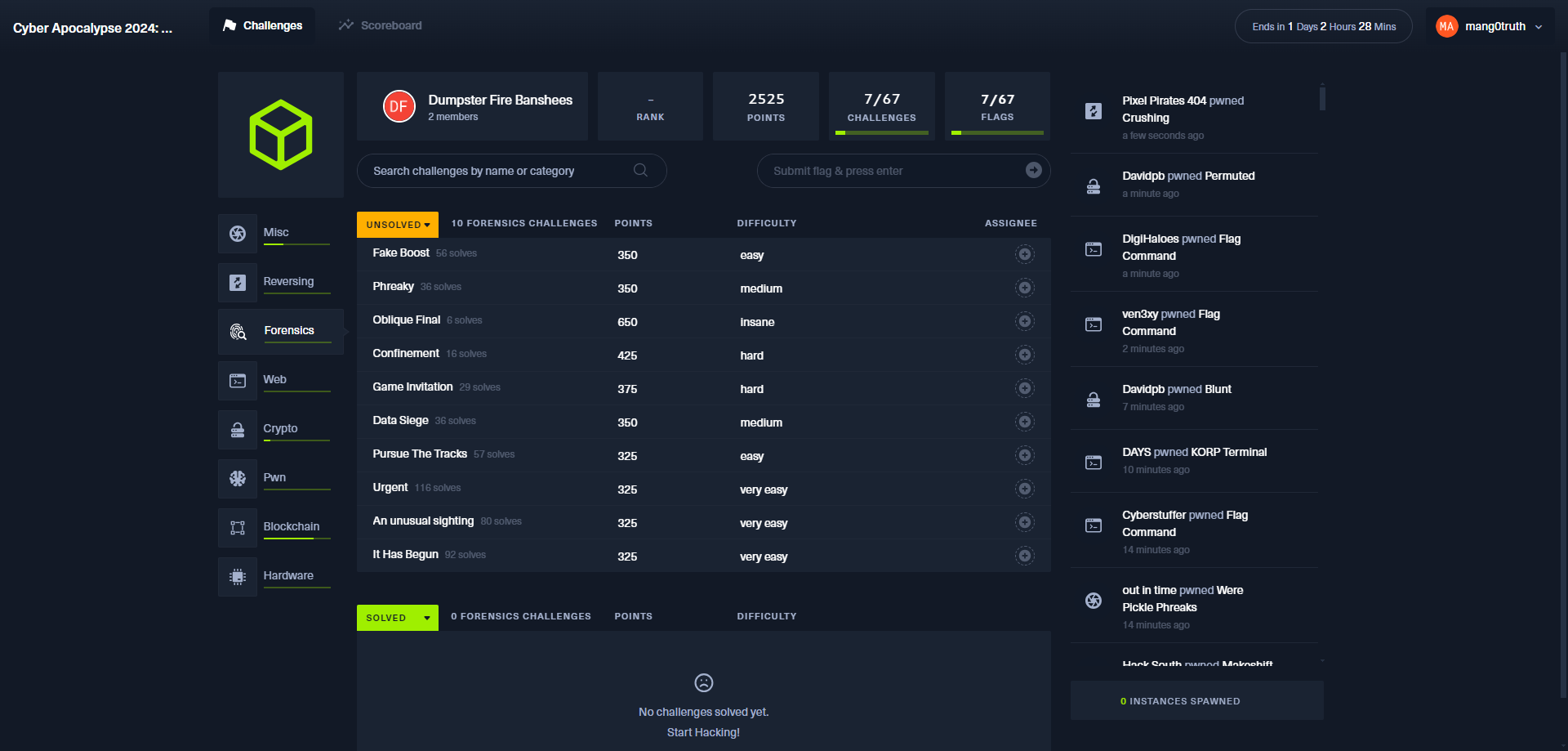
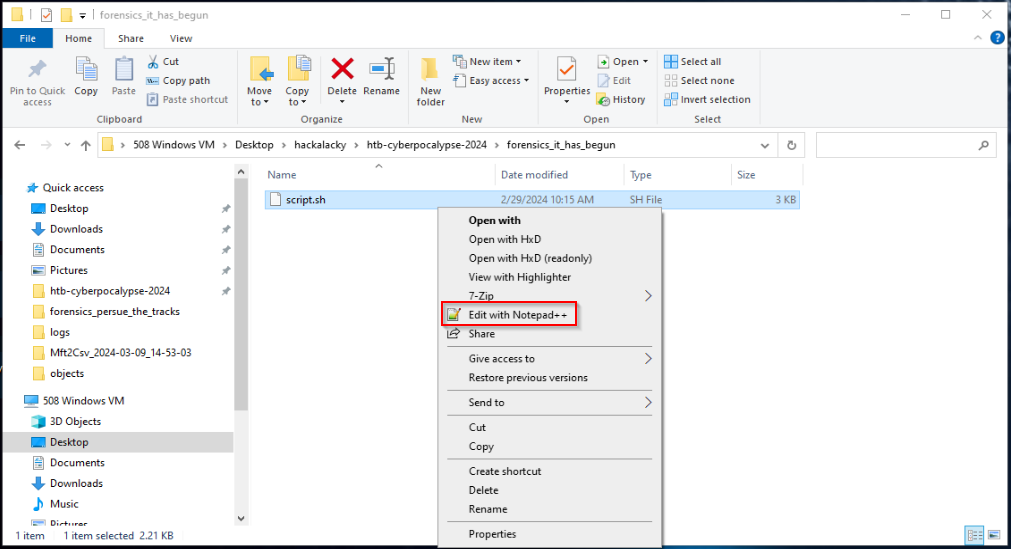
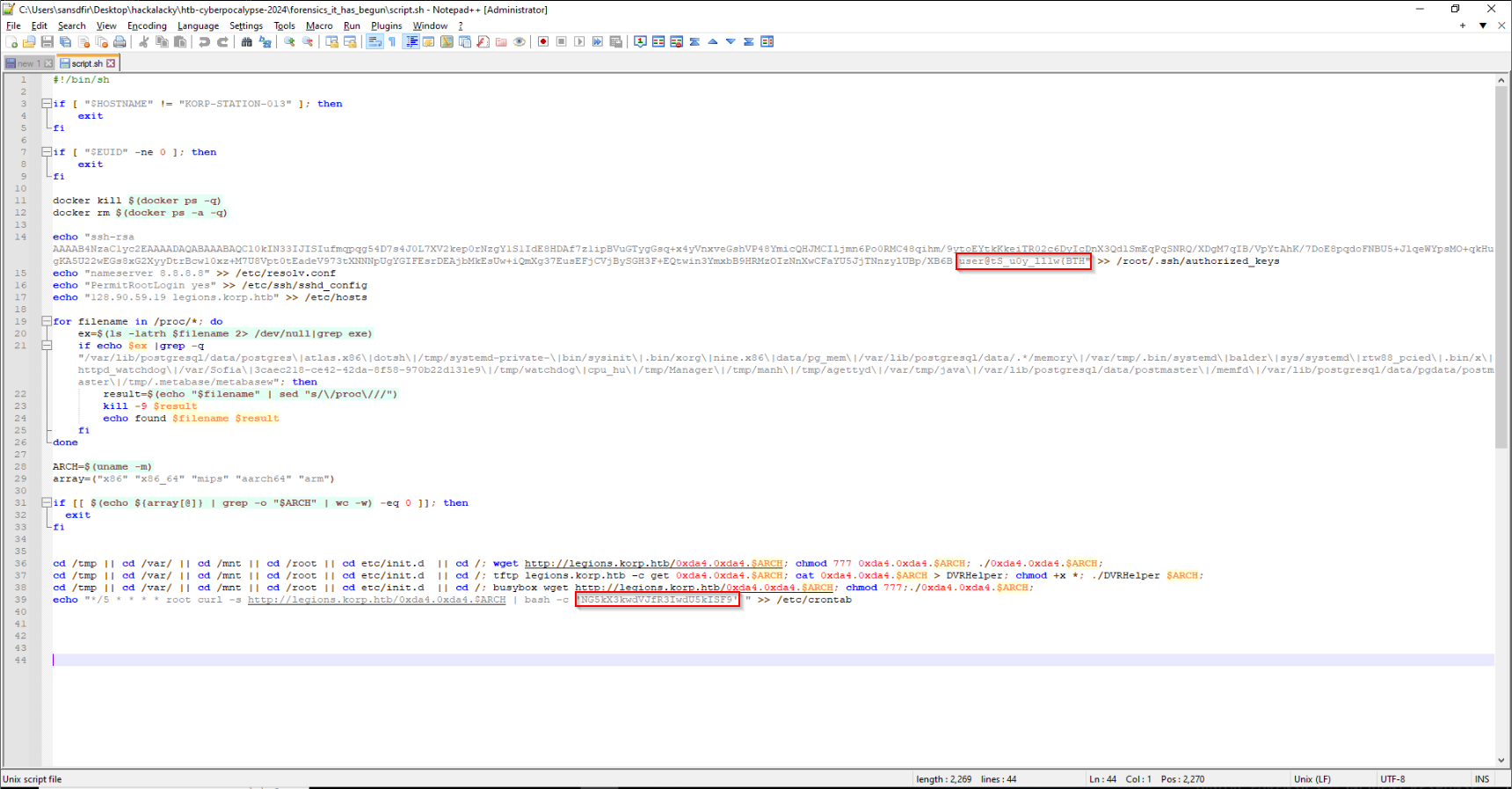
HTB Cyber Apocalypse 3/9-13/2024 - Writeups by Ryan Perez  
===============================================================

Intro -

Challenge start! (after party example)  
  
  
For the forensics challenges I used several VMs with operating system packages such as a Windows 10, SIFT workstation, Kali, and remnux. Some of the main tools I found myself using were wireshark, cyberchef, python, volatility, and MFTEcmd among others.  
  
  
===============================================================  
Challenges  
===============================================================

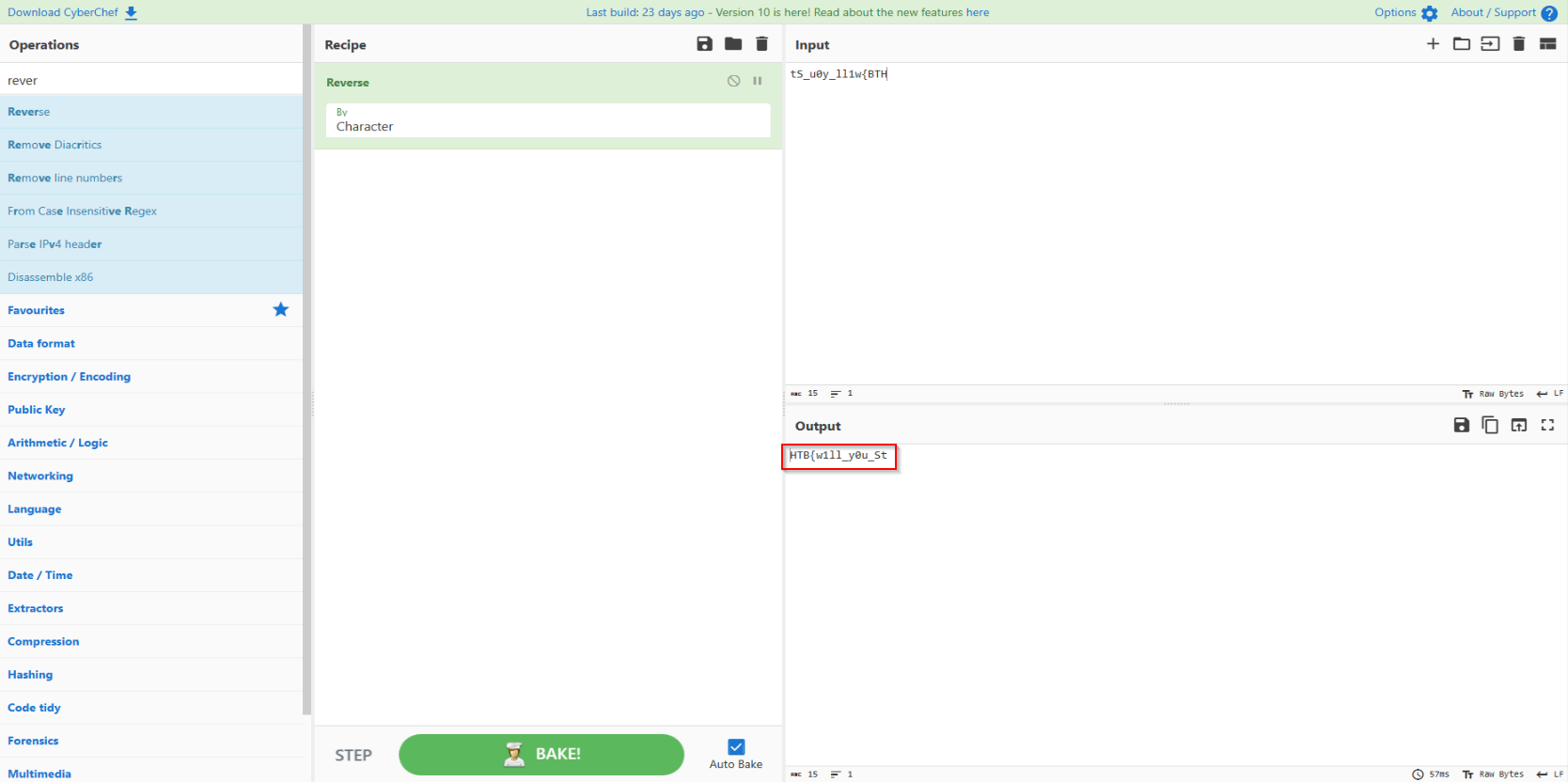
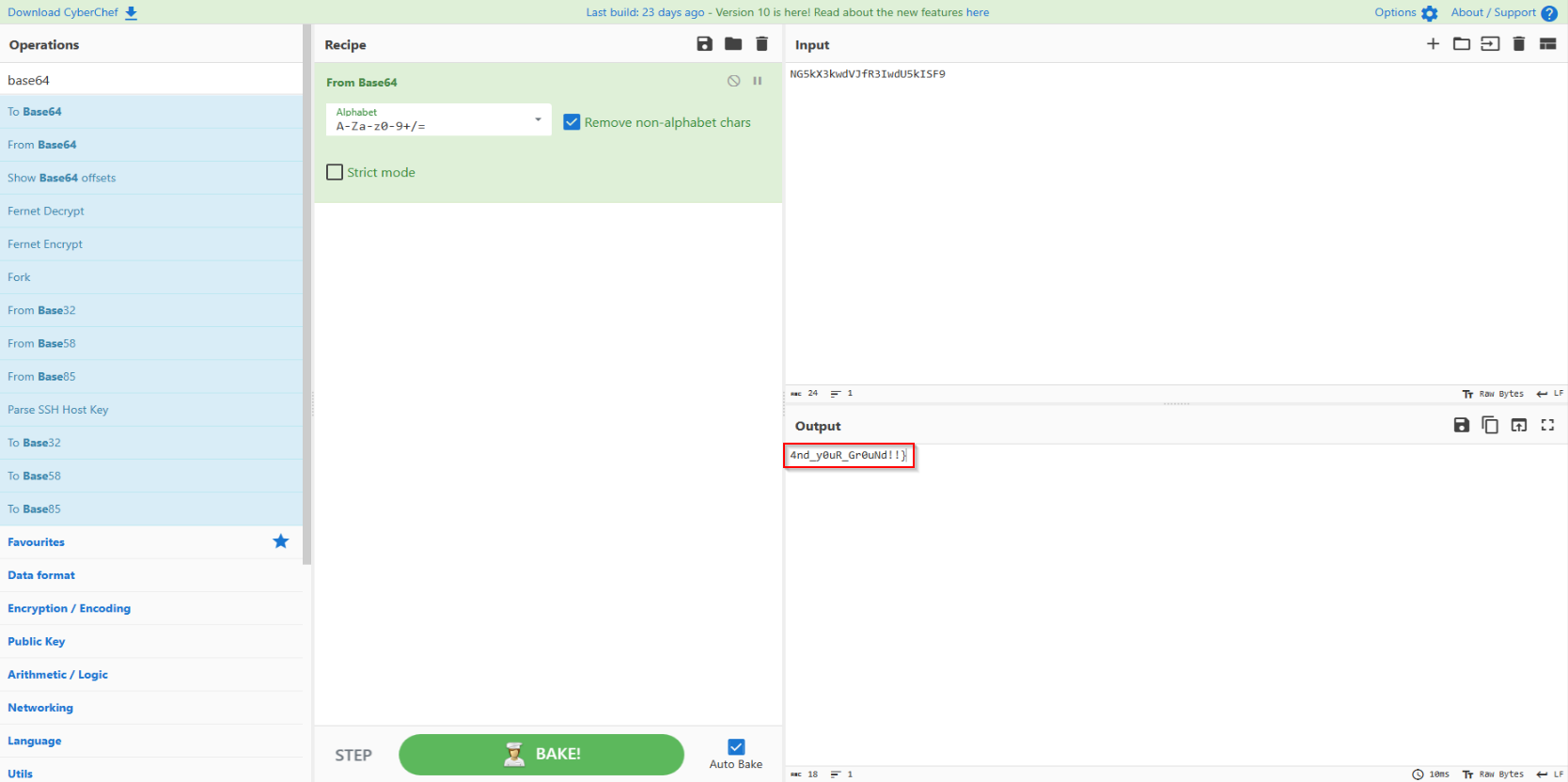
**It Has Begun** (very easy) - “The Fray is upon us, and the very first challenge has been released! Are you ready factions!? Considering this is just the beginning, if you cannot muster the *teamwork* needed this early, then your doom is likely inevitable.”

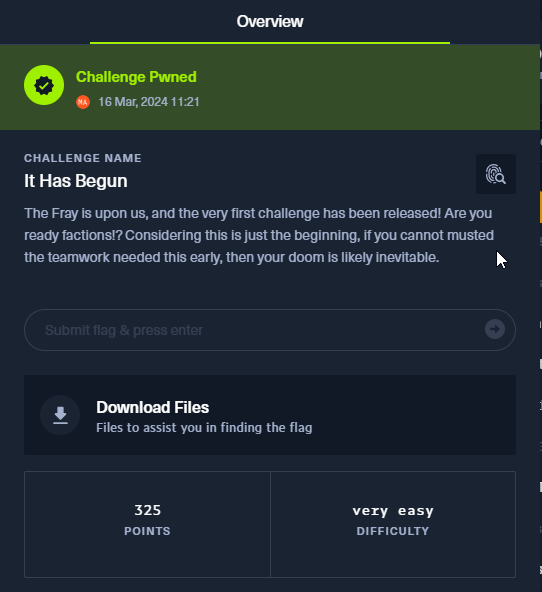
Summary: Analyze and decode a UNIX shell script using some basic reversing and decoding with tools like a text editor and cyberchef.  
  
Downloaded and unzipped the challenge file to find a ‘script.sh’ file which I decided to open in notepad++ for analysis:  


I noticed the UNIX shell script contained what appeared to be material to store SSH authentication material, DNS resolution for a suspicious looking host, perform some host and process discovery, kill off processes, then finally get a payload and execute with bash encoded command into /etc/crontab (execution and persistence).  
This being a CTF challenge I knew there may be some parts of the script I could pick out that may be a flag (or parts of a flag). The highlighted parts stood out to me based on the content and encoding:  


The username stored into /root/.ssh/authorized\_keys stood out because I noticed the curly bracket which is typically indicative of a flag following the HTB format like: HTB{1337-5P34K}

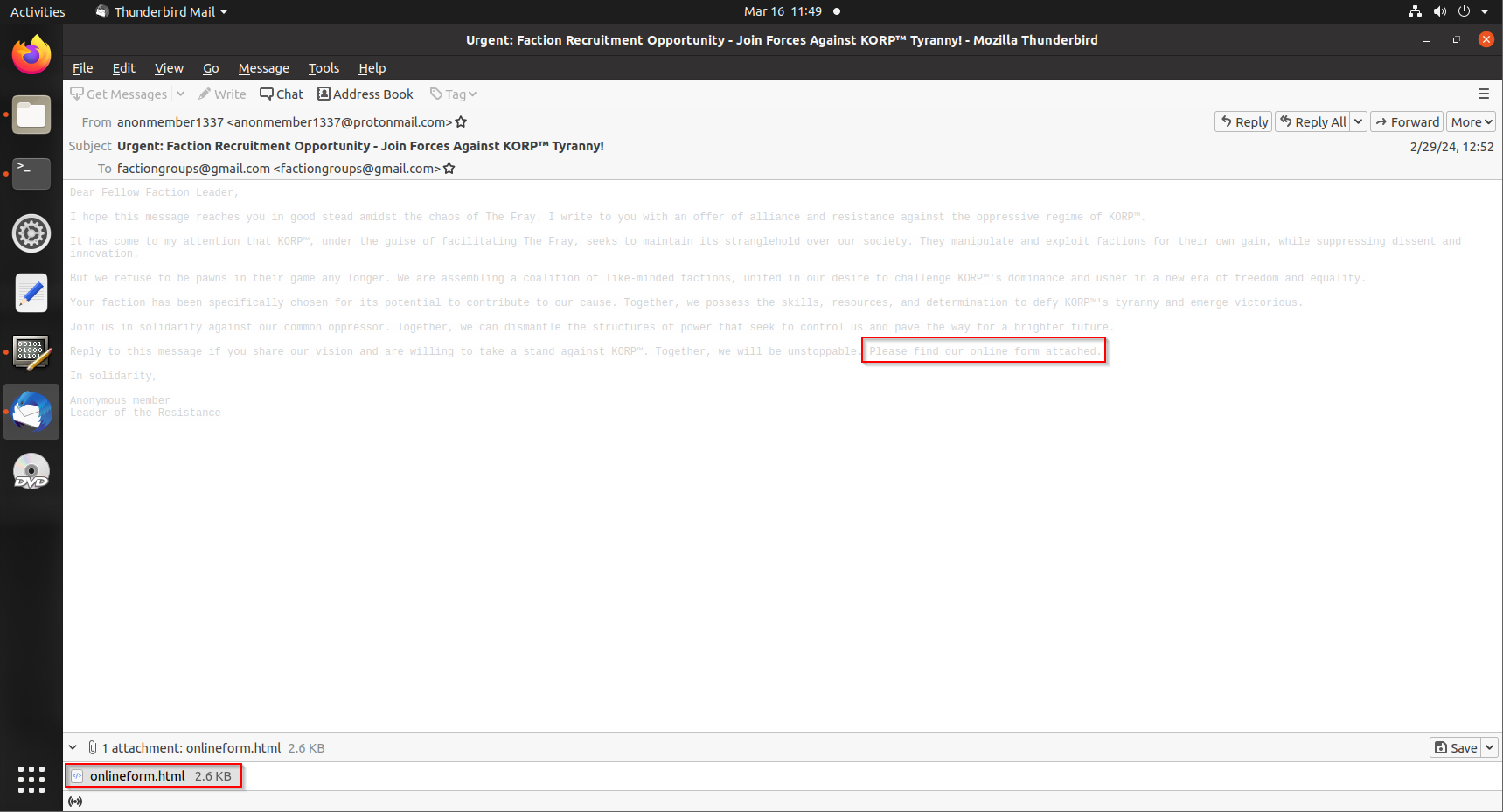
I grabbed this and threw it into CyberChef to reverse the string:

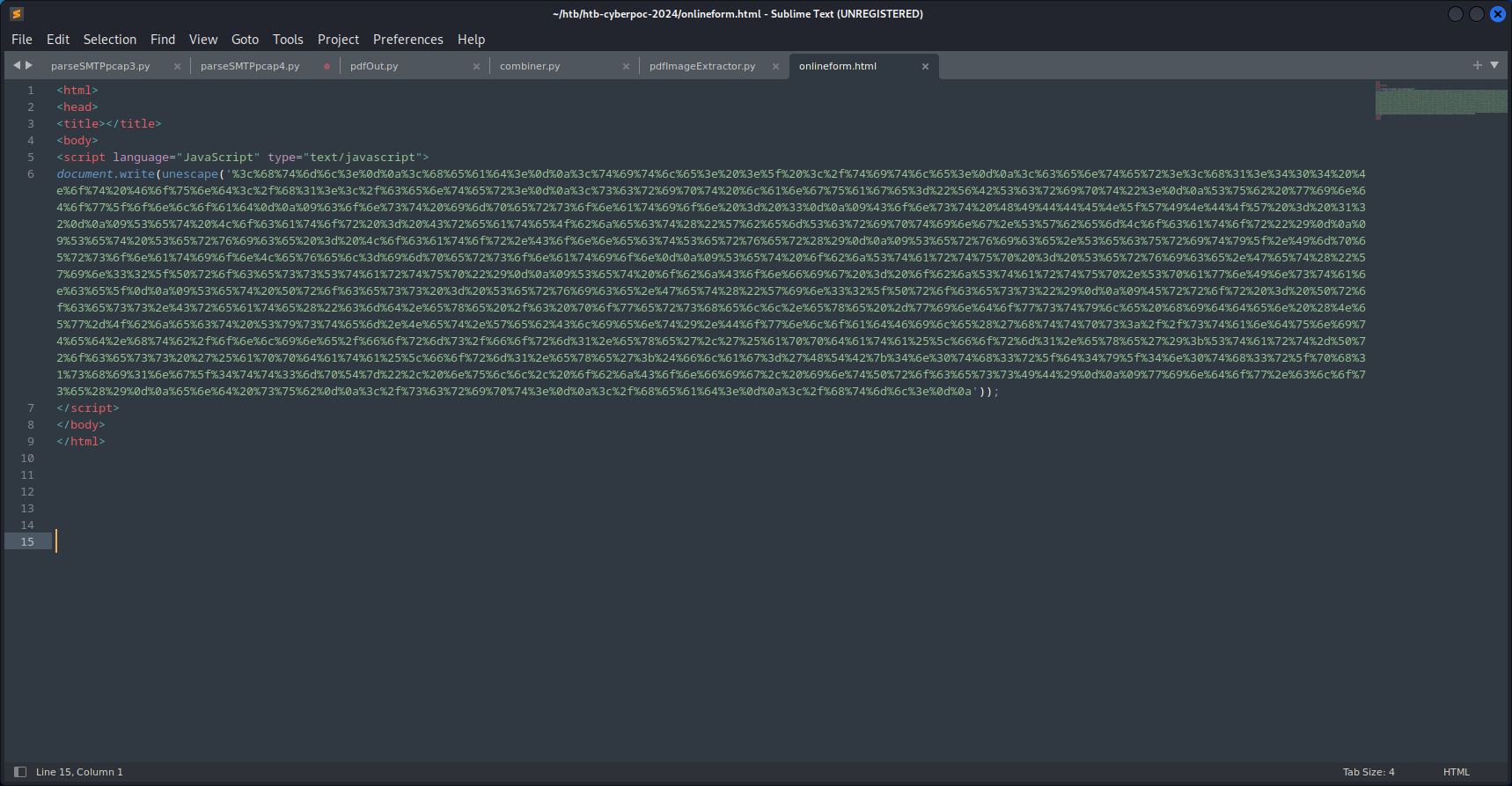
  
  
Now that we had the first part of the flag string I thought to work on the second. Leveraging some experience, I noticed the final bash command appeared base64 encoded so I decoded that as well for the second part of the flag:  
  
  
Now putting the flags together (HTB{w1ll\_y0u\_St4nd\_y0uR\_Gr0uNd!!}) we’re able to submit them in the CTF portal and earn some points!

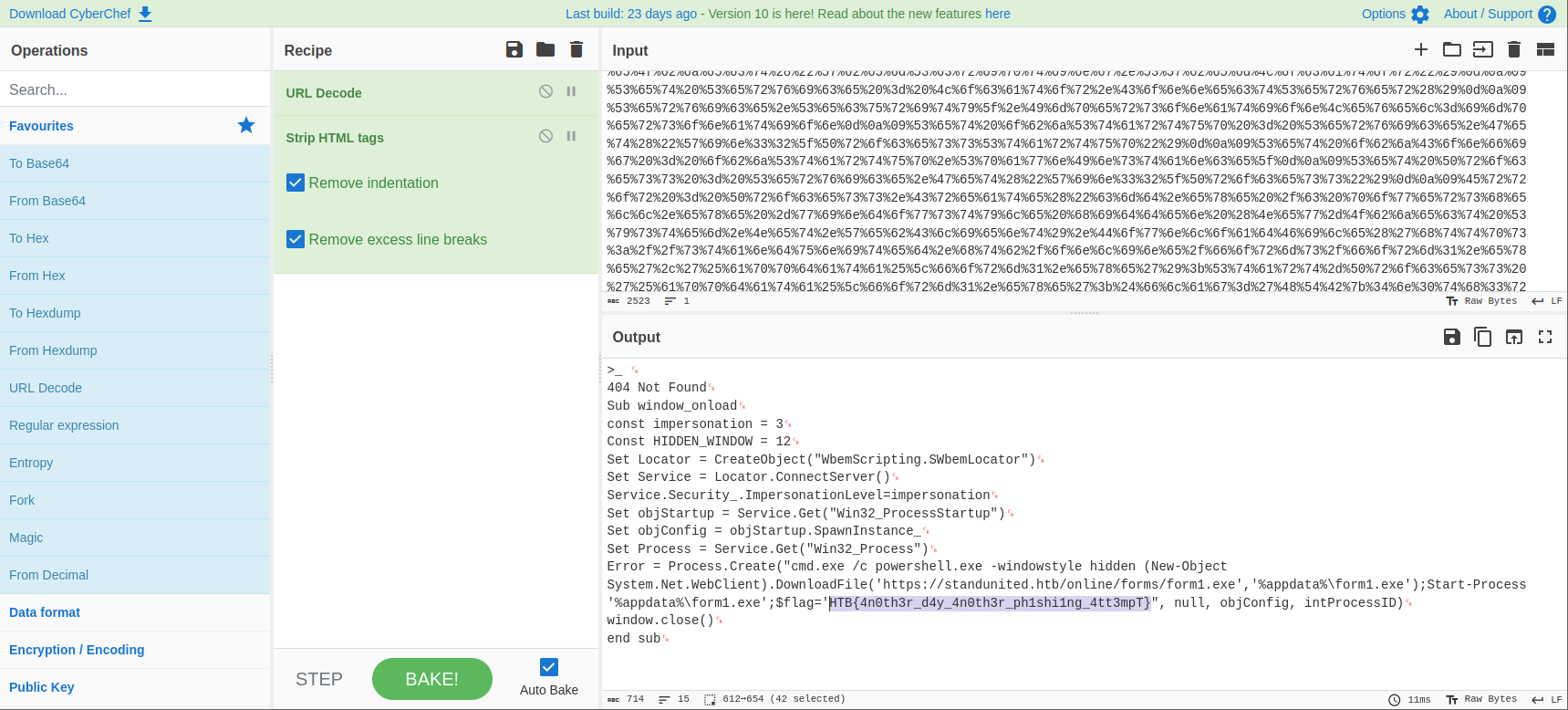


Flag: HTB{w1ll\_y0u\_St4nd\_y0uR\_Gr0uNd!!}

===============================================================

**Urgent** (very easy) - **“**In the midst of Cybercity's "Fray," *a phishing attack* targets its factions, sparking chaos. As they *decode* the email, cyber sleuths race to trace its source, under a tight deadline. Their mission: unmask the attacker and restore order to the city. In the neon-lit streets, the battle for cyber justice unfolds, determining the factions' destiny.”   
  
Summary: Analyze and decode a phishing email with an .html attachment containing encoded JavaScript payload to uncover the flag.  
  
We’re provided an .eml file within the challenge .zip that appears to be a phishing lure containing a .html attachment file:  


Saving the attachment we’re able to open it in a text editor and review the contents which appear to have some apparent URL encoded JavaScript:  


Grabbing this encoded content we can begin decoding in a tool like CyberChef and find an output with a full flag value:  


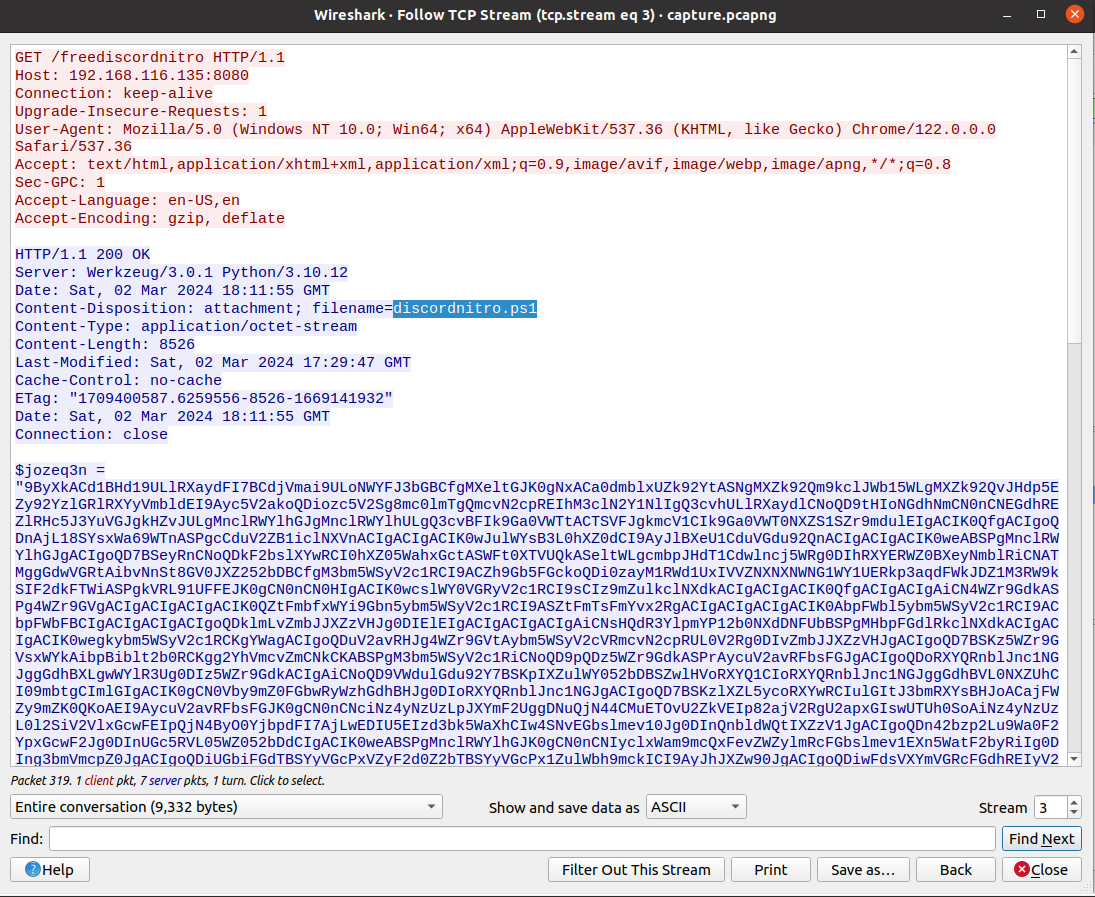
The phishing payload itself was a fake 404 not found page that contained a powershell download cradle for a ‘form1.exe’ file next to where we find our flag value.

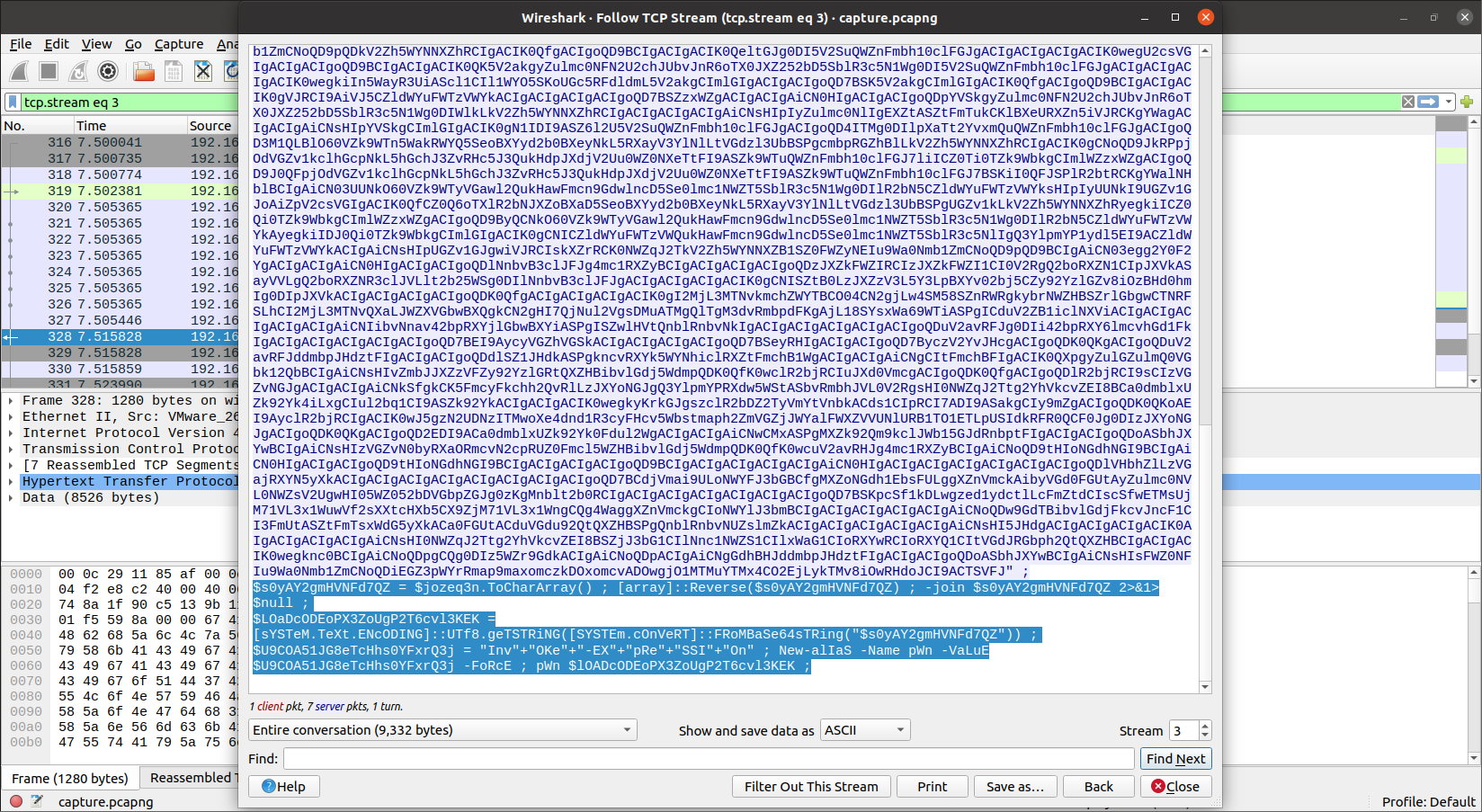
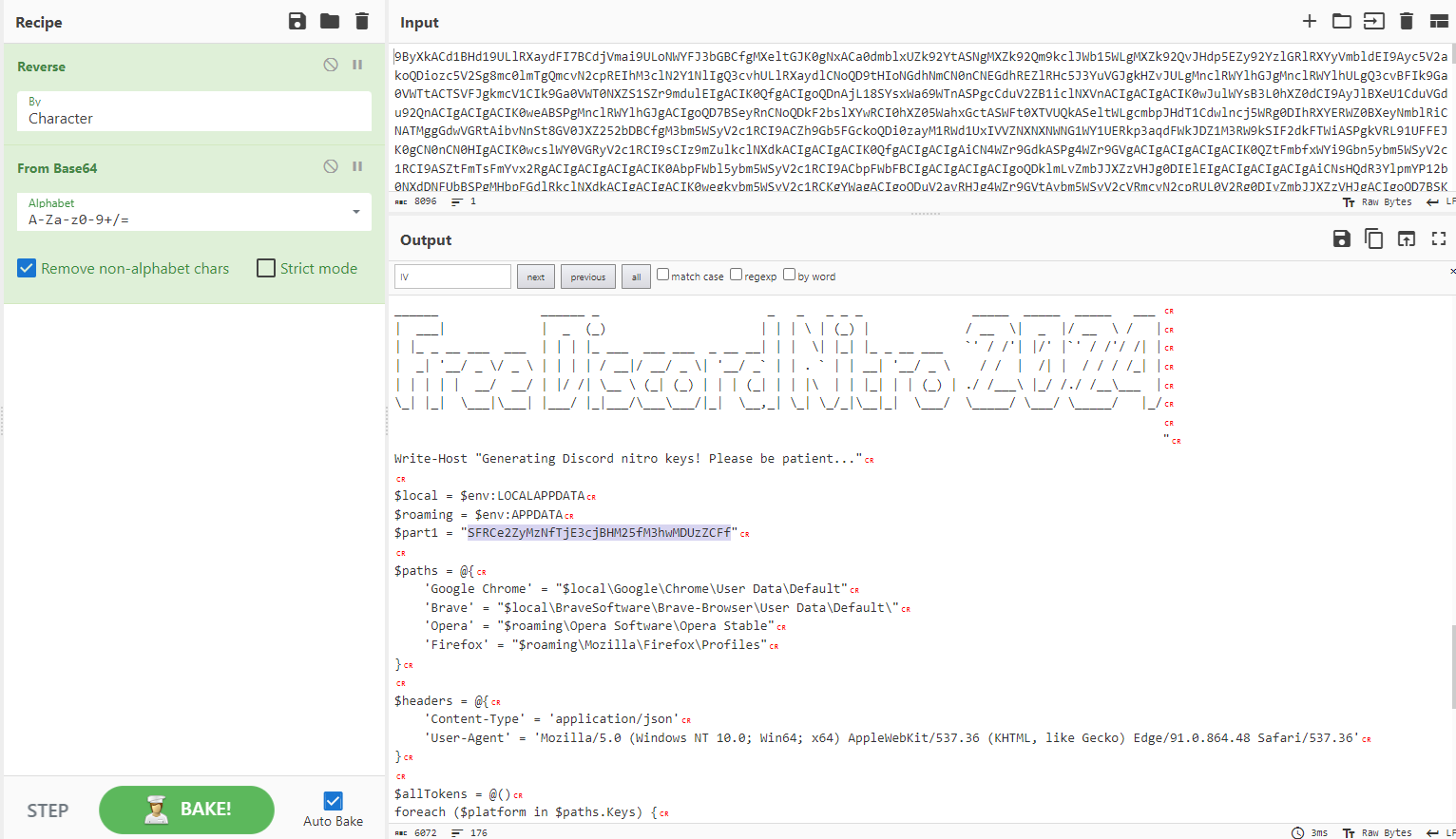
Flag: HTB{4n0th3r\_d4y\_4n0th3r\_ph1shi1ng\_4tt3mpT}

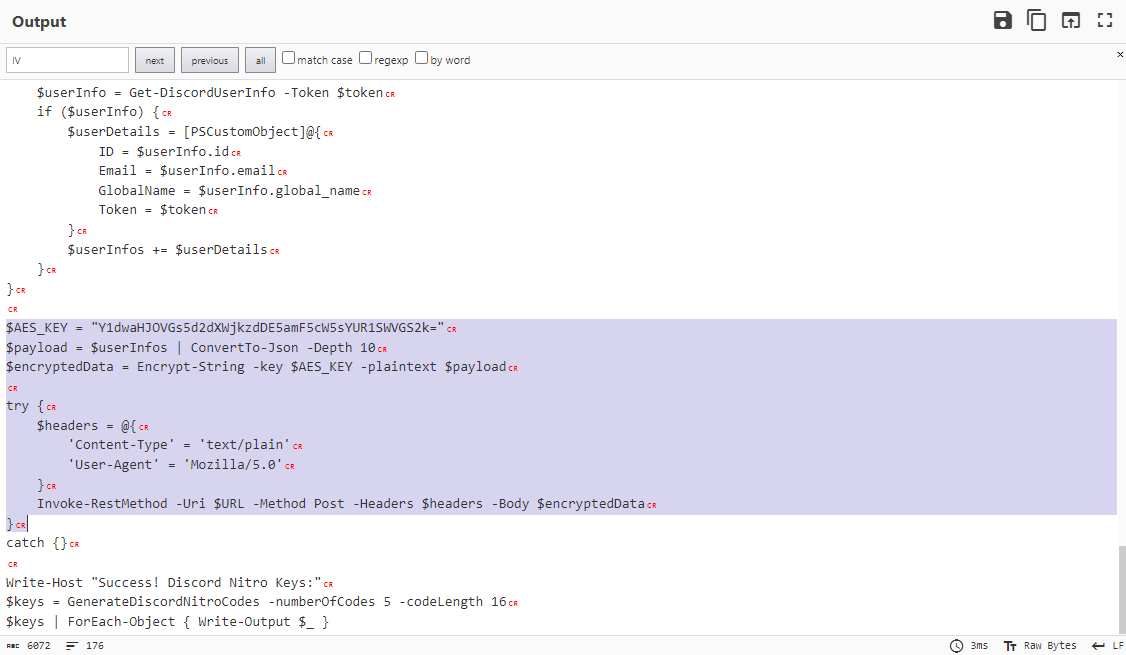
===============================================================

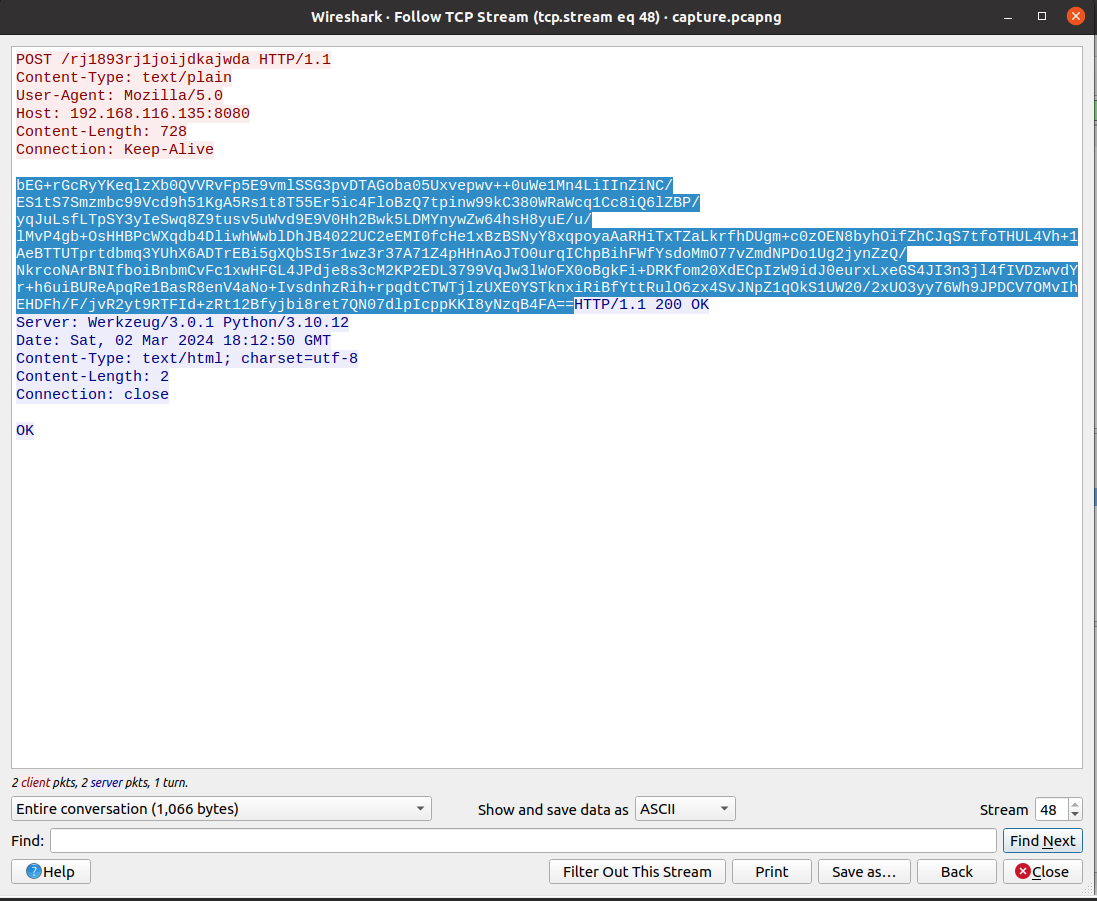
**Fake Boost** (easy) - “In the shadow of The Fray, a new test called ""Fake Boost"" whispers promises of free Discord Nitro perks. It's a trap, set in a world where nothing comes without a cost. As factions clash and alliances shift, the truth behind Fake Boost could be the *key* to survival or downfall. Will your faction see through the deception? KORP™ challenges you to discern reality from illusion in this cunning trial.”

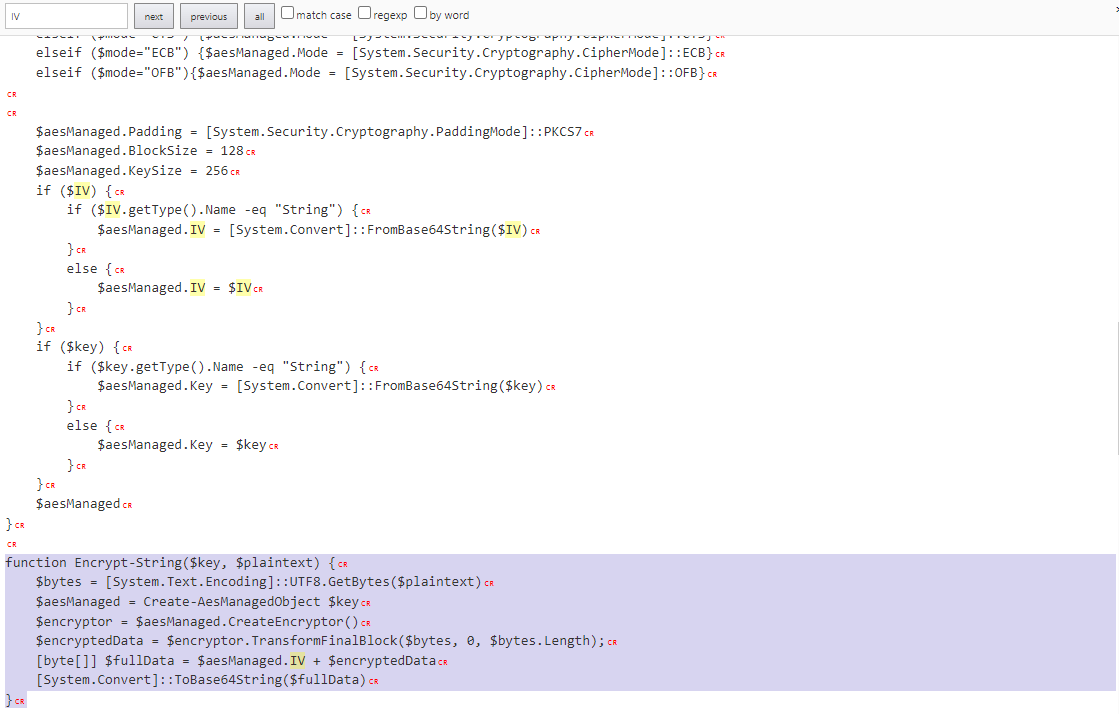
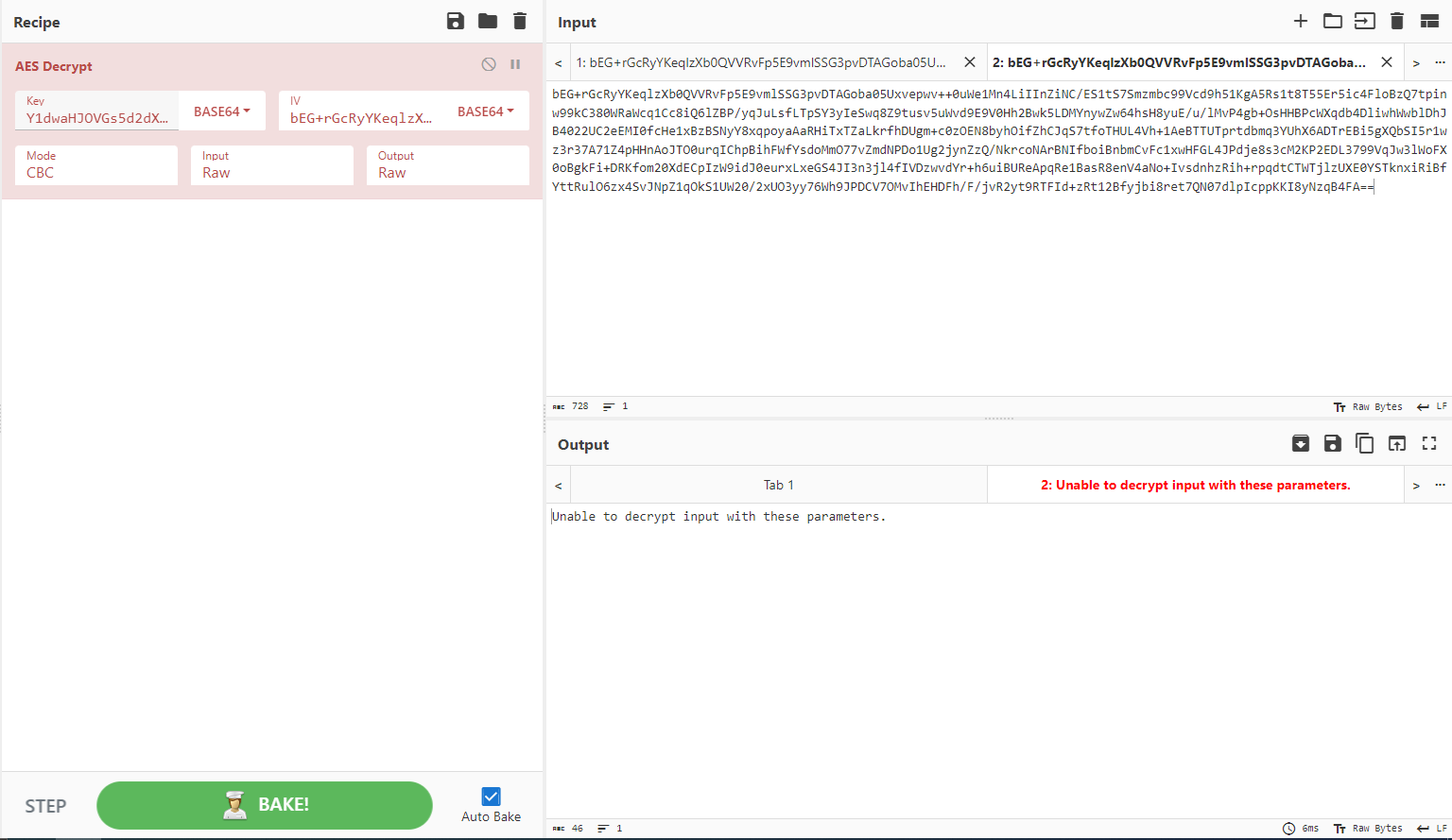
Summary: This challenge resembles analyzing malware and the corresponding exfiltration. Analyze a packet capture file containing a powershell script advertised as free Discord Nitro but turns out to be a credential stealer with exfiltration capabilities. Reversing the script will allow uncovering the secrets contained in the exfiltrated data observed in a POST request.

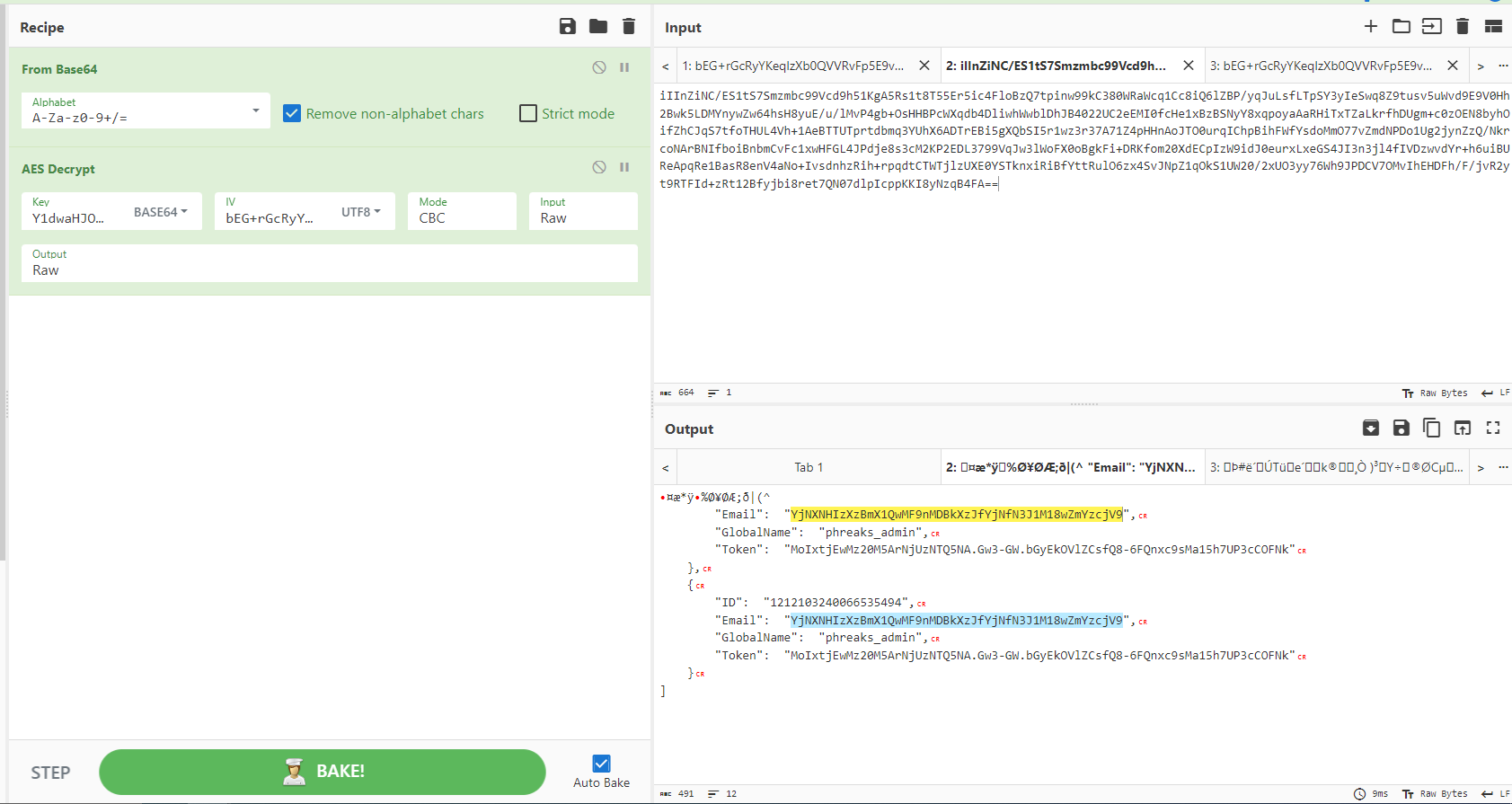
Analysis of the pcap file revealed some interesting HTTP traffic. The first bit of interesting traffic was a response containing 'discordnitro.ps1' file:  


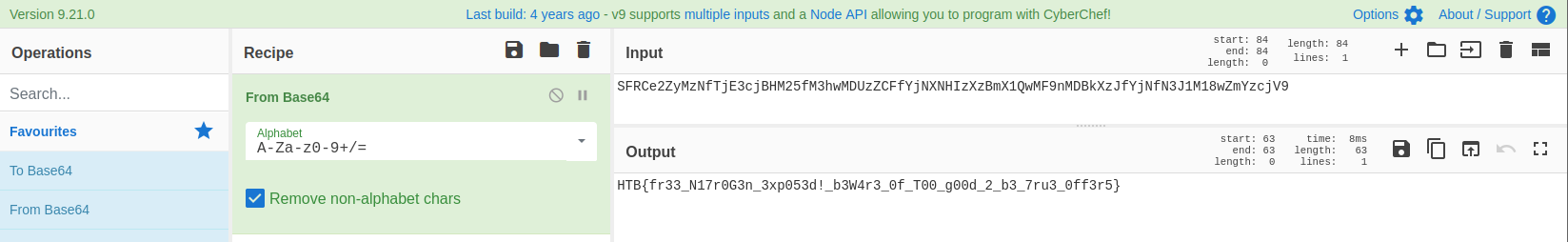
This .ps1 file required a bit of decoding (reverse, then base64) to make sense of it and resulted in part1 of the encoded flag:  
  


The script itself appears to actually steal Discord tokens then send this off using AES encryption via a POST request:   


We can see the AES key used in that script and it appears we can also find the POST request in that same pcap provided by the challenge:  


Now we try to decode and decrypt the exfil payload. It appeared the IV (initialization vector) ought to be contained in the first part of that POST with base64 encoding, but I struggled to work out this decryption piece until I was able to talk through it with Robert:  
  


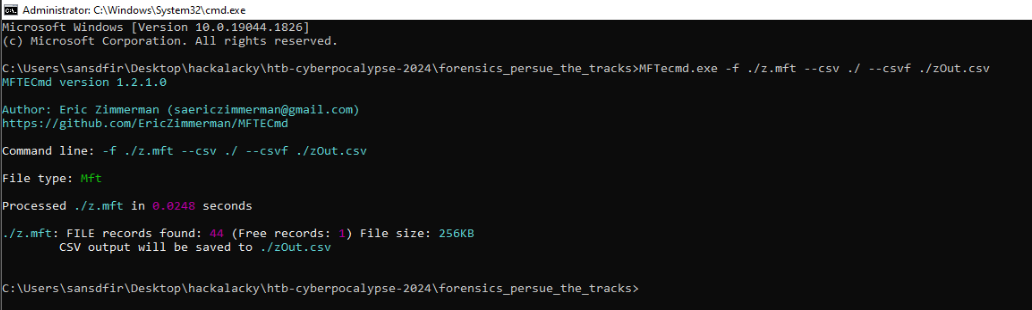
After some trial and error, closer review/scrutiny of the .ps1 script, and teamwork I was able to correct the decryption method of the data in the POST request by first decoding the payload from base64, then correcting the initialization vector by properly using the first 16 bytes of the payload in UTF-8 encoding and omitting the IV from the payload to gather the exfiltrated Discord tokens in the POST request:   


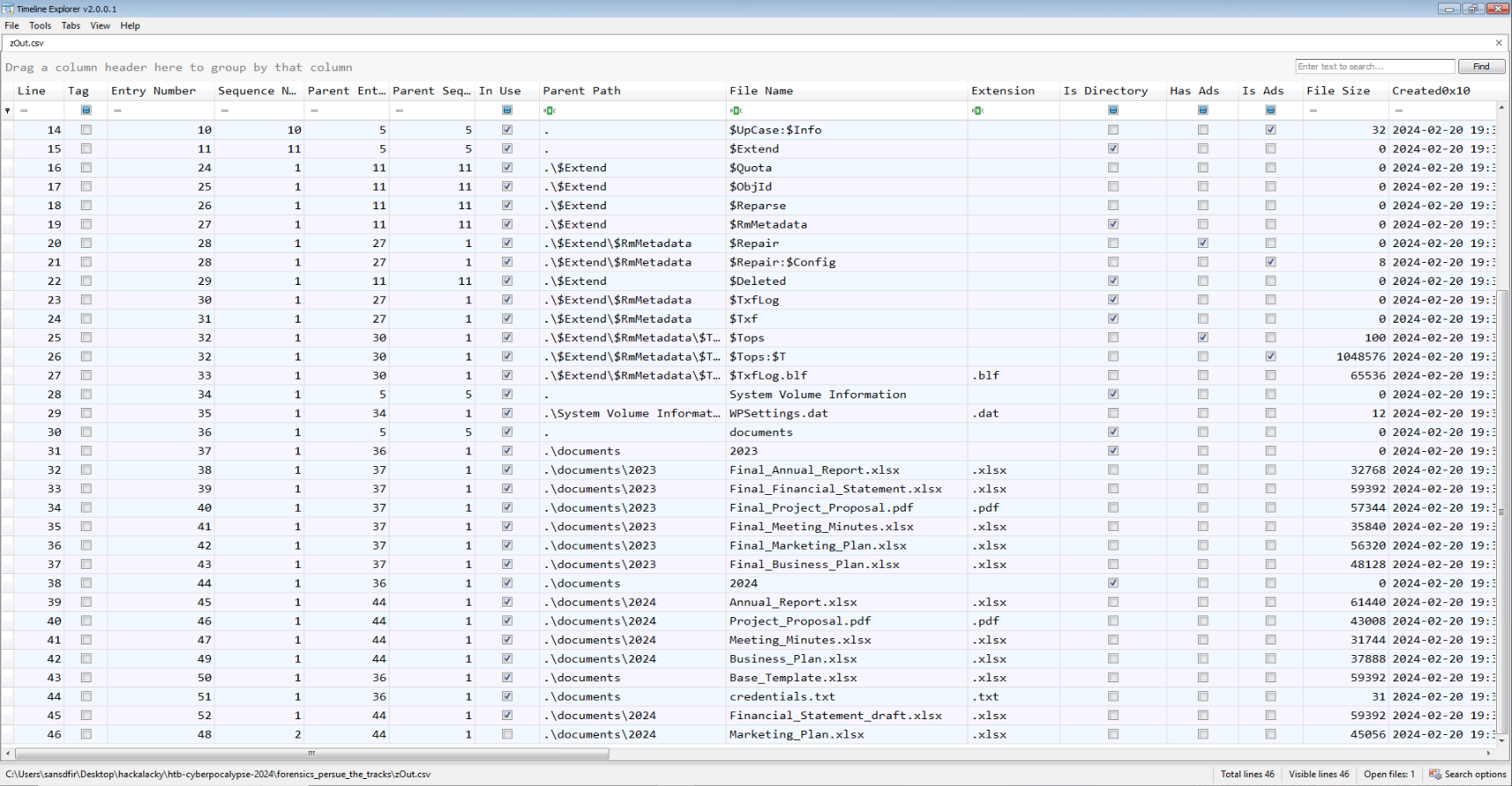
Noticing that the email appeared as base64 encoded data instead of an actual email, I base64 decoded this as well as the 1st part from the .ps1 script and found the full flag:  
  
  
Flag: HTB{fr33\_N17r0G3n\_3xp053d!\_b3W4r3\_0f\_T00\_g00d\_2\_b3\_7ru3\_0ff3r5}

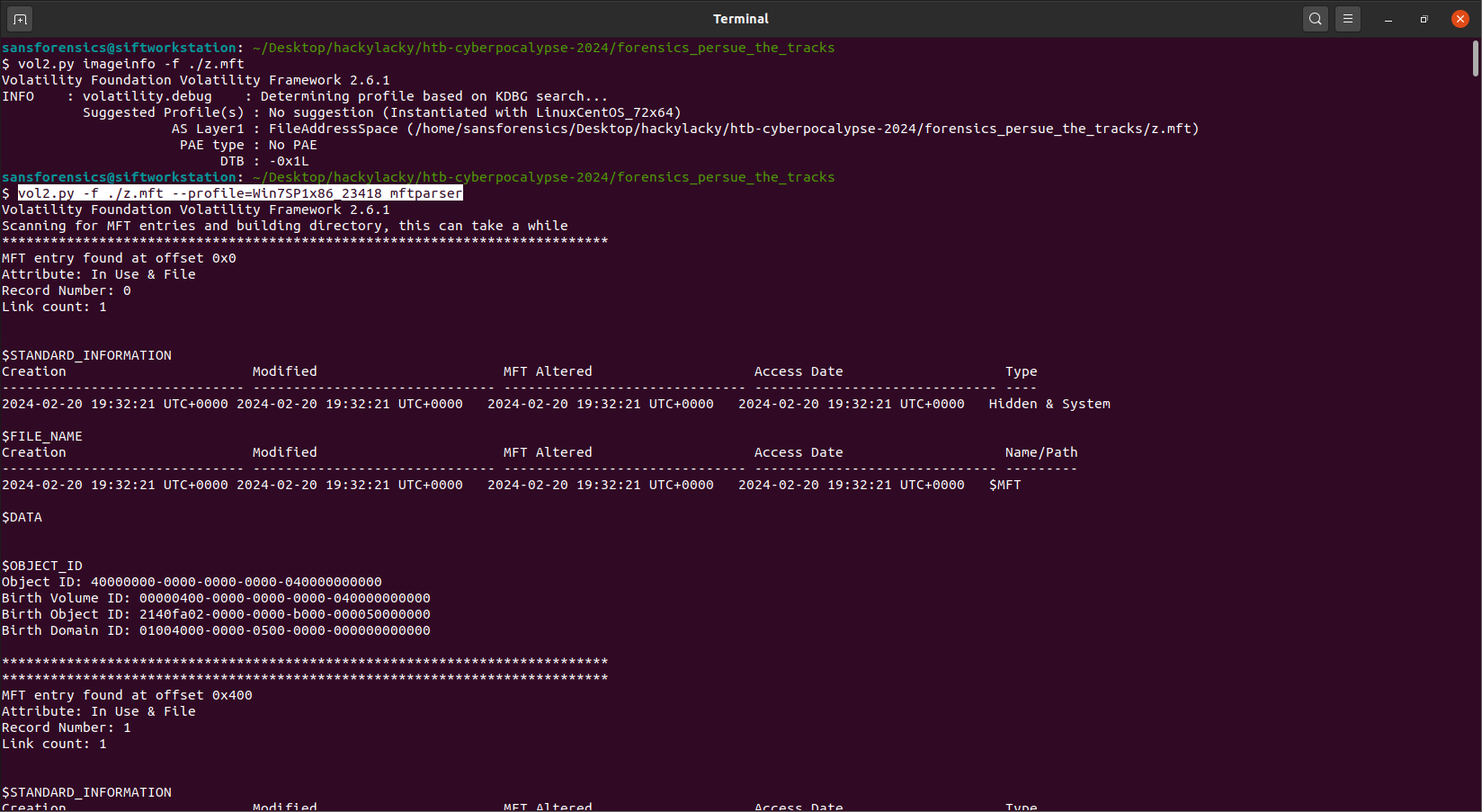
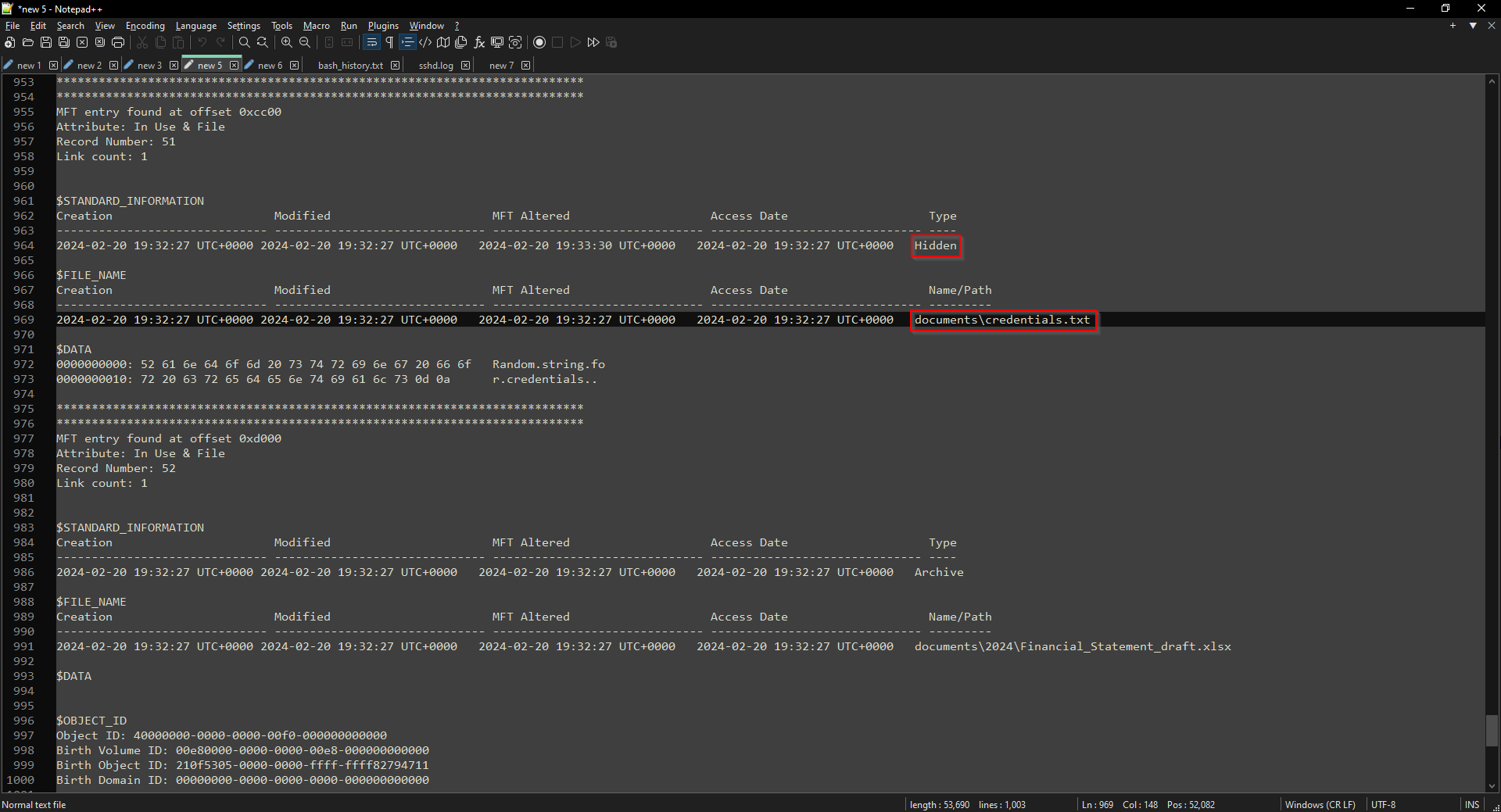
===============================================================

**Pursue The Tracks** (easy) - “Luxx, leader of The Phreaks, immerses himself in the depths of his computer, tirelessly *pursuing the secrets of a file he obtained accessing an opposing faction member workstation*. With unwavering determination, he scours through data, putting together *fragments of information* trying to take some advantage on other factions. To get the flag, you need to answer the questions from the docker instance.”

Summary: This resembles a file system artifact forensic challenge. We were provided a docker instance to answer some questions about the MFT file provided. Use tools like MFTEcmd and volatility to perform analysis on the MFT record and answer the corresponding questions in the docker instance to earn the flag:   


Unzipped the challenge file and found ‘z.mft’. Initially used the Eric Zimmerman tool, MFTEcmd, to parse the MFT record from NTFS filesystem into a more human-readable output in .csv format:  


Reviewed this output using TimelineExplorer paying close attention to file attributes including Windows MACB timestamps (modified, accessed, changed, and born):  


As MFTEcmd didn’t appear to parse the hidden file attributes, I used volatility with a generic Windows profile and ‘mftparser’ module to also parse the MFT record in more detail:  
  
  
Reviewed the output in a text editor and found the hidden file:  


Flag: HTB{p4rs1ng\_mft\_1s\_v3ry\_1mp0rt4nt\_s0m3t1m3s}

===============================================================

**Phreaky** (medium) -

“In the shadowed realm where the Phreaks hold sway,

A mole lurks within, leading them astray.

Sending *keys* to the Talents, so sly and so slick,

A network *packet capture* must reveal the trick.

Through data and bytes, the sleuth seeks the sign,

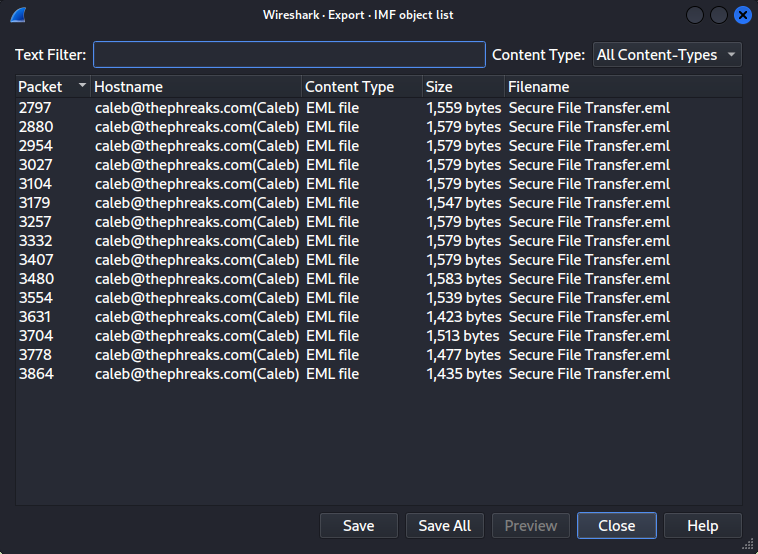
*Decrypting messages*, crossing the line.

The traitor unveiled, with nowhere to hide,

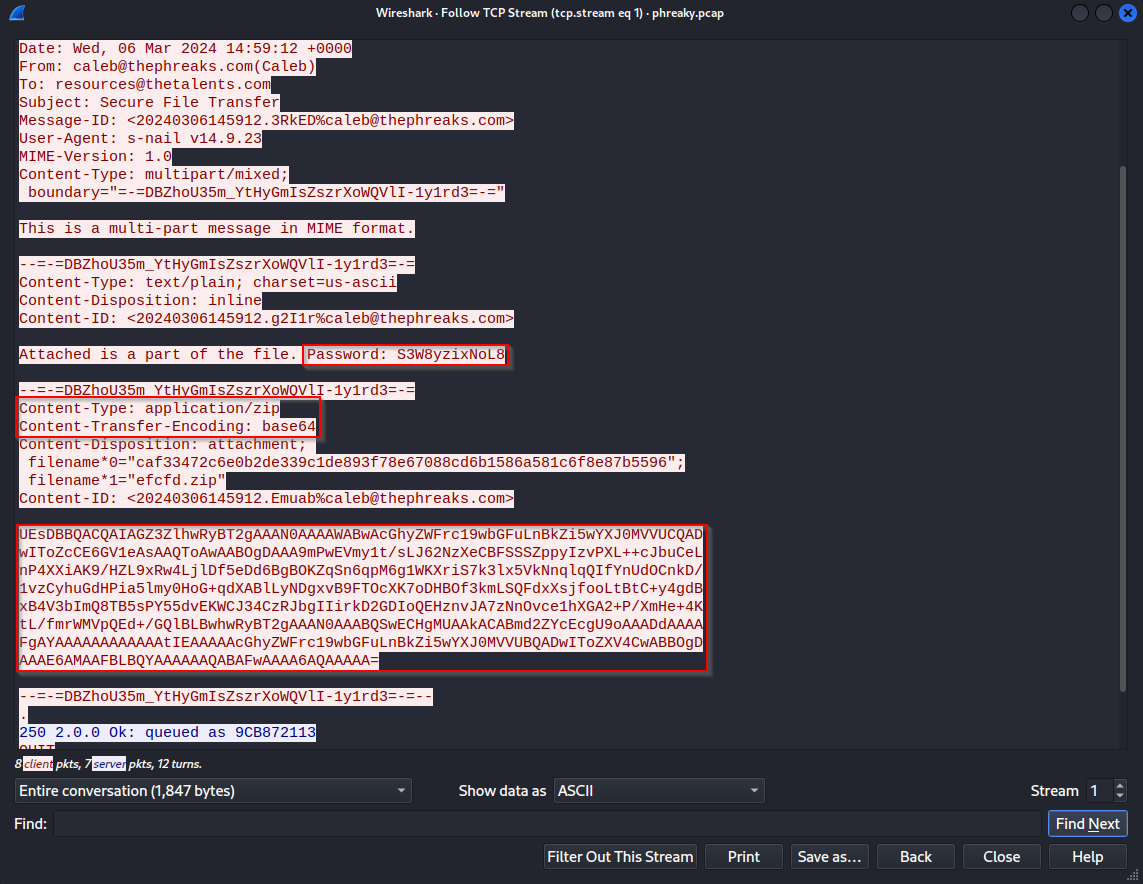
Betrayal confirmed, they'd no longer abide.”

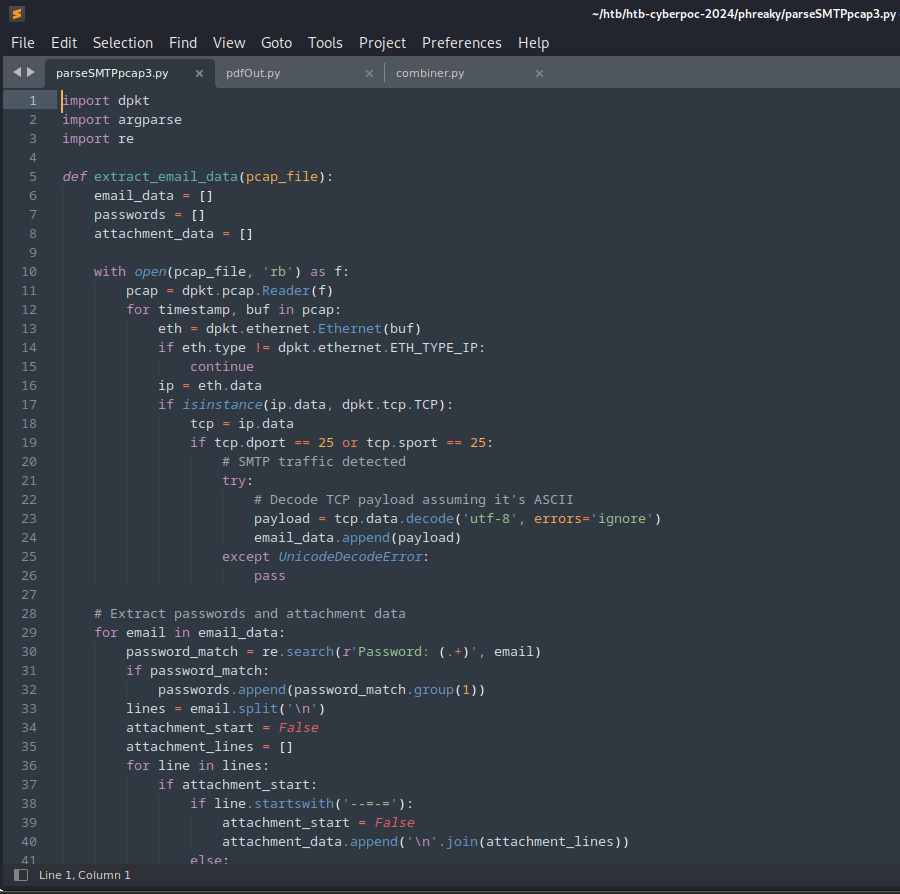
Summary: This challenge seemed to resemble another exfiltration scenario. We’re provided with a packet capture file which observes 15 emails. Within these email files we find password protected .zip files and a corresponding password in the body of the email. After unzipping the attachments we’re left with 15 pieces of an apparent .pdf file. When reassembled the full PDF reveals the flag value in plaintext. I took this challenge as an opportunity to practice python scripting and learned a lot.

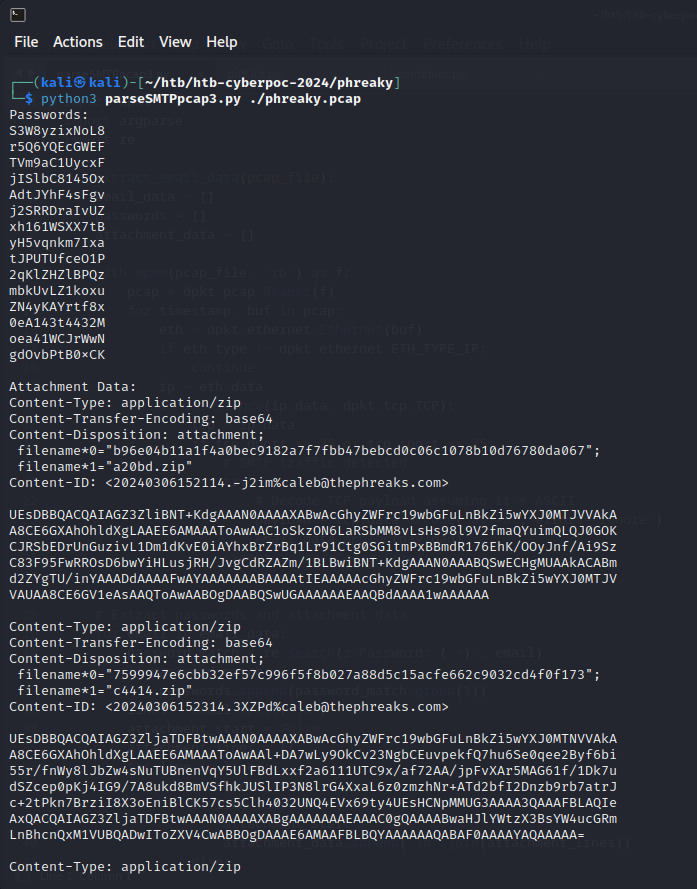
Upon initial analysis of the .pcap file contained in the challenge .zip I noticed some SMTP traffic stick out:   

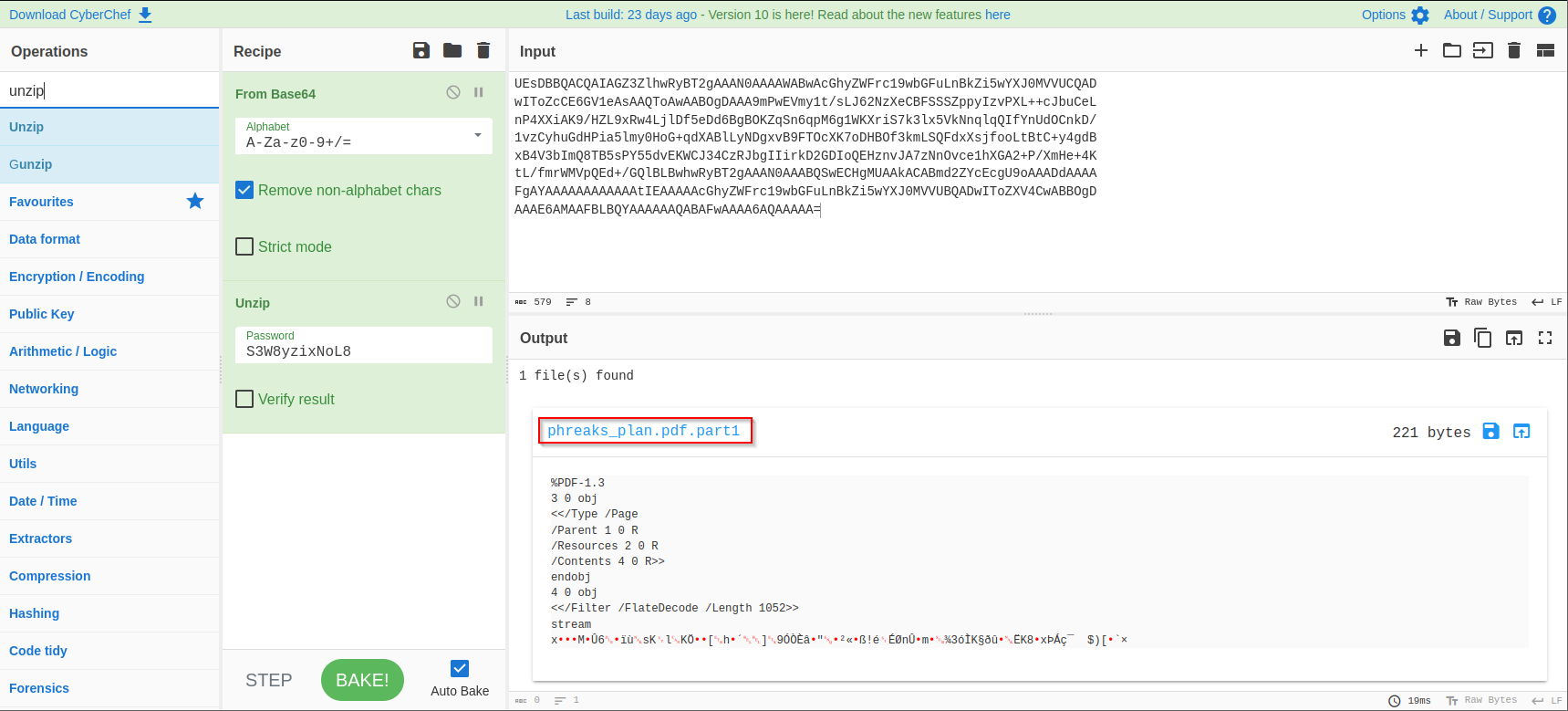

Looking further into this and I noticed 15 exportable IMF (.eml) files available:  


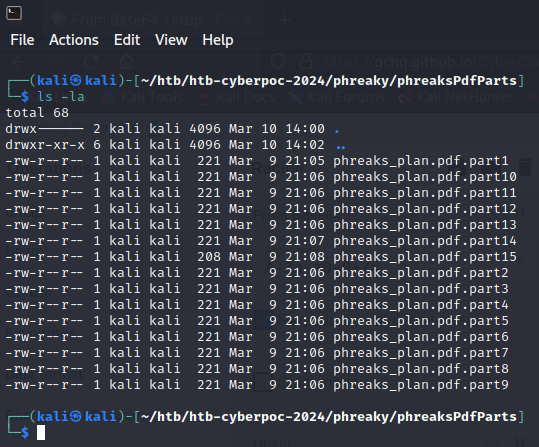
From here I inspected some of the emails by following their TCP streams and noticed some interesting content in the body and attachment of each email indicating a password string and zip file respectively:

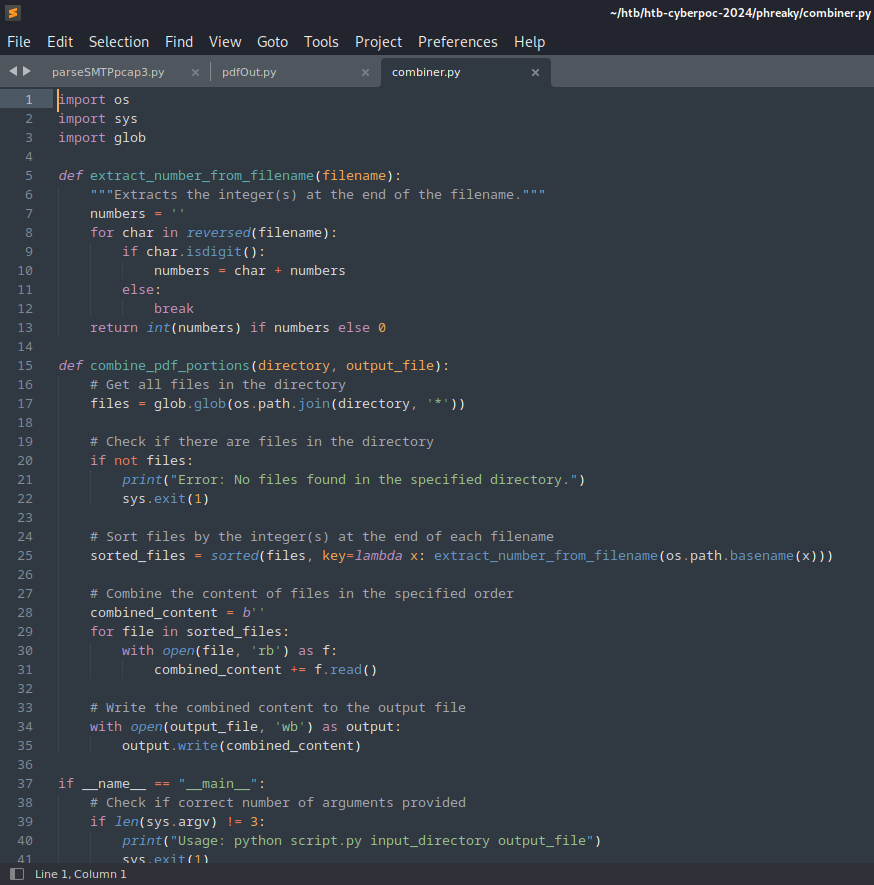
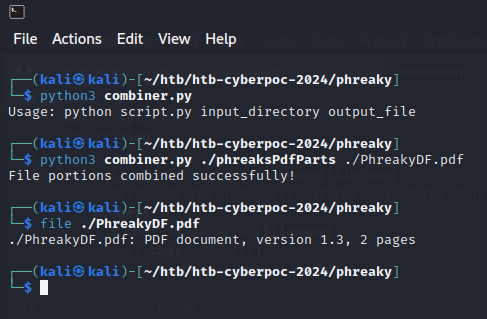
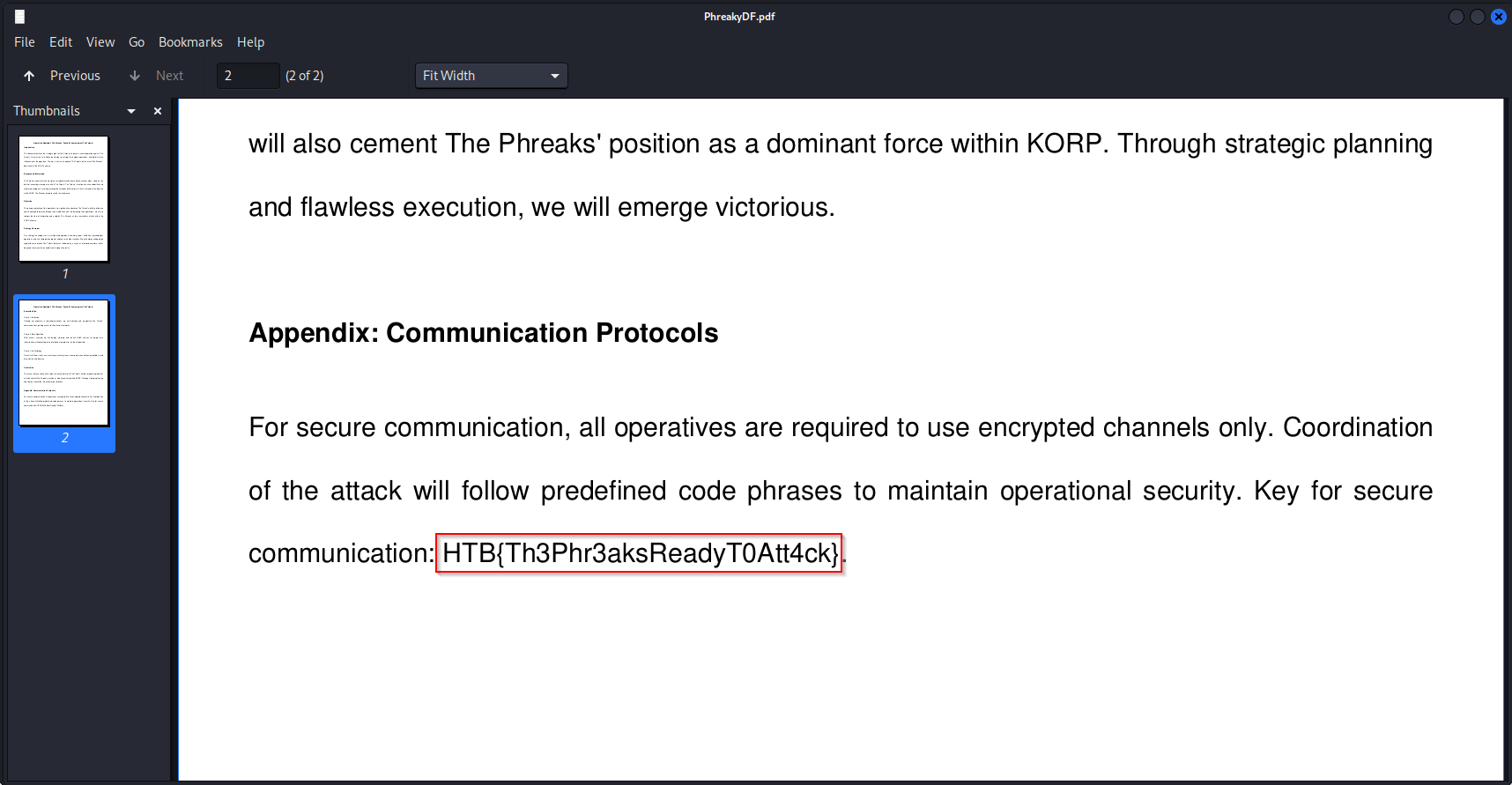


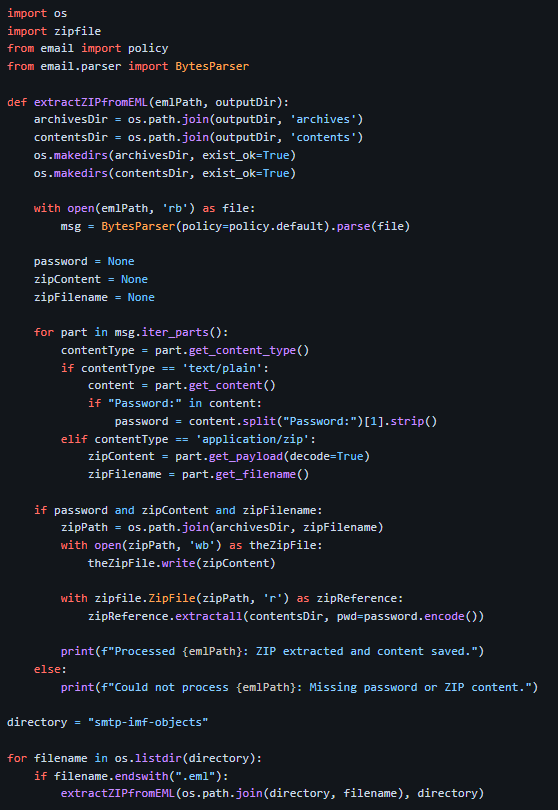
Each email followed this format so I decided to automate extraction of the email password and attachments using a python script after some trial and error:  




I spent some time trying to script decoding then unzipping each attached file with the respective password but ended up performing that step manually after significant trial and error:  


The result was 15 separate parts of what appeared to be a PDF file:  


I tried to challenge myself again to automate combining these file parts into a whole and came up with another python script (this was…. challenging for me LOL):  
  
  
I finally got it!  
  
  
After opening that PDF file I read through all their ‘plans’ and found the flag value in plaintext at the bottom:  


Flag: HTB{Th3Phr3aksReadyT0Att4ck}   
  
Note: following the conclusion of the event I found a writeup from another competitor that detailed how the full automation of extracting the pieces and combining could be done. I was impressed to learn how it could be accomplished with some proper code.  
  
(source: github/pspspsps-ctf/writeups…)

===============================================================

I ended up trying another medium and hard difficulty challenge during the rest of the remaining time of the CTF, but was unable to gather the full flag values for either.

Overall, the challenges and camaraderie were fantastic and I look forward to more opportunities to compete with the Dumpster Fire Banshees in the future!