Network Intrusion Detection System Using Machine Learning & Scapy

(Shoaib Raza, Lecturer in Department of Cyber Security,

Foundation for Advancement of Science and Technology (FAST) – National University)

Author Name/s Muhammad Hussain
Department of **Cyber Security**
Foundation for Advancement of Science and Technology (FAST) – National University.
Karachi, Pakistan
k213584@nu.edu.pk

Author Name/s Vania Abbas
Department of **Cyber Security**
Foundation for Advancement of Science and Technology (FAST) – National University.
Karachi, Pakistan
K214753@nu.edu.pk

***Abstract*—This report presents a novel approach to developing Network Intrusion Detection Systems (NIDS) using Machine Learning (ML). Traditional signature-based NIDS heavily rely on pre-defined patterns or signatures of known attacks, limiting their ability to detect zero-day attacks, resulting in a high number of false positives and an inability to detect polymorphic attacks [1]. Machine learning algorithms enhance traditional NIDS by improving detection accuracy, reducing false positives, and adapting to evolving threats. Unlike signature-based approaches, ML-based NIDS can detect both known and unknown threats, including zero-day attacks, by analyzing network traffic patterns and behavior. They continuously learn from new data, enabling them to effectively combat sophisticated and evolving threats without requiring frequent manual updates. Additionally, their scalability enables efficient processing of large volumes of network data in real-time, making them suitable for deployment in high-speed and high-traffic networks [2]. To create an ML-based NIDS, we first generated a labeled dataset comprising Denial of Service (DoS) attacks and NMAP scans. We utilized the Python network manipulation library called Scapy to sniff live packets and extract relevant features. Then, we employed Decision Trees, a classification algorithm, to train our model based on the generated dataset. Furthermore, we integrated signature-based detection using YARA rules and implemented a rule list for our NIDS system which allows it to analyze packets or not.**

Keywords-- *Network Intrusion Detection Systems (NIDS), Machine Learning, Signature-based Detection, Zero-day Attacks, False Positives, Decision Trees, Scapy, YARA Rules, Access Control List* ***(****key words****)***

I. Introduction

Network Intrusion Detection Systems (NIDS) play a crucial role in safeguarding network infrastructures against cyber threats by monitoring and analyzing network traffic for suspicious activities.

Traditional signature-based NIDS rely on pre-defined patterns or signatures of known attacks, which are limited in detecting zero-day exploits and often result in high false positive rates. The evolving threat landscape necessitates a more adaptive and effective approach to intrusion detection. This report presents a novel methodology for developing NIDS using Machine Learning (ML) algorithms, aiming to enhance detection accuracy, reduce false positives, and improve resilience against evolving threats.

The study encompasses the generation of a labeled dataset, feature extraction from network traffic, model training using ML algorithms, and integration of signature-based detection mechanisms. By combining the strengths of ML and signature-based detection, the proposed NIDS seeks to provide comprehensive and proactive defense against a wide range of cyber threats.

We are using a supervised machine learning algorithm called Decision Tree which works by forming decision nodes from the root node where the decision node is responsible for making decisions based on the input data it receives, each decision node has a leaf node which is the final prediction that the decision tree has made [3].

Since machine learning is a more compute-intensive process requiring high amount GPU, CPU, and memory resources, the system we generated is only limited to detecting ‘benign’, ‘DoS’, and ‘NMAP’ scans. We have integrated signature-based detection using YARA rules [4] which is a pattern matching framework aimed at malware analysis but can also be used for detection of anomalies in packets. The signature-based detection that we have implemented specifically targets HTTP payload received based on user input on our web application.

This project report advocates for the utilization of the Python Scapy library as a versatile toolkit for the implementation of Network Intrusion Detection Systems (NIDS) [5]. Scapy offers numerous advantages, particularly in its ease of use when working with network packets. Its flexibility and customization features allow users to tailor their NIDS implementation to specific project requirements. With comprehensive protocol support, Scapy facilitates the analysis of diverse network traffic, making it easier to detect various types of intrusions. Furthermore, Scapy's real-time packet sniffing capabilities simplify the capture and analysis of live network traffic, enabling users to develop effective real-time intrusion detection mechanisms effortlessly. Through seamless integration with machine learning frameworks, Scapy empowers users to combine packet-level analysis with advanced machine learning techniques, thereby enhancing the accuracy and effectiveness of their NIDS. Additionally, Scapy benefits from an active community of users and developers, providing valuable resources and support for users working with network packets. Overall, the Python Scapy library offers a practical and user-friendly approach to network security, facilitating packet-level analysis and intrusion detection.

Further improvements may be made to this project by writing a similar code in C++ programming language as it is 25 times faster than Python and using a library such as *libtins* which is faster as compared to Scapy and has very little overhead as compared to the *libpcap* library used by Scapy.

II. Methodology

We started designing our NIDS using the same approach that traditional NIDS systems utilize. A packet sniffing module which starts live packet capture on the interface that we provide. A filtration module which filters packets based on the destination IP address, if the destination IP address is the host computer where the NIDS is running on then the packet is sent forward to the rule parser module, else the packet is ignored. Then we have the rule parser module which checks if the packet is in the allow list or not, if it is in the allow list then the system check if the packet has HTTP payload in it and if the HTTP payload is present then the packet is sent to YARA rule matcher module. The packet is then forwarded to the feature extraction module as well which extracts the relevant features required by our decision tree model. Table 1 shows the relevant features that is required by our model. Once we receive the prediction from our trained model we check if the packet is benign or not, if it's not benign then the prediction is logged and an alert is shown on the NIDS's graphical user interface (GUI).

The program was in Python 3.11, even though the latest version of Python at the time of writing is Python 3.12.3, we had to use Python 3.11 because some of the modules that our Python program was using were not compatible with Python 3.12.3. The Python program utilizes the sniffing function of Scapy to capture live packets on the provided network interface card (NIC) where it is then sent for feature extraction. The sniffing process is done on a separate thread so that it does not interfere with the feature extraction and model prediction process since sniffing is a I/O blocking process which will pause the flow of execution until we sent process kill signal to the program. For machine learning we used Scikit Learn framework do train our model using the Decision Tree algorithm. Figure 1 illustrates the process of our NIDS system on detecting malicious packets.

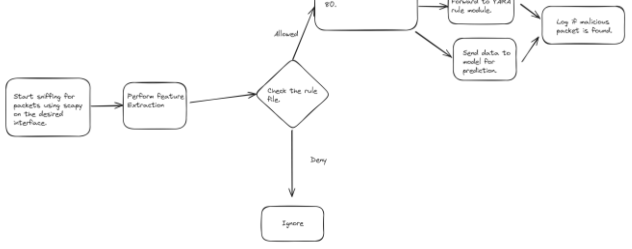

Start sniffing for packets using scapy on the desired interface.
Perform Feature Extraction
Check the rule file.
Allowed
Deny
Ignore
Packet has TCP and Port 80.
Forward to YARA rule module.
Send data to model for prediction.
Log if malicious packet is found.

Figure 1. Illustrates the process of our NIDS using Machine Learning.

Feature Names	**Description**
Dport	The destination port.
Sport	The source port.
Protocol	The protocol used in the packet which usually either TCP or UDP.
Flags	TCP flags. 0 if packet uses UDP protocol and 9 if there are no flags in the packet.
Time_bw_prev_packets	Time delta between the two packets.
Spkts	Size of packets coming from source in bytes.
Dpkts	Size of packet sent by host to source in bytes.
Pkt_len	Length of packet signified by the LEN field in a packet.
Ttl	Time to live of a packet.
Payload_size	The size of a payload carried by a packet.

Table 1. The table shows a list of features used by our model for prediction of malicious packets.

A. *YARA Module*

The YARA module of our program is responsible for identifying recognized patterns in the HTTP payload which is usually sent from a user using a POST request. The yara rules, which have the .yar file extension, are stored in the yara-rules directory. Upon starting the program all the yara rules are loaded into the program and if a packet with HTTP payload is detected it is then a new thread is created in python which handles the yara rule matching. The payload is extracted and the new thread that we created sends the payload to the yara rule matching function. Once the yara rule matching finds a signature match it logs the result and an alert is also shown on the NIDS's GUI.

B. *Feature Extraction and Model Prediction*

The feature extraction modules extract the relevant features, as mentioned in Table 1, and converts the list of features to a Pandas Dataframe and sends it to the model prediction module which uses the Decision Tree classifier algorithm. The model prediction has three different types of predictions:

1. Benign: This prediction is done when the packet received by the system is considered "harmless" and is not logged or marked as an alert.
2. DoS: This prediction is done when the packet is considered a Denial of Service attack.
3. NMAP: This prediction is done when a packet is considered to from a machine performing an NMAP scan. This is usually identified from the TCP flags or the packet payload sent to the victim machine.

C. *Generation of Datasets*

There were multiple datasets available for Network Intrusion Detection Systems (NIDS) that we could have chosen from to train our model. However, we opted to create our own dataset for several reasons. The dataset we created had fewer features compared to datasets such as those provided by Amazon or the KDD Cup 1999 dataset, which contain approximately 70 features and around 140 million lines of data in comma-separated format. These large datasets require significant memory resources, which we did not have access to. Additionally, the time required to extract relevant features from these datasets exceeded the project's timeframe. Moreover, our system lacked sufficient processing power to handle training with these datasets; attempting to train our model using the datasets from Amazon or KDD Cup 1999 resulted in our Python kernel crashing due to memory errors.

We chose to utilize Scapy and Pandas Dataframe for dataset creation. Scapy enabled us to limit the number of packets on which we performed feature extraction, while Pandas Dataframe facilitated data structuring and cleaning. Additionally, we performed one-hot encoding of the TCP flags collected, with each flag being replaced by a numerical value. In cases where multiple flags were present in a TCP packet, the numerical values were concatenated or joined. Table II provides a summary of the flags and their corresponding numerical representations.

During the data generation process, our Python script prompted for the dataset's label, filename, and the number of packets to be captured by Scapy while the program was running. Upon sniffing each packet, it was passed to the feature extraction module, which extracted the features outlined in Table I and appended them to a list of Python dictionaries. Once the packet sniffing process concluded, the script finalized the dataset creation by converting this list of dictionaries into a Pandas Dataframe, which was then saved as a Comma-Separated Values (CSV) file.

We first started by collecting benign packet data which involved in us downloading files from the internet, performing web application authentication, simulating failed login attempts, logging into SSH and FTP services, performing Linux and Windows software updates, browsing the internet, and streaming videos on popular streaming sites on various qualities. After collecting benign network data was done we proceeded with collecting the data for DoS attacks. We used Kali Linux 2024.1 which was running on Vmware Workstation Pro 17 on a bridged network. Both the host and guest machines were running on an ethernet network connection. We used the built in network scanning and DoS tool called Hping3. We used this tool to target popular ports where DoS attacks happen which are:

- 53, UDP – DNS
- 80 TCP – HTTP
- 8080 TPC – HTTP
- 443 TCP – HTTPS
- 8443 TCP – HTTPS

DoS attacks was performed on all these ports on regular intervals and their packet features were collected and stored in a CSV format. For detection of network scanning we utilized NMAP tool in Kali Linux which can perform network scans using various methods based on the provided flags to the NMAP command. Following scans were performed:

- TCP SYN scan
- TCP NULL scan
- TCP XMAS scan
- TCP UDP Scan
- NMAP Scan with scan intensity 4
- NMAP Vulnerability Scan
- NMAP OS fingerprinting scan

Same as the DoS attack data generation, we performed NMAP scans on regular intervals and collected ~200,000 packets of NMAP scans.

D. Model Training and Testing

Decision Trees have emerged as a popular choice for Network Intrusion Detection Systems (NIDS) due to their interpretability, feature importance ranking, scalability, and adaptability. Their transparent decision-making process facilitates understanding and interpretation by security analysts, while their ability to rank features aids in prioritizing relevant attributes for intrusion detection. Decision Trees offer computational efficiency and scalability, enabling timely analysis of large volumes of network traffic data. Additionally, their adaptability allows for dynamic adjustments to evolving threats and changes in the network environment. Research and practical implementations have demonstrated the effectiveness of Decision Trees in NIDS, making them a valuable tool for enhancing network security. Figure 2 explains how decision trees will work for our NIDS system.

For training our model we used Decision Tree, and we used the Python library called Scikit Learn as it comes with the Decision Trees classification algorithm and other functions which cleans and splits our data into training and test sets.

Leveraging scikit-learn's implementation of Decision Trees, security analysts can efficiently develop NIDS models with robust features such as interpretability, feature importance ranking, and scalability. scikit-learn's user-friendly interface and extensive documentation streamline the model development process, enabling rapid experimentation and evaluation of NIDS architectures. Furthermore, scikit-learn offers seamless integration with other machine learning algorithms and tools, facilitating the construction of ensemble models and hybrid NIDS architectures. Through its comprehensive functionality and ease of use, scikit-learn empowers security practitioners to deploy effective and adaptable NIDS solutions in diverse network environments.

There are a total of 10 features that we extracted from our packets. Figure 2 shows a histogram plot of our dataset.

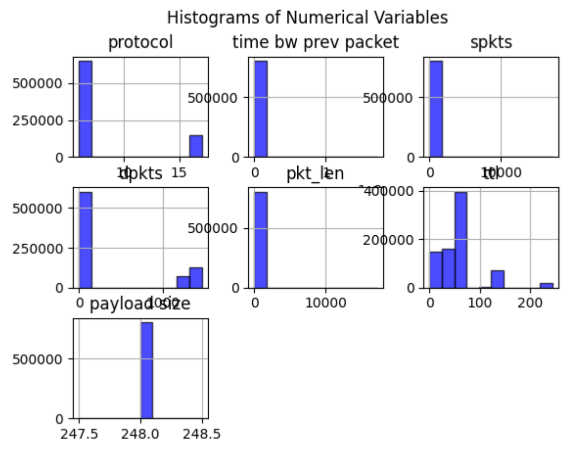


Figure 2. Histogram of the generated dataset

III. Testing On A Live System

A. *System Specifications*

The following NIDS system requires Python 3.11 to be installed on the system with the required modules present in the requirements.txt file. Python's package manager, pip, should be used to install the required modules. Moreover, a python virtual environment is highly recommended when installing the modules and running the program as it prevents any collision between system wide installed packages.

The program was run on the following system:

- AMD Ryzen 5 3600 @ 3.6 GHz
- 16 GB DDR4 3200 MHz RAM
- NVIDIA RTX 2060 Super
- Windows 11 Pro 23H2

We also tested the NIDS on our Linux virtual machine which was running on Vmware Workstation Pro 17. The attacking machine that we used was a laptop running Ubuntu 22.04 having the following specifications:

- Intel Core i3 – 1215U @ 1.2 GHz
- 8GB LPDDR4 3200MHZ RAM
- Mesa Intel Graphics (ADL GT2)
- Kali Linux 2024.1

The attacking machine has both the software, Hping3 and NMAP installed on it. First we will start our NIDS program on our windows machine. Figure 3 shows the user interface of our program.

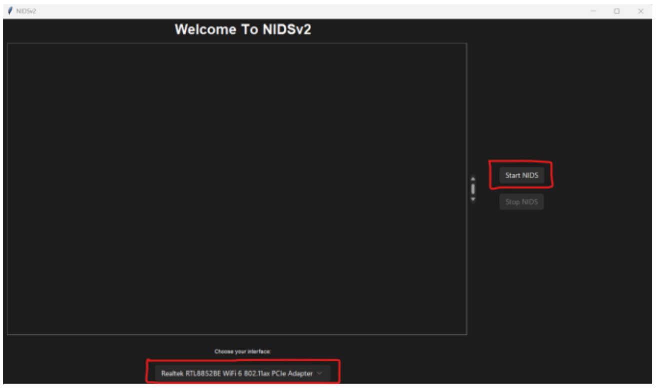
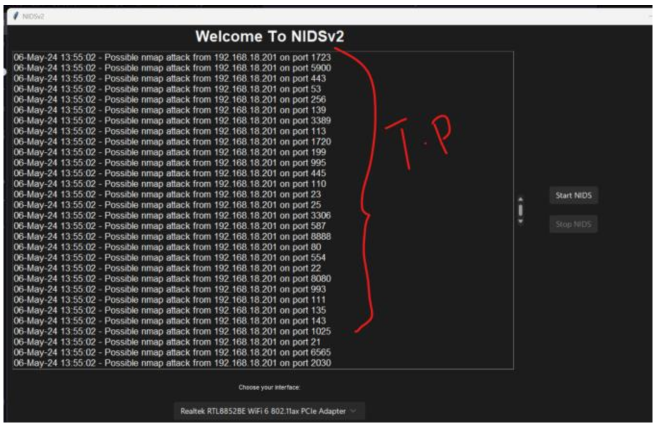


Figure 3. Shows the interface dropdown menu and starting the NIDS button

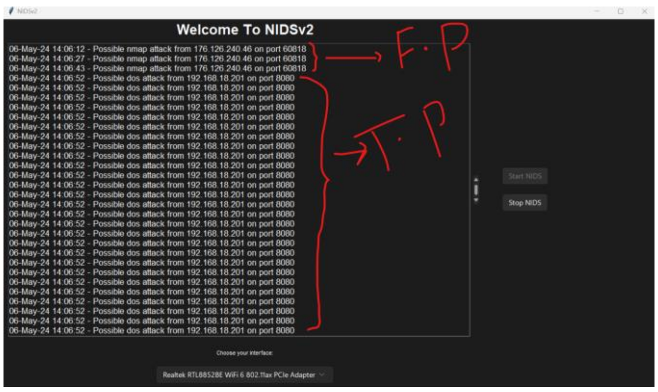
We will now start sniffing by clicking on the 'Start NIDS' button, before starting the NIDS we also have to choose the correct interface for Scapy to sniff on. On Kali Linux we will first run the NMAP command and will that our NIDS system has detected the NMAP Scan. Figure 4 shows that our NIDS system has detected an NMAP scan on our victim machine.



```
06-May-24 13:55:02 - Possible nmap attack from 192.168.18.201 on port 1723
06-May-24 13:55:02 - Possible nmap attack from 192.168.18.201 on port 5900
06-May-24 13:55:02 - Possible nmap attack from 192.168.18.201 on port 443
06-May-24 13:55:02 - Possible nmap attack from 192.168.18.201 on port 53
06-May-24 13:55:02 - Possible nmap attack from 192.168.18.201 on port 256
06-May-24 13:55:02 - Possible nmap attack from 192.168.18.201 on port 139
06-May-24 13:55:02 - Possible nmap attack from 192.168.18.201 on port 3389
06-May-24 13:55:02 - Possible nmap attack from 192.168.18.201 on port 113
06-May-24 13:55:02 - Possible nmap attack from 192.168.18.201 on port 1720
06-May-24 13:55:02 - Possible nmap attack from 192.168.18.201 on port 199
06-May-24 13:55:02 - Possible nmap attack from 192.168.18.201 on port 995
06-May-24 13:55:02 - Possible nmap attack from 192.168.18.201 on port 445
06-May-24 13:55:02 - Possible nmap attack from 192.168.18.201 on port 110
06-May-24 13:55:02 - Possible nmap attack from 192.168.18.201 on port 23
06-May-24 13:55:02 - Possible nmap attack from 192.168.18.201 on port 25
06-May-24 13:55:02 - Possible nmap attack from 192.168.18.201 on port 3306
06-May-24 13:55:02 - Possible nmap attack from 192.168.18.201 on port 587
06-May-24 13:55:02 - Possible nmap attack from 192.168.18.201 on port 8888
06-May-24 13:55:02 - Possible nmap attack from 192.168.18.201 on port 80
06-May-24 13:55:02 - Possible nmap attack from 192.168.18.201 on port 554
06-May-24 13:55:02 - Possible nmap attack from 192.168.18.201 on port 22
06-May-24 13:55:02 - Possible nmap attack from 192.168.18.201 on port 8080
06-May-24 13:55:02 - Possible nmap attack from 192.168.18.201 on port 993
06-May-24 13:55:02 - Possible nmap attack from 192.168.18.201 on port 111
06-May-24 13:55:02 - Possible nmap attack from 192.168.18.201 on port 135
06-May-24 13:55:02 - Possible nmap attack from 192.168.18.201 on port 143
06-May-24 13:55:02 - Possible nmap attack from 192.168.18.201 on port 1025
06-May-24 13:55:02 - Possible nmap attack from 192.168.18.201 on port 21
06-May-24 13:55:02 - Possible nmap attack from 192.168.18.201 on port 6565
06-May-24 13:55:02 - Possible nmap attack from 192.168.18.201 on port 2030
```

Figure 4. Shows a detailed log of nmap scan shown on the GUI of NIDS program

Next, we will use Hping 3 to perform DoS attack on the victim machine. Figure 5 shows the logs on GUI displaying the DoS attack being detected.



```
06-May-24 14:06:12 - Possible nmap attack from 176.126.240.46 on port 60818
06-May-24 14:06:27 - Possible nmap attack from 176.126.240.46 on port 60818
06-May-24 14:06:43 - Possible nmap attack from 176.126.240.46 on port 60818
06-May-24 14:06:52 - Possible dos attack from 192.168.18.201 on port 8080
06-May-24 14:06:52 - Possible dos attack from 192.168.18.201 on port 8080
06-May-24 14:06:52 - Possible dos attack from 192.168.18.201 on port 8080
06-May-24 14:06:52 - Possible dos attack from 192.168.18.201 on port 8080
06-May-24 14:06:52 - Possible dos attack from 192.168.18.201 on port 8080
06-May-24 14:06:52 - Possible dos attack from 192.168.18.201 on port 8080
06-May-24 14:06:52 - Possible dos attack from 192.168.18.201 on port 8080
06-May-24 14:06:52 - Possible dos attack from 192.168.18.201 on port 8080
06-May-24 14:06:52 - Possible dos attack from 192.168.18.201 on port 8080
06-May-24 14:06:52 - Possible dos attack from 192.168.18.201 on port 8080
06-May-24 14:06:52 - Possible dos attack from 192.168.18.201 on port 8080
06-May-24 14:06:52 - Possible dos attack from 192.168.18.201 on port 8080
06-May-24 14:06:52 - Possible dos attack from 192.168.18.201 on port 8080
06-May-24 14:06:52 - Possible dos attack from 192.168.18.201 on port 8080
06-May-24 14:06:52 - Possible dos attack from 192.168.18.201 on port 8080
06-May-24 14:06:52 - Possible dos attack from 192.168.18.201 on port 8080
06-May-24 14:06:52 - Possible dos attack from 192.168.18.201 on port 8080
06-May-24 14:06:52 - Possible dos attack from 192.168.18.201 on port 8080
06-May-24 14:06:52 - Possible dos attack from 192.168.18.201 on port 8080
06-May-24 14:06:52 - Possible dos attack from 192.168.18.201 on port 8080
06-May-24 14:06:52 - Possible dos attack from 192.168.18.201 on port 8080
06-May-24 14:06:52 - Possible dos attack from 192.168.18.201 on port 8080
06-May-24 14:06:52 - Possible dos attack from 192.168.18.201 on port 8080
06-May-24 14:06:52 - Possible dos attack from 192.168.18.201 on port 8080
06-May-24 14:06:52 - Possible dos attack from 192.168.18.201 on port 8080
06-May-24 14:06:52 - Possible dos attack from 192.168.18.201 on port 8080
06-May-24 14:06:52 - Possible dos attack from 192.168.18.201 on port 8080
```

Figure 5. Shows a detailed log of DoS attack shown on the GUI of NIDS program.

Now, we will try to test our YARA rule matching module, I have a web application called Damn Vulnerable Web Application (DVWA) running on a virtual machine. I will start my NIDS on that Linux machine where I have hosted my Web Application on Port 80. Figure 6 shows that we can access the web application from our attacking machine.

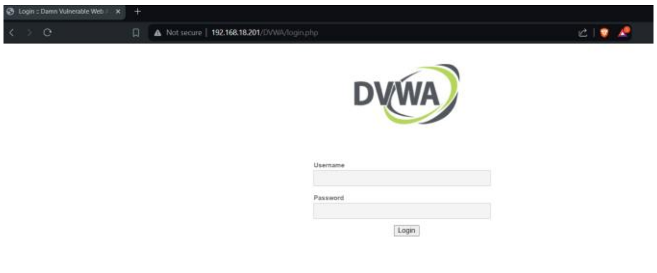
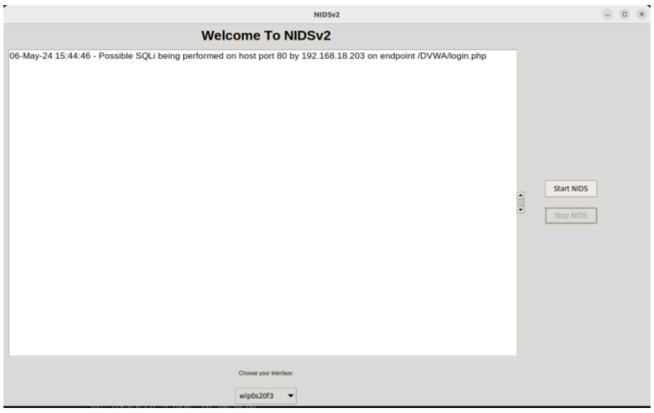

Login :: Damn Vulnerable Web
Not secure | 192.168.18.201/DVWA/login.php
DVWA
Username
Password
Login

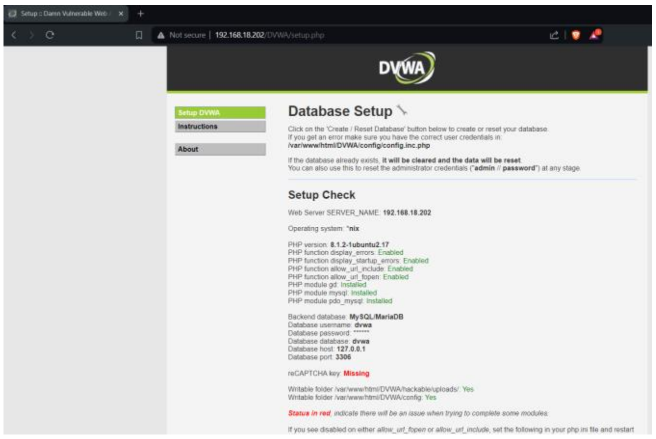
Figure 6. Shows an image of the web application running on a linux system.

I will run a basic SQL injection query `admin’ or 1=1` on the input form, the NIDS running on the victim machine should detect this SQL injection attack using YARA rules. Figure 7 shows a successful yara rule match bein shown on the NIDS.



```
06-May-24 15:44:46 - Possible SQLi being performed on host port 80 by 192.168.18.203 on endpoint /DVWA/login.php
```

Figure 7. Shows the result of the NIDS logs displaying a SQLi being found.


Setup :: Damn Vulnerable Web
Not secure | 192.168.18.202/DVWA/setup.php
DVWA
Setup DVWA
Instructions
About

Database Setup

Click on the 'Create / Reset Database' button below to create or reset your database.
If you get an error make sure you have the correct user credentials in:
/var/www/html/DVWA/config/config.inc.php

If the database already exists, **it will be cleared and the data will be reset**.
You can also use this to reset the administrator credentials ("**admin** // **password**") at any stage.

Setup Check

Web Server SERVER_NAME: **192.168.18.202**

Operating system: ***nix**

PHP version: **8.1.2-1ubuntu2.17**
PHP function display_errors: Enabled
PHP function display_startup_errors: Enabled
PHP function allow_url_include: Enabled
PHP function allow_url_fopen: Enabled
PHP module gd: Installed
PHP module mysql: Installed
PHP module pdo_mysql: Installed

Backend database: **MySQL/MariaDB**
Database username: **dvwa**
Database password: ******
Database database: **dvwa**
Database host: **127.0.0.1**
Database port: **3306**

reCAPTCHA key: **Missing**

Writable folder /var/www/html/DVWA/hackable/uploads/: Yes
Writable folder /var/www/html/DVWA/config: Yes

Status in red, indicate there will be an issue when trying to complete some modules.

If you see disabled on either *allow_url_fopen* or *allow_url_include*, set the following in your php.ini file and restart

Figure 8. Shows that the SQLi was successful and we got access to the page.

Figure 8 shows that the SQL injection that we performed was successful and it also matched the yara rule signature.

CONCLUSION

Overall, creating a Network Intrusion Detection System was a challenging and fun task that we accomplished. It helped in understanding a crucial phase of machine learning, that is, data collection and data cleaning. We also learned about classification algorithms while making our NIDS project and network analysis using scapy. We also learned how yara rules cannot be used to detect malware but also be used for signature detection in NIDS. We still think that there are improvements to be made in this project, mainly in our machine learning phase. We think that a better dataset can further improve our model, using an aggregated dataset is much preferable when designing and NIDS system. More attack features such as SSH brute-force, FTP brute-force, and web application brute-force should be introduced in the dataset so that the NIDS can detect multiple attacks. The performance of our NIDS is slow as compared to modern NIDS built by professional developers. We believe that performance can be enhanced by choosing a programming language which compiles the source code, such as C++, rather than interprets the code, such as Python.

REFERENCE

[1] M. Aljanabi, M. A. Ismail, and A. H. Ali, “Intrusion detection systems, issues, challenges, and needs,” *International Journal of Computational Intelligence Systems*, vol. 14, no. 1, p. 560, 2021. doi:10.2991/ijcis.d.210105.001

[2] Z. Ahmad, A. Shahid Khan, C. Wai Shiang, J. Abdullah, and F. Ahmad, “Network intrusion detection system: A systematic study of machine learning and Deep Learning Approaches,” Transactions on Emerging Telecommunications Technologies, vol. 32, no. 1, Oct. 2020. doi:10.1002/ett.4150

[3] B. Ingre, A. Yadav, and A. K. Soni, “Decision tree based intrusion detection system for NSL-KDD dataset,” Information and Communication Technology for Intelligent Systems (ICTIS 2017) - Volume 2, pp. 207–218, Aug. 2017. doi:10.1007/978-3-319-63645-0_23

[4] “Writing Yara Rules¶,” Writing YARA rules - yara 4.4.0 documentation, https://yara.readthedocs.io/en/stable/writingrules.html (accessed May 7, 2024).

[5] Scapy, https://scapy.net/ (accessed May 7, 2024).