

Forensics Part II

In Ur Reddit Crackin Ur Passwds

20 points

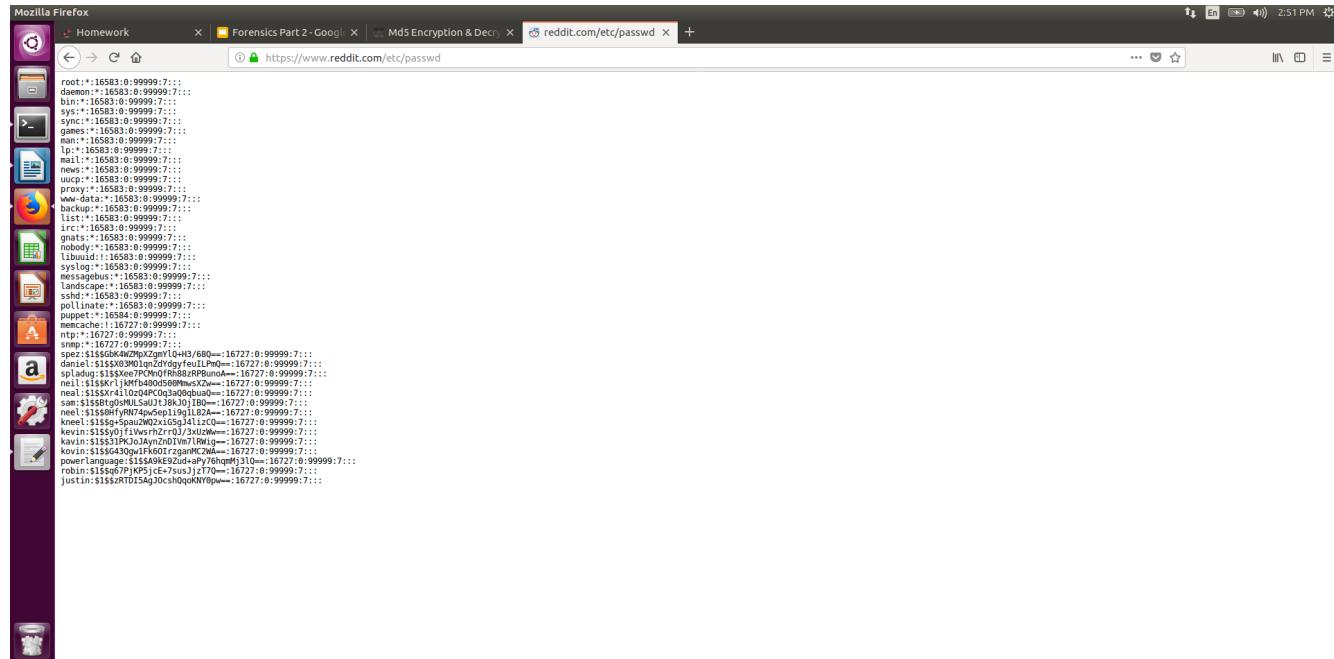
Problem:

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:

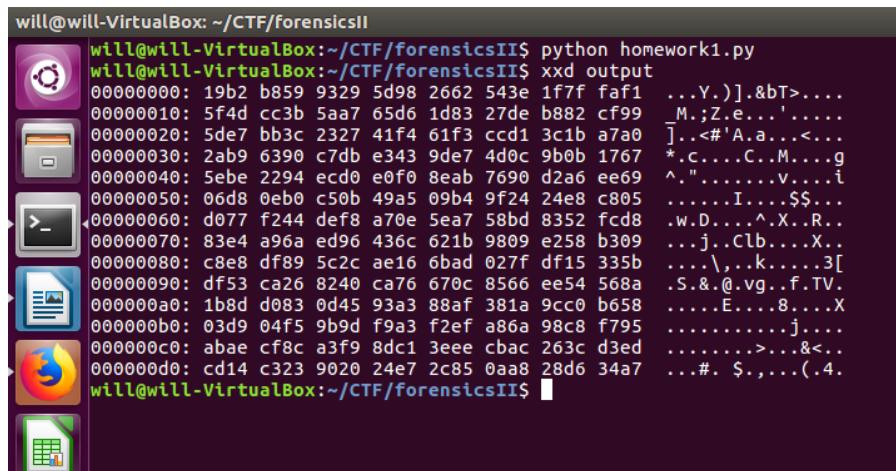


A screenshot of a Mozilla Firefox browser window. The address bar shows the URL <https://www.reddit.com/etc/passwd>. The page content displays the /etc/passwd file for various users. The file includes entries for root, daemon, bin, sys, sync, games, man, lp, mail, news, proxy, www-data, nobody, libuid, syslog, messagebus, landscape, sssd, puppet, memcache, mrtg, and snmp. Below this, specific user entries are listed: spez, daniel, spladug, neil, neal, sam, neel, kneel, kevin, kavin, kovin, powerlanguage, robin, and justin. Each user entry consists of a username followed by a password (all set to '16583:0:99999:7::'), a uid (e.g., 16583), a gid (e.g., 0), and a list of shell paths (e.g., '/bin/false'). The entire file is encoded in base64.

```
root:16583:0:99999:7::  
daemon:16583:0:99999:7::  
bin:16583:0:99999:7::  
sys:16583:0:99999:7::  
sync:16583:0:99999:7::  
games:16583:0:99999:7::  
man:16583:0:99999:7::  
lp:16583:0:99999:7::  
mail:16583:0:99999:7::  
news:16583:0:99999:7::  
proxy:16583:0:99999:7::  
www-data:16583:0:99999:7::  
nobody:16583:0:99999:7::  
list:16583:0:99999:7::  
irc:16583:0:99999:7::  
gnome:16583:0:99999:7::  
nobody:16583:0:99999:7::  
libuid:16583:0:99999:7::  
syslog:16583:0:99999:7::  
messagebus:16583:0:99999:7::  
landscape:16583:0:99999:7::  
sssd:16583:0:99999:7::  
puppet:16583:0:99999:7::  
memcache:16727:0:99999:7::  
mrtg:16727:0:99999:7::  
snmp:16727:0:99999:7::  
spez:$1$GQk4WZqoX2p104bJ6Bm--:16727:0:99999:7::  
daniel:$1$5x0e7PCNj0fR8bzRPbmno--:16727:0:99999:7::  
spladug:$1$5x0e7PCNj0fR8bzRPbmno--:16727:0:99999:7::  
neil:$1$5Kt1jMfH40d580mmsXzw--:16727:0:99999:7::  
neal:$1$5Rt0phULSalJ1J8kDlEo--:16727:0:99999:7::  
sam:$1$5Rt0phULSalJ1J8kDlEo--:16727:0:99999:7::  
neel:$1$5Rt0phULSalJ1J8kDlEo--:16727:0:99999:7::  
kneel:$1$5s6+SpWv02x1Os5j1LzQm--:16727:0:99999:7::  
kevin:$1$5s6+SpWv02x1Os5j1LzQm--:16727:0:99999:7::  
kavin:$1$5s6+SpWv02x1Os5j1LzQm--:16727:0:99999:7::  
kovin:$1$5s6+SpWv02x1Os5j1LzQm--:16727:0:99999:7::  
powerlanguage:$1$5s6+SpWv02x1Os5j1LzQm--:16727:0:99999:7::  
robin:$1$5s6+SpWv02x1Os5j1LzQm--:16727:0:99999:7::  
justin:$1$5s6+SpWv02x1Os5j1LzQm--:16727:0:99999:7::
```

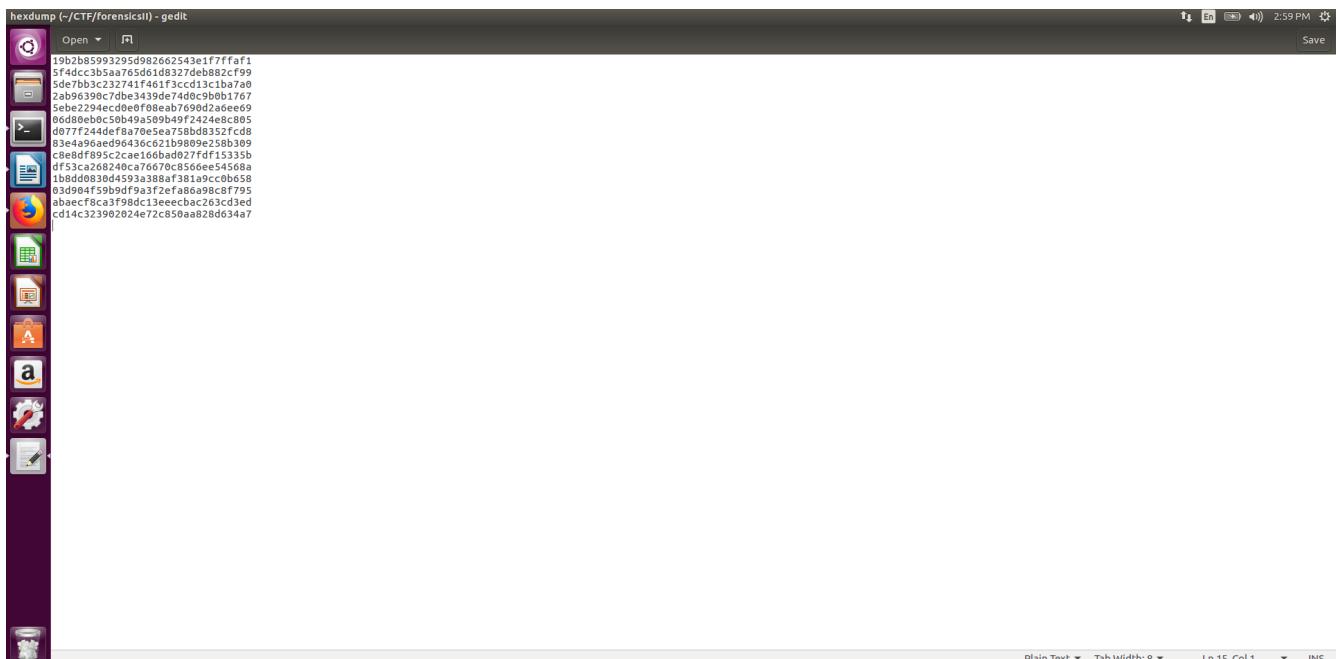
This shows a lot of unnecessary data and list of users in the form “USER_NAME\$1\$
\$BASE_64_ENCODED_STRING”, followed by more unnecessary data.

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



```
will@will-VirtualBox:~/CTF/forensicsII$ python homework1.py
will@will-VirtualBox:~/CTF/forensicsII$ xxd output
00000000: 19b2 b859 3295d982662543e1ff7faf1 ...
00000010: 5f4dc3b5aa765d61d8327de882cf99 ...
00000020: 5de7bb3c232741f461f3cccd13c1ba7a0 ...
00000030: 2ab9 6390 c7db e343 9de7 4d0c 9b0b 1767 ...
00000040: 5ebe 2294 ec00 e0f0 8eab 7690 d2a6 ee69 ...
00000050: 06d8 0eb0 c50b 49a5 09b4 9f24 24e8 c805 ...
00000060: d077 f244 def8 a70e 5ea7 58bd 8352 fcd8 ...
00000070: 83e4 a96a ed96 436c 621b 9809 e258 b309 ...
00000080: c8e8 df89 5c2c ae16 6bad 027f df15 335b ...
00000090: df53 ca26 8240 ca76 670c 8566 ee54 568a ...
000000a0: 1b8d d083 0d45 93a3 88af 381a 9cc0 b658 ...
000000b0: 03d9 04f5 9b9d f9a3 f2ef a86a 98c8 f795 ...
000000c0: abae cf8c a3f9 8dc1 3eee cbac 263c d3ed ...
000000d0: cd14 c323 9020 24e7 2c85 0aa8 28d6 34a7 ...
will@will-VirtualBox:~/CTF/forensicsII$
```

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.



```
19b2b8593295d982662543e1ff7faf1
5f4dc3b5aa765d61d8327de882cf99
5de7bb3c232741f461f3cccd13c1ba7a0
2ab96390c7db e343 9de7 4d0c 9b0b 1767
5ebe2294ec00 e0f0 8eab 7690 d2a6 ee69
06d80eb0c50b 09b4 9f24 24e8 c805
d077f244def8a70e 5ea7 58bd 8352 fcd8
83e4a96aed96436c621b9809e258b309
c8e8df895c2cae16db0<27fd15335b
df53ca268240ca760c8566ee54568a
1b8dd0830d4593a388af381a9cc0b658
03d904f59b9df9a3feefabaa98c8f795
abae cf8ca3f98d13eeecbac263cd3ed
cd14c323902024e72c850aa828d634a7
```

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?

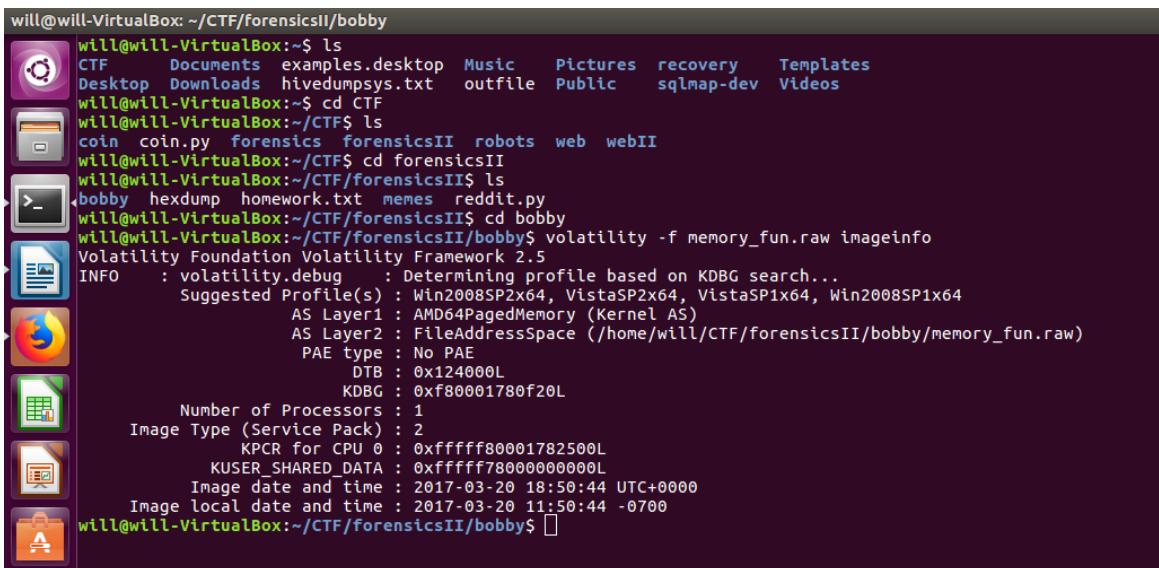
The memory capture is available here: https://drive.google.com/open?id=1E1sDeF_fgQjIDjkG-k3mg8p6KJEw5YCx

Please be aware the compressed file is 1.4GB and uncompressed it is 2GB. SHA256 checksum: 5cf8a9d3ecd420f1f66d26fd4e4894b8c62812a15c714928ed6b2e67841bb78f

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’.



```
will@will-VirtualBox: ~/CTF/forensicsII/bobby
will@will-VirtualBox:~$ ls
CTF      Documents examples.desktop  Music    Pictures   recovery   Templates
Desktop  Downloads hivedumpsys.txt  outfile  Public    sqlmap-dev Videos
will@will-VirtualBox:~/CTF$ ls
coin  coin.py  forensics  forensicsII  robots  web  webII
will@will-VirtualBox:~/CTF$ cd forensicsII
will@will-VirtualBox:~/CTF/forensicsII$ ls
bobby  hexdump  homework.txt  memes  reddit.py
will@will-VirtualBox:~/CTF/forensicsII$ cd bobby
will@will-VirtualBox:~/CTF/forensicsII/bobby$ volatility -f memory_fun.raw imageinfo
Volatility Foundation Volatility Framework 2.5
INFO    : volatility.debug    : Determining profile based on KDBG search...
Suggested Profile(s) : Win2008SP2x64, VistaSP2x64, VistaSP1x64, Win2008SP1x64
                      AS Layer1 : AMD64PagedMemory (Kernel AS)
                      AS Layer2 : FileAddressSpace (/home/will/CTF/forensicsII/bobby/memory_fun.raw)
                      PAE type : No PAE
                      DTB : 0x124000L
                      KDBG : 0xf80001780f20L
Number of Processors : 1
Image Type (Service Pack) : 2
                      KPCR for CPU 0 : 0xfffff80001782500L
                      KUSER_SHARED_DATA : 0xfffff78000000000L
Image date and time : 2017-03-20 18:50:44 UTC+0000
Image local date and time : 2017-03-20 11:50:44 -0700
will@will-VirtualBox:~/CTF/forensicsII/bobby$
```

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.

```

Values:
will@will-VirtualBox:~/Downloads$ volatility hivelist -f memory_fun.raw --profile=Win2008SP2x64
Volatility Foundation Volatility Framework 2.5
Virtual Physical Name
-----
0xfffff88000012010 0x000000000c5d8010 [no name]
0xfffff88000033370 0x000000000c523370 \REGISTRY\MACHINE\SYSTEM
0xfffff88000064010 0x000000006c6d6010 \REGISTRY\MACHINE\HARDWARE
0xfffff880000f5010 0x0000000078551010 \SystemRoot\System32\Config\DEFAULT
0xfffff8800091f010 0x0000000077d7b010 \SystemRoot\System32\Config\SAM
0xfffff88000921010 0x0000000077e7d010 \SystemRoot\System32\Config\SECURITY
0xfffff8800092b010 0x0000000077d58010 \SystemRoot\System32\Config\COMPONENTS
0xfffff88001207010 0x00000000785aa010 \SystemRoot\System32\Config\SOFTWARE
0xfffff88004fa1010 0x0000000073f04010 \Device\HarddiskVolume1\Boot\BCD
0xfffff8800563f4b0 0x000000006f53b4b0 \??\C:\Windows\ServiceProfiles\NetworkService\ntuser.dat
0xfffff880057134b0 0x000000006f2404b0 \??\C:\Windows\ServiceProfiles\LocalService\ntuser.dat
0xfffff88005816010 0x000000005ea54010 \??\C:\Users\Administrator\ntuser.dat
0xfffff88005879010 0x00000000568e0010 \??\C:\Users\Administrator\AppData\Local\Microsoft\Windows\UsrClass.dat
0xfffff88006980010 0x000000005d299010 \??\C:\Users\bbowman\ntuser.dat
0xfffff88006b84010 0x0000000024df0010 \??\C:\Users\bbowman\AppData\Local\Microsoft\Windows\UsrClass.dat
0xfffff880072c6010 0x0000000016749010 \??\C:\Windows\System32\SMI\Store\Machine\SCHEMA.DAT
will@will-VirtualBox:~/Downloads$ 
```

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.

```

will@will-VirtualBox:~/CTF/forensicsII/bobby
will@will-VirtualBox:~/CTF/forensicsII/bobby$ more hashdumpagain.txt
Administrator:500:aad3b435b51404eeaad3b435b51404ee:328727b81ca05805a68ef26acb252039:::
Guest:501:aad3b435b51404eeaad3b435b51404ee:31d6cfed0d16ae931b73c59d7e0c089c0:::
bbowman:1000:aad3b435b51404eeaad3b435b51404ee:e83718c79607a3cc58c5b91b6ee3b80e:::
will@will-VirtualBox:~/CTF/forensicsII/bobby$ 
```

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**.

Hash	Type	Result
e83718c79607a3cc58c5b91b6ee3b80e	NTLM	venerable

Color Codes: Green: Exact match, Yellow: Partial match, Red: Not found.

[Download CrackStation's Wordlist](#)

Flag: flag{venerable}

Dank Memes and Broken Dreams

30 points

Problem:

When I grow up I'm going to be the l33t 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'.

```

will@will-VirtualBox:~/CTF/forensicsII/memes$ ls
dank_meme.stash
will@will-VirtualBox:~/CTF/forensicsII/memes$ binwalk -e dank_meme.stash
[...]
DECIMAL      HEXADECIMAL      DESCRIPTION
----          -----          -----
0            0x0              JPEG image data, JFIF standard 1.01
77531        0x12EDB         JPEG image data, JFIF standard 1.01
133948       0x20B3C         Zip archive data, at least v2.0 to extract, compressed size: 37904, uncompressed size: 37946, name: fbi.jpg
171917        0x29FBD         Zip archive data, at least v2.0 to extract, compressed size: 185010, uncompressed size: 185239, name: hacker.password.jpg
357004       0x5728C         Zip archive data, at least v2.0 to extract, compressed size: 170933, uncompressed size: 172777, name: hero.jpg
528003       0x80E83         Zip archive data, at least v2.0 to extract, compressed size: 324707, uncompressed size: 326064, name: l33t.png
852776       0xD0328         Zip archive data, at least v2.0 to extract, compressed size: 42012, uncompressed size: 42892, name: neo.jpg
894853       0xDA785         Zip archive data, at least v2.0 to extract, compressed size: 102772, uncompressed size: 102949, name: source.jpg
997693       0xF393D         Zip archive data, at least v2.0 to extract, compressed size: 802907, uncompressed size: 826034, name: soylent.jpg
1801229      0x187C0D        End of Zip archive
1801251      0x187C23        JPEG image data, JFIF standard 1.01
1801823      0x187E5F        TIFF image data, big-endian, offset of first image directory: 8
1874554      0x1C9A7A        JPEG image data, JFIF standard 1.01
1874936      0x1C9B8F        Copyright string: "Copyright (c) 1998 Hewlett-Packard Company"

will@will-VirtualBox:~/CTF/forensicsII/memes$ ls
dank_meme.stash _dank_meme.stash.extracted
will@will-VirtualBox:~/CTF/forensicsII/memes$ 

```

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.

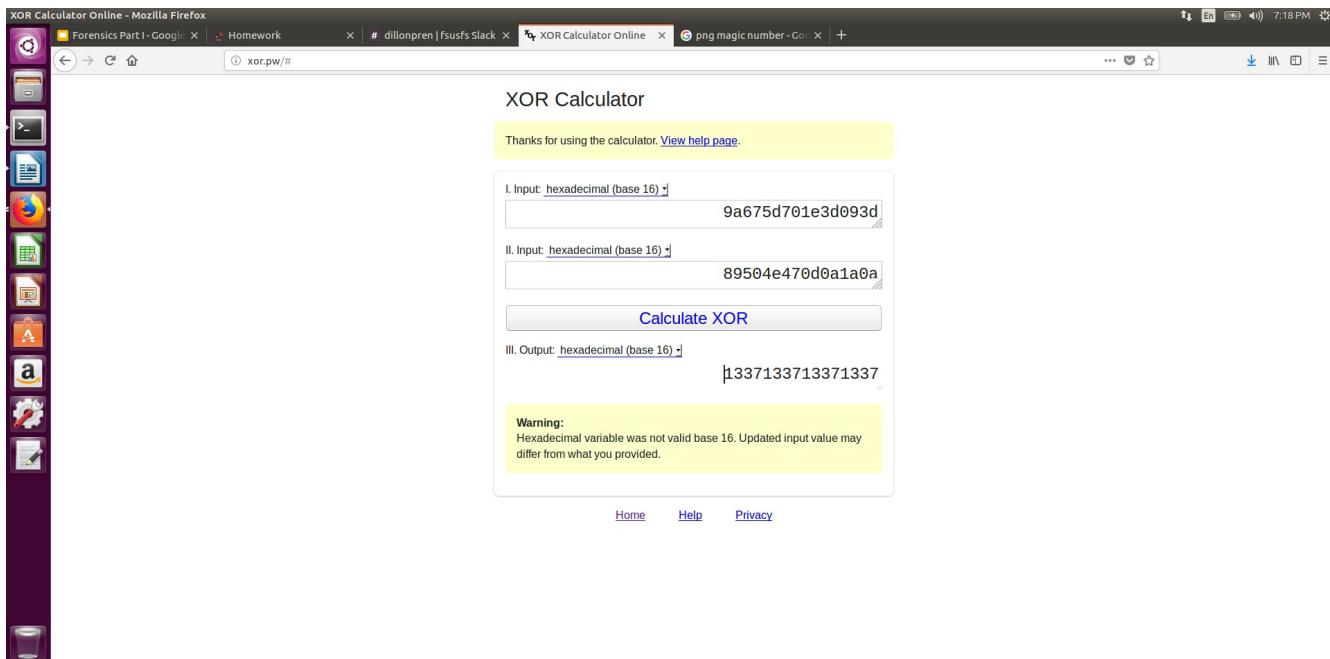
```

will@will-VirtualBox:~/CTF/forensicsII/memes/_dank_meme.stash.extracted$ ls
total 3420
drwxrwxr-x  2 will will   4096 Feb 18 19:04 .
drwxrwxr-x  3 will will   4096 Feb 18 19:04 ../
-rw-rw-r--  1 will will 1780690 Feb 18 19:04 20B3C.zip
-rw-rw-r--  1 will will    37946 Feb 11 14:54 fbi.jpg
-rw-rw-r--  1 will will   185239 Feb 11 14:54 hacker.password.jpg
-rw-rw-r--  1 will will   172777 Feb 11 14:39 hero.jpg
-rw-rw-r--  1 will will   326064 Feb 11 14:47 l33t.png
-rw-rw-r--  1 will will    42892 Feb 11 14:41 neo.jpg
-rw-rw-r--  1 will will   102949 Feb 11 14:41 source.jpg
-rw-rw-r--  1 will will   826034 Feb 11 14:52 soylent.jpg
will@will-VirtualBox:~/CTF/forensicsII/memes/_dank_meme.stash.extracted$ 

```

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 encryped, 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.

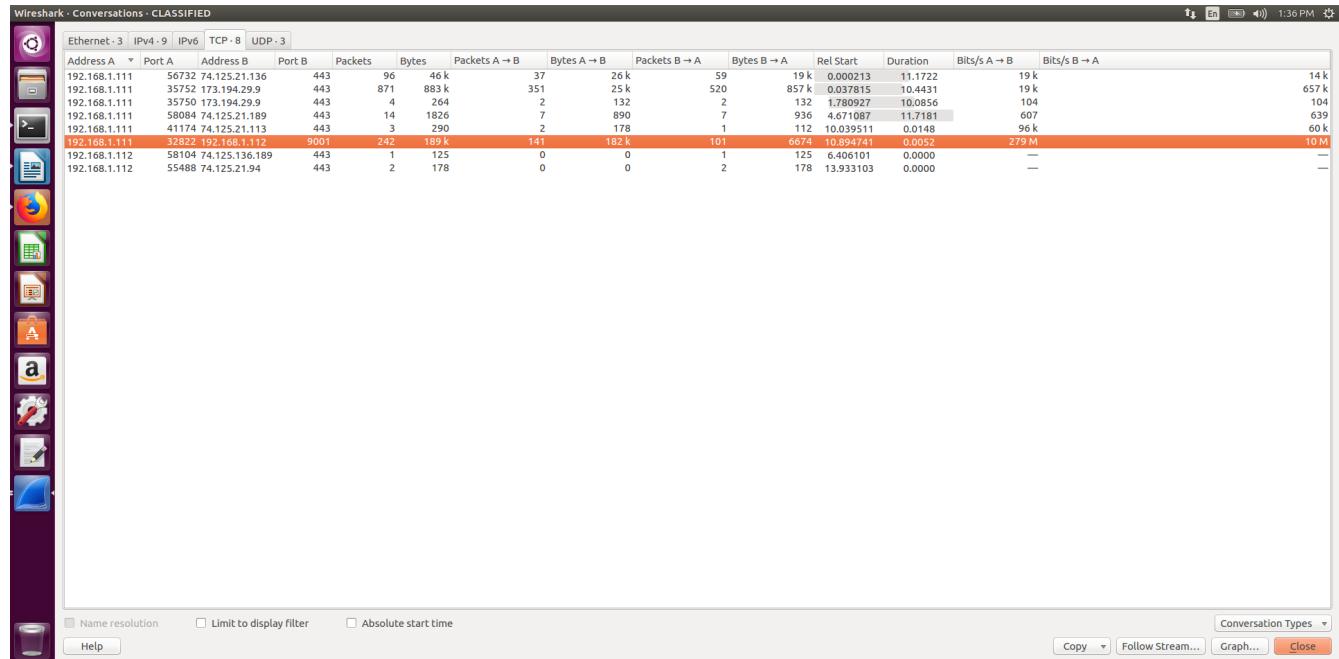
The intel can be found at <https://drive.google.com/open?id=1jGJGFfmRPSJ0JTq5UMcisI26MYnqUqq>

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

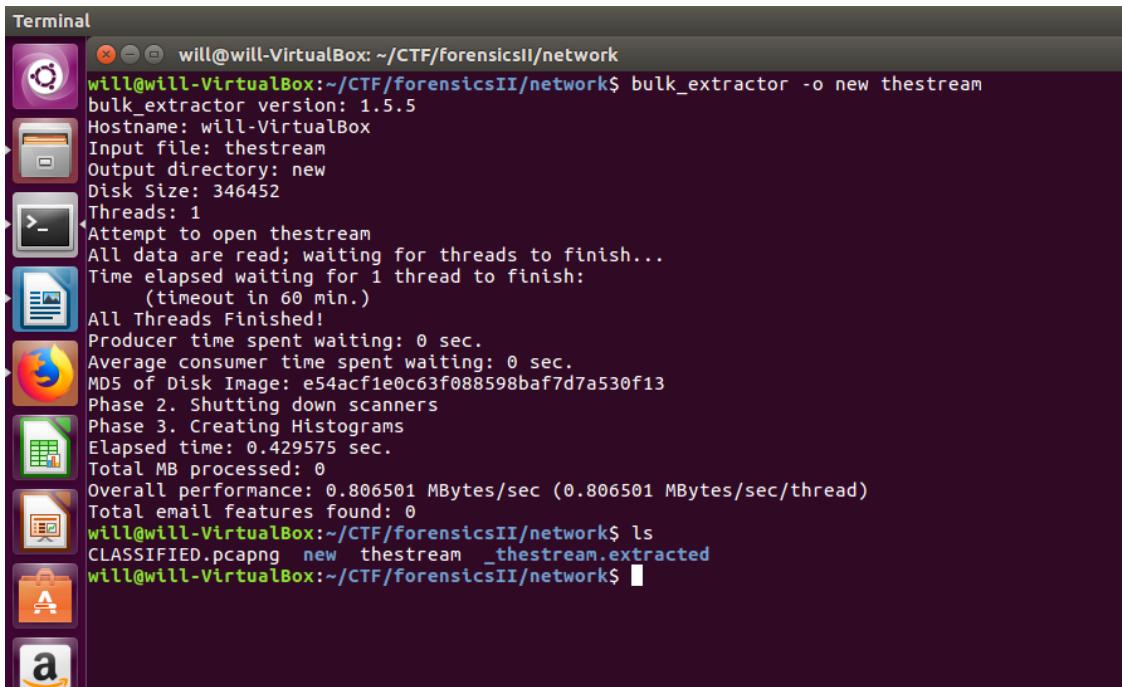
Solution:

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.

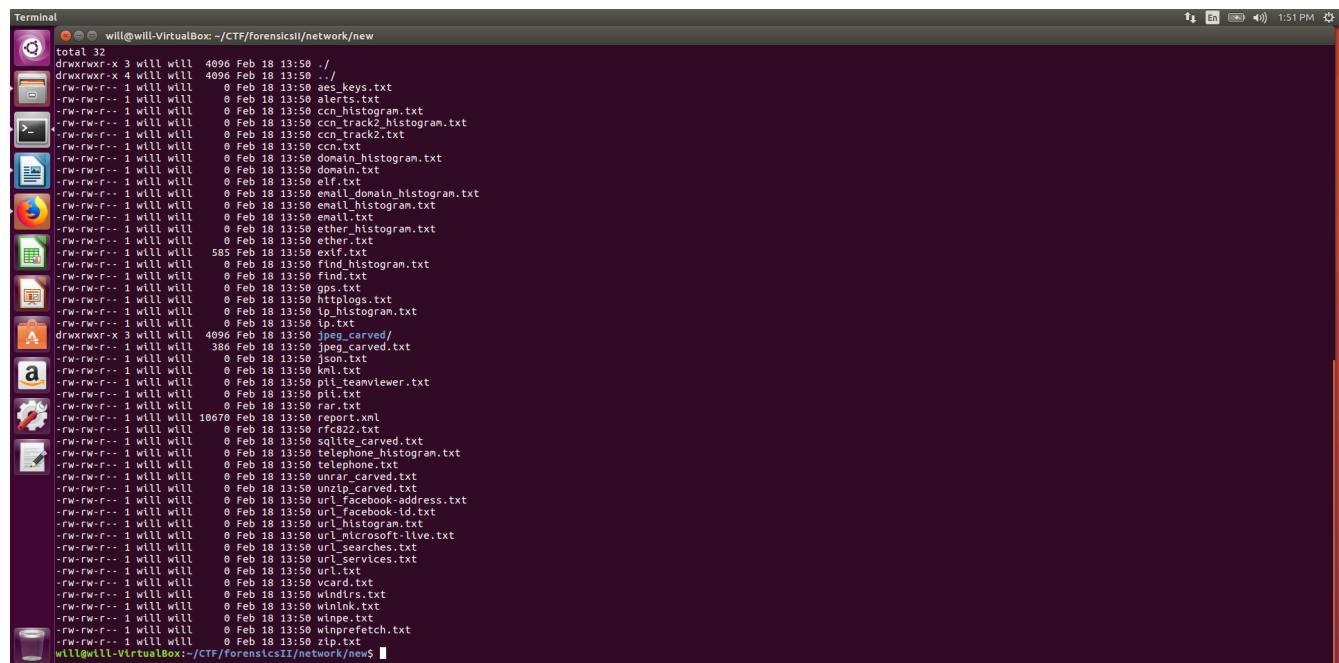


I saved the stream as ‘thestrream’. I tried carving the string using ‘binwalk -e thestream’. That extracted 12 sit files but, after a lot of playing around with them, I found nothing. I thought the names were a hex representation of the flag, but they were not. I then decided to try a tool I learned 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.



```
Terminal
will@will-VirtualBox:~/CTF/forensicsII/network$ bulk_extractor -o new thestream
bulk_extractor version: 1.5.5
Hostname: will-VirtualBox
Input file: thestream
Output directory: new
Disk Size: 346452
Threads: 1
Attempt to open thestream
All data are read; waiting for threads to finish...
Time elapsed waiting for 1 thread to finish:
    (timeout in 60 min.)
All Threads Finished!
Producer time spent waiting: 0 sec.
Average consumer time spent waiting: 0 sec.
MD5 of Disk Image: e54acf1e0c63f088598baf7d7a530f13
Phase 2. Shutting down scanners
Phase 3. Creating Histograms
Elapsed time: 0.429575 sec.
Total MB processed: 0
Overall performance: 0.806501 MBytes/sec (0.806501 MBytes/sec/thread)
Total email features found: 0
will@will-VirtualBox:~/CTF/forensicsII/network$ ls
CLASSIFIED.pcapng  new  thestream _thestream.extracted
will@will-VirtualBox:~/CTF/forensicsII/network$
```

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



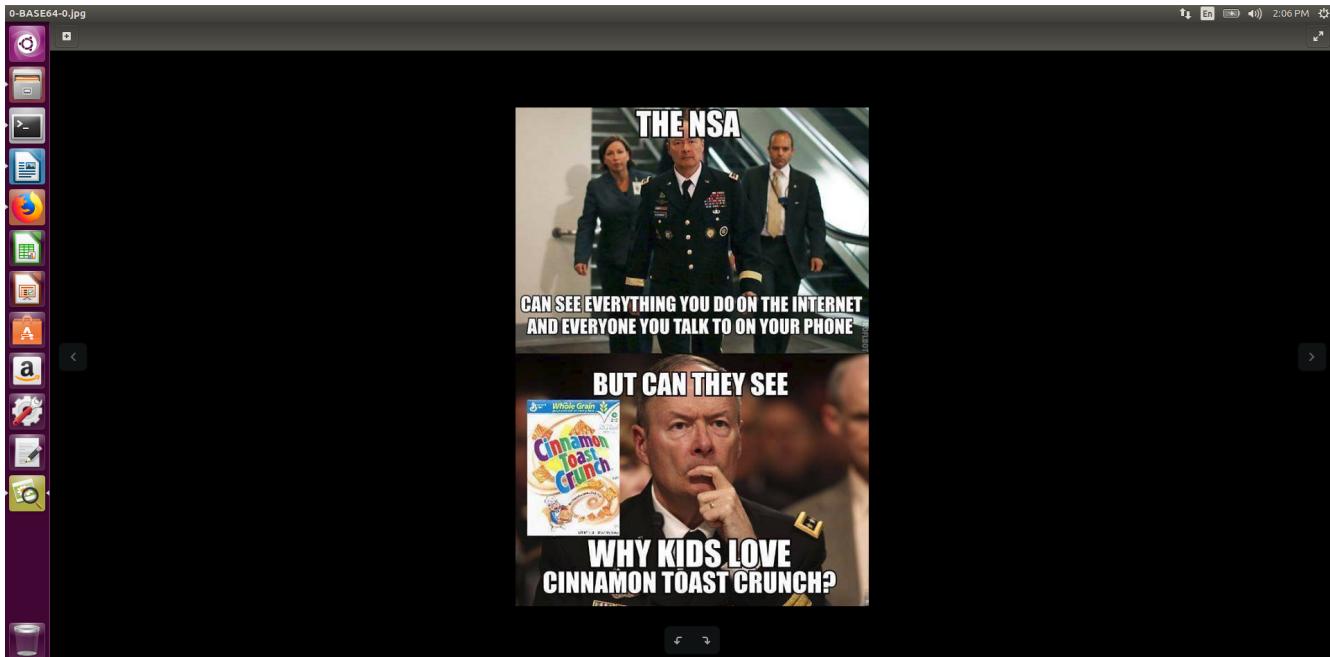
```
Terminal
will@will-VirtualBox:~/CTF/forensicsII/network/new
total 32
drwxrwxr-x 3 will will 4096 Feb 18 13:58 .
drwxrwxr-x 4 will will 4096 Feb 18 13:58 ../
-rw-rw-r-- 1 will will 0 Feb 18 13:58 aes.keys.txt
-rw-rw-r-- 1 will will 0 Feb 18 13:58 alerts.txt
-rw-rw-r-- 1 will will 0 Feb 18 13:58 ccn.histogram.txt
-rw-rw-r-- 1 will will 0 Feb 18 13:58 ccn.track2.histogram.txt
-rw-rw-r-- 1 will will 0 Feb 18 13:58 ccn.track2.txt
-rw-rw-r-- 1 will will 0 Feb 18 13:58 ccn.txt
-rw-rw-r-- 1 will will 0 Feb 18 13:58 domain.histogram.txt
-rw-rw-r-- 1 will will 0 Feb 18 13:58 domain.txt
-rw-rw-r-- 1 will will 0 Feb 18 13:58 elf.txt
-rw-rw-r-- 1 will will 0 Feb 18 13:58 email.histogram.txt
-rw-rw-r-- 1 will will 0 Feb 18 13:58 email.txt
-rw-rw-r-- 1 will will 0 Feb 18 13:58 ether.histogram.txt
-rw-rw-r-- 1 will will 0 Feb 18 13:58 ether.txt
-rw-rw-r-- 1 will will 580 Feb 18 13:58 find.txt
-rw-rw-r-- 1 will will 0 Feb 18 13:58 find.histogram.txt
-rw-rw-r-- 1 will will 0 Feb 18 13:58 find.txt
-rw-rw-r-- 1 will will 0 Feb 18 13:58 gps.txt
-rw-rw-r-- 1 will will 0 Feb 18 13:58 httplogs.txt
-rw-rw-r-- 1 will will 0 Feb 18 13:58 ip_histogram.txt
-rw-rw-r-- 1 will will 4096 Feb 18 13:58 jpeg_carved/
-rw-rw-r-- 1 will will 386 Feb 18 13:58 jpeg.carved.txt
-rw-rw-r-- 1 will will 0 Feb 18 13:58 json.txt
-rw-rw-r-- 1 will will 0 Feb 18 13:58 kml.txt
-rw-rw-r-- 1 will will 0 Feb 18 13:58 pil_teamviewer.txt
-rw-rw-r-- 1 will will 0 Feb 18 13:58 pil.txt
-rw-rw-r-- 1 will will 0 Feb 18 13:58 report.xml
-rw-rw-r-- 1 will will 10670 Feb 18 13:58 rfc822.txt
-rw-rw-r-- 1 will will 0 Feb 18 13:58 sqlite_carved.txt
-rw-rw-r-- 1 will will 0 Feb 18 13:58 telephone.histogram.txt
-rw-rw-r-- 1 will will 0 Feb 18 13:58 telephone.txt
-rw-rw-r-- 1 will will 0 Feb 18 13:58 url.histogram.txt
-rw-rw-r-- 1 will will 0 Feb 18 13:58 unzip.carved.txt
-rw-rw-r-- 1 will will 0 Feb 18 13:58 url.facebook-address.txt
-rw-rw-r-- 1 will will 0 Feb 18 13:58 url.facebook-id.txt
-rw-rw-r-- 1 will will 0 Feb 18 13:58 url.histogram.txt
-rw-rw-r-- 1 will will 0 Feb 18 13:58 url.microsoft-live.txt
-rw-rw-r-- 1 will will 0 Feb 18 13:58 url.searches.txt
-rw-rw-r-- 1 will will 0 Feb 18 13:58 url.services.txt
-rw-rw-r-- 1 will will 0 Feb 18 13:58 url.txt
-rw-rw-r-- 1 will will 0 Feb 18 13:58 vcard.txt
-rw-rw-r-- 1 will will 0 Feb 18 13:58 windirs.txt
-rw-rw-r-- 1 will will 0 Feb 18 13:58 winlnk.txt
-rw-rw-r-- 1 will will 0 Feb 18 13:58 wmppe.txt
-rw-rw-r-- 1 will will 0 Feb 18 13:58 wmppe.txt
-rw-rw-r-- 1 will will 0 Feb 18 13:58 zip.txt
will@will-VirtualBox:~/CTF/forensicsII/network/new$
```

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_histgram.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
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/000$ ls
0-BASE64-o.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.

```

Terminal will@will-VirtualBox: ~/CTF/forensicsII/network/new/jpeg_carved/000
No command 'exif' found, did you mean:
  Command 'exif' from package 'exif' (universe)
exif: command not found
will@will-VirtualBox:~/CTF/forensicsII/network/new/jpeg_carved/000$ exiftool *.jpg
File: ./0-BASE64-0.jpg
  File Version Number : 10.79
  File Name : 0-BASE64-0.jpg
  Directory : 
  File Size : 63 kB
  File Modification Date/Time : 2018:02:18 13:50:40-05:00
  File Access Date/Time : 2018:02:18 13:50:40-05:00
  File Change Date/Time : 2018:02:18 13:50:40-05:00
  File Permissions : rw-rw-r-
  File Type : JPEG
  File Type Extension : jpg
  MIME Type : image/jpeg
  JFIF Version : 1.01
  Exif Byte Order : BIG-endian (Motorola, MM)
  X Resolution : 72
  Y Resolution : 72
  Resolution Unit : inches
  Artist : 
  YCbCr Positioning : Centered
  Color Space : 
  Profile CMM Type : Unknown (lcms)
  Profile Version : 2.1.0
  Profile Class : Display Device Profile
  Color Space Data : RGB
  Profile Connection Space : XYZ
  Profile Date/Time : 2018:01:25 03:41:57
  Profile File Signature : sRGB
  Primary Platform : Apple Computer Inc.
  CMY Flags : Not Embedded, Independent
  Device Manufacturer : 
  Device Model : 
  Device Attributes : Reflective, Glossy, Positive, Color
  Viewing Intent : Perceptual
  Connection Space Illuminant : 0.9642 1 0.82491
  Profile Creator : Unknown (lcms)
  Profile ID : 0
  Profile Description : c2
  Profile Copyright : FB
  Red Matrix Column : 0.0642 1 0.02491
  Media Black Point : 0.01205 0.0125 0.01031
  Red Matrix Column : 0.43607 0.22249 0.01392
  Green Matrix Column : 0.38515 0.71687 0.09708
  Blue Matrix Column : 0.14307 0.00061 0.7141
  Red Tone Reproduction Curve : (Binary data 64 bytes, use -b option to extract)
  Green Tone Reproduction Curve : (Binary data 64 bytes, use -b option to extract)
  Blue Tone Reproduction Curve : (Binary data 64 bytes, use -b option to extract)
  Image Width : 510
  Image Height : 720
  Encoding Process : Baseline DCT, Huffman coding

```

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}}

Hex to ASCII text converter

Hex to ASCII text converter.

Enter 2 digits hex numbers with any prefix / postfix / delimiter and press the Convert button

(e.g. FF 43 5A 7F):

0x66,0x6c,0x61,0x67,0x7b,0x37,0x34,0x35,0x37,0x33,0x5f,0x59,0x30,0x75,0x5f,0x63,0x34,0x4e,0x5f,0x35,0x33,0x7d

flag{74573_Y0u_c4N_533}

rring data from cdnjs.cloudflare.com...