

CS790 Assignment 2

The Evil Eye Report

Kavya Gupta (22B1053)

Department of Computer Science,
Indian Institute of Technology Bombay

Approach

I observed that the Vanilla TOR uses (18) specific cipher suites (See Figure 1). So, for any TLS Client Hello, I get the cipher suite list and match it with these 18. If they match exactly, I deem the destination IP address as a TOR guard IP (and add it to a list). After this, any TCP packet, with (source/destination) IP same as any present in the list, is marked as a TOR packet.

```

  ✓ Cipher Suites (18 suites)
    Cipher Suite: TLS_AES_256_GCM_SHA384 (0x1302)
    Cipher Suite: TLS_CHACHA20_POLY1305_SHA256 (0x1303)
    Cipher Suite: TLS_AES_128_GCM_SHA256 (0x1301)
    Cipher Suite: TLS_ECDHE_ECDSA_WITH_AES_128_GCM_SHA256 (0xc02b)
    Cipher Suite: TLS_ECDHE_RSA_WITH_AES_128_GCM_SHA256 (0xc02f)
    Cipher Suite: TLS_ECDHE_ECDSA_WITH_CHACHA20_POLY1305_SHA256 (0xc030)
    Cipher Suite: TLS_ECDHE_RSA_WITH_CHACHA20_POLY1305_SHA256 (0xc031)
    Cipher Suite: TLS_ECDHE_ECDSA_WITH_AES_256_GCM_SHA384 (0xc032)
    Cipher Suite: TLS_ECDHE_RSA_WITH_AES_256_GCM_SHA384 (0xc033)
    Cipher Suite: TLS_ECDHE_ECDSA_WITH_AES_256_CBC_SHA (0xc00a)
    Cipher Suite: TLS_ECDHE_ECDSA_WITH_AES_128_CBC_SHA (0xc009)
    Cipher Suite: TLS_ECDHE_RSA_WITH_AES_128_CBC_SHA (0xc013)
    Cipher Suite: TLS_ECDHE_RSA_WITH_AES_256_CBC_SHA (0xc014)
    Cipher Suite: TLS_DHE_RSA_WITH_AES_128_CBC_SHA (0x0033)
    Cipher Suite: TLS_DHE_RSA_WITH_AES_256_CBC_SHA (0x0039)
    Cipher Suite: TLS_RSA_WITH_AES_128_CBC_SHA (0x002f)
    Cipher Suite: TLS_RSA_WITH_AES_256_CBC_SHA (0x0035)
    Cipher Suite: TLS_EMPTY_RENEGOTIATION_INFO_SCSV (0x00ff)

```

Figure 1: Vanilla TOR Cipher Suites

Note: I have made an assumption that a TOR IP remains a TOR IP (atleast for the time the Wireshark capture is recorded).

Implementation

- Programmed in Lua.
- Wireshark has this class called Dissector which is used to decode (dissect) packets. They extract the fields, call upper layer dissectors (if needed) and present the extracted info as part of a 'tree' in the packet info. So I had to create my own dissector function that will extract necessary information and deem a packet as TOR as told in the above approach.

- To do that, I had to first create a Proto (protocol) object and then wrote the dissector. In this function, I call the dissector of TCP first (which was acquired using `DissectorTable.get()`). This will chop down the packet and make fields (of TCP and TLS (if present)) available.
- Then I check if the packet is a TLS Client Hello packet by checking if `tls.handshake.type` is 1. I acquired its value using the class `Field`. Using `Field.new(<fieldname>)`, I can get the value of any field in the packet. Value of `tls.handshake.ciphersuites` field gives the list of cipher suites used in the packet and I check if it matches with the TOR cipher suites. If it does, I add the destination IP address (got using `Field.new("ip.dst")`) to a list (of TOR IPs).
- Then in the function, I extract the source IP and destination IP addresses and check if any of them is present in the list. If yes, I mark the packet as a TOR packet (to show this I just append " (Tor!)" to the Protocol column, see figure 2).
- Finally this dissector has to be added to the `DissectorTable`. Using `.add()`, I coded that if the protocol field of the IP header is 6 (which is TCP), then call my dissector function.

2608	21.470043	10.64.127.128	213.108.108.85	TCP	66	51517 → 5353 [ACK] Seq=118325 Ack=1217020 Win=131072 Len=0 TS
2609	21.471801	10.64.127.128	213.108.108.85	TCP	602	51517 → 5353 [PSH, ACK] Seq=118325 Ack=1217020 Win=131072 Len=0
2610	21.473535	213.108.108.85	10.64.127.128	TCP	66	5353 → 51517 [ACK] Seq=1217020 Ack=118861 Win=43008 Len=0 TSv
2611	21.502736	46.165.220.229	10.64.127.128	TLSv1.3 (Tor!)	602	Application Data
2612	21.503008	10.64.127.128	46.165.220.229	TCP (Tor!)	66	51521 → 443 [ACK] Seq=18679 Ack=70566 Win=130496 Len=0 TSval=
2613	21.504504	10.64.127.128	46.165.220.229	TLSv1.3 (Tor!)	602	Application Data
2614	21.504680	10.64.120.161	10.64.255.255	NBNS	110	Registration NB MACB00KATR-1DA2<00>
2615	21.506603	46.165.220.229	10.64.127.128	TCP (Tor!)	66	443 → 51521 [ACK] Seq=70566 Ack=19215 Win=45056 Len=0 TSval=
2616	21.542280	46.165.220.229	10.64.127.128	TCP (Tor!)	1450	443 → 51521 [ACK] Seq=70566 Ack=19215 Win=45056 Len=1384 TSv
2617	21.542547	10.64.127.128	46.165.220.229	TCP (Tor!)	66	51521 → 443 [ACK] Seq=19215 Ack=71950 Win=129664 Len=0 TSval=
2618	21.543251	46.165.220.229	10.64.127.128	TCP (Tor!)	130	443 → 51521 [PSH, ACK] Seq=71950 Ack=19215 Win=45056 Len=64 TS
2619	21.543297	10.64.127.128	46.165.220.229	TCP (Tor!)	66	51521 → 443 [ACK] Seq=19215 Ack=72014 Win=131008 Len=0 TSval=
2620	21.543499	46.165.220.229	10.64.127.128	TCP (Tor!)	1450	443 → 51521 [ACK] Seq=72014 Ack=19215 Win=45056 Len=1384 TSv

Figure 2: Tor display in Wireshark

Files Submitted

- `tor_detector.lua`
- `sample.pcap`
- `report.pdf`
- `README.txt`

References

- Little help from ChatGPT, Google.
- https://www.wireshark.org/docs/wsdg_html_chunked/wsluarm_modules.html: Wireshark Lua API Reference Manual