

CS751 - Network Engineering Assignment Report

On

Emulate TLS Fingerprinting on Network Namespaces

Submitted by

Nakul Istam (212CS014)

Rohit Nainwal(212CS025)

Samyak V. Moon (212CS026)

Course Instructor

Prof. Mohit P. Tahiliani



Computer Science and Engineering
National Institute of Technology Karnataka, Surathkal.
2021-2022

Aim

This project aims at use of JA3/JA3S library to emulate TLS fingerprinting in network namespaces.

Installation

Step 1: Install iperf using following command `sudo apt-get install iperf3`

Step 2: Install openssl. Following are the steps to install OpenSSL

1. Download the latest stable version of OpenSSL using following `wget` command and unpack it using `tar` command.

```
$ wget -c https://www.openssl.org/source/openssl-1.0.2p.tar.gz
$ tar -xzf openssl-1.0.2p.tar.gz
```

2. Move into the extracted directory, configure, build, after a successful build, test the libraries and install OpenSSL in the default location, which is `/usr/local/ssl`, by running the following commands.

```
$ cd openssl-1.0.2p/
$ ./config
$ make
$ make test
$ sudo make install
```

3. Move into the installation directory and view the various sub-directories and files using `ls` command.

```
$ cd /usr/local/ssl/
$ ls -l

drwxr-xr-x. 2 root root 4096 Aug 22 06:37 bin
drwxr-xr-x. 2 root root 4096 Aug 22 06:37 certs
drwxr-xr-x. 3 root root 4096 Aug 22 06:37 include
drwxr-xr-x. 4 root root 4096 Aug 22 06:37 lib
drwxr-xr-x. 6 root root 4096 Aug 22 06:36 man
drwxr-xr-x. 2 root root 4096 Aug 22 06:37 misc
-rw-r--r--. 1 root root 10835 Aug 22 06:37 openssl.cnf
drwxr-xr-x. 2 root root 4096 Aug 22 06:37 private
```

The following are important directories you need to take note of:

- bin – contains the openssl binary and some utility scripts.
- include/openssl – contains the header files needed for building your own programs that use libcrypto or libssl.
- lib – contains the OpenSSL library files.
- lib/engines – contains the OpenSSL dynamically loadable engines.
- man – contains the OpenSSL man-pages.
- share/doc/openssl/html – contains HTML rendition of the man-pages.
- certs – the default location for certificate files.
- private – the default location for private key files.

4. To check the version of OpenSSL you have just installed, run the following command.

```
$ /usr/local/ssl/bin/openssl version  
  
OpenSSL 1.0.2p 14 Aug 2018
```

5. To use the newly installed OpenSSL version on your system, you need to add the directory /usr/local/ssl/bin/ to your PATH, in the file ~/.bashrc (or the equivalent for your shell).

```
$ vim ~/.bashrc
```

Add this line at the bottom of the file.

```
export PATH="/usr/local/ssl/bin:${PATH}"
```

Save and close the file and reload the configuration using the command below.

```
$ source .bashrc
```

6. Now open a new terminal window and run the following commands to confirm that the new OpenSSL binary is located in your PATH and that you can run it without typing its full path.

```
$ openssl version

OpenSSL 1.0.2p 14 Aug 2018
```

In case of any doubt regarding openssl installation please refer following given link: [LINK:openssl](#)

Step 3: Install JA3 To install JA3, first install pyja3 module using following command

```
pip install pyja3
```

Step 4: Inorder to run JA3S we need JA3S.py file. Inorder to get it clone the below given repository

```
https://github.com/salesforce/ja3
```

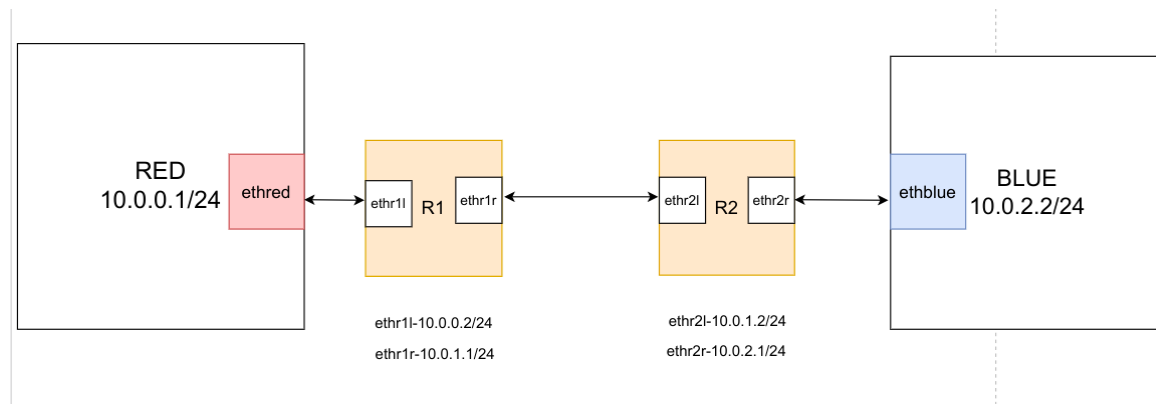
Step 5: Install WireShark. To install wireshark use following command

```
sudo apt install wireshark
```

Creating Network Namespaces

We emulate TLS fingerprinting on a topology which consist of two host and two routers. Below diagram shows the topology that we created. Note: Before creating namespace you should be in sudo mode. Use following command to get into sudo mode

```
sudo su
```



Here RED and BLUE are the two host and R1 and R2 are the two routers between RED and BLUE host. Below given script is used to create the topology.

```
#!/bin/bash
ip netns add red
ip netns add blue
ip netns add r1
ip netns add r2
ip link add ethred type veth peer name ethr1l
ip link add ethr1r type veth peer name ethr2l
ip link add ethr2r type veth peer name ethblue
ip link set ethred netns red
ip link set ethr1l netns r1
ip link set ethr1r netns r1
ip link set ethr2l netns r2
ip link set ethr2r netns r2
ip link set ethblue netns blue
ip netns exec red ip link set lo up
ip netns exec blue ip link set lo up
ip netns exec r1 ip link set lo up
ip netns exec r2 ip link set lo up
ip netns exec red ip link set ethred up
ip netns exec blue ip link set ethblue up
ip netns exec r1 ip link set ethr1l up
ip netns exec r1 ip link set ethr1r up
ip netns exec r2 ip link set ethr2l up
ip netns exec r2 ip link set ethr2r up
ip netns exec red ip address add 10.0.0.1/24 dev ethred
ip netns exec r1 ip address add 10.0.0.2/24 dev ethr1l
ip netns exec r1 ip address add 10.0.1.1/24 dev ethr1r
ip netns exec r2 ip address add 10.0.1.2/24 dev ethr2l
ip netns exec r2 ip address add 10.0.2.1/24 dev ethr2r
ip netns exec blue ip address add 10.0.2.2/24 dev ethblue
ip netns exec red ip route add default via 10.0.0.2 dev ethred
ip netns exec blue ip route add default via 10.0.2.1 dev ethblue
ip netns exec r1 ip route add default via 10.0.1.2 dev ethr1r
ip netns exec r2 ip route add default via 10.0.1.1 dev ethr2l
ip netns exec r1 sysctl -w net.ipv4.ip_forward=1
ip netns exec r2 sysctl -w net.ipv4.ip_forward=1

#for adding new host red
#for adding new host blue
#for adding router r1
#for adding router r2
#for adding virtual ethernet having endpoints names as ethred ethr1l
#for adding virtual ethernet having endpoints names as ethr1r ethr2l
#for adding virtual ethernet having endpoint names as ethr2r ethblue
#for connecting ethred interface of virtual ethernet to host red
#for connecting ethr1l interface of virtual ethernet to router r1
#for connecting ethr1r interface of virtual ethernet to router r1
#for connecting ethr2l interface of virtual ethernet to router r2
#for connecting ethr2r interface of virtual ethernet to router r2
#for connecting ethblue interface of virtual ethernet to host blue
#for setting loopback interface up mode in red host
#for setting loopback interface up mode in blue host
#for setting loopback interface up mode in r1 router
#for setting loopback interface up mode in r2 router
#for setting ethred interface up mode in red host
#for setting ethblue interface up mode in blue host
#for setting ethr1l interface up mode in r1 router
#for setting ethr1r interface up mode in r1 router
#for setting ethr2l interface up mode in r2 router
#for setting ethr2r interface up mode in r2 router
#allocation of ip address ethred
#allocation of ip address ethr1r
#allocation of ip address ethr1r
#allocation of ip address ethr2l
#allocation of ip address ethr2r
#allocation of ip address ethblue
#setting default route for ethred interface
#setting default route for ethblue interface
#setting default route for ethr1r interface
#setting default route for ethr2l interface
#enable ip forwarding at router r1
#enable ip forwarding at router r2
```

Copy the code in a file with extension .sh and run it using terminal with following command

\$./filename

(Note that here filename is written without the extension .sh)

Emulation of TLS Fingerprinting

Step1: Open two terminals. In one terminal open RED bash instance and in one terminal open BLUE bash instance using following

\$ ip netns exec red bash
\$ ip netns exec blue bash

```
root@nakul-GF63-Thin-9SC: /home/nakul/Desktop
root@nakul-GF63-Thin-9SC: /home/nakul/Desktop 105x53
root@nakul-GF63-Thin-9SC: /home/nakul/Desktop# ip netns exec red bash

root@nakul-GF63-Thin-9SC: /home/nakul
root@nakul-GF63-Thin-9SC: /home/nakul 105x53
root@nakul-GF63-Thin-9SC: /home/nakul# ip netns exec blue bash
```

Step 2: Now we will ping BLUE from RED and RED from BLUE as shown below.

```
root@nakul-GF63-Thin-9SC: /home/nakul/Desktop 105x53
root@nakul-GF63-Thin-9SC: /home/nakul/Desktop# ip netns exec red bash
root@nakul-GF63-Thin-9SC: /home/nakul/Desktop# ping 10.0.2.2
PING 10.0.2.2 (10.0.2.2) 56(84) bytes of data.
64 bytes from 10.0.2.2: icmp_seq=1 ttl=62 time=0.180 ms
64 bytes from 10.0.2.2: icmp_seq=2 ttl=62 time=0.092 ms
64 bytes from 10.0.2.2: icmp_seq=3 ttl=62 time=0.092 ms
64 bytes from 10.0.2.2: icmp_seq=4 ttl=62 time=0.092 ms
64 bytes from 10.0.2.2: icmp_seq=5 ttl=62 time=0.092 ms
64 bytes from 10.0.2.2: icmp_seq=6 ttl=62 time=0.078 ms
64 bytes from 10.0.2.2: icmp_seq=7 ttl=62 time=0.090 ms
64 bytes from 10.0.2.2: icmp_seq=8 ttl=62 time=0.089 ms
64 bytes from 10.0.2.2: icmp_seq=9 ttl=62 time=0.091 ms
64 bytes from 10.0.2.2: icmp_seq=10 ttl=62 time=0.090 ms
64 bytes from 10.0.2.2: icmp_seq=11 ttl=62 time=0.091 ms
64 bytes from 10.0.2.2: icmp_seq=12 ttl=62 time=0.092 ms
64 bytes from 10.0.2.2: icmp_seq=13 ttl=62 time=0.089 ms
64 bytes from 10.0.2.2: icmp_seq=14 ttl=62 time=0.086 ms
^C
... 10.0.2.2 ping statistics ...
14 packets transmitted, 14 received, 0% packet loss, time 13313ms
rtt min/avg/max/mdev = 0.078/0.096/0.180/0.023 ms
root@nakul-GF63-Thin-9SC: /home/nakul/Desktop#

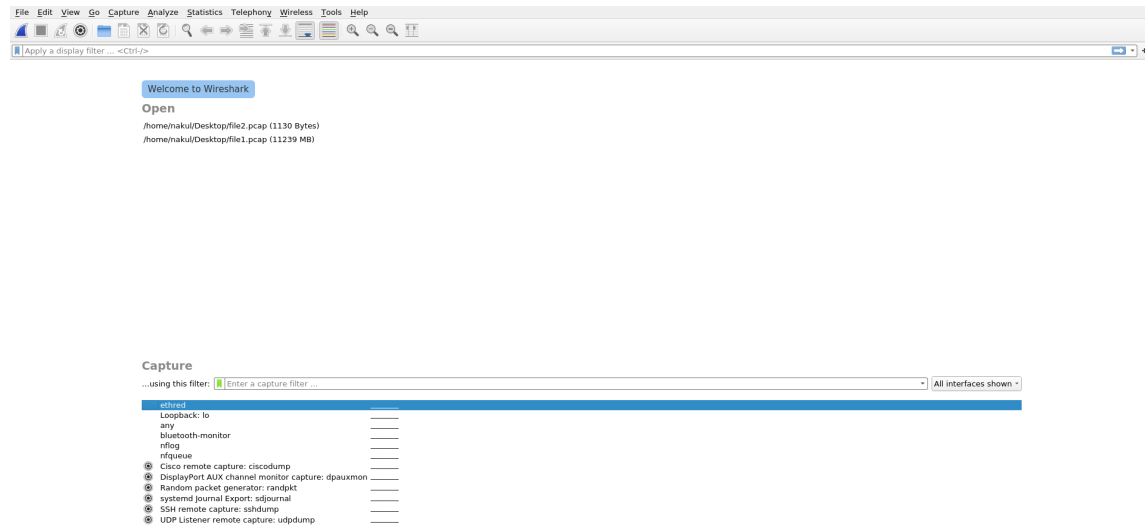
root@nakul-GF63-Thin-9SC: /home/nakul 105x53
root@nakul-GF63-Thin-9SC: /home/nakul# ip netns exec blue bash
root@nakul-GF63-Thin-9SC: /home/nakul# ping 10.0.0.1
PING 10.0.0.1 (10.0.0.1) 56(84) bytes of data.
64 bytes from 10.0.0.1: icmp_seq=1 ttl=62 time=0.111 ms
64 bytes from 10.0.0.1: icmp_seq=2 ttl=62 time=0.095 ms
64 bytes from 10.0.0.1: icmp_seq=3 ttl=62 time=0.096 ms
64 bytes from 10.0.0.1: icmp_seq=4 ttl=62 time=0.096 ms
64 bytes from 10.0.0.1: icmp_seq=5 ttl=62 time=0.095 ms
64 bytes from 10.0.0.1: icmp_seq=6 ttl=62 time=0.088 ms
64 bytes from 10.0.0.1: icmp_seq=7 ttl=62 time=0.099 ms
64 bytes from 10.0.0.1: icmp_seq=8 ttl=62 time=0.094 ms
64 bytes from 10.0.0.1: icmp_seq=9 ttl=62 time=0.095 ms
64 bytes from 10.0.0.1: icmp_seq=10 ttl=62 time=0.095 ms
64 bytes from 10.0.0.1: icmp_seq=11 ttl=62 time=0.087 ms
64 bytes from 10.0.0.1: icmp_seq=12 ttl=62 time=0.097 ms
64 bytes from 10.0.0.1: icmp_seq=13 ttl=62 time=0.096 ms
64 bytes from 10.0.0.1: icmp_seq=14 ttl=62 time=0.094 ms
^C
... 10.0.0.1 ping statistics ...
14 packets transmitted, 14 received, 0% packet loss, time 13295ms
rtt min/avg/max/mdev = 0.087/0.095/0.111/0.005 ms
root@nakul-GF63-Thin-9SC: /home/nakul#
```

Step 3: We open wireshark on RED Bash instance.

```
root@nakul-GF63-Thin-9SC: /home/nakul/Desktop 105x53
root@nakul-GF63-Thin-9SC: /home/nakul/Desktop# wireshark
QStandardPaths: XDG_RUNTIME_DIR not set, defaulting to '/tmp/runtime-root'
```

Step 4: In wireshark, choose RED interface(ethred) so that we can capture packets that are coming

to RED bash instance and going to BLUE bash instance.



Step 5: We created server on BLUE BASH instance on port 443 using following command

```
$ iperf3 -s -p 443
root@nakul-GF63-Thin-95C: /home/nakul 105x53
root@nakul-GF63-Thin-95C: /home/nakul# iperf3 -s -p 443
Server listening on 443
```

Note: We created server on port 443 because it is standard port of SSL.

Step 6: Now From RED Bash we are connecting to BLUE bash (which is our server) using OpenSSL. Below is the command

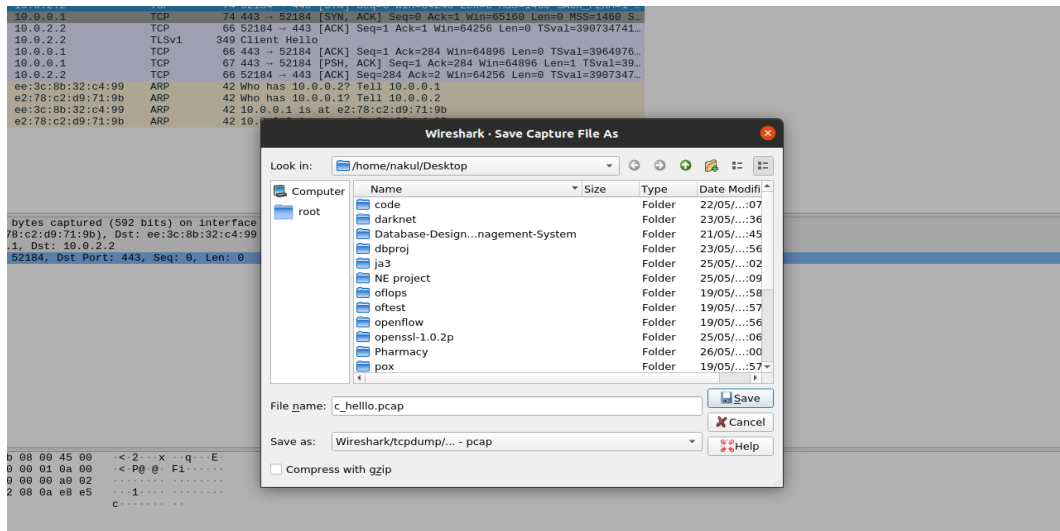
```
$ openssl s_client -connect 10.0.2.2:443
root@nakul-GF63-Thin-95C: /home/nakul 105x53
root@nakul-GF63-Thin-95C: /home/nakul# openssl s_client -connect 10.0.2.2:443
CONNECTED(00000003)
```

Step 7: Now as both RED and BLUE bash are connected via TCP handshake. After that TLS handshake will be done and during this period client hello packet will be transferred from client to server. We will capture this packet in wireshark on ethred namespace.

Vo.	Time	Source	Destination	Protocol	Length	Info
1	0.000000000	10.0.0.1	10.0.2.2	TCP	74	52184 → 443 [SYN] Seq=0 Win=64240 Len=0 MSS=1460 SACK_PERM=1 ...
2	0.000024024	10.0.2.2	10.0.0.1	TCP	74	443 → 52184 [SYN, ACK] Seq=0 Ack=1 Win=65160 Len=0 MSS=1460 S...
3	0.000030069	10.0.0.1	10.0.2.2	TCP	66	52184 → 443 [ACK] Seq=1 Ack=1 Win=64256 Len=0 TSval=390734741...
4	0.000212399	10.0.0.1	10.0.2.2	TLSv1	349	Client Hello
5	0.000221143	10.0.2.2	10.0.0.1	TCP	66	443 → 52184 [ACK] Seq=1 Ack=284 Win=64896 Len=0 TSval=3964976...
6	0.000259588	10.0.2.2	10.0.0.1	TCP	67	443 → 52184 [PSH, ACK] Seq=1 Ack=284 Win=64896 Len=1 TSval=39...
7	0.000276798	10.0.0.1	10.0.2.2	TCP	66	52184 → 443 [ACK] Seq=284 Ack=2 Win=64256 Len=0 TSval=3907347...
8	5.175891619	e2:78:c2:d9:71:9b	ee:3c:8b:32:c4:99	ARP	42	Who has 10.0.0.2? Tell 10.0.0.1
9	5.175849894	ee:3c:8b:32:c4:99	e2:78:c2:d9:71:9b	ARP	42	Who has 10.0.0.1? Tell 10.0.0.2
10	5.175921162	e2:78:c2:d9:71:9b	ee:3c:8b:32:c4:99	ARP	42	10.0.0.1 is at e2:78:c2:d9:71:9b
11	5.175941544	ee:3c:8b:32:c4:99	e2:78:c2:d9:71:9b	ARP	42	10.0.0.2 is at ee:3c:8b:32:c4:99

▶ Frame 1: 74 bytes on wire (592 bits), 74 bytes captured (592 bits) on interface ethred, id 0
 ▶ Ethernet II, Src: e2:78:c2:d9:71:9b (e2:78:c2:d9:71:9b), Dst: ee:3c:8b:32:c4:99 (ee:3c:8b:32:c4:99)
 ▶ Internet Protocol Version 4, Src: 10.0.0.1, Dst: 10.0.2.2
 ▶ Transmission Control Protocol, Src Port: 52184, Dst Port: 443, Seq: 0, Len: 0

Step 8: Save the captured packet in a .pcap file.



JA3 is a fingerprinting mechanism performed on a Client that uses TLS to connect with the Server. This is done by performing a series of operations on the ClientHello packet received in the first step of the TLS Negotiation processes.

If you want to see how JA3 is calculated please refer the link: [LINK:JA3](#)

Step 9: Now using JA3 we calculate TLS fingerprint. Below is the command for running ja3 on .pcap file.

```
$ ja3 filename.pcap
```

Note: We can give only .pcap file to ja3.



```
nakul@nakul-GF63-Thin-9SC: ~/Desktop
nakul@nakul-GF63-Thin-9SC: ~/Desktop 80x24
nakul@nakul-GF63-Thin-9SC:~/Desktop$ ja3 c_hello.pcap
[
  {
    "destination_ip": "10.0.2.2",
    "destination_port": 443,
    "ja3": "771,4866-4867-4865-49196-49200-159-52393-52392-52394-49195-49199-158-49188-49192-107-49187-49191-103-49162-49172-57-49161-49171-51-157-156-61-60-53-47-255,11-10-35-22-23-13-43-45-51,29-23-30-25-24,0-1-2",
    "ja3_digest": "c34a54599a1fbaf1786aa6d633545a60",
    "source_ip": "10.0.0.1",
    "source_port": 52184,
    "timestamp": 1653747718.843705
  }
]
nakul@nakul-GF63-Thin-9SC:~/Desktop$
```

Step 10: Inorder to Calculate JA3S we are going to start simple https server on BLUE bash. We need to create two certificates which will be used by the OpenSSL s.server command.

```
$ openssl req -x509 -newkey rsa:2048 -keyout key.pem -out cert.pem -days 365 -nodes
```

```

root@nakul-GF63-Thin-9SC:/home/nakul 105x53
root@nakul-GF63-Thin-9SC:/home/nakul# openssl req -x509 -newkey rsa:2048 -keyout key.pem -out cert.pem -days 365 -nodes
Generating a RSA private key
.....+++++
.....+++++
writing new private key to 'key.pem'
-----
You are about to be asked to enter information that will be incorporated
into your certificate request.
What you are about to enter is what is called a Distinguished Name or a DN.
There are quite a few fields but you can leave some blank
For some fields there will be a default value,
If you enter '.', the field will be left blank.
-----
Country Name (2 letter code) [AU]:

```

Step 11: Starting the openssl s_server.

```

root@nakul-GF63-Thin-9SC:/home/nakul# openssl s_server -key key.pem -cert cert.pem -accept 443 -www 10.0.2.2
Using default temp DH parameters
ACCEPT

```

Step 12: Now we connected OpenSSL s_server via RED Bash.

```

$ openssl s_client -connect 10.0.2.2:443
root@nakul-GF63-Thin-9SC:/home/nakul 105x53
root@nakul-GF63-Thin-9SC:/home/nakul# openssl s_client -connect 10.0.2.2:443
CONNECTED(00000003)

```

Step 13: These are the server details that we got on BLUE bash.

```

root@nakul-GF63-Thin-9SC:/home/nakul# openssl req -x509 -newkey rsa:2048 -keyout key.pem -out cert.pem -days 365 -nodes
Generating a RSA private key
.....+++++
writing new private key to 'key.pem'
-----
You are about to be asked to enter information that will be incorporated
into your certificate request.
What you are about to enter is what is called a Distinguished Name or a DN.
There are quite a few fields but you can leave some blank
For some fields there will be a default value,
If you enter '.', the field will be left blank.
-----
Country Name (2 letter code) [AU]:IN
State or Province Name (full name) [Some-State]:KARNATAKA
Locality Name (eg, city) []:MANGALORE
Organization Name (eg, company) [Internet Widgits Pty Ltd]:NITK
Organizational Unit Name (eg, section) []:EDU
Common Name (e.g. server FQDN or YOUR name) []:NIT
Email Address []:rohit@gmail.com

```

Step 14: By using wireshark we are able to capture client hello and server hello. save it as a .pcap file.

The image shows a Wireshark capture of network traffic on interface ethred. The packet list shows several TCP and TLSv1.3 packets. The selected packet (No. 14) is a TLSv1.3 client hello from 10.0.0.1 to 10.0.0.1. The packet details pane shows the following information:

- Frame 1: 74 bytes on wire (592 bits), 74 bytes captured (592 bits) on interface ethred, id 0
- Ethernet II, Src: e2:78:c2:d9:71:9b (e2:78:c2:d9:71:9b), Dst: ee:3c:8b:32:c4:99 (ee:3c:8b:32:c4:99)
- Internet Protocol Version 4, Src: 10.0.0.1, Dst: 10.0.0.2
- Transmission Control Protocol, Src Port: 52186, Dst Port: 443, Seq: 0, Len: 0

Step 15: First we have to go inside the ja3 folder which we clone from git <https://github.com/salesforce/ja3>. git Now by using ja3.py python script we calculate client hello fingerprint and using ja3s.py python script we calculate server hello fingerprint from same .pcap file.

```

nakul@nakul-GF63-Thin-9SC:~/Desktop/ja3/python$ python ja3s.py both.hello.pcap
/home/nakul/.local/lib/python3.8/site-packages/dpkt/ssl.py:350: UserWarning: TLSv1ServerHello.cipher_suite is deprecated and renamed to .cipher_suite
deprecation_warning("TLSv1ServerHello.cipher_suite is deprecated and renamed to .cipher_suite")
[10.0.0.1:52186] JA3S: 771,ciphersuite(0x1302,TLS_AES_256_GCM_SHA384),43-51 --> 451d535491f80bfc2f54456317f5e20
nakul@nakul-GF63-Thin-9SC:~/Desktop/ja3/python$ python ja3.py both.hello.pcap
[10.0.0.2:443] JA3: 771,4866-4867-4865-49196-49200-159-52393-52392-52394-49195-49199-158-49188-49192-107-49187-49191-103-49162-49172-57-49161-49171-51-157-156-61-60-53-47-255,11-10-35-22-23-13-43-45-51,29-23-30-25-24,0-1-2 --> c34a54599a1fba7f786aa6633545a60
nakul@nakul-GF63-Thin-9SC:~/Desktop/ja3/python$

```

1 TLS Fingerprinting using Mercury

step 1:For Installing Mercury first we have to clone git link <https://github.com/cisco/mercury.git>

step 2:Building and installing mercury .In the root directory, run

```
./configure
make
```

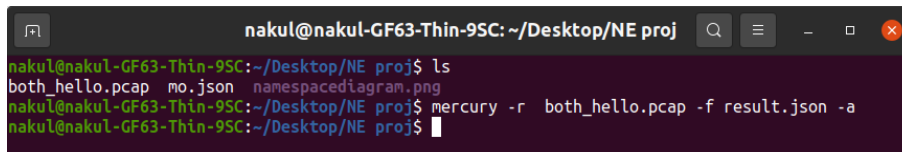
step 3:We will use the same steps as mentioned in TLS method to create a openssl server in blue host and connecting to it from red host and for capturing packets at red interface we used inbuilt packet capture of mercury.

```
root@nakul-GF63-Thin-9SC: /home/nakul/Desktop 104x25
root@nakul-GF63-Thin-9SC:/home/nakul/Desktop# mercury -c ethred -w fingerprint.pcap
mem: 166134784  frac: 0.010000
Notice: requested memory 165675008 will be less than desired memory 166134784
Requesting PACKET_RX_RING with 165675008 bytes (79 blocks of size 2097152) for thread 0
dropped root privileges
Thread 0 with thread id 140588331218688 started...
^C
shutting down: Interrupt
Thread 0 with thread id 140588331218688 exiting...
--
11 packets captured
930 bytes captured
11 packets seen by socket
0 packets dropped
0 socket queue freezes
root@nakul-GF63-Thin-9SC:/home/nakul/Desktop#
```

step 4:By using these command we calculate tls fingerprint using mercury

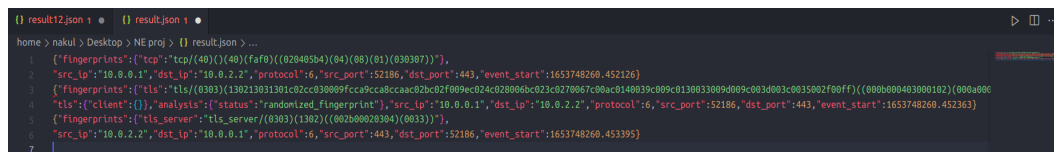
```
nakul@nakul-GF63-Thin-9SC: ~/Desktop
nakul@nakul-GF63-Thin-9SC:~/Desktop$ mercury -r fingerprint.pcap -f result12.json -a
nakul@nakul-GF63-Thin-9SC:~/Desktop$
```

step 5: By using these command we can calculate fingerprint in .json file using mercury.



```
nakul@nakul-GF63-Thin-9SC: ~/Desktop/NE proj
nakul@nakul-GF63-Thin-9SC:~/Desktop/NE proj$ ls
both_hello.pcap  mo.json  namespacediagram.png
nakul@nakul-GF63-Thin-9SC:~/Desktop/NE proj$ mercury -r both_hello.pcap -f result.json -a
nakul@nakul-GF63-Thin-9SC:~/Desktop/NE proj$
```

step 6: These is result of tls client hello and tls server hello fingerprint using mercury



```
{
  "fingerprints": [
    {
      "src_ip": "10.0.0.1", "dst_ip": "10.0.2.2", "protocol": "6", "src_port": "52186", "dst_port": "443", "event_start": "1653748260.452126"
    },
    {
      "fingerprints": [
        {
          "tls": {
            "client": {
              "analysis": {
                "status": "randomized_fingerprint",
                "src_ip": "10.0.0.1", "dst_ip": "10.0.2.2", "protocol": "6", "src_port": "52186", "dst_port": "443", "event_start": "1653748260.452126"
              }
            }
          }
        }
      ]
    },
    {
      "fingerprints": [
        {
          "tls_server": {
            "analysis": {
              "status": "randomized_fingerprint",
              "src_ip": "10.0.2.2", "dst_ip": "10.0.0.1", "protocol": "6", "src_port": "443", "dst_port": "52186", "event_start": "1653748260.453395"
            }
          }
        }
      ]
    }
  ]
}
```

these are few command useful for fingerprinting

```
mercury -c eth0 -w foo.pcap          # capture from eth0, write to foo.pcap
mercury -c eth0 -w foo.pcap -t cpu   # as above, with one thread per CPU
mercury -c eth0 -w foo.mcap -t cpu -s # as above, selecting packet metadata
mercury -r foo.mcap -f foo.json      # read foo.mcap, write fingerprints
mercury -r foo.mcap -f foo.json -a   # as above, with fingerprint analysis
mercury -c eth0 -t cpu -f foo.json -a # capture and analyze fingerprints
```

Reference Links

1. <https://infosecwriteups.com/demystifying-ja3-one-handshake-at-a-time-c80b04ccb393>
2. <https://github.com/salesforce/ja3.git>
3. <https://hackernoon.com/ja3-and-ja3s-in-security-monitoring-of-ssl-communication-6e1w348s>
4. <https://www.tecmint.com/install-openssl-from-source-in-centos-ubuntu/>
5. https://superhero.ninja/2015/07/22/create-a-simple-https-server-with-openssl-s_server/
5. <https://github.com/cisco/mercury.git/>