Assignment 3 – Network Security

CSCI 6706 Network Design

Christian Liu – B00415613
Computer Science Department
Dalhousie University
Halifax, NS
Chris.liu@dal.ca

Table of Contents

ABSTRACT

The report is to utilize several tools and technologies to detect the issues efficiently and effectively we want. I broke down the improvement process into three stages: **original snore configuration, several snort rules customization such as ICMP and DOS detection and ACL deployment**. During each stage, I came up with snort statistics report. At the end, I accumulate those reports on these three stages and visualized the comparison among those.

This report conducted many experimental exercises via penetration tests and provided lots of evidence to prove the improvements finally can make Snort work just what we want.

KEYWORDS

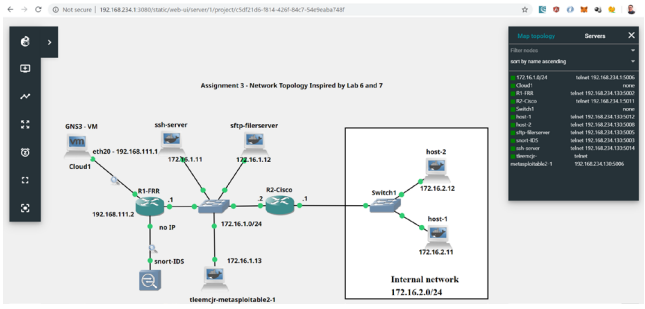
Snort, TCPReplay, TCPRewrite, Wireshark, Argus, Penetration, Iptables

1 Introduction

Network vulnerability detection is the main topic of this report. In this report, I will utilize several very popular network security tools, such as Snort, TCPRewrite, TCPReplay etc., to detect network issues during the network traffic sniffing. The WireShark can also help us with monitoring and preprocess the datasets captured from traffic. Moreover, I conducted many penetration tests against targeted victim machine installed on my GNS3 topology internal network (The internet connection was cut off during testing to prevent any packages leaking). As the result, I was able to accumulate and analyze the results from testing. At the end, I gave the conclusions based on my discovery and research. I also provision the further work based on this report.

2 Environment Configuration

The development work environment needs a little configuration, it is worth to mention in this document:



We are reusing the topology of lab 6 and 7 to complete this assignment. It needs a little more configuration on my local because I am using GNS 3 VM as the host machine. (GNS3 on Kali VM overloads my laptop, due to my limited resources on my laptop). Important, I imported "tleemcjr-metasploitable" docker image as a vulnerable target in this assignment.

I needed to manually install following tools for this assignment:

Net-tools

Argus-client

Besides, I need to configure a "virtual network interface" on my VM to connect with my topology.

TUN/TAP interfaces are virtual interfaces that are spawned from the base device /dev/net/tun and they pick up its system permissions so change the owning group to netdev instead of root[4].

sudo tunctl -u gns3 -g gns3 -t vip2

sudo ifconfig vip2 192.168.111.1 netmask 255.255.255.0 up

sudo ip route add 172.16.0.0/22 via 192.168.111.2 dev vip2

sudo ip route add 172.16.1.0/24 via 192.168.111.2 dev vip2

sudo ip route add 172.16.2.0/24 via 192.168.111.2 dev vip2

```
sudo chown gns3:gns3 /dev/net/tun
sudo ip link delete vip2
sudo tunctl -u gns3 -g gns3 -t vip2
sudo ifconfig vip2 192.168.111.1 netmask 255.255.255.0 up
sudo ip route add 172.16.0.0/22 via 192.168.111.2 dev vip2
sudo ip route add 172.16.1.0/24 via 192.168.111.2 dev vip2
sudo ip route add 172.16.2.0/24 via 192.168.111.2 dev vip2
sudo iptables -t nat -A POSTROUTING -o eth1 -j MASQUERADE
sudo iptables -A FORWARD -i vip2 -j ACCEPT
sudo iptables -A INPUT -i vip2 -j ACCEPT
gns3@gns3vm:~$
```

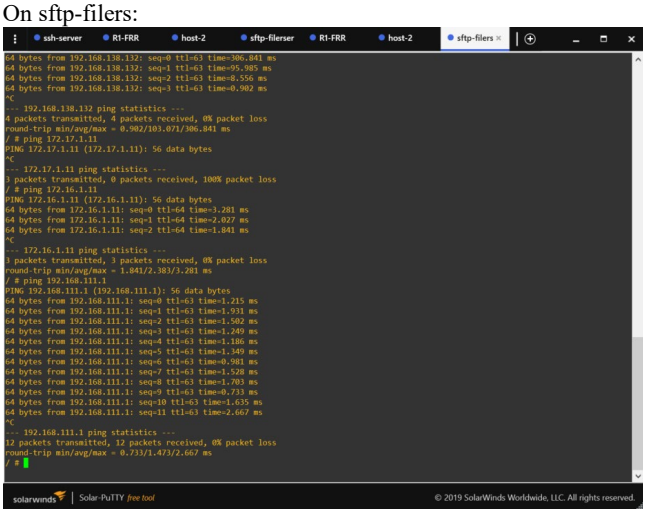
Then I need to add IP route configure in GNS3 VM via IpTables , in order to share connection with connected topology:

sudo iptables -t nat -A POSTROUTING -o eth1 -j MASQUERADE
sudo iptables -A FORWARD -i vip2 -j ACCEPT
sudo iptables -A INPUT -i vip2 -j ACCEPT

After configuration above, vip2 virutal network interface should now be available from GNS3 when run under your normal user account.

Then I tesed on topology to make sure they can communicat with each other:

On sftp-filers:

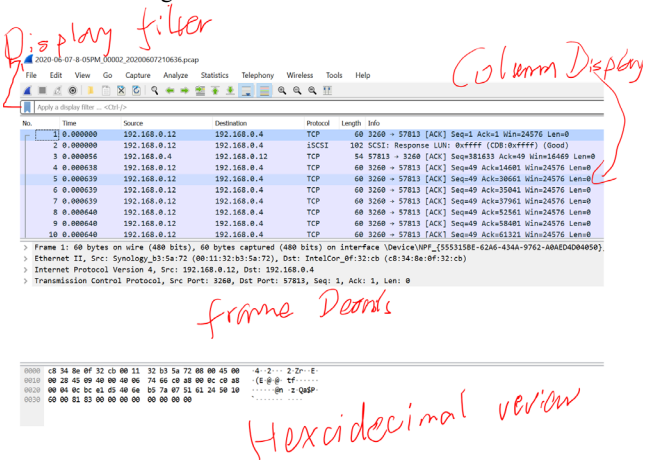


On GNS3 VM:

```
gns3@gns3vm: ~/assign3
gns3@gns3vm:~/assign3$ ping 192.168.111.2
PING 192.168.111.2 (192.168.111.2) 56(84) bytes of data.
64 bytes from 192.168.111.2: icmp_seq=1 ttl=64 time=0.254 ms
64 bytes from 192.168.111.2: icmp_seq=2 ttl=64 time=0.277 ms
64 bytes from 192.168.111.2: icmp_seq=3 ttl=64 time=0.275 ms
64 bytes from 192.168.111.2: icmp_seq=4 ttl=64 time=0.273 ms
64 bytes from 192.168.111.2: icmp_seq=5 ttl=64 time=0.229 ms
^C
--- 192.168.111.2 ping statistics ---
5 packets transmitted, 5 received, 0% packet loss, time 4065ms
rtt min/avg/max/mdev = 0.229/0.261/0.277/0.025 ms
gns3@gns3vm:~/assign3$ ping 172.16.1.11
PING 172.16.1.11 (172.16.1.11) 56(84) bytes of data.
64 bytes from 172.16.1.11: icmp_seq=1 ttl=63 time=1.64 ms
64 bytes from 172.16.1.11: icmp_seq=2 ttl=63 time=1.91 ms
64 bytes from 172.16.1.11: icmp_seq=3 ttl=63 time=1.84 ms
64 bytes from 172.16.1.11: icmp_seq=4 ttl=63 time=1.88 ms
^C
--- 172.16.1.11 ping statistics ---
4 packets transmitted, 4 received, 0% packet loss, time 3006ms
rtt min/avg/max/mdev = 1.642/1.822/1.913/0.106 ms
gns3@gns3vm:~/assign3$ ping 172.16.2.11
PING 172.16.2.11 (172.16.2.11) 56(84) bytes of data.
64 bytes from 172.16.2.11: icmp_seq=1 ttl=62 time=75.4 ms
64 bytes from 172.16.2.11: icmp_seq=2 ttl=62 time=18.1 ms
64 bytes from 172.16.2.11: icmp_seq=3 ttl=62 time=13.3 ms
^C
--- 172.16.2.11 ping statistics ---
3 packets transmitted, 3 received, 0% packet loss, time 2003ms
rtt min/avg/max/mdev = 13.332/35.670/75.496/28.231 ms
gns3@gns3vm:~/assign3$
```

2.1 Wireshark UI / Columns customization

Same as assignment 2, we want to customize the Wireshark UI a little for our assignment 3.

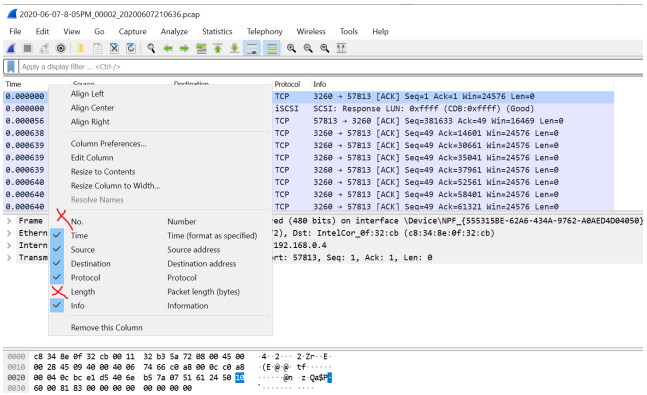



```
Flags: 0x010 (ACK)
   000. .... .... = Reserved: Not set
   ...0 .... .... = Nonce: Not set
   .... 0... .... = Congestion Window Reduced (CWR): Not set
   .... .0.. .... = ECN-Echo: Not set
   .... ..0. .... = Urgent: Not set
   .... ...1 .... = Acknowledgment: Set
   .... .... 0... = Push: Not set
   .... .... .0.. = Reset: Not set
   .... .... ..0. = Syn: Not set
   .... .... ...0 = Fin: Not set
   [TCP Flags: ·····A····]
```

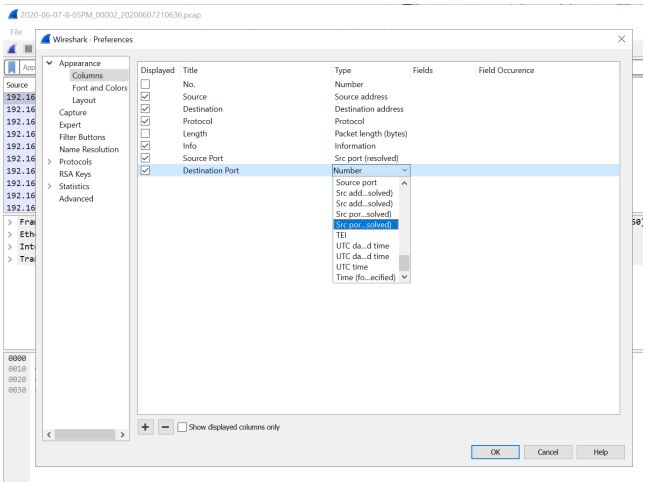
To be better to analyze the data, we might need extra columns for our convienent. Such as:

1. Source port
2. Desintation port
3. Timestamp
4. HTTP / HTTPS host
5. SNMP OID and Values

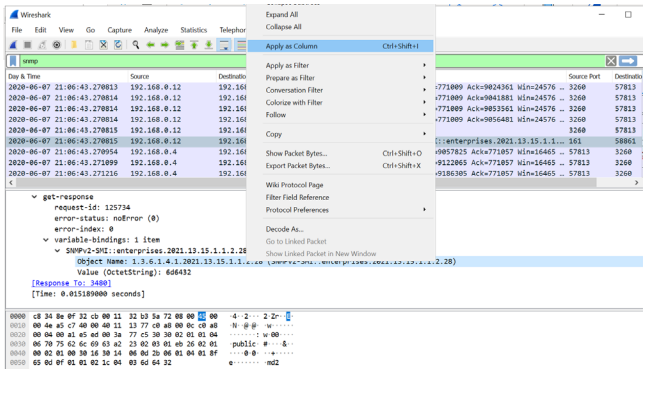
Remove unneccessary columns:



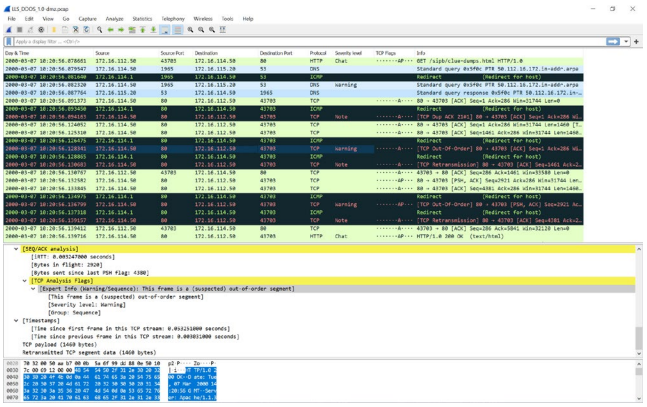
Adding wanted columns:



Adding custom columns (Src port, Dest port, TCP labels) as we need:

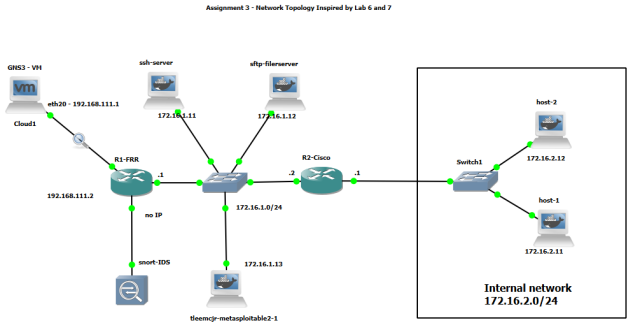


Customization result:



The screenshot above shows I successfully removed unnessaary columns, add my needed cusomized column Src port, Dest port and TCP labels. They can be prepared for data analytics in the later section.

2.2 Wireshark Monitoring



WireShark is our first step to sniff and monitor the packets during our detection period. It can help us with traffic identification and assurance. The screenshot above demonstrates the topology I will use for this investigation. It might be changed a little for complying with ACL testing in a later section.

2.3 Snort sniffing setup

On R1-FRR host, in order to transfer the traffic from GNS-VM to snort-IDS sever, we need to run the command on R1-FRR:

daemonlogger -i eth0 -o eth2

Because eth0 is the network interface to receive the traffic from GNS-VM and eth2 is the interface to talk with snort-IDS server.

Then on snort-IDS server side, we run the command:

snort -i eth0 -c /etc/snort/snort.conf -l /var/log/snort -A full

This command will sniff the packets on eth0, which is the interface connection with R1-FRR. Then it parses the configuration of snort to save the results in /var/log/snort folder.

It will generate the “alert” file, which contains alert information triggered by snort configuration. Rest are the general log files for sniffed packages information.

At this point, it is worth to take a look at the content of snort configuration, and explain a little:

2.3.1 Snore configuration usages

Snort is immensely powerful and popular network security monitoring and reporting tool. We can maximize the network vulnerabilities detection via Snort configuration. Snort configuration can do following things. Please note this section we are talk about original Snore configuration; I might have to adjust configuration to the attack. It also answers part of question 1 and 2.

Before our actual refinement of Snort configuration in the later section, it is worth to warm up with the original Snort configuration [6].

This file contains a sample snort configuration.
You should take the following steps to create your own custom configuration:
#
1) Set the network variables.
2) Configure the decoder
3) Configure the base detection engine
4) Configure dynamic loaded libraries
5) Configure preprocessors
6) Configure output plugins
7) Customize your rule set
8) Customize preprocessor and decoder rule set
9) Customize shared object rule set

Specifically, we can follow the general steps introduced by Lab 6 and 7 [6], like :

Step 1: Set the network variables.

In our case, it is 172.16.0.0/16
ipvar HOME_NET 172.16.0.0/16

Also, we indicate to receive all the traffic from everywhere:
ipvar EXTERNAL_NET any

We have a SSH server on our network:
ipvar SSH_SERVERS 172.16.1.2/32

Step 2: configure the decoder.

We do not have to check TCP general alerts, so we disable them:
Stop generic decode events:
config disable_decode_alerts

Stop Alerts on experimental TCP options
config disable_tcpopt_experimental_alerts

Stop Alerts on obsolete TCP options
config disable_tcpopt_obsolete_alerts

Stop Alerts on T/TCP alerts
config disable_tcpopt_ttcp_alerts

Stop Alerts on all other TCPOption type events:
config disable_tcpopt_alerts

Stop Alerts on invalid ip options
config disable_ipopt_alerts

However, we care about IP /TCP checksum:
config checksum_mode: all

Step 3: Configure the base detection engine.

Firstly, we need to configure PCRE match limitations:

config pcre_match_limit: 3500
config pcre_match_limit_recursion: 1500

Secondly, we configure the detection engine:

config detection: search-method ac-split search-optimize max-pattern-len 20

Lastly, we configure the event log:
config event_queue: max_queue 8 log 5 order_events content_length

Additionally, we set protocol asware flusing shreshould as 16000:
config paf_max: 16000

Step 4: we configure dynamic loaded libraries
This step is genrate configuration for configuring the working environment of Snort engine. So it is not related to alerts / events/ logs themselves.

```
# path to dynamic preprocessor libraries
dynamicpreprocessor directory /usr/local/lib/snort_dynamicpreprocessor/

# path to base preprocessor engine
dynamicengine /usr/local/lib/snort_dynamicengine/libsf_engine.so

# path to dynamic rules libraries
dynamicdetection directory /usr/local/lib/snort_dynamicrules
```

Step 5: We configure preprocessors
In this step, we normalize the packages information corresponding to different protocols, then we configure anomalies detection on different protocol.

```
preprocessor frag3_global: max_frags 65536
preprocessor frag3_engine: policy windows detect_anomalies
overlap_limit 10 min_fragment_length 100 timeout 180


# Target-Based stateful inspection/stream reassembly.  For more
inforation, see README.stream5
preprocessor stream5_global: track_tcp yes, \
  track_udp yes, \
  track_icmp no, \
  max_tcp 262144, \
  max_udp 131072, \
  max_active_responses 2, \
  min_response_seconds 5
preprocessor stream5_udp: timeout 180
```

Preprocessor also includes the part of requirements of our assignment. We can configure the performance statistics by sniffing the packets. In this part, we can configure inspection ports, parameters, heads etc.

Step 6: we configure output plugins

In this step, we can specify what kinds of output we need and what output files we need to generate. Like unified2, syslog and pcap dump files.

Step 7: we can customize our rule set
Original we include following rules for different attacks, later I will add my own rules to detect more potential issues, such as DoS / Port Scan:

```
include $RULE_PATH/local.rules
include $RULE_PATH/cs6706.rules
include $RULE_PATH/app-detect.rules
include $RULE_PATH/attack-responses.rules
include $RULE_PATH/backdoor.rules
include $RULE_PATH/bad-traffic.rules
include $RULE_PATH/ddos.rules
include $RULE_PATH/dns.rules
include $RULE_PATH/dos.rules
include $RULE_PATH/experimental.rules
include $RULE_PATH/exploit.rules
include $RULE_PATH/finger.rules
include $RULE_PATH/ftp.rules
include $RULE_PATH/icmp-info.rules
include $RULE_PATH/icmp.rules
include $RULE_PATH/imap.rules
include $RULE_PATH/info.rules
include $RULE_PATH/multimedia.rules
include $RULE_PATH/mysql.rules
include $RULE_PATH/netbios.rules
include $RULE_PATH/nntp.rules
include $RULE_PATH/oracle.rules
include $RULE_PATH/other-ids.rules
include $RULE_PATH/p2p.rules
include $RULE_PATH/policy.rules
include $RULE_PATH/pop2.rules
include $RULE_PATH/pop3.rules
include $RULE_PATH/rpc.rules
include $RULE_PATH/scan.rules
```

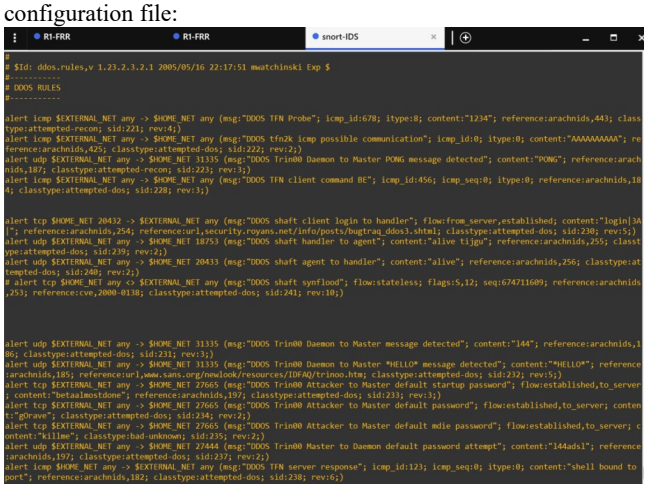
above MySQL and oracle might not do much with our assignment. We can consider removing them for performance purpose. However, the following rules need to be enabled, even they are coarse grained:

```
include $RULE_PATH/smtp.rules
include $RULE_PATH/snmp.rules
include $RULE_PATH/sql.rules
include $RULE_PATH/telnet.rules
include $RULE_PATH/tftp.rules
include $RULE_PATH/virus.rules
include $RULE_PATH/web-attacks.rules
include $RULE_PATH/web-cgi.rules
include $RULE_PATH/web-client.rules
include $RULE_PATH/web-coldfusion.rules
include $RULE_PATH/web-frontpage.rules
include $RULE_PATH/web-iis.rules
include $RULE_PATH/web-misc.rules
include $RULE_PATH/web-php.rules
include $RULE_PATH/x11.rules
```

Step 8, we might want to decoder alerts, but skip it at this moment.

Step 9: we can customize our shared object snort rules. They are do nothing about our assignments, we can level them by now.

For testing purpose, we can look at DDoS Rules specified in configuration file:



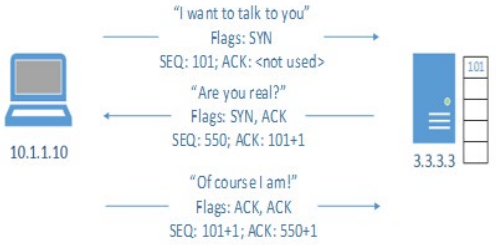
It tells how to decoder DDoS attacker packages information and generate alert information. It includes but not limited:

DDOS TFN Probe
DDOS tfn2k icmp possible communication
DDOS Trin00 Daemon to Master PONG
DDOS TFN client command BE
DDOS shaft client login to handler
DDOS shaft handler to agent
DDOS shaft agent to handler
DDOS shaft synflood
DDOS mstream agent to handler
DDOS mstream handler to agent
DDOS mstream handler ping to agent
DDOS mstream agent pong to handler
DDOS mstream client to handler
DDOS mstream handler to client
DDOS mstream client to handler
DDOS mstream handler to client
DDOS - TFN client command LE
DDOS Stacheldraht server spoof
DDOS Stacheldraht gag server response
DDOS Stacheldraht server response
DDOS Stacheldraht client spoofworks
DDOS Stacheldraht client check gag
DDOS Stacheldraht client check skillz
DDOS Stacheldraht handler->agent niggahbitch
DDOS Stacheldraht agent->handler skillz
DDOS Stacheldraht handler->agent ficken

Here, I feel interested in "syn-flood" and "icmp-flood" attack, so that I will keep an eye at this alert during Snort sniffing. Moreover, our Snort rules customization will surround DoS attack in the sections later.

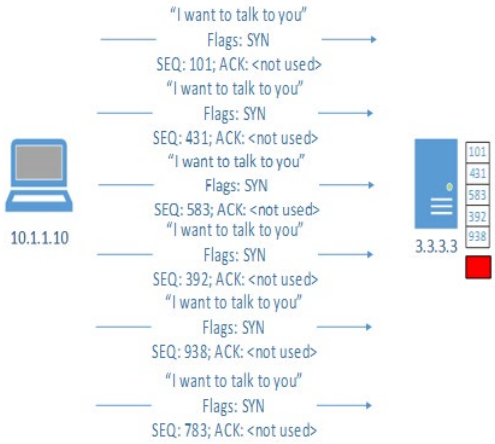
3 Question 1: Snort Detection of hands-on DDoS attack and DARPA Intrusion Detection Datasets

Before diving into this question, we need to recall SYN concepts[3]:



SYN is three ways hands sharking communication between server and client.

This type of communication can be compromised by malicious users to exploit its limited slots for pending connections. Like the screenshot below:



Note that during SYN flood the attacker usually spoofs and varies the source IP address, usually using PRNG, thus preventing the victim from inserting an IP filter for that IP address.

Since the attacker is spoofing the sources it is not easily possible to determine the number of attacking machines and it may also be just a single machine. One way to have an idea about the distance is to look at the packet TTL but be aware it can also be forged [6].

I made extra efforts on this question, besides testing existing datasets, I simulated DDoS attack for initial test.

3.1 Issuing the DDoS attack.

Before utilizing download files, I issued DDoS attack to test our topology a little bit.

Hping3 is a command-line oriented TCP/IP packet assembler/analyzer. The interface is inspired to the ping(8) Unix command, but Hping3 isn't only able to send ICMP echo requests. It supports TCP, UDP, ICMP and RAW-IP protocols, has a traceroute mode, the ability to send files between a covered channel, and many other features [9].

Basically, I am going to issue "**hping3**" command to simulate DDoS attack. However, before doing that, I was aware of SYN cookie configuration might impact on DDoS attack, because "**net.ipv4.tcp_syncookies = 1"** will partially mitigate the DDoS attack. So, we should disable syncookies at this point.

- Start up daemonlogger at R1-FRR:

daemonlogger -i eth0 -o eth2

```
/ # daemonlogger -i eth0 -o eth2
[-] Interface set to eth0
[-] Log filename set to "daemonlogger.pcap"
[-] Tap output interface set to eth2[-] Pidfile configured to "daemonlogger.pid"
[-] Pidpath configured to "/var/run"
[-] Rollover size set to 18446744071562067968 bytes
[-] Rollover time configured for 0 seconds
[-] Pruning behavior set to oldest IN DIRECTORY

-*> DaemonLogger <*-
Version 1.2.1
By Martin Roesch
(C) Copyright 2006-2007 Sourcefire Inc., All rights reserved

sniffing on interface eth0
```

We execute this command to sniff traffic coming from R1-FRR at snort-IDS now:

snort -i eth0 -c /etc/snort/snort.conf -l /var/log/snort -A full

```
        --== Initialization Complete ==--

   ,,_     -*> Snort! <*-
  o"  )~   Version 2.9.11.1 GRE (Build 268)
   ''''    By Martin Roesch & The Snort Team: http://www.snort.org/contact#team
           Copyright (C) 2014-2017 Cisco and/or its affiliates. All rights reserved.
           Copyright (C) 1998-2013 Sourcefire, Inc., et al.
           Using libpcap version 1.7.4
           Using PCRE version: 8.38 2015-11-23
           Using ZLIB version: 1.2.8

           Rules Engine: SF_SNORT_DETECTION_ENGINE  Version 3.0  <Build 1>
           Preprocessor Object: SF_SSLPP  Version 1.1  <Build 4>
           Preprocessor Object: SF_DNP3  Version 1.1  <Build 1>
           Preprocessor Object: SF_SIP  Version 1.1  <Build 1>
           Preprocessor Object: SF_MODBUS  Version 1.1  <Build 1>
           Preprocessor Object: SF_IMAP  Version 1.0  <Build 1>
           Preprocessor Object: SF_GTP  Version 1.1  <Build 1>
           Preprocessor Object: SF_DCERPC2  Version 1.0  <Build 3>
           Preprocessor Object: SF_FTPTELNET  Version 1.2  <Build 13>
           Preprocessor Object: SF_POP  Version 1.0  <Build 1>
           Preprocessor Object: SF_SDF  Version 1.1  <Build 1>
           Preprocessor Object: SF_SSH  Version 1.1  <Build 3>
           Preprocessor Object: SF_DNS  Version 1.1  <Build 4>
           Preprocessor Object: SF_SMTP  Version 1.1  <Build 9>
           Preprocessor Object: SF_REPUTATION  Version 1.1  <Build 1>
Commencing packet processing (pid=3863)
```

Next, we will try to execute the following command to simulate DDoS attack to our ftp file server (172.16.1.12) on our topology network.

Just recall, there are two general types of DoS attack:

ICMP flood attack:

```
sudo hping3 -i vip2 -V 172.16.1.13
```

```
gns3@gns3vm:~$ sudo hping3 --flood -c 10000 -i vip2 --rand-source -V 172.16.1.13

using vip2, addr: 192.168.111.1, MTU: 1500
HPING 172.16.1.13 (vip2 172.16.1.13): NO FLAGS are set, 40 headers + 0 data byte
s
hping in flood mode, no replies will be shown
^C
--- 172.16.1.13 hping statistic ---
50056364 packets transmitted, 0 packets received, 100% packet loss
round-trip min/avg/max = 0.0/0.0/0.0 ms
```

SYN flood attack:

```
sudo hping3 -S --flood -c 10000 -d 128 -w 64 -i vip2 --rand-source
-V -p 80 172.16.1.13
```

Nit is good practice to explain the options of "hping3" command to better understand malicious users' behaviors:

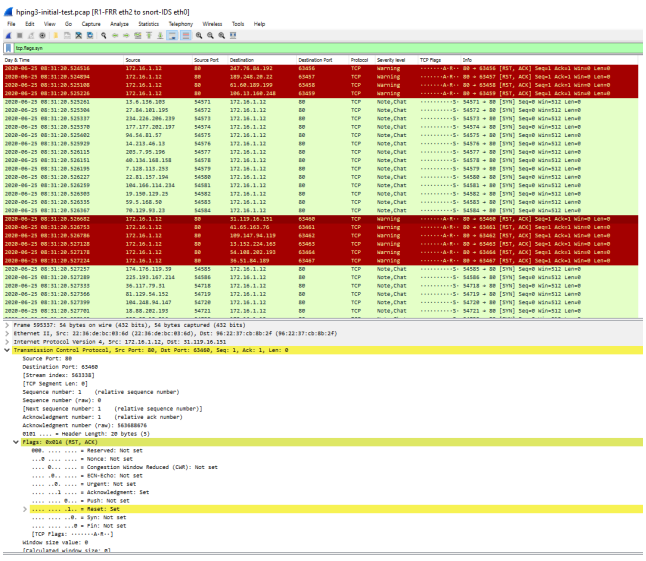
- **-S: assign SYN flag**
- **--flood: work as flood mode**
- **-c send 10000 pkgs**
- **-d 128: each pkg size**
- **-w: tcp windows**
- **-i: targeted network interface**
- **--rand-source: simulate DDoS via randomized source IP**
- **-V: output verbose detailed**
- **-p: port number we are attacking**
- **And destination IP: 172.16.1.13**

Now let us try SYN flood attack, later on I will test both after refining Snort rules, so that **I saved results to hping3-initial-test.pcap, which can be replayed for comparison between Snort rules modification before and after**:

```
gns3@gns3vm:~/assign3$ sudo tcpreplay -i vip2  -K hping3-initial-test.pcap
File Cache is enabled
Actual: 251649 packets (35187106 bytes) sent in 67.74 seconds
Rated: 519407.6 Bps, 4.15 Mbps, 3714.66 pps
Flows: 250464 flows, 3697.17 fps, 251635 flow packets, 14 non-flow
Statistics for network device: vip2
        Successful packets:        251649
        Failed packets:            0
        Truncated packets:         0
        Retried packets (ENOBUFS): 0
        Retried packets (EAGAIN):  0
gns3@gns3vm:~/assign3$
```

Above, I was trying to send 10000 packages over port 80 using random source IP address to our metasploitable2 within our network topology.

We can capture the traffic between R1-FRR and snort-IDS by Wireshark firstly:



We can say there are many abnormal unhealthy traffic were shown in wireshark UI.

To make it more precisely, I directly exectue the command "netstat" on vulerable targetted server "metasploitable2":

```
root@tleemcjr-metasploitable2-1:/# netstat
Active Internet connections (w/o servers)
Proto Recv-Q Send-Q Local Address           Foreign Address         State
tcp        0      0 172.16.1.13:www         75.74.110.159:1749      SYN_RECV
tcp        0      0 172.16.1.13:www         209.194.36.253:1738     SYN_RECV
tcp        0      0 172.16.1.13:www         103.124.250.210:1730    SYN_RECV
tcp        0      0 172.16.1.13:www         136.207.56.221:1634     SYN_RECV
tcp        0      0 172.16.1.13:www         119.32.1.5:1725         SYN_RECV
tcp        0      0 172.16.1.13:www         1.90.79.26:sa-msg-por   SYN_RECV
tcp        0      0 172.16.1.13:www         117.30.34.5:1663        SYN_RECV
tcp        0      0 172.16.1.13:www         77.237.45.195:1756      SYN_RECV
tcp        0      0 172.16.1.13:www         144.68.2.132:1734       SYN_RECV
tcp        0      0 172.16.1.13:www         118.31.36.49:1647       SYN_RECV
tcp        0      0 172.16.1.13:www         203.0.108.247:1683      SYN_RECV
tcp        0      0 172.16.1.13:www         184.68.220.28:1741      SYN_RECV
tcp        0      0 172.16.1.13:www         13.58.109.254:1681      SYN_RECV
tcp        0      0 172.16.1.13:www         163.162.244.113:1750    SYN_RECV
tcp        0      0 172.16.1.13:www         188.207.5.184:1768      SYN_RECV
tcp        0      0 172.16.1.13:www         76.216.156.216:1650     SYN_RECV
tcp        0      0 172.16.1.13:www         209.215.115.93:1711     SYN_RECV
tcp        0      0 172.16.1.13:www         243.89.145.50:1706      SYN_RECV
tcp        0      0 172.16.1.13:www         195.128.72.74:1760      SYN_RECV
tcp        0      0 172.16.1.13:www         203.246.216.145:1762    SYN_RECV
tcp        0      0 172.16.1.13:www         58.71.196.189:1753      SYN_RECV
tcp        0      0 172.16.1.13:www         44.65.166.92:1669       SYN_RECV
tcp        0      0 172.16.1.13:www         221.227.65.253:1666     SYN_RECV
```

Let us recall SYN possible states of a connection are as follows[5]:

ESTABLISHED - Both hosts are connected.
CLOSING - The remote host has agreed to close its connection.
LISTENING - Your computer is waiting to handle an incoming connection.
SYN_RCVD - A remote host has asked for you to start a connection.
SYN_SENT - Your computer has accepted to start a connection.
LAST_ACK - Your computer needs to obliterate (i.e. erase from memory) the packets before closing the connection.
TIMED_WAIT - See above.
CLOSE_WAIT - The remote host is closing its connection with your computer.
FIN_WAIT 1 - A client is closing its connection.
FIN_WAIT 2 - Both hosts have agreed to close the connection.

From the screenshot of "netstats" results on victim server. I can see that many connections stay in SYN_RECV state. And source IP are randomized to be simulated from different locations. The waiting time is the time between receiving the first and third packet in the 3-way handshake. It has been proved 5-6 packets stay in the same state SYN_RECV is overly aggressive case. The health situation is that only maximum 1/5 of the sockets should be in in SYN_RECV and the majority should be in ESTABLISHED or TIME_WAIT state.

Therefor, I can tell something is abnormal when I observe many SYN_RECV packets constantly. In conclusion, having many SYN RECV packets means a SYN Flood.

Next, I want to check Snort Statistics outputs on terminal console to hopefully identify those anomalies.

Packets I/O Total:

```
===============================================================================
Run time for packet processing was 249.602703 seconds
Snort processed 141183 packets.
Snort ran for 0 days 0 hours 4 minutes 9 seconds
   Pkts/min:        35295
   Pkts/sec:          567
===============================================================================
Memory usage summary:
  Total non-mmapped bytes (arena):       31608832
  Bytes in mapped regions (hblkhd):      29995008
  Total allocated space (uordblks):      23268816
  Total free space (fordblks):           8340016
  Topmost releasable block (keepcost):   2011424
===============================================================================
Packet I/O Totals:
   Received:       148010
   Analyzed:       141183 ( 95.387%)
    Dropped:         6827 (  4.409%)
   Filtered:            0 (  0.000%)
Outstanding:         6827 (  4.613%)
   Injected:            0
===============================================================================
```

Protocol Breakdowns

```
===============================================================================
Breakdown by protocol (includes rebuilt packets):
        Eth:       141183 (100.000%)
       VLAN:            0 (  0.000%)
        IP4:       141163 ( 99.986%)
       Frag:            0 (  0.000%)
       ICMP:            0 (  0.000%)
        UDP:            0 (  0.000%)
        TCP:       141163 ( 99.986%)
        IP6:            1 (  0.001%)
    IP6 Ext:            1 (  0.001%)
   IP6 Opts:            0 (  0.000%)
      Frag6:            0 (  0.000%)
     ICMP6:             1 (  0.001%)
      UDP6:             0 (  0.000%)
      TCP6:             0 (  0.000%)
    Teredo:             0 (  0.000%)
   ICMP-IP:             0 (  0.000%)
   IP4/IP4:             0 (  0.000%)
   IP4/IP6:             0 (  0.000%)
   IP6/IP4:             0 (  0.000%)
   IP6/IP6:             0 (  0.000%)
       GRE:             0 (  0.000%)
   GRE Eth:             0 (  0.000%)
  GRE VLAN:             0 (  0.000%)
   GRE IP4:             0 (  0.000%)
   GRE IP6:             0 (  0.000%)
GRE IP6 Ext:            0 (  0.000%)
   GRE PPTP:            0 (  0.000%)
   GRE ARP:             0 (  0.000%)
   GRE IPX:             0 (  0.000%)
  GRE Loop:             0 (  0.000%)
      MPLS:             0 (  0.000%)
       ARP:            19 (  0.013%)
       IPX:             0 (  0.000%)
  Eth Loop:             0 (  0.000%)
  Eth Disc:             0 (  0.000%)
  IP4 Disc:             0 (  0.000%)
  IP6 Disc:             0 (  0.000%)
  TCP Disc:             0 (  0.000%)
  UDP Disc:             0 (  0.000%)
 ICMP Disc:             0 (  0.000%)
All Discard:            0 (  0.000%)
      Other:            0 (  0.000%)
Bad Chk Sum:            0 (  0.000%)
    Bad TTL:            0 (  0.000%)
     S5 G 1:            0 (  0.000%)
     S5 G 2:            0 (  0.000%)
      Total:       141183
===============================================================================
```

Action Stats:

```
===============================================================================
Action Stats:
     Alerts:          227 (  0.161%)
     Logged:          227 (  0.161%)
     Passed:            0 (  0.000%)
Limits:
      Match:            0
      Queue:            0
        Log:            0
      Event:            0
      Alert:            0
Verdicts:
      Allow:       141183 ( 95.387%)
      Block:            0 (  0.000%)
    Replace:            0 (  0.000%)
  Whitelist:            0 (  0.000%)
  Blacklist:            0 (  0.000%)
     Ignore:            0 (  0.000%)
      Retry:            0 (  0.000%)
===============================================================================
```

Stream statistics:

```
Stream statistics:
          Total sessions: 84483
            TCP sessions: 84483
            UDP sessions: 0
           ICMP sessions: 0
             IP sessions: 0
              TCP Prunes: 0
              UDP Prunes: 0
             ICMP Prunes: 0
               IP Prunes: 0
TCP StreamTrackers Created: 84483
TCP StreamTrackers Deleted: 84483
             TCP Timeouts: 0
             TCP Overlaps: 0
      TCP Segments Queued: 0
    TCP Segments Released: 0
      TCP Rebuilt Packets: 0
       TCP Segments Used: 0
           TCP Discards: 0
                TCP Gaps: 0
     UDP Sessions Created: 0
     UDP Sessions Deleted: 0
           UDP Timeouts: 0
           UDP Discards: 0
                 Events: 58811
        Internal Events: 0
        TCP Port Filter
                Filtered: 0
               Inspected: 0
                 Tracked: 141163
        UDP Port Filter
                Filtered: 0
               Inspected: 0
                 Tracked: 0
```

From above screenshots, I found there were 227 (0.16%) alerts reported at the end. Then, we might want to visit snort logs a little bit under /var/log/snort:

root@snort-IDS:/var/log/snort# cat alert | more

```
root@snort-IDS:/var/log/snort# cat alert | grep "Priority: 1" | sort | uniq -c
root@snort-IDS:/var/log/snort# cat alert | grep "Priority: 2" | sort | uniq -c
    227 [Classification: Potentially Bad Traffic] [Priority: 2]
root@snort-IDS:/var/log/snort# cat alert | grep "Priority: 3" | sort | uniq -c
root@snort-IDS:/var/log/snort#
```

We can see above there is no alerts higher than Priority 2 reported, therefore, we need to improve the snort rules in later section. The only 227 bad traffic alerts are quite general information. And **it does not explain the situation on victim server why there are so many "SYN_RECV" stated packets. We need more specific alert setting up for potential and hidden issues**. So, we are going improve it in question 2.

Then, I saved this PCAP file into assignment 3 folder, which is shared on OneDrive:

- hping3-initial.pcap

Next, we want to download a suggested dataset from DARPA Intrusion Detection Datasets.

3.2 DARPA Intrusion Detection Datasets.

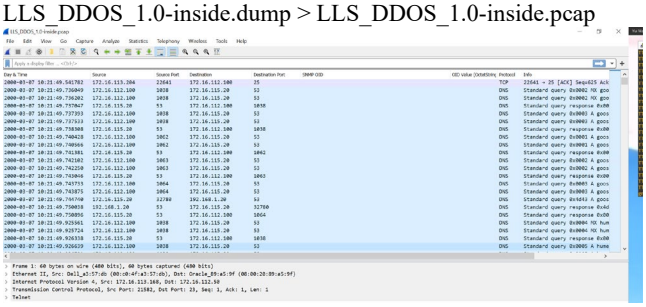
https://archive.ll.mit.edu/ideval/data/2000data.html

2000 DARPA Intrusion Detection Scenario Specific Data Sets
The content and labeling of data sets relies significantly on reports and feedback from consumers of this data. Please send feedback on this data set to Joshua W. Haines so that your ideas can be incorporated into future data sets. Thanks!

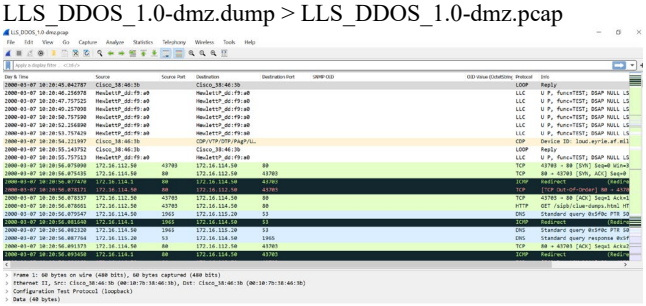
Overview
Off-line intrusion detection datasets were produced as per consensus from the Wisconsin Re-Think meeting and the July 2000 Hawaii PI meeting.
LLDOS 1.0 - Scenario One
This is the first attack scenario data set to be created for DARPA as a part of this effort. It includes a distributed denial of service attack run by a novice attacker. Future versions of this and other example scenarios will contain more stealthy attack versions.

This attack scenario is carried out over multiple network and audit sessions. These sessions have been grouped into 5 attack phases, over the course of which the attacker probes the network, breaks in to a host by exploiting the Solaris sadmind vulnerability, installs trojan mstream DDoS software, and launches a DDoS attack at an off site server from the comprismised host.

LLS_DDOS_1.0-inside.dump > LLS_DDOS_1.0-inside.pcap



LLS_DDOS_1.0-dmz.dump > LLS_DDOS_1.0-dmz.pcap



Uploaded PCAP files to GNS3 VM:

```
D:\assign3-datasets>scp LLS_DDOS_1.0-inside.pcap gns3@192.168.234.130:~/assign3/
gns3@192.168.234.130's password:
LLS_DDOS_1.0-inside.pcap                                    100%  116MB   2.1MB/s   00:54

D:\assign3-datasets>scp LLS_DDOS_1.0-dmz.pcap gns3@192.168.234.130:~/assign3/
gns3@192.168.234.130's password:
LLS_DDOS_1.0-dmz.pcap                                       100%   85MB   4.4MB/s   00:19

D:\assign3-datasets>
```

Those generated datasets were saved on my shared OneDrive folder [10].

Next, we want to utilize the techniques we learned from Assignment 2 to visualize and analyze the original datasets a little bit.

We also can run the command "tshark" to get a host list for the security check later on in the later section.

```
D:\assign3-datasets>tshark -r LLS_DDOS_1.0-inside_new_columns.pcap -q -z hosts > LLS_DDOS_1.0-inside.hosts
```

```
D:\assign3-datasets>tshark -r LLS_DDOS_1.0-inside_new_columns.pcap -q -z hosts
# TShark hosts output
#
# Host data gathered from LLS_DDOS_1.0-inside_new_columns.pcap

208.237.193.111 www.apartments.com
134.205.165.120 afpubs.hq.af.mil
204.71.191.203  ad.linkexchange.com
205.181.112.70  computershopper.zdnet.com
192.254.26.2    www.gtsi.com
204.123.2.94    jump.altavista.com
209.185.69.11   altavista.looksmart.com
208.141.130.161 ShopWorks.com
207.25.71.141   cnnsi.com
209.1.12.219    www.inktomi.com
209.67.227.100  www.mapsonus.com
192.215.17.71   img.cmpnet.com
205.153.28.61   freedom.charter.com
197.182.91.233  mars.avocado.net
205.188.146.109 ads.web.aol.com
209.1.224.190   pic.geocities.com
128.197.100.11  bumetb.bu.edu
32.97.253.109   search.borders.com
207.25.71.200   www.cnn.com
192.102.198.160 www.intel.com
135.8.60.182    beta.banana.edu
```

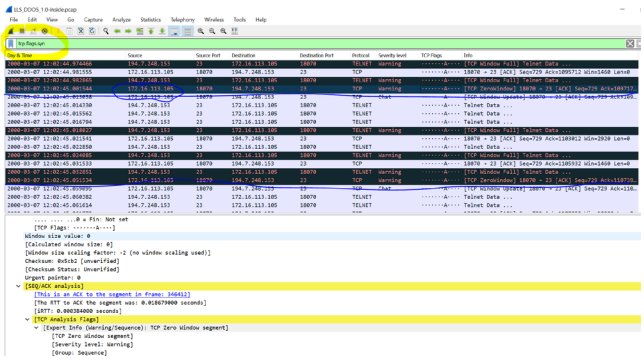
The result is quite long, I put in LLS_DDOS_1.0-inside.hosts which can be analyzed later on. We will do the same on LLS_DDOS_1.0-dmz_new_c0lumns.pcap.

```
D:\assign3-datasets>tshark -r LLS_DDOS_1.0-dmz_new_c0lumns.pcap -q -z hosts > LLS_DDOS_1.0-dmz_new_c0lumns.hosts

D:\assign3-datasets>tshark -r LLS_DDOS_1.0-dmz_new_c0lumns.pcap -q -z hosts
# TShark hosts output
#
# Host data gathered from LLS_DDOS_1.0-dmz_new_c0lumns.pcap

172.16.113.207  pigeon.eyrie.af.mil
132.52.128.21   www.vnc.aetc.af.mil
202.77.162.213  ppp5-213.att.net.hk
193.114.243.70  www.sbcwarburg.com
207.217.78.131  corp-img.earthlink.net
143.166.224.32  commerce.us.dell.com
207.46.150.42   bbs.msnbc.com
149.44.2.14     www.ctp.com
199.181.172.200 images2.nytimes.com
194.27.251.21   gamma.grape.mil
205.181.112.67  ads2.zdnet.com
196.227.33.20   ns.kiwi.org
207.87.128.113  counter.digits.com
199.94.99.2     uan.mitre.org
207.178.129.82  www.the-e-bus.com
159.133.105.3   www.tso.osd.mil
198.3.98.160    my.excite.com
203.103.130.23  www.yahoo.com.au
192.56.219.60   uswestdex.com
```

3.2.1 Wireshark UI Examine

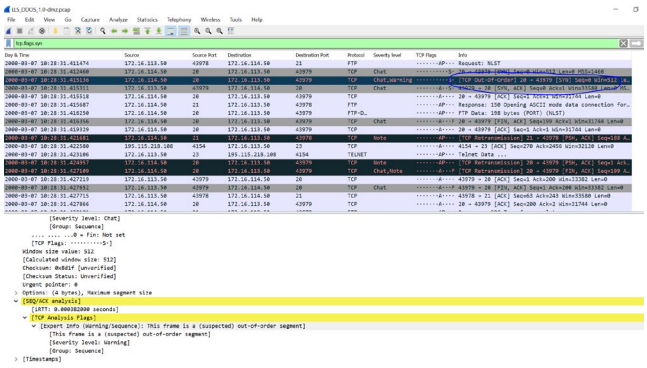
Once we have customized UI setup, we can start up a little bit simple data analytics (We will talk about it specifically in the later section).

For example, if we want to identify the suspicious devices which joined in our network during the monitoring period. We can track tcp.flags.syn activitity.



From the screenshot above, we firstly filter the keyword "**tcp.flags.syn**" , which is the criteria to examine the SYN issue. We loaded "LLS_DDOS_1.0-inside.dump" which originally downloaded from "**DARPA Intrusion Detection Datasets**". We can identify there were couple of suspicous activivities caused "TCP FULL window" or "TCP Zero windows". And the IP address seems unrecoginzed and randomzied. Therefore it is worth to continously investigated.

Let us take a look at another datasets "LLS_DDOS_1.0-dmz.dump":



From the screenshot above, we can identify many repeated "TCP out-of-order" warning happened on SYN communication.

Next, we want to use Argus and Visualizaiton tools to study the datasets a little bit.

3.2.2 Data Pre-processing

From the generated data (especially by WireShark), I utilized Argus command line to convert PCAP files into ARGUS files firstly, so that we can prepare for the data analytics. For example:

sudo argus -r LLS_DDOS_1.0-dmz.pcap -w LLS_DDOS_1.0-dmz.argus

Furthermore, we can consider about utilizing the RA command filter and fields parameters to generate customized statistical results:

Specify the fields to print. Ra uses a default printing field list, by specifying a field you can replace this list completely, or you can modify the existing default print list, using the optional '-' and '+[#]' form of the command. The available fields to print are [17]:

startime, lasttime, count, dur, avgdur,
saddr, daddr, proto, sport, dport, ipid,
stos, dtos, sttl, dttl, bytes, sbytes, dbytes,
pkts, spkts, dpkts, load, loss, rate,
srcid, ind, mac, dir, jitter, status, user,
win, trans, seq, vlan, mpls

I was doing this process for all the records:

- **Convert Wireshark PCAP data into ARGUS**

sudo argus -r LLS_DDOS_1.0-dmz.pcap -w LLS_DDOS_1.0-dmz.argus

sudo argus -r LLS_DDOS_1.0-inside.pcap -w LLS_DDOS_1.0-inside.argus

- **Customize the format of output and save into CSV file**

ra -c , -r LLS_DDOS_1.0-dmz.argus -s +label:20 +load +count + dur +mac +win + trans + mpls +rate +stos + dtos +sttl +dttl > LLS_DDOS_1.0-dmz.csv

ra -c , -r LLS_DDOS_1.0-inside.argus -s +label:20 +load +count + dur +mac +win + trans + mpls +rate +stos + dtos +sttl +dttl > LLS_DDOS_1.0-inside.csv

```
gns3@gns3vm:~/assign3$ ra -c, -r LLS_DDOS_1.0-dmz.argus -s +label:20 +load +count +dur
+mac +win +trans +mpls +rate +stos + dtos +sttl +dttl > LLS_DDOS_1.0-dmz.csv
```

Then I was planning to visualize the results based on the data analyzing of Argus. The processed Argus files were converted into CSV formatted files, and they were uploaded to OneDrive[12]:

We can go to the next step to visualize the data to make the data more understandable and eventually transform the data into information to the user.

Here I want to exploit the power of Python preprocess the data and feed preprocessed data into Tableau, which is one of the most popular data visualization tools on the market, chart to compare three days traffic. And furthermore, we even can break it down into more subtle comparison among the applications and devices. which is attached in this assignment. We notice the data combined with several major features:

```
df_lls_ddos = pd.read_csv(io.BytesIO(uploaded['LLS_DDOS_1.0-dmz.csv']))
df_lls_ddos.head()
```

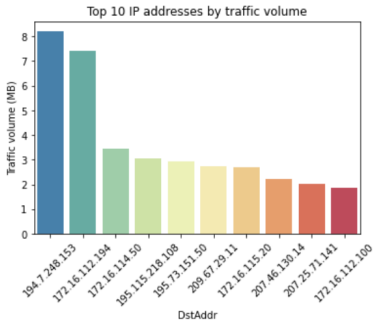
	StartTime	Flgs	Proto	SrcAddr	Sport	Dir	DstAddr	Dport	TotPkts	TotBytes	State	Label	Load	Dur	Trans	Rate
0	04:01:49.224899		man	0	0	NaN	0	0	0	0	STA	NaN	0.000000	0.015517	NaN	0.000000
1	17:20:45.042787	e	loop	00:10:7b:38:46:3b	NaN	->	00:10:7b:38:46:3b	NaN	1	60	INT	NaN	0.000000	0.000000	1.0	0.000000
2	17:20:46.256878	*	llc	08:00:09:dd:f9:a0	netbios	->	08:00:09:dd:f9:a0	NaN	4	216	INT	NaN	287.960846	4.500612	1.0	0.666576
3	17:20:52.256890	*	llc	08:00:09:dd:f9:a0	netbios	->	08:00:09:dd:f9:a0	NaN	4	216	REQ	NaN	259.202850	4.999945	1.0	0.600007
4	17:20:54.221997	*	llc	00:10:7b:38:46:3b	snap	->	01:00:0c:cc:cc:cc	snap	1	320	INT	NaN	0.000000	0.000000	1.0	0.000000

The screenshot above shows the datasets finally includes following fields:

Index(['**StartTime**', '**Flgs**', '**Proto**', '**SrcAddr**', '**Sport**', '**Dir**', '**DstAddr**', '**Dport**', '**TotPkts**', '**TotBytes**', '**State** ' , '**Load**', '**Dur**', '**Trans**', '**Rate**', '**sTos**', '**dTos**', '**sTtl**', '**dTtl**', '**Label**'],
dtype='object')

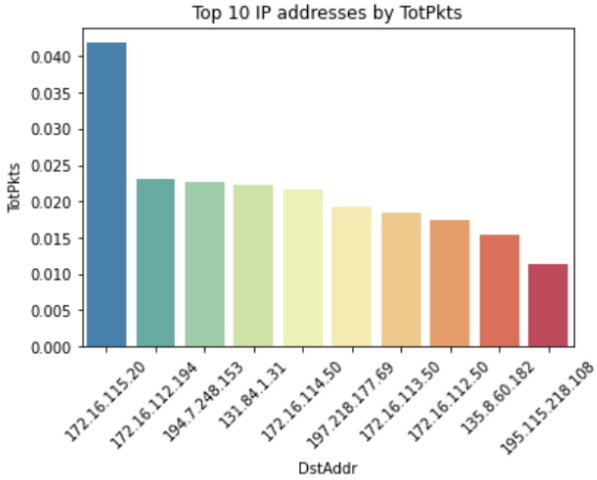
We noticed there is “label” field. It can be used for classification. There are NA values in the dataframe, we need to cleanup the data firstly, I did it via Python Pandas Lib. There are several data cleanup processes, like dropna(), drop unnecessary columns etc.

```
def srcAddTraffic(df):
  f, ax = plt.subplots(1, 1,figsize=(6, 4))

  per_ip = df.groupby("DstAddr", as_index=False).agg({"TotBytes":[np.sum]})

  p = per_ip.sort_values(by=("TotBytes","sum"),ascending=False)[:10]
  p["TotBytes","sum"] = p["TotBytes","sum"]/1000000
  sns.barplot(p["DstAddr"], p["TotBytes","sum"],palette="Spectral_r", ax=ax)

  ax.set_xticklabels(ax.xaxis.get_majorticklabels(), rotation=45)
  ax.set_ylabel("Traffic volume (MB)")
  ax.set_title("Top 10 IP addresses by traffic volume")

srcAddTraffic(df_lls_ddos)
```



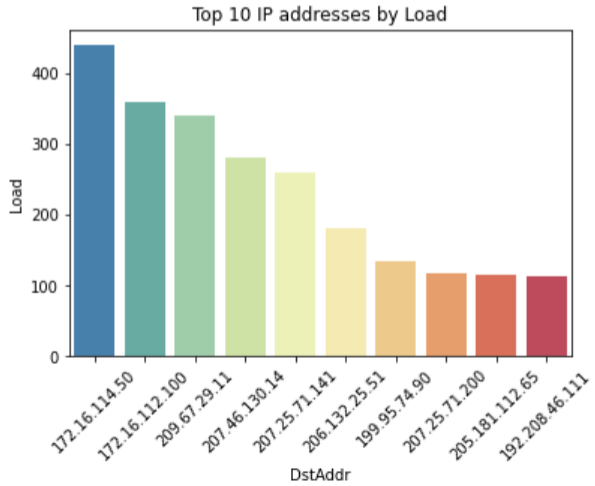
Top packages ranked by destination address

	DstAddr	TotPkts
		sum
4478	172.16.115.20	41933
3708	172.16.112.194	23179
4834	194.7.248.153	22623
3164	131.84.1.31	22318
4312	172.16.114.50	21597
4842	197.218.177.69	19162
4057	172.16.113.50	18362
3803	172.16.112.50	17346
3180	135.8.60.182	15442
4836	195.115.218.108	11387



Ranked by load

	DstAddr	Load sum
4312	172.16.114.50	4.398420e+08
3605	172.16.112.100	3.597730e+08
5280	209.67.29.11	3.409950e+08
5210	207.46.130.14	2.813516e+08
5196	207.25.71.141	2.603720e+08
5161	206.132.25.51	1.807246e+08
4864	199.95.74.90	1.355599e+08
5202	207.25.71.200	1.171785e+08
5144	205.181.112.65	1.163783e+08
4816	192.208.46.111	1.132706e+08



Next we need to utilize the Tcprewrite to rewrite the desitination address in datasets. Let it can work with our cusomized topology.

3.2.3 Rewrite the destination address

Tcprewrite usage:

```
gns3@gns3vm: ~/assign3
  -r, --portmap=str           Rewrite TCP/UDP ports
  -s, --seed=num              Randomize src/dst IPv4/v6 addresses w/ given seed
  -N, --pnat=str              Rewrite IPv4/v6 addresses using pseudo-NAT
  -S, --srcipmap=str          Rewrite source IPv4/v6 addresses using pseudo-NAT
  -D, --dstipmap=str          Rewrite destination IPv4/v6 addresses using pseudo-NAT
  -e, --endpoints=str         Rewrite IP addresses to be between two endpoints
  -b, --skipbroadcast         Skip rewriting broadcast/multicast IPv4/v6 addresses
  -C, --fixcsum               Force recalculation of IPv4/TCP/UDP header checksums
  -m, --mtu=num               Override default MTU length (1500 bytes)
      --mtu-trunc             Truncate packets larger then specified MTU
  -E, --efcs                  Remove Ethernet checksums (FCS) from end of frames
      --ttl=str               Modify the IPv4/v6 TTL/Hop Limit
      --tos=num               Set the IPv4 TOS/DiffServ/ECN byte
      --tclass=num            Set the IPv6 Traffic Class byte
      --flowlabel=num         Set the IPv6 Flow Label
  -F, --fixlen=str            Pad or truncate packet data to match header length
      --fuzz-seed=num         Fuzz 1 in X packets.  Edit bytes, length, or emulate packet drop
      --fuzz-factor=num       Set the Fuzz 1 in X packet ratio (default 1 in 8 packets)
      --skipl2broadcast       Skip rewriting broadcast/multicast Layer 2 addresses
      --dlt=str               Override output DLT encapsulation
      --enet-dmac=str         Override destination ethernet MAC addresses
      --enet-smac=str         Override source ethernet MAC addresses
      --enet-subsmac=str      Substitute MAC addresses
      --enet-mac-seed=num     Randomize MAC addresses
      --enet-mac-seed-keep-bytes=num Randomize MAC addresses
      --enet-vlan=str         Specify ethernet 802.1q VLAN tag mode
      --enet-vlan-tag=num     Specify the new ethernet 802.1q VLAN tag value
      --enet-vlan-cfi=num     Specify the ethernet 802.1q VLAN CFI value
      --enet-vlan-pri=num     Specify the ethernet 802.1q VLAN priority
      --hdlc-control=num      Specify HDLC control value
      --hdlc-address=num      Specify HDLC address
      --user-dlt=num          Set output file DLT type
      --user-dlink=str        Rewrite Data-Link layer with user specified data
  -d, --dbug=num              Enable debugging output
  -i, --infile=str            Input pcap file to be processed
  -o, --outfile=str           Output pcap file
  -c, --cachefile=str         Split traffic via tcpprep cache file
  -v, --verbose               Print decoded packets via tcpdump to STDOUT
  -A, --decode=str            Arguments passed to tcpdump decoder
      --skip-soft-errors      Skip writing packets with soft errors
  -V, --version               Print version information
  -h, --less-help             Display less usage information and exit
  -H, --help                  display extended usage information and exit
  -!, --more-help             extended usage information passed thru pager
      --save-opts[=arg]       save the option state to a config file
      --load-opts=str         load options from a config file
```

there are multiple usages to rewrite the destiantion IP, such as:

- Destipmap
- Pnet
- endpoints

I am here to using endpoints parameters, because destipmap and pnet only take IP map one time by another. Like the command below:

sudo tcprewrite --infile=LLS_DDOS_1.0-dmz_new_c0lumns_rewrite.pcap --outfile=LLS_DDOS_1.0-dmz_new_c0lumns_rewrite2.pcap --dstipmap=172.16.113.204:172.16.1.12,172.16.115.20:172.16.2.12 –fixcsum

```
gns3@gns3vm:~/assign3$ sudo tcprewrite --infile=LLS_DDOS_1.0-dmz_new_c0lumns_rewrite.pc
ap --outfile=LLS_DDOS_1.0-dmz_new_c0lumns_rewrite2.pcap --dstipmap=172.16.113.204:172.1
6.1.12,172.16.115.20:172.16.2.12 --fixcsum
```

It tries to rewrite 172.16.113.204 to 172.16.1.12 and 172.16.115.20 to 172.16.2.12. However, our datasets have many different IP.

Considering about ***LLS_DDOS*** datasets contain randomized IP and mask, endpoints option is able to rewrite destination IP to a range of IP address. Our topology has the IP range: 192.16.1.11 – 192.16.1.12 and 192.16.2.11 – 192.168.2.12.

- Prepare the tcprewrite cache files:

Hoever endpoints needs cache files to parses the pcap file and keeps track each time a host either behaves like a client or like a server[6], so we need "**Tcpprep**" to generate cache files firstly:

Prepare "LLS_DDOS_1.0-dmz_new_c0lumns.cache" cache file:

sudo tcpprep --auto=bridge --pcap=LLS_DDOS_1.0-dmz_new_c0lumns.pcap --cachefile=LLS_DDOS_1.0-dmz_new_c0lumns.cache

Prepare "LLS_DDOS_1.0-inside_new_columns.cache" cache file:

sudo tcpprep --auto=bridge --pcap=LLS_DDOS_1.0-inside_new_columns.pcap --cachefile=LLS_DDOS_1.0-inside_new_columns.cache

The results show as below in screenshot:

```
gns3@gns3vm:~/assign3$ ls
172.16.112.194.telnet
LLS_DDOS_1.0-dmz.argus
LLS_DDOS_1.0-dmz.csv
LLS_DDOS_1.0-dmz_new_c0lumns.pcap
LLS_DDOS_1.0-dmz_new_c0lumns_rewrite2.argus
LLS_DDOS_1.0-dmz_new_c0lumns_rewrite2.pcap
LLS_DDOS_1.0-dmz_new_c0lumns_rewrite.argus
LLS_DDOS_1.0-dmz_new_c0lumns_rewrite.pcap
LLS_DDOS_1.0-dmz.pcap
LLS_DDOS_1.0-inside.dump
LLS_DDOS_1.0-inside_new_columns.pcap
LLS_DDOS_1.0-inside.pcap
gns3@gns3vm:~/assign3$ sudo tcpprep --auto=bridge --pcap=LLS_DDOS_1.0-dmz_new_c0lumns.pcap --cachefile=LLS_DDOS_1.0-dmz_new_c0lumns.cache
gns3@gns3vm:~/assign3$
gns3@gns3vm:~/assign3$ sudo tcpprep --auto=bridge --pcap=LLS_DDOS_1.0-inside_new_columns.pcap --cachefile=LLS_DDOS_1.0-inside_new_columns.cac
he
gns3@gns3vm:~/assign3$ ls
172.16.112.194.telnet           LLS_DDOS_1.0-dmz_new_c0lumns_rewrite2.argus  LLS_DDOS_1.0-inside.dump
LLS_DDOS_1.0-dmz.argus          LLS_DDOS_1.0-dmz_new_c0lumns_rewrite2.pcap   LLS_DDOS_1.0-inside_new_columns.cache
LLS_DDOS_1.0-dmz.csv            LLS_DDOS_1.0-dmz_new_c0lumns_rewrite.argus   LLS_DDOS_1.0-inside_new_columns.pcap
LLS_DDOS_1.0-dmz_new_c0lumns.cache  LLS_DDOS_1.0-dmz_new_c0lumns_rewrite.pcap    LLS_DDOS_1.0-inside.pcap
LLS_DDOS_1.0-dmz_new_c0lumns.pcap   LLS_DDOS_1.0-dmz.pcap
gns3@gns3vm:~/assign3$
```

By the way, in case our pcap file size is quite large, we might encounter "maximum size exceeding" issue, we need to run command "pcapfix" to adjust pcap file head [7].

- Rewrite the PCAP destination IP:

As mentioned before, I was utilizing different options to generate rewriting files. Finally, I found endpoints might be a good choice in our cases (Just imagine hacker is issuing attacks from local network host 192.168.111.1 to the ssh server 172.16.1.12):

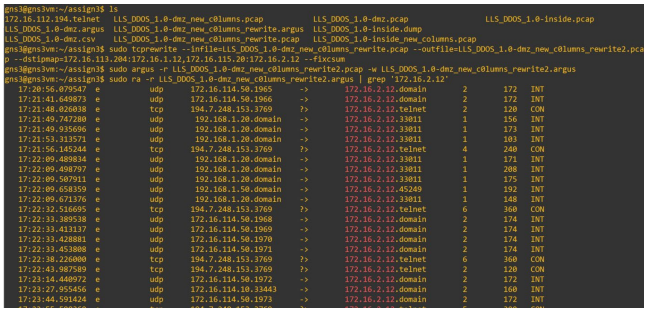
Rewrite "LLS_DDOS_1.0-dmz_new_c0lumns_rewrite.pcap":

sudo tcprewrite --infile=LLS_DDOS_1.0-dmz.pcap --outfile=LLS_DDOS_1.0-dmz_rewrite.pcap --endpoints=192.168.111.1:172.16.1.13 -b --cachefile=LLS_DDOS_1.0-dmz.cache

Rewrite "LLS_DDOS_1.0-inside_rewrite.pcap":

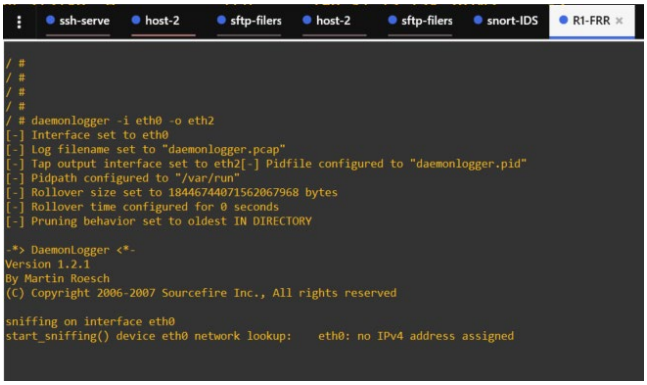
sudo tcprewrite --infile=LLS_DDOS_1.0-inside.pcap --outfile=LLS_DDOS_1.0-inside_rewrite.pcap --endpoints=192.168.111.1:172.16.1.13 -b --cachefile=LLS_DDOS_1.0-inside.cache

Test rewrite works:



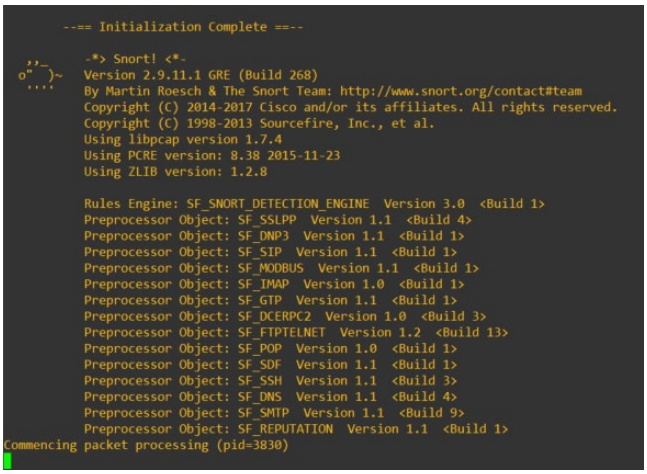
DaemonLogger starts up:

daemonlogger -i eth0 -o eth2

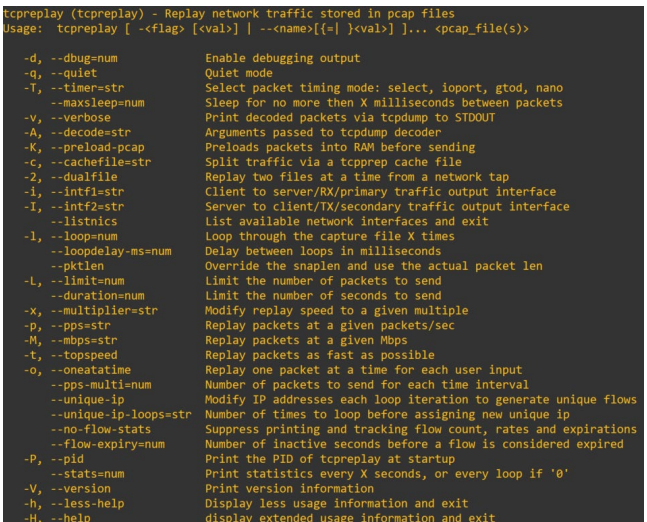


Snort IDS starts up:

snort -i eth0 -c /etc/snort/snort.conf -l /var/log/snort -A full



2.1.1 Replay the rewritten pcap



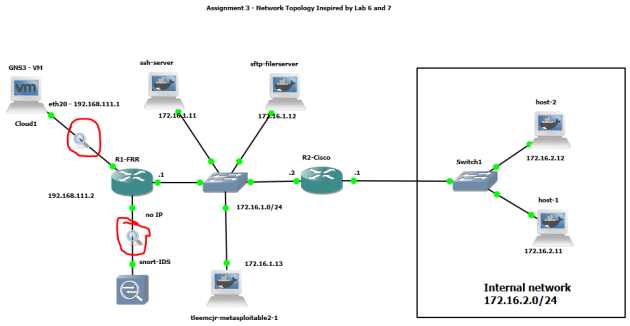
- Replay the new pcap files to vip2 virtual network interface of GNS3 VM.

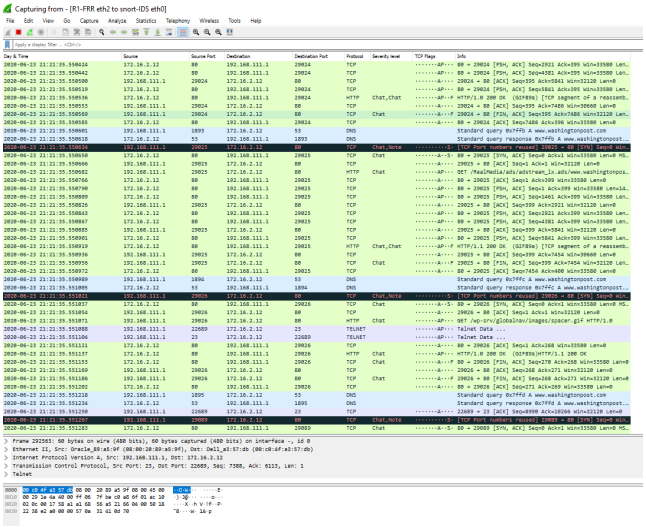
sudo tcpreplay -i vip2 -K -x 7.0 LLS_DDOS_1.0-inside_rewrite.pcap

- **-x play as 7 times speed (since -t will cause pkgs loss issue on my machine)**
- **-K preload pcap to speed up replay**

Above, as I investigated -t will omit many packages, because it transfers too fast. Then I switch to -x speedup options, it can not transfer all the packages. 7 times speedup is optimized number I can take, otherwise, it will take more than 4 hours to finish replaying by original speed.

First of all, let us activate the "capture" feature on GNS3 link between R1-FRR and Snort-IDS

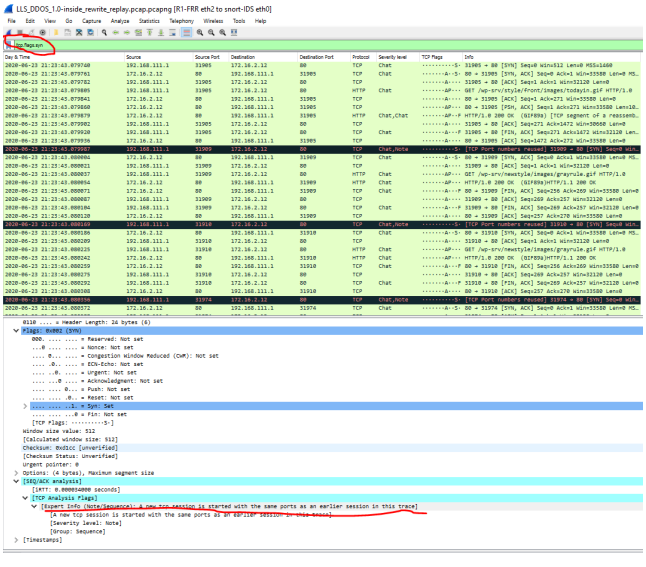




From Wireshark UI screenshot above, we can see the traffic is sniffed between [R1-FRR eth2 to snort-IDS eth0]. The datasets were already rewritten to adapt to our topology.

The output is saved by me as "LLS_DDOS_1.0-inside_rewrite_reply.pcap". The reviewer can find this file on my shared OneDrive [6].

Let us quickly examine the traffic displayed on WireShark UI. Because there are a lot of packets, We need to utilize the filter "tcp.flags.syn" to discover any potentials issues related to DDoS.



From the screenshot above, I am put "tcp.flags.syn" as filter string. We can see couple of red dark color highlighted packages seemed to have problem. For example, it noted as "A new tcp session is started with the same ports as an earlier session". However, it is hard to tell which one is actual attack from human's eyes. And it is tedious to check it one by one manly. Now we want to check Snort logs which automatically generated alerts and logs.

Run time for packet:

```
Run time for packet processing was 1741.573787 seconds
Snort processed 562229 packets.
Snort ran for 0 days 0 hours 29 minutes 1 seconds
   Pkts/min:        19387
   Pkts/sec:          322
```

Packet I/O Totals:

```
Packet I/O Totals:
   Received:       562229
   Analyzed:       562229 (100.000%)
    Dropped:            0 (  0.000%)
   Filtered:            0 (  0.000%)
Outstanding:            0 (  0.000%)
   Injected:            0
```

Breakdown by protocol

```
Breakdown by protocol (includes rebuilt packets):
        Eth:       563296 (100.000%)
       VLAN:            0 (  0.000%)
       IP4:        560245 ( 99.458%)
       Frag:            0 (  0.000%)
       ICMP:          219 (  0.039%)
        UDP:       124638 ( 22.127%)
        TCP:       435388 ( 77.293%)
       IP6:             1 (  0.000%)
   IP6 Ext:             1 (  0.000%)
  IP6 Opts:             0 (  0.000%)
      Frag6:            0 (  0.000%)
     ICMP6:             1 (  0.000%)
      UDP6:             0 (  0.000%)
      TCP6:             0 (  0.000%)
    Teredo:             0 (  0.000%)
   ICMP-IP:             0 (  0.000%)
   IP4/IP4:             0 (  0.000%)
  IP4/IP6:              0 (  0.000%)
  IP6/IP4:              0 (  0.000%)
  IP6/IP6:              0 (  0.000%)
       GRE:             0 (  0.000%)
   GRE Eth:             0 (  0.000%)
  GRE VLAN:             0 (  0.000%)
   GRE IP4:             0 (  0.000%)
   GRE IP6:             0 (  0.000%)
GRE IP6 Ext:            0 (  0.000%)
  GRE PPTP:             0 (  0.000%)
   GRE ARP:             0 (  0.000%)
   GRE IPX:             0 (  0.000%)
  GRE Loop:             0 (  0.000%)
      MPLS:             0 (  0.000%)
       ARP:          1934 (  0.343%)
       IPX:             0 (  0.000%)
  Eth Loop:             0 (  0.000%)
  Eth Disc:             0 (  0.000%)
  IP4 Disc:             0 (  0.000%)
  IP6 Disc:             0 (  0.000%)
  TCP Disc:             0 (  0.000%)
  UDP Disc:             0 (  0.000%)
 ICMP Disc:             0 (  0.000%)
All Discard:            0 (  0.000%)
      Other:         1116 (  0.198%)
Bad Chk Sum:         7323 (  1.300%)
   Bad TTL:             0 (  0.000%)
    S5 G 1:           335 (  0.059%)
    S5 G 2:           732 (  0.130%)
      Total:       563296
```

Alerts:

```
Action Stats:
     Alerts:          394 (  0.070%)
     Logged:          394 (  0.070%)
     Passed:            0 (  0.000%)
Limits:
      Match:            0
      Queue:            0
        Log:            0
      Event:            0
      Alert:           18
Verdicts:
      Allow:       555074 ( 98.727%)
      Block:            0 (  0.000%)
    Replace:            0 (  0.000%)
  Whitelist:         7155 (  1.273%)
  Blacklist:            0 (  0.000%)
     Ignore:            0 (  0.000%)
      Retry:            0 (  0.000%)
```

Stream statistics:

```
Stream statistics:
            Total sessions: 21725
              TCP sessions: 8785
              UDP sessions: 12940
             ICMP sessions: 0
               IP sessions: 0
                TCP Prunes: 132
                UDP Prunes: 0
               ICMP Prunes: 0
                 IP Prunes: 0
TCP StreamTrackers Created: 8825
TCP StreamTrackers Deleted: 8825
               TCP Timeouts: 41
               TCP Overlaps: 0
        TCP Segments Queued: 123380
      TCP Segments Released: 123380
        TCP Rebuilt Packets: 21789
          TCP Segments Used: 65813
               TCP Discards: 5
                   TCP Gaps: 6745
      UDP Sessions Created: 13287
      UDP Sessions Deleted: 13287
               UDP Timeouts: 347
               UDP Discards: 0
```

HTTP Inspect – encodings:

```
HTTP Inspect - encodings (Note: stream-reassembled packets included):
    POST methods:                       0
    GET methods:                        7316
    HTTP Request Headers extracted:     7319
    HTTP Request Cookies extracted:     0
    Post parameters extracted:          0
    HTTP response Headers extracted:    7303
    HTTP Response Cookies extracted:    592
    Unicode:                            0
    Double unicode:                     0
    Non-ASCII representable:            3
    Directory traversals:               0
    Extra slashes ("//"):               21
    Self-referencing paths ("./"):      0
    HTTP Response Gzip packets extracted: 0
    Gzip Compressed Data Processed:     n/a
    Gzip Decompressed Data Processed:   n/a
    Http/2 Rebuilt Packets:             0
    Total packets processed:            54577
```

SMTP Preprocessor Statistics

```
SMTP Preprocessor Statistics
  Total sessions                                : 254
  Max concurrent sessions                       : 184
  Base64 attachments decoded                    : 0
  Total Base64 decoded bytes                    : 0
  Quoted-Printable attachments decoded          : 0
  Total Quoted decoded bytes                    : 0
  UU attachments decoded                        : 0
  Total UU decoded bytes                        : 0
  Non-Encoded MIME attachments extracted        : 0
  Total Non-Encoded MIME bytes extracted        : 0
  SMTP Sessions fastpathed due to memcap exceeded: 3262
```

Therefore, from the screenshots of statistics of different perspectives, we found it reports 338 alerts, which occupied 0.088 % of total packages. Let us take a look at the alert log and find any interesting things. I saved this file as "LLS_DDOS_1.0-inside-before.alert" which can be viewed in shared OneDrive Folder [6].

By checking Priority:1 in alert, we found all the attacks belong to "Attempted Administrator Privilege Gain":

```
root@snort-IDS:/var/log/snort# cat alert | grep "Priority: 1" | sort | uniq -c
     17 [Classification: Attempted Administrator Privilege Gain] [Priority: 1]
```

- **17 [Classification: Attempted Administrator Privilege Gain] [Priority: 1]**

By checking "priority:2", we found several attack types, such as:

- **16 [Classification: Access to a Potentially Vulnerable Web Application] [Priority: 2]**
- **27 [Classification: Attempted Information Leak] [Priority: 2]**
- **89 [Classification: Decode of an RPC Query] [Priority: 2]**
- **19 [Classification: Misc Attack] [Priority: 2]**
- **[Classification: Potentially Bad Traffic] [Priority: 2]**

```
root@snort-IDS:/var/log/snort# cat alert | grep "Priority: 2" | sort | uniq -c
     16 [Classification: Access to a Potentially Vulnerable Web Application] [Priority: 2]
     27 [Classification: Attempted Information Leak] [Priority: 2]
     89 [Classification: Decode of an RPC Query] [Priority: 2]
     19 [Classification: Misc Attack] [Priority: 2]
      5 [Classification: Potentially Bad Traffic] [Priority: 2]
```

priority:3 and below are not suspicious activities.

```
root@snort-IDS:/var/log/snort# cat alert | grep "Priority: 3" | sort | uniq -c
    204 [Classification: Misc activity] [Priority: 3]
     17 [Classification: Not Suspicious Traffic] [Priority: 3]
```

- **204 [Classification: Misc activity] [Priority: 3]**
- **17 [Classification: Not Suspicious Traffic] [Priority: 3]**

Next, LLS_DDOS datasets contains another PCAP file: LLS_DDOS_1.0-dmz.pcap. We would better check it as well and make the comparison.

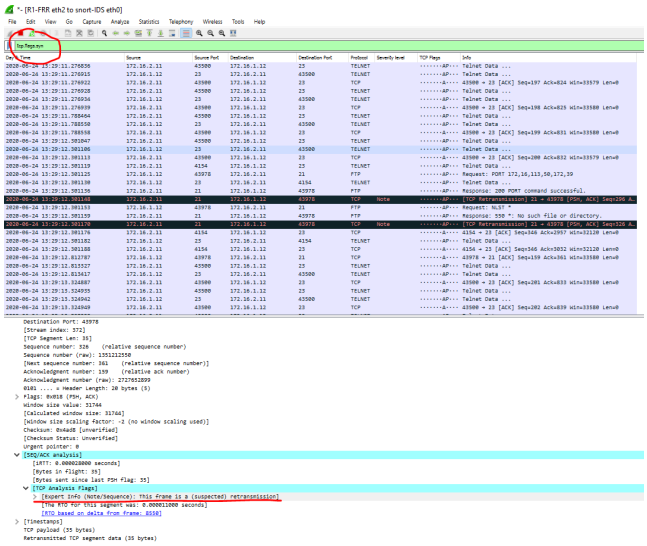
Same as "LLS_DDOS_1.0-inside.dump" file, we do the same preparation for "LLS_DDOS_1.0-dmz_new_c0lumns_rewrite.pcap". So, I setup daemonlogger on R1-RFF and started Snort command on the snort-IDS server. One more thing is I need to clear up logs under /var/log/snort/ to make sure new traffic obtains new logs. The I issued command on "GNS-VM" to replay the traffic:

sudo tcpreplay -i -x 7.0 vip2 LLS_DDOS_1.0-dmz_rewrite.pcap

After executing the following command:

```
gns3@gns3vm:~/assign3$ sudo tcpreplay -i vip2 LLS_DDOS_1.0-dmz_rewrite.pcap -t
```

I was continuously monitor sniffed traffic via WireShark:



From the screenshot above, I did identify some issues related to SYN communication, like retransmission of suspected frame. I will show the Snort statistics results and related alerts logs.

Run time and average rates:

```
===============================================================================
Run time for packet processing was 1779.467369 seconds
Snort processed 362321 packets.
Snort ran for 0 days 0 hours 29 minutes 39 seconds
   Pkts/min:        12493
   Pkts/sec:          203
===============================================================================
```

Packet I/O totals:

```
===============================================================================
Packet I/O Totals:
   Received:       362321
   Analyzed:       362321 (100.000%)
    Dropped:            0 (  0.000%)
   Filtered:            0 (  0.000%)
Outstanding:            0 (  0.000%)
   Injected:            0
===============================================================================
```

Protocol breakdown:

```
===============================================================================
Breakdown by protocol (includes rebuilt packets):
        Eth:       363380 (100.000%)
       VLAN:            0 (  0.000%)
        IP4:       359144 ( 98.834%)
       Frag:            0 (  0.000%)
       ICMP:         2970 (  0.817%)
        UDP:         3342 (  0.920%)
        TCP:       352832 ( 97.097%)
        IP6:            1 (  0.000%)
    IP6 Ext:            1 (  0.000%)
   IP6 Opts:            0 (  0.000%)
      Frag6:            0 (  0.000%)
     ICMP6:             1 (  0.000%)
      UDP6:            0 (  0.000%)
      TCP6:            0 (  0.000%)
    Teredo:            0 (  0.000%)
   ICMP-IP:            0 (  0.000%)
   IP4/IP4:            0 (  0.000%)
   IP4/IP6:            0 (  0.000%)
   IP6/IP4:            0 (  0.000%)
   IP6/IP6:            0 (  0.000%)
       GRE:            0 (  0.000%)
   GRE Eth:            0 (  0.000%)
  GRE VLAN:            0 (  0.000%)
   GRE IP4:            0 (  0.000%)
   GRE IP6:            0 (  0.000%)
GRE IP6 Ext:           0 (  0.000%)
  GRE PPTP:            0 (  0.000%)
   GRE ARP:            0 (  0.000%)
   GRE IPX:            0 (  0.000%)
  GRE Loop:            0 (  0.000%)
      MPLS:            0 (  0.000%)
       ARP:          468 (  0.129%)
       IPX:            0 (  0.000%)
  Eth Loop:         1163 (  0.320%)
  Eth Disc:            0 (  0.000%)
  IP4 Disc:            0 (  0.000%)
  IP6 Disc:            0 (  0.000%)
  TCP Disc:            0 (  0.000%)
  UDP Disc:            0 (  0.000%)
 ICMP Disc:            0 (  0.000%)
All Discard:           0 (  0.000%)
      Other:        2604 (  0.717%)
Bad Chk Sum:           0 (  0.000%)
    Bad TTL:           0 (  0.000%)
     S5 G 1:         381 (  0.105%)
     S5 G 2:         678 (  0.187%)
      Total:       363380
===============================================================================
```

Alerts Stats:

```
===============================================================================
Action Stats:
     Alerts:         1223 (  0.337%)
     Logged:         1223 (  0.337%)
     Passed:            0 (  0.000%)
Limits:
      Match:            0
      Queue:            0
        Log:            0
      Event:            0
      Alert:          109
Verdicts:
      Allow:       346937 ( 95.754%)
      Block:            0 (  0.000%)
    Replace:            0 (  0.000%)
  Whitelist:        15384 (  4.246%)
  Blacklist:            0 (  0.000%)
     Ignore:            0 (  0.000%)
      Retry:            0 (  0.000%)
===============================================================================
```

Stream Stats:

```
Stream statistics:
            Total sessions: 10299
              TCP sessions: 8926
              UDP sessions: 1373
             ICMP sessions: 0
               IP sessions: 0
                TCP Prunes: 0
                UDP Prunes: 0
               ICMP Prunes: 0
                 IP Prunes: 0
TCP StreamTrackers Created: 8927
TCP StreamTrackers Deleted: 8927
              TCP Timeouts: 3
              TCP Overlaps: 6
       TCP Segments Queued: 104853
     TCP Segments Released: 104853
       TCP Rebuilt Packets: 22999
         TCP Segments Used: 69891
              TCP Discards: 13
                 TCP Gaps: 6765
      UDP Sessions Created: 1373
      UDP Sessions Deleted: 1373
              UDP Timeouts: 0
              UDP Discards: 0
                    Events: 88127
          Internal Events: 0
          TCP Port Filter
                 Filtered: 0
                Inspected: 0
                  Tracked: 351773
          UDP Port Filter
                 Filtered: 0
                Inspected: 0
                  Tracked: 1373
```

HTTP Inspect:

```
HTTP Inspect - encodings (Note: stream-reassembled packets included):
    POST methods:                     0
    GET methods:                      7711
    HTTP Request Headers extracted:   7725
    HTTP Request Cookies extracted:   0
    Post parameters extracted:        1
    HTTP response Headers extracted:  7703
    HTTP Response Cookies extracted:  601
    Unicode:                          0
    Double unicode:                   0
    Non-ASCII representable:          0
    Directory traversals:             0
    Extra slashes ("//"):             24
    Self-referencing paths ("./"):    0
    HTTP Response Gzip packets extracted: 0
    Gzip Compressed Data Processed:   n/a
    Gzip Decompressed Data Processed: n/a
    Http/2 Rebuilt Packets:           0
    Total packets processed:          57178
```

SMTP stats:

```
SMTP Preprocessor Statistics
  Total sessions                              : 247
  Max concurrent sessions                     : 184
  Base64 attachments decoded                  : 0
  Total Base64 decoded bytes                  : 0
  Quoted-Printable attachments decoded        : 0
  Total Quoted decoded bytes                  : 0
  UU attachments decoded                      : 0
  Total UU decoded bytes                      : 0
  Non-Encoded MIME attachments extracted      : 0
  Total Non-Encoded MIME bytes extracted      : 0
  SMTP Sessions fastpathed due to memcap exceeded: 3242
```

POP Stats:

```
POP Preprocessor Statistics
  Total sessions                              : 16
  Max concurrent sessions                     : 12
  Base64 attachments decoded                  : 0
  Total Base64 decoded bytes                  : 0
  Quoted-Printable attachments decoded        : 0
  Total Quoted decoded bytes                  : 0
  UU attachments decoded                      : 0
  Total UU decoded bytes                      : 0
  Non-Encoded MIME attachments extracted      : 0
  Total Non-Encoded MIME bytes extracted      : 0
```

Based on alerts logs, I found priority 1 and 2 information:

By checking Priority:1 in alert, we found all the attacks belong to "Attempted Administrator Privilege Gain":

```
root@snort-IDS:/var/log/snort# cat alert | grep "Priority: 1" | sort | uniq -c
     38 [Classification: Attempted Administrator Privilege Gain] [Priority: 1]
```

- **38 [Classification: Attempted Administrator Privilege Gain] [Priority: 1]**

By examining the priority 2, we found the following unique alert types:

- **18 [Classification: Access to a Potentially Vulnerable Web Application] [Priority: 2]**
- **29 [Classification: Attempted Information Leak] [Priority: 2]**
- **160 [Classification: Decode of an RPC Query] [Priority: 2]**
- **123 [Classification: Misc Attack] [Priority: 2]**
- **20 [Classification: Potentially Bad Traffic] [Priority: 2]**

```
root@snort-IDS:/var/log/snort# cat alert | grep "Priority: 2" | sort | uniq -c
     18 [Classification: Access to a Potentially Vulnerable Web Application] [Priority: 2]
     29 [Classification: Attempted Information Leak] [Priority: 2]
    160 [Classification: Decode of an RPC Query] [Priority: 2]
    123 [Classification: Misc Attack] [Priority: 2]
     20 [Classification: Potentially Bad Traffic] [Priority: 2]
```

However, those alerts information are quite general, and I did not find DoS alert message, or they were overridden by other alerts. So, I am going to improve the configuration to discover those hidden issues.

Next section, I was going to seek efficient way to improve the Snort rules to detect potentials issues from previous samples.

4 Question 2: Improvement of detection rules

Here, in this section, I want to go further to discuss how to modify and refine the snort configuration to make it work just as what we want.

4.1 Locate Snort configuration file

Firstly, let us recall the Snort concepts: Snort is an open source network intrusion prevention system, capable of performing real-time traffic analysis and packet logging on IP networks. It can perform protocol analysis, content searching/matching, and can be used to detect a variety of attacks and probes, such as buffer overflows, stealth port scans, CGI attacks, SMB probes, OS fingerprinting attempts, and much more [8].

Usually, it has a major configuration file specify global work environment. More important, it can include external configuration for each specific attack category. For example, ICMP.rule under /etc/snort/snort.config.

At this point, I want to create a brand-new rule file, especially for the DoS attack which the original datasets targets on. Then include this rule file into snort.config global configures.

4.2 Make the rule for DoS

A denial-of-service attack (DoS attack) is a cyber-attack in which the perpetrator seeks to make a machine or network resource unavailable to its intended users by temporarily or indefinitely disrupting services of a host connected to the internet. Denial of service is typically accomplished by flooding the targeted machine or resource with superfluous requests in an attempt to overload systems and prevent some or all legitimate requests from being fulfilled [7].

Denial-of-service attacks are characterized by an explicit attempt by attackers to prevent legitimate use of a service. There are two general forms of DoS attacks: those that crash services and those that flood services. The most serious attacks are distributed. It is called DDoS. We can simulate it by issuing Hpring3 command with randomized source IP [7].

To explain my motivation and method of the rules I made for DoS, I want to show a real example:

```
root@tleemcjr-metasploitable2-1:/# netstat
Active Internet connections (w/o servers)
Proto Recv-Q Send-Q Local Address           Foreign Address         State
tcp        0      0 172.16.1.13:www         75.74.110.159:1749      SYN_RECV
tcp        0      0 172.16.1.13:www         209.194.36.253:1738     SYN_RECV
tcp        0      0 172.16.1.13:www         103.124.250.210:1730    SYN_RECV
tcp        0      0 172.16.1.13:www         136.207.56.221:1634     SYN_RECV
tcp        0      0 172.16.1.13:www         119.32.1.5:1725         SYN_RECV
tcp        0      0 172.16.1.13:www         1.90.79.26:sa-msg-por   SYN_RECV
tcp        0      0 172.16.1.13:www         117.30.34.5:1663        SYN_RECV
tcp        0      0 172.16.1.13:www         77.237.45.195:1756      SYN_RECV
tcp        0      0 172.16.1.13:www         144.68.2.132:1734       SYN_RECV
tcp        0      0 172.16.1.13:www         118.31.36.49:1647       SYN_RECV
tcp        0      0 172.16.1.13:www         203.0.108.247:1683      SYN_RECV
tcp        0      0 172.16.1.13:www         184.68.220.28:1741      SYN_RECV
tcp        0      0 172.16.1.13:www         13.58.109.254:1681      SYN_RECV
tcp        0      0 172.16.1.13:www         163.162.244.113:1750    SYN_RECV
tcp        0      0 172.16.1.13:www         188.207.5.184:1768      SYN_RECV
tcp        0      0 172.16.1.13:www         76.216.156.216:1650     SYN_RECV
tcp        0      0 172.16.1.13:www         209.215.115.93:1711     SYN_RECV
tcp        0      0 172.16.1.13:www         243.89.145.50:1706      SYN_RECV
tcp        0      0 172.16.1.13:www         195.128.72.74:1760      SYN_RECV
tcp        0      0 172.16.1.13:www         203.246.216.145:1762    SYN_RECV
tcp        0      0 172.16.1.13:www         58.71.196.189:1753      SYN_RECV
tcp        0      0 172.16.1.13:www         44.65.166.92:1669       SYN_RECV
tcp        0      0 172.16.1.13:www         221.227.65.253:1666     SYN_RECV
```

From the screenshot of "netstats" results on victim server. I can see that many connections stay in SYN_RECV state. And source IP are randomized to be simulated from different locations. The waiting time is the time between receiving the first and third packet in the 3-way handshake. It has been proved 5-6 packets stay in the same state SYN_RECV is overly aggressive case. The health situation is that only maximum 1/5 of the sockets should be in in SYN_RECV and the majority should be in ESTABLISHED or TIME_WAIT state.

Therefor, I can tell something is abnormal when I observe many SYN_RECV packets constantly. In conclusion, having many SYN RECV packets means a SYN Flood.

At this point, Snort is tested a lot from last section. Now, we need to write our own rules that will enable snort to detect more potential attack, such as DDoS etc.
Inspired by the instruction [10], I created own rules file

- **Step 1: define our own classification in classification.config under /etc/snort/:**

our own classifications
config classification: icmp-flood,Potential DoS attack,1
config classification: syn-flood,Potential DoS attack,1

```
# our own classifications
config classification: icmp-flood,Potential DoS attack,1
config classification: syn-flood,Potential DoS attack,1
```

I shall give a brief explanation about above coding:

Classification definition is csv comma separated file, we need to follow the fields to define each attribute names, like:

#
config classification:shortname,short description,priority
#

- **Step 2: define HOME_NET value in snort.conf under /etc/snort**

Setup the network addresses you are protecting
ipvar HOME_NET 172.16.1.0/24

```
# Setup the network addresses you are protecting
ipvar HOME_NET 172.16.1.0/24
```

Because our vulnerable target IP is 172.16.1.13, the line above will protect all subnets of 172.16.1.0/24.

- **Step 3: create a customized rule file under "/etc/snort/rules/", let us say "cs6706.rules".**

```
root@snort-IDS:/etc/snort/rules# ls
VRT-License.txt          community-web-misc.rules    mysql.rules          pua-toolbars.rules
app-detect.rules         community-web-php.rules     netbios.rules        rpc.rules
attack-responses.rules   content-replace.rules       nntp.rules           rservices.rules
backdoor.rules           cs6706.rules                oracle.rules         scada.rules
bad-traffic.rules        ddos.rules                  os-linux.rules       scan.rules
black_list.rules         deleted.rules               os-mobile.rules      server-apache.rules
blacklist.rules          dns.rules                   os-other.rules       server-iis.rules
```

- **Step 4: include this file in the Snore global configuration file: "/etc/snort/snort.conf"**

Sudo vim snort.conf:

```
root@snort-IDS:/etc/snort# ls
attribute_table.dtd   file_magic.conf  preproc_rules     rules          snort.conf  threshold.conf
classification.config  gen-msg.map     reference.config  sid-msg.map  so_rules    unicode.map
root@snort-IDS:/etc/snort# sudo vim snort.conf
```

########this is for customized rule files######
include $RULE_PATH/cs6706.rules

```
#include $RULE_PATH/community-web-misc.rules
#include $RULE_PATH/community-web-php.rules
########this is for customized rule files######
include $RULE_PATH/cs6706.rules
```

The content I will add in shows below:

alert icmp any any -> $HOME_NET any (msg:"Potential ICMP flood attack detected"; sid:6706001; classtype:icmp-flood; detection_filter:track by_dst, count 550, seconds 4;)

It is worth to explain the line above here:

Rule action includes several conditions which trigger the alerts, it includes Source IP, Source port

- Icmp – protocol we are looking at
- any – we will look at all sources.
- any – we will look at all ports.
- $HOME_NET –We are using the HOME_NET value from the snort.conf file.
- any – we will look at all ports on the protected network

there are also options for rules definition for customization:

- msg– we can customize the alert message at this point.
- sid:6706001 – snort rule ID. We should make it unique.
- classtype:I am using my own customized class "icmp-flood".
- detection_filter:track by_dst – we are tracking the destination IP address for detection.
- seconds 4 – detection interval period 4 seconds

- count 550 – during the detciion interval period, we are sampling 550 parckets.

alert tcp any any -> $HOME_NET 80 (flags: S; msg:"Potential SYN flood Attack detected"; flow:stateless; sid:6706002; classtype:syn-flood;detection_filter:track by_dst, count 15, seconds 5;)

In this rule we have changed the protocol to TCP and set the destination port number to 80. The keyword flag checks if specific TCP flag bits(in this case SYN flag) are present. The sampling period is set to 10 seconds. If during this time period more than 20 requests are detected, then we will receive the alert.

Finally, we have our own defined rules in cs6706.rules:

 # Copyright 2001-2005 Sourcefire, Inc. All Rights Reserved
#
 # This file may contain proprietary rules that were created, tested and
 # certified by Sourcefire, Inc. (the "VRT Certified Rules") as well as
 # rules that were created by Sourcefire and other third parties and
 # distributed under the GNU General Public License (the "GPL Rules"). The
 # VRT Certified Rules contained in this file are the property of
 # Sourcefire, Inc. Copyright 2005 Sourcefire, Inc. All Rights Reserved.
 # The GPL Rules created by Sourcefire, Inc. are the property of
 # Sourcefire, Inc. Copyright 2002-2005 Sourcefire, Inc. All Rights
 # Reserved. All other GPL Rules are owned and copyrighted by their
 # respective owners (please see www.snort.org/contributors for a list of
 # owners and their respective copyrights). In order to determine what
 # rules are VRT Certified Rules or GPL Rules, please refer to the VRT
 # Certified Rules License Agreement.
#
#
 # $Id: cs6706.rules,v 0.9 2020/06/26 10:17:51 Christian Gang Liu Exp $
#----------
 # Our Own DDOS RULES
#----------

alert icmp any any -> $HOME_NET any (msg:"Potential ICMP flood attack detected"; sid:6706001; classtype:icmp-flood; detection_filter:track by_dst, count 550, seconds 4;)

alert tcp any any -> $HOME_NET 80 (flags: S; msg:"Potential SYN flood Attack detected"; flow:stateless; sid:6706002; classtype:syn-flood; detection_filter:track by_dst, count 15, seconds 5;)

Next, I want to retest it with ICMP and SYN DOS attack by Hping3 command.

4.3 Hpring3 Simulation

Here, I was reusing the PCAP output of last time running Hpring3 command: **hping3-initial-test.pca** before we made rules change:

sudo tcpreplay -i vip2 -K -x 7.0 hping3-initial-test.pcap

```
gns3@gns3vm:~/assign3$ sudo tcpreplay -i vip2  -K -x 7.0 hping3-initial-test.pca
p
File Cache is enabled
Actual: 251649 packets (35187106 bytes) sent in 13.02 seconds
Rated: 2701910.8 Bps, 21.61 Mbps, 19323.36 pps
Flows: 250464 flows, 19232.36 fps, 251635 flow packets, 14 non-flow
Statistics for network device: vip2
        Successful packets:        251649
        Failed packets:            0
        Truncated packets:         0
        Retried packets (ENOBUFS): 0
        Retried packets (EAGAIN):  0
```

Then we should take a look at Snort stats and alerts:

Packet I/O Totals:

```
Packet I/O Totals:
   Received:       104330
   Analyzed:        90436 ( 86.683%)
    Dropped:        13894 ( 11.752%)
   Filtered:            0 (  0.000%)
Outstanding:        13894 ( 13.317%)
   Injected:            0
```

Breakdown by protocol

```
Breakdown by protocol (includes rebuilt packets):
        Eth:        90436 (100.000%)
       VLAN:            0 (  0.000%)
        IP4:        90418 ( 99.980%)
       Frag:            0 (  0.000%)
       ICMP:            0 (  0.000%)
        UDP:            0 (  0.000%)
        TCP:        90418 ( 99.980%)
        IP6:            0 (  0.000%)
    IP6 Ext:            0 (  0.000%)
   IP6 Opts:            0 (  0.000%)
      Frag6:            0 (  0.000%)
      ICMP6:            0 (  0.000%)
       UDP6:            0 (  0.000%)
       TCP6:            0 (  0.000%)
     Teredo:            0 (  0.000%)
    ICMP-IP:            0 (  0.000%)
    IP4/IP4:            0 (  0.000%)
    IP4/IP6:            0 (  0.000%)
    IP6/IP4:            0 (  0.000%)
    IP6/IP6:            0 (  0.000%)
        GRE:            0 (  0.000%)
    GRE Eth:            0 (  0.000%)
   GRE VLAN:            0 (  0.000%)
    GRE IP4:            0 (  0.000%)
    GRE IP6:            0 (  0.000%)
GRE IP6 Ext:            0 (  0.000%)
   GRE PPTP:            0 (  0.000%)
    GRE ARP:            0 (  0.000%)
    GRE IPX:            0 (  0.000%)
   GRE Loop:            0 (  0.000%)
       MPLS:            0 (  0.000%)
        ARP:           18 (  0.020%)
        IPX:            0 (  0.000%)
   Eth Loop:            0 (  0.000%)
   Eth Disc:            0 (  0.000%)
   IP4 Disc:            0 (  0.000%)
   IP6 Disc:            0 (  0.000%)
   TCP Disc:            0 (  0.000%)
   UDP Disc:            0 (  0.000%)
  ICMP Disc:            0 (  0.000%)
All Discard:            0 (  0.000%)
      Other:            0 (  0.000%)
Bad Chk Sum:            0 (  0.000%)
    Bad TTL:            0 (  0.000%)
     S5 G 1:            0 (  0.000%)
     S5 G 2:            0 (  0.000%)
      Total:        90436
```

Alerts stats:

```
Action Stats:
     Alerts:        37084 ( 41.006%)
     Logged:        37084 ( 41.006%)
     Passed:            0 (  0.000%)
Limits:
      Match:            0
      Queue:            0
        Log:            0
      Event:            0
      Alert:            0
Verdicts:
      Allow:        90436 ( 86.683%)
      Block:            0 (  0.000%)
     Replace:           0 (  0.000%)
  Whitelist:            0 (  0.000%)
  Blacklist:            0 (  0.000%)
     Ignore:            0 (  0.000%)
      Retry:            0 (  0.000%)
```

From above, we noticed alerts rates increased dramatically because of new rules setting. Let us look inside the logs:

cat alert | grep "Priority: 1" | sort | uniq -c

```
root@snort-IDS:/var/log/snort# cat alert | grep "Priority: 1" | sort | uniq -c
  36939 [Classification: Potential DoS attack] [Priority: 1]
root@snort-IDS:/var/log/snort#
```

- **36939 [Classification: Potential DoS attack] [Priority: 1]**

We identified Priority 1 alerts which classified as “Potential DoS attack”. That means new rules worked. Same things we can do with priority 2 and 3

```
root@snort-IDS:/var/log/snort# cat alert | grep "Priority: 2" | sort | uniq -c
    145 [Classification: Potentially Bad Traffic] [Priority: 2]
root@snort-IDS:/var/log/snort# cat alert | grep "Priority: 3" | sort | uniq -c
root@snort-IDS:/var/log/snort#
```

- **145 [Classification: Potentially Bad Traffic] [Priority: 2]**

We find 145 bad traffic alerts, which come from the original alert rules.

4.4 Port scan detection

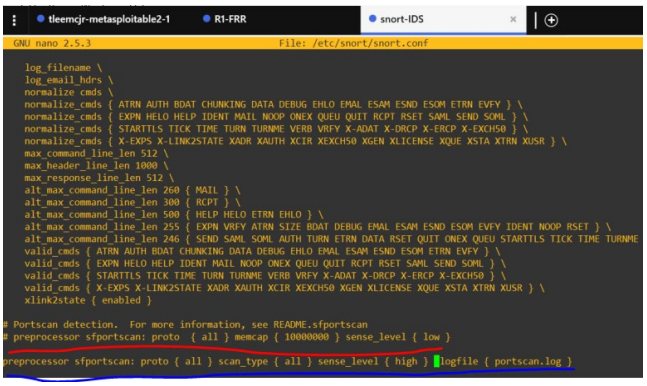
In addition, we might want to enable Port scan detection as well. It is easy to achieve [9].

Navigate to /etc/snort directory, and open /etc/snort/snort.conf. Then find the following line:

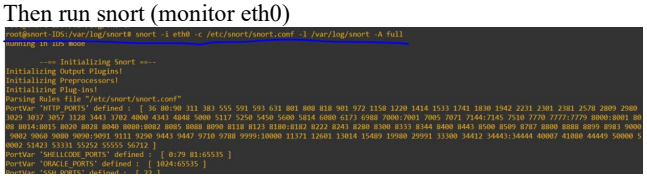
 # preprocessor sfportscan: proto { all } memcap { 10000000 } sense_level { low }

add the following below the line:

preprocessor sfportscan: proto { all } scan_type { all } sense_level { high } \ logfile { portscan.log }



Then run snort (monitor eth0)



perform port scan from my VM

$sudo nmap -sT -p1000-2000 172.16.1.13

```
gns3@gns3vm:~$ sudo nmap -sT -p 1000-2000 172.16.1.13

Starting Nmap 7.60 ( https://nmap.org ) at 2020-06-27 15:53 +03
Nmap scan report for 172.16.1.13
Host is up (0.0050s latency).
Not shown: 999 closed ports
PORT     STATE SERVICE
1099/tcp open  rmiregistry
1524/tcp open  ingreslock

Nmap done: 1 IP address (1 host up) scanned in 15.59 seconds
gns3@gns3vm:~$
```

Next, we want to check snort stats:

```
===============================================================================
Run time for packet processing was 29.20987 seconds
Snort processed 1576 packets.
Snort ran for 0 days 0 hours 0 minutes 29 seconds
   Pkts/sec:           54
===============================================================================
Memory usage summary:
  Total non-mmapped bytes (arena):       30081024
  Bytes in mapped regions (hblkhd):      29995008
  Total allocated space (uordblks):      23273216
  Total free space (fordblks):           6807808
  Topmost releasable block (keepcost):   461872
===============================================================================
Packet I/O Totals:
   Received:         1576
   Analyzed:         1576 (100.000%)
    Dropped:            0 (  0.000%)
   Filtered:            0 (  0.000%)
Outstanding:            0 (  0.000%)
   Injected:            0
===============================================================================
Breakdown by protocol (includes rebuilt packets):
        Eth:         1576 (100.000%)
       VLAN:            0 (  0.000%)
        IP4:         1572 ( 99.746%)
       Frag:            0 (  0.000%)
       ICMP:            4 (  0.254%)
        UDP:            0 (  0.000%)
        TCP:         1568 ( 99.492%)
        IP6:            0 (  0.000%)
    IP6 Ext:            0 (  0.000%)
   IP6 Opts:            0 (  0.000%)
```

Alert stats:

```
===============================================================================
Action Stats:
     Alerts:            3 (  0.190%)
     Logged:            3 (  0.190%)
     Passed:            0 (  0.000%)
Limits:
      Match:            0
      Queue:            0
        Log:            0
      Event:            0
      Alert:            0
Verdicts:
      Allow:         1576 (100.000%)
      Block:            0 (  0.000%)
    Replace:            0 (  0.000%)
  Whitelist:            0 (  0.000%)
  Blacklist:            0 (  0.000%)
     Ignore:            0 (  0.000%)
      Retry:            0 (  0.000%)
===============================================================================
```

From above, we did find alert. We want to see the details:

```
root@snort-IDS:/var/log/snort# cat alert
[**] [1:469:3] ICMP PING NMAP [**]
[Classification: Attempted Information Leak] [Priority: 2]
06/27-13:01:07.333771 192.168.111.1 -> 172.16.1.13
ICMP TTL:39 TOS:0x0 ID:13233 IpLen:20 DgmLen:28
Type:8  Code:0  ID:59751   Seq:0  ECHO
[Xref => http://www.whitehats.com/info/IDS162]

[**] [1:384:5] ICMP PING [**]
[Classification: Misc activity] [Priority: 3]
06/27-13:01:07.333771 192.168.111.1 -> 172.16.1.13
ICMP TTL:39 TOS:0x0 ID:13233 IpLen:20 DgmLen:28
Type:8  Code:0  ID:59751   Seq:0  ECHO

[**] [1:453:5] ICMP Timestamp Request [**]
[Classification: Misc activity] [Priority: 3]
06/27-13:01:07.333782 192.168.111.1 -> 172.16.1.13
ICMP TTL:51 TOS:0x0 ID:25733 IpLen:20 DgmLen:40
Type:13  Code:0  ID: 38730  Seq: 0  TIMESTAMP REQUEST

root@snort-IDS:/var/log/snort#
root@snort-IDS:/var/log/snort# cat alert | grep "Priority: 2" | sort | uniq -c
      1 [Classification: Attempted Information Leak] [Priority: 2]
root@snort-IDS:/var/log/snort#
root@snort-IDS:/var/log/snort# cat alert | grep "Priority: 3" | sort | uniq -c
      2 [Classification: Misc activity] [Priority: 3]
root@snort-IDS:/var/log/snort#
```

We found priority 2 alert: Attempted Information Leak.

and check the logs at /var/log/snort.

```
root@snort-IDS:/var/log/snort# ls
alert  portscan.log  snort.log.1593262860
root@snort-IDS:/var/log/snort# cat portscan.log
Time: 06/27-13:01:20.465030
event_ref: 0
192.168.111.1 -> 172.16.1.13 (portscan) TCP Portscan
Priority Count: 5
Connection Count: 11
IP Count: 1
Scanner IP Range: 192.168.111.1:192.168.111.1
Port/Proto Count: 11
Port/Proto Range: 443:1965

root@snort-IDS:/var/log/snort#
```

Till now, we already setup snort configuration for tuning up.
Next, we want to test our real datasets downloaded from DARPA.

4.5 DARPA Intrusion Detection Datasets

However,

1) **Replay LLS_DDOS_1.0-inside_rewrite.pcap with new Snort rules:**

sudo tcpreplay -i vip2 -K -x 7.0 LLS_DDOS_1.0-inside_rewrite.pcap

```
gns3@gns3vm:~/assign3$ sudo tcpreplay -i vip2  -K -x 7.0 LLS_DDOS_1.0-inside_rewrite.pcap
File Cache is enabled
Actual: 649787 packets (111685562 bytes) sent in 1698.07 seconds
Rated: 65771.8 Bps, 0.526 Mbps, 382.66 pps
Flows: 112405 flows, 66.19 fps, 646561 flow packets, 3084 non-flow
Statistics for network device: vip2
        Successful packets:        649787
        Failed packets:            0
        Truncated packets:         0
        Retried packets (ENOBUFS): 0
        Retried packets (EAGAIN):  0
```

On Snort side, we have stats like this:

Run time and average rates:

```
===============================================================================
Run time for packet processing was 1826.109403 seconds
Snort processed 507537 packets.
Snort ran for 0 days 0 hours 30 minutes 26 seconds
   Pkts/min:        16917
   Pkts/sec:          277
===============================================================================
```

Packet I/O totals:

```
===============================================================================
Packet I/O Totals:
   Received:       507537
   Analyzed:       507537 (100.000%)
    Dropped:            0 (  0.000%)
   Filtered:            0 (  0.000%)
Outstanding:            0 (  0.000%)
   Injected:            0
===============================================================================
```

Protocol breakdown:

```
===============================================================================
Breakdown by protocol (includes rebuilt packets):
        Eth:       508664 (100.000%)
       VLAN:            0 (  0.000%)
        IP4:       505757 ( 99.429%)
       Frag:            0 (  0.000%)
       ICMP:          207 (  0.041%)
        UDP:       107547 ( 21.143%)
        TCP:       398003 ( 78.245%)
        IP6:            2 (  0.000%)
    IP6 Ext:            2 (  0.000%)
   IP6 Opts:            0 (  0.000%)
      Frag6:            0 (  0.000%)
     ICMP6:             2 (  0.000%)
      UDP6:             0 (  0.000%)
      TCP6:             0 (  0.000%)
    Teredo:             0 (  0.000%)
    ICMP-IP:            0 (  0.000%)
    IP4/IP4:            0 (  0.000%)
    IP4/IP6:            0 (  0.000%)
    IP6/IP4:            0 (  0.000%)
    IP6/IP6:            0 (  0.000%)
        GRE:            0 (  0.000%)
    GRE Eth:            0 (  0.000%)
   GRE VLAN:            0 (  0.000%)
    GRE IP4:            0 (  0.000%)
    GRE IP6:            0 (  0.000%)
GRE IP6 Ext:            0 (  0.000%)
    GRE PPTP:           0 (  0.000%)
    GRE ARP:            0 (  0.000%)
    GRE IPX:            0 (  0.000%)
   GRE Loop:            0 (  0.000%)
       MPLS:            0 (  0.000%)
        ARP:         1806 (  0.355%)
        IPX:            0 (  0.000%)
   Eth Loop:            0 (  0.000%)
   Eth Disc:            0 (  0.000%)
   IP4 Disc:            0 (  0.000%)
   IP6 Disc:            0 (  0.000%)
   TCP Disc:            0 (  0.000%)
   UDP Disc:            0 (  0.000%)
  ICMP Disc:            0 (  0.000%)
All Discard:            0 (  0.000%)
      Other:         1099 (  0.216%)
Bad Chk Sum:        10475 (  2.059%)
    Bad TTL:            0 (  0.000%)
     S5 G 1:          391 (  0.077%)
     S5 G 2:          736 (  0.145%)
      Total:       508664
===============================================================================
```

Alerts Stats:

```
===============================================================================
Action Stats:
     Alerts:         4753 (  0.934%)
     Logged:         4753 (  0.934%)
     Passed:            0 (  0.000%)
Limits:
      Match:            0
      Queue:            0
        Log:            0
      Event:            0
      Alert:           18
Verdicts:
      Allow:       500836 ( 98.680%)
      Block:            0 (  0.000%)
    Replace:            0 (  0.000%)
  Whitelist:         6701 (  1.320%)
  Blacklist:            0 (  0.000%)
     Ignore:            0 (  0.000%)
      Retry:            0 (  0.000%)
===============================================================================
```

Stream Stats:

```
Stream statistics:
            Total sessions: 17555
              TCP sessions: 7202
              UDP sessions: 10353
             ICMP sessions: 0
               IP sessions: 0
                TCP Prunes: 348
                UDP Prunes: 0
               ICMP Prunes: 0
                 IP Prunes: 0
TCP StreamTrackers Created: 7288
TCP StreamTrackers Deleted: 7288
               TCP Timeouts: 86
               TCP Overlaps: 0
       TCP Segments Queued: 109185
     TCP Segments Released: 109185
       TCP Rebuilt Packets: 17276
         TCP Segments Used: 46988
               TCP Discards: 3
                   TCP Gaps: 5418
      UDP Sessions Created: 11498
      UDP Sessions Deleted: 11498
              UDP Timeouts: 1145
              UDP Discards: 0
                    Events: 115209
          Internal Events: 0
          TCP Port Filter
                 Filtered: 0
                Inspected: 0
                  Tracked: 386401
          UDP Port Filter
                 Filtered: 0
                Inspected: 0
                  Tracked: 10353
```

HTTP Inspect:

```
HTTP Inspect - encodings (Note: stream-reassembled packets included):
    POST methods:                       0
    GET methods:                        5822
    HTTP Request Headers extracted:     5825
    HTTP Request Cookies extracted:     0
    Post parameters extracted:          0
    HTTP response Headers extracted:    5791
    HTTP Response Cookies extracted:    528
    Unicode:                            0
    Double unicode:                     0
    Non-ASCII representable:            3
    Directory traversals:               0
    Extra slashes ("//"):               20
    Self-referencing paths ("./"):      0
    HTTP Response Gzip packets extracted: 0
    Gzip Compressed Data Processed:     n/a
    Gzip Decompressed Data Processed:   n/a
    Http/2 Rebuilt Packets:             0
    Total packets processed:            43591
```

SMTP stats:

```
SMTP Preprocessor Statistics
  Total sessions                                : 300
  Max concurrent sessions                       : 184
  Base64 attachments decoded                    : 0
  Total Base64 decoded bytes                    : 0
  Quoted-Printable attachments decoded          : 0
  Total Quoted decoded bytes                    : 0
  UU attachments decoded                        : 0
  Total UU decoded bytes                        : 0
  Non-Encoded MIME attachments extracted        : 0
  Total Non-Encoded MIME bytes extracted        : 0
  SMTP Sessions fastpathed due to memcap exceeded: 2575
```

POP Stats:

```
POP Preprocessor Statistics
  Total sessions                                : 16
  Max concurrent sessions                       : 11
  Base64 attachments decoded                    : 0
  Total Base64 decoded bytes                    : 0
  Quoted-Printable attachments decoded          : 0
  Total Quoted decoded bytes                    : 0
  UU attachments decoded                        : 0
  Total UU decoded bytes                        : 0
  Non-Encoded MIME attachments extracted        : 0
  Total Non-Encoded MIME bytes extracted        : 0
```

We can find above that Alerts reported 4753 (0.934%) which is much more than Snort rules customization. We can look inside Snort alerts to discover more:

```
root@snort-IDS:/var/log/snort# cat alert | grep "Priority: 1" | sort | uniq -c
     16 [Classification: Attempted Administrator Privilege Gain] [Priority: 1]
   4372 [Classification: Potential DoS attack] [Priority: 1]
root@snort-IDS:/var/log/snort#
root@snort-IDS:/var/log/snort# cat alert | grep "Priority: 2" | sort | uniq -c
     15 [Classification: Access to a Potentially Vulnerable Web Application] [Priority: 2]
     24 [Classification: Attempted Information Leak] [Priority: 2]
     84 [Classification: Decode of an RPC Query] [Priority: 2]
     27 [Classification: Misc Attack] [Priority: 2]
      5 [Classification: Potentially Bad Traffic] [Priority: 2]
root@snort-IDS:/var/log/snort#
root@snort-IDS:/var/log/snort# cat alert | grep "Priority: 3" | sort | uniq -c
    193 [Classification: Misc activity] [Priority: 3]
     17 [Classification: Not Suspicious Traffic] [Priority: 3]
root@snort-IDS:/var/log/snort#
```

Priority 1:

- 16 [Classification: Attempted Administrator Privilege Gain] [Priority: 1]
- 4372 [Classification: Potential DoS attack] [Priority: 1]

Priority 2:

- 15 [Classification: Access to a Potentially Vulnerable Web Application] [Priority: 2]
- 24 [Classification: Attempted Information Leak] [Priority: 2]
- 84 [Classification: Decode of an RPC Query] [Priority: 2]
- 27 [Classification: Misc Attack] [Priority: 2]
- [Classification: Potentially Bad Traffic] [Priority: 2]

Priority 3:

- 193 [Classification: Misc activity] [Priority: 3]
- 17 [Classification: Not Suspicious Traffic] [Priority: 3]

We can find Priority 1 has "• 4372 [Classification: Potential DoS attack] [Priority: 1]", which reported by our new DoS rule. And 24 information leaking reported by port scan detection. Let us look at another dataset.

2) Replay LLS_DDOS_1.0-dmz_rewrite.pcap with new Snort rules:

sudo tcpreplay -i vip2 -K -x 7.0 LLS_DDOS_1.0-dmz_rewrite.pcap

```
gns3@gns3vm:~/assign3$ sudo tcpreplay -i vip2  -K -x 7.0 LLS_DDOS_1.0-dmz_rewrite.pcap
File Cache is enabled
Actual: 394089 packets (83138946 bytes) sent in 1704.59 seconds
Rated: 48773.4 Bps, 0.390 Mbps, 231.19 pps
Flows: 54230 flows, 31.81 fps, 388951 flow packets, 4264 non-flow
Statistics for network device: vip2
        Successful packets:        394089
        Failed packets:            0
        Truncated packets:         0
        Retried packets (ENOBUFS): 0
        Retried packets (EAGAIN):  0
gns3@gns3vm:~/assign3$
```

On Snort side, we have stats like this:

Run time and average rates:

```
Run time for packet processing was 1742.78399 seconds
Snort processed 336040 packets.
Snort ran for 0 days 0 hours 29 minutes 2 seconds
   Pkts/min:        11587
   Pkts/sec:          192
```

Packet I/O totals:

```
Packet I/O Totals:
   Received:       336040
   Analyzed:       336040 (100.000%)
    Dropped:            0 (  0.000%)
   Filtered:            0 (  0.000%)
Outstanding:            0 (  0.000%)
   Injected:            0
```

Protocol breakdown:

```
Breakdown by protocol (includes rebuilt packets):
        Eth:       337136 (100.000%)
       VLAN:            0 (  0.000%)
        IP4:       332957 ( 98.760%)
       Frag:            0 (  0.000%)
       ICMP:         2891 (  0.858%)
        UDP:         3284 (  0.974%)
        TCP:       326782 ( 96.929%)
        IP6:            1 (  0.000%)
    IP6 Ext:            1 (  0.000%)
   IP6 Opts:            0 (  0.000%)
      Frag6:            0 (  0.000%)
      ICMP6:            1 (  0.000%)
       UDP6:            0 (  0.000%)
       TCP6:            0 (  0.000%)
     Teredo:            0 (  0.000%)
    ICMP-IP:            0 (  0.000%)
    IP4/IP4:            0 (  0.000%)
    IP4/IP6:            0 (  0.000%)
    IP6/IP4:            0 (  0.000%)
    IP6/IP6:            0 (  0.000%)
        GRE:            0 (  0.000%)
    GRE Eth:            0 (  0.000%)
   GRE VLAN:            0 (  0.000%)
    GRE IP4:            0 (  0.000%)
    GRE IP6:            0 (  0.000%)
GRE IP6 Ext:            0 (  0.000%)
   GRE PPTP:            0 (  0.000%)
    GRE ARP:            0 (  0.000%)
    GRE IPX:            0 (  0.000%)
   GRE Loop:            0 (  0.000%)
       MPLS:            0 (  0.000%)
        ARP:          460 (  0.136%)
        IPX:            0 (  0.000%)
   Eth Loop:         1136 (  0.337%)
   Eth Disc:            0 (  0.000%)
   IP4 Disc:            0 (  0.000%)
   IP6 Disc:            0 (  0.000%)
   TCP Disc:            0 (  0.000%)
   UDP Disc:            0 (  0.000%)
  ICMP Disc:            0 (  0.000%)
All Discard:            0 (  0.000%)
      Other:         2582 (  0.766%)
Bad Chk Sum:            0 (  0.000%)
    Bad TTL:            0 (  0.000%)
     S5 G 1:          427 (  0.127%)
     S5 G 2:          669 (  0.198%)
      Total:       337136
```

Alerts Stats:

```
Action Stats:
     Alerts:         6697 (  1.986%)
     Logged:         6697 (  1.986%)
     Passed:            0 (  0.000%)
Limits:
      Match:            0
      Queue:            0
        Log:            0
      Event:            0
      Alert:          101
Verdicts:
      Allow:       329059 ( 97.923%)
      Block:            0 (  0.000%)
    Replace:            0 (  0.000%)
  Whitelist:         6981 (  2.077%)
  Blacklist:            0 (  0.000%)
     Ignore:            0 (  0.000%)
      Retry:            0 (  0.000%)
```

Stream Stats:

```
Stream statistics:
            Total sessions: 9281
              TCP sessions: 7950
              UDP sessions: 1331
             ICMP sessions: 0
               IP sessions: 0
                TCP Prunes: 0
                UDP Prunes: 0
               ICMP Prunes: 0
                 IP Prunes: 0
TCP StreamTrackers Created: 7951
TCP StreamTrackers Deleted: 7951
                TCP Timeouts: 2
                TCP Overlaps: 0
         TCP Segments Queued: 96399
       TCP Segments Released: 96399
         TCP Rebuilt Packets: 20035
           TCP Segments Used: 54818
                TCP Discards: 7
                    TCP Gaps: 5925
        UDP Sessions Created: 1331
        UDP Sessions Deleted: 1331
                UDP Timeouts: 0
                UDP Discards: 0
                      Events: 90635
             Internal Events: 0
             TCP Port Filter
                    Filtered: 0
                   Inspected: 0
                     Tracked: 325686
             UDP Port Filter
                    Filtered: 0
                   Inspected: 0
                     Tracked: 1331
```

HTTP Inspect:

```
HTTP Inspect - encodings (Note: stream-reassembled packets included):
    POST methods:                       0
    GET methods:                        6749
    HTTP Request Headers extracted:     6759
    HTTP Request Cookies extracted:     0
    Post parameters extracted:          0
    HTTP response Headers extracted:    6719
    HTTP Response Cookies extracted:    551
    Unicode:                            0
    Double unicode:                     0
    Non-ASCII representable:            0
    Directory traversals:               0
    Extra slashes ("//"):               23
    Self-referencing paths ("./"):      0
    HTTP Response Gzip packets extracted: 0
    Gzip Compressed Data Processed:     n/a
    Gzip Decompressed Data Processed:   n/a
    Http/2 Rebuilt Packets:             0
    Total packets processed:            50090
```

SMTP stats:

```
SMTP Preprocessor Statistics
  Total sessions                                : 249
  Max concurrent sessions                       : 184
  Base64 attachments decoded                    : 0
  Total Base64 decoded bytes                    : 0
  Quoted-Printable attachments decoded          : 0
  Total Quoted decoded bytes                    : 0
  UU attachments decoded                        : 0
  Total UU decoded bytes                        : 0
  Non-Encoded MIME attachments extracted        : 0
  Total Non-Encoded MIME bytes extracted        : 0
  SMTP Sessions fastpathed due to memcap exceeded: 3027
```

POP Stats:

```
POP Preprocessor Statistics
  Total sessions                                : 16
  Max concurrent sessions                       : 12
  Base64 attachments decoded                    : 0
  Total Base64 decoded bytes                    : 0
  Quoted-Printable attachments decoded          : 0
  Total Quoted decoded bytes                    : 0
  UU attachments decoded                        : 0
  Total UU decoded bytes                        : 0
```

We can find above that Alerts reported 6697 (1.986%) which is much more than Snort rules customization. We can look inside Snort alerts to discover more:

```
root@snort-IDS:/var/log/snort# cat alert | grep "Priority: 1" | sort | uniq -c
     37 [Classification: Attempted Administrator Privilege Gain] [Priority: 1]
   5498 [Classification: Potential DoS attack] [Priority: 1]
root@snort-IDS:/var/log/snort# cat alert | grep "Priority: 2" | sort | uniq -c
     16 [Classification: Access to a Potentially Vulnerable Web Application] [Priority: 2]
     25 [Classification: Attempted Information Leak] [Priority: 2]
    156 [Classification: Decode of an RPC Query] [Priority: 2]
    124 [Classification: Misc Attack] [Priority: 2]
     20 [Classification: Potentially Bad Traffic] [Priority: 2]
root@snort-IDS:/var/log/snort# cat alert | grep "Priority: 3" | sort | uniq -c
    816 [Classification: Misc activity] [Priority: 3]
      5 [Classification: Not Suspicious Traffic] [Priority: 3]
root@snort-IDS:/var/log/snort#
```

Priority 1:

- 37 [Classification: Attempted Administrator Privilege Gain] [Priority: 1]
- 5498 [Classification: Potential DoS attack] [Priority: 1]

Priority 2:

- 16 [Classification: Access to a Potentially Vulnerable Web Application] [Priority: 2]
- 25 [Classification: Attempted Information Leak] [Priority: 2]
- 156 [Classification: Decode of an RPC Query] [Priority: 2]
- 124 [Classification: Misc Attack] [Priority: 2]
- 20 [Classification: Potentially Bad Traffic] [Priority: 2]

Priority 3:

- 816 [Classification: Misc activity] [Priority: 3]
- 5 [Classification: Not Suspicious Traffic] [Priority: 3]

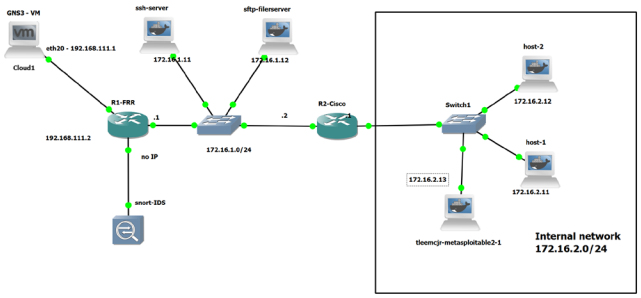
We can find Priority 1 has "• 5498 [Classification: Potential DoS attack] [Priority: 1]", which reported by our new DoS rule. And 25 information leaking reported by port scan detection.

Next, we want to step further to apply Access Control List on

4.6 Access Control List Deployment

In this stage, I want to step further to deploy ACL on R2 on my topology. The purpose is narrow down the detection sample and focus on the issues we want to detect. In order to achieve it, I need to moved victim machine behind R2. Then it will be allocated in Internal network with 172.16.2.13 IP.

4.6.1 Topology revise



From the new topology above, we can see victim server is within 172.16.2.0/24 subnet. Therefore, our ACL on R2 will target on this victim machine. I am going to follow Lab 6 guildline to complete ACL creation [12].

4.6.2 ACL creation

Here, I want to do three things in term of detection efficiency.

```
R2-Cisco#show ip access-lists

R2-Cisco#enable
R2-Cisco#
```

From above, we can see R2 has nothing about access-lists

```
R2-Cisco#show ip interface brief
Interface              IP-Address      OK? Method Status                Protocol
FastEthernet0/0        172.16.1.2      YES NVRAM  up                    up
FastEthernet0/1        172.16.2.1      YES NVRAM  up                    up
FastEthernet1/0        unassigned      YES NVRAM  administratively down down
FastEthernet2/0        unassigned      YES NVRAM  administratively down down
R2-Cisco#
```

Screenshot above tells internet network 172.16.2.0 connect with f0/1

i. Block all ping traffic to Internal network.

From the screenshot below, I was running the commands chain:

```
R2-Cisco(config)#access-list 101 deny icmp any any echo
R2-Cisco(config)#
R2-Cisco(config)#
R2-Cisco(config)#acces
R2-Cisco(config)#access-list 101 permit ip any any
R2-Cisco(config)#
R2-Cisco(config)#end
R2-Cisco#
*Mar  1 01:21:58.247: %SYS-5-CONFIG_I: Configured from console by console
R2-Cisco#
R2-Cisco#
R2-Cisco#conf t
Enter configuration commands, one per line.  End with CNTL/Z.
R2-Cisco(config)#inter f0/0
R2-Cisco(config-if)#
R2-Cisco(config-if)#ip acces
R2-Cisco(config-if)#ip access-group 101 in
R2-Cisco(config-if)#
```

```
R2-Cisco(config)#access-list 101 deny icmp any any echo
R2-Cisco(config)#
R2-Cisco(config)#
R2-Cisco(config)#acces
R2-Cisco(config)#access-list 101 permit ip any any
R2-Cisco(config)#
R2-Cisco(config)#end
R2-Cisco#
*Mar  1 01:21:58.247: %SYS-5-CONFIG_I: Configured from console by console
R2-Cisco#
R2-Cisco#
R2-Cisco#conf t
Enter configuration commands, one per line.  End with CNTL/Z.
R2-Cisco(config)#inter f0/0
R2-Cisco(config-if)#
R2-Cisco(config-if)#ip acces
R2-Cisco(config-if)#ip access-group 101 in
R2-Cisco(config-if)#
```

As the result, all the pings to 172.16.2.0/24 interal network will be filtered. Please check screenshot below:

```
gns3@gns3vm:~/assign3$ ping 172.16.2.13
PING 172.16.2.13 (172.16.2.13) 56(84) bytes of data.
From 172.16.1.2 icmp_seq=1 Packet filtered
From 172.16.1.2 icmp_seq=2 Packet filtered
From 172.16.1.2 icmp_seq=3 Packet filtered
From 172.16.1.2 icmp_seq=4 Packet filtered
From 172.16.1.2 icmp_seq=5 Packet filtered
From 172.16.1.2 icmp_seq=6 Packet filtered
From 172.16.1.2 icmp_seq=7 Packet filtered
From 172.16.1.2 icmp_seq=8 Packet filtered
From 172.16.1.2 icmp_seq=9 Packet filtered
From 172.16.1.2 icmp_seq=10 Packet filtered
From 172.16.1.2 icmp_seq=11 Packet filtered
^C
--- 172.16.2.13 ping statistics ---
11 packets transmitted, 0 received, +11 errors, 100% packet loss, time 10018ms

gns3@gns3vm:~/assign3$ ping 172.16.2.11
PING 172.16.2.11 (172.16.2.11) 56(84) bytes of data.
From 172.16.1.2 icmp_seq=1 Packet filtered
From 172.16.1.2 icmp_seq=2 Packet filtered
From 172.16.1.2 icmp_seq=3 Packet filtered
^C
--- 172.16.2.11 ping statistics ---
3 packets transmitted, 0 received, +3 errors, 100% packet loss, time 2004ms

gns3@gns3vm:~/assign3$ ping 172.16.2.12
PING 172.16.2.12 (172.16.2.12) 56(84) bytes of data.
From 172.16.1.2 icmp_seq=1 Packet filtered
From 172.16.1.2 icmp_seq=2 Packet filtered
From 172.16.1.2 icmp_seq=3 Packet filtered
From 172.16.1.2 icmp_seq=4 Packet filtered
From 172.16.1.2 icmp_seq=5 Packet filtered
```

ii. Deny SNMP queries to vulnerable metasploitable server.

For testing purpose, enable SNMP on metasploitable server firstly:

root@tleemcjr-metasploitable2-1:/# nano /etc/snmp/snmpd.conf

```
agentAddress udp:161,udp6:[::1]:161
rocommunity public
```

Restarted snmp service:

```
root@tleemcjr-metasploitable2-1:/# nano /etc/snmp/snmpd.conf
root@tleemcjr-metasploitable2-1:/# service snmpd restart
Restarting network management services: snmpd.
root@tleemcjr-metasploitable2-1:/#
```

Then, I extended 101 access list on R2:

```
R2-Cisco(config)#
R2-Cisco(config)#ip access-
R2-Cisco(config)#ip access-list exten
R2-Cisco(config)#ip access-list extended 101
R2-Cisco(config-ext-nacl)#
R2-Cisco(config-ext-nacl)#permit ip host 172.16.2.13 any
R2-Cisco(config-ext-nacl)#deny udp any any eq snmp
R2-Cisco(config-ext-nacl)#deny udp any any eq snmptrap
R2-Cisco(config-ext-nacl)#permit ip any any
R2-Cisco(config-ext-nacl)#
R2-Cisco(config-ext-nacl)#inter f0/0
R2-Cisco(config-if)#ip acc
R2-Cisco(config-if)#ip acce
R2-Cisco(config-if)#ip access-group 101 in
R2-Cisco(config-if)#end
R2-Cisco#
R2-Cisco#show
*Mar  1 01:54:36.603: %SYS-5-CONFIG_I: Configured from console by console
R2-Cisco#show ip acce
R2-Cisco#show ip access-lists
Extended IP access list 101
    10 deny icmp any any echo (60 matches)
    20 permit ip any any
    30 permit ip host 172.16.2.13 any
    40 deny udp any any eq snmp
    50 deny udp any any eq snmptrap
R2-Cisco#
```

```
Enter configuration commands, one per line.  End with CNTL/Z.
R2-Cisco(config)#
R2-Cisco(config)#ip access-
R2-Cisco(config)#ip access-list exten
R2-Cisco(config)#ip access-list extended 101
R2-Cisco(config-ext-nacl)#
R2-Cisco(config-ext-nacl)#permit ip host 172.16.2.13 any
R2-Cisco(config-ext-nacl)#deny udp any any eq snmp
R2-Cisco(config-ext-nacl)#deny udp any any eq snmptrap
R2-Cisco(config-ext-nacl)#permit ip any any
R2-Cisco(config-ext-nacl)#
R2-Cisco(config-ext-nacl)#inter f0/0
R2-Cisco(config-if)#ip acc
R2-Cisco(config-if)#ip acce
R2-Cisco(config-if)#ip access-group 101 in
R2-Cisco(config-if)#end
R2-Cisco#
R2-Cisco#show
*Mar  1 01:54:36.603: %SYS-5-CONFIG_I: Configured from console by console
R2-Cisco#show ip acce
R2-Cisco#show ip access-lists
Extended IP access list 101
    10 deny icmp any any echo (60 matches)
    20 permit ip any any
    30 permit ip host 172.16.2.13 any
    40 deny udp any any eq snmp
    50 deny udp any any eq snmptrap
R2-Cisco#
```

As the result, when we try to snmpwalk 172.16.2.13, it will end up with timeout. Please check screenshot below:

```
gns3@gns3vm:~/assign3$ sudo snmpwalk -Os -c public -v 2c 172.16.2.13
Timeout: No Response from 172.16.2.13
gns3@gns3vm:~/assign3$
```

iii. From internal network to the servers, allow only ssh, ftp, and http traffic.

Before doing modification of ACL, let us take a look at listening ports on metasploitables2:

root@tleemcjr-metasploitable2-1:/# netstat -tpl

```
root@tleemcjr-metasploitable2-1:/# netstat -tpl
Active Internet connections (only servers)
Proto Recv-Q Send-Q Local Address           Foreign Address         State       PID/Program name
tcp        0      0 *:telnet                *:*                     LISTEN      920/xinetd
tcp        0      0 *:postgresql            *:*                     LISTEN      602/postgres
tcp        0      0 *:smtp                  *:*                     LISTEN      558/master
tcp        0      0 *:microsoft-ds          *:*                     LISTEN      716/smbd
tcp        0      0 *:exec                  *:*                     LISTEN      920/xinetd
tcp        0      0 *:login                 *:*                     LISTEN      920/xinetd
tcp        0      0 *:shell                 *:*                     LISTEN      920/xinetd
tcp        0      0 *:8009                  *:*                     LISTEN      866/jsvc
tcp        0      0 *:6697                  *:*                     LISTEN      1010/unrealircd
tcp        0      0 *:mysql                 *:*                     LISTEN      271/mysqld
tcp        0      0 *:ircd                  *:*                     LISTEN      1010/unrealircd
tcp        0      0 *:rmiregistry           *:*                     LISTEN      980/rmiregistry
tcp        0      0 *:netbios-ssn           *:*                     LISTEN      716/smbd
tcp        0      0 *:5900                  *:*                     LISTEN      1005/Xtightvnc
tcp        0      0 *:46637                 *:*                     LISTEN      980/rmiregistry
tcp        0      0 *:sunrpc                *:*                     LISTEN      471/portmap
tcp        0      0 *:x11                   *:*                     LISTEN      1005/Xtightvnc
tcp        0      0 *:www                   *:*                     LISTEN      85/apache2
tcp        0      0 *:8787                  *:*                     LISTEN      984/ruby
tcp        0      0 *:8180                  *:*                     LISTEN      866/jsvc
tcp        0      0 *:ingreslock            *:*                     LISTEN      920/xinetd
tcp        0      0 *:ftp                   *:*                     LISTEN      920/xinetd
tcp6       0      0 [::]:ssh                [::]:*                  LISTEN      772/sshd
tcp6       0      0 [::]:postgresql         [::]:*                  LISTEN      602/postgres
tcp6       0      0 [::]:frox               [::]:*                  LISTEN      653/proftpd: (accep
tcp6       0      0 [::]:distcc             [::]:*                  LISTEN      175/distccd
root@tleemcjr-metasploitable2-1:/#
```

Then let us test another ports using **8009 opening port**, other than FTP, HTTP and SSH ports.

```
gns3@gns3vm:~/assign3$ telnet 172.16.2.13 8009
Trying 172.16.2.13...
Connected to 172.16.2.13.
Escape character is '^]'.
```

Above screenshot tells we can access victim server any opening ports.

Next, we need to modify the ACL on R2 to meet the requirements (FTP: TCP port 21, HTTP:80, HTTPS:443, SSH:22):

```
R2-Cisco(config)#
R2-Cisco(config)#ip access
R2-Cisco(config)#ip access-list ext
R2-Cisco(config)#ip access-list extended 101
```

```
R2-Cisco(config-ext-nacl)#
R2-Cisco(config-ext-nacl)#permit tcp any any eq 80
R2-Cisco(config-ext-nacl)#permit tcp any any eq 443
R2-Cisco(config-ext-nacl)#permit tcp any any eq 22
R2-Cisco(config-ext-nacl)#permit tcp any any eq 21
R2-Cisco(config-ext-nacl)#
R2-Cisco(config-ext-nacl)#int
R2-Cisco(config-ext-nacl)#int f0/0
R2-Cisco(config-if)#ip acc
R2-Cisco(config-if)#ip acce
R2-Cisco(config-if)#ip access-group 101 in
R2-Cisco(config-if)#end
R2-Cisco#
*Mar  1 02:54:04.891: %SYS-5-CONFIG_I: Configured from console by console
R2-Cisco#show ip acce
R2-Cisco#show ip access-lists
Extended IP access list 101
    10 deny icmp any any echo (69 matches)
    20 permit ip any any (166 matches)
    30 permit ip host 172.16.2.13 any
    40 deny udp any any eq snmp
    50 deny udp any any eq snmptrap
    60 permit tcp any any eq www
    70 permit tcp any any eq 443
    80 permit tcp any any eq 22
     90 permit tcp any any eq ftp
```

```
R2-Cisco(config)#
R2-Cisco(config)#ip access
R2-Cisco(config)#ip access-list ext
R2-Cisco(config)#ip access-list extended 101
R2-Cisco(config-ext-nacl)#
R2-Cisco(config-ext-nacl)#permit tcp any any eq 80
R2-Cisco(config-ext-nacl)#permit tcp any any eq 443
R2-Cisco(config-ext-nacl)#permit tcp any any eq 22
R2-Cisco(config-ext-nacl)#permit tcp any any eq 21
R2-Cisco(config-ext-nacl)#
R2-Cisco(config-ext-nacl)#int
R2-Cisco(config-ext-nacl)#int f0/0
R2-Cisco(config-if)#ip acc
R2-Cisco(config-if)#ip acce
R2-Cisco(config-if)#ip access-group 101 in
R2-Cisco(config-if)#end
R2-Cisco#
*Mar  1 02:54:04.891: %SYS-5-CONFIG_I: Configured from console by console
R2-Cisco#show ip acce
R2-Cisco#show ip access-lists
Extended IP access list 101
    10 deny icmp any any echo (69 matches)
    20 permit ip any any (166 matches)
    30 permit ip host 172.16.2.13 any
    40 deny udp any any eq snmp
    50 deny udp any any eq snmptrap
    60 permit tcp any any eq www
    70 permit tcp any any eq 443
    80 permit tcp any any eq 22
    90 permit tcp any any eq ftp
R2-Cisco#
```

The commands chain and screenshot tell I just allow TCP connection through port 22, 21, 80 and 443.

Let us do some smoke tests on the result:

On my GNS3-VM:

The following screenshot shows we can SSH victim server:

```
gns3@gns3vm:~/assign3$ ssh root@172.16.2.13
root@172.16.2.13's password:
Last login: Sat Jun 27 12:01:05 2020 from :0.0
Linux 32554753bfe5 4.13.0-21-generic #24-Ubuntu SMP Mon Dec 18 17:29:16 UTC 2017 x86_64

The programs included with the Ubuntu system are free software;
the exact distribution terms for each program are described in the
individual files in /usr/share/doc/*/copyright.

Ubuntu comes with ABSOLUTELY NO WARRANTY, to the extent permitted by
applicable law.

To access official Ubuntu documentation, please visit:
http://help.ubuntu.com/
You have mail.
root@tleemcjr-metasploitable2-1:~#
```

The following screenshot shows we can HTTP victim server:

```
gns3@gns3vm:~/assign3$ curl 172.16.2.13
<html><head><title>Metasploitable2 - Linux</title></head><body>
<pre>

                _                  _       _ _        _     _      ____
 _ __ ___   ___| |_ __ _ ___ _ __ | | ___ (_) |_ __ _| |__ | | ___|___ \
| '_ ` _ \ / _ \ __/ _` / __| '_ \| |/ _ \| | __/ _` | '_ \| |/ _ \ __) |
| | | | | |  __/ || (_| \__ \ |_) | | (_) | | || (_| | |_) | |  __// __/
|_| |_| |_|\___|\__\__,_|___/ .__/|_|\___/|_|\__\__,_|_.__/|_|\___|_____|
                            |_|


Warning: Never expose this VM to an untrusted network!

Contact: msfdev[at]metasploit.com

Login with msfadmin/msfadmin to get started


</pre>
<ul>
<li><a href="/twiki/">TWiki</a></li>
<li><a href="/phpMyAdmin/">phpMyAdmin</a></li>
<li><a href="/mutillidae/">Mutillidae</a></li>
<li><a href="/dvwa/">DVWA</a></li>
<li><a href="/dav/">WebDAV</a></li>
</ul>
</body>
</html>
```

The following screenshot shows we can FTP victim server:

```
gns3@gns3vm:~/assign3$ ftp 172.16.2.13
Connected to 172.16.2.13.
220 (vsFTPd 2.3.4)
Name (172.16.2.13:gns3):
```

The negative tests are shown below:

gns3@gns3vm:~/assign3$ telnet 172.16.2.13 8009

```
gns3@gns3vm:~/assign3$ telnet 172.16.2.13 8009
Trying 172.16.2.13...
telnet: Unable to connect to remote host: No route to host
```

The negative test above shows opening 8009 port does NOT work anymore, because "**telnet: Unable to connect to remote host: No route to host**"

The test cases above tells we can not telnet other ports than 22, 23, 80 and 443

4.6.3 Datasets destination IP rewriting

In this section, we need to rewrite destination IP of **LLS_DDOS_1.0-inside and LLS_DDOS_1.0-dmz** dataset to 172.16.2.13, because we moved this victim machine in internal network.

Rewrite "LLS_DDOS_1.0-dmz_new_c0lumns_rewrite.pcap":
sudo tcprewrite --infile=LLS_DDOS_1.0-dmz.pcap --outfile=LLS_DDOS_1.0-dmz_acl_rewrite.pcap --endpoints=192.168.111.1:172.16.2.13 -b --cachefile=LLS_DDOS_1.0-dmz.cache

Rewrite "LLS_DDOS_1.0-inside_rewrite.pcap":
sudo tcprewrite --infile=LLS_DDOS_1.0-inside.pcap --outfile=LLS_DDOS_1.0-inside_acl_rewrite.pcap --endpoints=192.168.111.1:172.16.2.13 -b --cachefile=LLS_DDOS_1.0-inside.cache

The results are shown below:

```
gns3@gns3vm:~/assign3$ sudo tcprewrite --infile=LLS_DDOS_1.0-dmz.pcap --outfile=LLS_DDOS_1.0-dmz_acl_r
ewrite.pcap --endpoints=192.168.111.1:172.16.2.13 -b --cachefile=LLS_DDOS_1.0-dmz.cache
gns3@gns3vm:~/assign3$
gns3@gns3vm:~/assign3$ sudo tcprewrite --infile=LLS_DDOS_1.0-inside.pcap --outfile=LLS_DDOS_1.0-inside
_acl_rewrite.pcap --endpoints=192.168.111.1:172.16.2.13 -b --cachefile=LLS_DDOS_1.0-inside.cache
gns3@gns3vm:~/assign3$
```

4.6.4 Tcpreplay

- **Replay LLS_DDOS_1.0-inside_acl_rewrite.pcap Results:**

Run time and average rates:

```
===============================================================================
Run time for packet processing was 2051.108195 seconds
Snort processed 529726 packets.
Snort ran for 0 days 0 hours 34 minutes 11 seconds
   Pkts/min:        15580
   Pkts/sec:          258
===============================================================================
```

Packet I/O totals:

```
===============================================================================
Packet I/O Totals:
   Received:       529726
   Analyzed:       529726 (100.000%)
    Dropped:            0 (  0.000%)
   Filtered:            0 (  0.000%)
Outstanding:            0 (  0.000%)
   Injected:            0
===============================================================================
```

Protocol breakdown:

```
===============================================================================
Breakdown by protocol (includes rebuilt packets):
        Eth:       530812 (100.000%)
       VLAN:            0 (  0.000%)
        IP4:       527891 ( 99.450%)
       Frag:            0 (  0.000%)
       ICMP:          210 (  0.040%)
        UDP:       110437 ( 20.805%)
        TCP:       417244 ( 78.605%)
        IP6:            0 (  0.000%)
    IP6 Ext:            0 (  0.000%)
   IP6 Opts:            0 (  0.000%)
      Frag6:            0 (  0.000%)
      ICMP6:            0 (  0.000%)
       UDP6:            0 (  0.000%)
       TCP6:            0 (  0.000%)
     Teredo:            0 (  0.000%)
    ICMP-IP:            0 (  0.000%)
    IP4/IP4:            0 (  0.000%)
    IP4/IP6:            0 (  0.000%)
    IP6/IP4:            0 (  0.000%)
    IP6/IP6:            0 (  0.000%)
        GRE:            0 (  0.000%)
    GRE Eth:            0 (  0.000%)
   GRE VLAN:            0 (  0.000%)
    GRE IP4:            0 (  0.000%)
    GRE IP6:            0 (  0.000%)
GRE IP6 Ext:            0 (  0.000%)
   GRE PPTP:            0 (  0.000%)
    GRE ARP:            0 (  0.000%)
    GRE IPX:            0 (  0.000%)
   GRE Loop:            0 (  0.000%)
       MPLS:            0 (  0.000%)
        ARP:         1814 (  0.342%)
        IPX:            0 (  0.000%)
   Eth Loop:            0 (  0.000%)
   Eth Disc:            0 (  0.000%)
   IP4 Disc:            0 (  0.000%)
   IP6 Disc:            0 (  0.000%)
   TCP Disc:            0 (  0.000%)
   UDP Disc:            0 (  0.000%)
  ICMP Disc:            0 (  0.000%)
All Discard:            0 (  0.000%)
      Other:         1107 (  0.209%)
Bad Chk Sum:        15975 (  3.010%)
    Bad TTL:            0 (  0.000%)
     S5 G 1:          374 (  0.070%)
     S5 G 2:          712 (  0.134%)
      Total:       530812
===============================================================================
```

Alerts Stats:

```
===============================================================================
Action Stats:
     Alerts:            0 (  0.000%)
     Logged:            0 (  0.000%)
     Passed:            0 (  0.000%)
Limits:
      Match:            0
      Queue:            0
        Log:            0
      Event:            0
      Alert:            0
Verdicts:
      Allow:       523070 ( 98.744%)
      Block:            0 (  0.000%)
    Replace:            0 (  0.000%)
  Whitelist:         6656 (  1.256%)
  Blacklist:            0 (  0.000%)
     Ignore:            0 (  0.000%)
      Retry:            0 (  0.000%)
===============================================================================
```

Stream Stats:

```
===============================================================================
Stream statistics:
            Total sessions: 17519
              TCP sessions: 7516
              UDP sessions: 10003
             ICMP sessions: 0
               IP sessions: 0
                TCP Prunes: 408
                UDP Prunes: 0
               ICMP Prunes: 0
                 IP Prunes: 0
TCP StreamTrackers Created: 7588
TCP StreamTrackers Deleted: 7588
               TCP Timeouts: 76
               TCP Overlaps: 0
      TCP Segments Queued: 114002
    TCP Segments Released: 114002
      TCP Rebuilt Packets: 18148
        TCP Segments Used: 49014
               TCP Discards: 57
                  TCP Gaps: 5685
     UDP Sessions Created: 11493
     UDP Sessions Deleted: 11493
               UDP Timeouts: 1490
               UDP Discards: 0
                     Events: 118310
          Internal Events: 0
          TCP Port Filter
                 Filtered: 0
                Inspected: 0
                  Tracked: 400183
          UDP Port Filter
                 Filtered: 0
                Inspected: 0
                  Tracked: 10003
===============================================================================
```

HTTP Inspect:

```
===============================================================================
HTTP Inspect - encodings (Note: stream-reassembled packets included):
    POST methods:                       0
    GET methods:                        6125
    HTTP Request Headers extracted:     6128
    HTTP Request Cookies extracted:     0
    Post parameters extracted:          0
    HTTP response Headers extracted:    6102
    HTTP Response Cookies extracted:    518
    Unicode:                            0
    Double unicode:                     0
    Non-ASCII representable:            3
    Directory traversals:               0
    Extra slashes ("//"):               23
    Self-referencing paths ("./"):      0
    HTTP Response Gzip packets extracted: 0
    Gzip Compressed Data Processed:     n/a
    Gzip Decompressed Data Processed:   n/a
    Http/2 Rebuilt Packets:             0
    Total packets processed:            45874
===============================================================================
```

SMTP stats:
N/A

POP Stats:
N/A

The interesting thing is there is NO alerts at all even though it confirmed to receive around 529726 packets. We still want to double check with alert logs:

```
root@snort-IDS:/var/log/snort# cat alert | grep "Priority: 1" | sort | uniq -c
root@snort-IDS:/var/log/snort# cat alert | grep "Priority: 2" | sort | uniq -c
root@snort-IDS:/var/log/snort# cat alert | grep "Priority: 3" | sort | uniq -c
root@snort-IDS:/var/log/snort#
```

The screenshot shown above tells NO alerts generated after we only allow FTP/HTTP/SSH via ACL. We want to take a look at another PCAP file.

- **Replay LLS_DDOS_1.0-dmz_acl_rewrite.pcap Results:**

Run time and average rates:

```
===============================================================================
Run time for packet processing was 1797.763877 seconds
Snort processed 337259 packets.
Snort ran for 0 days 0 hours 29 minutes 57 seconds
   Pkts/min:        11629
   Pkts/sec:          187
===============================================================================
```

Packet I/O totals:

```
Packet I/O Totals:
   Received:       337259
   Analyzed:       337259 (100.000%)
    Dropped:            0 (  0.000%)
   Filtered:            0 (  0.000%)
Outstanding:            0 (  0.000%)
   Injected:            0
```

Protocol breakdown:

```
Breakdown by protocol (includes rebuilt packets):
        Eth:       338345 (100.000%)
       VLAN:            0 (  0.000%)
        IP4:       334177 ( 98.768%)
       Frag:            0 (  0.000%)
       ICMP:         2970 (  0.878%)
        UDP:         3330 (  0.984%)
        TCP:       327877 ( 96.906%)
        IP6:            1 (  0.000%)
    IP6 Ext:            1 (  0.000%)
   IP6 Opts:            0 (  0.000%)
      Frag6:            0 (  0.000%)
      ICMP6:            1 (  0.000%)
       UDP6:            0 (  0.000%)
       TCP6:            0 (  0.000%)
     Teredo:            0 (  0.000%)
    ICMP-IP:            0 (  0.000%)
    IP4/IP4:            0 (  0.000%)
    IP4/IP6:            0 (  0.000%)
    IP6/IP4:            0 (  0.000%)
    IP6/IP6:            0 (  0.000%)
        GRE:            0 (  0.000%)
    GRE Eth:            0 (  0.000%)
   GRE VLAN:            0 (  0.000%)
    GRE IP4:            0 (  0.000%)
    GRE IP6:            0 (  0.000%)
GRE IP6 Ext:            0 (  0.000%)
   GRE PPTP:            0 (  0.000%)
    GRE ARP:            0 (  0.000%)
    GRE IPX:            0 (  0.000%)
   GRE Loop:            0 (  0.000%)
       MPLS:            0 (  0.000%)
        ARP:          461 (  0.136%)
        IPX:            0 (  0.000%)
   Eth Loop:         1132 (  0.335%)
   Eth Disc:            0 (  0.000%)
   IP4 Disc:            0 (  0.000%)
   IP6 Disc:            0 (  0.000%)
   TCP Disc:            0 (  0.000%)
   UDP Disc:            0 (  0.000%)
  ICMP Disc:            0 (  0.000%)
All Discard:            0 (  0.000%)
      Other:         2574 (  0.761%)
Bad Chk Sum:            0 (  0.000%)
    Bad TTL:            0 (  0.000%)
     S5 G 1:          426 (  0.126%)
     S5 G 2:          660 (  0.195%)
      Total:       338345
```

Alerts Stats:

```
Action Stats:
     Alerts:            0 (  0.000%)
     Logged:            0 (  0.000%)
     Passed:            0 (  0.000%)
Limits:
      Match:            0
      Queue:            0
        Log:            0
      Event:            0
      Alert:            0
Verdicts:
      Allow:       330275 ( 97.929%)
      Block:            0 (  0.000%)
    Replace:            0 (  0.000%)
  Whitelist:         6984 (  2.071%)
  Blacklist:            0 (  0.000%)
     Ignore:            0 (  0.000%)
      Retry:            0 (  0.000%)
```

Stream Stats:

```
Stream statistics:
            Total sessions: 9353
              TCP sessions: 8006
              UDP sessions: 1347
             ICMP sessions: 0
               IP sessions: 0
                TCP Prunes: 0
                UDP Prunes: 0
               ICMP Prunes: 0
                 IP Prunes: 0
TCP StreamTrackers Created: 8006
TCP StreamTrackers Deleted: 8006
               TCP Timeouts: 2
               TCP Overlaps: 4
        TCP Segments Queued: 96053
      TCP Segments Released: 96053
        TCP Rebuilt Packets: 20218
          TCP Segments Used: 56964
               TCP Discards: 15
                   TCP Gaps: 5975
      UDP Sessions Created: 1347
      UDP Sessions Deleted: 1347
               UDP Timeouts: 0
               UDP Discards: 0
                     Events: 93231
          Internal Events: 0
          TCP Port Filter
                  Filtered: 0
                Inspected: 0
                  Tracked: 326791
          UDP Port Filter
                  Filtered: 0
                Inspected: 0
                  Tracked: 1347
```

HTTP Inspect:

```
HTTP Inspect - encodings (Note: stream-reassembled packets included):
    POST methods:                       0
    GET methods:                        6804
    HTTP Request Headers extracted:     6816
    HTTP Request Cookies extracted:     0
    Post parameters extracted:          1
    HTTP response Headers extracted:    6777
    HTTP Response Cookies extracted:    569
    Unicode:                            0
    Double unicode:                     0
    Non-ASCII representable:            0
    Directory traversals:               0
    Extra slashes ("//"):               23
    Self-referencing paths ("./"):      0
    HTTP Response Gzip packets extracted: 0
    Gzip Compressed Data Processed:     n/a
    Gzip Decompressed Data Processed:   n/a
    Http/2 Rebuilt Packets:             0
    Total packets processed:            50368
```

SMTP stats:
N/A

POP Stats:
N/A

As expected, there is NO alerts like .Replay LLS_DDOS_1.0-inside_acl_rewrite.pcap file. We nearly can conclude that ACL can mitigate most of issues generated from original dataset.

```
root@snort-IDS:/var/log/snort# cat alert | grep "Priority: 1" | sort | uniq -c
root@snort-IDS:/var/log/snort# cat alert | grep "Priority: 2" | sort | uniq -c
root@snort-IDS:/var/log/snort# cat alert | grep "Priority: 3" | sort | uniq -c
root@snort-IDS:/var/log/snort#
root@snort-IDS:/var/log/snort#
```

From the results of two datasets, we can conclude ACL dramatically mitigate many types of attacks. Because we can only communicate with target via FTP/HTTP/SSH protocols.

Next, I want to create a tabular formatted data comparison among LLS_DDOS_1.0-dmz.dump and LLS_DDOS_1.0-inside.dump, based on three stages: Original, Applying new rules (DoS and Port detection) and ACL. Then visualize them based on statiscistic data. Finally, we can conclude from those comparison.

4.7 Visualization of DARPA and Hping3 Statistics Results Comparison

I accumulated the results from those several test cases and put into a single table. So, we can easily compare the values among each other. Besides, tabular formatted data will be used for Python Matplotlib visualization.

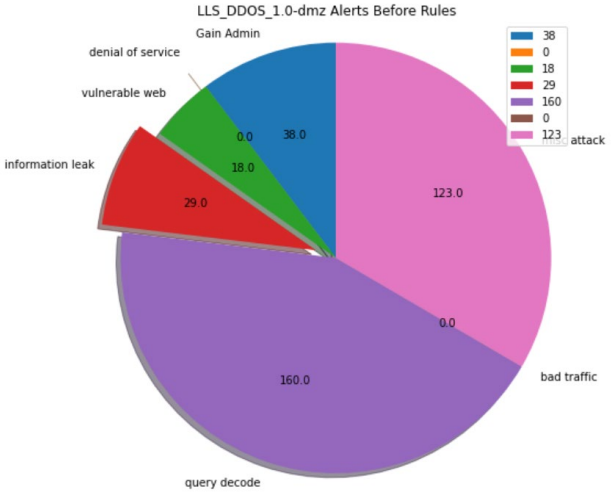
dataset	dmz-before	dmz-after-dos	dmz-after-dos-acl	inside-before	inside-after-dos	inside-after-dos-acl
duration	1779.467	1742.784	1797.764	1741.574	1826.109	2051.108
packages	362321	336040	337259	562229	507537	529726
whitelist	15384	6981	6984	7155	6701	6656
Alerts Total	1223	6697	0	394	4753	0
Admin Gain	38	37	0	17	37	0
DoS	0	5498	0	0	5498	0
Vulnerable web	18	16	0	16	16	0
Information leak	29	25	0	27	25	0
Query decode	160	156	0	89	156	0
badtraffic	0	20	0	5	20	0
miscattack	123	124	0	19	124	0

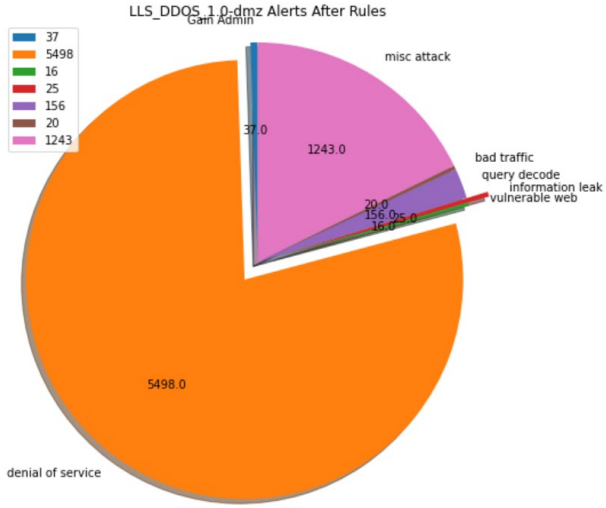
We can visualize analyzed results derived from the table above. I uploaded source code to my GitHub [12].

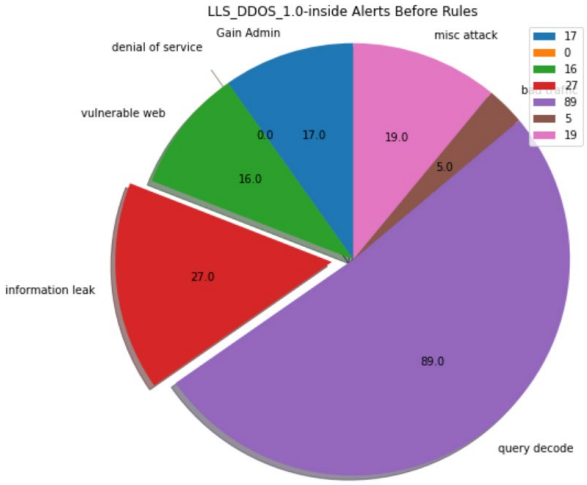
4.7.1 Pie chart of combination of alerts categories

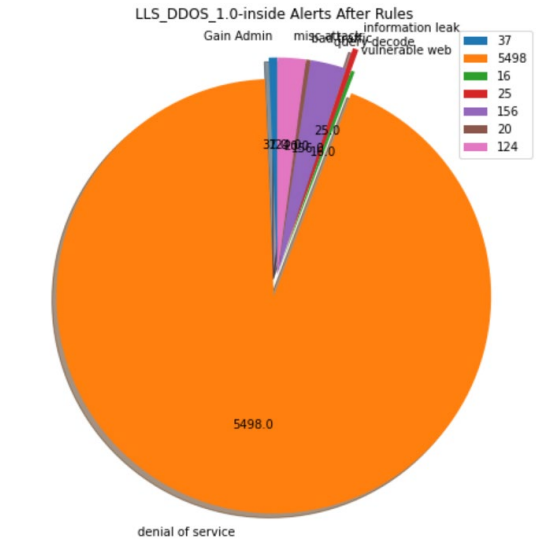
[https://matplotlib.org/3.1.1/gallery/pie_and_polar_charts/pie_features.html]:

```
###snort data analytics****
import pandas as pd
import numpy as np
from sklearn import metrics
import matplotlib.pyplot as plt
import seaborn as sns
import collections


def snortPie(values, title):
    labels = ['Gain Admin','denial of service','vulnerable web','information leak','quer
y decode','bad traffic','misc attack']
    sizes = np.array(values)
    explode = (0, 0.1, 0, 0.1, 0, 0, 0)

    def absolute_value(val):
        a  = np.round(val/100.*sizes.sum(), 0)
        return a
#autopct='%1.1f%%',
    fig1, ax1 = plt.subplots(1, 1,figsize=(8, 8))
    patches, texts, _ = ax1.pie(sizes, explode=explode, labels=labels, autopct=abso
lute_value, shadow=True, startangle=90)
    plt.legend(patches, sizes, loc="best")
    ax1.axis('equal')
    plt.title(title)
    plt.show()

####dmz-before
snortPie([38, 0,  18, 29, 160,  0,  123],'LLS_DDOS_1.0-dmz Alerts Before Rules')
####dmz-after-dos-rule
snortPie([37, 5498, 16, 25, 156,  20, 1243],'LLS_DDOS_1.0-
dmz Alerts After Rules')

####dmz-before
snortPie([17, 0,  16, 27, 89, 5,  19],'LLS_DDOS_1.0-inside Alerts Before Rules')
####dmz-after-dos-rule
snortPie([37, 5498, 16, 25, 156,  20, 124],'LLS_DDOS_1.0-
inside Alerts After Rules')
```









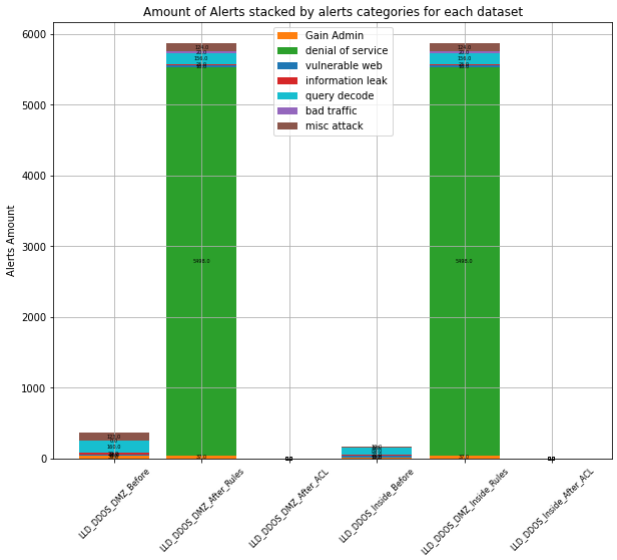
From the pie charts comparison, we can summarize several things:

- After refine Snort rules, we reported much more alerts than before.
- DoS attack is the most threats among all.
- ACL will mitigate most of threats

4.7.2 Stacked bar charts

At the end, we can conduct Stacked bar chart visualization to summize the comparison of alerts among all the datasets, like the graph below:

```
# plot snorts alert of each alert category (https://stackoverflow.com/questions/44309507/stacked-bar-plot-using-matplotlib)
def plot_stacked_bar(data, series_labels, category_labels=None,
                     show_values=False, value_format="{}", y_label=None,
                     colors=None, grid=True, reverse=False):

    ny = len(data[0])
    ind = list(range(ny))

    axes = []
    cum_size = np.zeros(ny)

    data = np.array(data)

    if reverse:
        data = np.flip(data, axis=1)
        category_labels = reversed(category_labels)

    for i, row_data in enumerate(data):
        color = colors[i] if colors is not None else None
        axes.append(plt.bar(ind, row_data, bottom=cum_size,
                            label=series_labels[i], color=color))
        cum_size += row_data

    if category_labels:
        plt.xticks(ind, category_labels, rotation=45, fontsize=8)
        # ax.set_xticklabels(ax.xaxis.get_majorticklabels(), rotation=45)

    if y_label:
        plt.ylabel(y_label)

    plt.legend()

    if grid:
        plt.grid()

    if show_values:
        for axis in axes:
            for bar in axis:
                w, h = bar.get_width(), bar.get_height()
                plt.text(bar.get_x() + w/2, bar.get_y() + h/2,
                         value_format.format(h), ha="center",
                         va="center",fontsize=5)

def snortStackedBar(df_snort, title):
    plt.figure(figsize=(10, 8))

    labels = ['Gain Admin','denial of service','vulnerable web','information leak','query decode','bad traffic','misc attack']
    labels_colors = ['tab:orange', 'tab:green', 'tab:blue', 'tab:red', 'tab:cyan', 'tab:purple', 'tab:brown']

    data = [
        df_snort.admingain,
        df_snort.DoS,
        df_snort.vulnerableweb,
        df_snort.informationleak,
        df_snort.querydecode,
        df_snort.badtraffic,
        df_snort.miscattack
    ]

    datasets = ['LLD_DDOS_DMZ_Before', 'LLD_DDOS_DMZ_After_Rules', 'LLD_DDOS_DMZ_After_ACL', 'LLD_DDOS_Inside_Before', 'LLD_DDOS_DMZ_Inside_Rules', 'LLD_DDOS_Inside_After_ACL']

    plot_stacked_bar(
        data,
        labels,
        category_labels=datasets,
        show_values=True,
        value_format="{:.1f}",
        colors=labels_colors,
        y_label="Alerts Amount"
    )

    plt.title(title)
    plt.show()

snortStackedBar(df_snort, 'Amount of Alerts stacked by alerts categories for each dataset')
```



From the alerts stacked by categories chart, we can easily discover the same things as pie charts:

- After refine Snort rules, we reported much more alerts than before.
- DoS attack is the most threats among all.
- ACL will mitigate most of threats

5 Future work and Conclusions

In conclusions, I experienced hand-on practices on Snort sniffing packets via TcpReplay tools. And I witnessed how rules configuration significantly compact on the network attacks detection through penetration test. We can discover DoS /DDoS/ Ports scan threats through Snort toolsets.

I could conduct more penetration tests and design efficient and effective methods to mitigate the network thread as future work.

REFERENCES

[1] DR. NUR ZINCIR-HEYWOOD, https://web.cs.dal.ca/~zincir/cs6706.html
[2] Iptables tricks : https://opensource.com/article/18/10/iptables-tips-and-tricks
[3] Tcprewrite manual : https://linux.die.net/man/1/tcprewrite
[4] Tcpprep manual: https://tcpreplay.appneta.com/wiki/tcpprep
[5] https://www.doyler.net/security-not-included/fixing-corrupted-capture-files
[6] DoS Concept: https://en.wikipedia.org/wiki/Denial-of-service_attack
[7] Chris Boyce, Nur Zincir-Heywood; "A comparison of four intrusion detection Commerce Research, 2003: https://web.cs.dal.ca/~zincir/bildiri/icecr03-cn.pdf
[8] Duc Le, Lab 6 and 7: https://dal.brightspace.com/
[9] DARPA Intrusion Detection Datasets. https://archive.ll.mit.edu/ideval/data/2000data.html
[10] Hping3: https://tools.kali.org/information-gathering/hping3
[11] Snort rules writing guide : https://paginas.fe.up.pt/~mgi98020/pgr/writing_snort_rules.htm
[12] Emergency rules: https://rules.emergingthreats.net/open/snort-2.9.0/emerging-all.rules
[13] FRR ACL: http://docs.frrouting.org/en/latest/filter.html
[14] Assignment 3 Datasets on OneDrive: https://1drv.ms/u/s!AidJC3PD0CHZndl1PmIa1P994rCP5w?e=d9vhbi
[15] Assignment 3 colab: https://github.com/chrisliu01/network-management/blob/master/assignment3.ipynb