# SMBLoris Attack Data Analysis

Ilnaz Nikseresht, Mahdi Rangchi, Laleh Ebdali Takalloo, Behnaz Saropourian, Somayeh Roshandel
*Department of Electrical and Computer Engineering, University of Victoria, Victoria, Canada*
*ilnaznikseresht@uvic.ca*, *rangchi@uvic.ca*, *laleht@uvic.ca*, *behnazsaropourian@uvic.ca*, *somayehr@uvic.ca*

**Abstract**—SMBLoris is a type of denial of service(DOS) attack which attempts to utilize the victim's memory space by sending it several TCP connection requests. The data interpretation and analysis has been conducted in order to extract this type of attack's patterns and unique characteristics that can be utilized in order to predict future similar attacks in the network.

**Keywords**— SMBLoris Attack, Denial of Service, Data Analytics

## I. Introduction

The focus of conducted research is on the SMBLoris attack which is an application-level denial of service that lies where SMB packets are processed and by initiation of several TCP connections targeting the critical services of victim, it attempts to consume all its available memory and cause an operating system to freeze and finally crash. It was discovered by two RiskSense security researchers Sean Dillon and Jenna Magius and took its name from Slowloris attack on Webserver back in 2009.[4] It is considerable that SMBLoris functions same as SlowLoris except that unlike its counterpart which targets the web services, it is performed via SMB and functions based on NetBios Session Service (NBSS) filling. Since each connection to NBSS allocates 128KB of memory that is released only when the connection is closed(after 30 seconds of no activity), in case the attacker uses its 65535 available TCP ports, more than 8GB is filled up and the NBSS memory saturation occurs and a manual server reboot would be required. Not only after having the connection opened for 30 seconds, the kernel gives up, but also during the 30 seconds, the allocated percentage of memory is useless for every other connection attempt. It is considerable that the allocated memory can not be swapped with that of the disk due to the fact that it's in the physical RAM. The NetBIOS service on port 139 is also exploitable but below report is based on port 445 which can be expanded to port 139 as well.

The scope of this project is to simulate the SMBLoris attack scenario by the help of a range of tools such as Kali Linux and Wireshark in order to generate the raw dataset. In addition, we use a variety of tools for data interpretation(i.e, CICFlowMeter) and data analysis(i.e, Jupyter Notebook) of the conducted attack.

Regarding the encountered limitations, it is noticeable that the represented data in the wireshark output's format is not useful in the analysis as all the data is presented in the Info column which is not distinct. Hence, in order to mitigate that issue, CICFlowMeter is used instead. However, it also does not present the clean data set. Although it is tried to clean the data, it can not be as efficient as that of the prepared data sets.

## II. Discussion

In order to simulate the SMBLoris attack, two machines were set up in Kali Linux, one performing as attacker and the other one as victim. Also, we have used a windows machine in order to generate normal Samba traffic before the attack starts, including browsing the shared files. The number of supported TCP connections on the victim has been increased to 65535 and the respective victim's port on which the attack is employed is 445. Before initiating the attack, Wireshark has been employed to capture the pre-attack packets which gives the ability to compare the normal traffic packets and that of the attack traffic. The generated Wireshark output lacks the proper formatting which is replaced by the output of the CICFlowMeter tool that attempts to organize data in a more proper format for data interpretation and analysis. Next, the data interpretation and analysis has been commenced by the help of CICFlowMeter output and Jupyter Notebook, respectively and the results are presented in the data analytics section.

### A. Security/Privacy Data Collection

As discussed before, SMBLoris is a remote, unauthenticated application-level denial of service (DoS) attack against Microsoft Windows operating systems which is caused by a very old memory-handling bug in the Server Message Block (SMB) network protocol implementation. The vulnerability lies in the way SMB packets are processed and memory is allocated. The attack is simulated to gather the dataset required for the analysis. In the attack scenario, the windows IP used to generate the normal pre-attack traffic is 192.168.254.1, attacker IP is 192.168.254.3 and that of the victim is 192.168.254.4. Pre-attack traffic has also been logged in order to have a glimpse of the normal traffic data and be able to later extract the attack traffic pattern from the raw dataset. Normal pre-attack traffic generation was done using the ping and browsing the samba server shared files from the windows and was captured by the Wireshark as depicted in figures 1 and 2.

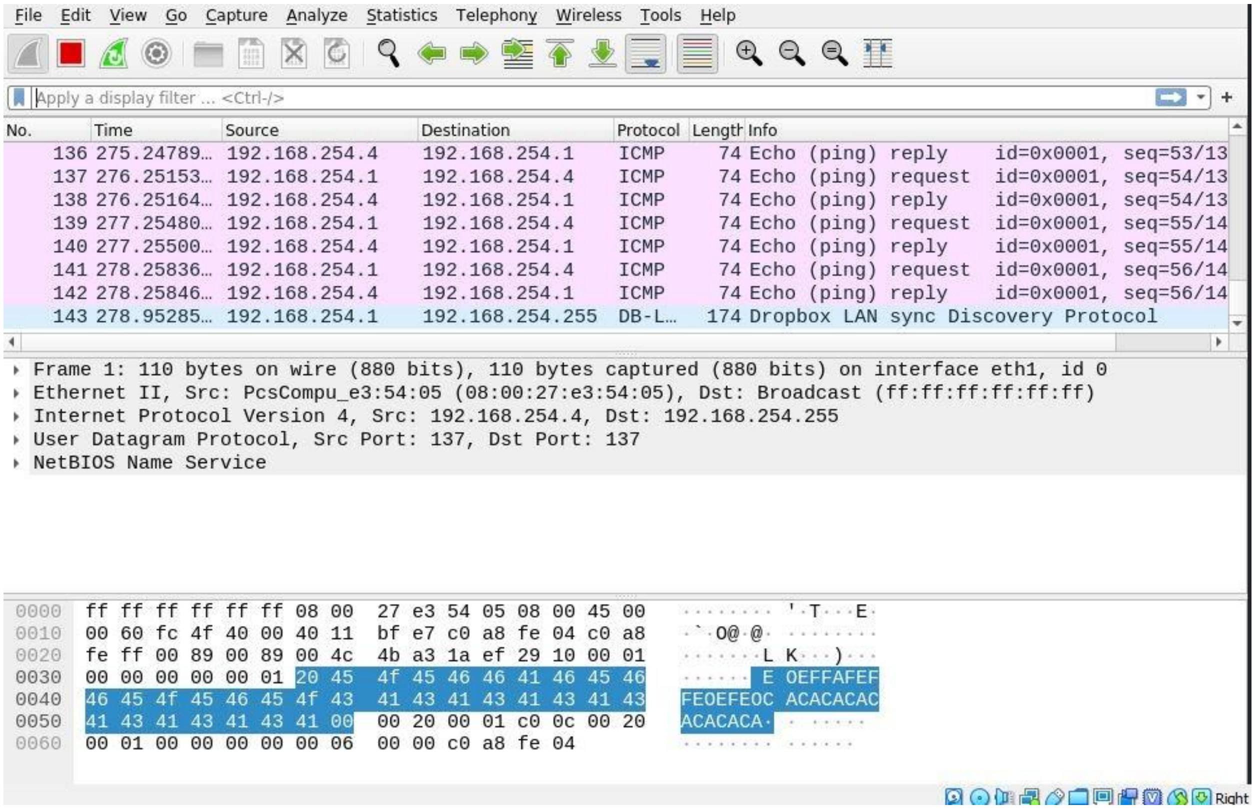



Fig.1.Pre-attack traffic generation by Ping

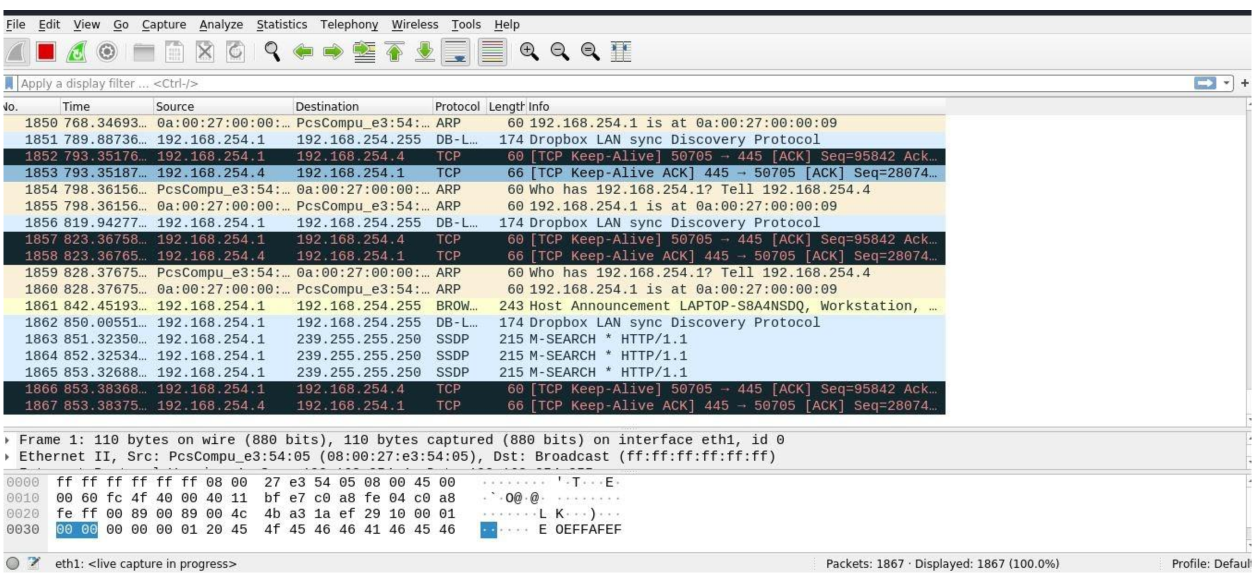



Fig.2. Pre-attack traffic generation by browsing

In order to simulate the SMBLoris attack, the Metasploit module in Kali linux was used on the attacker machine to set the victim's settings. Figure 3 represents the metasploit module settings on the target machine in Kali Linux. The settings include the remote host IP and remote host port. In addition, the number of open sockets has increased to 65535 to be able to compare the effect.

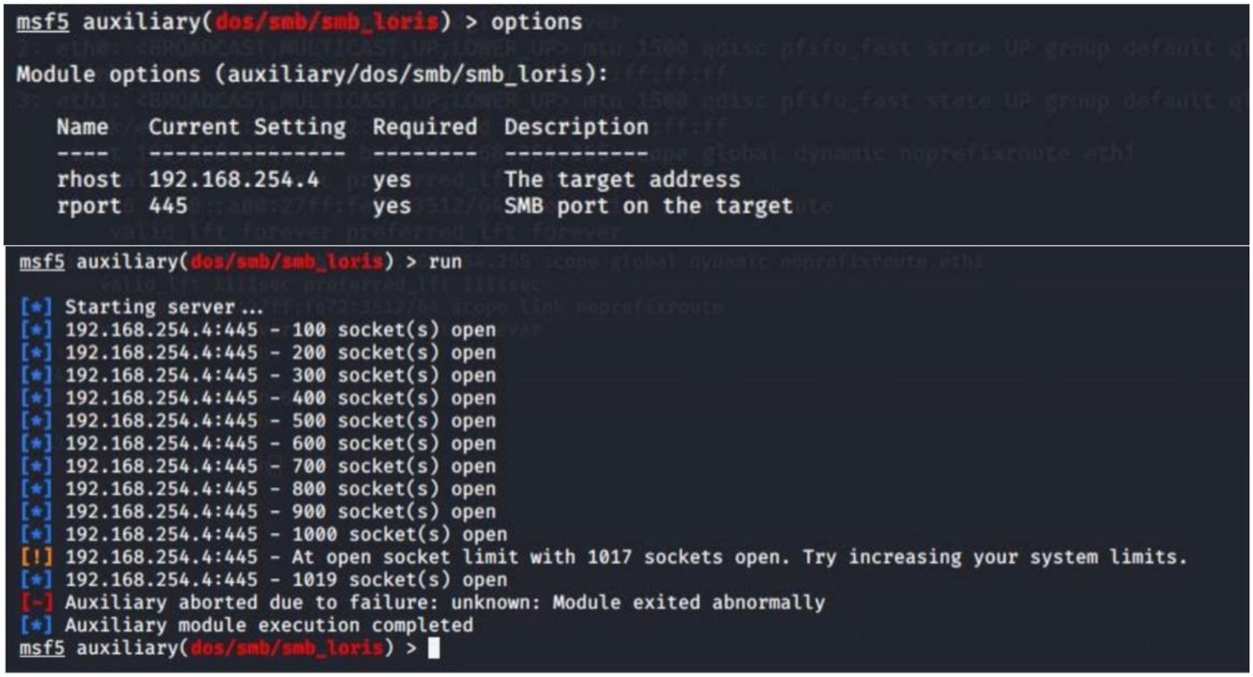
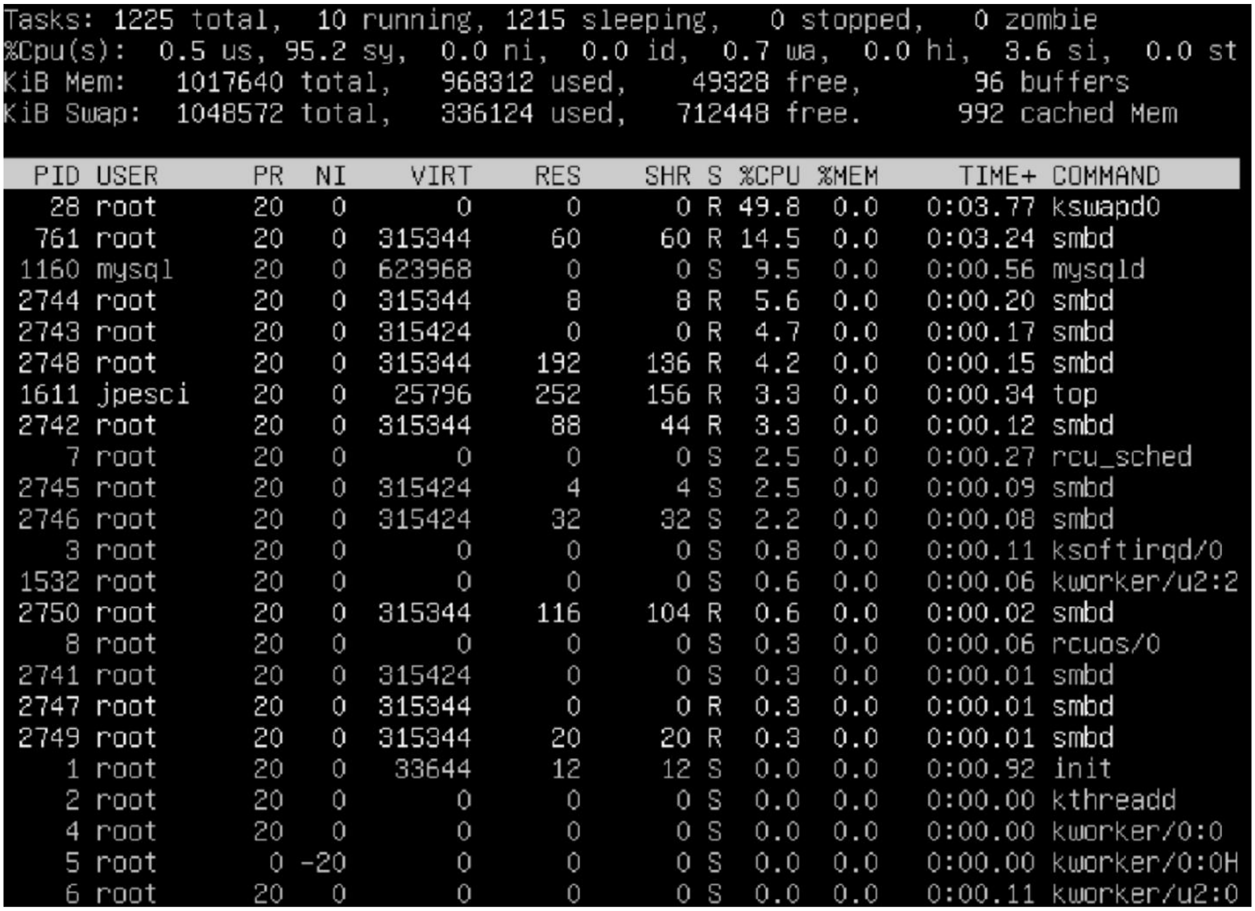



Fig.3. SMBLoris parameter settings in Kali

We have also logged the alarms showing up in the victim's machine during the attack with open sockets default value(i.e 1024) and 65535 which is the maximum number of open sockets. As can be observed in figure 5, with the initial settings, the victim computer has encountered no issue and has enough memory to operate but as the number of open sockets are increased to 65535(figure 6), the victim's memory is used up and it no longer has enough memory to accept new TCP connections and as a result it becomes unresponsive and the victim's system needs a reboot before becoming operational again.



```
Tasks: 1225 total,  10 running, 1215 sleeping,   0 stopped,   0 zombie
%Cpu(s):  0.5 us, 95.2 sy,  0.0 ni,  0.0 id,  0.7 wa,  0.0 hi,  3.6 si,  0.0 st
KiB Mem:   1017640 total,   968312 used,    49328 free,       96 buffers
KiB Swap:  1048572 total,   336124 used,   712448 free.      992 cached Mem

  PID USER      PR  NI    VIRT    RES    SHR S %CPU %MEM     TIME+ COMMAND
   28 root      20   0       0      0      0 R 49.8  0.0   0:03.77 kswapd0
  761 root      20   0  315344     60     60 R 14.5  0.0   0:03.24 smbd
 1160 mysql     20   0  623968      0      0 S  9.5  0.0   0:00.56 mysqld
 2744 root      20   0  315344      8      8 R  5.6  0.0   0:00.20 smbd
 2743 root      20   0  315424      0      0 R  4.7  0.0   0:00.17 smbd
 2748 root      20   0  315344    192    136 R  4.2  0.0   0:00.15 smbd
 1611 jpesci    20   0   25796    252    156 R  3.3  0.0   0:00.34 top
 2742 root      20   0  315344     88     44 R  3.3  0.0   0:00.12 smbd
    7 root      20   0       0      0      0 S  2.5  0.0   0:00.27 rcu_sched
 2745 root      20   0  315424      4      4 S  2.5  0.0   0:00.09 smbd
 2746 root      20   0  315424     32     32 S  2.2  0.0   0:00.08 smbd
    3 root      20   0       0      0      0 S  0.8  0.0   0:00.11 ksoftirqd/0
 1532 root      20   0       0      0      0 S  0.6  0.0   0:00.06 kworker/u2:2
 2750 root      20   0  315344    116    104 R  0.6  0.0   0:00.02 smbd
    8 root      20   0       0      0      0 S  0.3  0.0   0:00.06 rcuos/0
 2741 root      20   0  315424      0      0 S  0.3  0.0   0:00.01 smbd
 2747 root      20   0  315344      0      0 R  0.3  0.0   0:00.01 smbd
 2749 root      20   0  315344     20     20 R  0.3  0.0   0:00.01 smbd
    1 root      20   0   33644     12     12 S  0.0  0.0   0:00.92 init
    2 root      20   0       0      0      0 S  0.0  0.0   0:00.00 kthreadd
    4 root      20   0       0      0      0 S  0.0  0.0   0:00.00 kworker/0:0
    5 root       0 -20       0      0      0 S  0.0  0.0   0:00.00 kworker/0:0H
    6 root      20   0       0      0      0 S  0.0  0.0   0:00.11 kworker/u2:0
```

Fig.5. Victim processes during the attack with 1024 open sockets

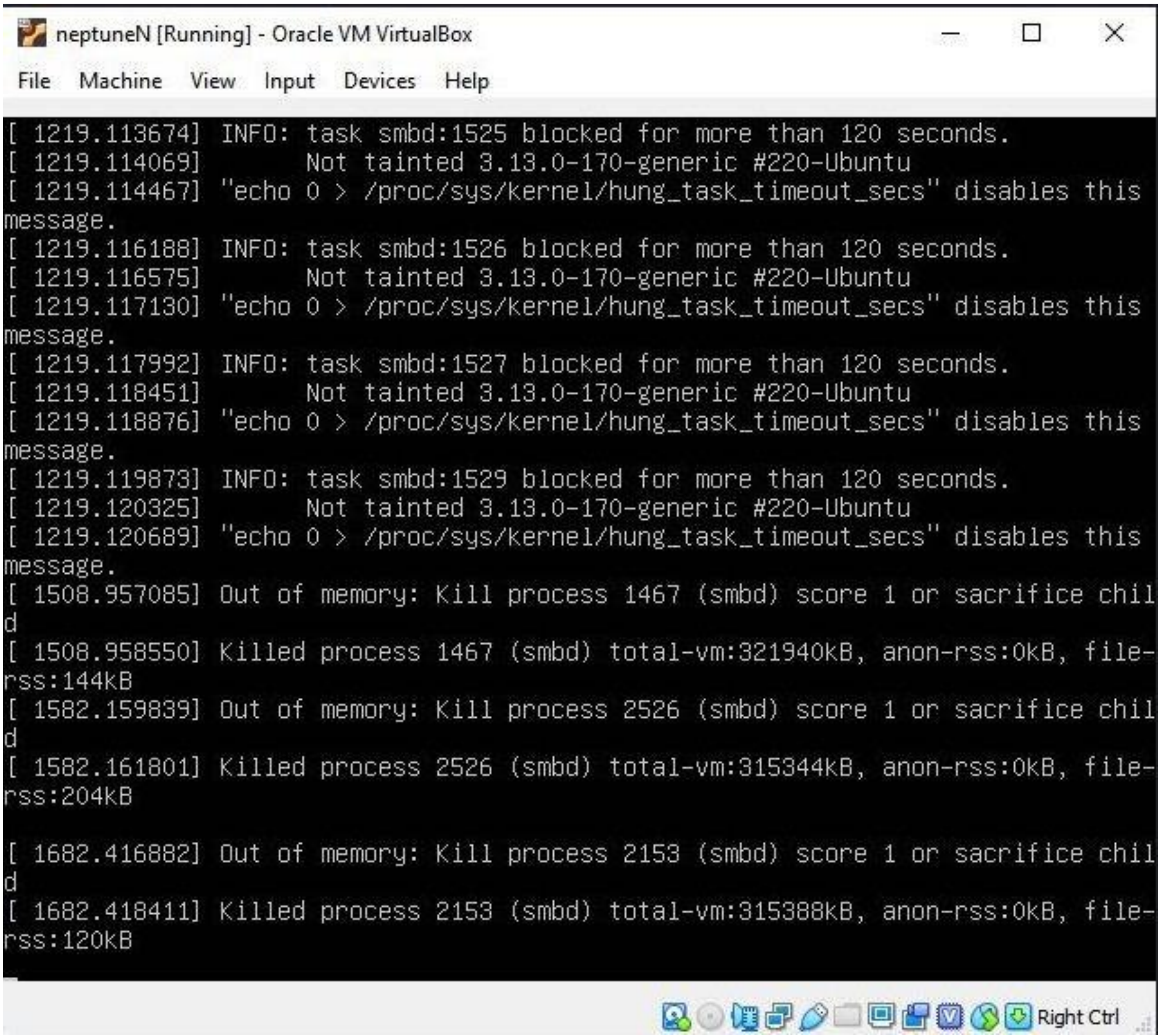



Fig.6. Victim processes during the attack with 65535 open sockets

### B. Security/Privacy Data Interpretation

In order to extract the relevant pattern to the conducted attack from the wireshark logs, below analysis is done. Based on the captured Wireshark log, the normal and attack traffic have been compared to get a glimpse of the attack patterns. It is known that the TCP port 445 is used for direct TCP/IP networking access without the need for the NETBIOS layer. As the SMBLoris is a form of DOS attack, it operates based on the number of TCP connections on port 445. As per figure 7, a normal TCP Connection is where we get the Push/Ack where the Push flag means the target network push the data directly to the receiving socket and not wait for any more packets and Fin/Ack where the Fin flag means that the sender finished talking to receiver but it will also listen to anything it has to say until it is done which is waiting for an ACK. This is when the number of concurrent open sockets is low and attack is not initiated . However, after running the exploit, the TCP flow changes to what is shown in figure 8 where there is a packet named as RST where the receiver states that there is no conversation and it is resetting the connection which depends on the number of open sockets and initiated TCP connections. Figure 9 is another capture from the transmitted packets during attack. The high number of retransmissions happens when the SYN packet is not acknowledged by the receiver, and the Kali/sender sends a TCP Retransmission. On the other hand, it confirms the attack scenario that the victim does not have enough memory resources to handle new TCP connections and hence do not respond to the already sent SYN packets. Hence, there will be TCP Retransmission packets sent from the attacker to the victim's machine. According to the general pattern of SMBLoris DOS attack, where there are many, say 200 for instance, SYN/ACK or FIN/ACK packets sent from a computer to a destination in a short period of time, say for instance, 10 seconds, there is a suspicious activity going on.

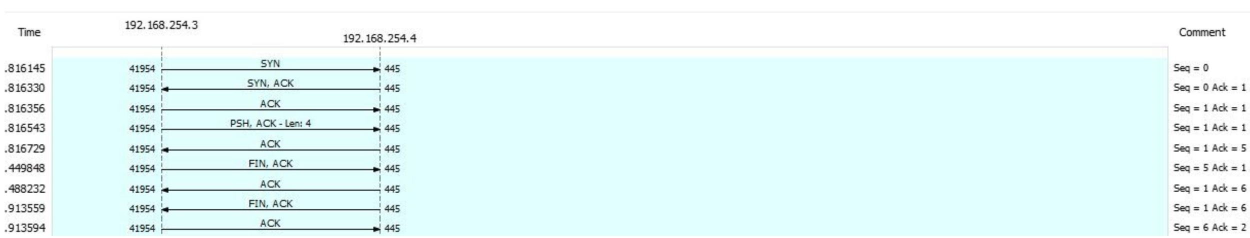



Fig.7. Wireshark capture of the transmitted packets before attack

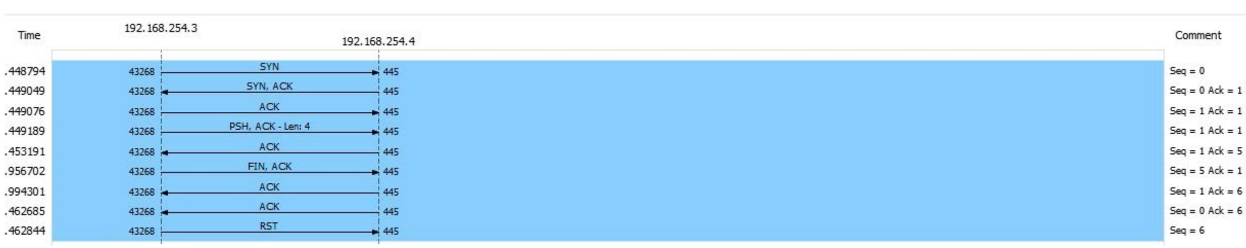



Fig.8. Wireshark capture of the transmitted packets during attack

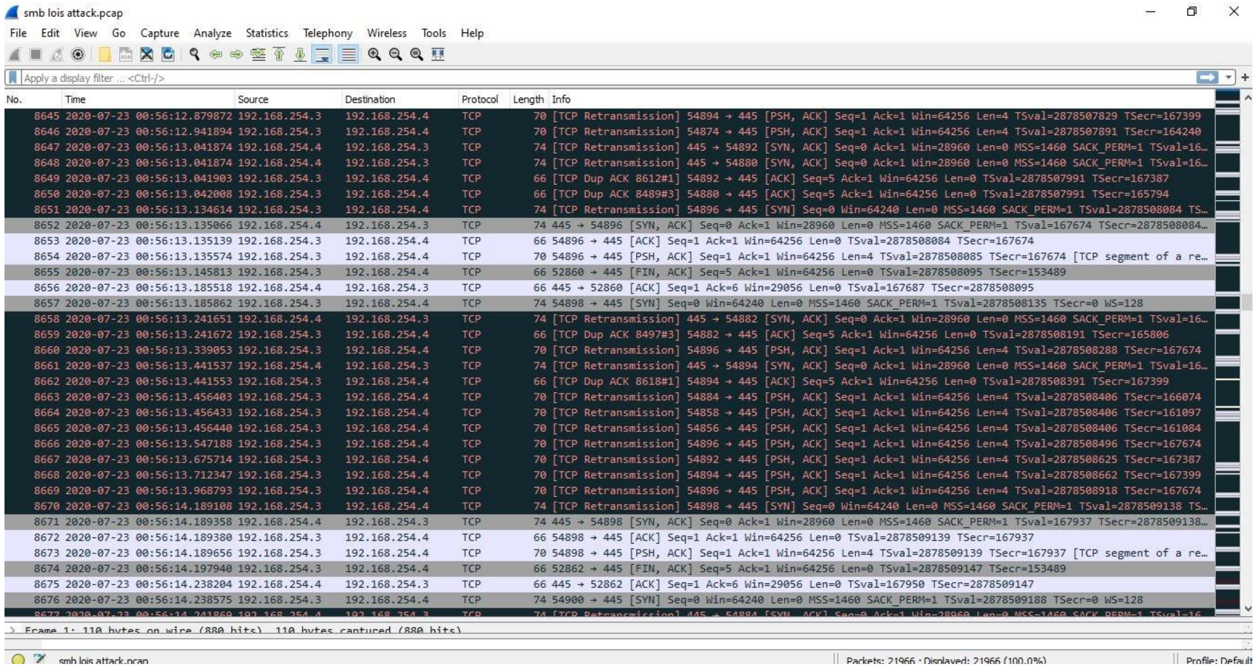


Fig.9. Wireshark capture of the transmitted packets during attack

Hence, so far we have interpreted the patterns of the wireshark log before and after the attack. Next, the outputs of the conducted attack in the Kali linux machine which was captured by the Wireshark and were in the form of .pcap and .csv, were analyzed. As can be observed in figure 10, .csv log, we have detailed information regarding the time(in nanoseconds), source/destination IP address and the transmitted signaling and data per connection.

| No. | Time | Source | Destinatio | Protocol | Length | Info |
|---|---|---|---|---|---|---|
| 1 | 0 | 192.168.56 | 192.168.56 | TCP | 74 | 59916 > 445 [SYN] Seq=0 Win=64240 Len=0 MSS=1460 SACK_PERM=1 TSval=347780005 TSecr=0 WS=128 |
| 2 | 0.000424 | 192.168.56 | 192.168.56 | TCP | 74 | 445 > 59916 [SYN, ACK] Seq=0 Ack=1 Win=28960 Len=0 MSS=1460 SACK_PERM=1 TSval=71753 TSecr=347780005 WS=128 |
| 3 | 0.00045 | 192.168.56 | 192.168.56 | TCP | 66 | 59916 > 445 [ACK] Seq=1 Ack=1 Win=64256 Len=0 TSval=347780006 TSecr=71753 |
| 4 | 0.000716 | 192.168.56 | 192.168.56 | TCP | 70 | 59916 > 445 [PSH, ACK] Seq=1 Ack=1 Win=64256 Len=4 TSval=347780006 TSecr=71753 [TCP segment of a reassembled PDU] |
| 5 | 0.000776 | 192.168.56 | 192.168.56 | TCP | 74 | 59918 > 445 [SYN] Seq=0 Win=64240 Len=0 MSS=1460 SACK_PERM=1 TSval=347780006 TSecr=0 WS=128 |
| 6 | 0.000974 | 192.168.56 | 192.168.56 | TCP | 66 | 445 > 59916 [ACK] Seq=1 Ack=5 Win=29056 Len=0 TSval=71753 TSecr=347780006 |
| 7 | 0.000974 | 192.168.56 | 192.168.56 | TCP | 74 | 445 > 59918 [SYN, ACK] Seq=0 Ack=1 Win=28960 Len=0 MSS=1460 SACK_PERM=1 TSval=71753 TSecr=347780006 WS=128 |
| 8 | 0.000996 | 192.168.56 | 192.168.56 | TCP | 66 | 59918 > 445 [ACK] Seq=1 Ack=1 Win=64256 Len=0 TSval=347780006 TSecr=71753 |
| 9 | 0.001148 | 192.168.56 | 192.168.56 | TCP | 70 | 59918 > 445 [PSH, ACK] Seq=1 Ack=1 Win=64256 Len=4 TSval=347780006 TSecr=71753 [TCP segment of a reassembled PDU] |
| 10 | 0.001228 | 192.168.56 | 192.168.56 | TCP | 74 | 59920 > 445 [SYN] Seq=0 Win=64240 Len=0 MSS=1460 SACK_PERM=1 TSval=347780006 TSecr=0 WS=128 |
| 11 | 0.001485 | 192.168.56 | 192.168.56 | TCP | 66 | 445 > 59918 [ACK] Seq=1 Ack=5 Win=29056 Len=0 TSval=71753 TSecr=347780006 |
| 12 | 0.001485 | 192.168.56 | 192.168.56 | TCP | 74 | 445 > 59920 [SYN, ACK] Seq=0 Ack=1 Win=28960 Len=0 MSS=1460 SACK_PERM=1 TSval=71753 TSecr=347780006 WS=128 |
| 13 | 0.001507 | 192.168.56 | 192.168.56 | TCP | 66 | 59920 > 445 [ACK] Seq=1 Ack=1 Win=64256 Len=0 TSval=347780007 TSecr=71753 |
| 14 | 0.001673 | 192.168.56 | 192.168.56 | TCP | 70 | 59920 > 445 [PSH, ACK] Seq=1 Ack=1 Win=64256 Len=4 TSval=347780007 TSecr=71753 [TCP segment of a reassembled PDU] |
| 15 | 0.001726 | 192.168.56 | 192.168.56 | TCP | 74 | 59922 > 445 [SYN] Seq=0 Win=64240 Len=0 MSS=1460 SACK_PERM=1 TSval=347780007 TSecr=0 WS=128 |
| 16 | 0.002011 | 192.168.56 | 192.168.56 | TCP | 66 | 445 > 59920 [ACK] Seq=1 Ack=5 Win=29056 Len=0 TSval=71753 TSecr=347780007 |
| 17 | 0.002011 | 192.168.56 | 192.168.56 | TCP | 74 | 445 > 59922 [SYN, ACK] Seq=0 Ack=1 Win=28960 Len=0 MSS=1460 SACK_PERM=1 TSval=71753 TSecr=347780007 WS=128 |
| 18 | 0.002035 | 192.168.56 | 192.168.56 | TCP | 66 | 59922 > 445 [ACK] Seq=1 Ack=1 Win=64256 Len=0 TSval=347780007 TSecr=71753 |
| 19 | 0.002227 | 192.168.56 | 192.168.56 | TCP | 70 | 59922 > 445 [PSH, ACK] Seq=1 Ack=1 Win=64256 Len=4 TSval=347780007 TSecr=71753 [TCP segment of a reassembled PDU] |
| 20 | 0.00228 | 192.168.56 | 192.168.56 | TCP | 74 | 59924 > 445 [SYN] Seq=0 Win=64240 Len=0 MSS=1460 SACK_PERM=1 TSval=347780007 TSecr=0 WS=128 |
| 21 | 0.002586 | 192.168.56 | 192.168.56 | TCP | 66 | 445 > 59922 [ACK] Seq=1 Ack=5 Win=29056 Len=0 TSval=71753 TSecr=347780007 |
| 22 | 0.002586 | 192.168.56 | 192.168.56 | TCP | 74 | 445 > 59924 [SYN, ACK] Seq=0 Ack=1 Win=28960 Len=0 MSS=1460 SACK_PERM=1 TSval=71753 TSecr=347780007 WS=128 |
| 23 | 0.002612 | 192.168.56 | 192.168.56 | TCP | 66 | 59924 > 445 [ACK] Seq=1 Ack=1 Win=64256 Len=0 TSval=347780008 TSecr=71753 |
| 24 | 0.00278 | 192.168.56 | 192.168.56 | TCP | 70 | 59924 > 445 [PSH, ACK] Seq=1 Ack=1 Win=64256 Len=4 TSval=347780008 TSecr=71753 [TCP segment of a reassembled PDU] |
| 25 | 0.002832 | 192.168.56 | 192.168.56 | TCP | 74 | 59926 > 445 [SYN] Seq=0 Win=64240 Len=0 MSS=1460 SACK_PERM=1 TSval=347780008 TSecr=0 WS=128 |
| 26 | 0.003085 | 192.168.56 | 192.168.56 | TCP | 66 | 445 > 59924 [ACK] Seq=1 Ack=5 Win=29056 Len=0 TSval=71753 TSecr=347780008 |
| 27 | 0.003085 | 192.168.56 | 192.168.56 | TCP | 74 | 445 > 59926 [SYN, ACK] Seq=0 Ack=1 Win=28960 Len=0 MSS=1460 SACK_PERM=1 TSval=71753 TSecr=347780008 WS=128 |
| 28 | 0.003108 | 192.168.56 | 192.168.56 | TCP | 66 | 59926 > 445 [ACK] Seq=1 Ack=1 Win=64256 Len=0 TSval=347780008 TSecr=71753 |
| 29 | 0.003247 | 192.168.56 | 192.168.56 | TCP | 70 | 59926 > 445 [PSH, ACK] Seq=1 Ack=1 Win=64256 Len=4 TSval=347780008 TSecr=71753 [TCP segment of a reassembled PDU] |
| 30 | 0.003304 | 192.168.56 | 192.168.56 | TCP | 74 | 59928 > 445 [SYN] Seq=0 Win=64240 Len=0 MSS=1460 SACK_PERM=1 TSval=347780009 TSecr=0 WS=128 |
| 31 | 0.003513 | 192.168.56 | 192.168.56 | TCP | 66 | 445 > 59926 [ACK] Seq=1 Ack=5 Win=29056 Len=0 TSval=71753 TSecr=347780008 |
| 32 | 0.003514 | 192.168.56 | 192.168.56 | TCP | 74 | 445 > 59928 [SYN, ACK] Seq=0 Ack=1 Win=28960 Len=0 MSS=1460 SACK_PERM=1 TSval=71753 TSecr=347780009 WS=128 |
| 33 | 0.003549 | 192.168.56 | 192.168.56 | TCP | 66 | 59928 > 445 [ACK] Seq=1 Ack=1 Win=64256 Len=0 TSval=347780009 TSecr=71753 |
| 34 | 0.003759 | 192.168.56 | 192.168.56 | TCP | 70 | 59928 > 445 [PSH, ACK] Seq=1 Ack=1 Win=64256 Len=4 TSval=347780009 TSecr=71753 [TCP segment of a reassembled PDU] |
| 35 | 0.003816 | 192.168.56 | 192.168.56 | TCP | 74 | 59930 > 445 [SYN] Seq=0 Win=64240 Len=0 MSS=1460 SACK_PERM=1 TSval=347780009 TSecr=0 WS=128 |

Fig.10. Output of wireshark from attack in .csv format

One of the drawbacks of the Wireshark .csv output is that it is not useful in either data interpretation or analysis, since the fields such as timestamp and info are not recognizable for analysis tools and it is rather difficult to separate data in each field. Therefore, as a remedy to this issue, another tool called CICFlowMeter[2] is used which reads a .pcap file as input and outputs a .csv file. In addition, it can also identify the TCP sessions and hence can output data such as the average size of the transmitted packets in each session, standard deviation of the packet size, etc.

In the next step, we try to utilize this tool in order to perform the data interpretation/analysis. As can be observed in figure 11, the CICFlowMeter has two modes, realtime and offline, respectively. In the realtime mode, it will scan the network card for the ongoing connections and in the offline mode, it will need a .pcap file as input. Since, we have already extracted the .pcap file from the Wireshark tool, we will use the offline option.

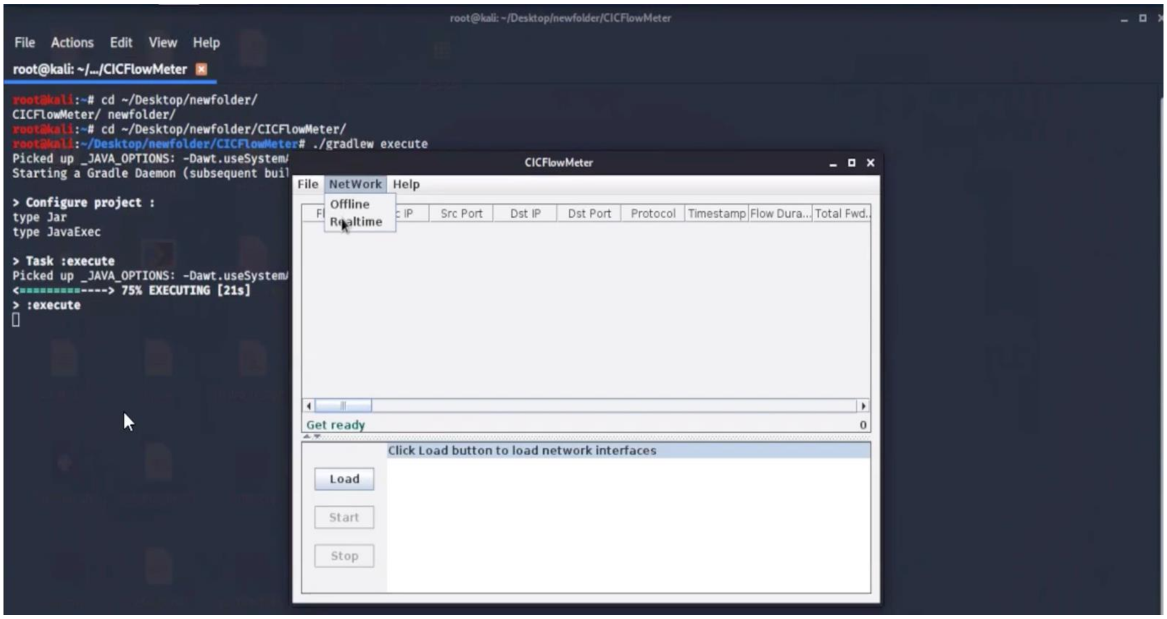
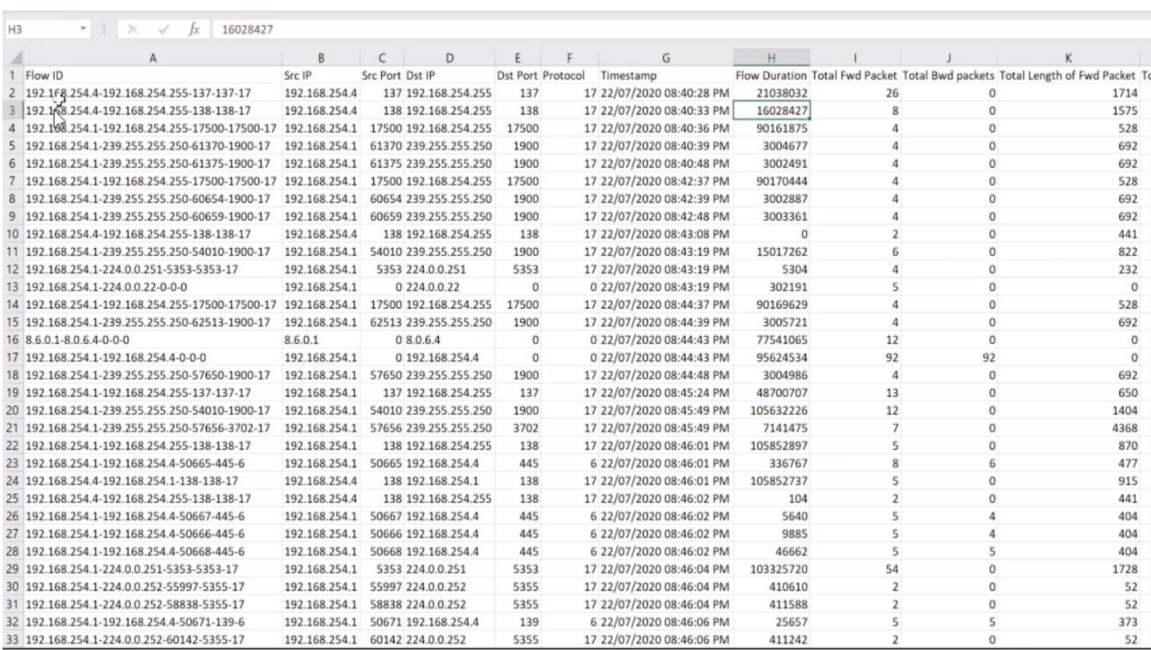



Fig.11. CICFlowMeter Operating Modes

Figure 12 depicts the output we got from the CICFlowMeter. Unlike the .csv output of the Wireshark, there are Src/Dest ports distinctively and the Timestamp is no longer in the form of nanoseconds. Flow Duration specifies the duration of each TCP connection. It is noticeable that the columns Src/Dest IP, Src/Dest port, Timestamp and Flow duration are used in our analysis which will be reviewed in the next section.



H3 16028427

| | A | B | C | D | E | F | G | H | I | J | K | |
|---|---|---|---|---|---|---|---|---|---|---|---|---|
| 1 | Flow ID | Src IP | Src Port | Dst IP | Dst Port | Protocol | Timestamp | Flow Duration | Total Fwd Packet | Total Bwd packets | Total Length of Fwd Packet | Tc |
| 2 | 192.168.254.4-192.168.254.255-137-137-17 | 192.168.254.4 | 137 | 192.168.254.255 | 137 | 17 | 22/07/2020 08:40:28 PM | 21038032 | 26 | 0 | 1714 | |
| 3 | 192.168.254.4-192.168.254.255-138-138-17 | 192.168.254.4 | 138 | 192.168.254.255 | 138 | 17 | 22/07/2020 08:40:33 PM | 16028427 | 8 | 0 | 1575 | |
| 4 | 192.168.254.1-192.168.254.255-17500-17500-17 | 192.168.254.1 | 17500 | 192.168.254.255 | 17500 | 17 | 22/07/2020 08:40:36 PM | 90161875 | 4 | 0 | 528 | |
| 5 | 192.168.254.1-239.255.255.250-61370-1900-17 | 192.168.254.1 | 61370 | 239.255.255.250 | 1900 | 17 | 22/07/2020 08:40:39 PM | 3004677 | 4 | 0 | 692 | |
| 6 | 192.168.254.1-239.255.255.250-61375-1900-17 | 192.168.254.1 | 61375 | 239.255.255.250 | 1900 | 17 | 22/07/2020 08:40:48 PM | 3002491 | 4 | 0 | 692 | |
| 7 | 192.168.254.1-192.168.254.255-17500-17500-17 | 192.168.254.1 | 17500 | 192.168.254.255 | 17500 | 17 | 22/07/2020 08:42:37 PM | 90170444 | 4 | 0 | 528 | |
| 8 | 192.168.254.1-239.255.255.250-60654-1900-17 | 192.168.254.1 | 60654 | 239.255.255.250 | 1900 | 17 | 22/07/2020 08:42:39 PM | 3002887 | 4 | 0 | 692 | |
| 9 | 192.168.254.1-239.255.255.250-60659-1900-17 | 192.168.254.1 | 60659 | 239.255.255.250 | 1900 | 17 | 22/07/2020 08:42:48 PM | 3003361 | 4 | 0 | 692 | |
| 10 | 192.168.254.4-192.168.254.255-138-138-17 | 192.168.254.4 | 138 | 192.168.254.255 | 138 | 17 | 22/07/2020 08:43:08 PM | 0 | 2 | 0 | 441 | |
| 11 | 192.168.254.1-239.255.255.250-54010-1900-17 | 192.168.254.1 | 54010 | 239.255.255.250 | 1900 | 17 | 22/07/2020 08:43:19 PM | 15017262 | 6 | 0 | 822 | |
| 12 | 192.168.254.1-224.0.0.251-5353-5353-17 | 192.168.254.1 | 5353 | 224.0.0.251 | 5353 | 17 | 22/07/2020 08:43:19 PM | 5304 | 4 | 0 | 232 | |
| 13 | 192.168.254.1-224.0.0.22-0-0-0 | 192.168.254.1 | 0 | 224.0.0.22 | 0 | 0 | 22/07/2020 08:43:19 PM | 302191 | 5 | 0 | 0 | |
| 14 | 192.168.254.1-192.168.254.255-17500-17500-17 | 192.168.254.1 | 17500 | 192.168.254.255 | 17500 | 17 | 22/07/2020 08:44:37 PM | 90169629 | 4 | 0 | 528 | |
| 15 | 192.168.254.1-239.255.255.250-62513-1900-17 | 192.168.254.1 | 62513 | 239.255.255.250 | 1900 | 17 | 22/07/2020 08:44:39 PM | 3005721 | 4 | 0 | 692 | |
| 16 | 8.6.0.1-8.0.6.4-0-0-0 | 8.6.0.1 | 0 | 8.0.6.4 | 0 | 0 | 22/07/2020 08:44:43 PM | 77541065 | 12 | 0 | 0 | |
| 17 | 192.168.254.1-192.168.254.4-0-0-0 | 192.168.254.1 | 0 | 192.168.254.4 | 0 | 0 | 22/07/2020 08:44:43 PM | 95624534 | 92 | 92 | 0 | |
| 18 | 192.168.254.1-239.255.255.250-57650-1900-17 | 192.168.254.1 | 57650 | 239.255.255.250 | 1900 | 17 | 22/07/2020 08:44:48 PM | 3004986 | 4 | 0 | 692 | |
| 19 | 192.168.254.1-192.168.254.255-137-137-17 | 192.168.254.1 | 137 | 192.168.254.255 | 137 | 17 | 22/07/2020 08:45:24 PM | 48700707 | 13 | 0 | 650 | |
| 20 | 192.168.254.1-239.255.255.250-54010-1900-17 | 192.168.254.1 | 54010 | 239.255.255.250 | 1900 | 17 | 22/07/2020 08:45:49 PM | 105632226 | 12 | 0 | 1404 | |
| 21 | 192.168.254.1-239.255.255.250-57656-3702-17 | 192.168.254.1 | 57656 | 239.255.255.250 | 3702 | 17 | 22/07/2020 08:45:49 PM | 7141475 | 7 | 0 | 4368 | |
| 22 | 192.168.254.1-192.168.254.255-138-138-17 | 192.168.254.1 | 138 | 192.168.254.255 | 138 | 17 | 22/07/2020 08:46:01 PM | 105852897 | 5 | 0 | 870 | |
| 23 | 192.168.254.1-192.168.254.4-50665-445-6 | 192.168.254.1 | 50665 | 192.168.254.4 | 445 | 6 | 22/07/2020 08:46:01 PM | 336767 | 8 | 6 | 477 | |
| 24 | 192.168.254.4-192.168.254.1-138-138-17 | 192.168.254.4 | 138 | 192.168.254.1 | 138 | 17 | 22/07/2020 08:46:01 PM | 105852737 | 5 | 0 | 915 | |
| 25 | 192.168.254.4-192.168.254.255-138-138-17 | 192.168.254.4 | 138 | 192.168.254.255 | 138 | 17 | 22/07/2020 08:46:02 PM | 104 | 2 | 0 | 441 | |
| 26 | 192.168.254.1-192.168.254.4-50667-445-6 | 192.168.254.1 | 50667 | 192.168.254.4 | 445 | 6 | 22/07/2020 08:46:02 PM | 5640 | 5 | 4 | 404 | |
| 27 | 192.168.254.1-192.168.254.4-50666-445-6 | 192.168.254.1 | 50666 | 192.168.254.4 | 445 | 6 | 22/07/2020 08:46:02 PM | 9885 | 5 | 4 | 404 | |
| 28 | 192.168.254.1-192.168.254.4-50668-445-6 | 192.168.254.1 | 50668 | 192.168.254.4 | 445 | 6 | 22/07/2020 08:46:02 PM | 46662 | 5 | 5 | 404 | |
| 29 | 192.168.254.1-224.0.0.251-5353-5353-17 | 192.168.254.1 | 5353 | 224.0.0.251 | 5353 | 17 | 22/07/2020 08:46:04 PM | 103325720 | 54 | 0 | 1728 | |
| 30 | 192.168.254.1-224.0.0.252-55997-5355-17 | 192.168.254.1 | 55997 | 224.0.0.252 | 5355 | 17 | 22/07/2020 08:46:04 PM | 410610 | 2 | 0 | 52 | |
| 31 | 192.168.254.1-224.0.0.252-58838-5355-17 | 192.168.254.1 | 58838 | 224.0.0.252 | 5355 | 17 | 22/07/2020 08:46:04 PM | 411588 | 2 | 0 | 52 | |
| 32 | 192.168.254.1-192.168.254.4-50671-139-6 | 192.168.254.1 | 50671 | 192.168.254.4 | 139 | 6 | 22/07/2020 08:46:06 PM | 25657 | 5 | 5 | 373 | |
| 33 | 192.168.254.1-224.0.0.252-60142-5355-17 | 192.168.254.1 | 60142 | 224.0.0.252 | 5355 | 17 | 22/07/2020 08:46:06 PM | 411242 | 2 | 0 | 52 | |

Fig.12.Output of CICFlowMeter from attack in .csv format

In summary, we have come to know that the SMB loris DOS attack can be detected based on the number of Syn/Ack, Fin/Ack or RST packets sent to a specific destination. We had difficulty in order to detect that data from the Wireshark .csv output and hence decided to move forward by using a tool called CICFlowMeter to render data to generate refined output in a useful format to our analysis.

### C. Security/Privacy Data Analytics

The objective of this section is to demonstrate the capability of conducted data analysis methods in order to detect future DOS attacks (i.e,SMB LORIS) in the network environment. Initially, we begin the analysis by loading data and parsing it, but before that, we have performed the data cleaning by omitting the fields with destination port 0 and data values of infinity. Lastly, as depicted in figure 13, the data.shape is printed in order to identify the number of rows and columns and there are 235032 rows and 84 columns to the output .csv file

```
In [245]: import pandas as pd
          import numpy as np
          from sklearn import preprocessing
          import seaborn as sns
          import matplotlib.pyplot as plt
          import warnings
          warnings.filterwarnings('ignore')
          path_to_dataset="smb lois attack.pcap_Flow.csv"
          #We are going to play with timestamp, so parse it
          data = pd.read_csv(path_to_dataset,header=0, low_memory=False, parse_dates=['Timestamp'])
          data = data[data['Dst Port'] != 0]
          data = data[data.values != 'Infinity']
          #Set data index equal to timestamp so we can groupby records based on time
          data.index=data.Timestamp
          #Rows and columns count
          print("Dataframe shape: ", data.shape)

          Dataframe shape:  (235032, 84)
```

Fig.13. Reading and cleaning data in JupyterNotebook

.

As observed in figure 14, Data.head() function represents a sneak peak of the few first columns of the output data.

```
[2]: #Lets have a look on first few rows
     data.head()
```

```
[2]:                                                     Flow ID  \
Timestamp
2020-07-22 20:40:36  192.168.254.1-192.168.254.255-17500-17500-17
2020-07-22 20:40:36  192.168.254.1-192.168.254.255-17500-17500-17
2020-07-22 20:40:36  192.168.254.1-192.168.254.255-17500-17500-17
2020-07-22 20:40:36  192.168.254.1-192.168.254.255-17500-17500-17
2020-07-22 20:40:36  192.168.254.1-192.168.254.255-17500-17500-17

                            Src IP  Src Port          Dst IP  Dst Port  \
Timestamp
2020-07-22 20:40:36  192.168.254.1     17500  192.168.254.255     17500
2020-07-22 20:40:36  192.168.254.1     17500  192.168.254.255     17500
2020-07-22 20:40:36  192.168.254.1     17500  192.168.254.255     17500
2020-07-22 20:40:36  192.168.254.1     17500  192.168.254.255     17500
2020-07-22 20:40:36  192.168.254.1     17500  192.168.254.255     17500


                     Protocol           Timestamp  Flow Duration  \
Timestamp
2020-07-22 20:40:36        17 2020-07-22 20:40:36       90161875
2020-07-22 20:40:36        17 2020-07-22 20:40:36       90161875
2020-07-22 20:40:36        17 2020-07-22 20:40:36       90161875
2020-07-22 20:40:36        17 2020-07-22 20:40:36       90161875
2020-07-22 20:40:36        17 2020-07-22 20:40:36       90161875

                     Total Fwd Packet  Total Bwd packets  ...  \
Timestamp                                                 ...
2020-07-22 20:40:36                 4                  0  ...
2020-07-22 20:40:36                 4                  0  ...
2020-07-22 20:40:36                 4                  0  ...
2020-07-22 20:40:36                 4                  0  ...
2020-07-22 20:40:36                 4                  0  ...

                     Fwd Seg Size Min  Active Mean  Active Std  Active Max  \
Timestamp
2020-07-22 20:40:36                 8          0.0         0.0         0.0
2020-07-22 20:40:36                 8          0.0         0.0         0.0
2020-07-22 20:40:36                 8          0.0         0.0         0.0
2020-07-22 20:40:36                 8          0.0         0.0         0.0
2020-07-22 20:40:36                 8          0.0         0.0         0.0

                     Active Min     Idle Mean      Idle Std      Idle Max  \
Timestamp
2020-07-22 20:40:36         0.0  3.988662e+14  7.977324e+14  1.595465e+15
2020-07-22 20:40:36         0.0  3.988662e+14  7.977324e+14  1.595465e+15
2020-07-22 20:40:36         0.0  3.988662e+14  7.977324e+14  1.595465e+15
2020-07-22 20:40:36         0.0  3.988662e+14  7.977324e+14  1.595465e+15
2020-07-22 20:40:36         0.0  3.988662e+14  7.977324e+14  1.595465e+15
```

Fig.14. Output data generated by the JupyterNotebook

In our analysis, we have focused on utilizing SRC/Dest IP, SRC/Dest port, timestamp and flow duration columns as well as SYN/ACK, FIN/ACK and RST from the 84 output columns as depicted in figure 15.

```
In [247]: #Column names
          list(data.columns)

Out[247]: ['Flow ID',
           'Src IP',
           'Src Port',
           'Dst IP',
           'Dst Port',
           'Protocol',
           'Timestamp',
           'Flow Duration',
           'Total Fwd Packet',
           'Total Bwd packets',
           'Total Length of Fwd Packet',
           'Total Length of Bwd Packet',
           'Fwd Packet Length Max',
           'Fwd Packet Length Min',
           'Fwd Packet Length Mean',
           'Fwd Packet Length Std',
           'Bwd Packet Length Max',
           'Bwd Packet Length Min',
           'Bwd Packet Length Mean',
           'Bwd Packet Length Std',
           'Flow Bytes/s',
           'Flow Packets/s',
           'Flow IAT Mean',
           'Flow IAT Std',
           'Flow IAT Max',
           'Flow IAT Min',
           'Fwd IAT Total',
           'Fwd IAT Mean',
           'Fwd IAT Std',
           'Fwd IAT Max',
           'Fwd IAT Min',
           'Bwd IAT Total',
           'Bwd IAT Mean',
```

Fig.15. Dataset Columns List in JupyterNotebook

Figure 16 depicts the density of the number of TCP session connections grouped by the destination IP in each 10 seconds interval time. Based on the result, there are two spikes in an interval of 15 minutes, 20:55 and 21:05 respectively, which is far from normal behavior and matches the attack scenario. However, generally we cannot correlate the high number of TCP session connections to any attack as the IP may belong to a website which is hosting a special event, for instance. In addition, it is noticeable that unlike the number of TCP connections that we have earlier set in Kali to 65535, the number of TCP session connections in below snapshot is 40000 and 45000 which can be viewed as the number of TCP session connections possible on the victim's system before it runs out of memory and hangs.

```
[4]: #For each IP in Dst IP column, how many tcp connections in 10 sec intervals
     data.resample('10s')['Dst IP'].count().plot(figsize=(20,7))
     plt.xlabel('Time')
     plt.ylabel('Density of TCP connections')

[4]: Text(0, 0.5, 'Density of TCP connections')
```

```
In [248]: #For each IP in Dst IP column, how many tcp connections in 10 sec intervals
          data.resample('10s')['Dst IP'].count().plot(figsize=(20,7))
          plt.xlabel('Time')
          plt.ylabel('Density of TCP connections')

Out[248]: Text(0, 0.5, 'Density of TCP connections')
```

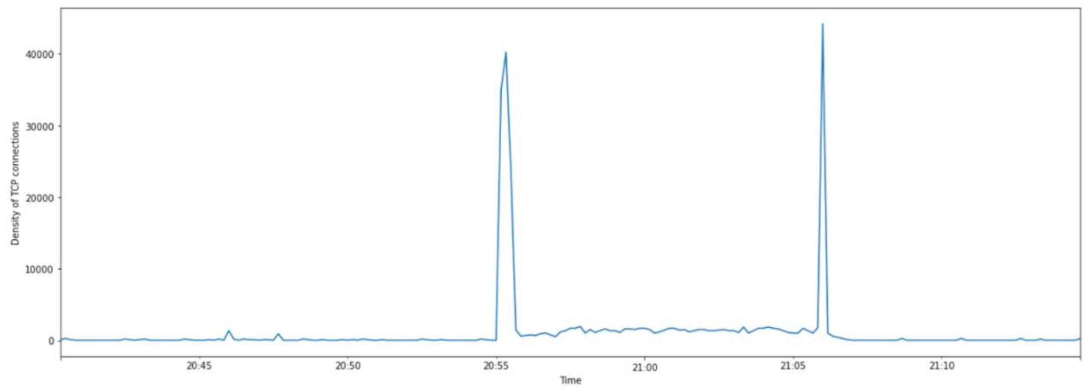



Fig.16. Density of the number of TCP session connections grouped by the destination IP

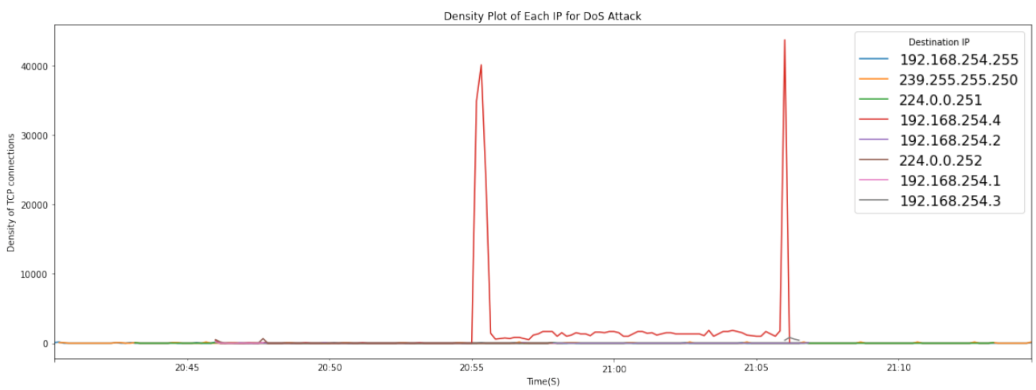
In figure 17, the density of the IP addresses TCP session connections involved in the attack scenario is analyzed in order to confirm if they match with what was configured earlier. As depicted below, two main IP addresses involved in the scenario are 192.168.254.4 and 192.168.254.3 which are the same as victim and attacker machine IPs, respectively.

```
[5]: # Density Plot for each IP
IPs=data['Dst IP'].unique()
plt.figure(figsize=(20,7))

for IP in IPs:
    # Subset to the destination port
    subset = data[data['Dst IP'] == IP]
    subset.resample('10s')['Dst IP'].count().plot(figsize=(20,7),label = IP)

# Plot formatting
plt.legend(prop={'size': 16}, title = 'Destination IP')
plt.title('Density Plot of Each IP for DoS Attack')
plt.xlabel('Time(S)')
plt.ylabel('Density of TCP connections')
```

```
[5]: Text(0, 0.5, 'Density of TCP connections')
```



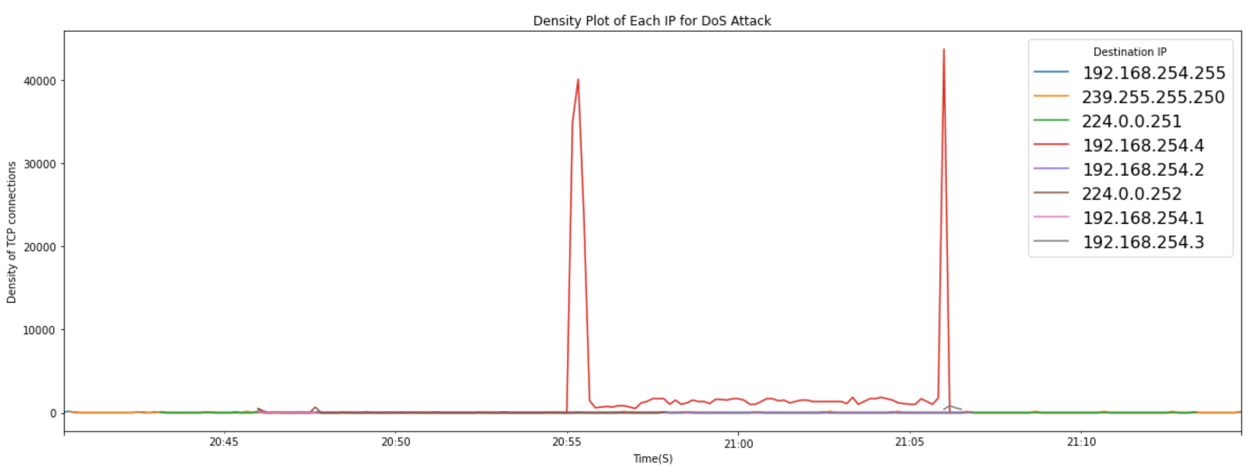




Fig.17 Density of IP addresses TCP session connection involved in attack scenario

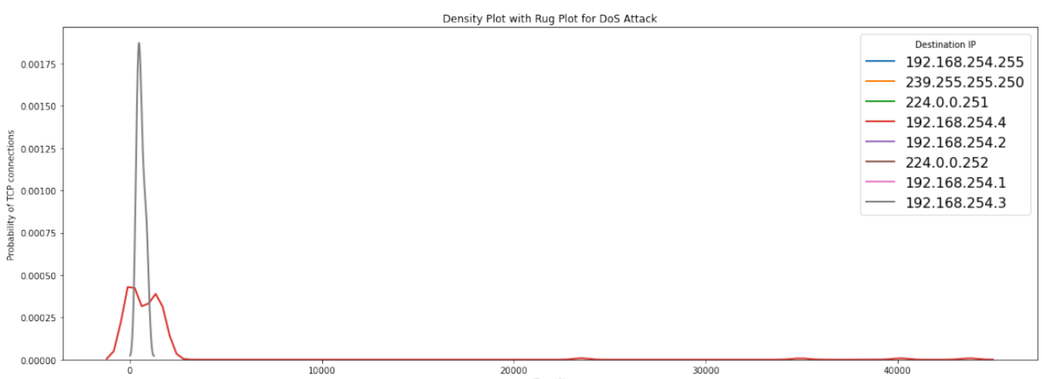
Due to the scaling issue of the last plot, we have also attempted to generate a Rug Plot based on the KDE method which operates based on the probability density function and depicts the probability of having TCP connections in a period of time. As observed in figure 18, the attacker has the maximum probability of having TCP connections at time 0. Next highest probability belongs to the Victim's IP. As it is mentioned before, the SMB loris attack targets the victim with a high number of TCP connections to which the victim needs to respond and since the memory resources of the victim is limited, after a certain number of TCP SYN requests, it has used up all its resources and there is no more resources available to be allocated to new TCP connections and hence there is no response to the attacker's request. So it is rational to see that not only the number of packets but also the probability of generated TCP connections originating from the attacker's side outweigh that of the victim's side. That is the reason the attacker's IP spike is almost 0.00175 and that of the victim is almost 0.0005. In addition, we can conclude that compared to other network entities, most of the communication is between the attacker and the victim.

[6]:
```python
# Density Plot with Rug Plot
IPs=data['Dst IP'].unique()
plt.figure(figsize=(20,7))

for IP in IPs:
    # Subset to the destination port
    subset = data[data['Dst IP'] == IP]
    # Draw the density plot
    sns.distplot(subset.resample('10s')['Dst IP'].count(), hist = False, kde =␣
↪True,
                 kde_kws = {'linewidth': 2},
                 label = IP)

# Plot formatting
plt.legend(prop={'size': 16}, title = 'Destination IP')
plt.title('Density Plot with Rug Plot for DoS Attack')
plt.xlabel('Time(S)')
plt.ylabel('Probability of TCP connections')
```

[6]: Text(0, 0.5, 'Probability of TCP connections')



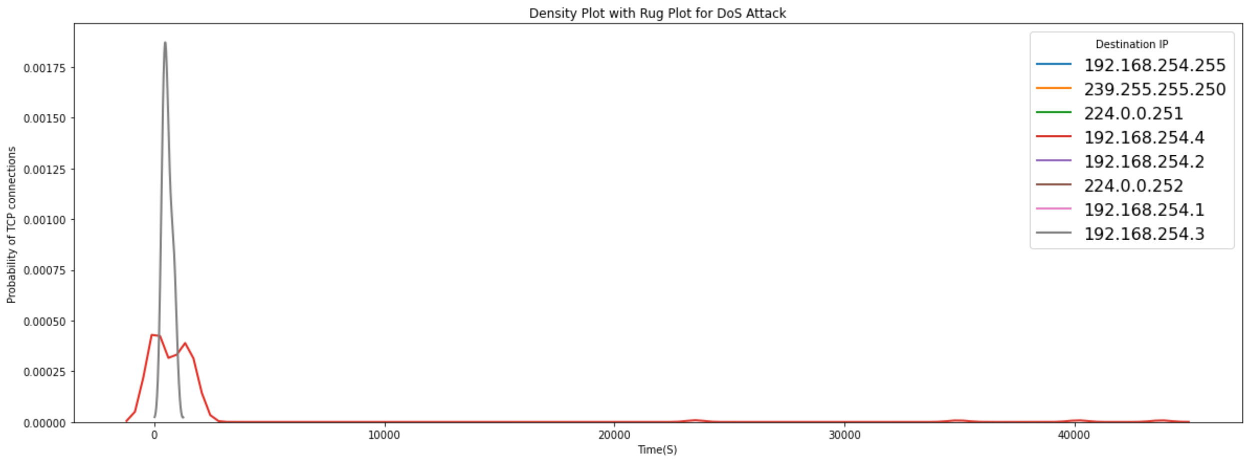



Fig.18 Density plot with Rug plot for DOS attack

As the result of the above discussions, we have reached the conclusion that the communication is between the attacker's IP(192.168.254.3) and the victim's IP(192.168.254.4). Based on a filtered subset of data from the original dataset, which encompasses the communication between the attacker and the victim, we attempted to generate the density of the number of TCP connections again in order to confirm that the plots timing, numbers, etc follow the attack procedure. As observed in figure 19, based on the density of TCP session connections, there exist two spikes at 20:55 and 21:05 which match the time of both attacks and hence confirm the first conclusion driven before.

Conclusion: Most of communications are between gray and red IPs (192.168.254.4 and 192.168.254.3)

[7]:
```
#Plot our suspicious subset of dataset
df_suspicious=data[(data['Dst IP'] =='192.168.254.4') & (data['Src IP'] =='192.
↪168.254.3')]
df_suspicious.resample('5s')['Src IP'].count().plot(figsize=(20,7))
plt.xlabel('Time')
plt.ylabel('Density of TCP connections')
```

[7]:
```
Text(0, 0.5, 'Density of TCP connections')
```

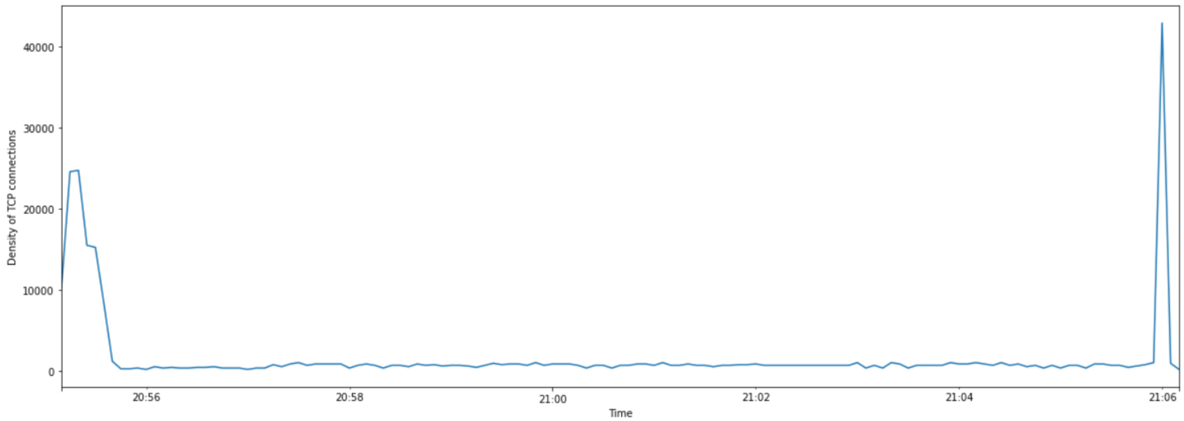



Fig.19. Density of TCP connections

From another point of view, we count the unique used destination ports for each involved IP in figure 20. As depicted below, the victim has used two ports as SMB loris operated on port 445 and the ping operation is performed on port 21. The attacker, on the other hand, utilized 14 ports at a time which is suspicious. That is why we have conducted the same study based on the source port as well based on which the attacker IP has had 1701 ports involved in initiation of TCP connections during this period which explains why 14 distinct destination ports have been involved. It is noticeable that since the attacker has opened 1701 TCP connections and the memory space of the victim is limited, it only had enough resources to reply to 14 of those ports. It is hence concluded that the IP 192.168.254.3 is the attacker in this scenario since it has initiated TCP connections on 1701 of its ports in a period of 15 minutes. Since we have earlier seen that the density of TCP connections numbers is about 40,000 and above at the time of attack, we have further analyzed the logs and come to know that some of the 1700 ports are utilized to initiate multiple TCP connections.

```
[8]: #Validating our conclusioins from other points of view
df2 = data.groupby("Dst IP")
print('IP','\t\t','Port')
df2['Dst Port'].nunique()
```

```
IP              Port

[8]: Dst IP
192.168.254.1       3
192.168.254.2       1
192.168.254.255     3
192.168.254.3      14
192.168.254.4       2
224.0.0.251         1
224.0.0.252         1
239.255.255.250     2
Name: Dst Port, dtype: int64
```

Our victim is 192.168.254.4 with 2 of its ports as destination port Also suspicious attacker is communicated on 14 different ports

```
[9]: df2 = data.groupby("Src IP")
print('IP','\t\t','Port')
print(df2['Src Port'].nunique())
```

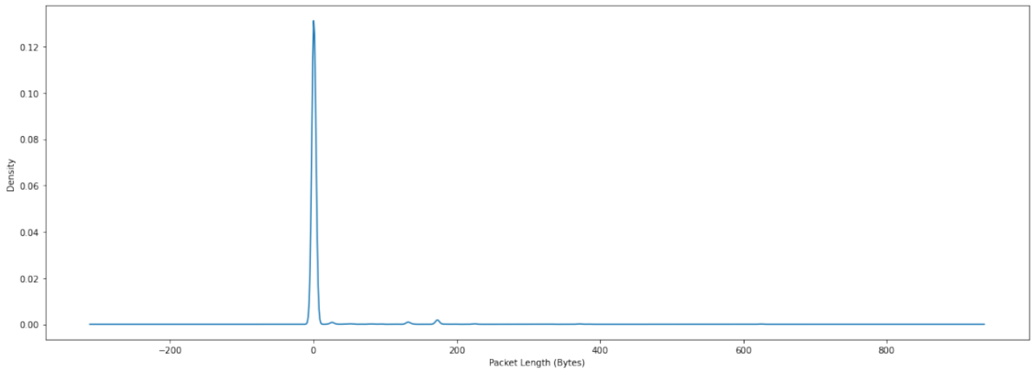
```
IP              Port
Src IP
192.168.254.1      67
192.168.254.3    1701
192.168.254.4       5
Name: Src Port, dtype: int64
```

Fig.20. The unique used destination ports for each involved

The length of the packets has also been taken into consideration and were plotted in figure 21. It is observed that most of the packets are of a very small size and that some of the packets are of negative size which is related to the lack of data cleaning. The small size of the packets explains that most of the packets are related to the SMBLoris attack and are of the type TCP connection. In order to confirm that the pattern matches that of the suspicious subset of data related to the IPs 192.168.254.3 and 192.168.254.4, we run the same experiment only on that specific subset of data to check the length of packets which further confirms that the attack data packets are of a very small size.

[10]:
```
#Plotting packets length density
plt.figure()
data['Packet Length Mean'].plot(kind = 'density',figsize=(20,7))
plt.xlabel('Packet Length (Bytes)')
plt.ylabel('Density')
```

[10]: Text(0, 0.5, 'Density')



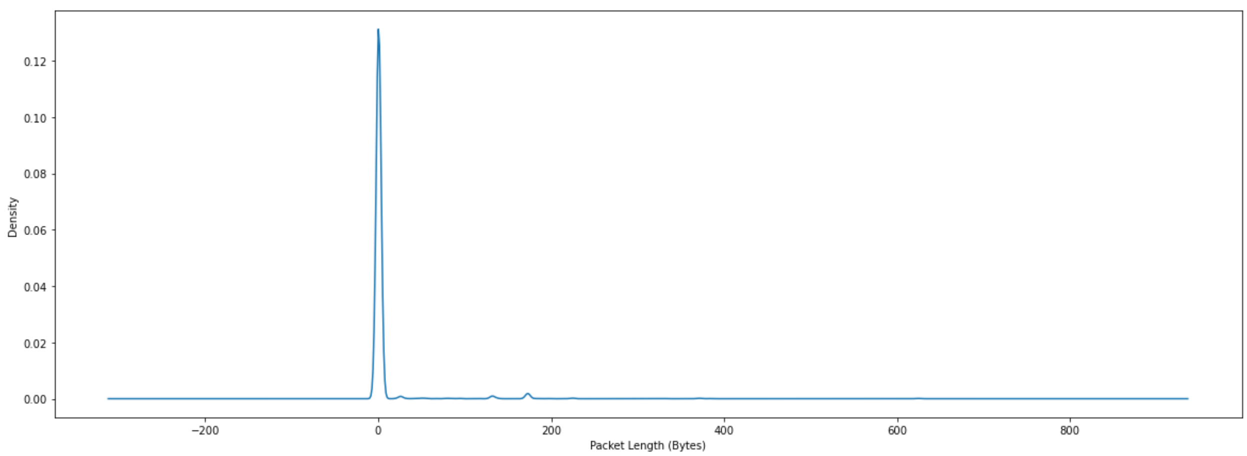
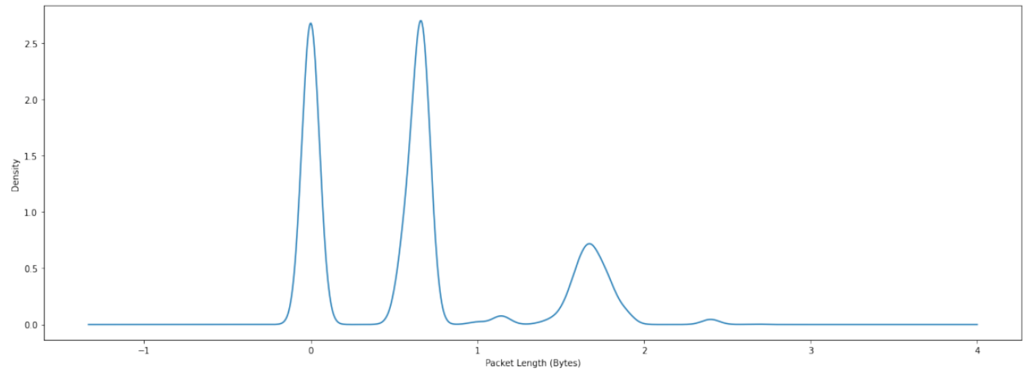



Fig.21. Packet length of whole data set

```
[11]: df_suspicious['Packet Length Mean'].plot(kind = 'density',figsize=(20,7))
      plt.xlabel('Packet Length (Bytes)')
      plt.ylabel('Density')
```

```
[11]: Text(0, 0.5, 'Density')
```



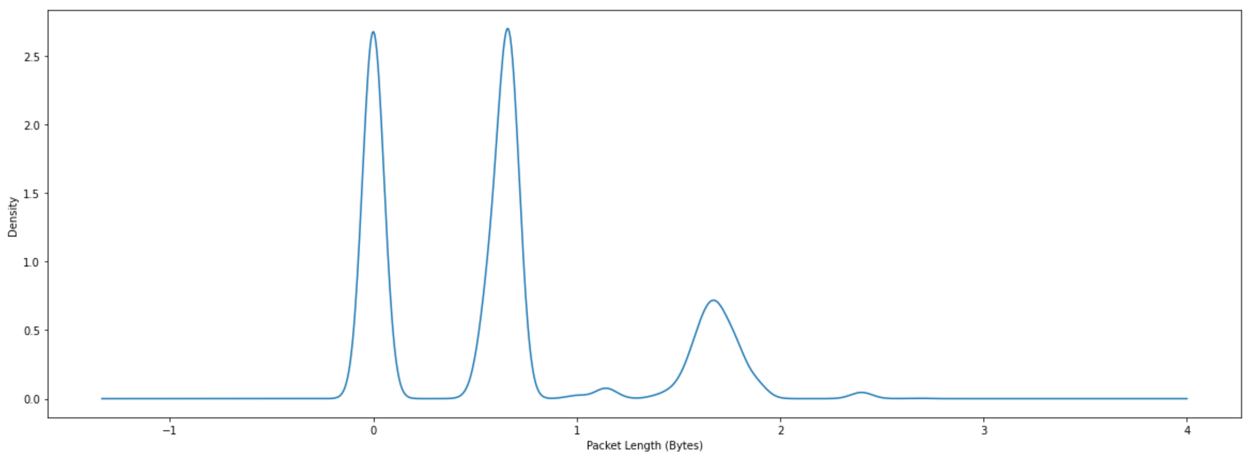




Fig.22. Packet length in bytes of suspicious subset of data

Also, the source and destination ports related to the suspicious data set were plotted in figure 23 and it is observed that the destination port is 445 which is used for the SMBLoris attack.

```
[12]: #From which source ports to which destination ports?
print('Source port count:' ,df_suspicious['Src Port'].nunique(),'\t',␣
 ↪'Destination port count:',df_suspicious['Dst Port'].nunique())
plt.figure(figsize=(5,4))
sns.boxplot(
    data=df_suspicious,
    x='Dst Port',
    y='Src Port',
    color='red')
```

```
Source port count: 1700          Destination port count: 1
```

```
[12]: <matplotlib.axes._subplots.AxesSubplot at 0x223d0d55948>
```

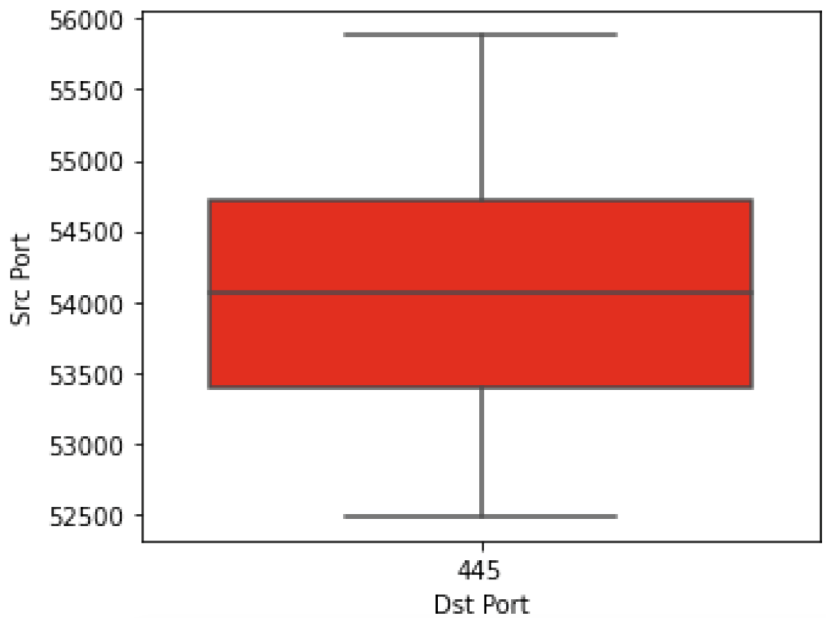



Fig.23. Source/Destination ports of the suspicious dataset

As mentioned earlier, the SMBLoris attack traffic is different from that of the normal traffic in terms of the number of SYN packets generated from the attacker's machine towards the victim in a short period of time. Figure 24 depicts the ratio of the number of SYN packets to the total number of packets in each TCP connection of the attacker and the windows machine. As can be observed, the windows machine has very few spikes which are related to the ping and samba file browsing whereas the attacker traffic shows that a high percentage of the packets are having SYN flags from 20:55 to 21:05 which is the attack period.

```
[13]: # Density Plot for each IP
      IPs=data['Src IP'].unique()
      plt.figure(figsize=(20,7))
      syn_flag_ratio=[]
      for IP in IPs:
          # Subset to the destination port
          subset = data[data['Src IP'] == IP]
          syn_count=subset['SYN Flag Count'].astype(int)
          total_fw=subset['Total Fwd Packet'].astype(int)
          syn_flag_ratio.append(syn_count/total_fw)



      syn_flag_ratio[2].plot(label='192.168.254.3')
      syn_flag_ratio[0].plot(label='192.168.254.1')
      plt.legend(prop={'size': 16}, title = 'Source IP')
      plt.xlabel('Time')
      plt.ylabel('Syn Flag Ratio')
```

```
[13]: Text(0, 0.5, 'Syn Flag Ratio')
```

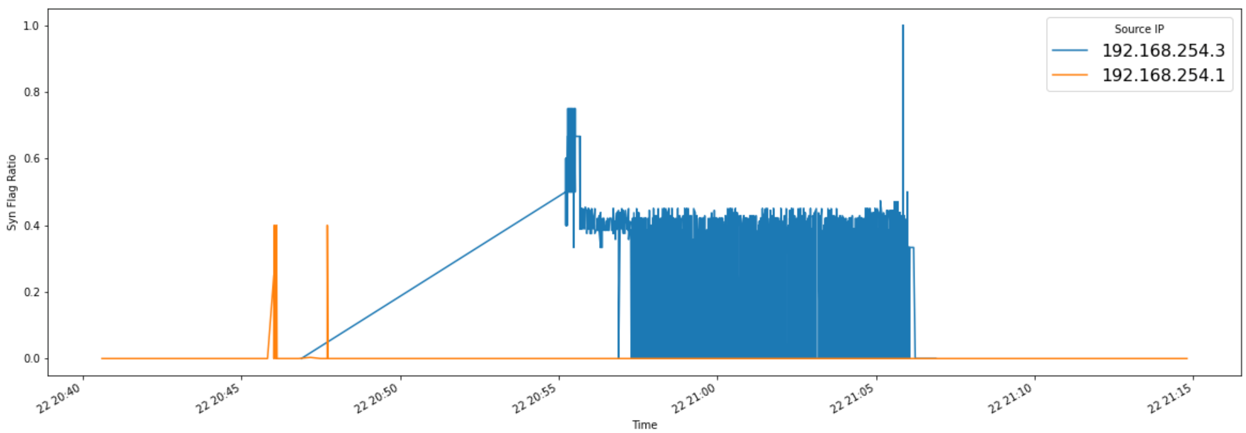


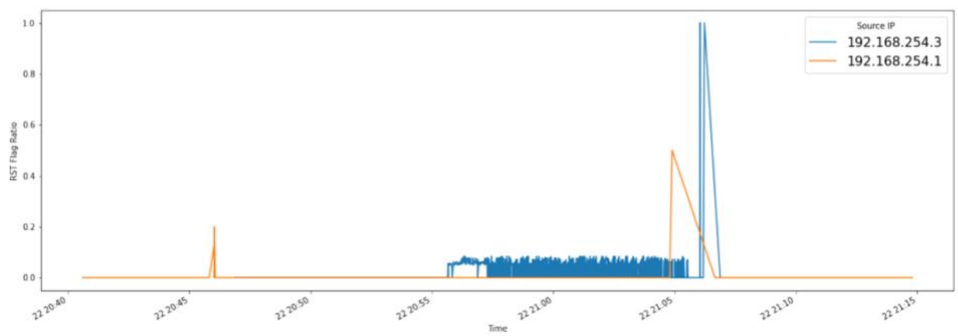

Fig.24. Attacker’s ratio of the number of SYN packets to the total number of packets in each TCP connection

As another measure of the SMBLoris attack, the number of RST packets has also been analyzed and as can be seen below, short after the attack started (20:56), the number of RST packets originated from the attacker towards the victim ,due to no response from the victim, is increased and continued till the time (21:07) that manual reboot has been conducted.

```
[34]: # Density Plot for each IP
      IPs=data['Src IP'].unique()
      plt.figure(figsize=(20,7))
      syn_flag_ratio=[]
      for IP in IPs:
          # Subset to the destination port
          subset = data[data['Src IP'] == IP]
          syn_count=subset['RST Flag Count'].astype(int)
          total_fw=subset['Total Fwd Packet'].astype(int)
          syn_flag_ratio.append(syn_count/total_fw)


      plt.legend(prop={'size': 16}, title = 'Source IP')
      plt.xlabel('Time')
      plt.ylabel('RST Flag Ratio')

[34]: Text(0, 0.5, 'RST Flag Ratio')
```



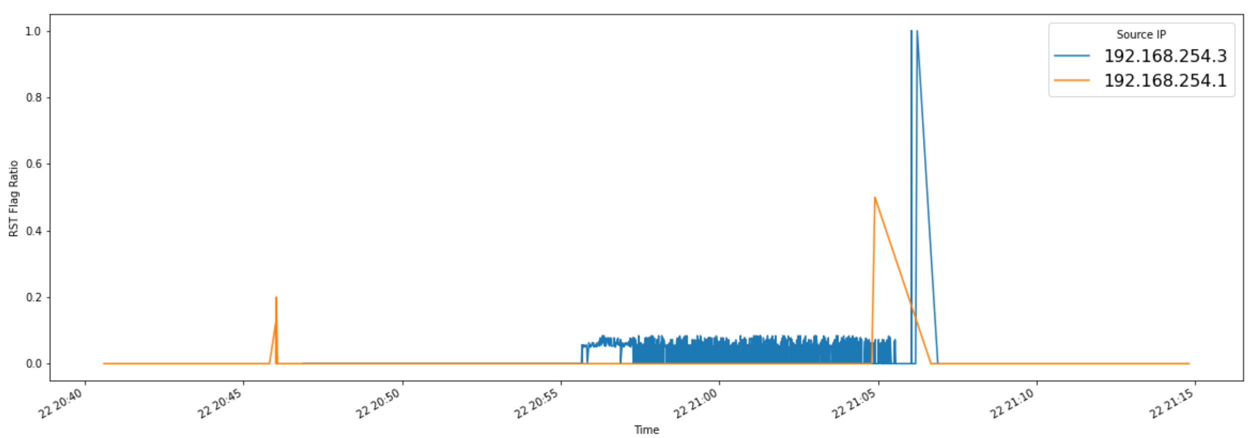




Fig.25. Attacker's RST flag ratio

As mentioned before, another measure of SMBLoris attack is the number of FIN packets generated from the attacker to the victim's machine. As observed in figure 26, there is a meaningful discrepancy between the FIN flag ratio initiated from the attacker's side and that of the victim during a short period of time which confirms the attack's pattern.

[38]:
```python
# Density Plot for each IP
IPs=data['Src IP'].unique()
plt.figure(figsize=(20,7))
syn_flag_ratio=[]
for IP in IPs:
    # Subset to the destination port
    subset = data[data['Src IP'] == IP]
    syn_count=subset['FIN Flag Count'].astype(int)
    total_fw=subset['Total Fwd Packet'].astype(int)
    syn_flag_ratio.append(syn_count/total_fw)



syn_flag_ratio[2].plot(label='192.168.254.3')
syn_flag_ratio[0].plot(label='192.168.254.1')
plt.legend(prop={'size': 16}, title = 'Source IP')
plt.xlabel('Time')
plt.ylabel('FIN Flag Ratio')
```

[38]:
```
Text(0, 0.5, 'FIN Flag Ratio')
```

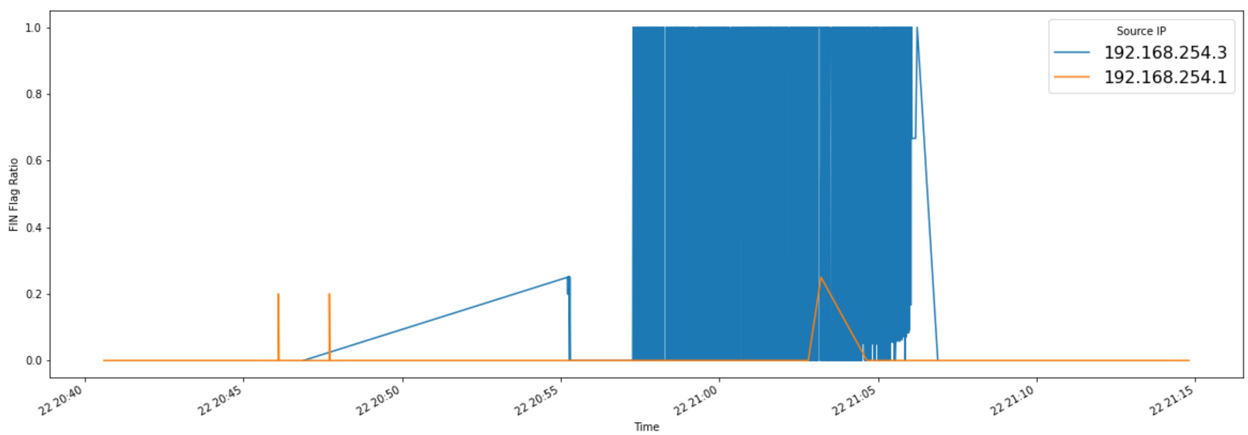



Fig.26. Attacker's FIN flag ratio

## III. Conclusions

SMBLoris attack is a type of denial of service attack which initiates several TCP connections in order to consume the victim's resources and is different from the normal traffic in the number of transmitted SYN, FIN/ACK and RST between involved parties. As a result of this report's analysis, SMBLoris attack pattern matches the transmission of a high number of previously-mentioned packets in a short duration of time.

## Abbreviations

SMBLoris--------------------------------------------------------------------------------------Samba Loris
DoS----------------------------------------------------------------attack Denial-of-Service (DoS) attack
TCP-------------------------------------------------------------------------Transmission Control Protocol
SYN---------------------------------------------------------------------------------------------
Synchronize
SYN/ ACK---------------------------------------------------------------- Synchronize/Acknowledgment
ACK--------------------------------------------------------------------------------------- Acknowledgment
FIN/ACK------------------------------------------------------------------------ Finish/ Acknowledgment
PSH/ACK------------------------------------------------------------------------ Push/ Acknowledgment

**References**


[1] Information Security Analytics: Finding Security Insights, Patterns, and Anomalies in Big Data, Talabis, McPherson, Miyamoto, Martin
[2] https://github.com/ahlashkari/CICFlowMeter
[3] https://www.secpod.com/smbloris
[4]https://www.kali.org
[5]https://www.wireshark.org
[6]https://jupyter.org