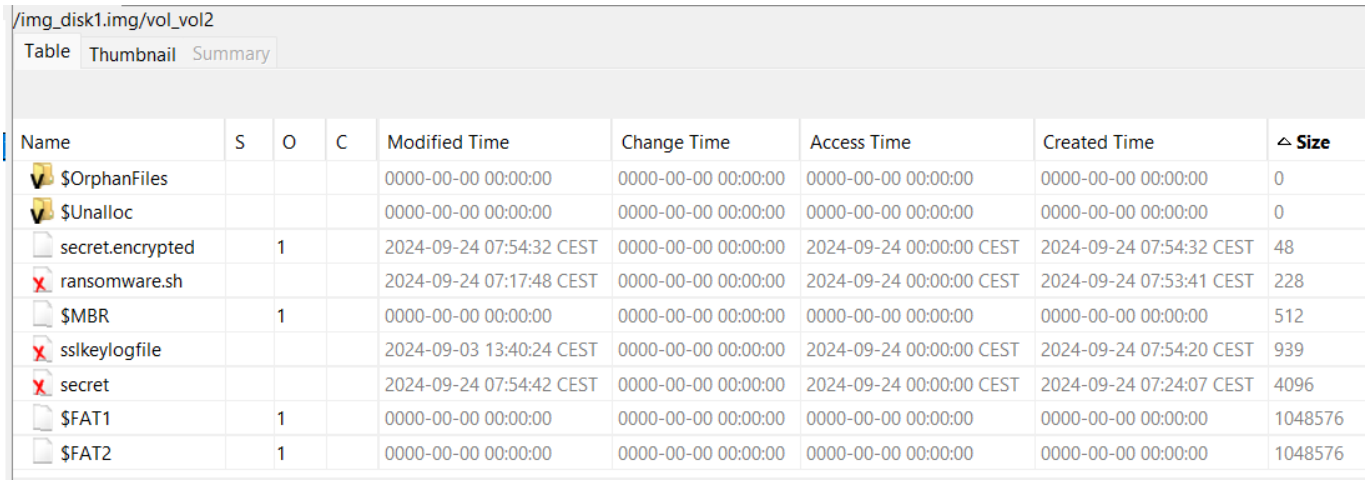
Quickly looking over it we can see a lot of the following character: "C", "T", "F", "O", "R" and "["; further all these characters have the exact same number of occurrences. This is most likely an attempt to obfuscate the flag. Searching for "]" in the files gives us 1 matched character! By either going backwards from this character until the format is correct OR removing all occurrences of characters except the first will give us our third flag, `CTF[OR]`:



disk.img.gz - Part 1

After extracting the .img file we open it in any analysis tool (or mount it). In this example "Autopsy" was used, mainly because I have used it before and had it ready.

After autopsy finishes analyzing the file we can look into the FAT32 file system extracted and note the following files:



/img_disk1.img/vol_vol2

Table | Thumbnail | Summary

Name	S	O	C	Modified Time	Change Time	Access Time	Created Time	Size
$OrphanFiles				0000-00-00 00:00:00	0000-00-00 00:00:00	0000-00-00 00:00:00	0000-00-00 00:00:00	0
$Unalloc				0000-00-00 00:00:00	0000-00-00 00:00:00	0000-00-00 00:00:00	0000-00-00 00:00:00	0
secret.encrypted		1		2024-09-24 07:54:32 CEST	0000-00-00 00:00:00	2024-09-24 00:00:00 CEST	2024-09-24 07:54:32 CEST	48
ransomware.sh				2024-09-24 07:17:48 CEST	0000-00-00 00:00:00	2024-09-24 00:00:00 CEST	2024-09-24 07:53:41 CEST	228
$MBR		1		0000-00-00 00:00:00	0000-00-00 00:00:00	0000-00-00 00:00:00	0000-00-00 00:00:00	512
sslkeylogfile				2024-09-03 13:40:24 CEST	0000-00-00 00:00:00	2024-09-24 00:00:00 CEST	2024-09-24 07:54:20 CEST	939
secret				2024-09-24 07:54:42 CEST	0000-00-00 00:00:00	2024-09-24 00:00:00 CEST	2024-09-24 07:24:07 CEST	4096
$FAT1		1		0000-00-00 00:00:00	0000-00-00 00:00:00	0000-00-00 00:00:00	0000-00-00 00:00:00	1048576
$FAT2		1		0000-00-00 00:00:00	0000-00-00 00:00:00	0000-00-00 00:00:00	0000-00-00 00:00:00	1048576

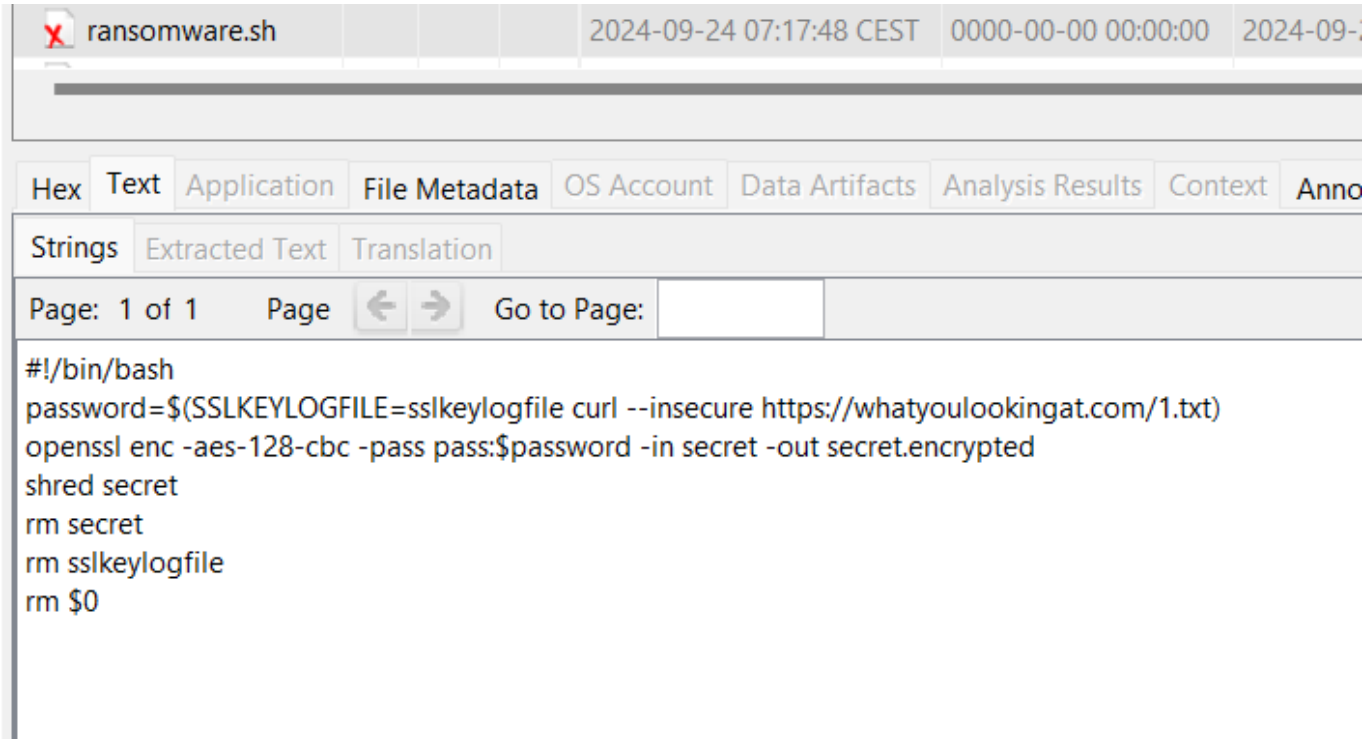
There are a couple out of the ordinary files here:

- secret.encrypted
- ranswomare.sh
- secret
- sslkeylogfile

Lets start by looking at the `ransomware.sh` file that has been deleted.

Ransomware.sh

Opening it in autopsy we can see a short bash script that appears to have been executed:



ransomware.sh 2024-09-24 07:17:48 CEST 0000-00-00 00:00:00 2024-09-

Hex | Text | Application | File Metadata | OS Account | Data Artifacts | Analysis Results | Context | Anno

Strings | Extracted Text | Translation

Page: 1 of 1 Page Go to Page:

```
#!/bin/bash
password=$(SSLKEYLOGFILE=sslkeylogfile curl --insecure https://whatyoulookingat.com/1.txt)
openssl enc -aes-128-cbc -pass pass:$password -in secret -out secret.encrypted
shred secret
rm secret
rm sslkeylogfile
rm $0
```

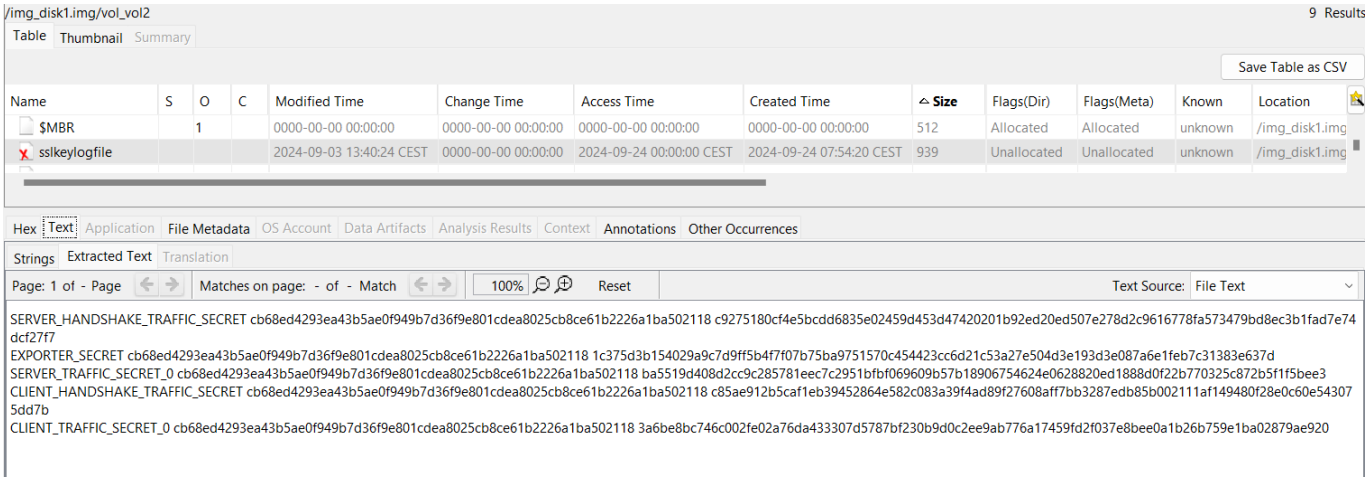
Lets do a quick walkthrough of what it does:

- Starts by declaring the `password` variable
- Sets the content of the local variable `SSLKEYLOGFILE` to the content of the file `sslkeylogfile`
- Sets the password variable to the content of the `https://whatyoulookingat.com/1.txt` file from the cURL request.

- Encrypts the `secret`files using AES-128-CBC with the password gotten from the previous step and saves it to `secret.encrypted`.
- Finally it overwrites the original `secret`file, before deleting it along with `sslkeylogfile` and finally itself.

Looking at the `secret` file we can see that it has %100 been overwritten, and there is no way to recover it in the given time frame. Therefore we need to decrypt the encrypted version, using the correct password gotten from the web resource.

Finally the `sslkeylogfile` contains what seems to be valid SSL session keys. Most likely needed to use to decrypt SSL traffic in the future:

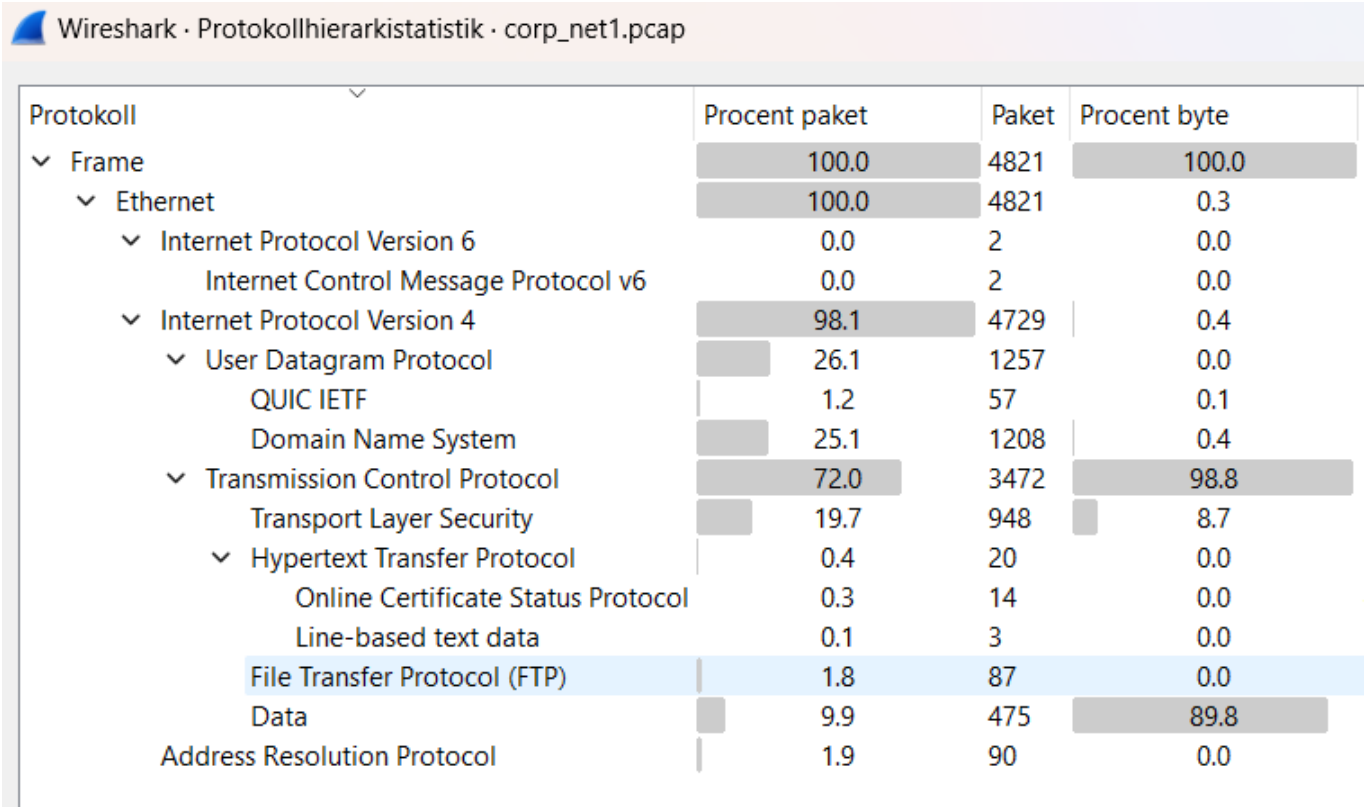


The two files we need for further analysis are `secret.encrypted` and `sslkeylogfile`, we extract them from the disk and keep them in mind.

Since there is nothing we can do unless we get the password we move on to the next file to analysis.

corp_net1.pcap

Opening the file using wireshark we can see that the captured traffic is from 2024-09-03 & 2024-09-04. Looking over the protocol statistics we can see some interesting and of note protocols that are was used:



Wireshark · Protokollhierarkistatistik · corp_net1.pcap

Protokoll	Procent paket	Paket	Procent byte
Frame	100.0	4821	100.0
Ethernet	100.0	4821	0.3
Internet Protocol Version 6	0.0	2	0.0
Internet Control Message Protocol v6	0.0	2	0.0
Internet Protocol Version 4	98.1	4729	0.4
User Datagram Protocol	26.1	1257	0.0
QUIC IETF	1.2	57	0.1
Domain Name System	25.1	1208	0.4
Transmission Control Protocol	72.0	3472	98.8
Transport Layer Security	19.7	948	8.7
Hypertext Transfer Protocol	0.4	20	0.0
Online Certificate Status Protocol	0.3	14	0.0
Line-based text data	0.1	3	0.0
File Transfer Protocol (FTP)	1.8	87	0.0
Data	9.9	475	89.8
Address Resolution Protocol	1.9	90	0.0

The following was noted:

- Once again we have `FTP` traffic, indicating that some files have been transferred.
- We can see `HTTP`traffic, along with `TLS` traffic, indicating that websites have been visited with both encrypted and unencrypted channels.
- A large amount of `DNS` lookups have been made.

To examine further we could also filter out the traffic, so only the packets related to the suspicious IP address (195.200.72.82) or the workstation (CTF-PC01).

FTP

To start of we filter the data so we can get the first packet that is sent using FTP, it is also done from our suspicious IP. If we follow the TCP stream from that packet we can see that a file has been uploaded to the server named `puzzle.exe`:

Wireshark · Följ TCP-ström (tcp.stream eq 59) · corp_net1.pcap

```
220 INetSim FTP Service ready.
USER anonymous
331 Please specify the password.
PASS NcFTP@
230 Login successful.
PWD
257 "/"
FEAT
500 Unknown command.
HELP SITE
214-The following commands are recognized.
 ABOR CDUP CWD  DELE HELP LIST MKD  MODE
 NLST NOOP PASS PORT PWD  QUIT RETR RMD
 STAT STOR STRU SYST TYPE USER
214 Help OK.
CLNT NcFTPPut 3.2.6 linux-x86_64-libc5
500 Unknown command.
CWD /
250 Directory successfully changed.
TYPE I
200 Switching to BINARY mode.
SIZE puzzle.exe
500 Unknown command.
PORT 192,168,137,28,136,69
200 PORT command successful.
STOR puzzle.exe
150 Ok to send data.
226 File receive OK.
MDTM 20240828055906 puzzle.exe
500 Unknown command.
QUIT
221 Goodbye.
```

Looking at the following TCP stream and view it in hex view we can see the starting values, `4D 5A`, matches that of a DOS executable. Here we do the same technique with the first pcap file we analyzed (view as binary and save). Repeating this for all FTP uploads we get the following files:

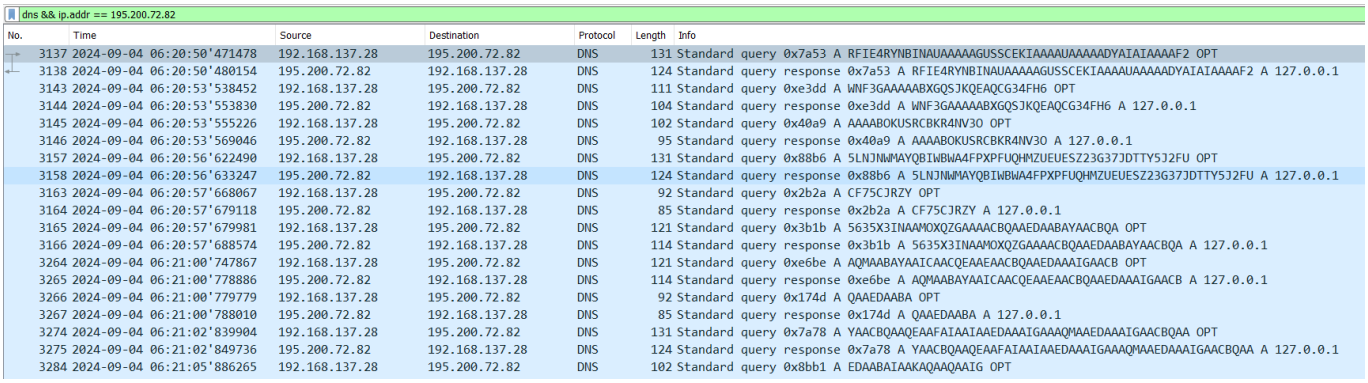
- puzzle.exe
- Recycle-Bin.zip
- archive

We save these and will analyze them later.

DNS

Filtering out the DNS traffic we see nothing out of the ordinary at the start, just usual and legit sites being visited. This is except one, `whatyoulookingat.com`, which we will get to in the next session. After a while we

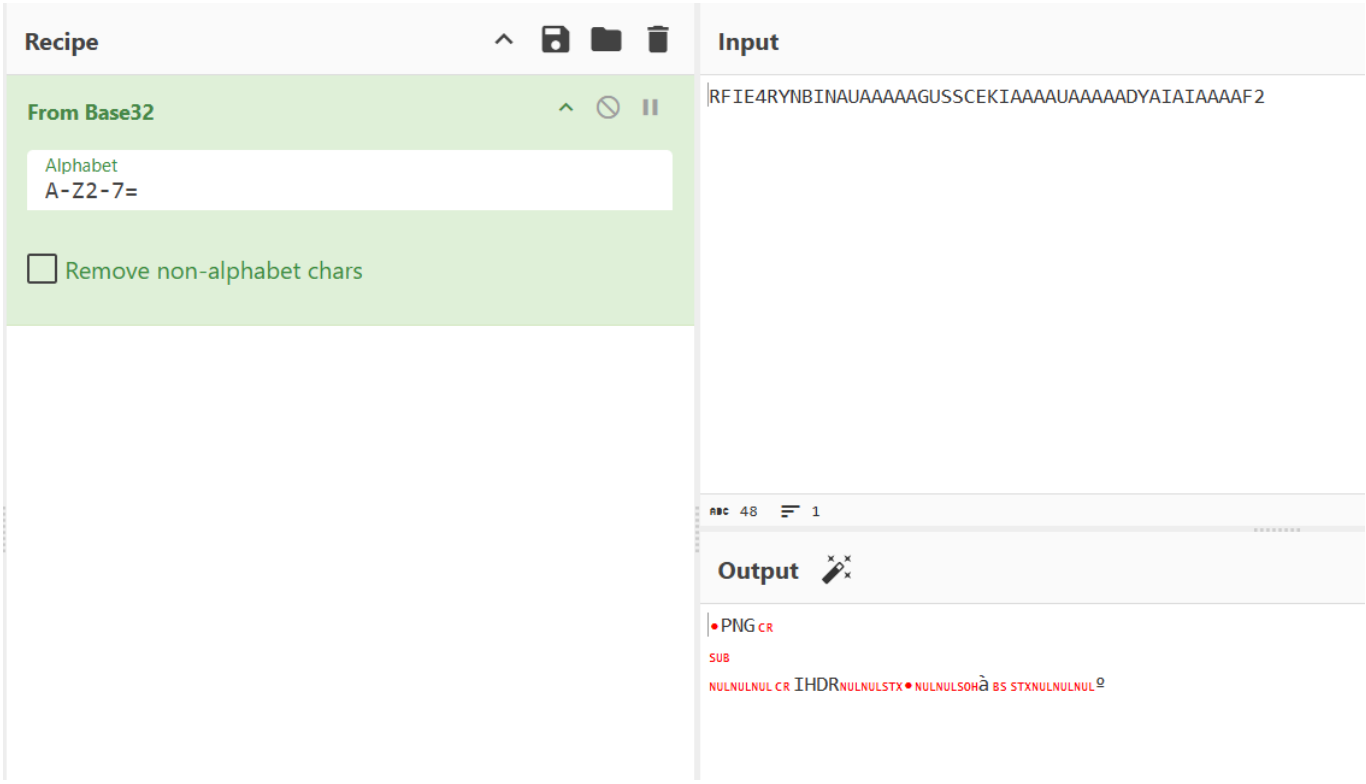
can see queries and responses containing the suspicious IP address,many looking "strange" and not being valid domains:



dns && ip.addr == 195.200.72.82

No.	Time	Source	Destination	Protocol	Length	Info
3137	2024-09-04 06:20:50'471478	192.168.137.28	195.200.72.82	DNS	131	Standard query 0x7a53 A RFIE4RYNBINAUAAAAAGUSSCEKIAAAAUAAAAADYAIAIAAAAF2 OPT
3138	2024-09-04 06:20:50'480154	195.200.72.82	192.168.137.28	DNS	124	Standard query response 0x7a53 A RFIE4RYNBINAUAAAAAGUSSCEKIAAAAUAAAAADYAIAIAAAAF2 A 127.0.0.1
3143	2024-09-04 06:20:53'538452	192.168.137.28	195.200.72.82	DNS	111	Standard query 0xe3dd A WNF3GAAAAABXGQSJKQEAQCG34FH6 OPT
3144	2024-09-04 06:20:53'553830	195.200.72.82	192.168.137.28	DNS	104	Standard query response 0xe3dd A WNF3GAAAAABXGQSJKQEAQCG34FH6 A 127.0.0.1
3145	2024-09-04 06:20:53'555226	192.168.137.28	195.200.72.82	DNS	102	Standard query 0x40a9 A AAAABOKUSRCBKR4NV3O OPT
3146	2024-09-04 06:20:53'569046	195.200.72.82	192.168.137.28	DNS	95	Standard query response 0x40a9 A AAAABOKUSRCBKR4NV3O A 127.0.0.1
3157	2024-09-04 06:20:56'622490	192.168.137.28	195.200.72.82	DNS	131	Standard query 0x88b6 A 5LNJNWMAYQBIWBWA4FPXPFUQHMZUEUESZ23G37JDTTY5J2FU OPT
3158	2024-09-04 06:20:56'633247	195.200.72.82	192.168.137.28	DNS	124	Standard query response 0x88b6 A 5LNJNWMAYQBIWBWA4FPXPFUQHMZUEUESZ23G37JDTTY5J2FU A 127.0.0.1
3163	2024-09-04 06:20:57'668067	192.168.137.28	195.200.72.82	DNS	92	Standard query 0x2b2a A CF75CJRZY OPT
3164	2024-09-04 06:20:57'679118	195.200.72.82	192.168.137.28	DNS	85	Standard query response 0x2b2a A CF75CJRZY A 127.0.0.1
3165	2024-09-04 06:20:57'679981	192.168.137.28	195.200.72.82	DNS	121	Standard query 0x3b1b A 5635X3INAAMOXQZGAAAACBQAAEDAABAYAACBQA OPT
3166	2024-09-04 06:20:57'688574	195.200.72.82	192.168.137.28	DNS	114	Standard query response 0x3b1b A 5635X3INAAMOXQZGAAAACBQAAEDAABAYAACBQA A 127.0.0.1
3264	2024-09-04 06:21:00'747867	192.168.137.28	195.200.72.82	DNS	121	Standard query 0xe6be A AQMAABAYAAICAACQEAAEAACBQAAEDAAAIGAACB OPT
3265	2024-09-04 06:21:00'778886	195.200.72.82	192.168.137.28	DNS	114	Standard query response 0xe6be A AQMAABAYAAICAACQEAAEAACBQAAEDAAAIGAACB A 127.0.0.1
3266	2024-09-04 06:21:00'779779	192.168.137.28	195.200.72.82	DNS	92	Standard query 0x174d A QAAEDAABA OPT
3267	2024-09-04 06:21:00'788010	195.200.72.82	192.168.137.28	DNS	85	Standard query response 0x174d A QAAEDAABA A 127.0.0.1
3274	2024-09-04 06:21:02'839904	192.168.137.28	195.200.72.82	DNS	131	Standard query 0x7a78 A YAACBQAAQEAAFAIAAIAAEDAAAIGAAAQMAAEDAAAIGAACBQAA OPT
3275	2024-09-04 06:21:02'849736	195.200.72.82	192.168.137.28	DNS	124	Standard query response 0x7a78 A YAACBQAAQEAAFAIAAIAAEDAAAIGAAAQMAAEDAAAIGAACBQAA A 127.0.0.1
3284	2024-09-04 06:21:05'886265	192.168.137.28	195.200.72.82	DNS	102	Standard query 0x8bb1 A EDAABAIAAKAQAAQAAIG OPT

By analyzing the first query done of `RFIE4RYNBINAUAAAAAGUSSCEKIAAAAUAAAAADYAIAIAAAAF2` (using a encryption/decoding detection tool) we can see that it is being identified as "Base62" encoding. If we decode it we get the following:

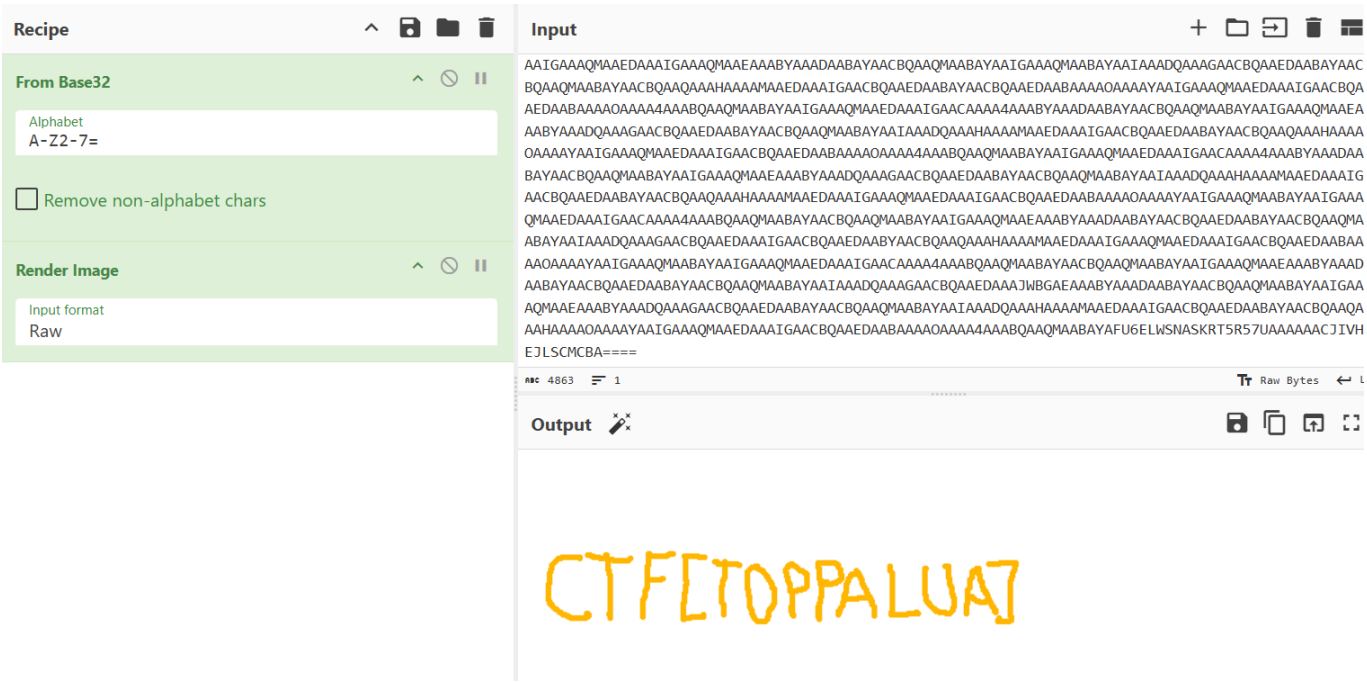


The "png" string returned indicates that this is the start of an image. Trying to render this gives nothing, meaning that we need the remaining parts of the image.

The full image is most likely a combination of all the DNS queries done by this IP, added together, base62 decoded and rendered. By using Tshark we can filter the traffic to only contain DNS, the IP 195.200.72.82 and to finally only print out the DNS name field:

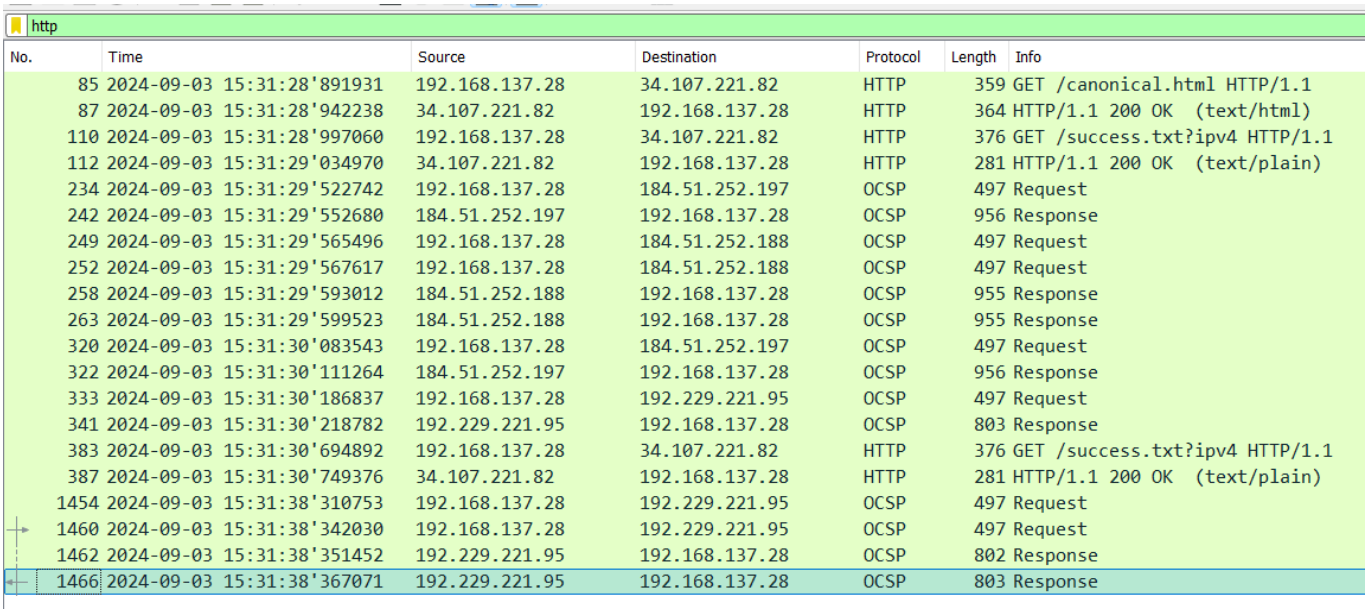
```
tshark -r corp_net1.pcap -Y "ip.dst == 195.200.72.82 and dns" -T fields -e
dns.qry.name
```

We save the output to a file, remove all the newlines, decoding and rendering it gives us the fourth flag `CTF[TOPPALUA]`:



HTTP / TLS traffic

Filtering out HTTP traffic does not yeild a lot.



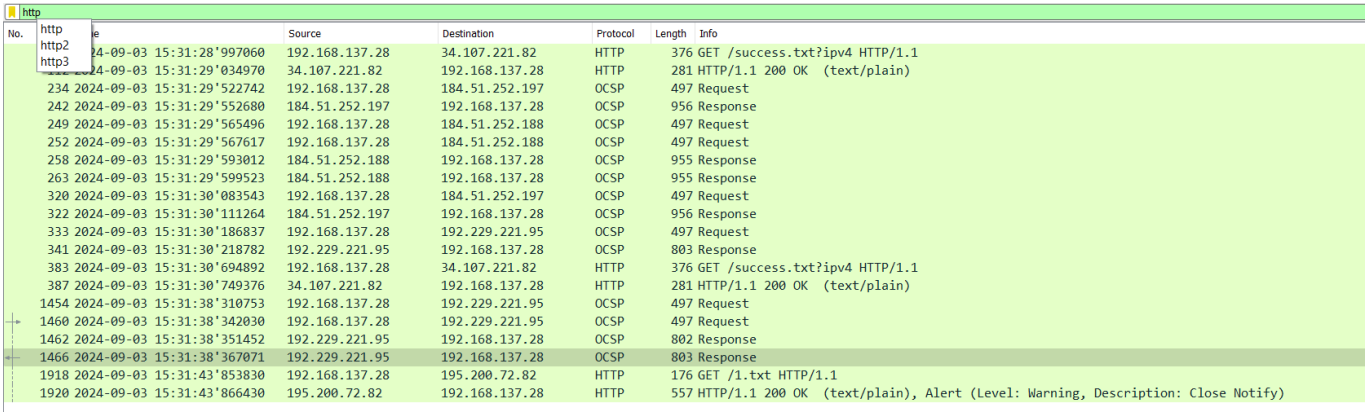
http

No.	Time	Source	Destination	Protocol	Length	Info
85	2024-09-03 15:31:28'891931	192.168.137.28	34.107.221.82	HTTP	359	GET /canonical.html HTTP/1.1
87	2024-09-03 15:31:28'942238	34.107.221.82	192.168.137.28	HTTP	364	HTTP/1.1 200 OK (text/html)
110	2024-09-03 15:31:28'997060	192.168.137.28	34.107.221.82	HTTP	376	GET /success.txt?ipv4 HTTP/1.1
112	2024-09-03 15:31:29'034970	34.107.221.82	192.168.137.28	HTTP	281	HTTP/1.1 200 OK (text/plain)
234	2024-09-03 15:31:29'522742	192.168.137.28	184.51.252.197	OCSP	497	Request
242	2024-09-03 15:31:29'552680	184.51.252.197	192.168.137.28	OCSP	956	Response
249	2024-09-03 15:31:29'565496	192.168.137.28	184.51.252.188	OCSP	497	Request
252	2024-09-03 15:31:29'567617	192.168.137.28	184.51.252.188	OCSP	497	Request
258	2024-09-03 15:31:29'593012	184.51.252.188	192.168.137.28	OCSP	955	Response
263	2024-09-03 15:31:29'599523	184.51.252.188	192.168.137.28	OCSP	955	Response
320	2024-09-03 15:31:30'083543	192.168.137.28	184.51.252.197	OCSP	497	Request
322	2024-09-03 15:31:30'111264	184.51.252.197	192.168.137.28	OCSP	956	Response
333	2024-09-03 15:31:30'186837	192.168.137.28	192.229.221.95	OCSP	497	Request
341	2024-09-03 15:31:30'218782	192.229.221.95	192.168.137.28	OCSP	803	Response
383	2024-09-03 15:31:30'694892	192.168.137.28	34.107.221.82	HTTP	376	GET /success.txt?ipv4 HTTP/1.1
387	2024-09-03 15:31:30'749376	34.107.221.82	192.168.137.28	HTTP	281	HTTP/1.1 200 OK (text/plain)
1454	2024-09-03 15:31:38'310753	192.168.137.28	192.229.221.95	OCSP	497	Request
1460	2024-09-03 15:31:38'342030	192.168.137.28	192.229.221.95	OCSP	497	Request
1462	2024-09-03 15:31:38'351452	192.229.221.95	192.168.137.28	OCSP	802	Response
1466	2024-09-03 15:31:38'367071	192.229.221.95	192.168.137.28	OCSP	803	Response

If we recall the information we got from `disk.img` that was present in the ransomware, we know a HTTP request (through cURL) was done to the domain `whatyoulookingat.com` with the SSL session keys that we have. Further the date of the current traffic also corresponds to that of the modification that of the `sslkeylogfile`.

If there was a HTTP request done, there is a high chance for a DNS lookup made to the domain. From looking at the DNS traffic in the previous section we saw that there had been mentions of the domain in the DNS records.

This confirms that the request done is most likely encrypted and we need to use the SSL session keys the decrypt the traffic. To do this we import the file in Wireshark (preferences -> protocols -> TLS) and reload. Once again we filter for HTTP and we can now see the request present in the ransomware:



Looking at the response we see the following:

```
HTTP/1.1 200 OK
Date: Tue, 03 Sep 2024 15:31:43 GMT
Content-Length: 17
Connection: Close
Server: INetSim HTTPs Server
Content-Type: text/plain

pheiph0Xeiz8OhNa
```

With that we have the password, "pheiph0Xeiz8OhNa" ,used to encrypt the `secret` file and can decrypt it.

disk.img - Part 2

Back to `sceret.encryped` we no have the password and can decrypt it. Since we know from the ransomware that it was encrypted using OpenSSL we can decrypt it the same way with the following command:

```
openssl enc -aes-128-cbc -pass pass:pheiph0Xeiz8OhNa -in secret.encrypted -out
secret.decrypted
```

Executing this in a terminal we get no warning of bad decrypt, indicating the password was correct and reading the output gives us our fifth flag `CTF[OPPORTUNISTICALLY]`:

```
love@:~$ openssl enc -d -aes-128-cbc -pass pass:pheiph0Xeiz8OhNa -in secret.encrypted -out secret.decrypted
*** WARNING : deprecated key derivation used.
Using -iter or -pbkdf2 would be better.
love@:~$ cat secret.decrypted
CTF[OPPORTUNISTICALLY]
love@:~$
```

With this the disk is done and we can move on.

corp_net2.pcap

Looking over this traffic capture we can see that it dates from earlier that the first one (2024-08-29) and contains less amount of packets in total. When viewing the protocol hierarchy we can see that it is mainly encrypted protocols, not good for us:

Protokoll	Procent paket	Paket	Procent byte	Byte	Bitar/s	Slutpaket	Slutbyte	Slutbit/s
Frame	100.0	13869	100.0	9467076	756k	0	0	0
Ethernet	100.0	13869	2.1	194166	15k	0	0	0
Internet Protocol Version 4	96.7	13414	2.8	268280	21k	0	0	0
Transmission Control Protocol	80.6	11177	83.8	7930067	633k	6489	6043486	482k
Transport Layer Security	31.8	4410	78.5	7433409	593k	4349	7105788	567k
VSS Monitoring Ethernet trailer	1.4	192	0.0	384	30	192	384	30
Malformed Packet	0.6	87	0.0	0	0	87	0	0
Data	0.2	22	0.0	22	1	22	22	1
Kerberos	0.1	14	0.1	9143	730	14	9143	730
Distributed Computing Environment / Remote Procedure Call (DCE/RPC)	0.1	14	0.0	4316	344	6	2704	215
DRSUAPI	0.0	6	0.0	624	49	6	624	49
DCE/RPC Endpoint Mapper	0.0	2	0.0	292	23	2	292	23
Apache JServ Protocol v1.3	0.1	8	0.3	25265	2017	8	25265	2017
Hypertext Transfer Protocol	0.0	2	0.0	653	52	2	653	52
User Datagram Protocol	15.7	2178	0.2	17424	1391	0	0	0
QUIC IETF	8.0	1107	9.5	901721	72k	1093	888454	70k
Malformed Packet	0.0	4	0.0	0	0	4	0	0
Domain Name System	6.7	923	0.9	88060	7032	923	88060	7032
Multicast Domain Name System	0.5	67	0.0	2488	198	67	2488	198
Link-local Multicast Name Resolution	0.5	63	0.0	2280	182	63	2280	182
Data	0.1	16	0.0	64	5	16	64	5
Simple Service Discovery Protocol	0.1	8	0.0	1400	111	8	1400	111
NetBIOS Name Service	0.0	4	0.0	224	17	4	224	17
Internet Control Message Protocol	0.4	59	0.1	8258	659	0	0	0
Domain Name System	0.4	59	0.1	6134	489	59	6134	489
Internet Protocol Version 6	2.3	314	0.1	12560	1003	0	0	0
Transmission Control Protocol	1.3	182	0.3	30708	2452	90	3624	289
NetBIOS Session Service	0.6	84	0.2	22741	1816	0	0	0
SMB2 (Server Message Block Protocol version 2)	0.6	82	0.2	22267	1778	82	22267	1778
SMB (Server Message Block Protocol)	0.0	2	0.0	138	11	2	138	11
Hypertext Transfer Protocol	0.1	8	0.0	3943	314	6	1688	134
Line-based text data	0.0	2	0.0	1599	127	2	1793	143
User Datagram Protocol	0.9	129	0.0	1032	82	0	0	0
Multicast Domain Name System	0.5	67	0.0	2488	198	67	2488	198
Link-local Multicast Name Resolution	0.4	62	0.0	2236	178	62	2236	178
Internet Control Message Protocol v6	0.0	3	0.0	72	5	3	72	5
Address Resolution Protocol	1.0	141	0.1	6396	510	141	6396	510

We have some interesting though that might be worth analyzing:

- HTTP
- SMB/SMB2
- Kerberos

Lets start with the HTTP traffic. We can see one GET request has been done and multiple "PROPFIND" request being done.

http

No.	Time	Source	Destination	Protocol	Length	Info
5528	2024-08-29 17:13:41'865099	192.168.200.52	34.149.169.35	HTTP	516	GET / HTTP/1.1
5531	2024-08-29 17:13:41'870424	34.149.169.35	192.168.200.52	HTTP	245	HTTP/1.1 301 Moved Permanently
10599	2024-08-29 17:14:16'810021	fe80::f786:f9ae:ab5…	fe80::dcf6:cc94:c4f…	HTTP	203	OPTIONS /share HTTP/1.1
10602	2024-08-29 17:14:16'820487	fe80::dcf6:cc94:c4f…	fe80::f786:f9ae:ab5…	HTTP	294	HTTP/1.1 200 OK
10620	2024-08-29 17:14:16'936635	fe80::f786:f9ae:ab5…	fe80::dcf6:cc94:c4f…	HTTP	233	PROPFIND /share HTTP/1.1
10626	2024-08-29 17:14:16'949065	fe80::dcf6:cc94:c4f…	fe80::f786:f9ae:ab5…	HTTP	92	HTTP/1.1 401 Unauthorized (text/html)
10637	2024-08-29 17:14:16'953826	fe80::f786:f9ae:ab5…	fe80::dcf6:cc94:c4f…	HTTP	316	PROPFIND /share HTTP/1.1 , NTLMSSP_NEGOTIATE
10639	2024-08-29 17:14:16'967032	fe80::dcf6:cc94:c4f…	fe80::f786:f9ae:ab5…	HTTP	871	HTTP/1.1 401 Unauthorized , NTLMSSP_CHALLENGE (text/html)
10640	2024-08-29 17:14:16'968591	fe80::f786:f9ae:ab5…	fe80::dcf6:cc94:c4f…	HTTP	860	PROPFIND /share HTTP/1.1 , NTLMSSP_AUTH, User: LAB\CTF
10642	2024-08-29 17:14:16'983408	fe80::dcf6:cc94:c4f…	fe80::f786:f9ae:ab5…	HTTP	226	HTTP/1.1 200 OK

Further analysis of these request shows that these are authentication attempts to a WebDav share. The authentication method used is NTLMSSP. If we filter the traffic to match NTLMSSP we get the earlier mentioned SMB2 traffic:

ntlmssp

No.	Time	Source	Destination	Protocol	Length	Info
10248	2024-08-29 17:14:14'038434	fe80::f786:f9ae:ab5…	fe80::dcf6:cc94:c4f…	SMB2	240	Session Setup Request, NTLMSSP_NEGOTIATE
10250	2024-08-29 17:14:14'043830	fe80::dcf6:cc94:c4f…	fe80::f786:f9ae:ab5…	SMB2	412	Session Setup Response, Error: STATUS_MORE_PROCESSING_REQUIRED, NTLMSSP_CHALLENGE
10251	2024-08-29 17:14:14'044291	fe80::f786:f9ae:ab5…	fe80::dcf6:cc94:c4f…	SMB2	681	Session Setup Request, NTLMSSP_AUTH, User: LAB\CTF
10287	2024-08-29 17:14:14'188585	fe80::f786:f9ae:ab5…	fe80::dcf6:cc94:c4f…	SMB2	240	Session Setup Request, NTLMSSP_NEGOTIATE
10288	2024-08-29 17:14:14'194702	fe80::dcf6:cc94:c4f…	fe80::f786:f9ae:ab5…	SMB2	412	Session Setup Response, Error: STATUS_MORE_PROCESSING_REQUIRED, NTLMSSP_CHALLENGE
10290	2024-08-29 17:14:14'195645	fe80::f786:f9ae:ab5…	fe80::dcf6:cc94:c4f…	SMB2	681	Session Setup Request, NTLMSSP_AUTH, User: LAB\CTF
10316	2024-08-29 17:14:14'230719	fe80::f786:f9ae:ab5…	fe80::dcf6:cc94:c4f…	SMB2	240	Session Setup Request, NTLMSSP_NEGOTIATE
10317	2024-08-29 17:14:14'233878	fe80::dcf6:cc94:c4f…	fe80::f786:f9ae:ab5…	SMB2	412	Session Setup Response, Error: STATUS_MORE_PROCESSING_REQUIRED, NTLMSSP_CHALLENGE
10318	2024-08-29 17:14:14'234325	fe80::f786:f9ae:ab5…	fe80::dcf6:cc94:c4f…	SMB2	681	Session Setup Request, NTLMSSP_AUTH, User: LAB\CTF
10424	2024-08-29 17:14:16'385582	fe80::f786:f9ae:ab5…	fe80::dcf6:cc94:c4f…	SMB2	240	Session Setup Request, NTLMSSP_NEGOTIATE
10425	2024-08-29 17:14:16'390096	fe80::dcf6:cc94:c4f…	fe80::f786:f9ae:ab5…	SMB2	412	Session Setup Response, Error: STATUS_MORE_PROCESSING_REQUIRED, NTLMSSP_CHALLENGE
10426	2024-08-29 17:14:16'390639	fe80::f786:f9ae:ab5…	fe80::dcf6:cc94:c4f…	SMB2	681	Session Setup Request, NTLMSSP_AUTH, User: LAB\CTF
10454	2024-08-29 17:14:16'448260	fe80::f786:f9ae:ab5…	fe80::dcf6:cc94:c4f…	SMB2	240	Session Setup Request, NTLMSSP_NEGOTIATE

Here we can see multiple failed attempts to connect to the share using the "LAB\CTF" user. Since SMB2 can use NTLMv2 as authentication and we have all the necessary parts to reconstruct the hash, we could extract it and try to crack it.

You could manually recreate the hashes by analyzing the request and response, or use a CLI scripts (such as the following) to extract them. In this case I went with NTLMRawUnHide script. Running the script on the file extracts multiple hashes:

```
Searching corp_net2.pcap for NTLMv2 hashes...

Found NTLMSSP Message Type 1 : Negotiation

Found NTLMSSP Message Type 2 : Challenge
    > Server Challenge       : 60d718c41edbd4c4

Found NTLMSSP Message Type 3 : Authentication
    > Domain                 : LAB
    > Username               : CTF
    > Workstation            : CTF-PC01

NTLMv2 Hash recovered:
CTF::LAB:60d718c41edbd4c4:427c20311f93ee56cf8723011f179bb8:0101000000000000042eb8f48fada018c77116ffd11b6250000000002000
8004d00550044004a0001001e00570049004e002d00350046005600370004d004e00550042004100300004b0004003400570049004e002d00350046005
60037004d004e00550042004100300004b002e004d00550044004a002e004c004f00430041004c00030014004d00550044004a002e004c004f0043004
1004c00050014004d00550044004a002e004c004f00430041004c00070008000042eb8f48fada0106000400020000000800300030000000000000000
000000000200000fd6f62357ae25314e0b8f63ab17e5d1821479e68ec22e7ee70827eadb2f393200a0010000000000000000000000000000000000000
900140063006900660073002f00640063006300300031000000000000000000

Found NTLMSSP Message Type 1 : Negotiation

Found NTLMSSP Message Type 2 : Challenge
    > Server Challenge       : 863f7c84e4d8402d
```

Do get this to a format suitable for cracking we run the following `python3 NTLMRawUnHide.py -i corp_net2.pcap -o hashes -q`, which saves the output to `hashes`.

Finally we can try to crack them using ex. hashcats. Since we got a wordlist (`WORDLIST.txt`) from the first step, lets first try running this list against them. After executing hashcat with the appropriate mode (5600), the hashes and the wordlist, we get that hashcat has successfully cracked the hash. Showing the result gives us or sixth flag by appending "CTF" to the password, `CTF[RHODE_ISLAND_Z]`:

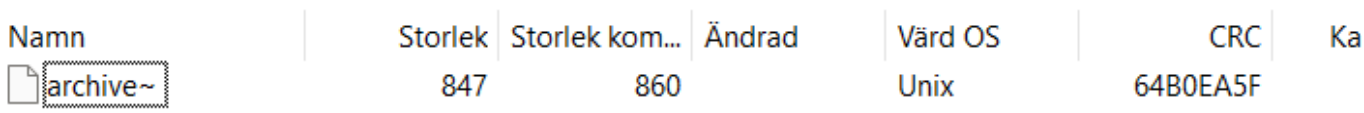
```
Session..........: hashcat
Status...........: Cracked
Hash.Mode........: 5600 (NetNTLMv2)
Hash.Target......: hashes
Time.Started.....: Mon Oct 28 21:46:44 2024, (0 secs)
Time.Estimated...: Mon Oct 28 21:46:44 2024, (0 secs)
Kernel.Feature...: Pure Kernel
Guess.Base.......: File (WORDLIST.txt)
Guess.Queue......: 1/1 (100.00%)
Speed.#1.........:    72665 H/s (0.29ms) @ Accel:256 Loops:1 Thr:1 Vec:8
Recovered........: 13/13 (100.00%) Digests (total), 13/13 (100.00%) Digests (new), 13/13 (100.00%) Salts
Progress.........: 2600/2600 (100.00%)
Rejected.........: 0/2600 (0.00%)
Restore.Point....: 0/200 (0.00%)
Restore.Sub.#1...: Salt:12 Amplifier:0-1 Iteration:0-1
Candidate.Engine.: Device Generator
Candidates.#1....: [123456] -> [SEPTEMBER]

Started: Mon Oct 28 21:46:42 2024
Stopped: Mon Oct 28 21:46:46 2024
love@:~/hashcat$ ./hashcat -a 0 -m 5600 hashes WORDLIST.txt --force --show
CTF::LAB:60d718c41edbd4c4:427c20311f93ee56cf8723011f179bb8:0101000000000000042eb8f48fada018c77116ffd11b6250000000002000
8004d00550044004a0001001e00570049004e002d003500460056003700 4d004e0055004200410030004b0004003400570049004e002d00350046005
60037004d004e0055004200410030004b002e004d00550044004a002e004c004f0043004 1004c00300140004d00550044004a002e004c004f0043004 1004c00500140004d00550044004a002e004c004f0043004 1004c00700080000042eb8f48fada0106000400020000000800300030000000000000000
0000000002000 0fd6f62357ae25314e0b8f63ab17e5d1821479e68ec22e7ee70827eadb2f393200a00100000000000000000000000000000000000000
900140063006900660073002f00640063006300300031000000000000000:[RHODE_ISLAND_Z]
```

An alternative way of solving this found afterwards was extracting the kerberos credentials present through the use of ex. Networkminer. Afterwards this can be cracked, which results in the same flag:

```
Session..........: hashcat
Status...........: Cracked
Hash.Mode........: 19900 (Kerberos 5, etype 18, Pre-Auth)
Hash.Target......: $krb5pa$18$CTF$LAB.LOCAL$1bbdb63ecab6986bdf7e3f4f8c...e94566
Time.Started.....: Mon Oct 28 21:55:50 2024, (1 sec)
Time.Estimated...: Mon Oct 28 21:55:51 2024, (0 secs)
Kernel.Feature...: Pure Kernel
Guess.Base.......: File (WORDLIST.txt)
Guess.Queue......: 1/1 (100.00%)
Speed.#1.........:      183 H/s (0.75ms) @ Accel:256 Loops:64 Thr:1 Vec:8
Recovered........: 1/1 (100.00%) Digests (total), 1/1 (100.00%) Digests (new)
Progress.........: 200/200 (100.00%)
Rejected.........: 0/200 (0.00%)
Restore.Point....: 0/200 (0.00%)
Restore.Sub.#1...: Salt:0 Amplifier:0-1 Iteration:4032-4095
Candidate.Engine.: Device Generator
Candidates.#1....: [123456] -> [SEPTEMBER]

Started: Mon Oct 28 21:55:41 2024
Stopped: Mon Oct 28 21:55:52 2024
love@:~/hashcat$ ./hashcat -a 0 -m 19900 '$krb5pa$18$CTF$LAB.LOCAL$1bbdb63ecab6986bdf7e3f4f8c635a9ea69adca80f81fc56d50ba
384db4270fb994653eade08e0e79b5d7ed45f8ae3be7e89fa753be94566' WORDLIST.txt --force --show
$krb5pa$18$CTF$LAB.LOCAL$1bbdb63ecab6986bdf7e3f4f8c635a9ea69adca80f81fc56d50ba384db4270fb994653eade08e0e79b5d7ed45f8ae3b
e7e89fa753be94566:[RHODE_ISLAND_Z]
love@:~/hashcat$
```

archive

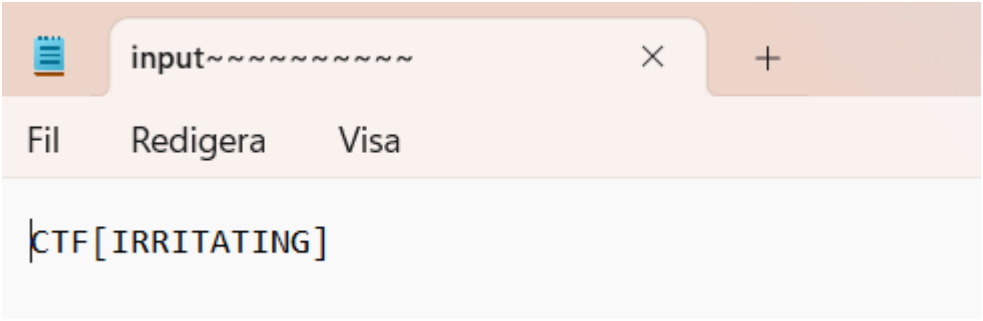
From `corp_net1.pcap` we acquired a file called `archive`. Because of the name lets try opening it using an archive program such as 7zip.



Namn	Storlek	Storlek kom...	Ändrad	Värd OS	CRC	Ka
archive~	847	860		Unix	64B0EA5F	

Inside it is another file, `archive~`. This is also an archive, interesting. Opening this file also contains a file, `archive~~`, which is yet another archive.

This repetition goes on for a while. In the archive tool we continue to navigate into the constantly new archives, after a while the file name changes to `input`. After a while we reach `input~~~~~~~~~~` which prompts us to open it with another program. Opening it using a text editor shows us the content and gives us the seventh flag, `CTF[IRRITATING]`:



Recycle-Bin.zip

Lets move on the next file gotten from `corp_net1.pcap`. We start of by unziping the file to extract its contents. Navigating further into the folders, we finally get to the user SID folder, that contains a bunch of deleted files:

$RHPN5NJ	2024-10-29 11:23	Filmapp	
$I0BN3IA.pdf	2024-08-26 19:56	Adobe Acrobat-d...	1 kB
$I0DKZRU.txt	2024-08-26 19:39	Textdokument	1 kB
$I0G1V6C.txt	2024-08-26 19:56	Textdokument	1 kB
$I0P8KGD.txt	2024-08-26 19:51	Textdokument	1 kB
$I0S90LB.txt	2024-08-26 19:56	Textdokument	1 kB
$I1APZ4K.txt	2024-08-26 19:39	Textdokument	1 kB
$I1E5HE8.txt	2024-08-26 19:39	Textdokument	1 kB
$I1QVT66.txt	2024-08-26 19:56	Textdokument	1 kB
$I2UBK07.jpg	2024-08-26 19:51	JPG-fil	1 kB
$I3K0VQX.txt	2024-08-26 19:56	Textdokument	1 kB
$I3NTYNY.txt	2024-08-26 19:38	Textdokument	1 kB
$I3O9S42.txt	2024-08-26 19:56	Textdokument	1 kB
$I3Q2KVM.txt	2024-08-26 19:56	Textdokument	1 kB
$I3YRO5G.txt	2024-08-26 19:56	Textdokument	1 kB
$I5JW396.txt	2024-08-26 19:56	Textdokument	1 kB
$I5QEV86.jpg	2024-08-26 19:56	JPG-fil	1 kB
$I5Y6MFY.txt	2024-08-26 19:56	Textdokument	1 kB

Many of the files are just 1 kB in size and contains the path to the original file as it content:

```
# Example
C:\Users\aleksej.pazjitnov\Documents\Min_Biografi_del_67.txt
```

Lets start by analyzing the files with actual content in them, these includes various PDFs, images, two archives and one executable. The executable when opened shows it being `HFS - HTTP File Server`, this is most likely a program used in helping with exfiltration and persistence on the attacked system.

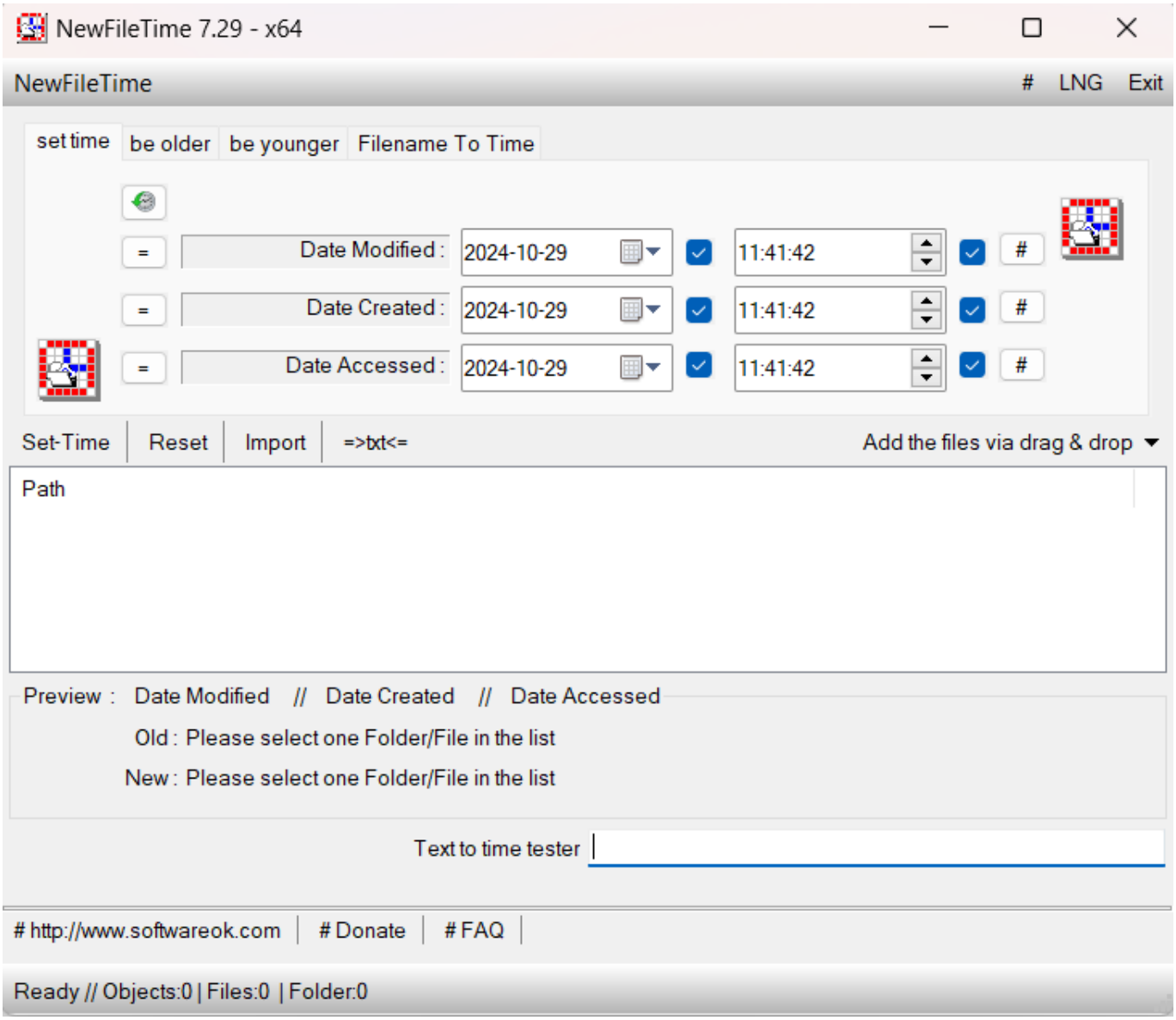
Analyzing the images and PDFs using techniques such as binarywalking, extracting strings and steganography gives nothing. Meaning these files are most likely red hearings.

Further there are two archives present, `$RZ8ZSGB.zip` and `$RCXHUFJ.zip`, the latter being much larger in size than the former. Lets start by looking at the former, inside is a folder named `Bioagrafi` that contains a bunch of .txt files. All are named `Min_Biografi_del_XX.txt`, with XX being a number up till 69. Opening the first one gives us `Lorem`, and further analysis of the other parts shows that this is just the "Lorem Ipsum" phrase that has been split up.

Inside the other archive we find the following, a text file and executable:

Namn	Storlek	Storlek kom...	Ändrad	Skapad	Använd	Attribut
NewFileTime.txt	308	81	2019-08-11 18:36			A
NewFileTime_x64.exe	426 760	253 847	2024-07-24 08:38			-rwxrwxrwx

Opening the executable `NewFileTime_x64.exe` we can see that it is a shareware that lets you manipulate timestamps on files and folders:



This has most likely been used as a "timestomping" tool, a technique were you hide your steps by modifying creation, accesses and modifications dates on files. Since we have nothing else to go on, lets look over the timestamps on the files in the archive.

Most of the files was last modified in August 2024, except two text files that stand out. These were last modified "1984-06-06". One of them contains the following content:

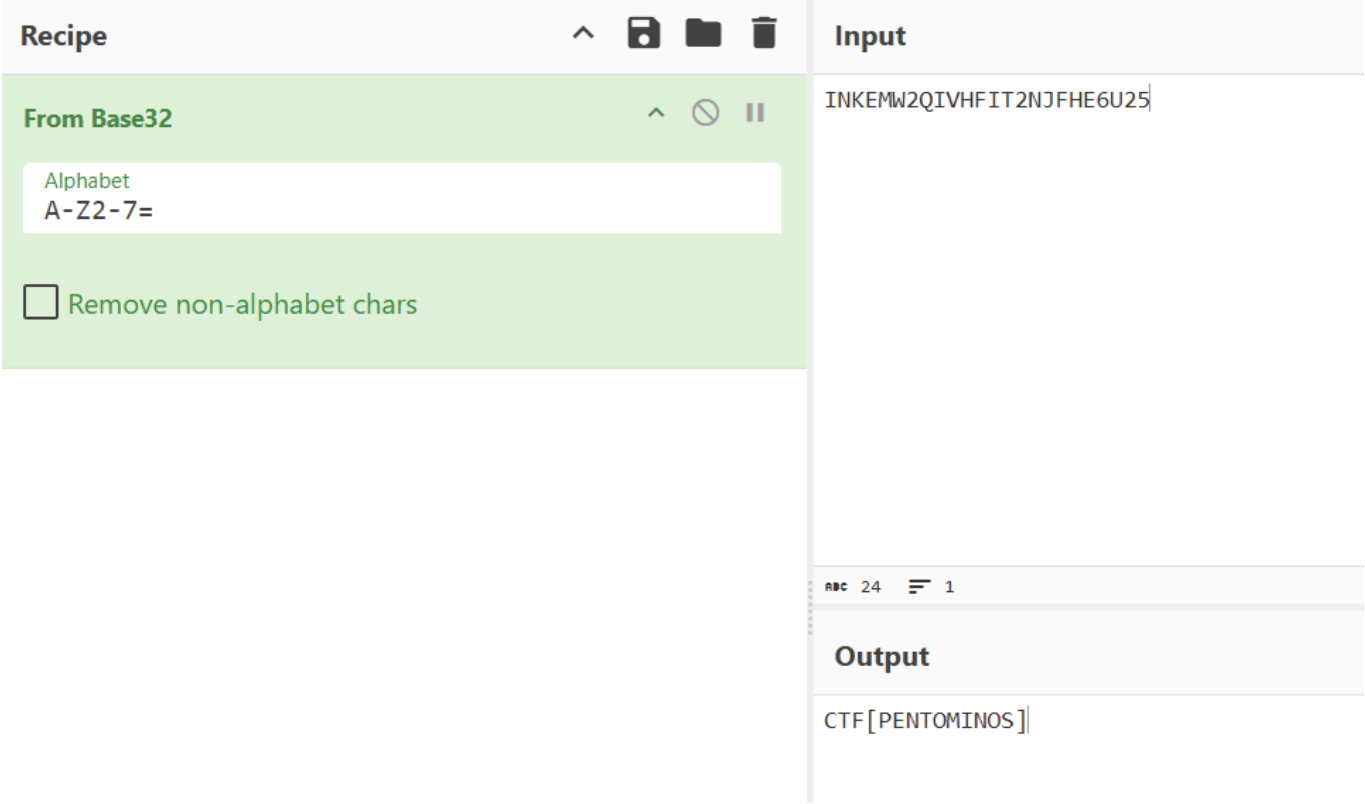
```
Jamborino Jamboro Archipelago Island Apple Horse
Monkey Dog
Kit Kat House Flower
```

This does not really give us anything, so lets open the second one:

```
C:\Users\aleksej.pazjitnov\Downloads\INKEMW2QIVHFIT2NJFHE6U25.txt
```

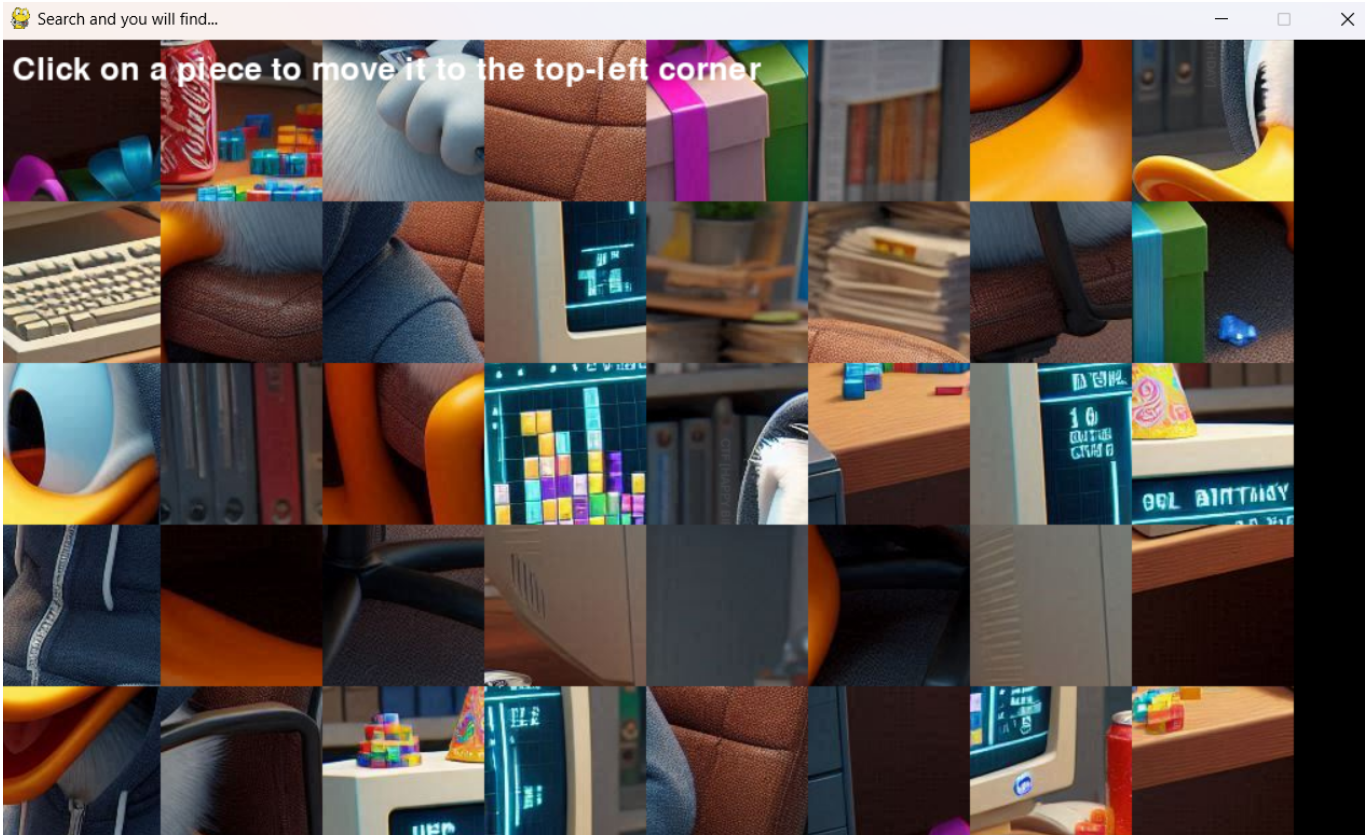
This is referencing the filepath of the deleted file, but there is something strange about the filename, `INKEMW2QIVHFIT2NJFHE6U25.txt`. Removing the file extension and analyzing the name shows us that is

most likely base32 encoded. Decoding this gives us our eight flag `CTF[PENTOMINOS]`:



puzzle.exe

We will start of by running the program to see what it is. It appears to be a puzzle that when solved will show an AI generated image that has been split:



We could try to solve it and we might get the final flag, but instead lets do some further analysis.

To start of lets try to analyze if there are any strings or hidden parts that can be found before trying to decompile it using a program such as "IDA Pro". Running the `strings` command on the executable show us that the final strings found to be that of `python312.dll`, meaning this could be a python program. If we search for strings again and search for `pyinstaller` we get a match:

```
zPYZ-00.pyz
8python312.dll
love@:~$ strings puzzle.exe | grep pyinstaller
_pyinstaller_pyz
love@:~$
```

This indicates the program has been compiled using pyinstaller. Since it is possible to decompile using a simple script, there is no need open it in a more advanced decompiler.

Running the following script, `pyinstxtractor.py` on the executable successfully identifies it as a python program and extracts the source files:

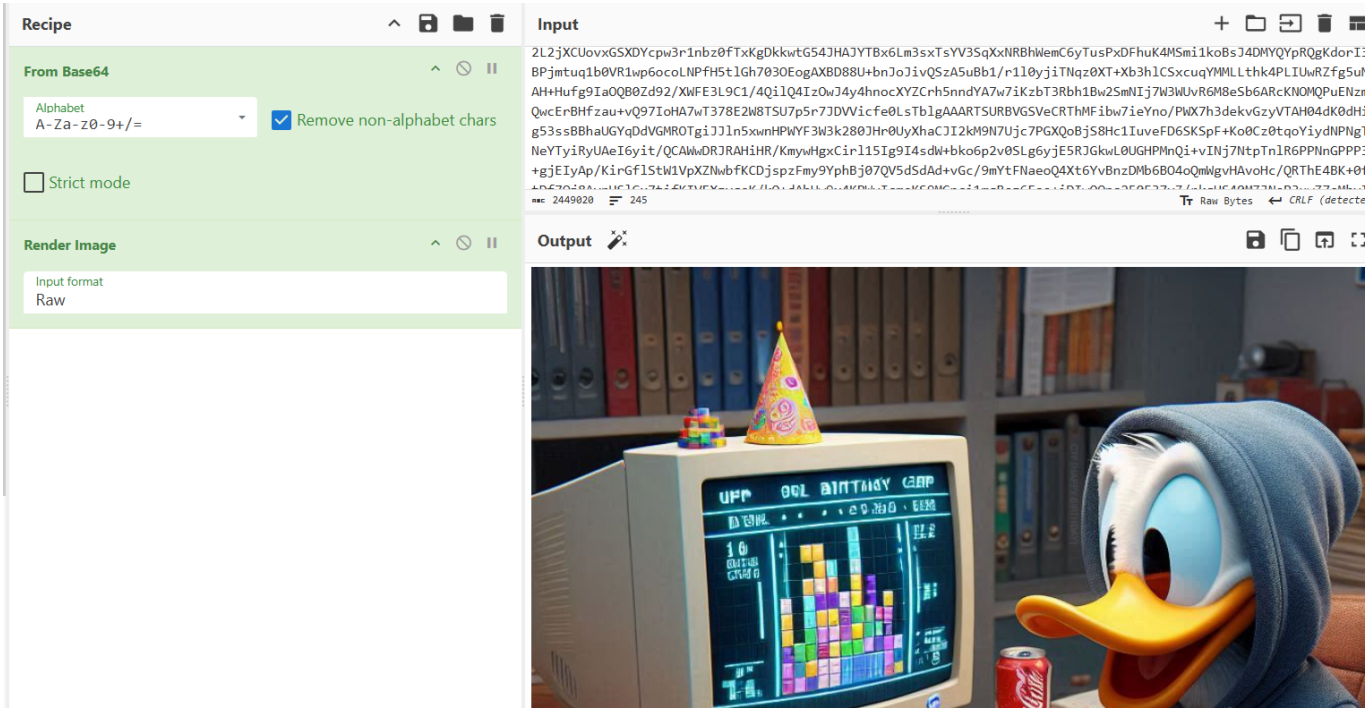
```
love@:~$ python3 pyinstxtractor.py puzzle.exe
[+] Processing puzzle.exe
[+] Pyinstaller version: 2.1+
[+] Python version: 3.12
[+] Length of package: 18773937 bytes
[+] Found 214 files in CArchive
[+] Beginning extraction...please standby
[+] Possible entry point: pyiboot01_bootstrap.pyc
[+] Possible entry point: pyi_rth_inspect.pyc
[+] Possible entry point: pyi_rth_pkgres.pyc
[+] Possible entry point: pyi_rth_setuptools.pyc
[+] Possible entry point: pyi_rth_multiprocessing.pyc
[+] Possible entry point: pyi_rth_pkgutil.pyc
[+] Possible entry point: puzzle_new.pyc
[!] Warning: This script is running in a different Python version than the one used to build the executable.
[!] Please run this script in Python 3.12 to prevent extraction errors during unmarshalling
[!] Skipping pyz extraction
[+] Successfully extracted pyinstaller archive: puzzle.exe

You can now use a python decompiler on the pyc files within the extracted directory
love@:~$
```

Lets open the extracted source inside a more advanced editor such as Vs Code. Among the source files we see mostly standard files related to various python libraries, the one standing out is `puzzle_new.pyc`. This is most likely the games main entry point.

If we open the files with out decompiling it we can still get some information. We can see a bunch of function calls along with a large base64 encoded image:

Since the program was compiled using Python 3.12, the known CLI tools to decompile .pyc files are not gonna work. There is a site (https://pylingual.io/) that allows for decompiling, but instead lets see if we can decode this string and see what the image is.

Decoding it and then rendering the results gives us what appears to be the image in the executable before it gets split:

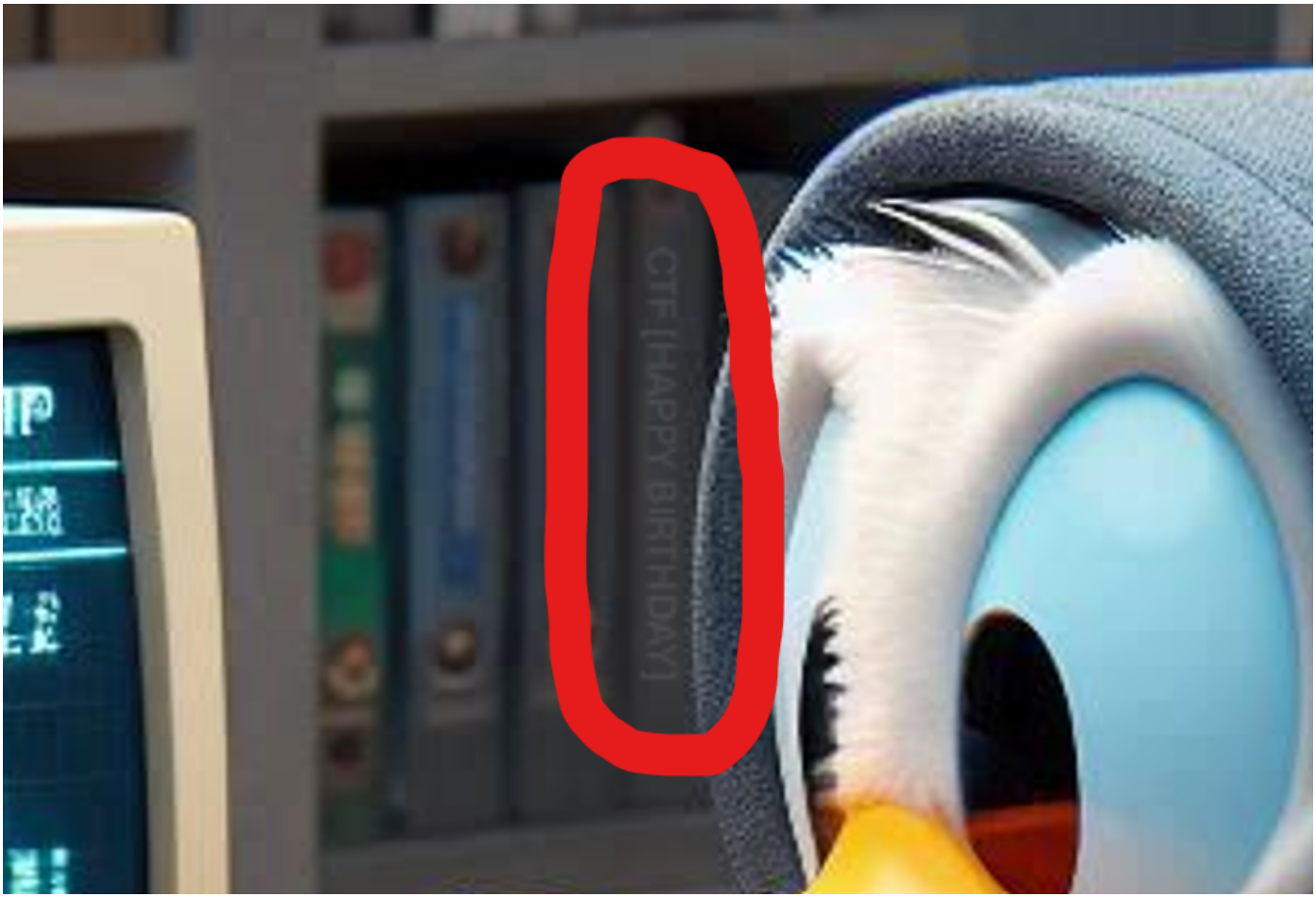


We will save it incase it contains any hints.

At this point, I had no clue on what to do, since all my assumptions did not result in anything. Decompiling the `.pyc` file gave nothing except in how to program worked, showing that nothing happens if you complete the puzzle. The other source files did not contain anything of interest either. The image itself did not give any result when analyzing it by looking for hidden files inside of it, strings of interest and changing color coding.

In a final desperate attempt lets look at the image in a standard image viewer incase there was something that was overlooked. There should be nothing since the image is obviously AI generated..

When zooming in on the computer screen to see if there was a hint there, since it shows a Tetris like game and an earlier flag was "RHODE_ISLAND_Z" there might be a clue here. Thats when I noticed there was something on the books next to the computer, text. This was the ninth and final flag, `CTF[HAPPY BIRTHDAY]`:



The program did not lie when it said "Search and you will find...", the flag was right there.