HTB Sherlock - Compromised Writeup

Description:

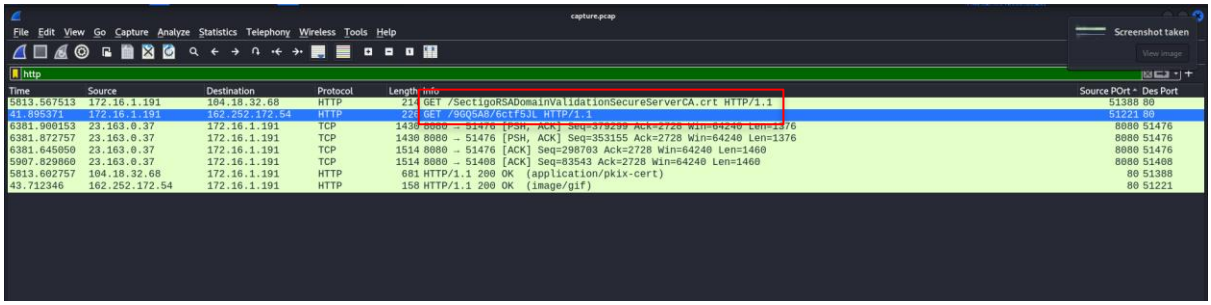
Our SOC team detected suspicious activity in Network Traffic, the machine has been compromised and company information that should not have been there has now been stolen – it's up to you to figure out what has happened and what data has been taken.

Solution:

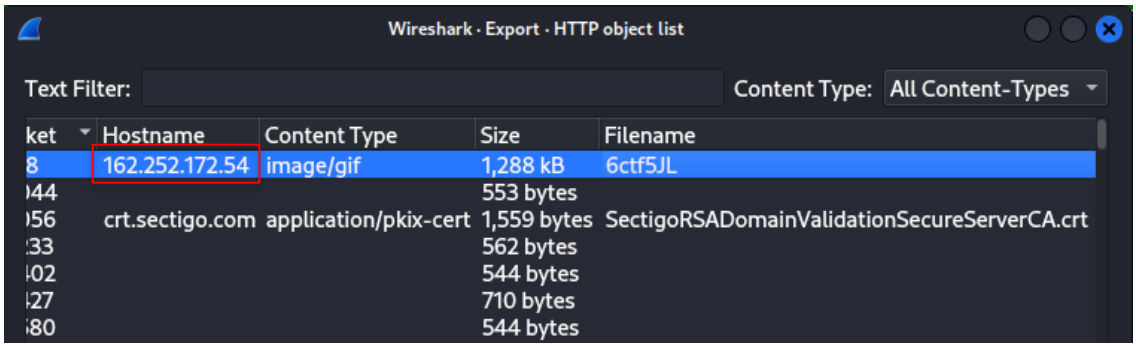
After downloading the zip file given, I extract it with the password *hacktheblue* given in the description. Inside the folder is a .pcap file so… it's time for Wireshark!

Task 1: What is the IP address used for initial access?

For Task 1, I filtered out HTTP traffic to focus on GET requests, as the description indicated the intruder downloaded something from the web. GET requests are commonly used for downloading files.



After examining the files, I found that *SectigoRSADomainValidationSecureServerCA.crt* wasn't suspicious, so I ignored it. However, the file `6ctf5JL` seemed suspicious, so I exported it and discovered an IP address: **162.252.172.54**, which was the answer for Task 1.



Task 2: What is the SHA256 hash of the malware?

After exporting out the suspicious file `6ctf5JL`, I did a file command on the filename to check the type of the file:

```
┌──(kali㉿kali)-[~/HTB-Sherlock]
└─$ ls
6ctf5JL  capture.pcap

┌──(kali㉿kali)-[~/HTB-Sherlock]
└─$ file 6ctf5JL
6ctf5JL: PE32 executable (DLL) (GUI) Intel 80386, for MS Windows, 6 sections
```

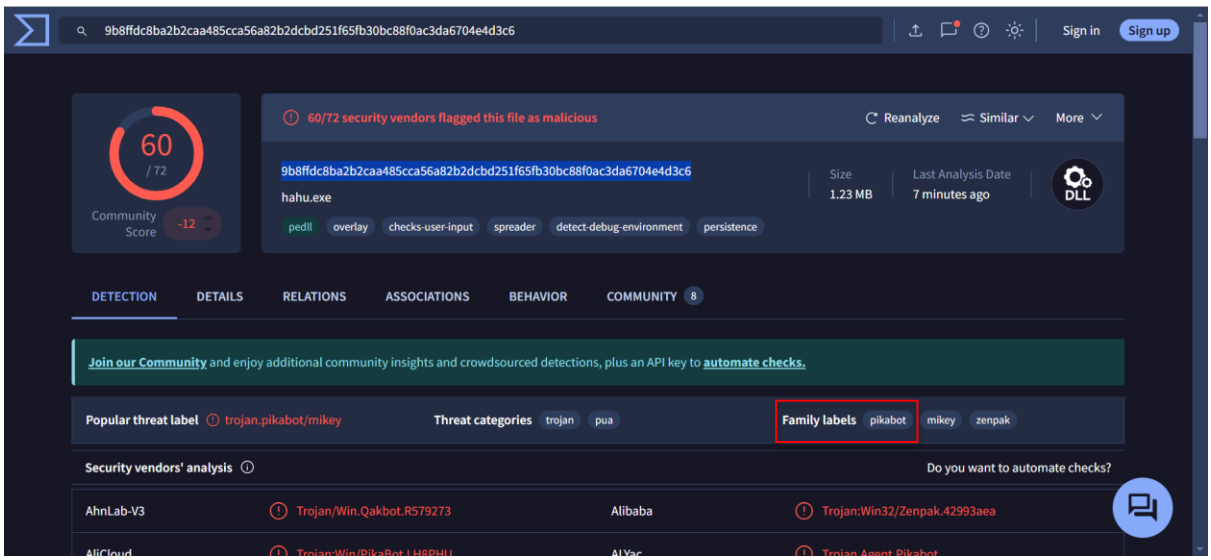
It turns out to be an executable file which proves that it is more likely the malware that we are looking for. After that, I use the sha256sum command to turn it into SHA256 hash.

```
┌──(kali㉿kali)-[~/HTB-Sherlock]
└─$ sha256sum 6ctf5JL
9b8ffdc8ba2b2caa485cca56a82b2dcbd251f65fb30bc88f0ac3da6704e4d3c6  6ctf5JL
```

There you go the answer for Task 2 is:
9b8ffdc8ba2b2caa485cca56a82b2dcbd251f65fb30bc88f0ac3da6704e4d3c6

Task 3: What is the Family label of the malware? (starts with p)

I copied the hash into Virus Total to analyse the malware. VirusTotal aggregates antivirus results to identify malware and its family, making it useful for confirming the threat's origin. (It is also in the hint to check out virus total 😉)

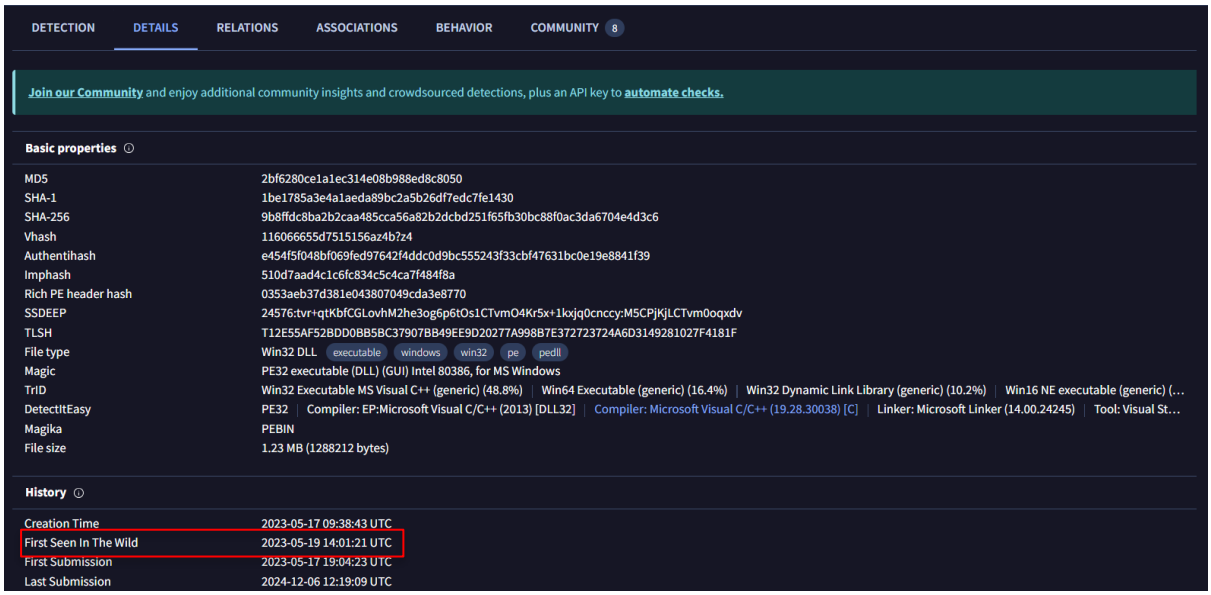


As you can see, we can find the family label that starts with p under detection section.

Therefore, the answer for task 3 is: **pikabot**

Task 4: When was the malware first seen in the wild (UTC)?

We can find the answer under the Details section:

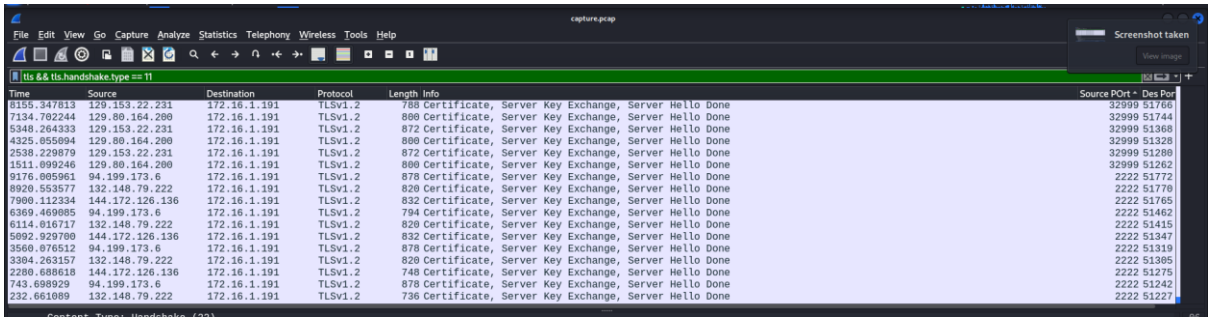


The answer for Task 4 is: **2023-05-19 14:01:21**

Task 5: The malware used HTTPS traffic with a self-signed certificate. What are the ports, from smallest to largest? (three ports)

After reading the task, I focused on the keywords *HTTPS*, *self-signed certificate,* and *ports*.

The first thing I did is to filtered the packets using *tls* in Wireshark, as TLS 1.2 indicates HTTPS traffic. To narrow down further, I applied the filter *tls.handshake.type == 11*, which isolates certificates exchanged during the handshake process.



Next, I sorted the source ports in ascending order where I found four distinct source ports: 443, 2078, 2222, and 32999.

Source POrt	Des Port
443	51218
443	51260
2078	51226
2078	51268
2078	51271
2078	51291
2078	51300
2078	51336
2078	51345
2078	51369
2078	51396
2078	51748
2078	51762
2078	51767
2078	51768
2222	51227
2222	51242
2222	51275
2222	51305
2222	51319
2222	51347
2222	51415
2222	51462
2222	51765
2222	51770
2222	51772
32999	51262
32999	51280
32999	51328
32999	51368
32999	51744
32999	51766

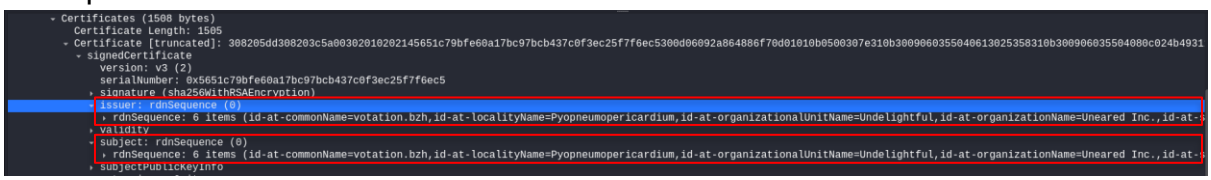
Moving on, I started researching on what self-signed certificates are and found out that it is a digital certificate that is issued and signed by the same entity, rather than a trusted Certificate Authority (CA). This means that the certificate's issuer and subject fields are identical, which distinguishes it from a CA-signed certificate.

To identify self-signed certificates, I compared the *issuer* and *subject* fields of each certificate. If they matched, the certificate was self-signed; otherwise, it was signed by a CA.

Example 1:

The issuer and subject does not match so this is not a self-signed certificate.

Example 2:



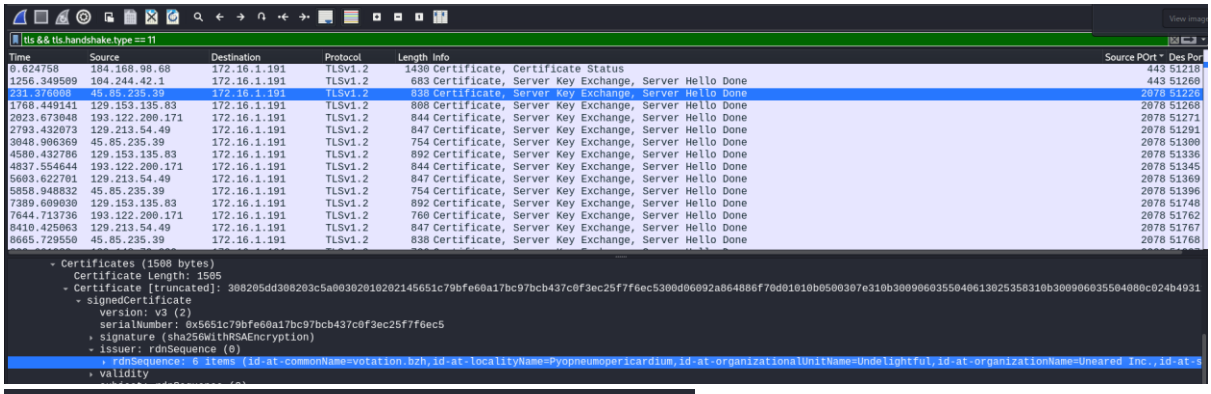
The issuer and subject matches, therefore this is a self-signed certificate.

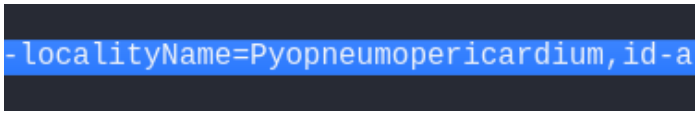
After reviewing all 32 filtered packets, I found two certificates that were CA-signed and the rest were self-signed. The self-signed certificates were exclusively associated with ports 2078, 2222, and 32999.

Voilà! The answer for task 5 is **2078, 2222, 32999**.

Task 6: What is the id-at-localityName of the self-signed certificate associated with the first malicious IP?

We can find the answer easily by reviewing the first self-signed certificate on port 2078 and find:

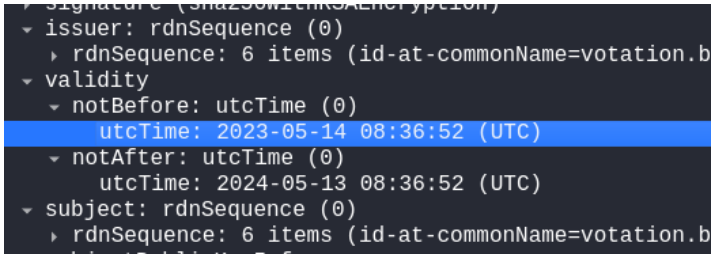




The answer for task 6 is: **Pyopneumopericardium.**

Task 7: What is the notBefore time(UTC) for this self-signed certificate?

You can find the answer directly in the validity section in between issuer and subject

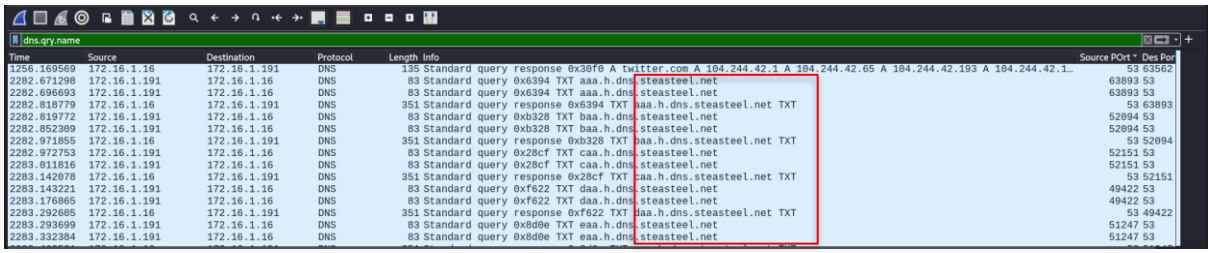


The answer for task 7 is: **2023-05-14 08:36:52.**

Task 8: What was the domain used for tunneling?

Tunneling in this context refers to malware encapsulating its communication within legitimate protocols to evade detection. DNS traffic, for instance, is commonly used due to its legitimate and frequent presence in networks, making it less suspicious.

To identify the domain used for tunneling, I filtered the DNS traffic in Wireshark using *dns.qry.name*. This filter isolates DNS query packets, helping me focus on domains being queried during the network activity.



Upon analyzing the filtered results, I noticed a series of unusual DNS queries to subdomains of steasteel.net (e.g., aaa.h.dns.steasteel.net, baa.h.dns.steasteel.net, caa.h.dns.steasteel.net, etc.). The repetitive pattern of these subdomains, combined with the structured naming convention, strongly indicates potential DNS tunneling activity.

The answer for task 8 is: **steasteel.net.**