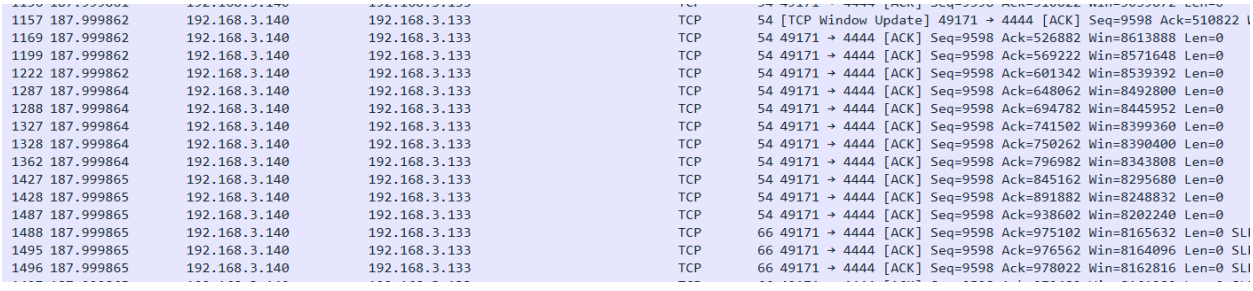
WolvCTF 2024

Eternally Pwned: Infiltration [forensics]

> I recently had my passwords and other sensitive data leaked, but I have no idea how. Can you figure out how the attacker got in to my PC?

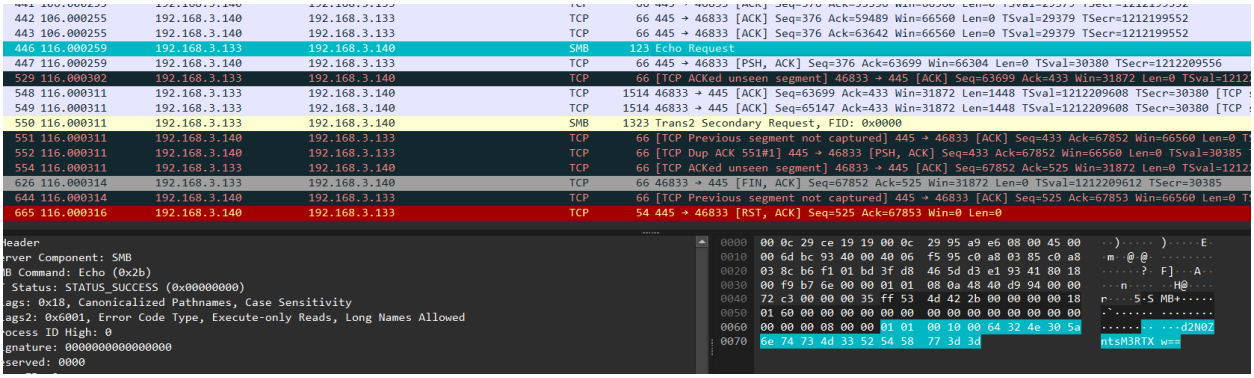
Flag: wctf{l3tS_3teRn4lLy_g0_bLU3_7n9wm4iWnL}

The challenge provides a pcap file that contains the network connection for the victim. From the file, we can easily identify that the victim's IP is 192.168.3.140. While scrolling down, one section of the connection interests me.

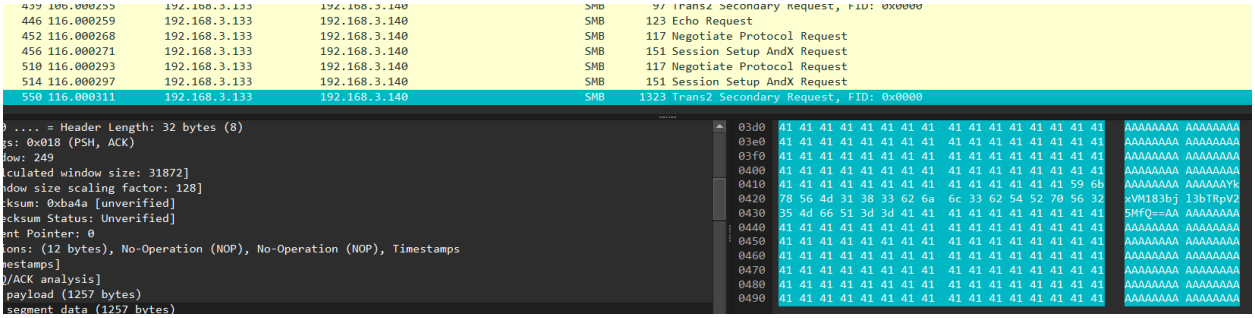


1157	187.999862	192.168.3.140	192.168.3.133	TCP	54 [TCP Window Update] 49171 → 4444 [ACK] Seq=9598 Ack=510822
1169	187.999862	192.168.3.140	192.168.3.133	TCP	54 49171 → 4444 [ACK] Seq=9598 Ack=526882 Win=8613888 Len=0
1199	187.999862	192.168.3.140	192.168.3.133	TCP	54 49171 → 4444 [ACK] Seq=9598 Ack=569222 Win=8571648 Len=0
1222	187.999862	192.168.3.140	192.168.3.133	TCP	54 49171 → 4444 [ACK] Seq=9598 Ack=601342 Win=8539392 Len=0
1287	187.999864	192.168.3.140	192.168.3.133	TCP	54 49171 → 4444 [ACK] Seq=9598 Ack=648062 Win=8492800 Len=0
1288	187.999864	192.168.3.140	192.168.3.133	TCP	54 49171 → 4444 [ACK] Seq=9598 Ack=694782 Win=8445952 Len=0
1327	187.999864	192.168.3.140	192.168.3.133	TCP	54 49171 → 4444 [ACK] Seq=9598 Ack=741502 Win=8399360 Len=0
1328	187.999864	192.168.3.140	192.168.3.133	TCP	54 49171 → 4444 [ACK] Seq=9598 Ack=750262 Win=8390400 Len=0
1362	187.999864	192.168.3.140	192.168.3.133	TCP	54 49171 → 4444 [ACK] Seq=9598 Ack=796982 Win=8343808 Len=0
1427	187.999865	192.168.3.140	192.168.3.133	TCP	54 49171 → 4444 [ACK] Seq=9598 Ack=845162 Win=8295680 Len=0
1428	187.999865	192.168.3.140	192.168.3.133	TCP	54 49171 → 4444 [ACK] Seq=9598 Ack=891882 Win=8248832 Len=0
1487	187.999865	192.168.3.140	192.168.3.133	TCP	54 49171 → 4444 [ACK] Seq=9598 Ack=938602 Win=8202240 Len=0
1488	187.999865	192.168.3.140	192.168.3.133	TCP	66 49171 → 4444 [ACK] Seq=9598 Ack=975102 Win=8165632 Len=0 SL
1495	187.999865	192.168.3.140	192.168.3.133	TCP	66 49171 → 4444 [ACK] Seq=9598 Ack=976562 Win=8164096 Len=0 SL
1496	187.999865	192.168.3.140	192.168.3.133	TCP	66 49171 → 4444 [ACK] Seq=9598 Ack=978022 Win=8162816 Len=0 SL

By looking at the uncommon port (or commonly used as reverse shell port), this tells us that the attacker has IP of 192.168.3.133. This challenge is about infiltration, how the sensitive data is leaked to the attacker. So, I applied the filter `ip.src == 192.168.3.140 && ip.dst == 192.168.3.133` to look for any infiltration tracks. While following one of the huge chunks of TCP, I found a smb echo request with a base64 data `d2N0ZntsM3RTXw==` which then decoded as the first part of flag: `wctf{l3tS_` .



Majority of the protocols are either TCP or SMB related, so I decided to dive back to SMB stuff first by applying filter `smb` . After the echo request, I saw another huge chunk of Trans2 Secondary Request Data that has two different base64 encoded data hidden within it.



Encoded data: `M3RlUm40bEx5X2cwXw==` `YkxVM183bjl3bTRpV25MfQ==`

Decoded data: `3teRn4lLy_g0_` `bLU3_7n9wm4iWnL}`

Eternally Pwned: Persistence [forensics]

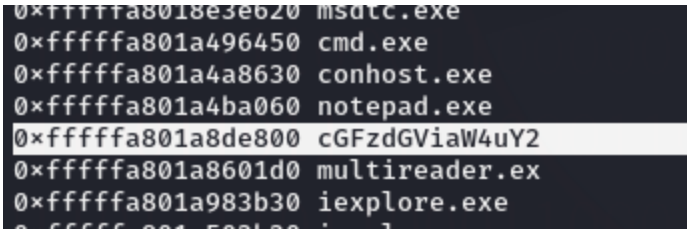
> I get that the attackers were in my PC, but how did they achieve persistence?

Flag: wctf{v0lAt1l3_m3m0ry_4qu1r3D_a3fe9fn3al}

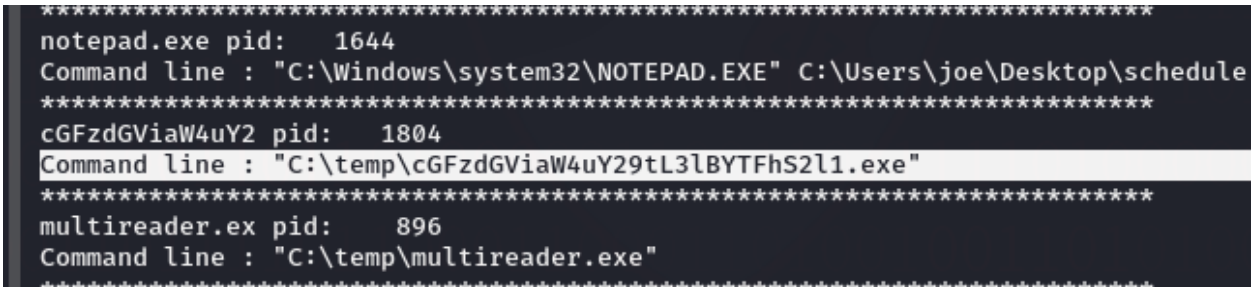
Before we start doing memory analysis, we need to look for the profile used. Using vol3 (rather than waiting for vol2 to finish imageinfo), we can get the following info which deduced that the profile is `Win7SP1x64_23418` .

```
Is64Bit True
IsPAE   False
layer_name      0 WindowsIntel32e
memory_layer    1 WindowsCrashDump64Layer
base_layer      2 FileLayer
KdDebuggerDataBlock     0xf80001a430a0
NTBuildLab      7601.17514.amd64fre.win7sp1_rtm.
CSDVersion      1
KdVersionBlock  0xf80001a43068
Major/Minor     15.7601
MachineType     34404
KeNumberProcessors      1
SystemTime      2024-03-09 12:05:40
NtSystemRoot    C:\Windows
NtProductType   NtProductServer
NtMajorVersion  6
NtMinorVersion  1
PE MajorOperatingSystemVersion  6
PE MinorOperatingSystemVersion  1
PE Machine      34404
PE TimeDateStamp        Sat Nov 20 09:30:02 2010
```

Talking about persistence, we need to check the processes running to identify if there's any suspicious ones.



From here, we can see this random name process weird. Running `cmdline` , we can look at the full filename.



Noticing the filename is actually encoded in base64 [BIG THANKS to warlocksmurf], we can decode it and eventually it will give us a pastebin link. The flag lies in the pastebin.

`cGFzdGViaW4uY29tL3lBYTFhS2l1` → pastebin.com/yAa1aKiu

WOLPHV I: Reconnaissance [osint]

> A new ransomware group you may have heard about has emerged: **WOLPHV**
>
> There's already been reports of their presence in articles and posts.

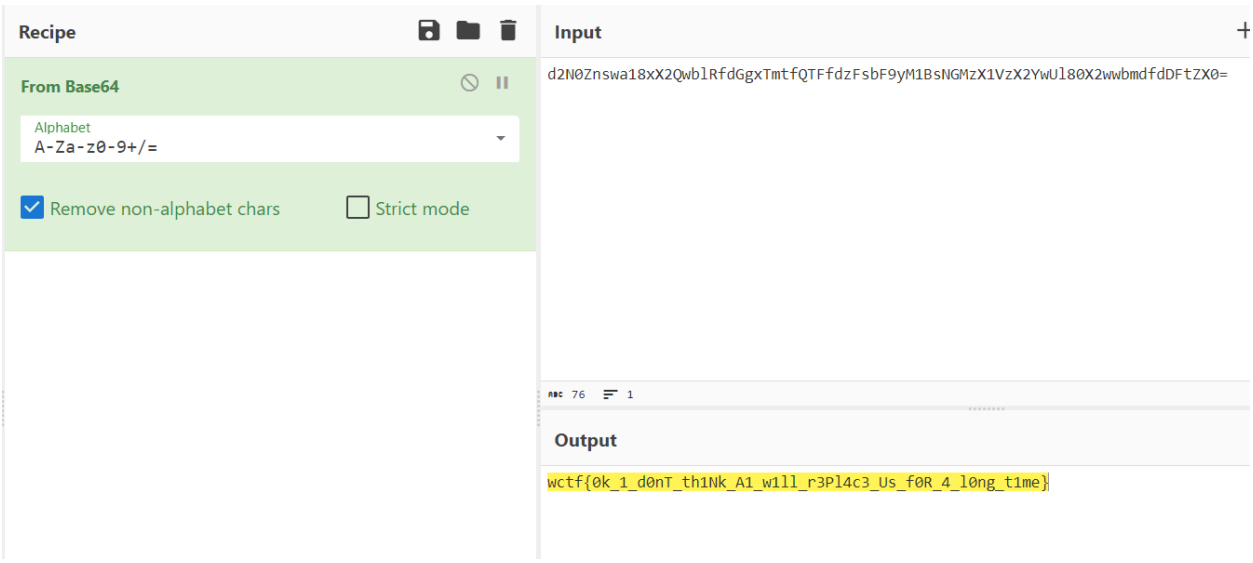
> NOTE: Wolphv's twitter/X account and https://wolphv.chal.wolvsec.org/ are **out of scope** for all these challenges. Any flags found from these are not a part of these challenges
>
> This is a start to a 5 part series of challenges. Solving this challenge will unlock **WOLPHV II: Infiltrate**

Flag: wctf{0k_1_d0nT_th1Nk_A1_w1ll_r3Pl4c3_Us_f0R_4_l0ng_t1me}

Talking about a new ransomware group called `WOLPHV`. With some googling, there are only a few relevant results:

- https://thecyberexpress.com/new-wolphv-ransomware-group-on-the-dark-web/
- https://www.linkedin.com/posts/coy-peterman-35474_new-wolphv-ransomware-group-on-the-dark-web-activity-7113152189342081025-GtTT
- https://twitter.com/FalconFeedsio/status/1706989111414849989

When looking at the tweet, there was one comment that was different. After base64 decode, we can retrieve the flag.

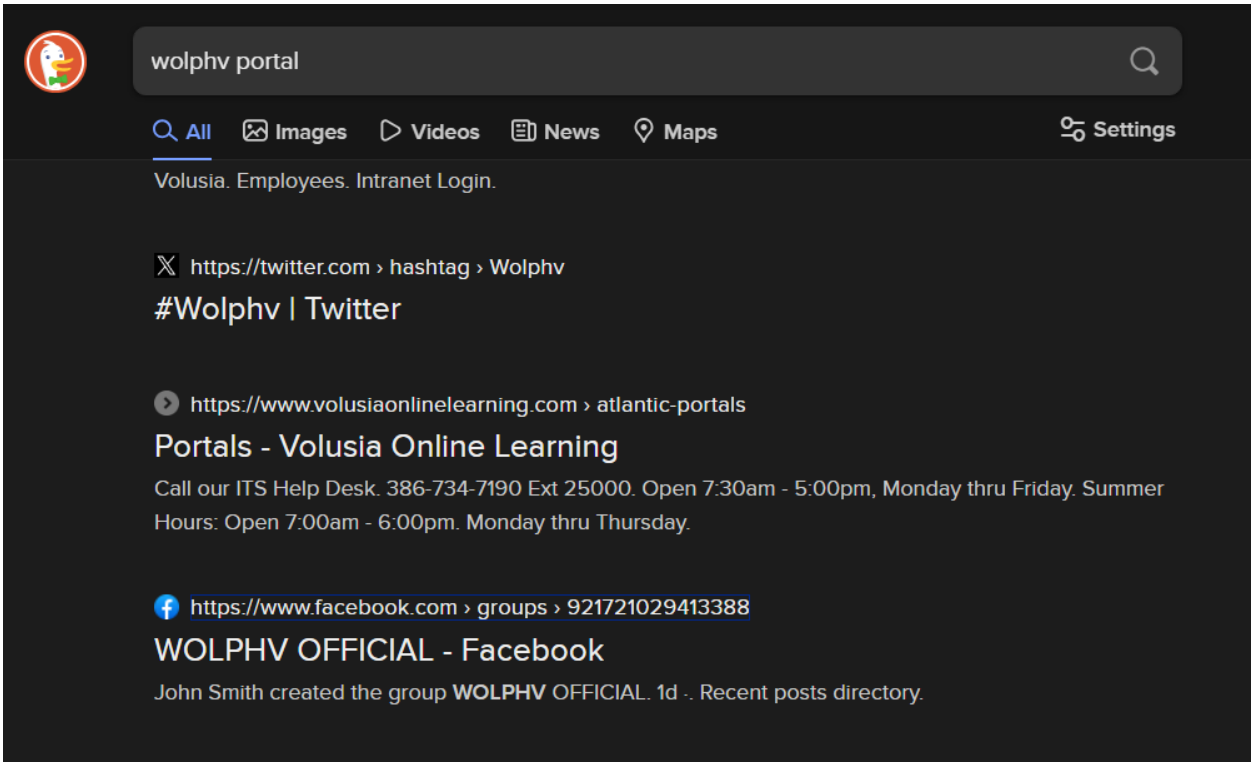


WOLPHV II: Infiltrate [osint]

> Since the WOLPHV twitter/x is out of comission now, I wonder where else the official WOLPHV group posts on social media. Maybe we can also infiltrate what they use to message each other
>
> NOTE: Wolphv's twitter/X account and https://wolphv.chal.wolvsec.org/ are **out of scope** for all these challenges. Any flags found from these are not a part of these challenges
>
> Solving this challege will unlock **WOLPHV III**, **WOLPHV IV**, and **WOLPHV V**

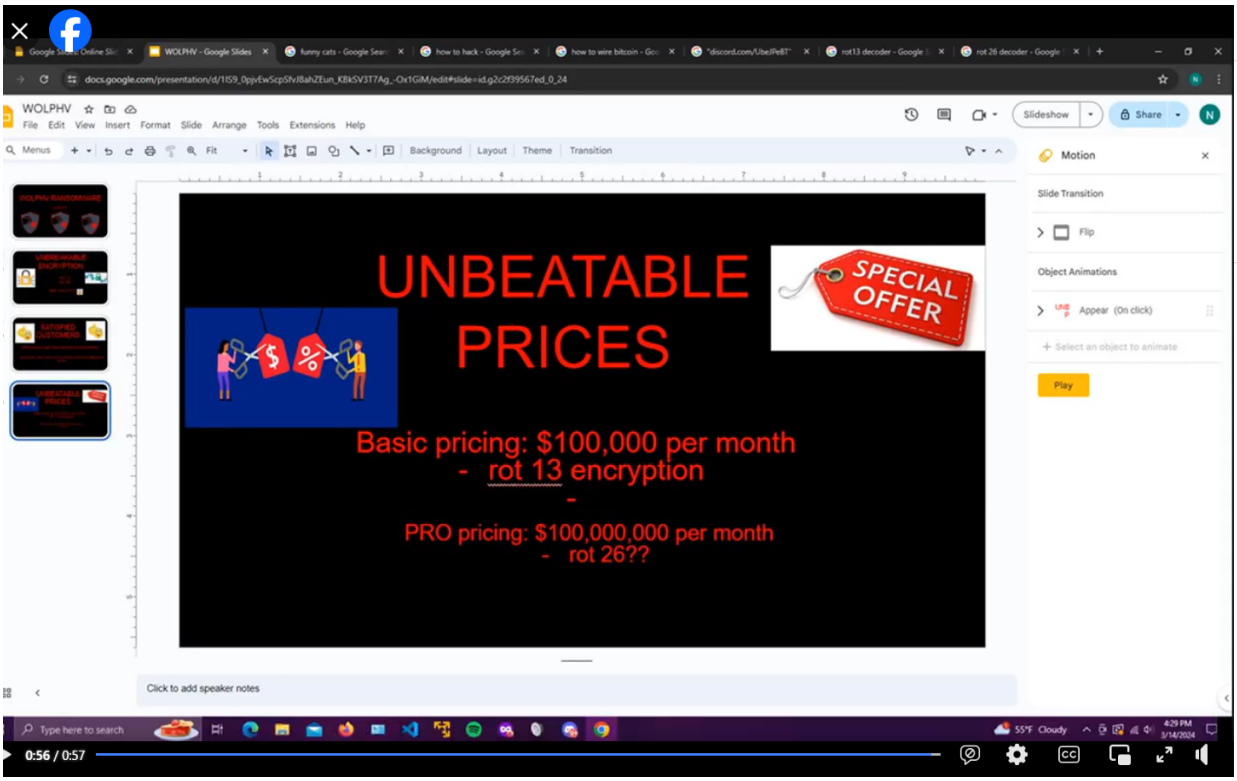
Flag: wctf{r0t_52_w0ulD_b3_cr4zy_fRfr}

From the scenario, we knew that the wolphv twitter/x was out of comission and they were using other social media. From the previous post by cyberexpress, they mentioned something about WOLPHV using a dark web portal. So, I used duckduckgo as my search engine and looked for any potential portal used.

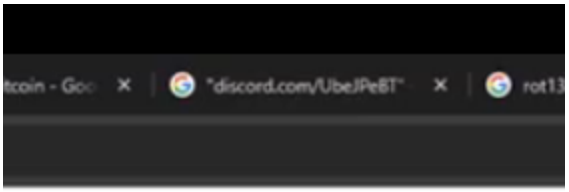


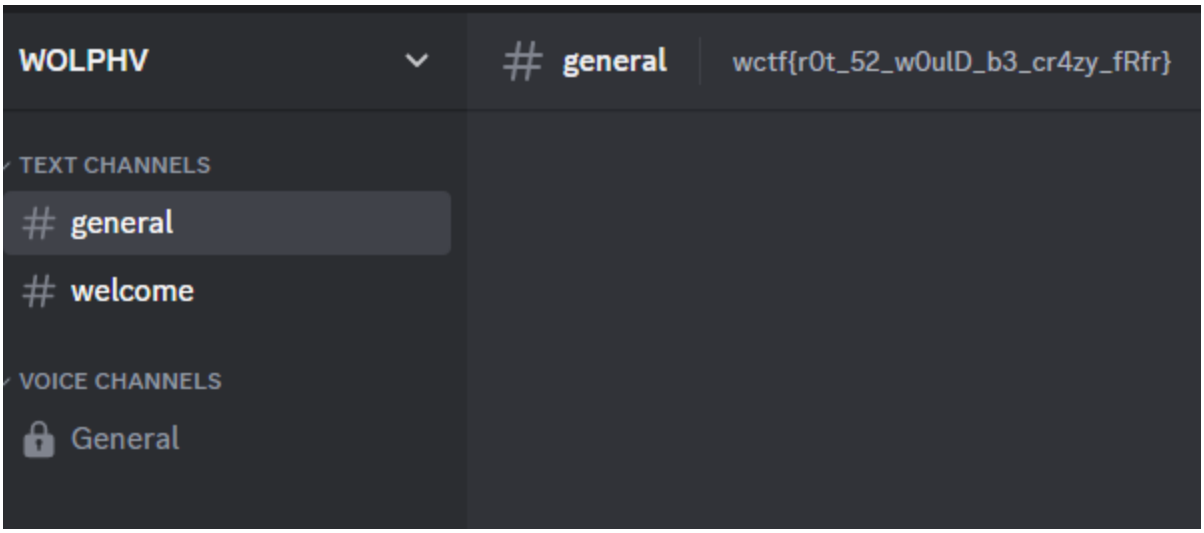
When searching, I found this facebook public group which could be a potential place they used to message each other - https://www.facebook.com/groups/921721029413388/

Looking at the group, there was nothing interesting except for one video.



From the first sight, I saw nothing. However, after taking a close look, I realised that one of the tabs was a discord invite link: `discord.com/UbeJPeBT` .





From the discord channel, we can see the flag lying around.

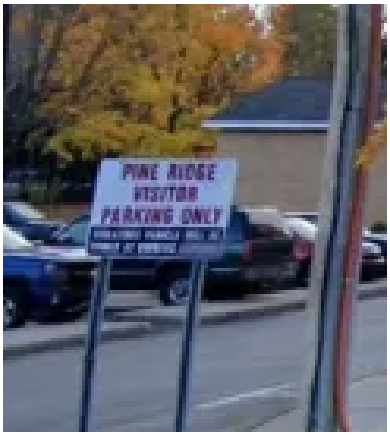
WOLPHV III: p1nesh4dow48 [osint]

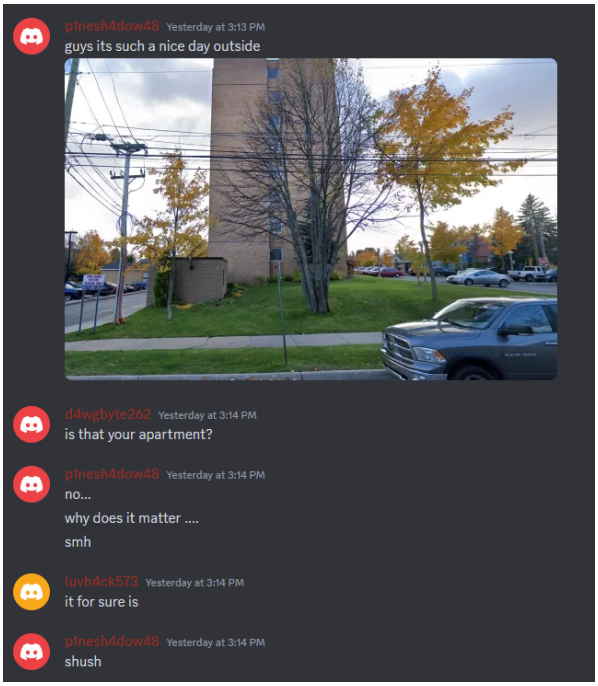
> Locate p1nesh4dow48's home coordinates and submit them as `wctf{x.xxx,x.xxx}` rouded
>
> For example, if the location was the BBB at the Unviersity of Michigan the flag would be `wctf{42.293,-83.717}`
>
> You have 10 tries to get the location. If you are *sure* you have location but the flag does not work, make a ticket
>
> NOTE: Wolphv's twitter/X account and https://wolphv.chal.wolvsec.org/ are out of scope for all these challenges. Any flags found from these are not a part of these challenges

Flag: wctf{46.545,-87.388}

At first, I tried to use some of the information i knew. From the chat, p1nesh4dow48 was the admin of WOLPHV, meaning he is a uofm (University of Michigan) student. Considering this, I looked for places near the university by checking places nearby pizza subway and mcdonalds which I then realized I'm on the wrong path (cause that's was someone's place, not what we're looking for).

=== [RESET MY BRAIN] ===

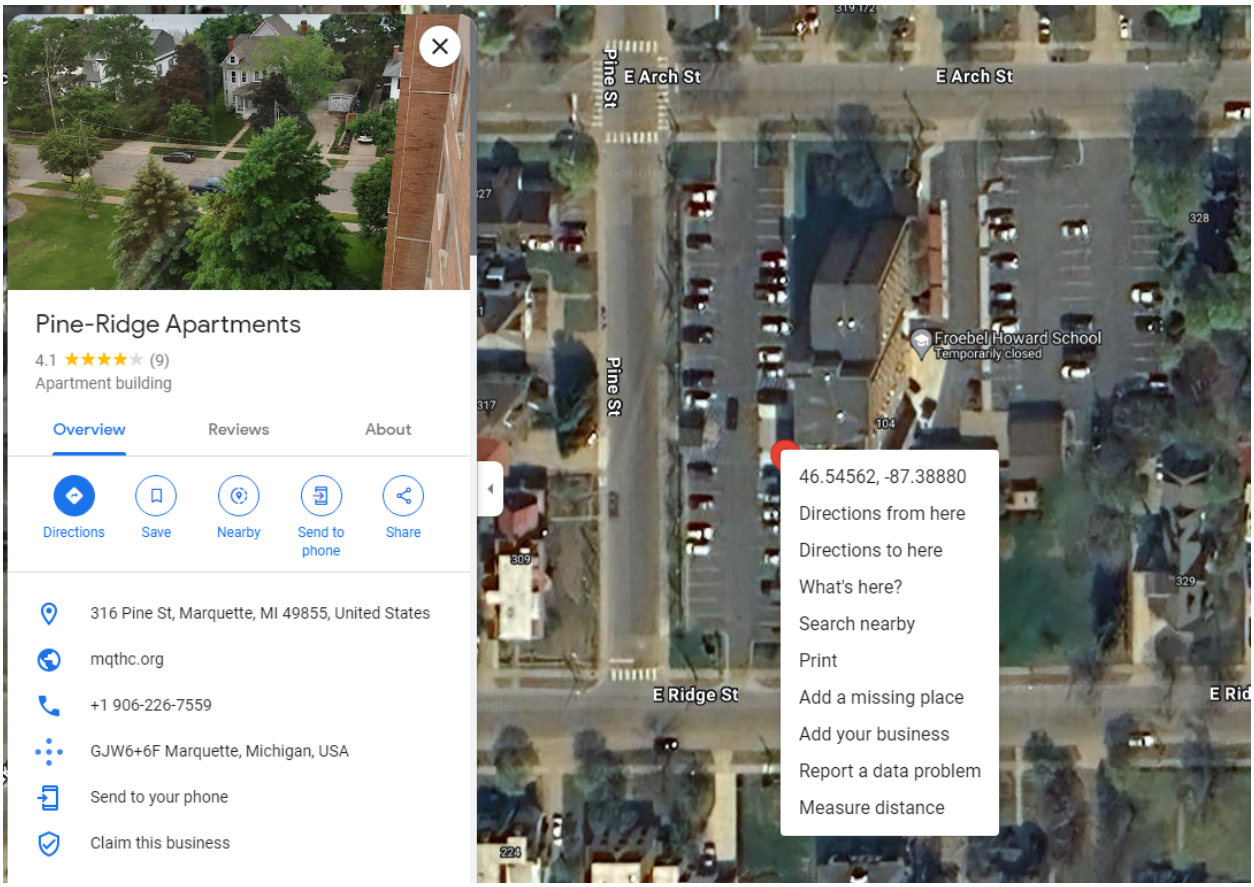




The picture above was sent by p1nesh4dow48. From the bottom left corner, we can see its related to 'Pine Ridge'. Also, someone in the chat mentioned that it could be an apartment. With the information we have, we can look for 'Pine Ridge Apartment' on Google Map.

After a LONGGG search, I finally found the exact place (which I then realized I could have put 'Michigan' when I was searching to speed up the process 🙄). We

can easily identify from the carparks on its both sides which matched the picture.



Limited 1 [crypto]

> It's pretty easy to find random integers if you know the seed, but what if every second has a different seed?

Flag: wctf{f34R_0f_m1ss1ng_0ut}

The challenge provided the source code.

```
import time
import random
import sys

if __name__ == '__main__':
    flag = input("Flag? > ").encode('utf-8')
    correct = [189, 24, 103, 164, 36, 233, 227, 172, 244, 21
3, 61, 62, 84, 124, 242, 100, 22, 94, 108, 230, 24, 190, 23,
228, 24]
    time_cycle = int(time.time()) % 256
    if len(flag) != len(correct):
        print('Nope :(')
        sys.exit(1)
    for i in range(len(flag)):
        random.seed(i+time_cycle)
        if correct[i] != flag[i] ^ random.getrandbits(8):
            print('Nope :(')
            sys.exit(1)
    print(flag)
```

From the code, we know that for every time updated, the value of random.getrandbits will be changed and thus affects the xor key for the subsequent characters. Knowing that the flag format is `wctf{…}` , we can try to find the starting time cycle and the ending time cycle to know the time used and frequent of seed change when running the program.

Using the code below, we found the potential starting time_cycle value.

```
for i in range(256):
    random.seed(i)
    if 189 ^ random.getrandbits(8) == ord('w'):
        print(i)
# 86
# 188
```

```
>>> random.seed(1+86)
>>> chr(24 ^ random.getrandbits(8))
'='

>>> random.seed(1+188)
>>> chr(24 ^ random.getrandbits(8))
'c'
```

From the code above, we proofed that the time_cycle starts from 188 because the second character of the flag matched the output, which is `c` .

Now, we need to get the time_cycle value when XORing the last character of the flag with the following code.

```
for i in range(256):
        random.seed(i)
                if 24 ^ random.getrandbits(8) == ord('}'):
                        print(i)
# 88
# 204
# 212
```

When counting the length of flag, surprisingly its 212 - 188 + 1 = 25. Assuming that each character uses `time_cycle + 1` in each loop, I wrote the following code and retrieved the flag:

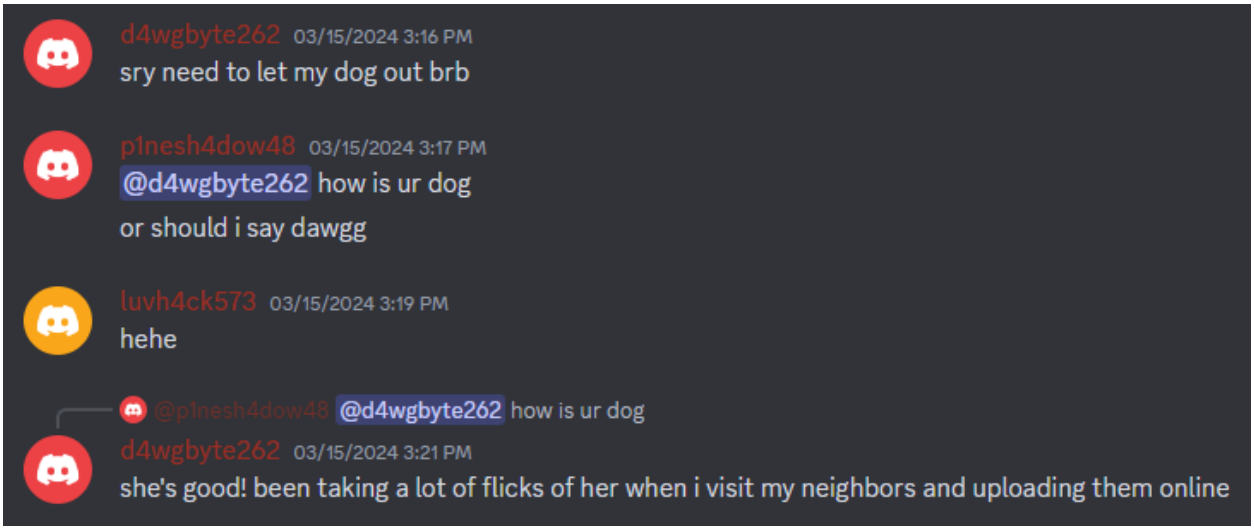
```
flag = ''
start = 188
for i in range(len(correct)):
    random.seed(start + i)
    flag += chr(correct[i] ^ random.getrandbits(8))

print(flag)
```

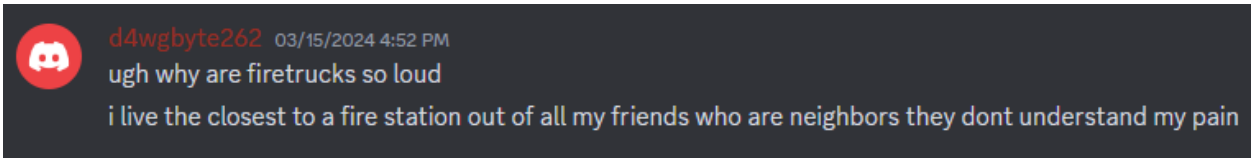
WOLPHV IV: d4wgbyte262 [osint]

> Locate d4wgbyte262's home coordinates and submit them as `wctf{x.xxxx,x.xxxx}` rouded
>
> For example, if the location was the table next to front entrace to the BBB at the Unviersity of Michigan the flag would be `wctf{42.2926,-83.7162}`
>
> You will be able to get this specific
>
> You have 10 tries to get the location. If you are *sure* you have location but the flag does not work, make a ticket

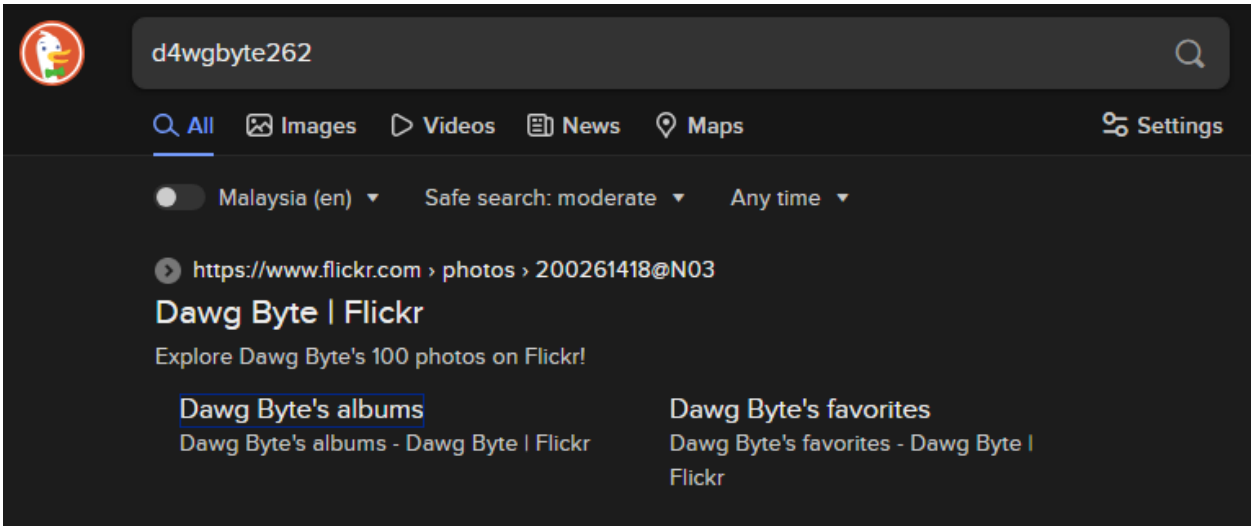
wctf{51.0912,-113.9562}

To look for d4wgbyte262's home coordinates, there were some hints from the discord chat.

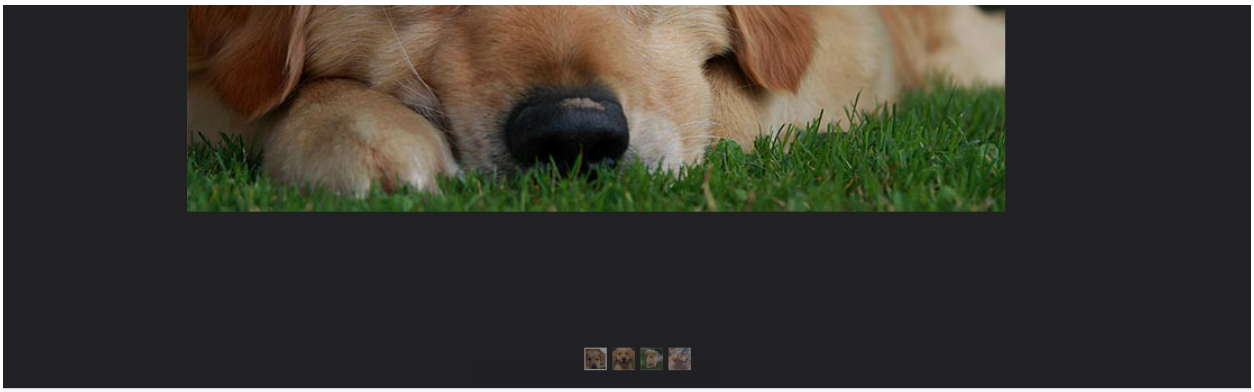




Knowing that d4wgbyte262 might have an account posting his dog pictures and his house is the nearest to the fire station compared to his neighbours. Using duckduckgo search engine (because google was not showing anything), I found his account on Flickr. (actually taking flicks is a hint)



In his album, he mentioned "my dog at my neighbors' houses, well one of the pics was at my house but you get it" which means one of the coordinates of these photos is his house. Flickr contains exif data - location for all pictures - from what I see. By pressing the map on the EXIF data section, it brought us to a bird view of all locations that all those photos in his album was taken.



Dawg Byte + Follow

dog19

109 views 0 faves 0 comments

Uploaded on March 15, 2024

© All rights reserved

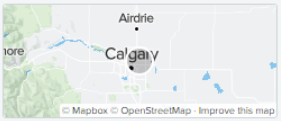
Show your appreciation with the gift of Flickr Pro

Login to comment

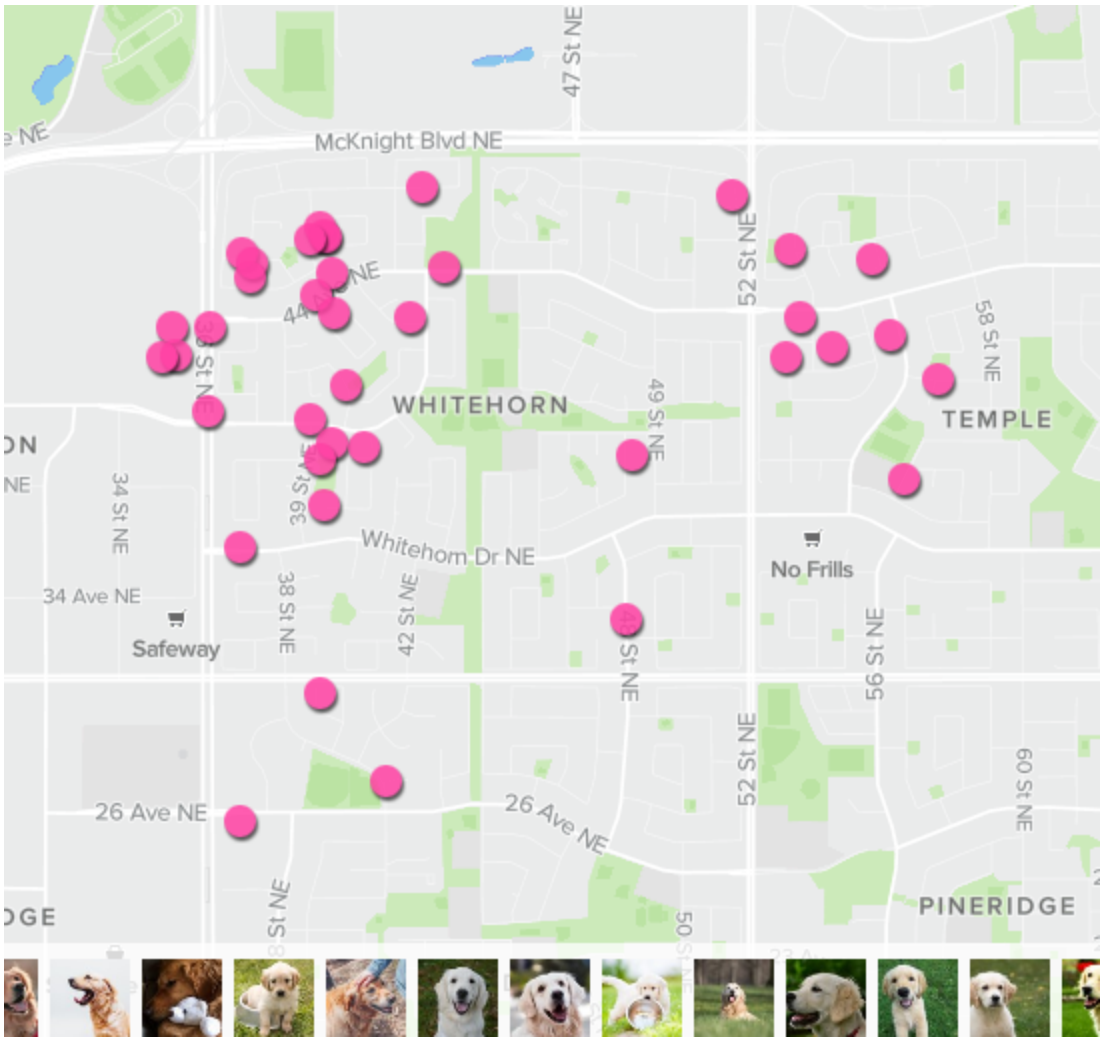
Add comment

Hide EXIF

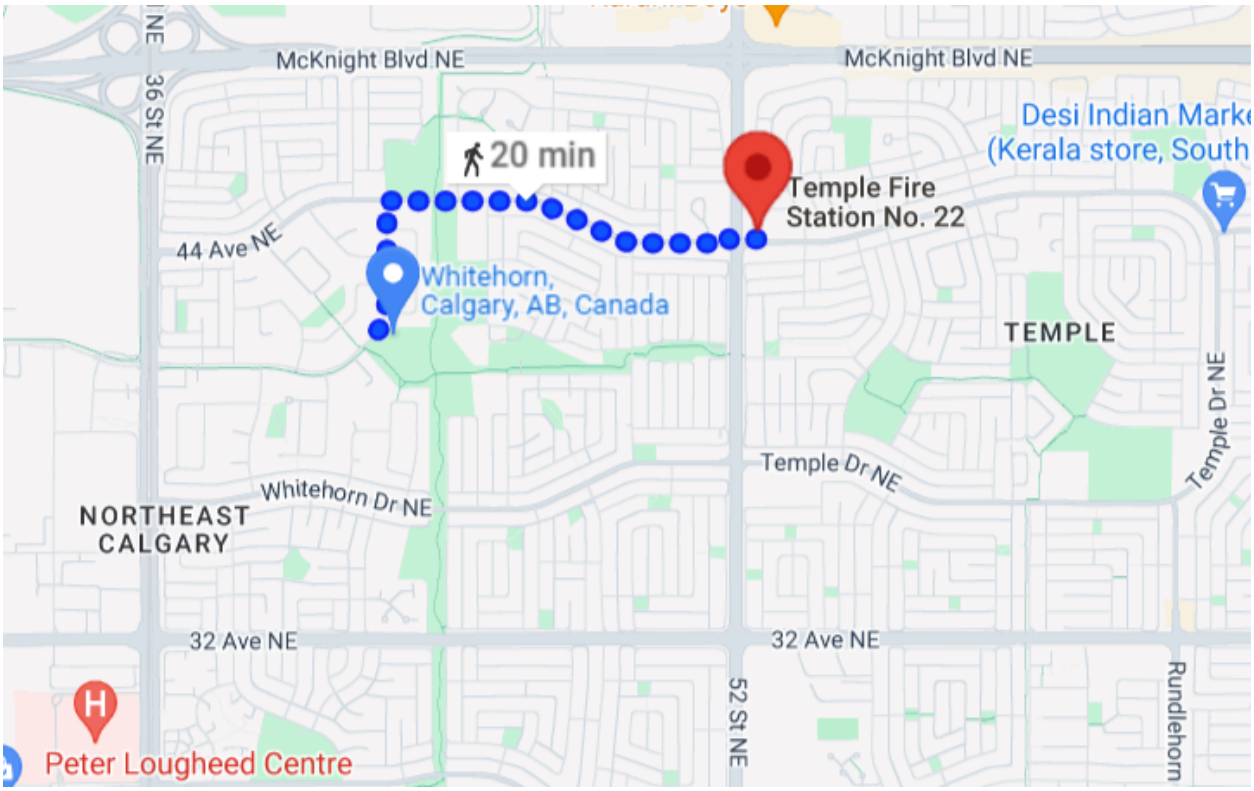
X-Resolution - 72 dpi
Y-Resolution - 72 dpi
YCbCr Positioning - Centered
GPS Version ID - 2.3.0.0
GPS Latitude Ref - North
GPS Latitude - 51 deg 5' 28.22"
GPS Longitude Ref - West
GPS Longitude - 113 deg 58' 22.60"
GPS Altitude Ref - Above Sea Level
GPS Altitude - 1044.854 m



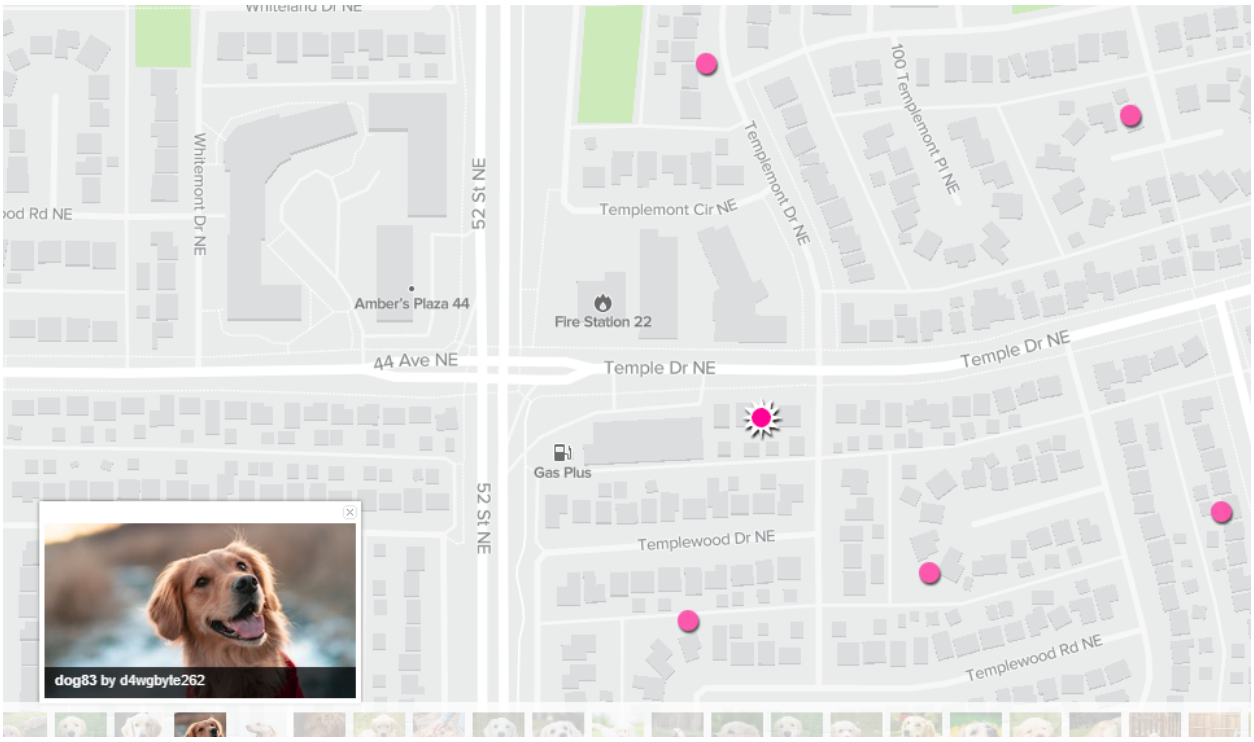
Whitehorn, Alberta, Canada

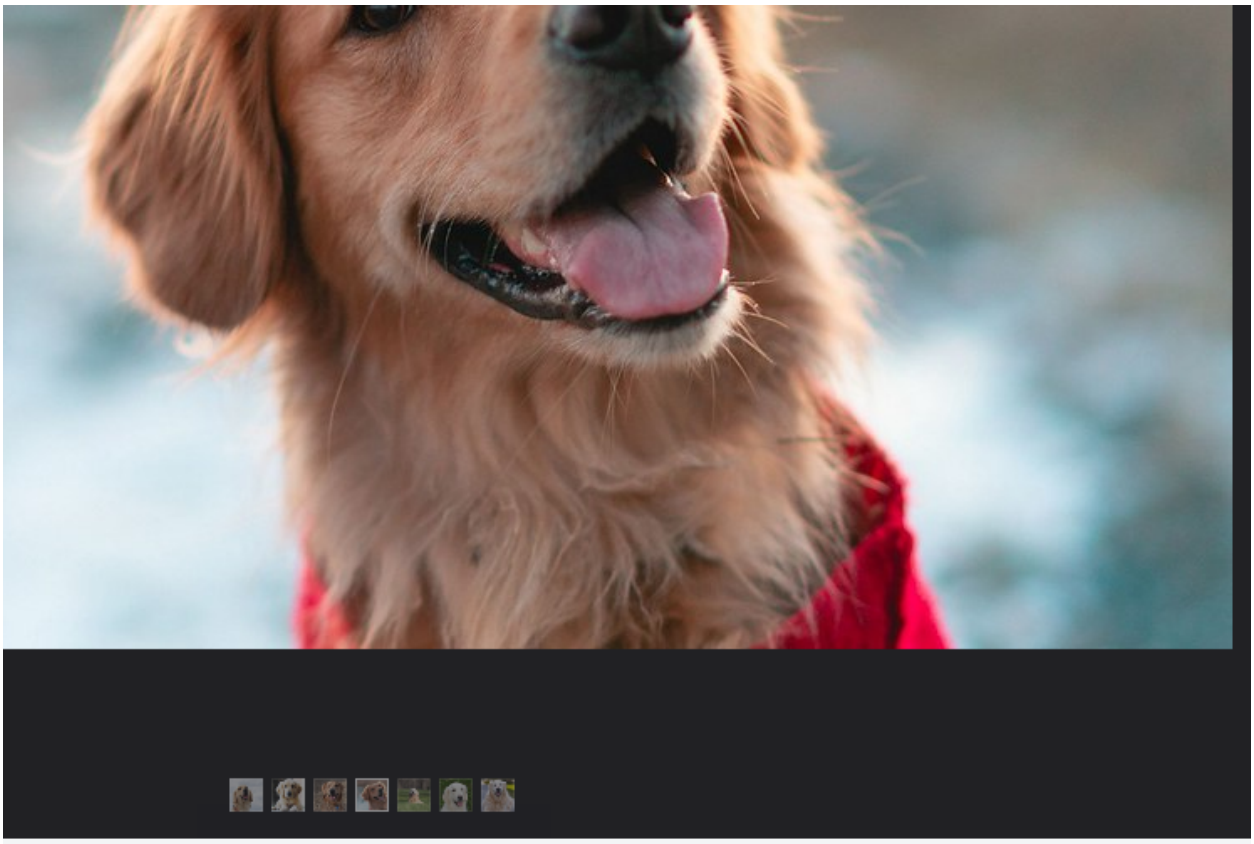


This whole map gave us a brief area of where he and his neighbors live. With this location information, we can narrow down to search for the nearest fire station to Whitehorn.



Comparing with other locations, we found the nearest location to the fire station, which is his house and retrieve the GPS coordinates from the EXIF data.





0
views

0
faves

0
comments

Uploaded on March 14, 2024

© All rights reserved

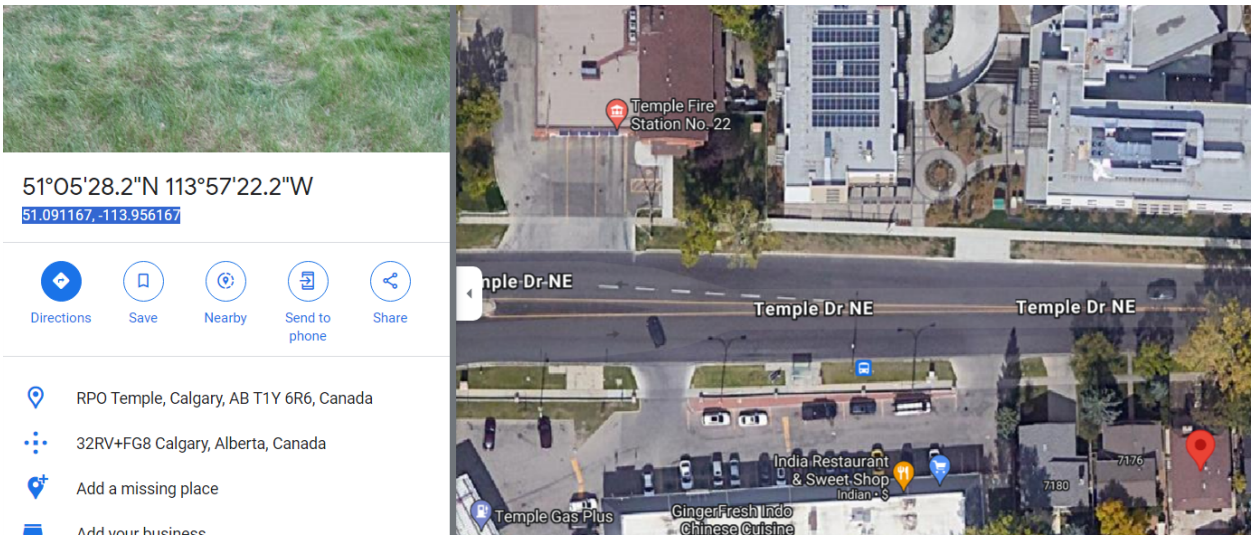
of Flickr Pro

Add comment

Hide EXIF

X-Resolution - 72 dpi
Y-Resolution - 72 dpi
YCbCr Positioning - Centered
GPS Version ID - 2.3.0.0
GPS Latitude Ref - North
GPS Latitude - 51 deg 5' 28.20"
GPS Longitude Ref - West
GPS Longitude - 113 deg 57' 22.20"
GPS Altitude Ref - Above Sea Level

Temple, Alberta, Canada

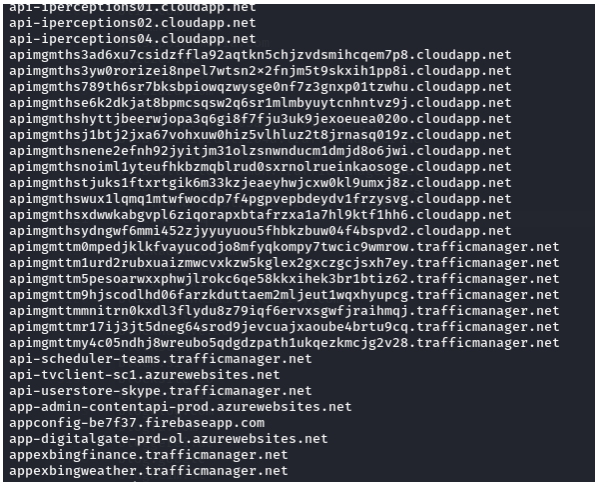


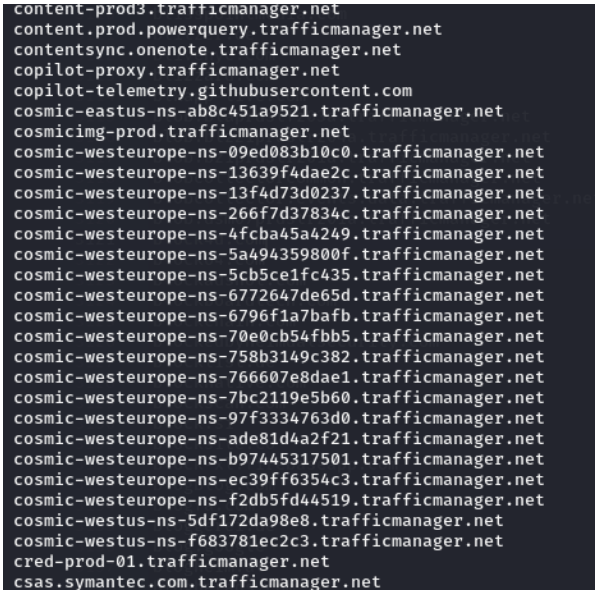
Log Analysis [forensics]

> Hi there incident responder. So we have
> this company that was breached sometime last week, but their SOC team
> only keeps HTTP request logs :(We took down all of our `wolvsecsolutions` websites as a precaution.
>
> *Maybe there's still a way to figure out what happened? Why did*
> *they click on a suspicious link? Somebody told me there's a flag on the*
> *link now?*

Flag: wctf{ph1sh3r5_l0v3_c0py1ng_d0m41n_n4m35}

At first, while looking at the logs, I noticed there’s a lot of random value in subdomain, which made me thinking, is it extracting values from the url?





After some trial and error from the above urls with hex and base64, I realized its not THAT hard. I'll show you some of the dumb stuff I did.

```
# Retrieve urls only
grep -i "^Host:" logs.txt | cut -d " " -f 2- | sort | uniq |
less

# Grep for base64 encoded formats in subdomain
grep -i "^Host:" logs.txt | cut -d " " -f 2- | sort | uniq |
cut -d '.' -f 1 | grep -P '[A-Za-z0-9+/]{4,}' | less

# Grep for the domain for each urls without tld
grep -i "^Host:" logs.txt | cut -d " " -f 2- | rev | cut -d "
" -f 2 | rev | sort | uniq | less

# Suspicious links may have longer length
grep -i "^Host:" logs.txt | cut -d " " -f 2- | sort | uniq |
awk 'length($0) > 30' | less
```

Then, I realized the scenario mentioned something about `wolvsecsolutions` taken down. Let's take a look at the results.

```
┌──(kali㉿kali)-[~/Downloads]
└─$ grep -i "^Host:" logs.txt | cut -d " " -f 2- | sort | uniq | grep 'wolvsec'
data.wolvsecsolutions.wolvctf.io
dev2.wolvsecsolutions.wolvctf.io
dev3.wolvsecsolutions.wolvctf.io
dev.wolvsecsolutions.wolvctf.io
files.wolvsecsolutions.wolvctf.io
help.wolvsecsolutions.wolvctf.io
solutions.wolvsecsolutions.wolvctf.io
support.wolvsecsolutions.wolvctf.io
wolvsecsolutions-okntin33tq-ul.a.run.app
wolvsecsolutions.wolvctf.io
```

Seems like we found a phishing link that tried to imitate the original website. AND from the link we got the flag.