KARYA ILHAM PUBLISHING
CONNECTING MINDS, BRIDGING IDEAS, INSPIRING READERS

Journal of Advanced Research in Computing and Applications

Journal homepage:
https://karyailham.com.my/index.php/arca
2462-1927

WoodPecker Network-based Intrusion Detection System for Distributed Denial of Service (DDoS) Attacks using Rule-based Approach

Muhammad Ikhmal Ariff Mohd Rozi[1,*], Isredza Rahmi A Hamid[1], Noryusliza Abdullah[1], Wahidah Md Shah[2]

[1] Faculty of Computer Science and Information Technology, Universiti Tun Hussein Onn Malaysia, Johor, Malaysia
[2] Faculty of Information and Communication Technology, Universiti Teknikal Melaka, Melaka, Malaysia

ARTICLE INFO

Article history:
Received 27 November 2026
Received in revised form 20 February 2026
Accepted 15 April 2026
Available online 4 May 2026

Keywords:
Network-based intrusion detection systems (NIDS); Distributed Denial-of-Service (DDoS) Attacks; False Alarm Rate

ABSTRACT

Network-based Intrusion Detection Systems (NIDS) are intelligent systems that passively monitor network traffic to identify potential security threats, including intrusions and attacks on critical services such as Hypertext Transfer Protocol (HTTP), Server Message Block (SMB), and Secure Shell (SSH). This paper introduces WoodPecker, a NIDS designed to address inherent limitations in existing systems, with a specific focus on the common issue of high false alarm rates. WoodPecker aims to develop a rule-based NIDS capable of effectively detecting and preventing Distributed Denial-of-Service (DDoS) attacks and malicious activities. The system operates by examining incoming Internet packets and matching them with predefined rules in the WoodPecker setup. If DDoS attacks are identified in the packets, they are labelled as infected. The WoodPecker system is developed using the Python programming language, adopting a prototype model methodology. The outcome of this project is the development of an effective NIDS that can accurately detect DDoS attacks while minimizing false alarms. The percentage of passed test cases is 93% while the false error test case is 7%. The anticipated result of WoodPecker is to significantly enhance the effectiveness of network security measures.

1. Introduction

Network Intrusion Detection System (NIDS) is a crucial security software designed to monitor network activity and detect malicious behaviour or policy violations. By analysing real-time network traffic, NIDS can promptly notify administrators of suspicious activities, including unauthorized resources access attempts and the present of known malicious software. This additional layer of security helps organizations protect their networks and systems against cyber threats. NIDS operates by examining various data points from different sources across the network, such as packet headers, statistics, and protocol to identify any sign malicious or abnormal behaviour.

* *Corresponding author.*
E-mail address: ikhmalariff69@gmail.com

https://doi.org/10.37934/arca.43.1.5162

However, one major challenge in NIDS system is the is the frequent occurrence of false alarms [1]. The rate of false alarm rate is often higher than genuine threats, which allow actual intrusions or attack to go undetected. Several factors contribute to false alarms in NIDS, includes misconfiguration, where inaccurate settings can lead to false alarms [2-3]. Additionally, NIDS may generate alarm for normal activities, such as an authorized users accessing restricted resources.

To address this issue, NIDS has been proposed as a solution to reduce false alarm rate. This involves carefully cautiously deactivating attack signatures that are not harmful to the user's environment after extensive examination. The objective of this paper is to design a Network-based Intrusion Detection System using an object-oriented approach to detect DDoS attacks. Additionally, the goal is to develop NIDS for detecting DDoS attacks and evaluate it in terms of system functionality. The NIDS consists of five main modules: Intrusion Detection System (IDS) configuration, network traffic capture, IDS process core, IDS rule and IDS result module.

2. Related Work

This section explains the Intrusion Detection System (IDS), types of IDS, NIDS architecture, and existing intrusion detection systems.

2.1 Intrusion Detection System (IDS)

An Intrusion Detection System (IDS) is a system that designed to monitor network traffic and identify suspicious activity or intrusions. It serves as a security system that detects unauthorized attempts to break into networks. When properly configured, an IDS can monitor both incoming and outgoing network traffic, analyse network patterns continuously, and promptly raise an alarm when it detects suspicious intrusions or activities [4].

Most IDS systems utilize signature analysis systems and employ simple pattern matching techniques. They track network activity and examine it for specific types of planned attacks. IDSs are predominantly signature-based or knowledge-based, including model-based ID, state transition analysis, pattern monitoring, and ID expert systems [5]. Notably, many antiviruses software utilizes signature-based IDS.

2.2 Types of Intrusion Detection System (IDS)

There are two types of IDS that are Network-based Intrusion Detection System (NIDS) and Host-based Intrusion Detection System (HIDS).

1) Network-based Intrusion Detection System (NIDS): A network-based intrusion detection system (NIDS) is a security tool that monitors network traffic, examines signs of malicious activity or policy breaches, and alerts administrators to any abnormal behaviour. NIDS can be deployed to monitor various types of network traffic, including LANs, WANs, and the Internet. They can be installed as hardware or software. NIDS employs a variety of detection techniques, including as signature-based detection, anomaly-based detection, and behavioural-based detection, to find harmful activities. While anomaly-based detection looks for patterns that depart from what is seen as normal behaviour, signature-based detection searches for patterns in network data that match known harmful patterns. NIDS provides continuous monitoring network activity, offering real-time visibility into potential security threats. It can be customized to focus on specific types of network activity or threats. This makes NIDS more effective in identifying and alerting to potential security breaches. However, NIDS may generate false alarms, which occur when the system triggers an alert for normal

or intrusion activity. Factors contributing to false alarms include misconfiguration, normal network activity, hardware or software failures, and human error. Additionally, NIDS can only monitor network activity passing through the network interface where the system is installed, limiting its ability to detect intrusions on individual host computers or involving encrypted traffic.

2) Host-based Intrusion Detection System (HIDS): Host-based Intrusion Detection System (HIDS) are software that monitor specific computer systems for malicious activity. Unlike NIDS, which focusses on network traffic, HIDS examines the activity of individual hosts in detail and can identify processes and users engaged in malicious activity. HIDS can directly access and monitor data files and system processes targeted by attacks, enabling the detection of attempted attacks. It also can be employed on a firewall to enhance its security, and the ruleset can be customized to meet specific requirements, resulting in better efficiency, and reduced Central Processing Unit (CPU) overhead. HIDS offers detailed monitoring of individual hosts, allowing the identification of malicious processes and users [6]. HIDS can directly access and monitor targeted data files and system processes, facilitating the detection of attempted attacks. However, HIDS is limited to monitoring activity on a single host and may not detect attacks originating from other sources or targeting other hosts. It may require significant resources, including CPU and memory, which can impact performance [6].

2.3 Comparison of NIDS and HIDS

Table 1 presents the comparison between the NIDS and HIDS. Based on the advantages offered by each system, this project has decided to develop a NIDS named WoodPecker. The decision is driven by several key benefits provided by NIDS. Firstly, NIDS enables analysis through network activity, allowing for comprehensive monitoring of potential security threats. Secondly, NIDS offers real-time monitoring capabilities, ensuring prompt detection of suspicious activities. Additionally, NIDS can focus on specific types of attacks, enhancing its effectiveness in targeted threat detection. Lastly, NIDS can access and monitor network traffic throughout the entire network, providing broader coverage and visibility.

Table 1
Comparison between NIDS and HIDS

	NIDS	HIDS
Analysis	Network activity.	Single host activity.
Monitoring	Provide real-time monitoring.	No real-time monitoring.
Customization	Can focus on specific types of attacks.	Cannot focus on specific types of attacks.
Access	Throughout network.	Throughout monitor data files.

2.4 Study of Existing NIDS

This section discusses about the existing Network-based Intrusion Detection System (NIDS) system that are Snort, Suricata and Zeek. Snort is a widely used free and open-source Intrusion Detection System (IDS) with millions of users. It serves as a network-centric solution, effectively identifying system interruptions by analysing network traffic. Operating in various modes such as Sniffer mode, NIDS mode, and Packet logger mode, Snort offers versatile functionality. It relies on a "known bad" or "suspected bad" approach, continuously monitoring network activity to detect patterns associated with harmful actions [7]. The system captures and analyses data, storing it in a MySQL database [8]. Notable features include timeliness, high detection rate, low false alarms, specificity, scalability, and ease of use via command line. The NIDS mode, using rules to detect

intrusion activity, is considered the most crucial for intrusion detection. Overall, Snort is a robust and multifaceted tool that provides valuable insights and safeguards networks against potential threats.

Suricata is a high-performance, multithreaded network Intrusion Detection System (IDS) designed to analyse traffic within a specific network. It functions as an IDS, monitoring LAN traffic for suspicious or malicious activity against network hosts. Additionally, Suricata acts as an Intrusion Prevention System (IPS) and a Network Monitoring Engine (NSM) to prevent potentially harmful network activity. The IPS functionality is achieved through an inline mode IPS device, which employs detection technologies such as signature-based detection for known patterns, statistical anomaly-based detection for deviations from normal behaviour, and stateful protocol analysis to analyse network flow [9]. The system's objective is to demonstrate how attackers may bypass security measures, prompting immediate reaction through the activation of Suricata's inline mode. This involves routing incoming and outgoing traffic through a queue for analysis using tools like IPTables and Netfilter in GNU/Linux systems. Precise rules are set to direct traffic to a special queue, NFQueue, ensuring Suricata analyses queued packets for drop and accept functions.

Zeek is a Network Intrusion Detection System (NIDS) tool focused solely on IDS mode. In its architecture, workers act as agents on network devices, managing logs for the Zeek Manager. The Zeek Manager comprises an event engine converting network packets into events and a policy script interpreter applying Zeek rules to detect malicious behaviour. Zeek's scalable design allows performance enhancement by allocating additional hardware resources to workers and management. Notably, Zeek stands out for its anomaly-based detection capability, absent in Snort and Suricata, making it prominent in zero-day assault scenarios [9]. Despite having a limited set of pre-set signatures, Zeek provides a comprehensive analysis of network traffic, offering a powerful, versatile, and open-source solution. However, its rule base is smaller compared to Snort and Suricata. Zeek's advantages include its powerful security language, effective threat hunting through behavioural analysis, and the ability to analyse encrypted traffic using indicators like JA3 fingerprint and TLS ciphers, addressing privacy concerns associated with the rise in encrypted traffic volume [9].

3. Methodology

Figure 1 presents six phases of the Prototype Model that are Planning phase, Analysis phase, Design phase, Implementation phase, Prototype phase and Testing phase [10]. The Prototype model is selected due to its ability to effectively reduce the number of iterations required in the System Development Life Cycle (SDLC). This results in time savings and increases the probability of user satisfaction.

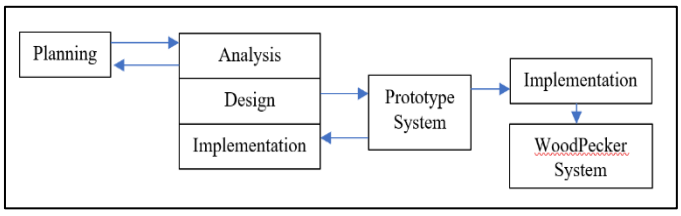

Fig. 1. Prototype Model [10]

3.1 System Development Phase Activities

There are five phases involved in the prototype model. Table 2 shows the system development activities and the deliverables for each phase.

Table 2
System development activities

Phase	Activities	Deliverables
Planning	Work scheduling, problem identification, scope, and objective.	Gantt chart and proposal.
Analysis	Collect and analyse information.	System Requirements and software hardware requirement.
Design	Design user architecture with the suitable programming language.	Architecture, system flow and process of NIDS.
Implementation	Carry out the testing system and fix the errors.	System program code.
Prototype 1	Identify the problems that exist in the system and repair the existing system. Repetition of the planning phase until the implementation phase.	System prototype.
Prototype 2	Problem identification and repair the developed system.	System prototype.

The prototyping model is an iterative software development approach that involves creating a working prototype of the system before full development. It begins with gathering initial requirements, followed by quick design and construction of a basic prototype.

Stakeholders evaluate the prototype, providing feedback for iterative refinement. Once stakeholders are satisfied, the final system is developed using the refined prototype as a basis. The prototyping model enables early user involvement, better understanding of requirements, and risk mitigation. However, it may compromise scalability and performance if not managed carefully. Overall, it is a flexible and collaborative approach that facilitates faster feedback cycles and tailored solutions.

4. System Analysis and Design

This section explains the analysis and design of the WoodPecker Network-based Intrusion Detection System (NIDS) that have been proposed in this project.

4.1 System Architecture

Figure 2 shows the system architecture of the proposed NIDS called WoodPecker. WoodPecker is designed to monitor both incoming and outgoing internet traffic to detect potential threats. To ensure comprehensive visibility, NIDS sensors are strategically positioned throughout the network, including on Local Area Network (LAN) and Demilitarized Zones (DMZs).

In signature-based detection, the characteristics of data packets are compared against a database of known fraudulent signatures. This allows the NIDS to identify and flag any malicious or suspicious behavior observed on the network. For instance, the system can detect rapid spikes in network traffic and generate warnings that prompt further investigation.

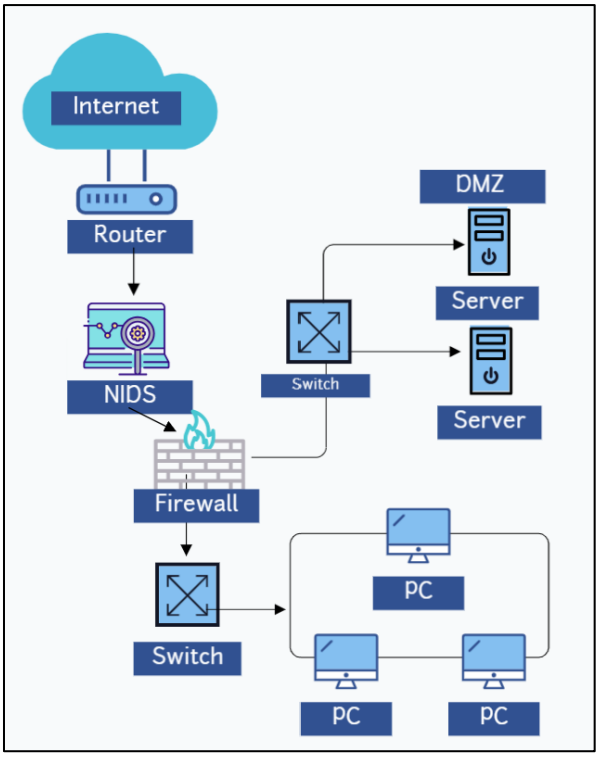

Fig. 2. Architecture of WoodPecker

4.2 Use Case Diagram

The use-case diagram is used in this project to define the functionalities and the access right that can be performed and carried out by the user. Figure 3 shows the use case diagram of the user regarding in the WoodPecker, which is receiving of incoming packets, examining of received packets, details report of infected packets, and authentication of IP address.

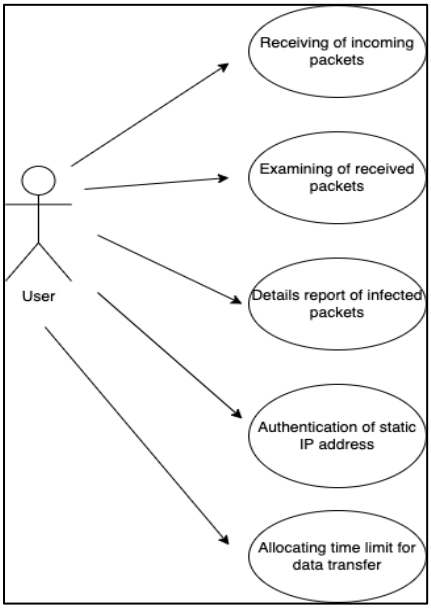

Fig. 3. Use case diagram

There are five modules; receiving of incoming packets, examining of received packets, details report of infected packets, authentication of static Internet protocol (IP) address, and allocating time

limit for data transfer which can be accessed by user in the WoodPecker. The process begins with the user receiving incoming packets from the internet. These packets are then scrutinized using NIDS software, which matches them with the rules that have been set up in WoodPecker. If the packets are found to contain viruses listed in the database, they are identified as infected packets. Additionally, users have the option to authenticate static IP addresses by adding them to the authenticated IP addresses database.

5. Result and Discussion

In this section, the screen capture of the system and code segment and the result of the WoodPecker system will be shown. The implemented of WoodPecker interface, WoodPecker main function and the result of the WoodPecker functionality test will be discussed.

5.1 WoodPecker Rules

Figure 4 shows the set of rules in WoodPecker system. These rules are written in Snort rule syntax, which is used in intrusion detection and prevention systems. Each rule defines a specific condition or pattern to be matched against network traffic.

These rules demonstrate different scenarios and conditions for detecting specific network activities, such as DNS requests, NTP traffic, port scans, and protocol-specific detections like HTTP and SSH. Each rule is designed to trigger an alert or action when the specified condition is met during network traffic analysis.

The provided rules are written in Snort IDS rule syntax [11]. Each rule is designed to detect specific types of network traffic based on certain conditions. Table 3 shows the breakdown of the written rules.

```
alert udp any any -> 8.8.8.8 53 (msg:"DNS to google")
alert udp any any -> 8.8.8.4 53 (msg:"DNS to google")
alert udp any any -> any 123 (msg:"NTP")
alert tcp any any -> any 123 (msg:"NTP")
alert tcp 192.168.0.1 any -> 192.168.0.0/16 any (msg:"Gateway to subnet")
alert tcp any any -> 192.168.1.102 any (ack: 0;flags: SYN;msg: TCP Port Scan)
alert tcp any any -> any any (msg:"Naive";)
alert udp any any -> any any (msg:"Naive";)
alert tcp any any -> any any (msg:"Naive";)
alert tcp any any -> any any (msg:"aggressive";)
alert udp any any -> any any (msg:"aggressive";)
alert tcp any any -> any any (msg:"aggressive";)
alert tcp 127.0.0.1 any -> 192.168.1.0 any (msg:"r1 test for subnet 192.168.1.0";)
alert tcp 127.0.0.1 any -> any 23,25,21 (msg:"r2 tcp traffic from any port going to
Telnet, FTP, and SSH severs ports";)
alert udp 127.0.0.1 any -> any 10000:20000 (msg:"r3 udp traffic from any port and
destination ports ranging from 10000 to 20000";)
alert udp 127.0.0.1 any -> any any (tos:33;msg:"r4 tos is 33";)
alert tcp 127.0.0.1 any -> any any (len:24;msg:"r5 len is 24";)
alert tcp 127.0.0.1 any -> any any (offset:111;msg:"r6 offset is 111";)
alert tcp 127.0.0.1 any -> any any (seq:5;msg:"r7 tcp seq is 5";)
alert tcp 127.0.0.1 any -> any any (ack:6;msg:"r8 tcp ack is 6";)
alert tcp 127.0.0.1 any -> any any (flags:SP;msg:"r9 SYN Packet";)
alert http 127.0.0.1 any -> any 80 (http_request:"GET";msg:"r10 GET!";)
alert tcp 127.0.0.1 any -> any 22 (content:"/bin/sh";msg:"r11 Remote shell
execution!";)
alert tcp 127.0.0.1 any -> any any (tos:123;seq:15;ack:16;msg:"r12";)
```

Fig. 4. List rules in WoodPecker

Table 3
Description of WoodPecker rules

No	Rules	Rules Description	Alert
1.	alert udp any any -> 8.8.8.8 53 (msg:"DNS to google")	Alert UDP traffic from any source and any port to destination IP 8.8.8.8 on port 53. Message: "DNS to google."	E3w[pDNS to google
2.	alert udp any any -> 8.8.8.4 53 (msg:"DNS to google")	Alert UDP traffic from any source and any port to destination IP 8.8.8.4 on port 53. Message: "DNS to google."	DNS to google
3.	alert udp any any -> any 123 (msg:"NTP")	Alert UDP traffic from any source and any port to any destination on port 123. Message: "NTP."	NTP
4.	alert tcp any any -> any 123 (msg:"NTP")	Alert TCP traffic from any source and any port to any destination on port 123. Message: "NTP."	NTP
5.	alert tcp 192.168.0.1 any -> 192.168.0.0/16 any (msg:"Gateway to subnet")	Alert TCP traffic from source IP 192.168.0.1 and any port to any destination within the 192.168.0.0/16 subnet on any port. Message: "Gateway to subnet."	Gateway to subnet
6.	alert tcp any any -> 192.168.1.102 any (ack: 0;flags: SYN;msg: TCP Port Scan)	Alert TCP traffic from any source and any port to destination IP 192.168.1.102 on any port, with acknowledgment flag set to 0 and SYN flag set. Message: "TCP Port Scan."	TCP Port Scan
7.	alert tcp any any -> any any (msg:"Naive";)	Alert TCP traffic from any source and any port to any destination on any port. Message: "Naive."	Naïve
8.	alert udp any any -> any any (msg:"Naive";)	Alert UDP traffic from any source and any port to any destination on any port. Message: "Naive."	Naïve
9.	alert tcp any any -> any any (msg:"Naive";)	Alert TCP traffic from any source and any port to any destination on any port. Message: "Naive."	Naïve
10.	alert tcp any any -> any any (msg:"aggressive";)	Alert TCP traffic from any source and any port to any destination on any port. Message: "Aggressive."	Aggressive
11.	alert udp any any -> any any (msg:"aggressive";)	Alert UDP traffic from any source and any port to any destination on any port. Message: "Aggressive."	Aggressive
12.	alert tcp any any -> any any (msg:"aggressive";)	Alert TCP traffic from any source and any port to any destination on any port. Message: "Aggressive."	Aggressive
13.	alert tcp 127.0.0.1 any -> 192.168.1.0 any (msg:"r1 test for subnet 192.168.1.0";)	Alert TCP traffic from source IP 127.0.0.1 and any port to any destination within the 192.168.1.0 subnet on any port. Message: "r1 test for subnet 192.168.1.0."	r1 test for subnet 192.168.1.0
14.	alert tcp 127.0.0.1 any -> any 23,25,21 (msg:"r2 tcp traffic from any port going to Telnet, FTP, and SSH severs ports";)	Alert TCP traffic from source IP 127.0.0.1 and any port to any destination on ports 23, 25, and 21. Message: "r2 tcp traffic from any port going to Telnet, FTP, and SSH server ports."	r2 tcp traffic from any port going to Telnet, FTP, and SSH server ports.
15.	alert udp 127.0.0.1 any -> any 10000:20000 (msg:"r3 udp traffic from any port and	Alert UDP traffic from source IP 127.0.0.1 and any port to any	r3 udp traffic from any port and

	destination ports ranging from 10000 to 20000";)	destination on ports ranging from 10000 to 20000. Message: "r3 udp traffic from any port and destination ports ranging from 10000 to 20000."	destination ports ranging from 10000 to 20000.
16.	alert udp 127.0.0.1 any -> any any (tos:33;msg:"r4 tos is 33";)	Alert UDP traffic from source IP 127.0.0.1 and any port to any destination on any port, with TOS (Type of Service) value set to 33. Message: "r4 tos is 33."	r4 tos is 33.
17.	alert tcp 127.0.0.1 any -> any any (len:24;msg:"r5 len is 24";)	Alert TCP traffic from source IP 127.0.0.1 and any port to any destination on any port, with a packet length of 24. Message: "r5 len is 24."	r5 len is 24.
18.	alert tcp 127.0.0.1 any -> any any (offset:111;msg:"r6 offset is 111";)	Alert TCP traffic from source IP 127.0.0.1 and any port to any destination on any port, with an offset value of 111. Message: "r6 offset is 111."	r6 offset is 111
19.	alert tcp 127.0.0.1 any -> any any (seq:5;msg:"r7 tcp seq is 5";)	Alert TCP traffic from source IP 127.0.0.1 and any port to any destination on any port, with a sequence number of 5. Message: "r7 tcp seq is 5."	r7 tcp seq is 5.
20.	alert tcp 127.0.0.1 and -> any any (ack:6;msg:"r8 tcp ack is 6";)	Alert TCP traffic from source IP 127.0.0.1 and any port to any destination on any port, with an acknowledgment number of 6. Message: "r8 tcp ack is 6."	r8 tcp ack is 6.
21.	alert tcp 127.0.0.1 any -> any any (flags:SP;msg:"r9 SYN Packet";)	Alert TCP traffic from source IP 127.0.0.1 and any port to any destination on any port, with SYN and PSH flags set. Message: "r9 SYN Packet."	r9 SYN Packet.
22.	alert http 127.0.0.1 any -> any 80 (http_request:"GET";msg:"r10 GET!";)	Alert HTTP traffic from source IP 127.0.0.1 and any port to any destination on port 80, with an HTTP request method of "GET." Message: "r10 GET!"	r10 GET!
23.	alert tcp 127.0.0.1 any -> any 22 (content:"/bin/sh";msg:"r11 Remote shell execution!";)	Alert TCP traffic from source IP 127.0.0.1 and any port to any destination on port 22, with the content "/bin/sh" in the packet payload. Message: "r11 Remote shell execution!"	r11 Remote shell execution!
24.	alert tcp 127.0.0.1 any -> any any (tos:123;seq:15;ack:16;msg:"r12";)	Alert TCP traffic from source IP 127.0.0.1 and any port to any destination on any port, with TOS value of 123, sequence number of 15, and acknowledgment number of 16. Message: "r12.”	r12.

5.2 Test Case Plan

The test case plan will be carried out to determine the result of WoodPecker. The test case plan shows the list of test plan and the result of the testing. The expected result of the test plan is also

included in Table 4 as standard test to the WoodPecker system. The test case plan is shown in Table 4.

Table 4
Test Case plan

No.	Test Cases	Description	Result	Expected Result
TEST 100				
1.	TEST_100_001	System should be able to connect to the local network.	Pass	Able to connect to local network
2.	TEST_100_002	System should be able to read the rules file from the rules.txt files.	Pass	Able to read the rules.txt file
3.	TEST_100_003	System should be able to scan and sniff the network based on the rules that have been set.	Pass	Able to scan and sniff the network
4.	TEST_100_004	System should be able to list out the alert message based on the rules that have been matched.	Pass	Able to list out the alert message
5.	TEST_100_005	System should be able to view details of the network traffic.	Pass	Able to view details of network traffic
6.	TEST_100_006	System should be able to view network traffic activity.	Fail	Able to view network traffic activity
TEST_200				
1.	TEST_200_001	Scanning button in WoodPecker should be able to function well	Pass	Scanning button should function well
2.	TEST_200_002	Log button in WoodPecker should be able to function well	Pass	Log button should function well
3.	TEST_200_003	Rules button in WoodPecker should be able to function well	Pass	Rules button should function well
4.	TEST_200_004	User manual button in WoodPecker should be able to function well	Pass	User manual button should function well
5.	TEST_200_005	Quit button in WoodPecker should be able to function well	Pass	Quit button should function well
TEST 300				
1.	TEST_300_001	System should be able to view the WoodPecker detection rules	Pass	Able to view the rules
2.	TEST_300_002	System should be able to edit the WoodPecker detection rules	Pass	Able to edit the rules
3.	TEST_300_003	System should be able to add the WoodPecker detection rules	Pass	Able to add the rules
4.	TEST_300_004	System should be able to delete the WoodPecker detection rules	Pass	Able to delete the rules

There is a total of 15 test cases used to test the implementation of WoodPecker system. The WoodPecker system passes 14 out of 15 test cases which is equivalent to 93 percent of the test cases. Table 5 shows the overall test result of WoodPecker system.

Table 5.
Overall Test Case result WoodPecker

ID	Total Test Case	Total Passed Test Case
TEST_100	6	5
TEST_200	5	5
TEST 300	4	4
Overall Total	15	14

Figure 5 shows the summary of overall test result in the form of pie chart. The percentage of passed test cases is 93% while the false error test case is 7%.

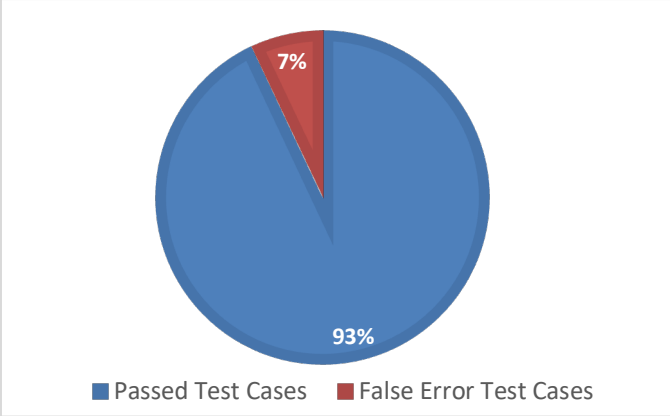

Fig. 5. Summary Test Case result WoodPecker

5.3 Comparison Existing System with the Proposed System

Table 6 shows the comparison of existing systems that are Snort, Suricata and Zeek with the proposed system, WoodPecker. All systems provide real-time traffic monitoring except Suricata. The similarity of the Snort and Suricata with WoodPecker is all systems have packet logging and analysis protocol function to detect network intrusion. However, Snort and Suricata did not provide create logs functions. Additionally, Zeek and WoodPecker provide real-time traffic monitoring and create logs functions. Yet, Zeek did not provide packet logging and analysis protocol function as other system did. Therefore, our proposed WoodPecker system differ that others in such a way that we provide more functions that can complement other systems.

Table 6
Comparison existing system with WoodPecker

	Existing System			
Features	Snort	Suricata	Zeek	Woodpecker
Real-time traffic monitor	√	X	√	√
Packet Logging	√	√	X	√
Analysis Protocol	√	√	X	√
Create logs	X	X	√	√
False error rate	92%	23%	21%	7%

The false error rate in the test case plan for the login functionality of system. The error stemmed from an incorrect expected result, leading to misleading outcomes during testing. The system emphasizes the importance of accurate test case planning and discusses the implications of false errors on the overall testing process. By recognizing and learning from this false error, system testing practices can be improved to ensure more reliable and efficient testing procedures.

6. Conclusions

The WoodPecker system have successfully achieved all objectives and requirements. The main objective was to create a system capable of capturing network packets, analyzing them to detect DDoS attacks, and generating automated alerts. The system also provides various benefits to the users, including reducing the need for 24-hour network traffic monitoring, being cost-effective compared to other intrusion detection systems (IDS), offering detailed reports for DDoS attack detection, and being compatible with multiple platforms such as MacOS, Windows, and Linux.

However, the WoodPecker system does have certain limitations. Firstly, the detection rules can only be changed and updated by the developers, which restricts flexibility for users. Secondly, the system can only scan one network packet at a time, limiting their scanning capabilities. Lastly, the system lacks advanced analysis reporting features such as graph reports or pattern reports, which could provide more in-depth insights.

In terms of future work, there are several areas where the WoodPecker system can be improved. Firstly, the rule setting functionality can be enhanced by allowing users to change and update detection rules using the frontend interface. Additionally, adding alarms to the DDoS attack alerts would improve their effectiveness. Lastly, enhancing the performance of the system to monitor multiple networks simultaneously would make them more adept at detecting DDoS attacks.

In conclusion, the WoodPecker system have successfully accomplished their objectives, providing benefits such as cost and time savings for network traffic monitoring. While there are limitations and areas for improvement, future work can address these issues and enhance the system functionalities. Overall, the WoodPecker system have fulfilled the requirements and hold potential for assisting users in effectively monitoring and detecting DDoS attacks on their networks.

Acknowledgement
This research was supported by Universiti Tun Hussein Onn Malaysia (UTHM) through TIER 1 (Vot. Q167).

References

[1] Fernandes, Chrystinne, Simon Miles, and Carlos José Pereira Lucena. "Detecting false alarms by analyzing alarm-context information: Algorithm development and validation." *JMIR Medical Informatics* 8, no. 5 (2020): e15407. https://doi.org/10.2196/15407

[2] Tran, Tung. "Misconfiguration analysis of network access control policies." Master's thesis, University of Waterloo, 2009.

[3] Alicea, Michael, and Izzat Alsmadi. "Misconfiguration in Firewalls and Network Access Controls: Literature Review." *Future Internet* 13, no. 11 (2021): 283. https://doi.org/10.3390/fi13110283

[4] Mihret, Estifanos Tilahun. "Intrusion Detection System-IDS." *American Journal of Computer Science and Information Technology* 9, no. 8 (2021): 1-5.

[5] Tom Chen and Patrick J. Walsh. "Chapter 3 - Guarding Against Network Intrusions." In *Network and System Security (Second Edition)*, pp. 57–82. Elsevier, 2014. https://doi.org/10.1016/b978-0-12-416689-9.00003-4

[6] Azad, Tariq Bin. "Chapter 7 - Locking Down Your XenApp Server." In *Securing Citrix Presentation Server in the Enterprise*, pp. 487-555. Elsevier, 2008. https://doi.org/10.1016/B978-1-59749-281-2.00007-X

[7] Bada, G., W. Nabare, and D. Quansah. "Comparative Analysis of the Performance of Network Intrusion Detection Systems: Snort Suricata and Bro Intrusion Detection Systems in Perspective." *International Journal of Computer Applications* 176, no. 40 (2020): 39-44. https://doi.org/10.5120/ijca2020920513

[8] Zhou, Zhimin, Zhongwen Chen, Tiecheng Zhou, and Xiaohui Guan. "The study on network intrusion detection system of Snort." In *2010 international conference on networking and digital society*, vol. 2, pp. 194-196. IEEE, 2010. https://doi.org/10.1109/ICNDS.2010.5479341

[9] Waleed, Abdul, Abdul Fareed Jamali, and Ammar Masood. "Which open-source ids? snort, suricata or zeek." *Computer Networks* 213 (2022): 109116. https://doi.org/10.1016/j.comnet.2022.109116

[10] Oppenheimer, Daniel M., Joshua B. Tenenbaum, and Tevye R. Krynski. "Categorization as causal explanation: Discounting and augmenting in a Bayesian framework." In *Psychology of Learning and Motivation*, vol. 58, pp. 203-231. Academic Press, 2013. https://doi.org/10.1016/B978-0-12-407237-4.00006-2

[11] Stakhanova, Natalia, and Ali A. Ghorbani. "Managing intrusion detection rule sets." In *Proceedings of the Third European Workshop on System Security*, pp. 29-35. 2010. https://doi.org/10.1145/1752046.1752051