William Hupp
Forensics Part 2
CIS 5930 Assignment 5

Forensics Part II

In Ur Reddit Crackin Ur Passwds

20 points

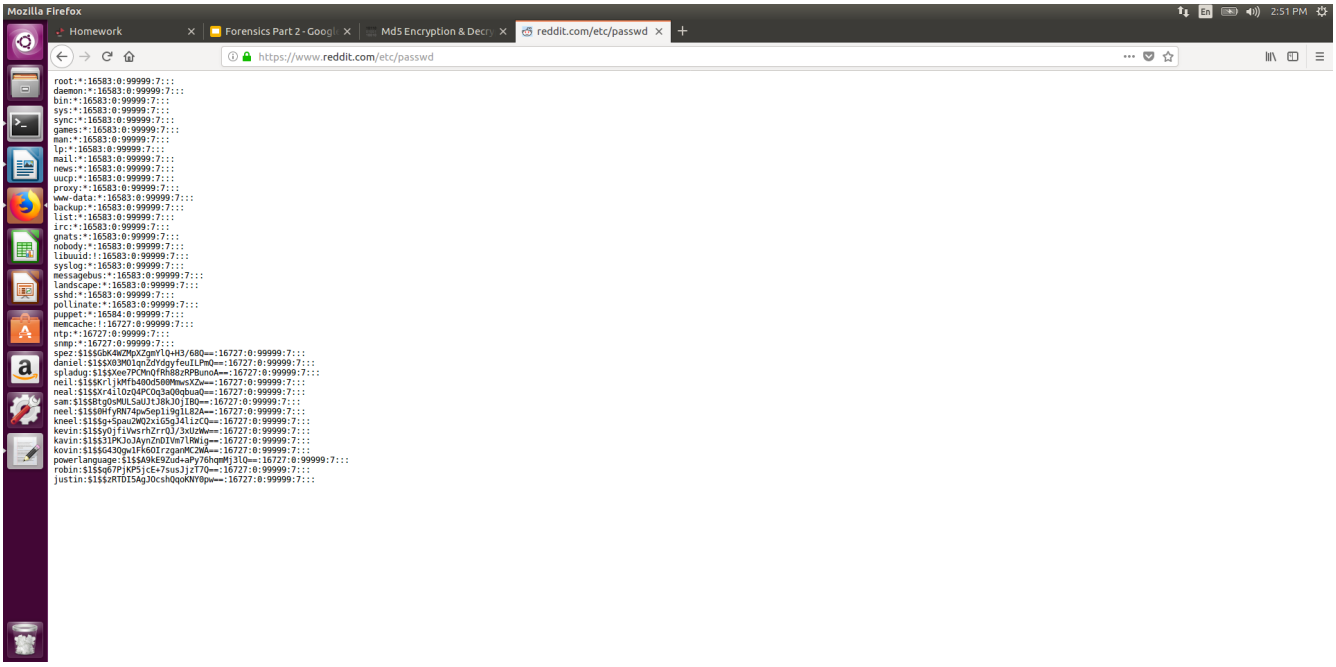
Problem:

Reddit left their /etc/passwd file exposed to the internet. I want to you get me all the listed passwords. In your write up please submit the passwords for spez, daniel, spladug, neil, neal, sam, neel, kneel, kevin, kavin, kovin, powerlanguage, robin, and justin. You can check your work by submitting neil's password to the CTF site using the format flag{neil's_password}. For an interesting easter egg, try accessing their /etc/passwd file while you are logged into reddit.

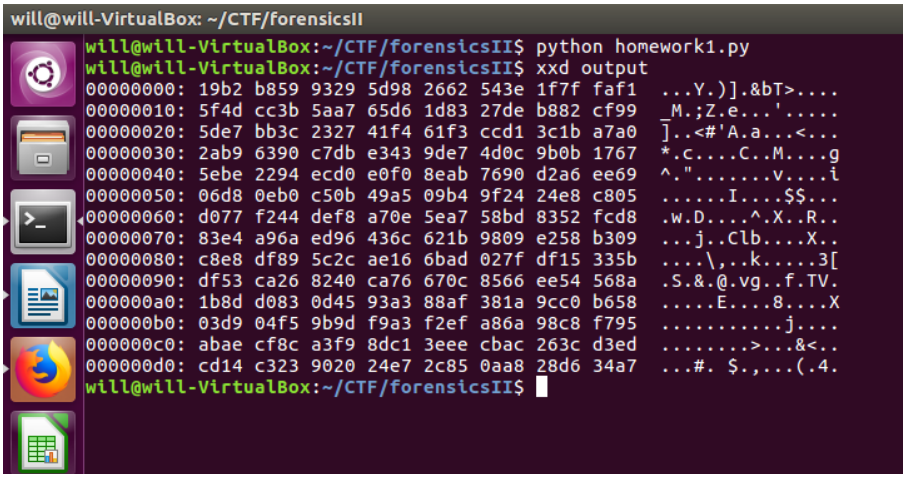
www.reddit.com/etc/passwd

Solution:

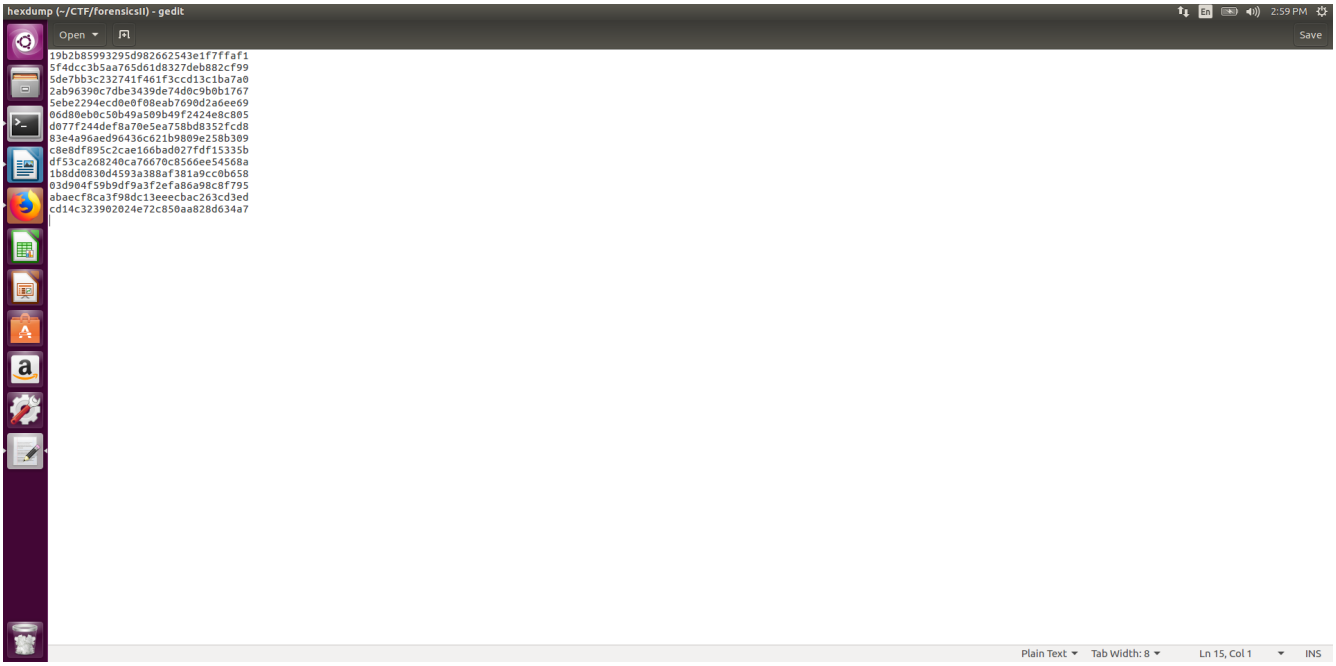
The website is as follows:



This shows a lot of unnecessary data and list of users in the form “USER_NAME1 $BASE_64_ENCODED_STRING”, followed by more unnecessary data.
Jacob informed me that $1$$ indicates that it is md5 encoded. Inside the $$ would be the the salt value. Therefore, there is no salt value. I realized what follows $1$$ is base64 encoded because it is padded with ==. I tried base64 decoding each string and running that through an md5 decoder, but that did not work. It wasn’t working because the base64 decoder was giving unprintable data and therefore the incorrect hash was being searched. To fix this, I wrote a python script which I attached to this report as ‘reddit.py’. In the script, I created an array of all the base64 strings, then I base64 decode the strings and write them to a file. To avoid the unprintable characters, I simply hexdump the file using ‘xxd’ and use the base64 decoded string’s hex value to look up the md5 hash. The xxd dump is below



I proceeded to write the xxd output to a file using ‘xxd output > somefile’. The ‘output’ file is the file I wrote all my base64 decoded strings to. In ‘somefile’ I removed all characters that were not hex and also removed all white spaces.



I proceeded to copy the hex into an md5 decoder found at http://md5decrypt.net and got the following passwords.

Spez = shill
Daniel = password
Spladug = yee
Neil = hunter2
Neal = secret
Sam = dog
Neel = cat
Kneel = fish
Kevin = garbage
Kavin = computer
Kovin = fish2
Powerlanguage = eggdog
Robin = bird
Justin = case

flag = flag{hunter2}

Bobby Bowden Is Still Topical, Right?

20 points

Problem:

Greetings IT intern! Bobby Bowden forgot his logon password. In between coffee runs could you please find the time to recove it for him?

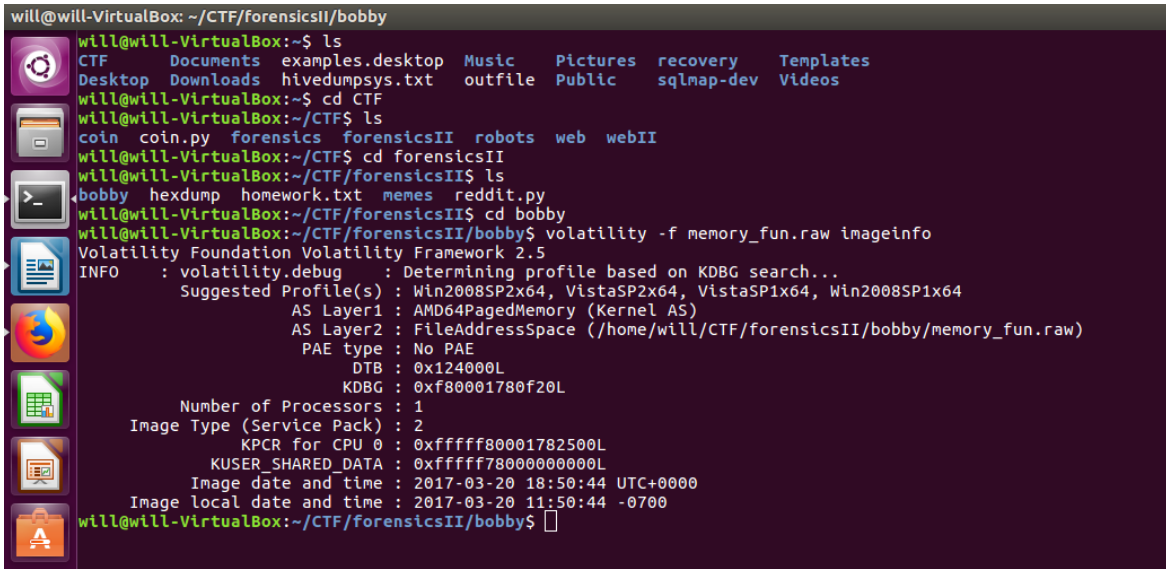
The memory capture is available here: https://drive.google.com/open?id=1E1sDeF_fgQjIDjkG-k3mg8p6KJEw5YCx

Please be aware the compressed file is 1.4GB and uncompressed it is 2GB. SHA256 checksum: 5cf8a9d3ecd420f1f66d26fdae4894b8c62812a15c714928ed6b2e67841bb78f

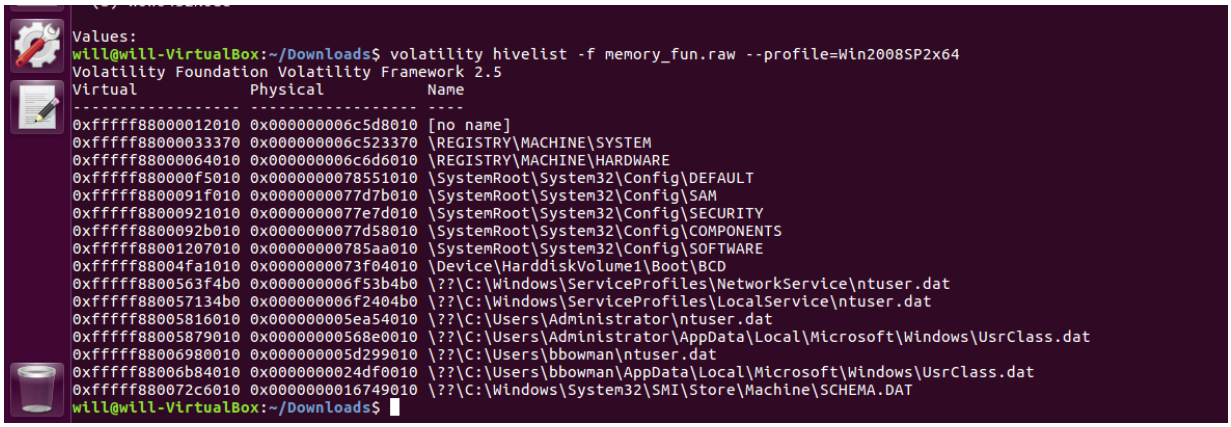
Note, when you recover the password, submit it to the CTF site in the format flag{recovered_password}.

Solution:

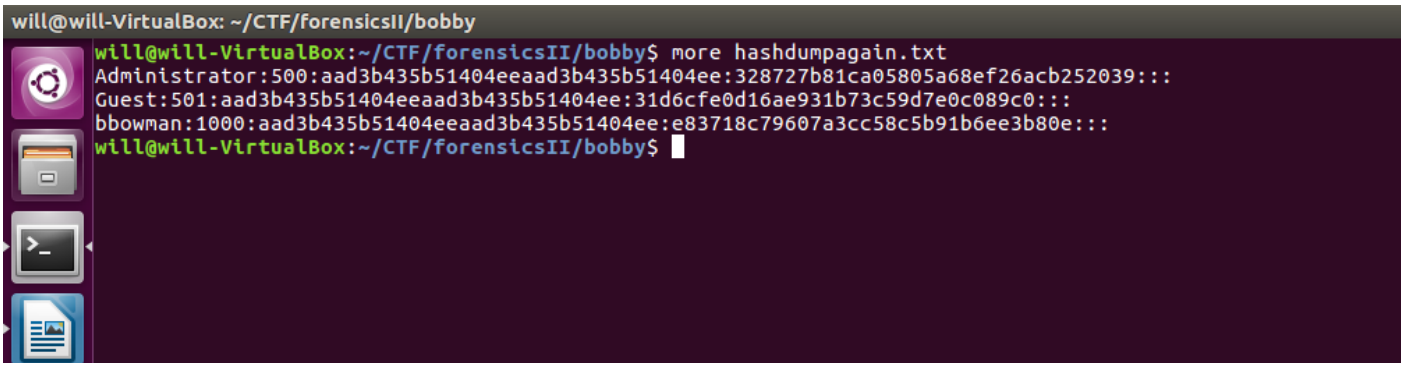
Firstly, I downloaded the file and extracted the compressed file. I then analyzed the notes and found the writeup at https://cyberarms.wordpress.com/2011/11/04/memory-forensics-how-to-pull-passwords-from-a-memory-dump/ extremely useful for helping me to walk through this problem. I installed volatility because this is a memory forensics problem. I attempted to get information on the system image using the command 'volatility -f memory_fun.raw imageinfo'.



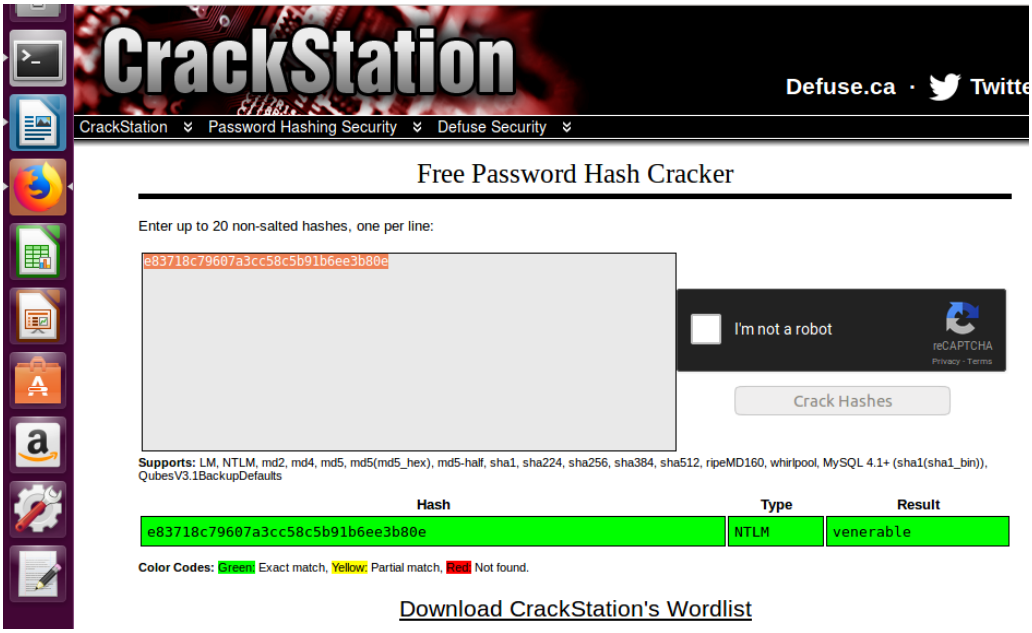
This informed me that the profile type is Win2008SP2x86. This information is useful for finding hives and executing volatility commands. I tried using the 'dumpregistry' command but that yielded no results. I also tried 'printkey' command but the keys didn't yield any information on the password hash. However, it did point out there was a password file. I then used the command 'volatility hivelist -f memory_fun.raw –profile=Win2008SP2x64' to get the hives. This yielded the following output.



Upon inspection of the notes, I learned that the passwords are stored in the SAM and SYSTEM hives. I tried the 'hivedump' command. I output the command to a text file and grepped Password on the files for different SYSTEM and SAM hives. I didn't find anything interesting. I then used the command 'volatility hashdump -f memory_fun.raw --profile=Win2008SP2x64 -y 0xfffff88000033370 -s 0xfffff8800091f010 > hashdump.txt' I found this command on the walk through on Cyberarms. The command is in the format of -y ADDRESS_SYSTEM_HIVE -s ADDRESS_SAM_HIVE. This prints the hash of the SYSTEM and SAM. Upon opening 'hashdump.txt' I see a hash for Bobby Bowden.



According to the notes, windows hashes are typically in NTLM. I tried copying everything after :1000 into a decoder found at https://crackstation.net/. This yielded no results. I then tried the characters in between '1000:' and the next ':'. This also yielded no results. After that, I tried the next string of characters starting after the second ':' and before ':::' This worked. The string 'e83718c79607a3cc58c5b91b6ee3b80e' decoded gives the output venerable.



Flag: flag{venerable}

Dank Memes and Broken Dreams

30 points

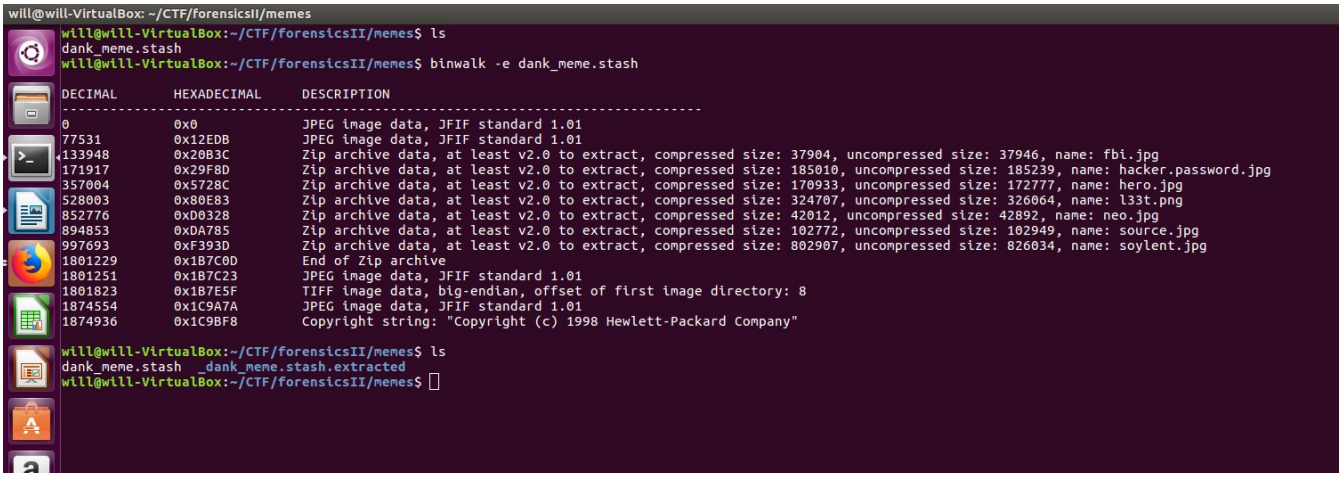
Problem:

When I grow up I'm going to be the l33test h4x0r with the dankest memes. Just try and find my flag.

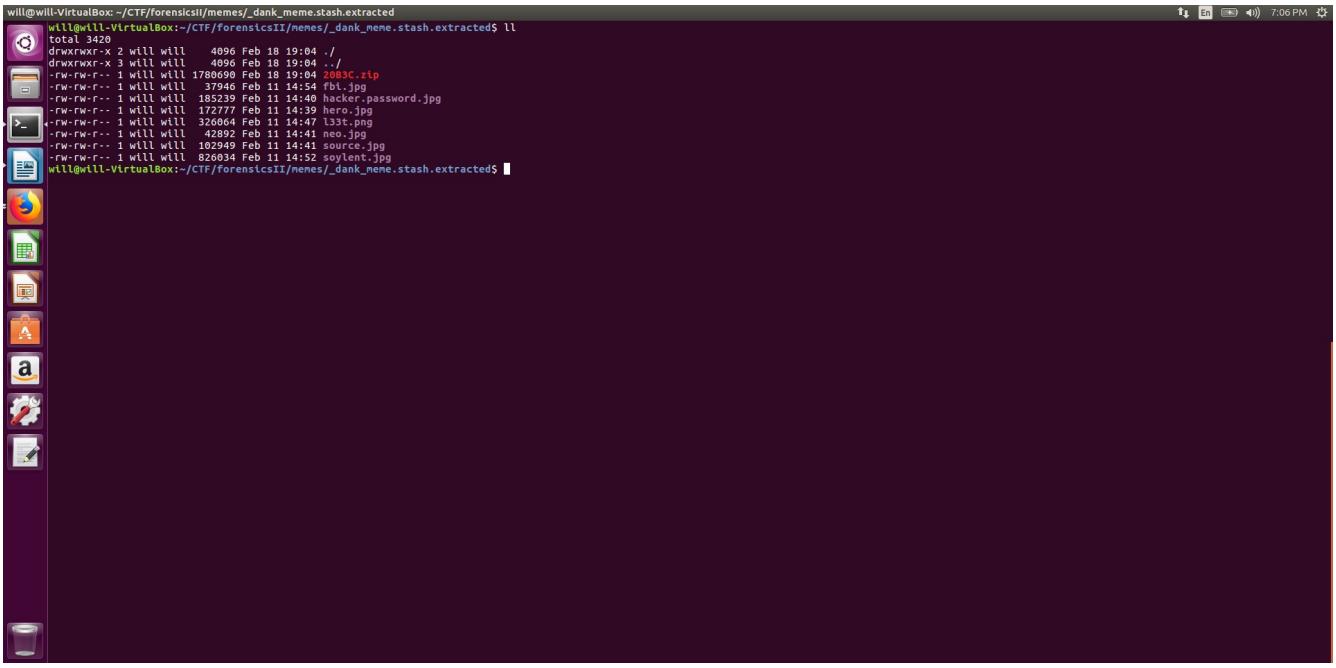
My stash can be found at https://drive.google.com/open?id=1AOvBLP88f6ohg0dz7-49pdrjnqSi0tT2

Solution:

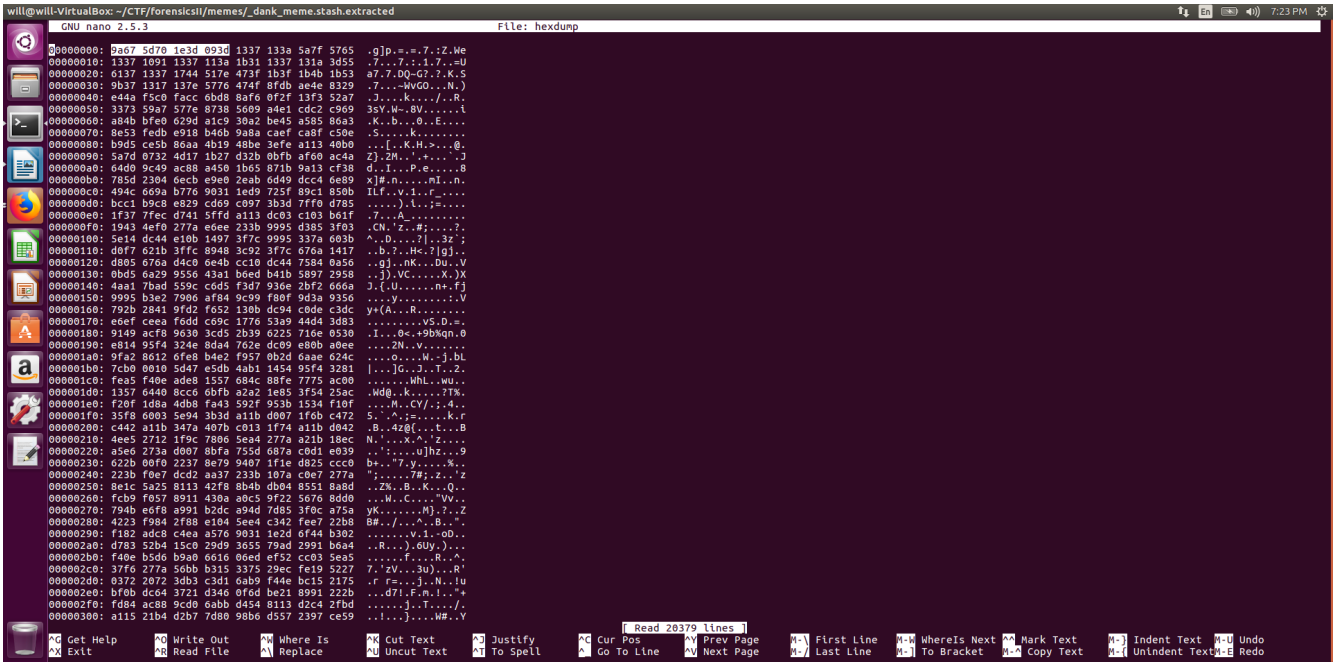
First, I downloaded the image. The image is called ‘dank_meme.stash’. Since I knew there could be stuff hidden in the image, I carved the image. I used binwalk as my carving tool. I ran the command ‘binwalk -e dank_meme.stash’.



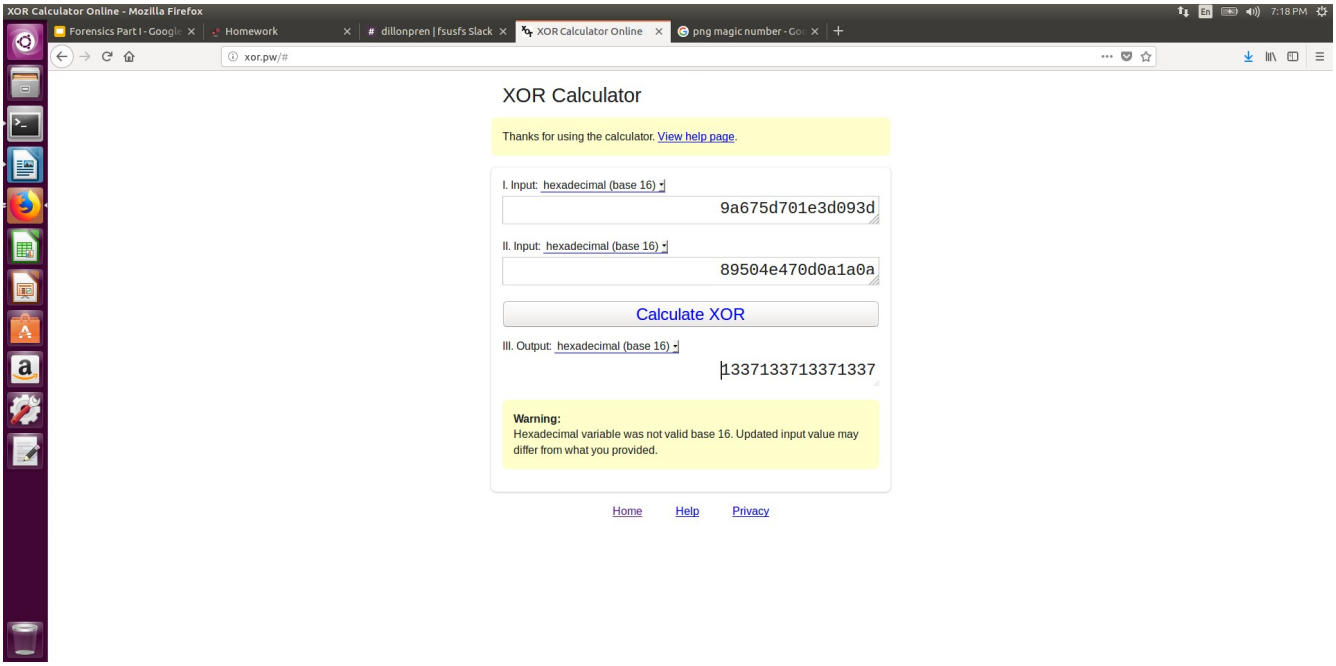
This created a folder called '_dank_meme.stash.extracted' Inside the dankmeme.stash.extracted folder there was a bunch of jpgs and a png file.



The jpgs are all memes. However, the file 'l33t.png' would not open. I ran the file command on it and learned that it is not a png. It is binary data. Looking back at the Forensics week 1 slides, I see an example of a jpeg that is binary data. I will use the principles from that example to solve the rest of the problem. Assuming that 'l33t.png' is xor encryed, I must find the key to decrypt it. Assuming the file is actually a png, I took the first 8 bytes of the file 'l33t.png' and xor'd them with the magic number for a png. I took the first 8 bytes of l33t.png because the first 8 bytes of a file are usually the magic number. To view the bytes of l33t.png I hex dumped the png file using the command 'xxd l33t.png > hexdump'. Hexdump is shown below.



Inside the hexdump file I copied the first 8 bytes and used the xor tool found at www.xor.pw between those 8 bytes and the magic number for a png file. A simple Google search told me that the magic number for a png file is 89 50 4e 47 0d 0a 1a 0a.



This told me that the key is 1337. Since 1337 was repeating, I figured for even bytes the key is 13 and for odd bytes the key is 37. I modified the script given in the inclass examples to match these keys. I wrote the output of the decryption to output.png. After running the script and opening output.png I see the flag. The script is attached to my submission as xor.py.

Flag - flag{4m_11_l33t_Y3t}

Unsolved Equation Group

30 points

Problem:

We recently got our hands on a slice of NSA network traffic in which two of their operatives exchange a file related to the agency's biggest mystery. Access the file and see if you can beat them to the solution.

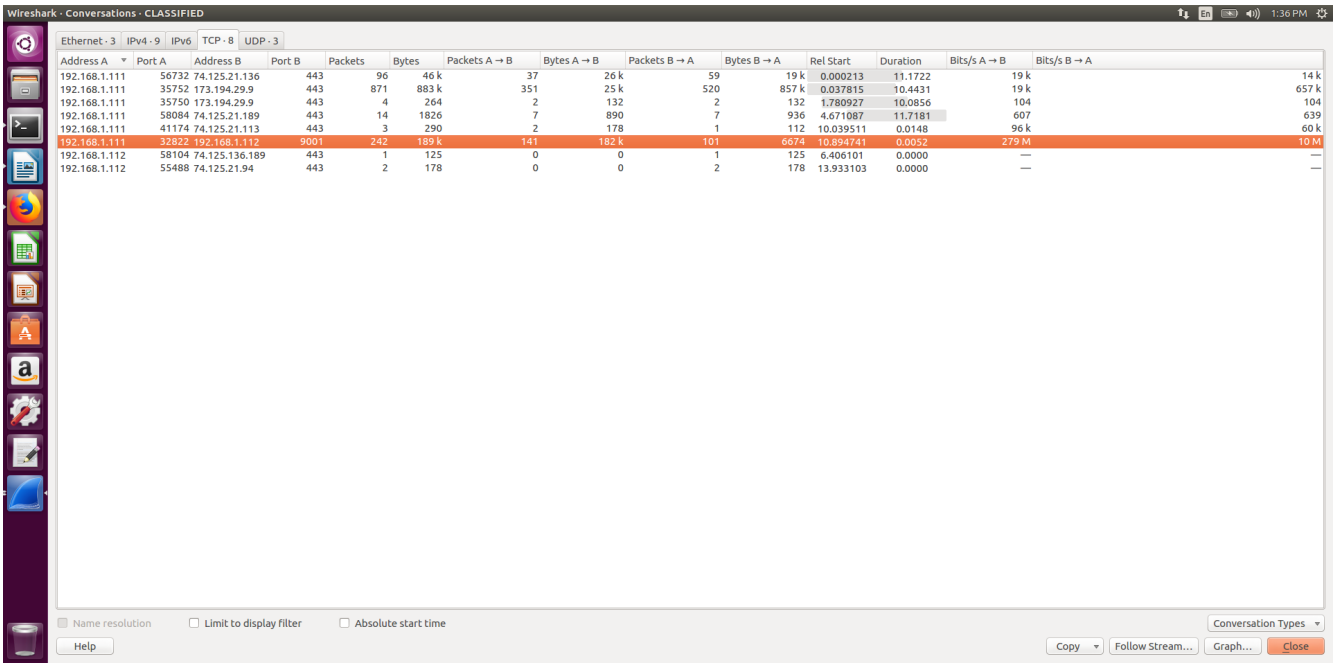
The intel can be found at https://drive.google.com/open?id=1jGJGFfmRPShJ0JTq5UMcisI26MYnqUqq

Hint: The pcap contains internal and external network traffic. Machines on the internal network are likely to have similar IP Addresses.

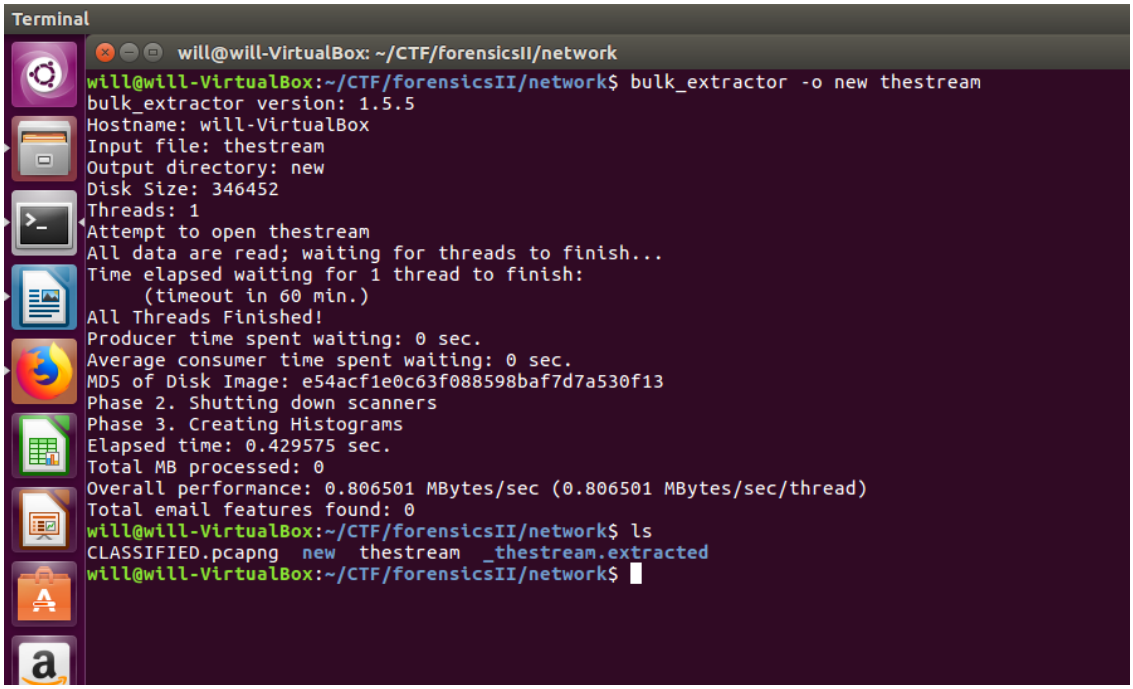
Solution:

Since this is a networking problem, I assumed I would have to use Wireshark. I opened up the pcapng In Wireshark. Looking at the notes on Wireshark, I searched for packets containing the string ‘flag’. After that did not work, I went to statistics and viewed conversations. I noticed the hint in the problem that internal traffic uses similar IP addresses. Since I was looking for a conversation between two agents, I figured the conversation I needed would be internal traffic. Since files are transferred using

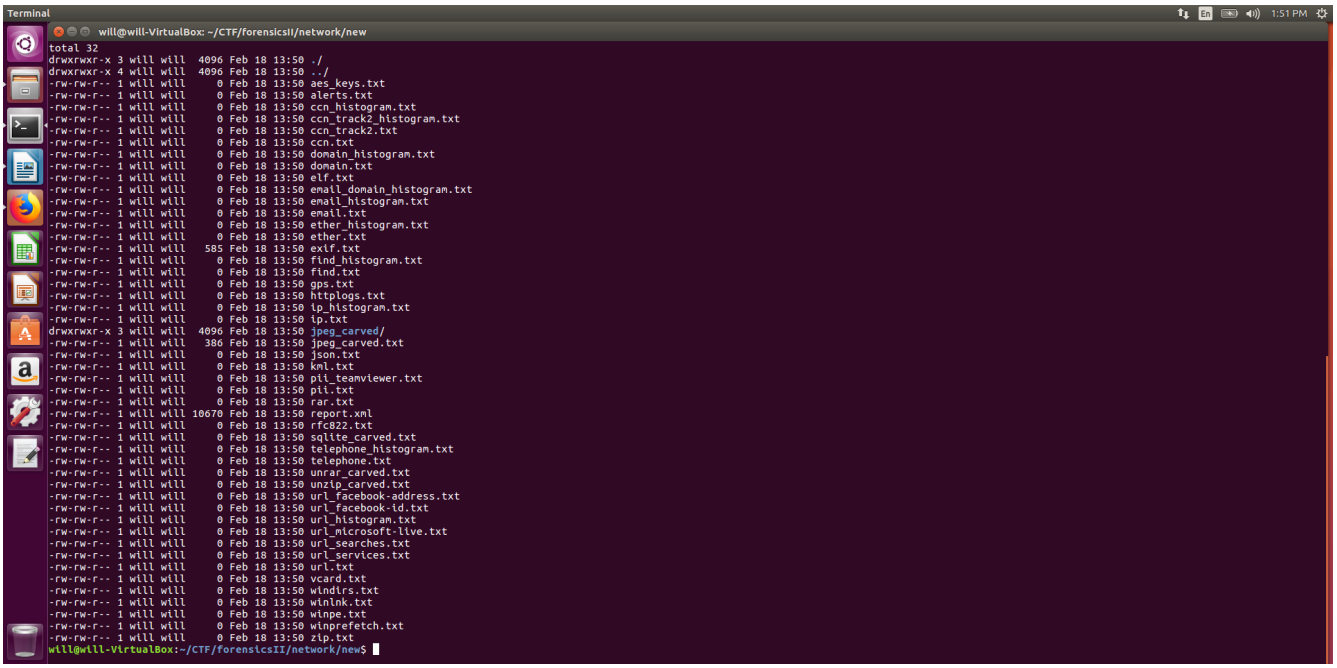
TCP, I looked at TCP conversations. There was only one TCP conversation that contained similar IP addresses. The two IP addresses were 192.168.1.111 and 192.168.1.112. I followed that stream and saved it to my computer.



I saved the stream as ‘thestream’. I tried carving the string using ‘binwalk -e thestream’. That extracted 12 sit files but,after a lot of playing around with them, I found nothing. I thought the names were a hex representation of the flag, but they were not. I then decided to try a tool I learned about on my own, ‘bulk_extractor’. I ran the command ‘bulk_extractor -o new thestream’ to bulk_extractor the TCP stream file and put the contents of the extraction in the ‘new’ directory.

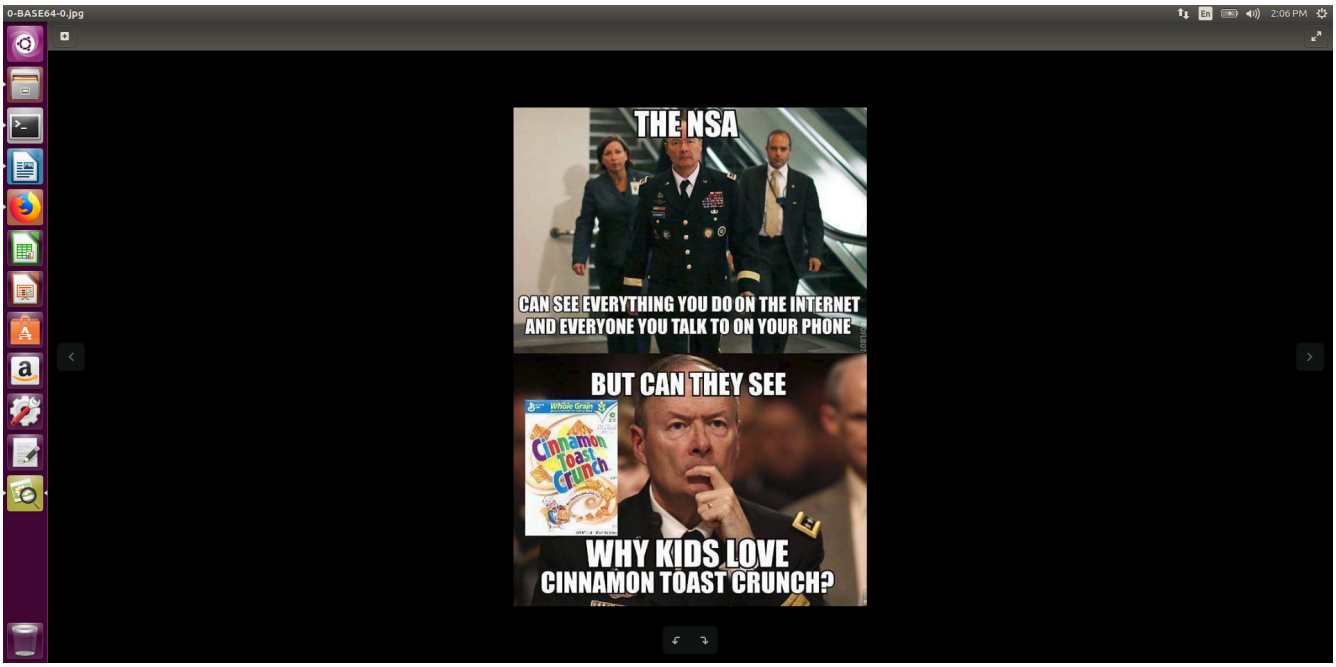


Inside the new directory I noticed a directory called ‘jpeg_carved’.

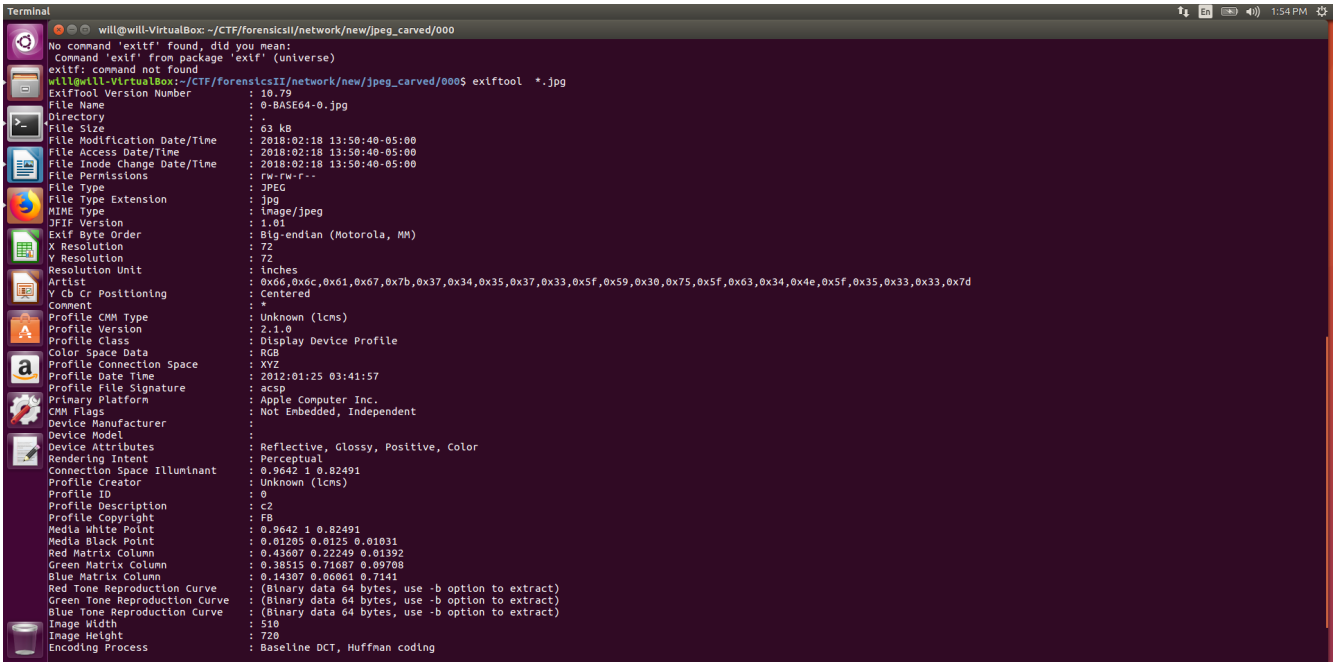


This contained a jpeg image of a meme.

```
bash: cd: jpeg_carved: No such file or directory
will@will-VirtualBox:~/CTF/forensicsII/network/new$ ls
aes_keys.txt              domain_histogram.txt        ether_histogram.txt  httplogs.txt      kml.txt             sqlite_carved.txt         url_facebook-id.txt     vcard.txt
alerts.txt                domain.txt                  ether.txt            ip_histogram.txt  pii_teamviewer.txt  telephone_histogram.txt   url_histogram.txt       windirs.txt
ccn_histogram.txt         elf.txt                     exif.txt             ip.txt            pii.txt             telephone.txt             url_microsoft-live.txt  winlnk.txt
ccn_track2_histogram.txt  email_domain_histogram.txt  find_histogram.txt   jpeg_carved       rar.txt             unrar_carved.txt          url_searches.txt        winpe.txt
ccn_track2.txt            email_histogram.txt         find.txt             jpeg_carved.txt   report.xml          unzip_carved.txt          url_services.txt        winprefetch.txt
ccn.txt                   email.txt                   gps.txt              json.txt          rfc822.txt          url_facebook-address.txt  url.txt                 zip.txt
will@will-VirtualBox:~/CTF/forensicsII/network/new$ cd jpeg_carved
will@will-VirtualBox:~/CTF/forensicsII/network/new/jpeg_carved$ ls
000
will@will-VirtualBox:~/CTF/forensicsII/network/new/jpeg_carved$ cd 000
will@will-VirtualBox:~/CTF/forensicsII/network/new/jpeg_carved/000$ ls
0-BASE64-0.jpg
will@will-VirtualBox:~/CTF/forensicsII/network/new/jpeg_carved/000$
```



The file is called 0-BASE64-o.jpg. However there was no base64 encodings in the file. Looking back at the week 1 forensics notes, I see there is a tool called ‘exiftool’ that displays information on a jpeg file. I ran the command ‘exiftool *.jpg’ and got the following output.



In the artist section, there is a sequence of hex values. After running that hex string through a hex to ascii converter found at https://www.rapidtables.com/convert/number/hex-to-ascii.html I get the flag.

Flag - flag{{74573_Y0u_c4N_533}