A Packet Scripting Model for Real-Time Detection of Cyber Attacks

Samera Uga Otor
Department of Mathematics and Computer Science, Benue State University Makurdi
Gboko Road, Makurdi, Benue State, Nigeria.

Beatrice Obianiberi Akumba
Department of Mathematics and Computer Science, Benue State University Makurdi
Gboko Road, Makurdi, Benue State, Nigeria

Adekunle Adedotun Adeyelu
Department of Mathematics and Computer Science, Benue State University Makurdi
Gboko Road, Makurdi, Benue State, Nigeria.

Joshua Ingya
Department of Mathematics and Computer Science, Benue State University Makurdi
Gboko Road, Makurdi, Benue State, Nigeria

ABSTRACT
The dangers of cyberattacks have impacted many businesses and individuals by causing damage to computer systems and networks through malware infiltration, disruption of business activities, and stealing of credentials from users. More often, antiviruses and firewalls have been the first line of defense in the past decades. However, they have proven to be unreliable in recent years due to the evolution of cyber threats, threat landscape in general and zero-day attacks which are new threats developed by hackers, and so, are not known to traditional security defenses. This paper developed a packet scripting model to analyze packets and detect attacks in real-time. The model incorporates the functionalities of TCPdump for packet analysis and Snort which utilizes custom rules to detect attacks in real-time. Practical implementation was achieved through a controlled virtual sandbox environment consisting of virtual machines in a hypervisor, mimicking real-world scenarios for accurate evaluation. Finally, the model's performance was assessed using the real time captured packets to test how well it responds to network traffics. Furthermore, the custom rules were evaluated using an existing bench mark data set to determine how well the rules perform. Results show detection accuracy among others of above 90% for both model dataset and existing dataset.

Keywords
Packet scripting model, real-time detection, cyberattacks, Snort.

1. INTRODUCTION
The influx of cyberattacks globally has affected many businesses, companies, and organizations both private and government. This is due to the ever-changing techniques employed by threat actors. Most organizations often rely on traditional security measures like firewalls and antiviruses but have always been evaded by hackers making them insufficient to protect their devices and networks. A recent report by IBM Security on the cost of data breach 2023, underscores this concern, revealing that the financial toll of a data breach has soared to an astonishing average of $4.45 million per breach [1]. The United States, in particular, bears the brunt with an average cost reaching an alarming rate of $9.05 million. These exorbitant figures encompass not only direct expenses related to detection and response efforts but also the insidious collateral damage inflicted on reputations and the erosion of customer trust.

To safeguard computer systems, networks, and sensitive data from unauthorized access, exploitation, or damage, cybersecurity measures play a crucial role. Among these measures, packet analysis stands out as an integral part of the process. Packet analysis involves inspecting individual data units, known as packets, as they traverse a network. This enables security analysts to gain valuable insights into network activities and identify any suspicious behavior indicative of cyberattacks. Cyberattacks manifest in a variety of forms, each orchestrated by threat actors to compromise information systems, disrupt operations, or steal sensitive data. These attacks include malware infections, denial-of-service (DoS) attacks, phishing attempts, and other exploitation techniques. Effective detection mechanisms must be in place to identify and thwart these threats successfully.

Intrusion detection plays a key role in cybersecurity by identifying unauthorized access or malicious activities within a network. Intrusion detection systems (IDS) monitor network traffic, analyze packet payloads, and compare observed behavior against known attack signatures or patterns.

To address the pressing need for real-time attack detection, particularly in resource-limited environments. Real-time detection of cyberattack has emerged as a critical necessity. By facilitating the immediate analysis of network data, real-time detection enables prompt identification and mitigation of potential cyber threats. This timely response helps prevent significant harm, minimizing the impact on organizational operations and data integrity.

This study focuses on the development of a packet scripting model. This model is an automated approach that leverages scripting techniques, such as bash, and network utilities, such as TCPdump and snort, to analyze network packets and detect attacks in real-time. The primary objective is to provide a lightweight and resource-efficient solution for real-time attack detection.

The proposed packet scripting model builds upon the integration of TCPdump and Snort. TCPdump, is a packet capture and analysis tool [2] that ensures efficient processing of network packets, while Snort, an open-source intrusion detection and prevention system (IDPS) (https://www.snort.org/)[3]that continuously monitors network traffic for signs of malicious activity.

By seamlessly integrating TCPdump and Snort, the packet scripting model was designed to analyze network traffic, swiftly identify potential threats, and automatically respond to them in real-time. This significantly enhanced the efficacy of real-time cyberattack detection, particularly in environments with limited resources.

2. RELATED WORKS

With the increase in use of the internet and technological advancement, cyber threats have become more prevalent. Threat actors benefits from disrupting operations of businesses and cause downtime of network services of companies making security of enterprise networks a big challenge. According to [4], security practitioners have become over-reliant on traditional security measures like intrusion detection systems (IDSs) built on machine learning models which are insufficient because they rely more on already captured datasets which may not contain zero day attacks due to the continuous growth in sophistication and complexity of these attacks. As such, [5] highlighted the importance of network monitoring as a way of repelling attacks and suggested the use of tools for packet capturing and detection of malicious traffic as a good measure of detecting these attacks.

TCPdump, Windump, Honeypot, Snort, suricata, Netflow, Wireshark among others have been identified to monitor network and analyze network traffic [6]. They use predefined rules and pattern matching to identify potential threats. Intrusion Prevention systems (IPS) systems, like Zeek and Sguil, take a step further by not only detecting but also actively preventing intrusions. They employ techniques such as secure mobile agents, virtual machines, and high-throughput string matching for enhanced security [6].

Therefore, [7] presents a forensic tool with the integration of Snort to detect intrusions. Snort was utilized as a packet sniffer to monitor network traffic in real-time for any malicious network behavior or traffic. Despite the Snort’s high accuracy of detection its capability still depends on the nature of rules, the activities being monitored and the context in which it is deployed.

In order to preserve filtering integrity by ensuring that no packet is by passed without being detected, [8], presents two adaptive approaches that provides scalability at the node levels by continuously updating and distributing signature rules and traffic amongst nodes. The approach provides a flexible system by having Snort pre-installed and running on all the nodes. The research shows that the suggested algorithms can divide the workload evenly, allowing the system to scale by altering the number of virtual components that analyze network data.

[9] proposed a cloud-based system for detecting and preventing DDOS/DOS attacks using a Hybrid Intrusion Detection and Prevention System. This system combines signature-based methods and Genetic Algorithms to safeguard the confidentiality, integrity, and availability of cloud services. The approach involves using Snort-IDS and the Splunk web framework for visualization and the genetic algorithm to build anomaly detection model for both benign and zero-day attacks. To achieve a high detection rate, the study improved on the existing Snort-IDS rules. To further enhance snort rules, [10] introduced a heuristic based Snort. This was with the view to help Snort detect specific cyberattacks more effectively. The heuristic preprocessor algorithm was able to rate each packet according to the source IP address and flag assigned.

For the purpose of early detection [11] introduced a detection technique that employed Snort. The system involves feature selection, outlier detection, and classification to enhance intrusion detection accuracy. The novel contributions include an intelligent feature selection method and an entropy-based weighted outlier detection approach. The system's effectiveness lies in its ability to identify intruders early and notify administrators using the Telegram application. The study successfully detected various attacks such as Port scanning, FTP brute force, SSH brute force, and DDoS attacks. Future research suggests adding more rules to Snort for enhanced network security.

Leveraging on the effective packet checking and filtering of extended Berkeley Packet filter (eBPF) capability of the Linux Kernel, [12] developed an IDS. The IDS comprises of two components that runs both on the linux kernel and the user space. Due to the restrictions in the eBPF, some of the default snort rulesets were modified and used by the IDS. The eBPF that runs on the kernel matches the pattern of the packets with the rules and drops the portion that does not match any of the rules. At the user space, the packets left by the eBPF are examined to find matching rule. Experimental results show that the maximum throughput of the IDS system outperformed that of Snort by a factor of 3.

From the literature reviewed and the study in [13], open-source Intrusion Detection Systems (IDSs) like Snort and Suricata are diverse in their configuration of rules, IP address blacklisting and the nature of alert reporting. For instance, these frameworks and tools oftentimes require significant expertise to configure and deploy effectively and their use can generate a high number of false positives. After thorough investigation by [13], the diversity depends on the context in which they are deployed for in real life scenarios. While the study provides valuable insights into IDS diversity, the authors emphasize the need for further analysis in longer durations and real-world deployments to evaluate the practical impact of this diversity.

Additionally, [14] explored the detection rate and precision of Snort, ModSecurity and Nemesida using default rulesets. Results show that the maximum detection rate achieved by default rules and rules set according to operational environment are not sufficient to protect a system effectively since it is lower than expected for known attacks. Therefore, it was opined that only those rules suitable for a particular operational environment should be activated at a time for optimal performance. Moreover, the choice of predefined settings initiated strongly impacts its detection capability and false alarm rate. Likewise, machine learning models require large amounts of training data and can be computationally expensive to run, making them less practical for smaller organizations or those with limited resources [15]. Hence, this study developed a resource-efficient real-time detection models that leverage the capabilities of packet scripting tools like TCPdump and Snort for IDS.

3. METHODOLOGY

The model leverages the functionalities of TCPdump and Snort for real-time cyberattack detection. By combining TCPdump's packet capture efficiency with Snort's intrusion detection capabilities, a robust and responsive intrusion detection system was established. Kali Linux was used as the penetration testing and ethical hacking operating system on Ubuntu to simulates the vulnerable target machine. VMware Workstation Pro was used to create a virtual environment, while a custom rule file, and a Bash script was defined by incorporating TCPdump and Snort for packet analysis.

First a virtual environment was set up and configured. Then a custom bash script was executed to monitor, analyse and evaluate the output. It is as outlined below:

(1) Virtual Environment Setup: Using VMware Workstation Pro, Kali Linux, and Ubuntu, virtual machines were configured to establish a controlled environment for the experiment. The topology of the setup is as shown in Figure 1.

(2) Custom rules: A set of custom rules were defined in a file known as "custom.conf" file. These rules were used by Snort for detection of intrusions or raising an alarm of traffic that traverses the network. Sample rules that were defined in the file custom.conf are:
alert tcp any any -> 192.168.169.130 22 (msg:"Hydra SSH Brute Force Attack Detected"; flags: S; threshold: type limit, track by_src, count 1, seconds 10; sid:1000001;)

Possible DoS Attack
alert tcp any any -> 192.168.169.130 3000 (msg: "DoS SYN Flood Attack Detected"; flags: S; threshold: type both, track by_src, count 10, seconds 10; sid: 1000002;)

NMAP Scan
alert tcp any any -> 192.168.169.130 80 (msg:"Nmap Port Scan Detected"; threshold: type limit, track by_src, count 10, seconds 120; sid:1000003; rev:2;)

Normal ICMP traffic
alert icmp any any -> any any (msg:"Normal ICMP Traffic"; threshold: type limit, track by_src, count 5, seconds 60; sid:1000004; rev:1;)

They can be further seen in Figure 2.

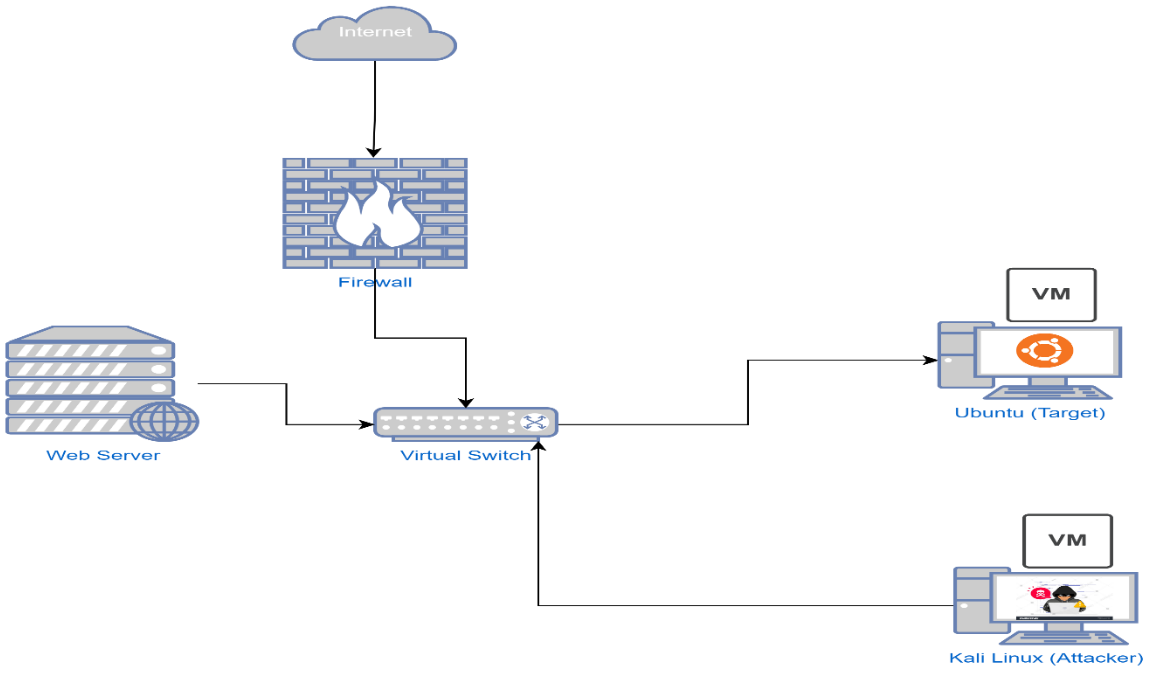
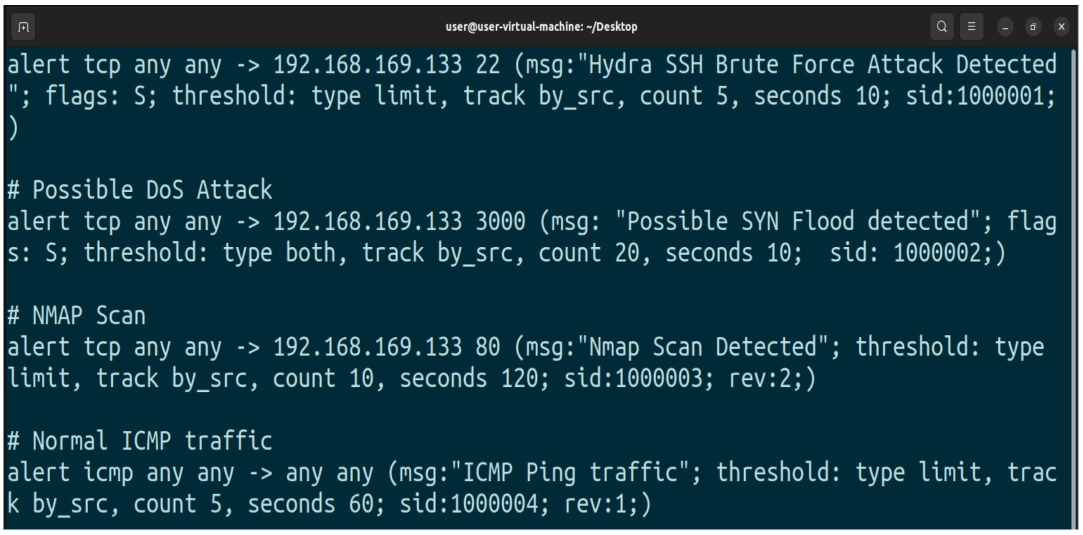


Figure 1: Proposed virtual environment network topology



```
alert tcp any any -> 192.168.169.133 22 (msg:"Hydra SSH Brute Force Attack Detected
"; flags: S; threshold: type limit, track by_src, count 5, seconds 10; sid:1000001;
)

# Possible DoS Attack
alert tcp any any -> 192.168.169.133 3000 (msg: "Possible SYN Flood detected"; flag
s: S; threshold: type both, track by_src, count 20, seconds 10;  sid: 1000002;)

# NMAP Scan
alert tcp any any -> 192.168.169.133 80 (msg:"Nmap Scan Detected"; threshold: type
limit, track by_src, count 10, seconds 120; sid:1000003; rev:2;)

# Normal ICMP traffic
alert icmp any any -> any any (msg:"ICMP Ping traffic"; threshold: type limit, trac
k by_src, count 5, seconds 60; sid:1000004; rev:1;)
```

Figure 2: Defined custom rules in the "custom.conf" file

(3) Rule Path Specification: The path to the custom.conf file was included in snort.conf file which is the default configuration file for Snort located in the /etc/snort/ directory as follows:
# my custom rules
include /home/user/Desktop/custom.conf

This helps the snort to locate the rules and use them for intrusion detection.

(4) Development of the Packet Scripting Model: The packet scripting model, named "SimpleNetAnalyzer.sh," was written using Bash Scripting. The script was written to capture, analyze network packets and detect intrusions or incidents.

(5) Script Execution and Packet Capture: The bash script was executed on the target, Ubuntu VM, utilizing TCPdump to capture incoming and outgoing network packets in real-time. Different traffic scenes were initiated by the Kali Linux attack machine unto the Ubuntu target machine. The model was able to capture packets and detect the various attacks by the Kali Linux attack machine.

(6) Intrusion Detection with Snort: The captured packets were parsed and analyzed by Snort, to detect potential threats or malicious network traffic based on the rules written.

(7) Real-Time Monitoring and Attack Logging: The detected attacks were logged in real-time in the log file. This provided insights on the various kinds of attacks the model was able to detect. The log files help the user to review the packets that came into the network when they are not available to watch the network. The logs are as shown in Figure 3.

Analyzing the captured PCAP and Snort log files reveals network anomalies or signs of compromise in Wireshark as shown in Figure 4. For instance, dissecting a packet, its window size was 1024 bytes which could be indicative of an nmap scan or port scan. Another noteworthy instance involves noticing numerous SYN packets accompanied by RST bits which could indicate a possible Denial of Service (DoS) attack. While ICMP packets as defined in the rules indicate normal traffic. A bunch of SSH traffic encrypts data. However, further investigation from the conversations window shows multiple attempted logins with success on port 22 which could signify a possible SSH brute force attack.

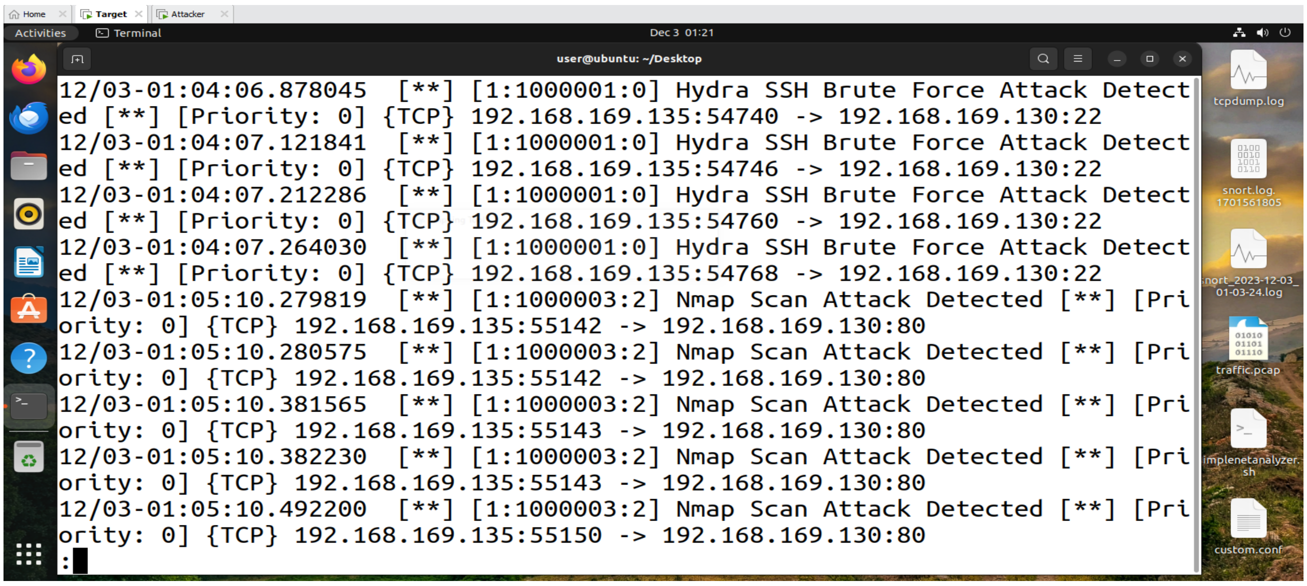
A large no. of SYN packet was also sent from a potential attack source IP address to the target destination IP address (Figure 5). To confirm this, filters were used to help with such results, for example: tcp.flags.syn == 1 and tcp.flags.ack == 0.
Also, as seen from Figure 6 those packets from the source IP to the destination IP with no replies from the target, which is also a good indication that it received a large no. of traffic without being able to respond back.

Furthermore, ICMP packets considered to be normal traffic, where simply used to test if devices are up and running. As shown in Figure 7

4. RESULTS AND DISCUSSIONS

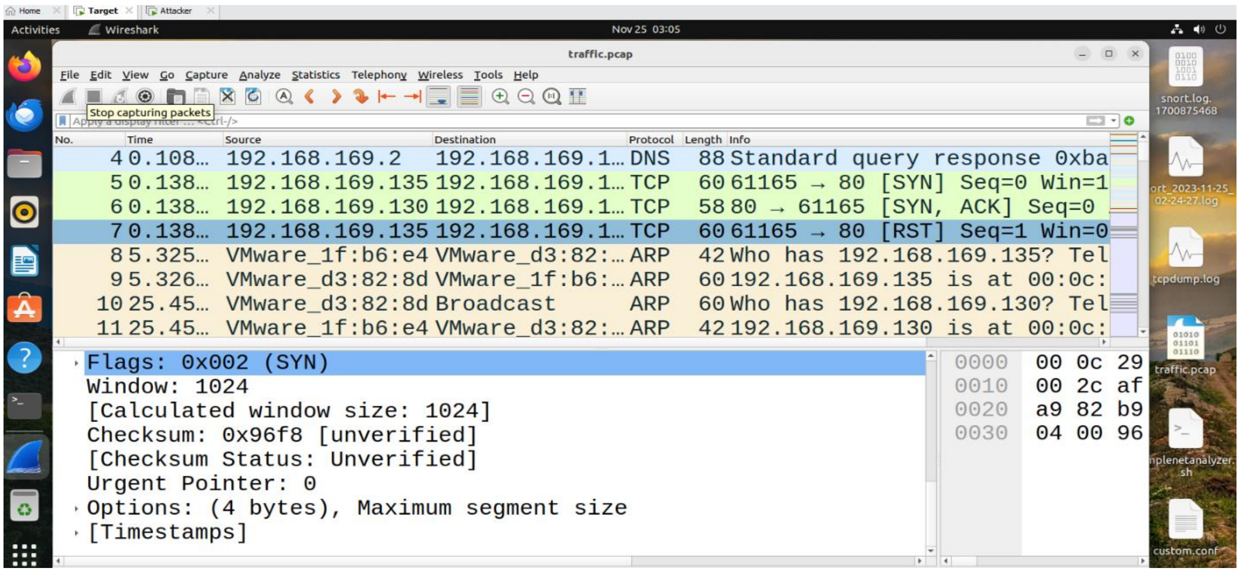
The system was evaluated to ascertain the performance of the system using accuracy, precision, recall and F1 score as metrics. To test the effectiveness of the custom rules, the evaluation was done using the captured logs in real time and an existing dataset MQTT-IoT dataset by [16]. This was with the view to determine how effective the defined custom rules performed even with existing dataset.

The packets captured were logged in a file. they consists of both attacks and benign traffic with information such as the timestamp, source, and destination IP address and a count of how many of the traffic captured were genuine referred to as True Positives (TP), how many were correctly identified as benign, True Negatives (TN), how many were not correctly identified as attacks, False Positives (FP), and how many were genuine but were not identified, False Negatives (FN). This was used to evaluate the metrics.



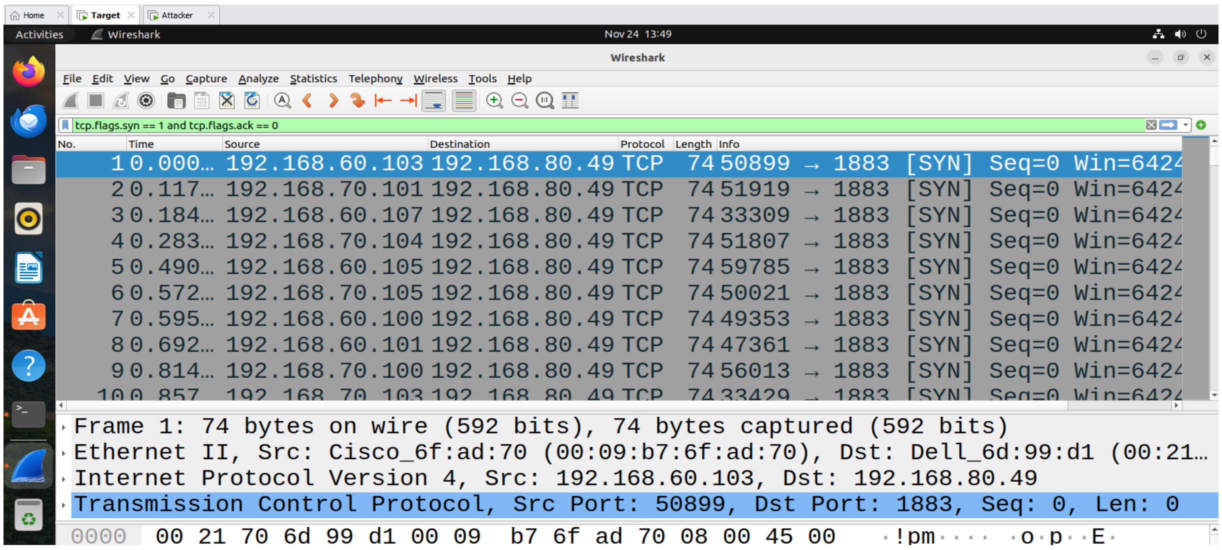
```
12/03-01:04:06.878045  [**] [1:1000001:0] Hydra SSH Brute Force Attack Detect
ed [**] [Priority: 0] {TCP} 192.168.169.135:54740 -> 192.168.169.130:22
12/03-01:04:07.121841  [**] [1:1000001:0] Hydra SSH Brute Force Attack Detect
ed [**] [Priority: 0] {TCP} 192.168.169.135:54746 -> 192.168.169.130:22
12/03-01:04:07.212286  [**] [1:1000001:0] Hydra SSH Brute Force Attack Detect
ed [**] [Priority: 0] {TCP} 192.168.169.135:54760 -> 192.168.169.130:22
12/03-01:04:07.264030  [**] [1:1000001:0] Hydra SSH Brute Force Attack Detect
ed [**] [Priority: 0] {TCP} 192.168.169.135:54768 -> 192.168.169.130:22
12/03-01:05:10.279819  [**] [1:1000003:2] Nmap Scan Attack Detected [**] [Pri
ority: 0] {TCP} 192.168.169.135:55142 -> 192.168.169.130:80
12/03-01:05:10.280575  [**] [1:1000003:2] Nmap Scan Attack Detected [**] [Pri
ority: 0] {TCP} 192.168.169.135:55142 -> 192.168.169.130:80
12/03-01:05:10.381565  [**] [1:1000003:2] Nmap Scan Attack Detected [**] [Pri
ority: 0] {TCP} 192.168.169.135:55143 -> 192.168.169.130:80
12/03-01:05:10.382230  [**] [1:1000003:2] Nmap Scan Attack Detected [**] [Pri
ority: 0] {TCP} 192.168.169.135:55143 -> 192.168.169.130:80
12/03-01:05:10.492200  [**] [1:1000003:2] Nmap Scan Attack Detected [**] [Pri
ority: 0] {TCP} 192.168.169.135:55150 -> 192.168.169.130:80
:
```

Figure 3: Log file showing the different attacks detected



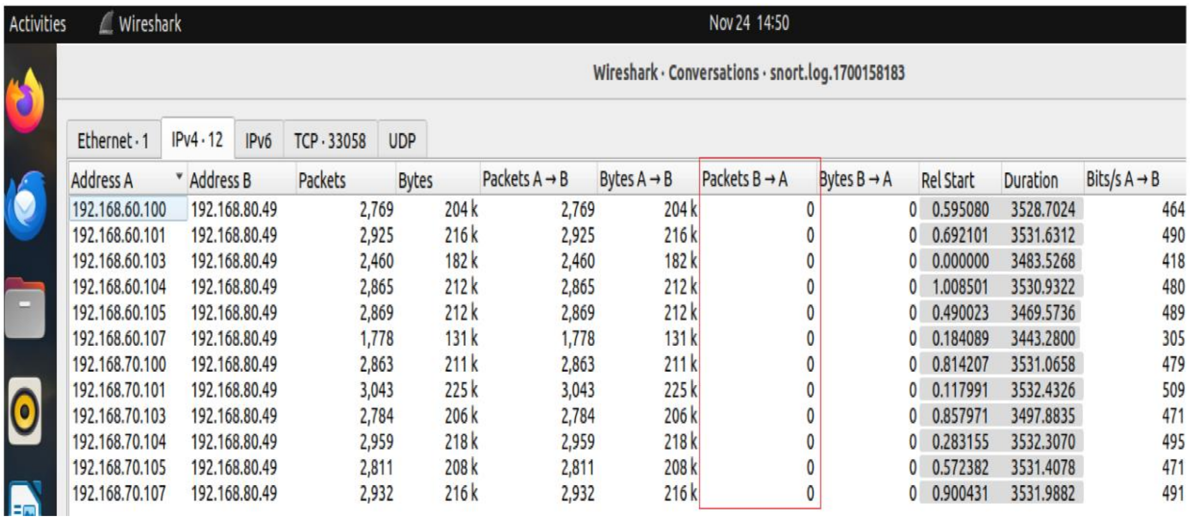
```
No. Time     Source          Destination     Protocol Length Info
  4 0.108…   192.168.169.2   192.168.169.1…  DNS      88 Standard query response 0xba
  5 0.138…   192.168.169.135 192.168.169.1…  TCP      60 61165 → 80 [SYN] Seq=0 Win=1
  6 0.138…   192.168.169.130 192.168.169.1…  TCP      58 80 → 61165 [SYN, ACK] Seq=0
  7 0.138…   192.168.169.135 192.168.169.1…  TCP      60 61165 → 80 [RST] Seq=1 Win=0
  8 5.325…   VMware_1f:b6:e4 VMware_d3:82:…  ARP      42 Who has 192.168.169.135? Tel
  9 5.326…   VMware_d3:82:8d VMware_1f:b6:…  ARP      60 192.168.169.135 is at 00:0c:
 10 25.45…   VMware_d3:82:8d Broadcast       ARP      60 Who has 192.168.169.130? Tel
 11 25.45…   VMware_1f:b6:e4 VMware_d3:82:…  ARP      42 192.168.169.130 is at 00:0c:

› Flags: 0x002 (SYN)                                     0000  00 0c 29
  Window: 1024                                           0010  00 2c af
  [Calculated window size: 1024]                         0020  a9 82 b9
  Checksum: 0x96f8 [unverified]                          0030  04 00 96
  [Checksum Status: Unverified]
  Urgent Pointer: 0
› Options: (4 bytes), Maximum segment size
› [Timestamps]
```

Figure 4: Analysis of the captured traffic in Wireshark


Home
Target
Attacker
Activities
Wireshark
Nov 24 13:49
Wireshark
File Edit View Go Capture Analyze Statistics Telephony Wireless Tools Help

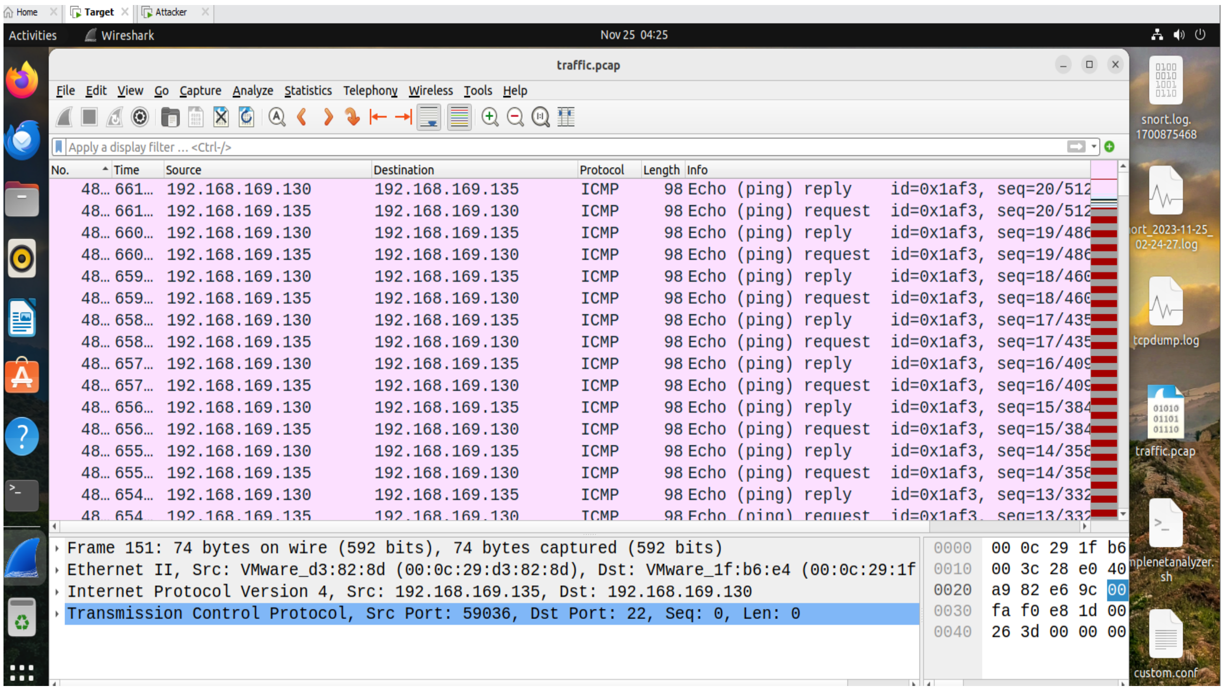
```
tcp.flags.syn == 1 and tcp.flags.ack == 0

No. Time    Source         Destination   Protocol Length Info
  1 0.000…  192.168.60.103 192.168.80.49 TCP      74 50899 → 1883 [SYN] Seq=0 Win=6424
  2 0.117…  192.168.70.101 192.168.80.49 TCP      74 51919 → 1883 [SYN] Seq=0 Win=6424
  3 0.184…  192.168.60.107 192.168.80.49 TCP      74 33309 → 1883 [SYN] Seq=0 Win=6424
  4 0.283…  192.168.70.104 192.168.80.49 TCP      74 51807 → 1883 [SYN] Seq=0 Win=6424
  5 0.490…  192.168.60.105 192.168.80.49 TCP      74 59785 → 1883 [SYN] Seq=0 Win=6424
  6 0.572…  192.168.70.105 192.168.80.49 TCP      74 50021 → 1883 [SYN] Seq=0 Win=6424
  7 0.595…  192.168.60.100 192.168.80.49 TCP      74 49353 → 1883 [SYN] Seq=0 Win=6424
  8 0.692…  192.168.60.101 192.168.80.49 TCP      74 47361 → 1883 [SYN] Seq=0 Win=6424
  9 0.814…  192.168.70.100 192.168.80.49 TCP      74 56013 → 1883 [SYN] Seq=0 Win=6424
 10 0.857…  192.168.70.103 192.168.80.49 TCP      74 33429 → 1883 [SYN] Seq=0 Win=6424

› Frame 1: 74 bytes on wire (592 bits), 74 bytes captured (592 bits)
› Ethernet II, Src: Cisco_6f:ad:70 (00:09:b7:6f:ad:70), Dst: Dell_6d:99:d1 (00:21…
› Internet Protocol Version 4, Src: 192.168.60.103, Dst: 192.168.80.49
› Transmission Control Protocol, Src Port: 50899, Dst Port: 1883, Seq: 0, Len: 0
0000  00 21 70 6d 99 d1 00 09  b7 6f ad 70 08 00 45 00   ·!pm····  ·o·p··E·
```

Figure 5: DoS SYN Flood traffic in Wireshark



Address A	Address B	Packets	Bytes	Packets A → B	Bytes A → B	Packets B → A	Bytes B → A	Rel Start	Duration	Bits/s A → B
192.168.60.100	192.168.80.49	2,769	204 k	2,769	204 k	0	0	0.595080	3528.7024	464
192.168.60.101	192.168.80.49	2,925	216 k	2,925	216 k	0	0	0.692101	3531.6312	490
192.168.60.103	192.168.80.49	2,460	182 k	2,460	182 k	0	0	0.000000	3483.5268	418
192.168.60.104	192.168.80.49	2,865	212 k	2,865	212 k	0	0	1.008501	3530.9322	480
192.168.60.105	192.168.80.49	2,869	212 k	2,869	212 k	0	0	0.490023	3469.5736	489
192.168.60.107	192.168.80.49	1,778	131 k	1,778	131 k	0	0	0.184089	3443.2800	305
192.168.70.100	192.168.80.49	2,863	211 k	2,863	211 k	0	0	0.814207	3531.0658	479
192.168.70.101	192.168.80.49	3,043	225 k	3,043	225 k	0	0	0.117991	3532.4326	509
192.168.70.103	192.168.80.49	2,784	206 k	2,784	206 k	0	0	0.857971	3497.8835	471
192.168.70.104	192.168.80.49	2,959	218 k	2,959	218 k	0	0	0.283155	3532.3070	495
192.168.70.105	192.168.80.49	2,811	208 k	2,811	208 k	0	0	0.572382	3531.4078	471
192.168.70.107	192.168.80.49	2,932	216 k	2,932	216 k	0	0	0.900431	3531.9882	491

Figure 6: DoS attack with zero replies from the target



```
traffic.pcap
No.     Time    Source           Destination      Protocol  Length Info
 48…  661…  192.168.169.130  192.168.169.135  ICMP      98 Echo (ping) reply    id=0x1af3, seq=20/512
 48…  661…  192.168.169.135  192.168.169.130  ICMP      98 Echo (ping) request  id=0x1af3, seq=20/512
 48…  660…  192.168.169.130  192.168.169.135  ICMP      98 Echo (ping) reply    id=0x1af3, seq=19/486
 48…  660…  192.168.169.135  192.168.169.130  ICMP      98 Echo (ping) request  id=0x1af3, seq=19/486
 48…  659…  192.168.169.130  192.168.169.135  ICMP      98 Echo (ping) reply    id=0x1af3, seq=18/460
 48…  659…  192.168.169.135  192.168.169.130  ICMP      98 Echo (ping) request  id=0x1af3, seq=18/460
 48…  658…  192.168.169.130  192.168.169.135  ICMP      98 Echo (ping) reply    id=0x1af3, seq=17/435
 48…  658…  192.168.169.135  192.168.169.130  ICMP      98 Echo (ping) request  id=0x1af3, seq=17/435
 48…  657…  192.168.169.130  192.168.169.135  ICMP      98 Echo (ping) reply    id=0x1af3, seq=16/409
 48…  657…  192.168.169.135  192.168.169.130  ICMP      98 Echo (ping) request  id=0x1af3, seq=16/409
 48…  656…  192.168.169.130  192.168.169.135  ICMP      98 Echo (ping) reply    id=0x1af3, seq=15/384
 48…  656…  192.168.169.135  192.168.169.130  ICMP      98 Echo (ping) request  id=0x1af3, seq=15/384
 48…  655…  192.168.169.130  192.168.169.135  ICMP      98 Echo (ping) reply    id=0x1af3, seq=14/358
 48…  655…  192.168.169.135  192.168.169.130  ICMP      98 Echo (ping) request  id=0x1af3, seq=14/358
 48…  654…  192.168.169.130  192.168.169.135  ICMP      98 Echo (ping) reply    id=0x1af3, seq=13/332
 48…  654…  192.168.169.135  192.168.169.130  ICMP      98 Echo (ping) request  id=0x1af3, seq=13/332

› Frame 151: 74 bytes on wire (592 bits), 74 bytes captured (592 bits)
› Ethernet II, Src: VMware_d3:82:8d (00:0c:29:d3:82:8d), Dst: VMware_1f:b6:e4 (00:0c:29:1f
› Internet Protocol Version 4, Src: 192.168.169.135, Dst: 192.168.169.130
› Transmission Control Protocol, Src Port: 59036, Dst Port: 22, Seq: 0, Len: 0

0000  00 0c 29 1f b6
0010  00 3c 28 e0 40
0020  a9 82 e6 9c 00
0030  fa f0 e8 1d 00
0040  26 3d 00 00 00
```

Home
Target
Attacker
Activities
Wireshark
Nov 25 04:25
File Edit View Go Capture Analyze Statistics Telephony Wireless Tools Help
Apply a display filter … <Ctrl-/>
snort.log.1700875468
ort_2023-11-25_02-24-27.log
tcpdump.log
traffic.pcap
nplenetanalyzer.sh
custom.conf

Figure 7: ICMP traffic

To evaluate the MQTT-IoT dataset, the following steps were followed:

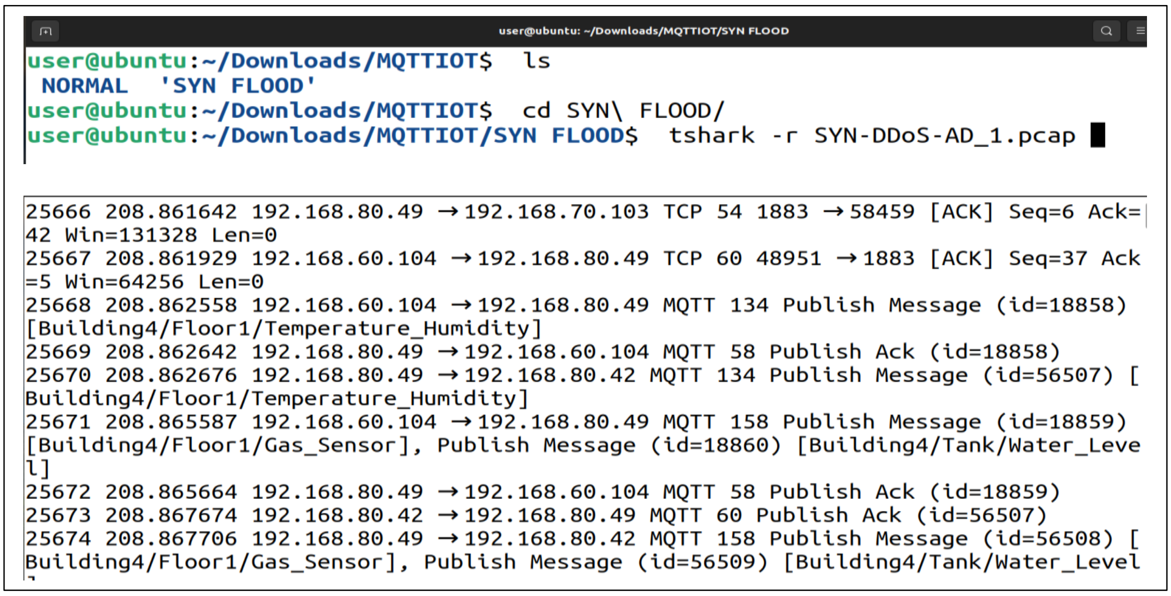
1. Feature Extraction: Tshark which is a terminal-based tool for packet analysis was used in this case to read the pcap files from the MQTT-IoT dataset which gave an insight to defining Snort rules to be used by snort in reading the files as seen in Figure 8.

2. Rule definition: the rules were modified to conform to the features seen in both the normal and malicious pcap files.

3. Next, Snort was used to read the pcap files for potential alerts for both the normal and SYN Flood pcap files.

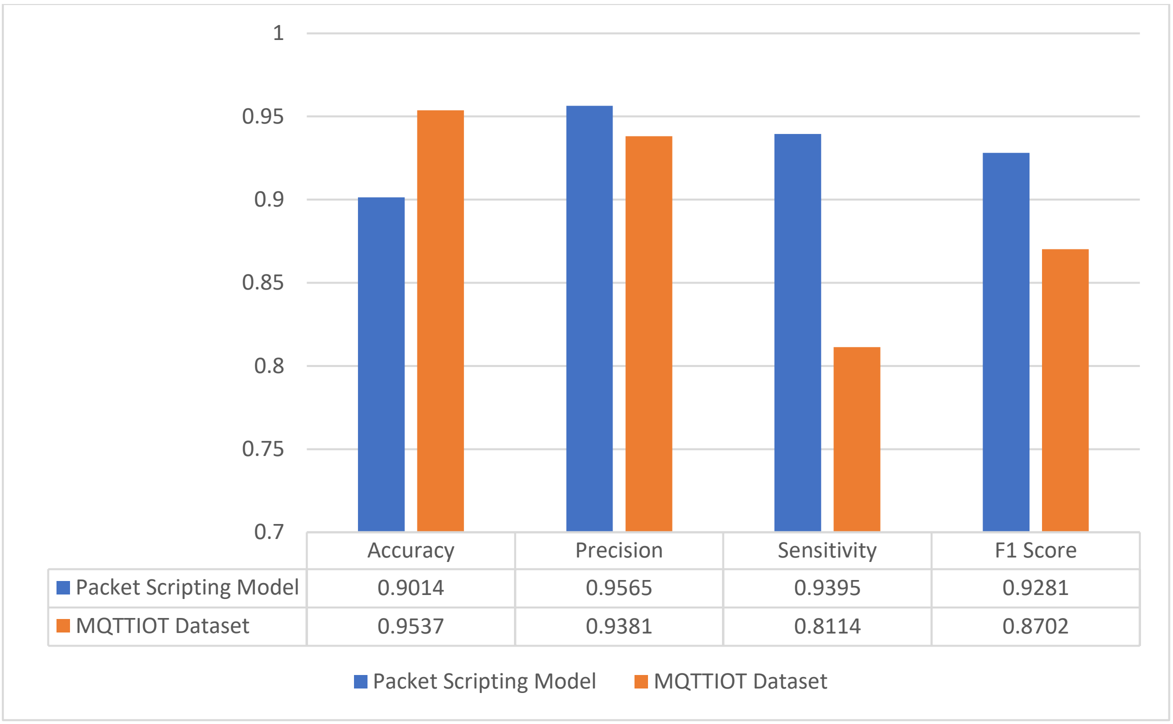
The result obtained from the proposed real time packet scripting model and the existing dataset using the custom rules for accuracy, precision, recall and F1 score are as presented in Figure 9.

The results illustrates how well both systems performed under similar scenarios. The packet scripting model achieved an accuracy of 90.14% while the MQTT-IOT dataset achieved an accuracy of 95.65%.



```
user@ubuntu: ~/Downloads/MQTTIOT/SYN FLOOD
user@ubuntu:~/Downloads/MQTTIOT$  ls
 NORMAL  'SYN FLOOD'
user@ubuntu:~/Downloads/MQTTIOT$  cd SYN\ FLOOD/
user@ubuntu:~/Downloads/MQTTIOT/SYN FLOOD$  tshark -r SYN-DDoS-AD_1.pcap

25666 208.861642 192.168.80.49 →192.168.70.103 TCP 54 1883 →58459 [ACK] Seq=6 Ack=
42 Win=131328 Len=0
25667 208.861929 192.168.60.104 →192.168.80.49 TCP 60 48951 →1883 [ACK] Seq=37 Ack
=5 Win=64256 Len=0
25668 208.862558 192.168.60.104 →192.168.80.49 MQTT 134 Publish Message (id=18858)
[Building4/Floor1/Temperature_Humidity]
25669 208.862642 192.168.80.49 →192.168.60.104 MQTT 58 Publish Ack (id=18858)
25670 208.862676 192.168.80.49 →192.168.80.42 MQTT 134 Publish Message (id=56507) [
Building4/Floor1/Temperature_Humidity]
25671 208.865587 192.168.60.104 →192.168.80.49 MQTT 158 Publish Message (id=18859)
[Building4/Floor1/Gas_Sensor], Publish Message (id=18860) [Building4/Tank/Water_Leve
l]
25672 208.865664 192.168.80.49 →192.168.60.104 MQTT 58 Publish Ack (id=18859)
25673 208.867674 192.168.80.42 →192.168.80.49 MQTT 60 Publish Ack (id=56507)
25674 208.867706 192.168.80.49 →192.168.80.42 MQTT 158 Publish Message (id=56508) [
Building4/Floor1/Gas_Sensor], Publish Message (id=56509) [Building4/Tank/Water_Level
```

Figure 8: Feature extraction from the MQTT-IoT dataset using Tshark



	Accuracy	Precision	Sensitivity	F1 Score
🟦 Packet Scripting Model	0.9014	0.9565	0.9395	0.9281
🟧 MQTTIOT Dataset	0.9537	0.9381	0.8114	0.8702

Figure 9: Results for the proposed model versus MQTTIOT dataset

This is consistent with the result obtained using some machine learning models in [16]. Therefore, the custom rules defined in our proposed model are effective in detecting attacks. The Precision, Recall and F1 scores for the packet scripting model are 95.65%, 93.95% and 92.81% while that of the MQTT-IOT dataset are 93.81%, 81.14% and 87.702% using the custom rules defined.

5. CONCLUSIONS AND RECOMMENDATIONS

This paper proposed a packet scripting model used to detect attacks in real time. The model was developed using snort and TCPdump. A virtual environment was set up using VMware Workstation Pro, custom rules were defined using bash script incorporating TCPdump and Snort for packet analysis. The bash script was executed on the target Ubuntu VM, utilizing TCPdump to capture incoming and outgoing network packets in real-time.

The captured packets were parsed and analyzed by Snort, to detect potential threats or malicious network traffic based on the rules written. The captured packets were analysed using accuracy, precision, recall and F1 score as metrics. Furthermore, the custom rules were tested on an existing dataset, the MQTT-IOT by [16] to determine how effective the rules perform in detecting attacks. Results showed that our defined rules perform very well in detecting attacks in real time and already captured data. The study recommends that the custom rules be further tested on other datasets.

6. REFERENCES

[1] Pop, C. (2024). Endpoint Protector Blog. The Cost of a Data Breach in 2023: Data Loss Prevention: https://www.endpointprotector.com/blog/cost-of-a-data-breach-2023/ January 2nd 2024

[2] Sanders, C., & Smith, J. (2014). Packet analysis. *Applied Network Security Monitoring*, 341–384. https://doi.org/10.1016/b978-0-12-417208-1.00013-1

[3] Snort. *Network Intrusion Detection & Prevention System*. (n.d.). https://www.snort.org/

[4] Prabhu, S., & Bhat, S. (2020). Cyber Attacks Mitigation: Detecting Malicious Activities in Network Traffic – A Review of Literature. *International Journal of Case Studies in Business, IT, and Education (IJCSBE)*, 4(2), 40-64 August 2020. http://doi.org/10.5281

[5] Alsharabi, N., Alqunun, M., & Murshed, B. A. (2023). Detecting unusual activities in local network using Snort and wireshark tools. *Journal of Advances in Information Technology*, *14*(4), pp 616–624. https://doi.org/10.12720/jait.14.4.616-624

[6] Thapa, S., & Mailewa, A. (2020, April 28). *The role of intrusion detection/prevention systems in Modern Computer Networks: A Review*. Easy Chair Home Page. https://easychair.org/publications/preprint/jMT5

[7] Mogaji, S. A., Ayeni, O. A., & Olutayo, V. A. (2022). Analysis of digital forensics in the implementation of intrusion detection using Snort. *FUOYE Journal of Pure and Applied Sciences (FJPAS)*, *7*(1), pp 100–107. https://doi.org/10.55518/fjpas.ijms6335

[8] Haugerud, H., Tran, H. N., Aitsaadi, N., & Yazidi, A. (2021). A dynamic and scalable parallel network intrusion detection system using intelligent rule ordering and network function virtualization. *Future Generation Computer Systems*, 124, pp 254–267. https://doi.org/10.1016/j.future.2021.05.037

[9] Nsabimana, T., Bimenyimana, C. I., Odumuyiwa, V., & Hounsou, J. T. (2020). Detection and prevention of criminal attacks in cloud computing using a hybrid intrusion detection systems. *In Intelligent Human Systems Integration 2020, Proceedings of the 3rd International Conference on Intelligent Human Systems Integration (IHSI 2020): Integrating People and Intelligent Systems, February 19-21, 2020, Modena, Italy* (pp.667-676)

[10] Gdowski, B., Kościej, R., & Niemiec, M. (2021). Heuristic-based intrusion detection functionality in a snort environment. *Information &amp; Security: An International Journal*, *50*, pp 23–36. https://doi.org/10.11610/isij.5010

[11] Erlansari, A., Coastera, F. F., & Husamudin, A. (2020). Early intrusion detection system (IDS) using Snort and telegram approach. *SISFORMA*, *7*(1), pp 21–27. https://doi.org/10.24167/sisforma.v7i1.2629

[12] Wang S. & Chang J. (2022). Design and implementation of an intrusion detection system by using Extended BPF in the Linux kernel, *Journal of Network and Computer Applications*, 198, 103283, ISSN 1084-8045, https://doi.org/10.1016/j.jnca.2021.103283.

[13] Asad, H., & Gashi, I. (2021). Dynamical analysis of diversity in rule-based open source network intrusion detection systems - *empirical software engineering*. SpringerLink. October 22 https://link.springer.com/article/10.1007%2Fs10664-021-10046-w

[14] Díaz-Verdejo J, Muñoz-Calle J, Estepa Alonso A, Estepa Alonso R, Madinabeitia G (2022). On the Detection Capabilities of Signature-Based Intrusion Detection Systems in the Context of Web Attacks. *Applied Sciences*. 2022; 12(2):852. https://doi.org/10.3390/app12020852

[15] Al-Fawa'reh, M., Al-Fayoumi, M., Nashwan, S., & Fraihat, S. (2022). Cyber threat intelligence using PCA-DNN model to detect abnormal network behavior. *Egyptian Informatics Journal*, *23*(2), pp 173–185. https://doi.org/10.1016/j.eij.2021.12.001

[16] Alatram, A., Sikos, L. F., Johnstone, M., Szewczyk, P., & Kang, J. J. (2023), DoS/DDoS-MQTT-IoT: A dataset for evaluating intrusions in IoT networks using the MQTT protocol, *Computer Networks,* 231, 109809, ISSN 1389-1286, https://doi.org/10.1016/j.comnet.2023.109809.