TryHackMe: TShark

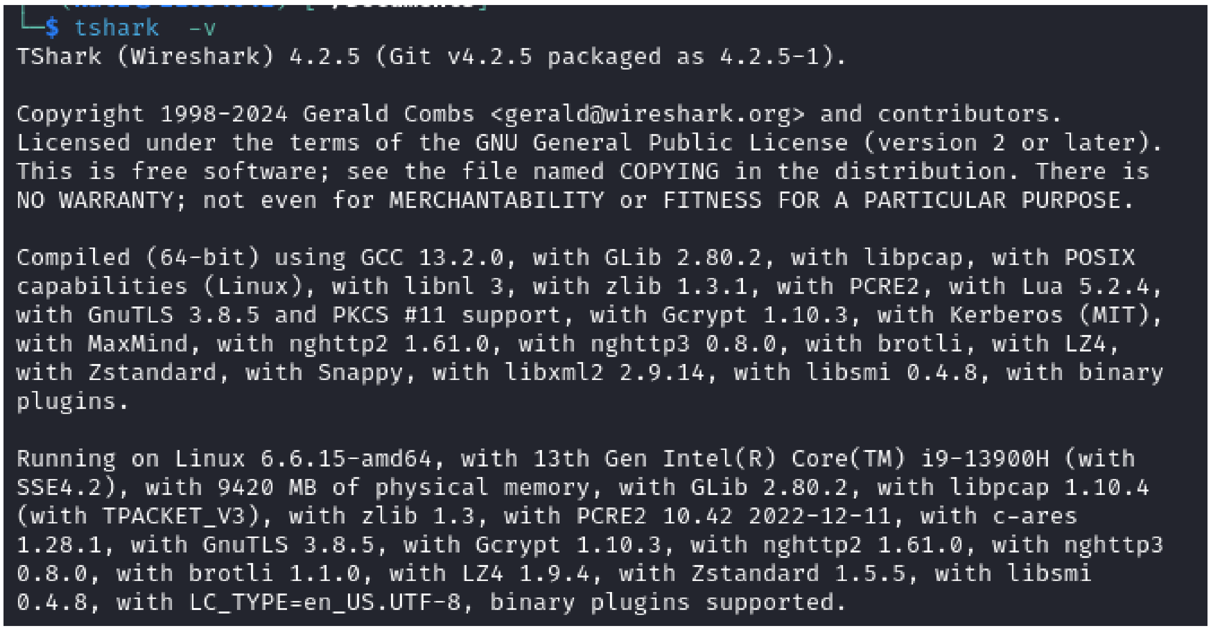
Recently, I completed a TryHackMe room called called TShark. This beginner-friendly room provided a fantastic introduction to TShark's capabilities, and I'd like to share my experience with you all. I really hope you enjoy this room as much as I did.

1. Introduction to TShark

TShark is a network protocol analyser, and is quite literally a CLI version of Wireshark. It enables you to capture packets or read packets from a pcap file. TSharks native capture file format is pcapng which is also supported by Wireshark and many other network forensic tools. If you start tshark without any options/flags, it will act basically the same as tcpdump. When run with the -r option and specifying a capture file, TShark will read packets from the file and display a summary line for each packet read from the file. For more information on TShark, visit the official website here.

2. Verifying TShark Installation

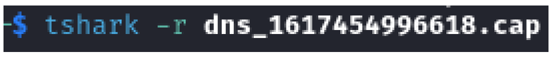
To verify that tshark is installed (should come installed with Wireshark), enter the following command:



```
└─$ tshark -v
TShark (Wireshark) 4.2.5 (Git v4.2.5 packaged as 4.2.5-1).

Copyright 1998-2024 Gerald Combs <gerald@wireshark.org> and contributors.
Licensed under the terms of the GNU General Public License (version 2 or later).
This is free software; see the file named COPYING in the distribution. There is
NO WARRANTY; not even for MERCHANTABILITY or FITNESS FOR A PARTICULAR PURPOSE.

Compiled (64-bit) using GCC 13.2.0, with GLib 2.80.2, with libpcap, with POSIX
capabilities (Linux), with libnl 3, with zlib 1.3.1, with PCRE2, with Lua 5.2.4,
with GnuTLS 3.8.5 and PKCS #11 support, with Gcrypt 1.10.3, with Kerberos (MIT),
with MaxMind, with nghttp2 1.61.0, with nghttp3 0.8.0, with brotli, with LZ4,
with Zstandard, with Snappy, with libxml2 2.9.14, with libsmi 0.4.8, with binary
plugins.

Running on Linux 6.6.15-amd64, with 13th Gen Intel(R) Core(TM) i9-13900H (with
SSE4.2), with 9420 MB of physical memory, with GLib 2.80.2, with libpcap 1.10.4
(with TPACKET_V3), with zlib 1.3, with PCRE2 10.42 2022-12-11, with c-ares
1.28.1, with GnuTLS 3.8.5, with Gcrypt 1.10.3, with nghttp2 1.61.0, with nghttp3
0.8.0, with brotli 1.1.0, with LZ4 1.9.4, with Zstandard 1.5.5, with libsmi
0.4.8, with LC_TYPE=en_US.UTF-8, binary plugins supported.
```

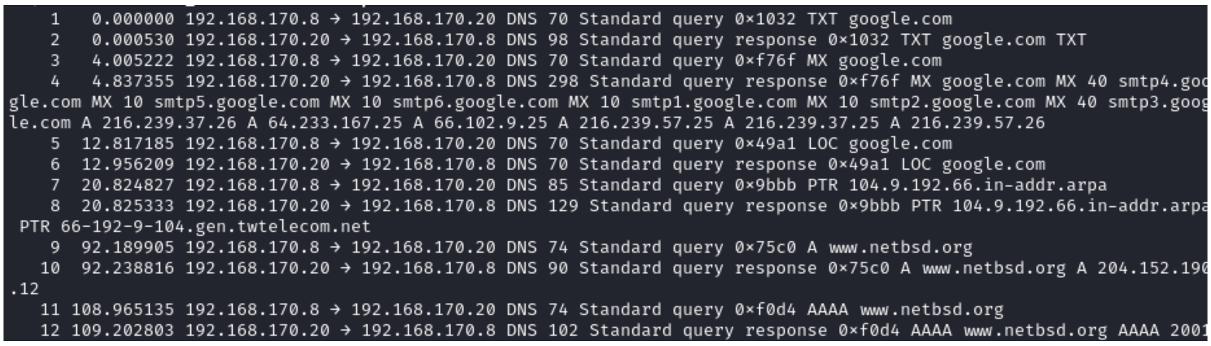
3. Reading a Capture File

To start reading a file, all we need to do is use the -r switch:



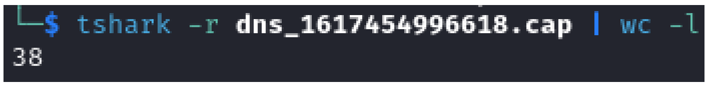
```
$ tshark -r dns_1617454996618.cap
```

You are then provided with a bunch of raw output:



```
    1   0.000000 192.168.170.8 → 192.168.170.20 DNS 70 Standard query 0×1032 TXT google.com
    2   0.000530 192.168.170.20 → 192.168.170.8 DNS 98 Standard query response 0×1032 TXT google.com TXT
    3   4.005222 192.168.170.8 → 192.168.170.20 DNS 70 Standard query 0×f76f MX google.com
    4   4.837355 192.168.170.20 → 192.168.170.8 DNS 298 Standard query response 0×f76f MX google.com MX 40 smtp4.goo
gle.com MX 10 smtp5.google.com MX 10 smtp6.google.com MX 10 smtp1.google.com MX 10 smtp2.google.com MX 40 smtp3.goog
le.com A 216.239.37.26 A 64.233.167.25 A 66.102.9.25 A 216.239.57.25 A 216.239.37.25 A 216.239.57.26
    5  12.817185 192.168.170.8 → 192.168.170.20 DNS 70 Standard query 0×49a1 LOC google.com
    6  12.956209 192.168.170.20 → 192.168.170.8 DNS 70 Standard query response 0×49a1 LOC google.com
    7  20.824827 192.168.170.8 → 192.168.170.20 DNS 85 Standard query 0×9bbb PTR 104.9.192.66.in-addr.arpa
    8  20.825333 192.168.170.20 → 192.168.170.8 DNS 129 Standard query response 0×9bbb PTR 104.9.192.66.in-addr.arpa
 PTR 66-192-9-104.gen.twtelecom.net
    9  92.189905 192.168.170.8 → 192.168.170.20 DNS 74 Standard query 0×75c0 A www.netbsd.org
   10  92.238816 192.168.170.20 → 192.168.170.8 DNS 90 Standard query response 0×75c0 A www.netbsd.org A 204.152.190
.12
   11 108.965135 192.168.170.8 → 192.168.170.20 DNS 74 Standard query 0×f0d4 AAAA www.netbsd.org
   12 109.202803 192.168.170.20 → 192.168.170.8 DNS 102 Standard query response 0×f0d4 AAAA www.netbsd.org AAAA 2001
```

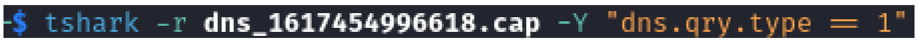
If you wish to count how many packets are in the packet capture, you can pipe it to wc -l:



```
└─$ tshark -r dns_1617454996618.cap | wc -l
38
```

4. Utilising Display Filters

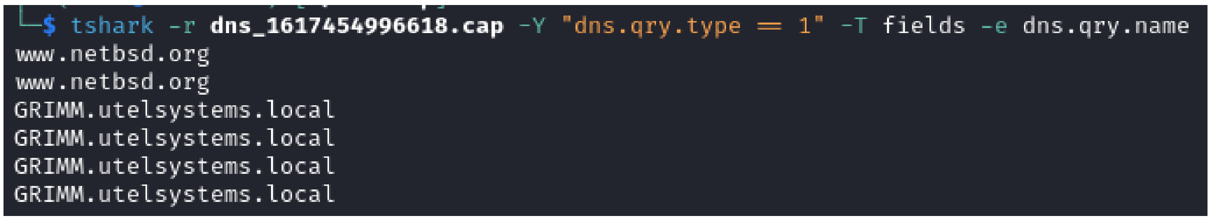
Similar to Wireshark, we can utilise Wireshark display filters to narrow down the packets even further. For example, if you were only interested in DNS A records you would simply need to enter:



```
-$ tshark -r dns_1617454996618.cap -Y "dns.qry.type == 1"
```

5. Extracting Specific Fields

Display filters are added by using the -Y switch as demonstrated above. If we want to cut out even more noise, we can extract specific fields using the -T fields and -e fieldname switches:



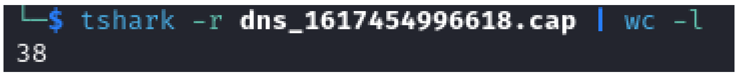
```
└─$ tshark -r dns_1617454996618.cap -Y "dns.qry.type == 1" -T fields -e dns.qry.name
www.netbsd.org
www.netbsd.org
GRIMM.utelsystems.local
GRIMM.utelsystems.local
GRIMM.utelsystems.local
GRIMM.utelsystems.local
```

As you can see, the command above only prints out the dns.qry.name field which is simply the domain that was resolved.

6. Challenge

How many packets are in the dns.cap file?

As demonstrated previously, we can just read the pcap file and pipe it to wc -l:

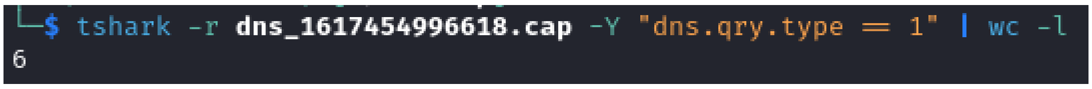


```
└─$ tshark -r dns_1617454996618.cap | wc -l
38
```

The answer is 38.

How many A records are in the capture? (including responses)

To find all the A records, all we need to do is enter the display filter ‘dns.qry.type == 1’ after the -Y switch:

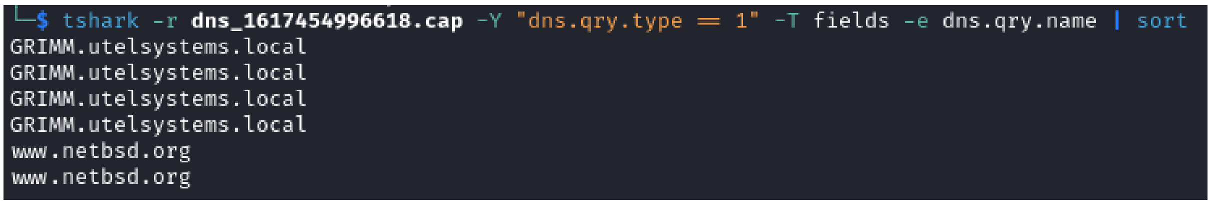


```
└─$ tshark -r dns_1617454996618.cap -Y "dns.qry.type == 1" | wc -l
6
```

The answer is 6.

Which A record was present the most?

All we need to do is remove wc -l from the previous command and pipe it to sort instead:



```
└─$ tshark -r dns_1617454996618.cap -Y "dns.qry.type == 1" -T fields -e dns.qry.name | sort
GRIMM.utelsystems.local
GRIMM.utelsystems.local
GRIMM.utelsystems.local
GRIMM.utelsystems.local
www.netbsd.org
www.netbsd.org
```

As you can see, the ‘GRIMM.utelsystems.local’ is resolved the most.

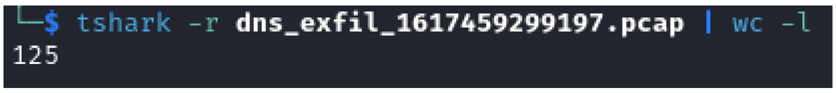
Scenario: We’ve been alerted that a host in our network has been exfiltrating data over DNS, can you find it?

Use the attached file to analyse in Wireshark and TShark to find the exfiltrated data. As you identify suspicious items in Wireshark, pivot to TShark to extract relevant information.

I am personally going to try and only use TShark as this is the tool I want to practice with (not Wireshark).

How many packets are in this capture?

Like done in the previous questions, we can determine the number of packets in a pcap file by piping it to wc -l:

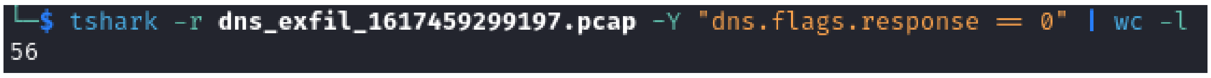


```
└─$ tshark -r dns_exfil_1617459299197.pcap | wc -l
125
```

The answer is 125.

How many DNS queries are in this pcap? (Not responses!)

As explained in the hint, we can use the ‘dns.flags.response == 0’ flag like as follows:

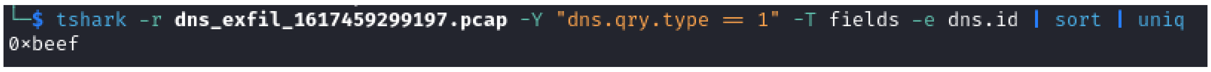


```
└─$ tshark -r dns_exfil_1617459299197.pcap -Y "dns.flags.response == 0" | wc -l
56
```

The answer is 56.

What is the DNS transaction ID of the suspicious queries (in hex)?

If you enter the following command, we are able to sort for all DNS queries and display uniq ID field values:

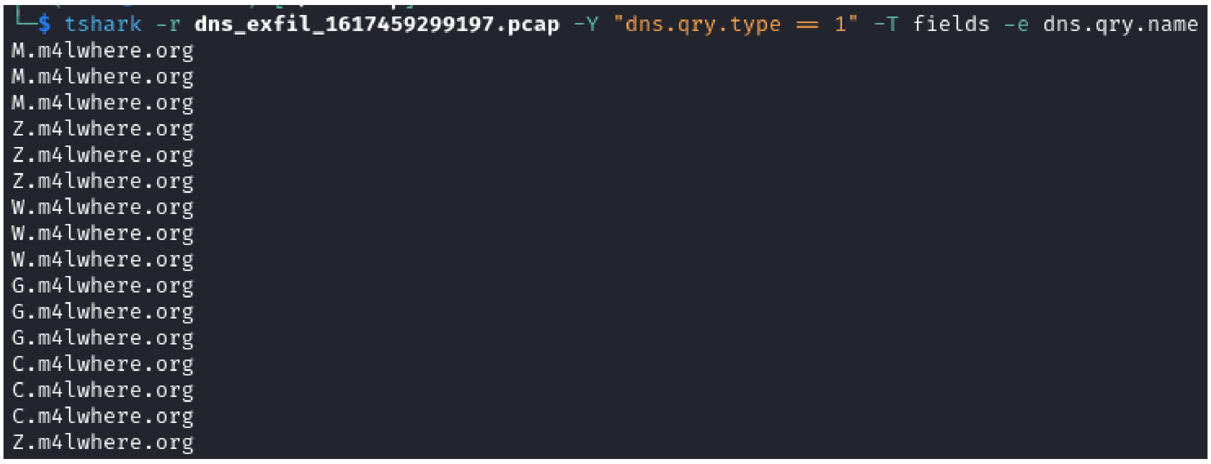


```
└─$ tshark -r dns_exfil_1617459299197.pcap -Y "dns.qry.type == 1" -T fields -e dns.id | sort | uniq
0×beef
```

As you can see, 0xbeef is the only query ID and is therefore the answer.

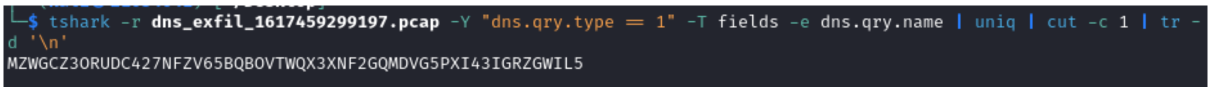
What is the string extracted from the DNS queries?

If you explore all the DNS queries and look for unique results, you can notice that the subdomain for each query is different (classic data exfil technique):



```
└─$ tshark -r dns_exfil_1617459299197.pcap -Y "dns.qry.type == 1" -T fields -e dns.qry.name
M.m4lwhere.org
M.m4lwhere.org
M.m4lwhere.org
Z.m4lwhere.org
Z.m4lwhere.org
Z.m4lwhere.org
W.m4lwhere.org
W.m4lwhere.org
W.m4lwhere.org
G.m4lwhere.org
G.m4lwhere.org
G.m4lwhere.org
C.m4lwhere.org
C.m4lwhere.org
C.m4lwhere.org
Z.m4lwhere.org
```

Leet's extract the first character of each query like as follows:

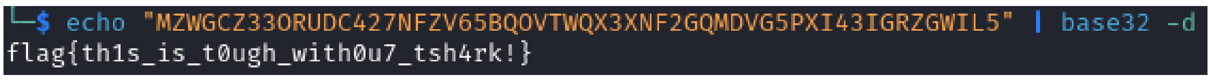


```
└─$ tshark -r dns_exfil_1617459299197.pcap -Y "dns.qry.type == 1" -T fields -e dns.qry.name | uniq | cut -c 1 | tr -
d '\n'
MZWGCZ3ORUDC427NFZV65BQBOVTWQX3XNF2GQMDVG5PXI43IGRZGWIL5
```

cut -c 1 simply cuts and prints the first character, it is then piped to tr -d which removes the newline character, so it prints in the answer format. Simply copy and paste the output which is the answer.

What is the flag?

The previous text is base32 encoded as stated in the hint, to decode it, you can use Cyberchef and other similar tools, or you can echo the string and pipe it to base32 -d:



```
└─$ echo "MZWGCZ33ORUDC427NFZV65BQOVTWQX3XNF2GQMDVG5PXI43IGRZGWIL5" | base32 -d
flag{th1s_is_t0ugh_with0u7_tsh4rk!}
```

This TryHackMe room provided an excellent introduction to using TShark for network packet analysis and forensic investigations. It was my first time using TShark, and I thoroughly enjoyed it. Feel free to reach out if you need any help or have feedback on this writeup.