Fake Packet Generation and Detection in Computer Networks

Abhishek Raj
Computer Science and Engineering
Indian Institute of Technology, Dharwad

Rishit Saiya
Computer Science and Engineering
Indian Institute of Technology, Dharwad

***Abstract*—This paper touches upon fake packet generation and their detection using Snort, simulation of a IDS (Intrusion Detection System). This is a base for understanding the Man in the Middle Attacks and their working using Scapy, Wireshark & Snort and networking techniques.**

I. Introduction

In recent times, most of the devices and appliances are now connected together under a network where they can communicate among each other using a set of rules and networking compliance [1]. This totally opens up the surface area of attack on such network instigated a fatal Man in the Middle Attack[2].

This report aims to show some demonstration on how network traffic (in form of packets) can be obfuscated using fake packet generation. We further elaborate on detection of such fake packets in network as a mitigation technique.

II. Analysis of Components

A. Computer Network

A computer network [3] is a group of computers that use a set of common communication protocols over digital interconnections for the purpose of sharing resources located on or provided by the network nodes. The interconnections between nodes are formed from a broad spectrum of telecommunication network technologies, based on physically wired, optical, and wireless radio-frequency methods that may be arranged in a variety of network topologies.

B. Fake Packet

Fake packet are packets that appear as if they are part of the normal communication stream.

1) Fake Packet Injection: Packet injection [4] (also known as forging packets or spoofing packets) in computer networking, is the process of interfering with an established network connection by means of constructing packets to appear as if they are part of the normal communication stream. The packet injection process allows an unknown third party to disrupt or intercept packets from the consenting parties that are communicating, which can lead to degradation or blockage of users' ability to utilize certain network services or protocols. Packet injection is commonly used in man-in-the-middle attacks and denial-of-service attacks.

C. Man In the Middle Attack

Man in the Middle Attack [2] is being used by the NSA over a decade to reverse engineer softwares to help analyze malicious code[17] and malware and can give cybersecurity professionals a better understanding of potential vulnerabilities in their network and systems.

D. Scapy

Scapy [5] is a packet manipulation tool for computer networks, originally written in Python. It can forge or decode packets, send them on the wire, capture them, and match requests and replies. It can also handle tasks like scanning, tracerouting, probing, unit tests, attacks, and network discovery. In this research, we have tried to use Scapy as a tool for fake packet generation.

E. Snort

Snort [6] is an open-source, free and lightweight network intrusion detection system (NIDS) software for Linux and Windows to detect emerging threats. It sniffs packets and spools them straight to the disk and can daemonize itself for background packet logging. We used Snort in this research to detect and analyze the incoming stream of fake packets from another system under same network.

F. Wireshark

Wireshark [7] is a free and open-source packet analyzer. It is used for network troubleshooting, analysis, software and communications protocol development. We used Wireshark in our research to understand analyze pcap files and obtain corresponding graphs and curves.

G. Nmap

Nmap [8] is a free and open-source network scanner. Nmap is used to discover hosts and services on a computer network by sending packets and analyzing the responses. Nmap provides a number of features for probing computer networks, including host discovery and service and operating system detection.

H. State of the Art - From Raw Packet Capture to Advanced Detection Mechanisms

[9] Network Traffic Analysis can be performed in many different ways. Here is a list of the features that characterizes network traffic. Each of these features is part of the OSI model [10]:

- Source & Destination IPs: Provide the source and destination addresses of every packet.
- Protocol: The transport protocol. Typically TCP or UDP.
- Source & Destination Ports: Complete the source and destination addresses.
- Size: The size of the packets.
- Flags: Whether the packet has some flag bits set. These could be: urgent, SYN, ACK, FIN, etc.
- Payload: The data itself, that will be delivered to the application running on destination address and port.

Each of these features can provide valuable information for a NIDS. Today, many corporate switches can export raw data, NetFlow, sFlow or similar data. Net- Flow data contains Source and Destination IP and port, and the amount of traffic transferred per flow.

On a higher level, it is also possible to analyze the payload of every packet. However this requires a full understanding of the protocols by the analyzer, as well as a full access to the traffic, which is not easily scalable.

III. Approaches

A. Initial Approach

In order to get an overall idea about our research topic and to get initial implementation start working, we did a simple IDS implementation which is explained below.

1) Implementation: We setup a system with Ubuntu [12] as base Operating System and a Virtual Machine called Kioptrix [11], where the latter was set as a device connected to the same LAN (Local Area Network) of the former. We thereby integrated Nmap and Snort as IDS in the above mentioned OS respectively. Our main goal was to test limitation of detection for various IDS connected to internet of devices in the same network as well as remote network.

We essentially tried sniff the other active system on the network using the IDS integrated with the detecting system. In Figure 1 and Figure 2, we see that the IDS are switched to sniffing mode.

In the Figure 3 and Figure 4, we can see that each active systems were detected as a traffic in their respective IDS.

Inference: With this small rather effective approach, we learnt that the demonstration of fake packet and detection through a remote system to a desired network is constrained with existing conditions and modern state of the art security measures on a local network only. So, in order to, further escalate our implementation, we built our final path for obfuscation implementation.

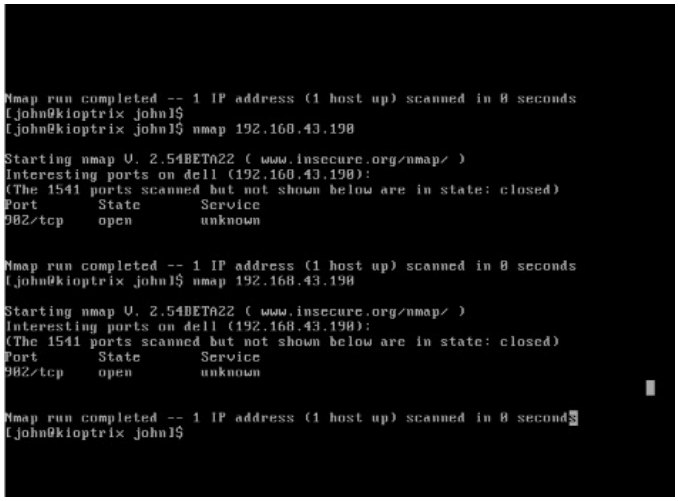

Fig. 1. Nmap Scan (Kioptrix OS) before detection

Fig. 2. Snort Scan (Ubuntu OS) before detection

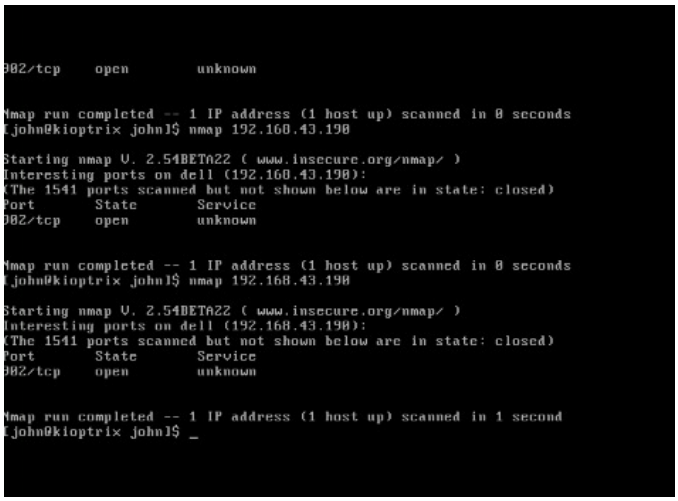

Fig. 3. Nmap Scan (Kioptrix OS) after detection

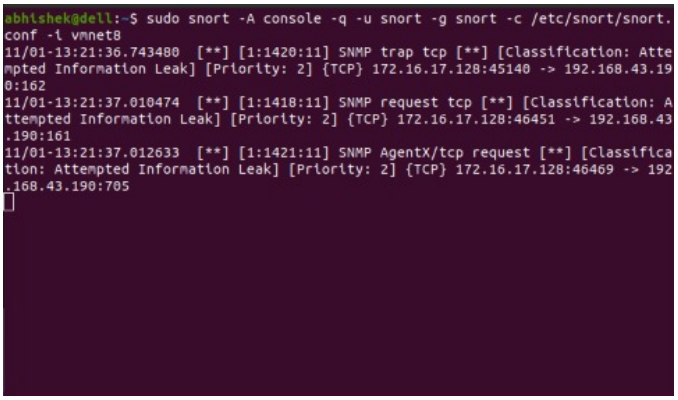

Fig. 4. Snort Scan (Ubuntu OS) after detection

B. Final Approach

With some concepts grasped from the Initial Approach and many trials with our setup and selection of tools, we finally decided upon the following Implementation.

1) Implementation: We started with deploying 2 unique Operating Systems (Ubuntu) in Virtual Machine environment. This was done in order to ensure that all the devices involved in implementation are connected to same LAN.

C. Nomenclature & Environment Setup

We integrated Scapy in OS1 (here on referred as Attacker) and Snort in OS2 (here on referred as Target) as our environment setting up step.

D. Fake Packet Generation (Phase-I)

As mentioned above, we have used Scapy to generate packets and customize the protocols to experiment with the scope of generation as well as the IDS at the Target. WLOG, the sequence of generation of packet was done at the Attacker's end. We initially began with some standard packet generation from Attacker and sending to the Target. The following command was used to generate simple packets stream:

```
send(IP(dst="<dst IP>")/TCP(),count=50)
```

E. Fake Packet Detection (Phase-I)

On the Target's end we used Snort to detect incoming stream of fake packets which were sent by Attacker. Figure 5 shows the Snort detection of stream of fake packets generated from the attacker's end. In order to be affirmative about the authenticity of Snort results, we also sniffed the traffic using Wireshark as well. Figure 6 shows the Wireshark sniffed traffic of fake packets generated from the attacker's end. The following command was used to detect simple packet stream on Target end:

```
sudo snort -A console -q -u snort -g snort
-c /etc/snort/snort.conf -i ens33
```

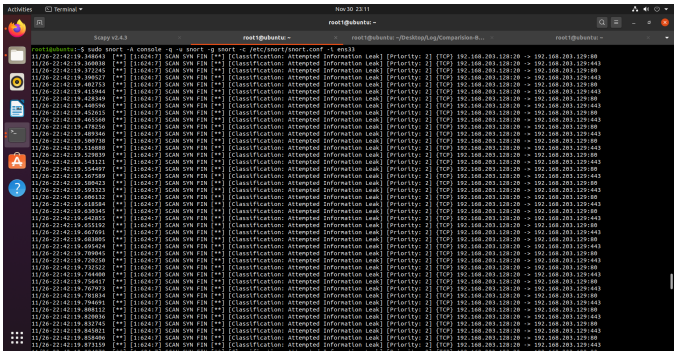

Fig. 5. Snort Sniffing Results - Initial Test

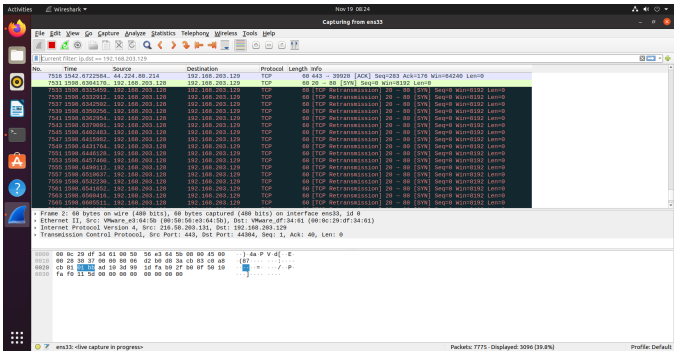

Fig. 6. Wireshark Sniffing Results - Initial Test

IV. Advanced Implementation

In this implementation, we took a step further and tried to mimic the whole generation of fake packet and its detection in attacker's and target sides respectively.

Now, that we had established that fake packet can be generated in a computer network and can be detected as well, we thought of some advanced and novel technique to detect/blacklist the fake packets in a network.

A. Configuration of Snort & Scapy for Detection

Snort: As Snort is an open source utility, we altered the rules in configuration files named `snort.conf`, parallel to our research and requirements.

The reputation preprocessor [13] was created to allow Snort to use a file full of just IP addresses to identify bad hosts and trusted hosts. Malicious IP addresses are stored in blacklists, and trusted IP addresses are stored in whitelists. In the standard installations of Snort, the configuration file is placed at **`/etc/snort/snort.conf`**. In the `snort.conf`, we started with configuring `preprocessor reputation` as follows:

```
preprocessor reputation: \
  memcap 500, \
  priority whitelist, \
  nested_ip inner, \
  scan_local, \
  whitelist $WHITE_LIST_PATH/white_list.rules, \
```

```
blacklist $BLACK_LIST_PATH/black_list.rules
```

Also, we had to configure WHITE_LIST_PATH & BLACK_LIST_PATH as well. It was done as follows (Consider `/etc/snort/rules/iplists` as X):

```
var WHITE_LIST_PATH X
var BLACK_LIST_PATH X
```

Now that we declared that our subnets will be stored in `white_list.rules` & `black_list.rules` files, we declared the absolute path for the files as follows:

```
sudo mkdir X
sudo touch X/black_list.rules
sudo touch X/white_list.rules
```

Scapy: Scapy did not require an external configuration for our requirements, so we did not change anything there.

V. PCAP Encapsulation in Snort & Scapy

Our final goal was to compare and analyse various detection techniques/algorithms which Scapy & Snort were using. So further in this report, we would be comparing the capacity of packets detection over time intervals for Snort vs Scapy.

Before we did that, we had to ensure that there was a proper way to generate `pcap` files in Snort as well as Scapy, so that we could later analyse the `pcap` files in Wireshark.

A. Snort PCAP Encapsulation

Snort stores all its log files at **`/var/log/snort`** with filename as `snort.log.<timestamp>` with `pcap` filetype. So, we extracted the file using timestamp and further analyzed it in Wireshark.

B. Scapy PCAP Encapsulation

After sniffing all the packets, we run the following commands to save the output as `pcap` file to further analyze in Wireshark:

```
a = _
wrpcap("test.pcap", a)
```

VI. Comparison in Detection Efficiency across Scapy & Snort

A. *Whitelist Traffic*

This is a type of traffic which is not blacklisted by Target system and Target's end allows all types of traffic interceptions to that particular subnet in the computer network.

1) Snort Detection: [14] We used the following command for interception of traffic: (Consider `/etc/snort/snort.conf` as X)

```
sudo snort -A console -q -c X -i ens33
```

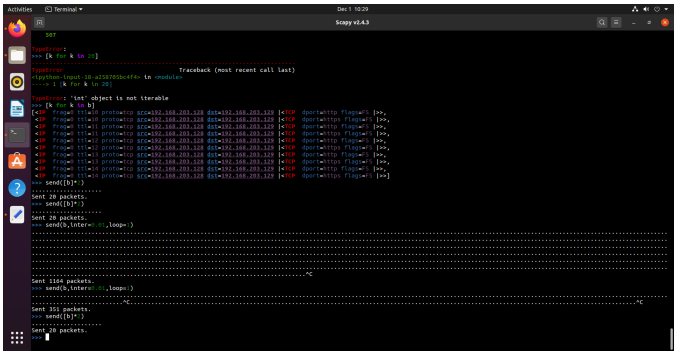

Fig. 7. Scapy - Sending fake TCP packets

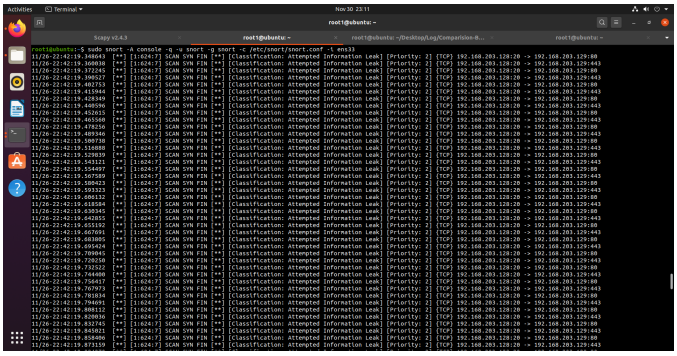

Fig. 8. Snort Sniffing Results - Whitelist Traffic

2) Scapy Detection: [15] We used the following command for interception of traffic:

```
sniff(filter="tcp and host <src IP>",
prn=lambda x:x.summary())
```

Since, the Whitelist traffic is same as in the case of Fake Packet Detection (Phase-I), hence the detection by Snort and Scapy will be similar and rather trivial in Whitelist traffic. However they are shown in Figure 8 and Figure 9.

The Whitelist Traffic which was incoming from Attacker's to Target's end, was intercepted. It was then encapsulated in `pcap` file by the process which is mentioned *PCAP Encapsulation in Snort & Scapy* section.

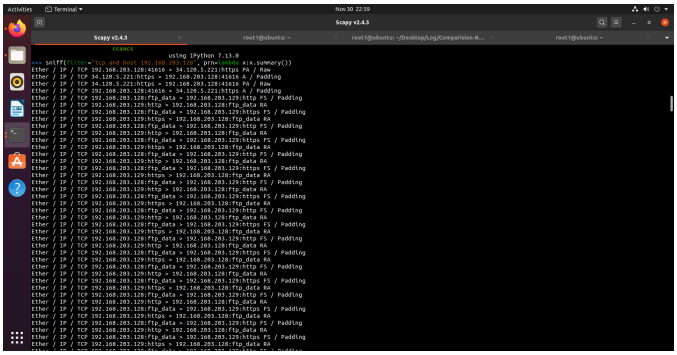

Fig. 9. Scapy Sniffing Results - Whitelist Traffic

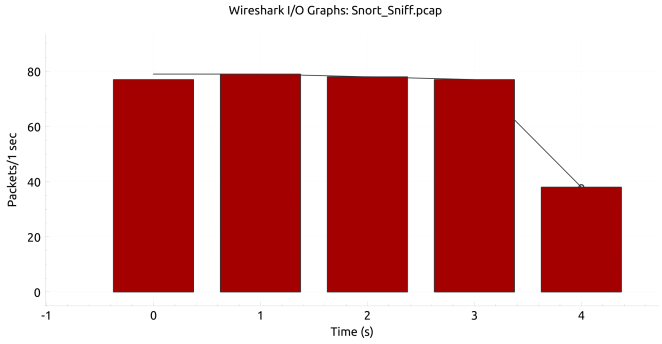

Fig. 10. PCAP Analysis - Snort IDS - Whitelist Traffic

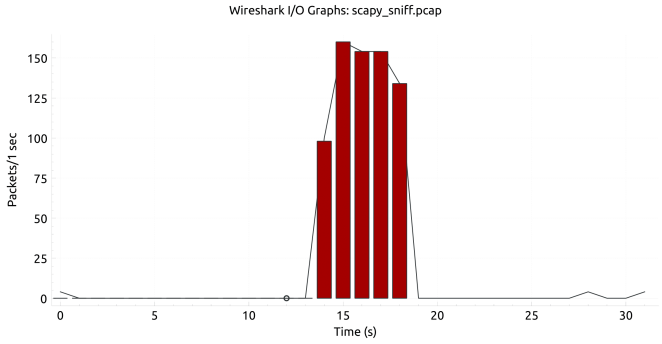

Fig. 11. PCAP Analysis - Scapy IDS - Whitelist Traffic

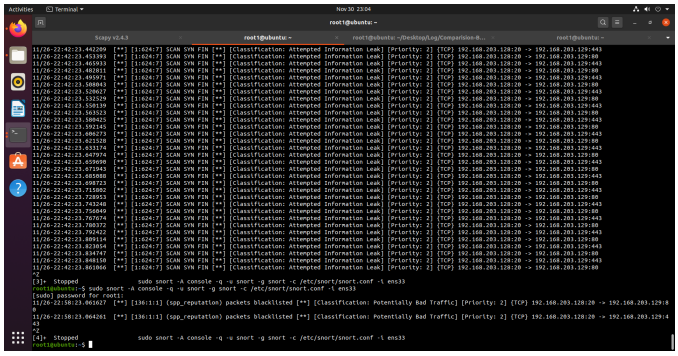

Fig. 12. Snort Sniffing Results - Blacklist Traffic

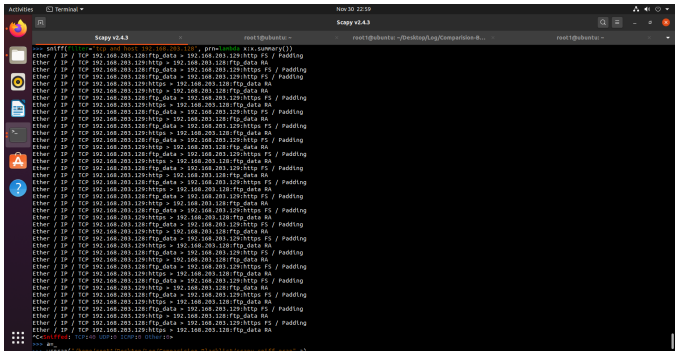

Fig. 13. Scapy Sniffing Results - Blacklist Traffic

Furthermore, Figure 10 & Figure 11 show the Graphical Analysis of `pcap` files in Wireshark [16] of Number of Packets received in intervals of time for Snort & Sniff IDS for Whitelist traffic.

*3) **Comparison - Whitelist Traffic**:* Both Scapy and Snort did great job detecting the traffic here, but since there are several rules which were integrated in Snort, it is slightly slower than Scapy in detection. But because of configuration of those rules, Snort doesn't show every packet which is being sent or received. It shows the packets on priority level of intrusion.

There is no such scope of liberty in Scapy. From Figure 7, we can see that there are 351 packets being sent through Scapy and from graph we can clearly see that Scapy (Figure 11) has detected way more than number of packets sent. Those other extra packets detected in Scapy are from other traffic under the same network. On the contrary, Snort (Figure 10) only shows 351 packets because of the configuration of rules done in Snort.

B. Blacklist Traffic

This is a type of traffic which is blacklisted on a Target system and Target's end doesn't allow all types of traffic interceptions to that particular subnet in the computer network. This is a mitigation technique to non-trusted sources which are under the same network and can cause potential threat to subnets, IoT Devices, other systems under network.

1) Snort Detection: We used the following command for interception of traffic. (Consider `/etc/snort/snort.conf` as X)

```
sudo snort -A console -q -c X -i ens33
```

Since, the Blacklist traffic is different, the Snort Detection also blacklists those packets and those are shown in Figure 12. It also notifies us of the potential threat of incoming traffic packets on Snort IDS.

2) Scapy Detection:

```
sniff(filter="tcp and host <src IP>",
prn=lambda x:x.summary())
```

Furthermore, Figure 14 & Figure 15 show the Graphical Analysis of `pcap` files in Wireshark [16] of Number of Packets received in intervals of time for Snort & Sniff IDS for Blacklist traffic.

*3) **Comparison - Blacklist Traffic**:* Unlike Whitelist Traffic in here, because of the inclusion of the blacklist traffic rule in Snort, it detects and blocks the incoming packets after identifying the packets to be from blacklisted IP addresses. It is because of that we see only 2 packets received (Figure 12) out of 20 packets sent (Figure 7).

On the other hand, Scapy does not have such scope of freedom and hence it detects all the packets being sent from Attacker's end and also some extra traffic in the same network. From graphs we can also observe that Snort (Figure 14) has detected 2 packets and Scapy (Figure 15) has detected

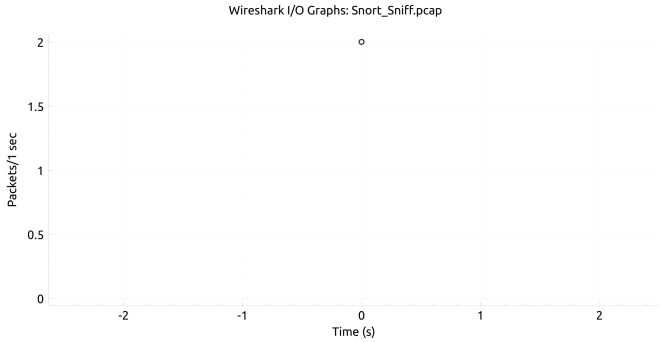

Fig. 14. PCAP Analysis - Snort IDS - Blacklist Traffic

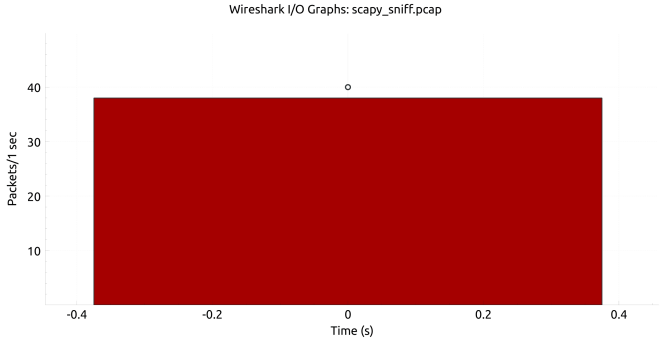

Fig. 15. PCAP Analysis - Scapy IDS - Blacklist Traffic

40 packets. It is because it's count had been set to 40. So after detecting 40 packets, it stopped. Out of which 38 shows TCP error (the red region), because Snort detected them and blocked those packets.

VII. Reconnaissance on Blacklisted IPs

Let us recapitulate on all what research and development was proposed in above scenarios. We generated some fake packets which do not belong to conventional internet traffic but possesses standard protocols like TCP, UDP, etc. We later went on to detect those newly added packets in systems connected under same LAN.

Now, as a priority to keep your system safe (Mitigation Technique), we used the technique of Blacklisting those sub-nets which can pose a potential threat to all the system which are connected under same LAN. Now, in this section, we propose some basic Reconnaissance over the Blacklisted IPs obtained above.

A. Network Reconnaissance on Blacklists

Proceeding over the same idea, we just glanced over some techniques to get some holistic idea of blacklisted IPs. We developed a simple python script which would give a graph diagram to get more idea on how those Blacklisted IPs/subnets are connected over to DNS servers.

```
# !/usr/bin/env python3
from scapy.all import *
```

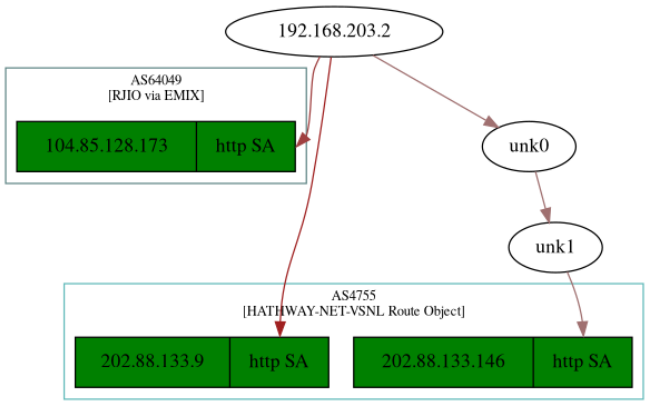

Fig. 16. Graph [Example 1] - Blacklist IP Traffic

```
hosts = ["<host 1>",
"<host 2>", ...]

res,unans = traceroute(hosts)
res.graph(target="> traceroute_graph.svg")
```

Example-2:
We tried the following example where we tried to include the Attacker Blacklisted IP address in the *hosts* array. The script for the same is as follows:

```
# !/usr/bin/env python3
from scapy.all import *

hosts = ["<dst IP>"]

res,unans = traceroute(hosts)
res.graph(target="> traceroute_graph.svg")
```

Example-2:
We also tried following example of sub domains under IIT Dharwad to get to know about incoming traffic and its sources among various Internet Service providers and their DNS servers.

```
# !/usr/bin/env python3
from scapy.all import *

hosts = ["moodle.iitdh.ac.in",
"iitdh.ac.in", "smp.iitdh.ac.in",
"cdc.iitdh.ac.in", "gitea.iitdh.ac.in"]

res,unans = traceroute(hosts)
res.graph(target="> traceroute_graph.svg")
```

B. Graph and Inference

After running the script, we obtained the graph as shown in Figure 16 and Figure 17 for Example 1 and Example 2 respectively.

In Example 1, we had taken the Attacker's IP Address `(src IP)` as the host and tried to get a basic reconnaissance

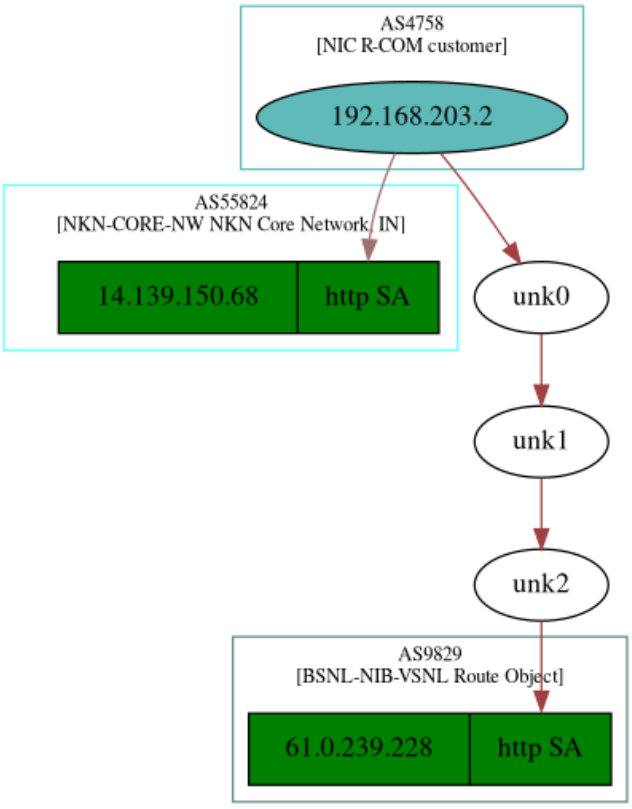

Fig. 17. Graph [Example 2] - Blacklist IP Traffic

on it. As we can see in Figure 16, that one of the ISPs is Hathway Net and other is RJIO. We also got some of the IP addresses `104.85.128.173 & 202.88.133.9/146` using `traceroute` commands in the scripts. A common methodology of attacker would yield potential threat to the Server's IP addresses exposed in Graph. So, putting some good security measures would help us to mitigate obfuscation of malicious traffic and corresponding potential threats.

In Example 2, we had taken the some of sub domains of our Institute domain of `.iitdh.ac.in` as the hosts and tried to get a basic reconnaissance on it. As we can see in Figure 17, that one of the ISPs is NKN (National Knowledge Network) and other is BSNL. We also got some of the IP addresses `14.139.150.68 & 61.0.239.228` using `traceroute` commands in the scripts. A common methodology of attacker would yield potential threat to the Server's IP addresses exposed in Graph. So, putting some good security measures would help us to mitigate obfuscation of malicious traffic and corresponding potential threats.

VIII. Conclusion

The above research focuses upon one of most prevalent Man in the Middle attacks of Fake Packet Generation and Detection Computer Networks. It mentions variety and possibilities of Fake Packet Generation using Scapy Library in Python3. Post various configurations in open source IDS like Snort, we were able to demonstrate the Detection of such non-conventional traffic across various systems connected under same LAN.

As a mitigation Technique, we proposed solutions to segregate IP addresses/subnets into Whitelist and Blacklists Traffic. In this way, we were to restrict systems from receiving potential malicious packets. This also mitigates the threat of circulation of malicious packets which in turn are Trojan to Exploits and hence can essentially lead to failure/crash of Computer Network and systems under that LAN.

By all counts, and with proven results, we also developed a script which gives us some basic information about incoming traffic, DNS server, its routing, etc. in a Graph format.

Acknowledgment

The authors would like to thank the mentor, Prof. Siba Swain (Assistant Professor, Computer Science and Engineering, IIT Dharwad) for giving us the opportunity to work under him. The authors would also like to thank Jay Garchar (Computer Science and Engineering Department, Senior Year, IIT Dharwad) for his constant support and guidance over selection of tools for implementations and helping us in bottlenecks throughout the course of Research and Development.

References

[1] Network Traffic Obfuscation and Automated Internet Censorship
https://arxiv.org/pdf/1605.04044.pdf

[2] Man in the Middle Attack
https://en.wikipedia.org/wiki/Man-in-the-middle_attack

[3] Computer Network
https://en.wikipedia.org/wiki/Computer_network

[4] Packet Injection
https://en.wikipedia.org/wiki/Packet_injection

[5] Scapy
https://scapy.net

[6] Snort
https://www.snort.org

[7] Wireshark
https://www.wireshark.org

[8] Nmap
https://nmap.org

[9] Malicious Traffic Detection in Local Networks with Snort
https://infoscience.epfl.ch/record/141022/files/pdm.pdf

[10] OSI Model
https://en.wikipedia.org/wiki/OSI_model

[11] Kioptrix
https://www.vulnhub.com/entry/kioptrix-level-1-1,22/

[12] Ubuntu OS Distribution
https://ubuntu.com

[13] Reputation Prepocessor
https://sublimerobots.com/2015/12/the-snort-reputation-preprocessor/

[14] Snort Detection
https://resources.infosecinstitute.com/topic/snort-rules-workshop-part-one/

[15] Scapy Techniques & Detection
https://scapy.net/conf/scapy_csw05.pdf

[16] PCAP Analysis & Graphs in Wireshark
https://www.wireshark.org/docs/wsug_html_chunked/ChStatIOGraphs.html

[17] John Hammond, *Working of Ghidra*, https://www.youtube.com/watch?v=aCWI61QX1OU