Ph4nt0m 1ntrud3r - Writeup

June 13th 2025

Maciej Jędrak, Ludwik Rydzak

Platform: PicoCTF

Category: Forensics

Tags: [tshark, wireshark]

Description

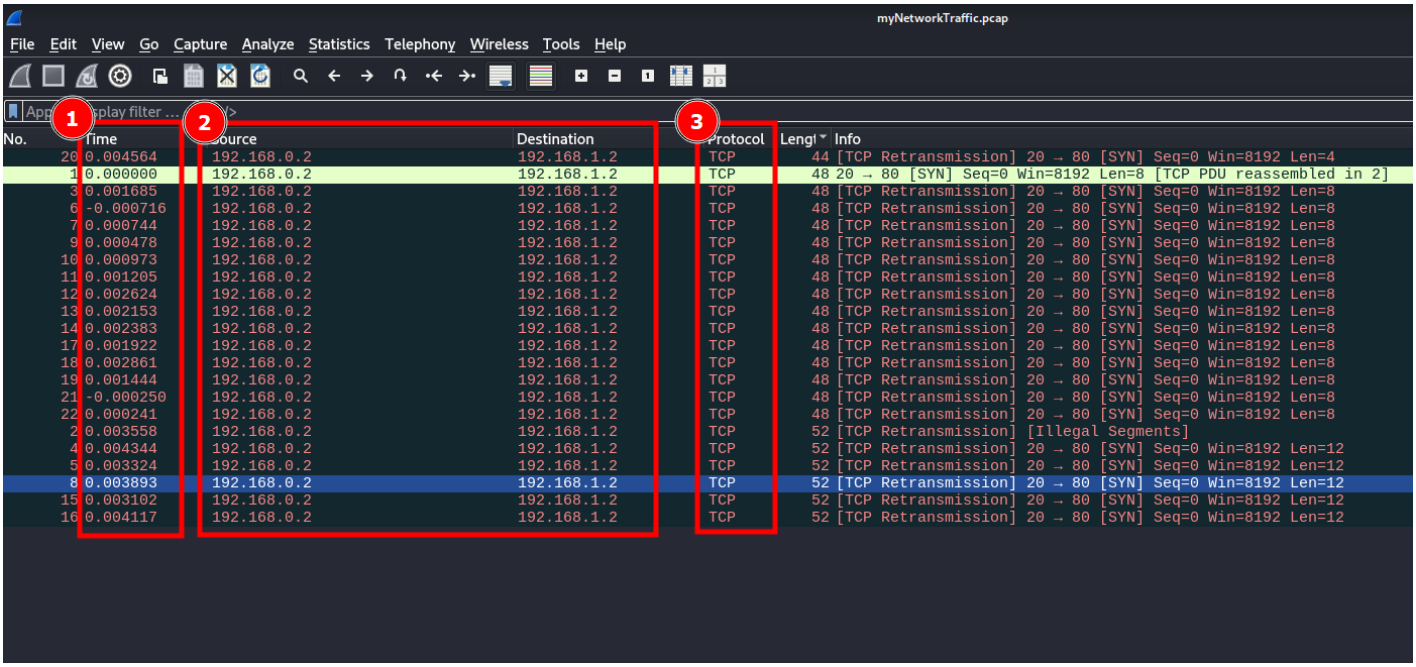
A digital ghost has breached my defenses, and my sensitive data has been stolen! Your mission is to uncover how this phantom intruder infiltrated my system and retrieve the hidden flag. To solve this challenge, you'll need to analyze the provided PCAP file and track down the attack method. The attacker has cleverly concealed his moves in well timely manner. Dive into the network traffic, apply the right filters and show off your forensic prowess and unmask the digital intruder!

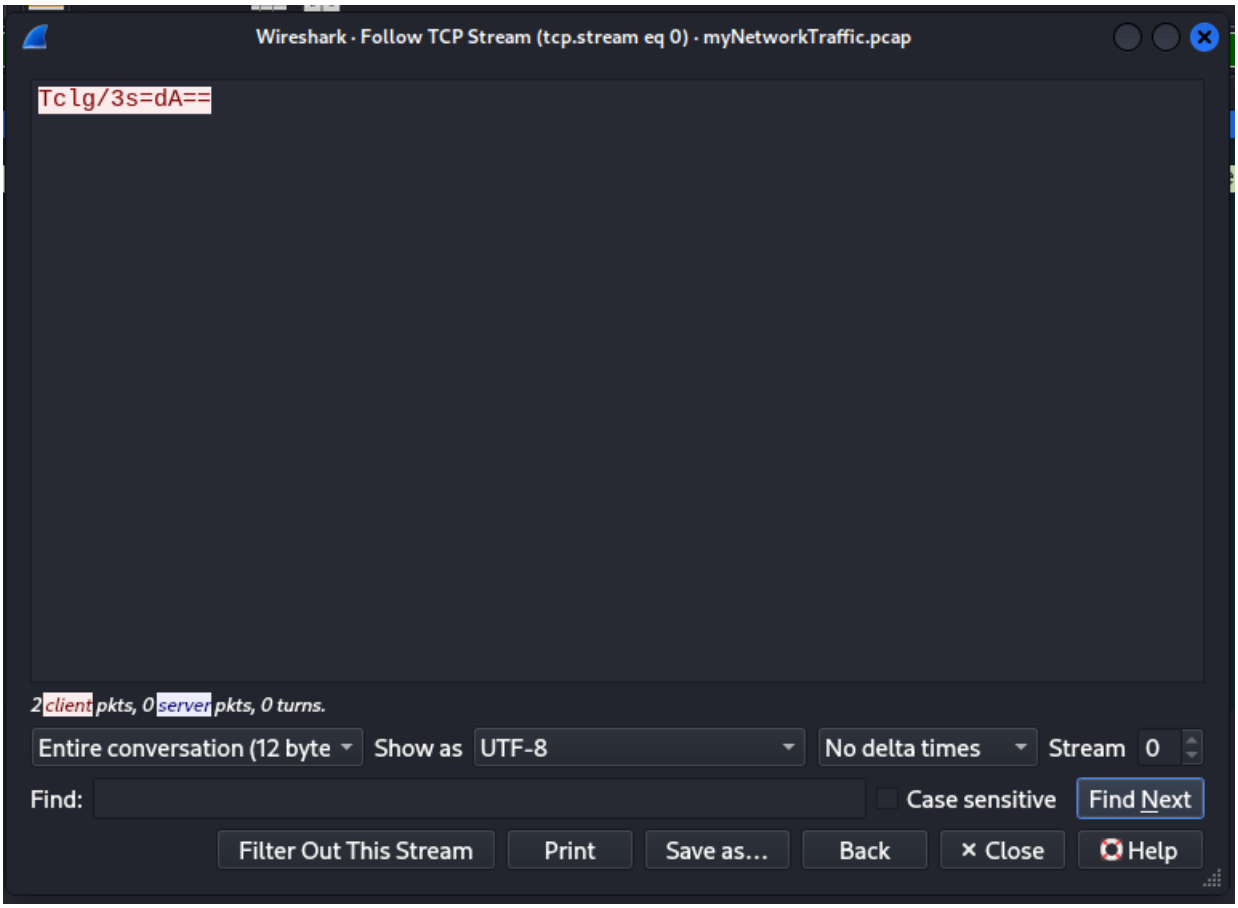
Overview

In this CTF challenge, the objective was to extract a flag from network traffic captured in a file. The process began with the acquisition of a packet capture (PCAP) file named `myNetworkTraffic.pcap`

Solution

The first step involved opening the PCAP file using Wireshark, a widely-used network protocol analyzer that enables users to capture and interactively browse the traffic running on a computer network. This provided an overview of the network traffic, which was primarily composed of TCP protocol communications containing obfuscated text.





To further analyze the traffic, the decision was made to utilize TShark, which is the command-line version of Wireshark and is useful for processing and analyzing packet data in a more automated fashion.

1. **Command:** `tshark -r myNetworkTraffic.pcap`
   This command reads the specified PCAP file and displays the packet data in the terminal. It allows for a quick review of the captured packets, providing insights into the types of traffic present.

```
┌──(kali㉿kali)-[~/pico]
└─$ tshark -r myNetworkTraffic.pcap
    1   0.000000  192.168.0.2 → 192.168.1.2  TCP 48 20 → 80 [SYN] Seq=0 Win=8192 Len=8
    2   0.003558  192.168.0.2 → 192.168.1.2  TCP 52 [TCP Retransmission] [Illegal Segments]
    3   0.001685  192.168.0.2 → 192.168.1.2  TCP 48 [TCP Retransmission] 20 → 80 [SYN] Seq=0 Win=8192 Len=8
    4   0.004344  192.168.0.2 → 192.168.1.2  TCP 52 [TCP Retransmission] 20 → 80 [SYN] Seq=0 Win=8192 Len=12
    5   0.003324  192.168.0.2 → 192.168.1.2  TCP 52 [TCP Retransmission] 20 → 80 [SYN] Seq=0 Win=8192 Len=12
    6  -0.000716  192.168.0.2 → 192.168.1.2  TCP 48 [TCP Retransmission] 20 → 80 [SYN] Seq=0 Win=8192 Len=8
    7   0.000744  192.168.0.2 → 192.168.1.2  TCP 48 [TCP Retransmission] 20 → 80 [SYN] Seq=0 Win=8192 Len=8
    8   0.003893  192.168.0.2 → 192.168.1.2  TCP 52 [TCP Retransmission] 20 → 80 [SYN] Seq=0 Win=8192 Len=12
    9   0.000478  192.168.0.2 → 192.168.1.2  TCP 48 [TCP Retransmission] 20 → 80 [SYN] Seq=0 Win=8192 Len=8
   10   0.000973  192.168.0.2 → 192.168.1.2  TCP 48 [TCP Retransmission] 20 → 80 [SYN] Seq=0 Win=8192 Len=8
   11   0.001205  192.168.0.2 → 192.168.1.2  TCP 48 [TCP Retransmission] 20 → 80 [SYN] Seq=0 Win=8192 Len=8
   12   0.002624  192.168.0.2 → 192.168.1.2  TCP 48 [TCP Retransmission] 20 → 80 [SYN] Seq=0 Win=8192 Len=8
   13   0.002153  192.168.0.2 → 192.168.1.2  TCP 48 [TCP Retransmission] 20 → 80 [SYN] Seq=0 Win=8192 Len=8
   14   0.002383  192.168.0.2 → 192.168.1.2  TCP 48 [TCP Retransmission] 20 → 80 [SYN] Seq=0 Win=8192 Len=8
   15   0.003102  192.168.0.2 → 192.168.1.2  TCP 52 [TCP Retransmission] 20 → 80 [SYN] Seq=0 Win=8192 Len=12
   16   0.004117  192.168.0.2 → 192.168.1.2  TCP 52 [TCP Retransmission] 20 → 80 [SYN] Seq=0 Win=8192 Len=12
   17   0.001922  192.168.0.2 → 192.168.1.2  TCP 48 [TCP Retransmission] 20 → 80 [SYN] Seq=0 Win=8192 Len=8
   18   0.002861  192.168.0.2 → 192.168.1.2  TCP 48 [TCP Retransmission] 20 → 80 [SYN] Seq=0 Win=8192 Len=8
   19   0.001444  192.168.0.2 → 192.168.1.2  TCP 48 [TCP Retransmission] 20 → 80 [SYN] Seq=0 Win=8192 Len=8
   20   0.004564  192.168.0.2 → 192.168.1.2  TCP 44 [TCP Retransmission] 20 → 80 [SYN] Seq=0 Win=8192 Len=4
   21  -0.000250  192.168.0.2 → 192.168.1.2  TCP 48 [TCP Retransmission] 20 → 80 [SYN] Seq=0 Win=8192 Len=8
   22   0.000241  192.168.0.2 → 192.168.1.2  TCP 48 [TCP Retransmission] 20 → 80 [SYN] Seq=0 Win=8192 Len=8
```

2. **Command:** `tshark -r myNetworkTraffic.pcap -T fields -e tcp.payload`
   This command extracts the TCP payloads from the packets in the PCAP file. By specifying `-T fields` and `-e tcp.payload`, it filters the output to show only the TCP payload data, which is essential for identifying any hidden information or flags.

```
┌──(kali㉿kali)-[~/pico]
└─$ tshark -r myNetworkTraffic.pcap -T fields -e tcp.payload
54636c672f33733d
626e52666447673064413d3d
5248786874533 43d
4e6a5a6b4d474a6d59673d3d
657a463058336330 63773d3d
524d712b77544d3d
37754443636c673d
587a4d3063336c6664413d3d
4f77466550304d3d
3470635977546 73d
326437314b5a493d
6f46705a5047383d
716f39717069593d
4a62473251377 73d
63476c6a62304e5552673d3d
596d68664e484a664f513d3d
5a314764796a6b3d
684b765a4b47413d
3677343651373 03d
66513d3d
39447049626b413d
514b7a46582b633d
```

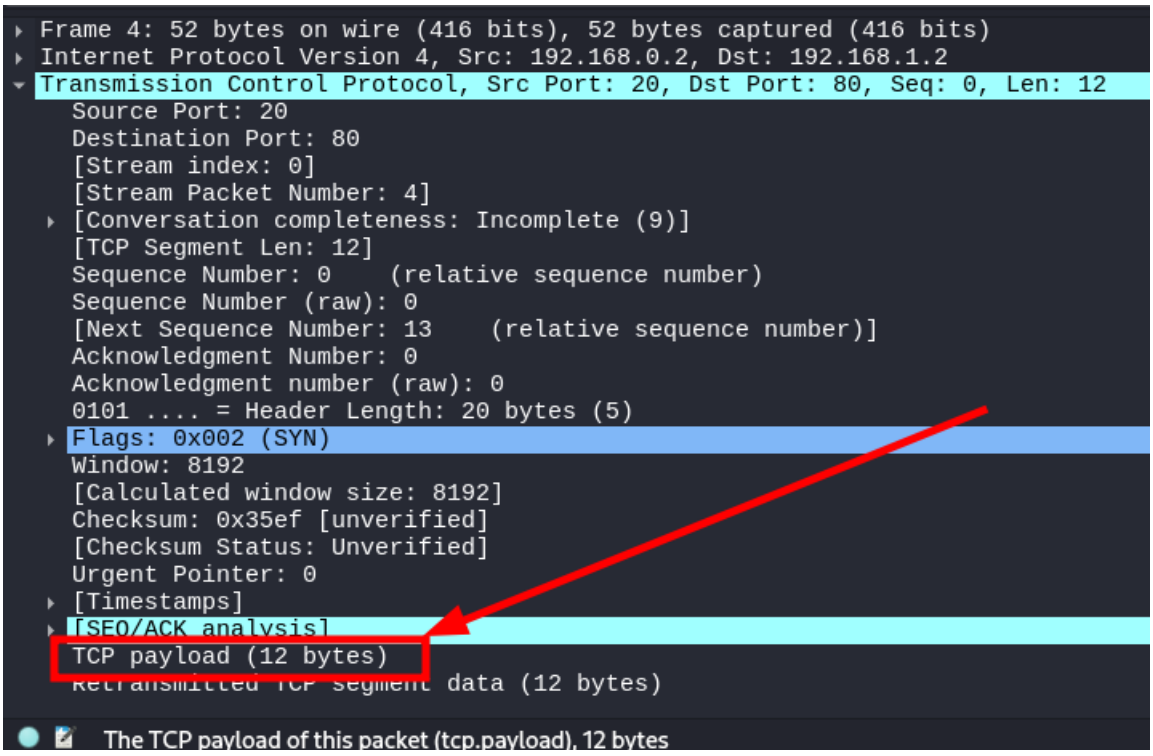
3. **Command:** `tshark -r myNetworkTraffic.pcap -T fields -e tcp.payload | xxd -p -r | base64 -d`
   This command takes the TCP payloads extracted from the previous command and pipes them through two additional processes. The `xxd -p -r` command converts the hexadecimal representation of the payload back into binary format.

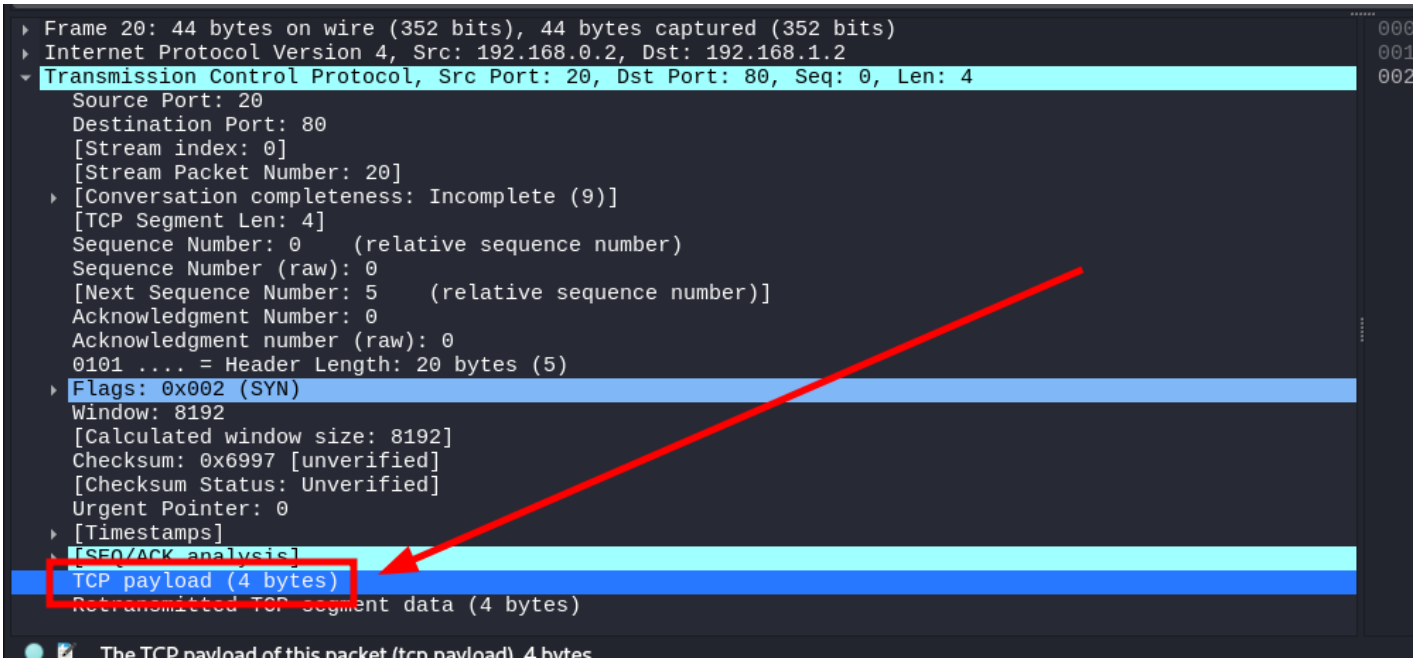
```
┌──(kali㉿kali)-[~/pico]
└─$ tshark -r myNetworkTraffic.pcap -T fields -e tcp.segment_data | xxd -p -r
Tclg/3s=bnRfdGg0dA==RHxhtS4=NjZkMGJmYg==ezF0X3c0cw==RMq+wTM=7uDCclg=XzM0c3lfdA==OwFeP0M=4pcYwTg=2d71KZI=oFpZPG8=qo9qpiY=JbG2Q7
w=cGljb0NURg==YmhfNHJfOQ==Z1Gdyjk=hKvZKGA=6w46Q70=fQ==9DpIbkA=QKzFX+c=
```

Convertion revealed additional encoding. That is why there was a need to include `base64 -d` command in the pipeline that decodes any Base64-encoded data. This step was crucial in revealing the underlying information hidden within the TCP payloads.

```
┌──(kali㉿kali)-[~/pico]
└─$ tshark -r myNetworkTraffic.pcap -T fields -e tcp.segment_data | xxd -p -r | base64 -d
M◆`◆{nt_th4tD|a◆.66d0bfb{1t_w4sD'◆3◆◆◆rX_34sy_t;^?C◆◆ ◆8◆◆◆)◆◆ZY<o◆◆j◆&%◆◆C◆picoCTFbh_4r_9gQ◆◆9◆◆◆(`◆:C◆}◆:Hn@@◆◆_◆
```

5. **Command:** `tshark -r myNetworkTraffic.pcap -T fields -e tcp.payload -Y "tcplen==12" | xxd -p -r | base64 -d`
   Here, the command is modified to filter the TCP payloads to only those with a length of 12 bytes using the display filter `-Y "tcplen==12"`. This targeted approach helps in narrowing down the search for the flag by focusing on payloads of a specific size, which may contain the flag. Investigating packet using wireshark revealed that size of TCP payload fluctuates around 12 Bytes.






```
┌──(kali㉿kali)-[~/pico]
└─$ tshark -r myNetworkTraffic.pcap -Y "tcp.len==12" -T fields -e tcp.payload | xxd -p -r | base64 -d

nt_th4t66d0bfb{1t_w4s_34sy_tpicoCTFbh_4r_9
```

By then it was clear that this is a right path, but unfortunately there was something wrong with the order. Basing on the first column with timing of each packet it was quite obvious that it needs to be fixed.

6. **Command:** `tshark -r myNetworkTraffic.pcap -T fields -e frame.time -e tcp.payload -Y "tcplen==12 || tcp.len==4" | sort -k4 | awk '{print $6}' | xxd -p -r | base64 -d`
   In the final command, the filtering is expanded to include TCP payloads with a length of either 12 bytes or 4 bytes (`-Y "tcplen==12 || tcp.len==4"` , there were also 8-Bytes-long packets but they didn't contain any relevant data) and sorting packets in right time manner (`-e frame.time`). The results are sorted based on the fourth field, and the sixth field is extracted using `awk` . The extracted data is then processed through `xxd -p -r` and `base64 -d` to decode the information. This comprehensive command ultimately revealed the flag hidden within the network traffic.

References

https://www.wireshark.org/docs/wsug_html_chunked/

https://www.wireshark.org/docs/man-pages/tshark.html