Machine Learning: Preventing Network Abnormalities

Author: Chad Mascari, cmasli@protonmail.com
Advisor: *Dr. Tim Proffitt*

Accepted: *17 July 2024*

Abstract

The Department of Defense (DoD) developed and published multiple zero trust documents describing the zero trust principles that DoD organizations should achieve. The documents state that organizations will need to rely on Artificial Intelligence, machine learning, and automation to reduce the time a security practitioner needs to monitor, detect, and prevent unauthorized user and device access to network resources. The DoD operates endpoint devices and networks disconnected from the public internet, driving a need for disconnected machine learning models. The research paper outlines the potential for an on-premises machine learning algorithm at the endpoint device to analyze normal and abnormal network traffic and automatically implement Windows Defender Firewall rulesets. The research outlines the challenges to implementing this concept at the endpoint device instead of relying on centralized or cloud-based machine learning platforms.

1. Introduction

The Department of Defense Office of the Chief Information Officer (DoD CIO) published multiple documents related to zero trust architecture and zero trust network designs. They outline architectural and engineering guidance to create information and operational technology environments that proactively reduce inherent trust for users, devices, and applications to access, share, store, process, and modify an organization's data. The three primary documents from the DoD CIO's office are the DoD Zero Trust Reference Architecture (ZTRA), the DoD Zero Trust Overlays, and the DoD Zero Trust Strategy.

The DoD's ZTRA provides a strategic framework for architects and engineers to design a more secure networked environment regardless of the infrastructure and location of the users, devices, applications, data, and services (*Department of Defense Zero Trust Reference Architecture,* 2022). The DoD ZT Overlay establishes pillars, capabilities, activities, and expected outcomes that guide the implementation, integration, cultural adoptions, and governance of technical solutions for organizations to align with ZTRA and Zero Trust Strategy (*Department of Defense Zero Trust Overlays,* 2024). The DoD Zero Trust Strategy is the DoD's plan to modernize the DoD's networked environments to resist attacks and data compromise by Fiscal Year 2027. The documents rely on technical principles, security controls, and technical capabilities in Artificial Intelligence, Machine Learning, and Automation to decrease the time for security practitioners to detect, prevent, and respond to cyber-based threats and attacks.

The DoD will leverage Artificial Intelligence and Machine Learning (AI/ML) to effectively analyze large datasets to detect, prevent, monitor, and correct deviations from established and authorized baselines within a networked environment according to the organization's governance and policy. A policy decision point (PDP) consists of a Policy Administrator (PA) and a Policy Engine (PE) to determine authorized access to network resources. The PA defines the organization's data access policy for the PE to analyze information and determine if the user's and device's attributes are authorized to access a network resource. The Policy Enforcement Points (PEPs) are technical controls that enforce the PDP's policy decision to access network resources in real time for a user and device (Rose et al., 2020).

Organizations with networked environments disconnected from the public internet require high-fidelity, micro-segmented security controls that leverage machine learning and automation to mitigate risk of lateral movement and unauthorized endpoint data access associated with processing highly sensitive data. Security practitioners must analyze data flowing through firewalls, intrusion prevention systems, and intrusion detection systems for deviations from an authorized policy decision. A PDP needs to analyze the information, or logs captured from the devices to determine a policy decision effectively. Machine learning will enable the PDP to interpret the information for a policy decision and pass the decision to a PEP. ZTRA relies on automation by integrating technologies between the data plane and control plane to push a decision from the PDP to the PEP seamlessly.

Developing a machine learning model that analyzes packet capture (PCAP) data on the endpoint would collapse the PDP and PEP architecture to Python and Windows Defender Firewall installed on the endpoint device. Shifting the PDP and PEP architecture to an endpoint device will enable organizations to implement high-fidelity security controls for endpoints and networks disconnected from the public internet. The research analyzes the requirements to develop a machine learning model in Python capable of analyzing PCAP datasets, automatically labeling PCAP datasets based on well-known Indicators of Compromise (IoCs), and automatically deploying Windows Defender Firewall rulesets. Implementing a novel data labeling methodology based on specific IoCs for supervised machine learning algorithms significantly enhances network analysis performance on a Windows 11 endpoint, specifically optimizing system resource utilization in terms of memory and CPU consumption.

2. Research Methodology

The research will test the impacts of a Python script using the XGBoost Machine Learning library on a Windows 11 virtual system over four separate tests. The dependent variables will be the system's resources; memory and Compute Processing Units (CPUs).

The tests will measure the system's resource utilization to classify, label, and analyze PCAP datasets. The control variable for the first three tests is a benign PCAP dataset and XGBoost. Controlling the dataset during the first three tests will determine if the methodology to classify and label the dataset impacts the system's resources. The Python script will need to consume processing power and speed to classify the packet data based on indicators of compromise. As the script classifies the dataset, Python will consume memory to apply labels to the classified packet data. XGBoost will need to consume processing power to analyze the labelled dataset. The fourth test will change the PCAP dataset to an independent variable. This last phase of testing will determine if the size of the dataset impacts system resources.

The independent variable for the fourth test is a smaller benign PCAP dataset. The controlled variables will be XGBoost as the machine learning model and the methodology used in Test 3 to classify and label the dataset. The dependent variables will be the system's memory and CPU. Python will consume less memory to classify and label a smaller dataset size since less data needs to be processed by the script. The four tests intend to identify methods to efficiently employ machine learning algorithms without impacting the system's resources or to assist organizations to plan for system resource requirements to employ machine learning algorithms at an endpoint device.

2.1. Literature Review for ML Firewall Rule Generation

Organizations can only partially adopt commercial AI/ML capabilities if vendors produce solutions sold as software-as-a-service, requiring public internet access. Previous research concludes that machine learning models can differentiate between malicious and regular traffic and produce applicable firewall rules from the model's analysis (Verbruggen, 2014). Verbruggen's study compared multiple ML models and provided the results from their PCAP analysis. Decision Tree ML models may misrepresent and mis-categorize datasets, producing a higher rate of false positives and true negatives, resulting in unwanted firewall rules that may allow malicious traffic or prevent authorized traffic (Iandoli et al., 2021). Iandoli's research solely discusses the results of the ML model but does not discuss how the researcher developed their models and the environmental impacts on the system used for analysis. Other machine learning projects developed methods to implement ML models to analyze PCAPs but forced users to label packets within a comma-separated values file manually.

2.2. Developing a Machine Learning Model

"Basic Machine Learning: Network Anomaly Detection" is a blog post that inspired the ML model developed for the project (Nauni, 2021). The blog post outlines five key steps to build, train, and test machine learning models for network anomaly detection. The five steps are dataset creation, feature extraction from datasets, cleaning, labeling, and then training the ML model (Nauni, 2021). The blog post provided a GitHub link to the author's Python Scripts for their network analysis ML model.

The author's GitHub page provided four independent scripts to extract features from the datasets, cleaning the datasets, labelling the datasets, and training the ML model. The process was not automated to do each step within a single Python script. This process forced the user to rely on TShark to extract features from the PCAP dataset. The ML model did not natively accept PCAP datasets. The user had to convert the extracted features from the PCAP dataset to a comma-separated values (CSV) dataset and iteratively save new CSV datasets to clean and label the dataset before the ML model could be trained.

Reviewing the blog post resulted in a better understanding of machine learning concepts for PCAP analysis, methods to automate the steps into a single script, and to provide a user-friendly model that natively handled PCAP datasets.

3. Test Design

3.1. Dataset Creation, Labeling, Feature Extraction

The datasets used for the ML model consist of pre-existing PCAP files from other open-source repositories that provided benign, malicious, and mixed datasets. The resource for datasets was the Stratosphere Research Laboratory's Stratosphere IPS datasets. Sebastian Garcia created the research lab for his PhD program at the Czech Technical University (CTU) of Prague. Researchers at the Stratosphere Labs developed datasets that advance the research of AI/ML for network defense and provide detailed information about each dataset. The Stratosphere IPS datasets provided a constant control variable to test dataset labeling features for the ML model in this research project (Garcia, 2017).

Tests 1, 2, and 3 used the 2017-05-02_kali-normal.pcap file contained in the CTU-Normal-22 dataset repository (Garcia, 2017). The PCAP was captured on a Kali Linux system using Dumpcap (WireShark) and contains typical connections to websites in Alexa's top 1,000. The dataset contains 2,616,990 packets, the average packet size is 621.98 bytes, the file size is 1,716 MB, and the data size is 1,627 MB.

Test 4 will use the 2017-05-02_kali-normal.pcap file contained in the CTU-Normal-21 dataset repository (Garcia, 2017). Garcia captured the dataset under the same conditions as the CTU-Normal-22 dataset, but the file size is 311 MBs and contains 442,488 packets.

3.2. Dataset Feature Extraction and Labelling

The Python script used Scapy to extract features from the PCAP files and label the datasets based on IoCs. The extracted features were the source and destination IP addresses, IP Total Length, Time to Live, Protocols, fragmented packets, Window Size, TCP flags, TCP and UDP ports, HTTP request and response payloads, and SMB header data.

The script labelled the features by returning a 1 for malicious packets and a 0 for benign packets. The script natively handled cleaning the datasets by assigning a null value if specified features did not exist within the dataset. The script analyzed the Server Message Block versions 1 and 2 (SMB) headers, Domain Name Service (DNS) protocol and UDP port 53, the Transmission Control Protocol (TCP) flags, the payload for Hypertext Transport Protocol (HTTP) requests against the benign PCAP datasets.

Analyzing SMBv1 and v2 could indicate potential lateral movement of malware if the connection accessed a user directory within the PCAP. Analyzing UDP Port 53 and the DNS protocol intended to detect potential botnet activity. Analyzing TCP flags like Finish (FIN), Push (PSH), and Urgent (URG) could indicate either botnet activity or network scan activity. Analyzing HTTP requests accessing a path starting with /admin could identify unauthorized privileged access to a web server. XGBoost used the labelled features to produce the number of malicious and benign packets, an accuracy score, and the importance or weight of the feature when evaluating the dataset.

3.3. ML Model

The ML model uses the XGBoost Python library. XGBoost is a supervised learning model based on the gradient boosting technique and uses labeled datasets to train and predict outcomes. The XGBoost library is available in multiple coding languages like Python, C++, and Ruby (xgboost developers, 2022). The research evaluated the model based on the number of packets labeled as malicious versus benign and the accuracy percentage from the dataset.

3.4. Test System

The system used to develop and test the Python script was a virtual machine hosted on VMware Workstation Pro. The operating system for the virtual machine was Windows 11 Home Version 10.0.22631 Build 22631. The virtual machine had 32GB of memory, 12 CPU cores, and 64 GB of NVMe Hard Drive space. The physical host's operating system was Windows 11 Pro, the CPU was an Intel Core i7-8700K at 3.70 GHz CPU, and the physical system had four 16 GB DDR4-3200 memory.

4. Findings and Discussion

4.1. Evaluating the Effectiveness of an ML Model

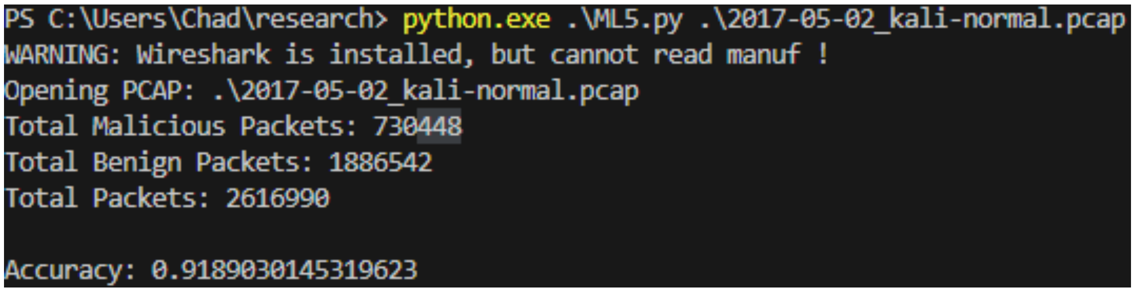
Organizations may need to develop custom and on-premises AI/ML solutions capable of analyzing packet captures for malicious, abnormal, and deviated network traffic and automate the implementation of firewall rules to prevent unauthorized traffic at an endpoint device. Security practitioners are encouraged to validate the performance of machine learning models against actual data since network flows and traffic change quickly, causing a high degree of drift from authorized baselines. Security practitioners should also validate a machine learning algorithm against malicious, standard, and background PCAP datasets. Malicious datasets should include all the indicators that need to be detected by the algorithm, typical datasets test the rate of false positives and true negatives, and background datasets test the algorithm's performance to address the resource requirements for the system running the machine learning model (Garcia, 2017).

Based on the test requirements to evaluate an effective machine learning model, the project's scope was reduced to test benign PCAP datasets against the detection rate of false positives and true negatives and the resource consumption needed to analyze PCAP files. The scope was also reduced because the laboratory environment required more system resources to generate Windows Defender firewall rulesets. However, it still provided enough findings to evaluate the effectiveness of the machine learning model.

4.2. Test 1

4.2.1. Evaluating False Positive and True Negative Rates

The first test was completed against the benign PCAP dataset to baseline the machine learning model by judging the rate of false positives and true negatives from the dataset to gauge the model's accuracy. Figure 1 displays the malicious and benign packets the machine learning model labeled. The model labeled 730,448 packets as malicious and 1,886,542 packets as benign, and the model resulted in a 91.8% accuracy.

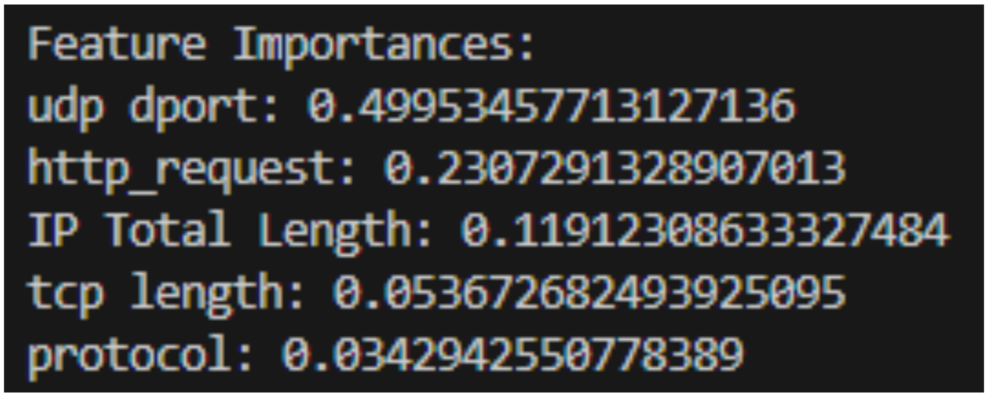


```
PS C:\Users\Chad\research> python.exe .\ML5.py .\2017-05-02_kali-normal.pcap
WARNING: Wireshark is installed, but cannot read manuf !
Opening PCAP: .\2017-05-02_kali-normal.pcap
Total Malicious Packets: 730448
Total Benign Packets: 1886542
Total Packets: 2616990

Accuracy: 0.9189030145319623
```

Figure 1: Labelled Malicious and Benign Packets for Test 1

The expected result should be 0 malicious packets, 2,616,990 packets as benign, and an accuracy of 100%. In Figure 2, the most crucial feature analyzed by the machine learning model was the UDP Destination port 53 to analyze DNS traffic for malicious activity. Since the script marks any packet containing UDP and a destination port 53, the results indicate that the conditions to label DNS traffic as malicious are too generic.



```
Feature Importances:
udp dport: 0.49953457713127136
http_request: 0.2307291328907013
IP Total Length: 0.11912308633327484
tcp length: 0.053672682493925095
protocol: 0.0342942550778389
```

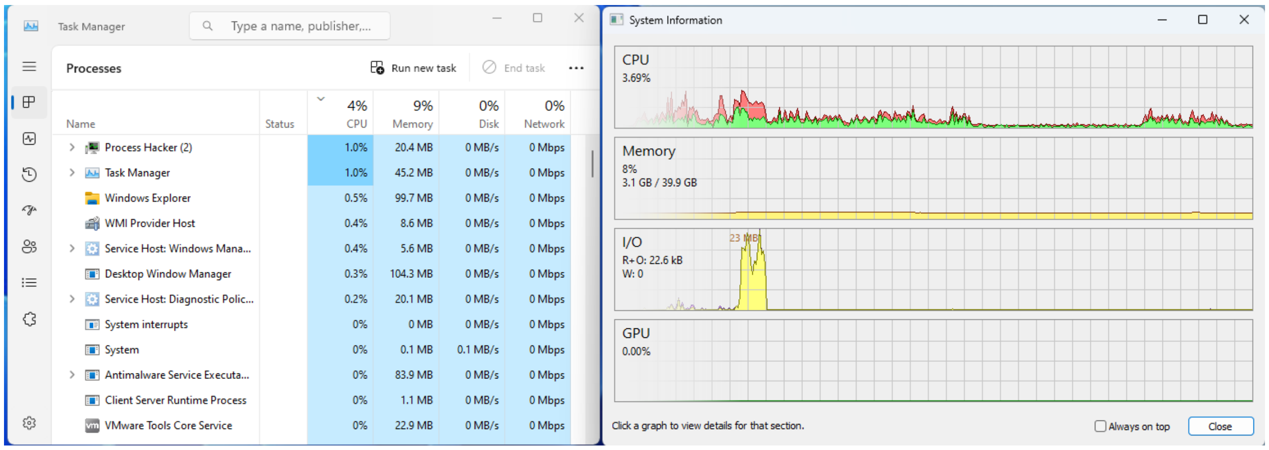
Figure 2: Feature Importance for Test 1

The discrepancy between the expected and actual results is due to the conditions that label all DNS traffic in the PCAP dataset as malicious. The second test evaluated the system resource utilization to analyze the PCAP dataset without modifying the methodology to label the dataset or the PCAP dataset used to evaluate the machine learning model.

4.2.2. Evaluating System Resource Utilization

Process Hacker, Task Manager, and Process Explorer evaluated the system's performance and resource utilization during the test. In Figure 3, displays the baseline of the system's resource utilization before the tests were performed. While the test system was idling with PowerShell open, the system utilized the following percentages of resources:

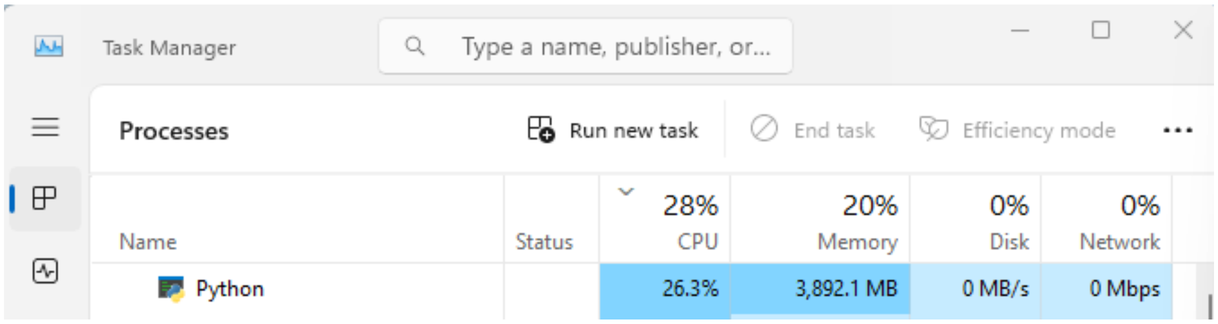
- CPU: 3 to 5% across all cores
- Memory: 8% to 9% of all memory space
- Disk: 0% total utilization across all disks
- GPU: 0%



Name	Status	4% CPU	9% Memory	0% Disk	0% Network
Process Hacker (2)		1.0%	20.4 MB	0 MB/s	0 Mbps
Task Manager		1.0%	45.2 MB	0 MB/s	0 Mbps
Windows Explorer		0.5%	99.7 MB	0 MB/s	0 Mbps
WMI Provider Host		0.4%	8.6 MB	0 MB/s	0 Mbps
Service Host: Windows Mana...		0.4%	5.6 MB	0 MB/s	0 Mbps
Desktop Window Manager		0.3%	104.3 MB	0 MB/s	0 Mbps
Service Host: Diagnostic Polic...		0.2%	20.1 MB	0 MB/s	0 Mbps
System interrupts		0%	0 MB	0 MB/s	0 Mbps
System		0%	0.1 MB	0.1 MB/s	0 Mbps
Antimalware Service Executa...		0%	83.9 MB	0 MB/s	0 Mbps
Client Server Runtime Process		0%	1.1 MB	0 MB/s	0 Mbps
VMware Tools Core Service		0%	22.9 MB	0 MB/s	0 Mbps

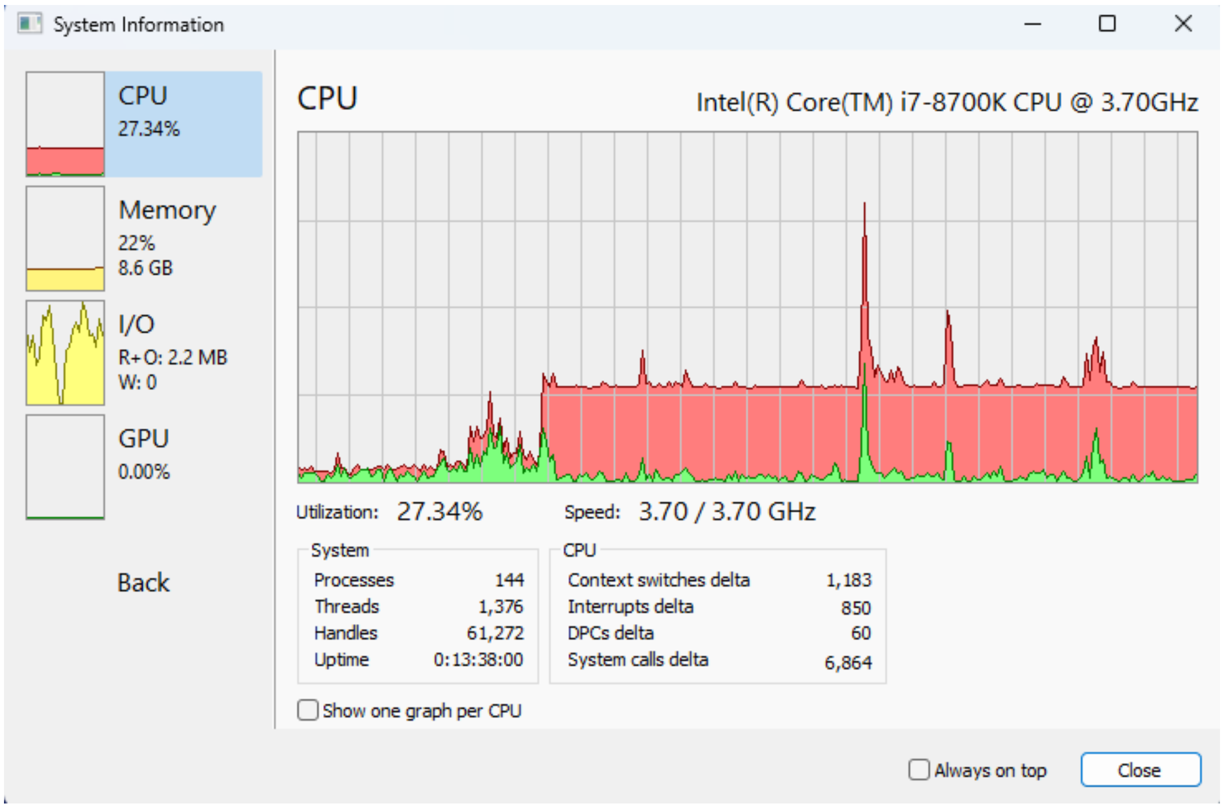
Figure 3: Baseline System Resource Utilization

At the initial execution of the machine learning model, the system consumed 28% of all CPU cores. In Figures 4 and 5, Python.exe consumed approximately 26.3% of the CPU cores; the system consumed a consistent range between 28 to 30% of CPU utilization. Python.exe consistently consumed 24% to 28% of CPU cores during the program's execution. The system's disk utilization increased from 1% to 2% while Python executed the machine learning model. When the ML Model analyzed the dataset and produced results, Python.exe's CPU utilization spiked from 80% to 90%.



Name	Status	28% CPU	20% Memory	0% Disk	0% Network
Python		26.3%	3,892.1 MB	0 MB/s	0 Mbps

Figure 4: System Resource Utilization During Test 1



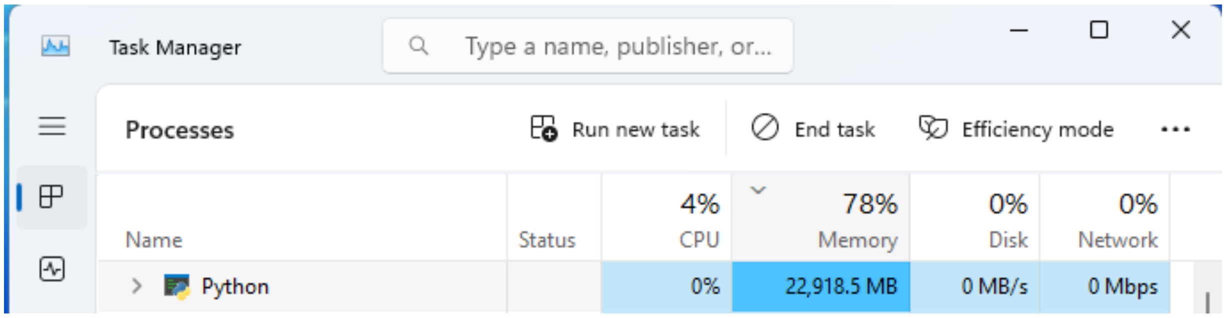
Utilization: 27.34% Speed: 3.70 / 3.70 GHz

System	
Processes	144
Threads	1,376
Handles	61,272
Uptime	0:13:38:00

CPU	
Context switches delta	1,183
Interrupts delta	850
DPCs delta	60
System calls delta	6,864

Figure 5: System's CPU Utilization During Test 1

Figure 6 displays Python.exe's memory utilization at the end of the model's analysis since Python stores the analyzed data within variables. Python.exe consumed approximately 22,918.5 MB, resulting in the system consuming 78% of total memory space.



Name	Status	4% CPU	78% Memory	0% Disk	0% Network
Python		0%	22,918.5 MB	0 MB/s	0 Mbps

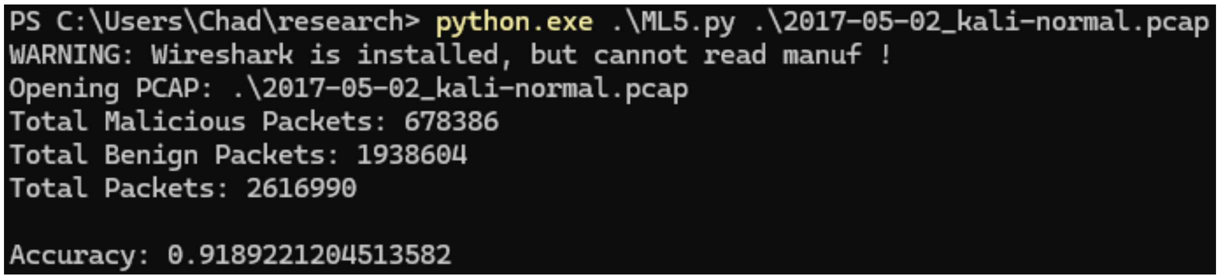
Figure 6: Memory Utilization After Test 1

4.3. Test 2

The second test changed the data labeling conditions to analyze DNS packets. The script only labels malicious DNS traffic if the destination UDP port is 53 and the Destination IP Address is not the DNS server used in the PCAP dataset. The dataset only contained one DNS server. The script should label fewer malicious packets from the benign PCAP dataset, resulting in a more effective Windows Defender Firewall rule that would not block all UDP destination port 53 traffic. The change to labeling DNS traffic should not affect system resource utilization. The script must store the equivalent number of packets in memory for the model to analyze.

4.3.1. Evaluating False Positive and True Negative Rates

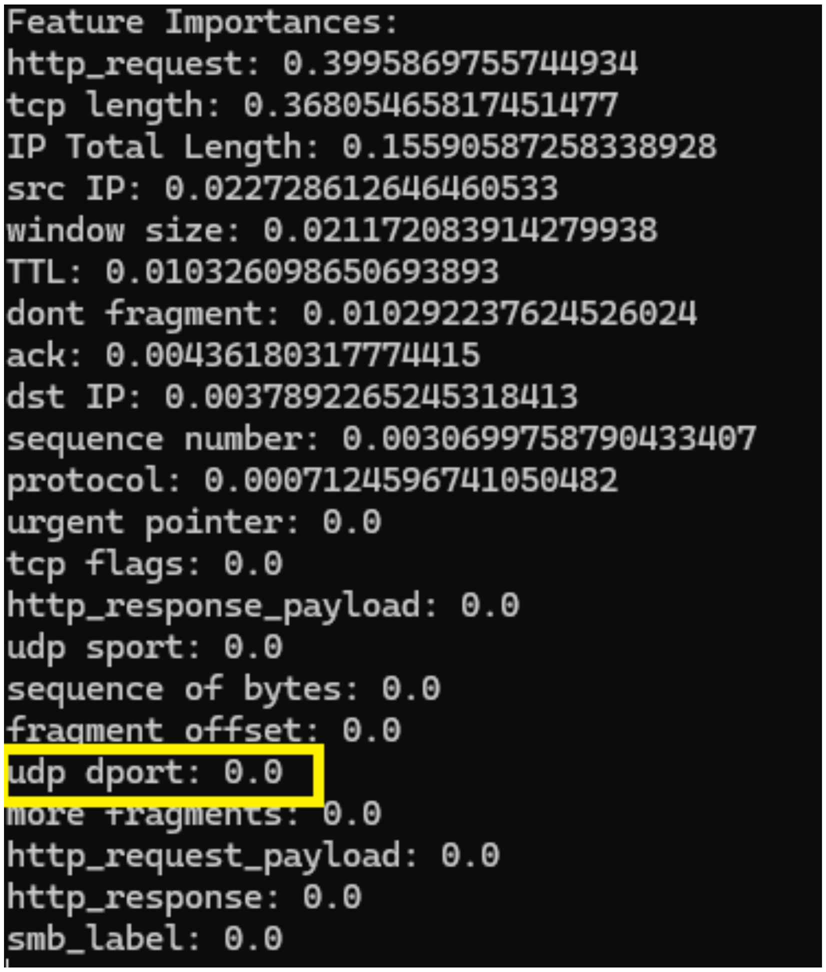
The second test resulted in 678,386 malicious packets, 1,938,604 benign packets, and an accuracy rate of 91.8%. Defining a new methodology to label malicious datasets reduces the number of malicious packets in the PCAP dataset by 52,062 between the two tests. The accuracy of the machine learning model was not affected between the two tests. The feature importance of the UDP's destination port decreased to a weight of 0 for the model to analyze the benign dataset.



```
PS C:\Users\Chad\research> python.exe .\ML5.py .\2017-05-02_kali-normal.pcap
WARNING: Wireshark is installed, but cannot read manuf !
Opening PCAP: .\2017-05-02_kali-normal.pcap
Total Malicious Packets: 678386
Total Benign Packets: 1938604
Total Packets: 2616990

Accuracy: 0.9189221204513582
```

Figure 7: Labelled Malicious and Benign Packets for Test 2



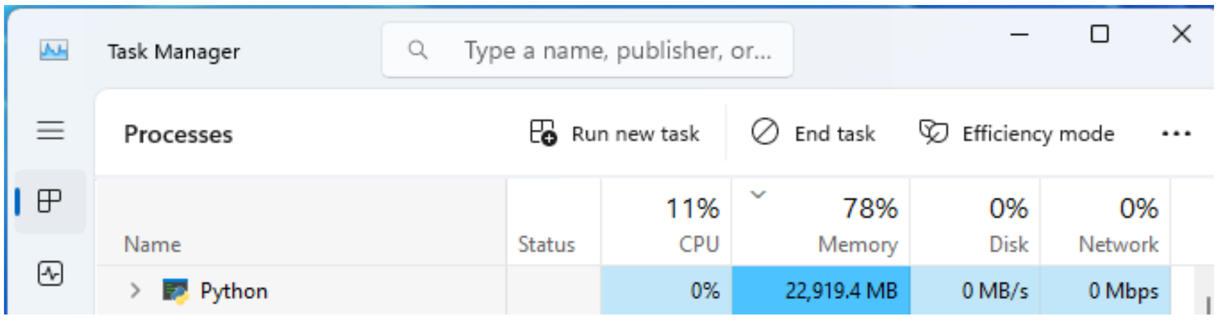
```
Feature Importances:
http_request: 0.3995869755744934
tcp length: 0.36805465817451477
IP Total Length: 0.15590587258338928
src IP: 0.022728612646460533
window size: 0.021172083914279938
TTL: 0.010326098650693893
dont fragment: 0.010292237624526024
ack: 0.00436180317774415
dst IP: 0.0037892265245318413
sequence number: 0.0030699758790433407
protocol: 0.0007124596741050482
urgent pointer: 0.0
tcp flags: 0.0
http_response_payload: 0.0
udp sport: 0.0
sequence of bytes: 0.0
fragment offset: 0.0
udp dport: 0.0
more fragments: 0.0
http_request_payload: 0.0
http_response: 0.0
smb_label: 0.0
```

Figure 8: Feature Importance for Test 2

Changing the data labeling conditions for DNS did not significantly affect the model. However, the change would affect the decision of the model to implement a Windows Defender Firewall rule for DNS since the traffic would have been malicious in Test 1 but not Test 2.

4.3.2. Evaluating System Resource Utilization

Changing the methodology to label DNS packets in the dataset did not significantly change the system resource utilization. During the second test, Python.exe utilized the same CPU resources as annotated during the first test. After the second test, Python.exe consumed .8 MB more of the system's memory, a total of 22,919.4 MBs.



Name	Status	11% CPU	78% Memory	0% Disk	0% Network
Python		0%	22,919.4 MB	0 MB/s	0 Mbps

Figure 9: Memory Utilization After Test 2

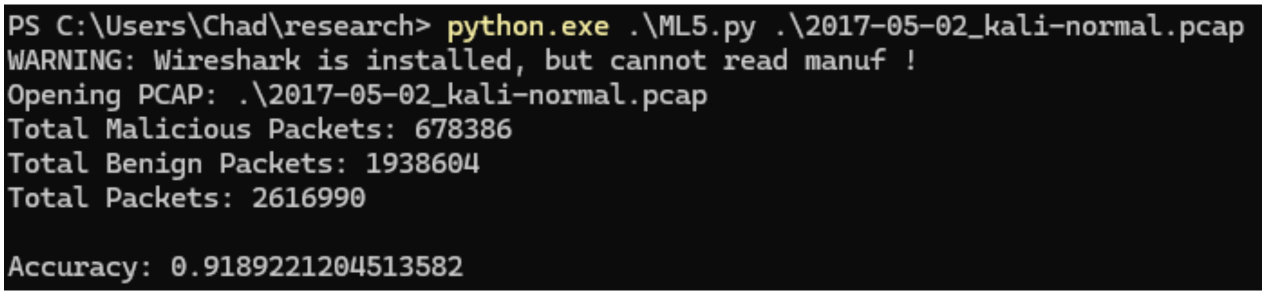
The script must consume fewer system resources to be an effective model that automatically creates Windows Defender Firewall rulesets as the model analyzes mixed datasets and PCAP files.

4.4. Test 3

Test 3 removes all data labeling conditions for DNS traffic as malicious or benign. Test 3 intends to affect the memory resource utilization by Python.exe. The script should not need to store as much data from the packet for the machine learning model to analyze and should reduce the memory required to analyze the benign PCAP dataset. The number of malicious and benign packets should not change between Test 2 and Test 3 and thus will not affect the effectiveness of developing Windows Defender Firewall rulesets.

4.4.1. Evaluating False Positive and True Negative Rates

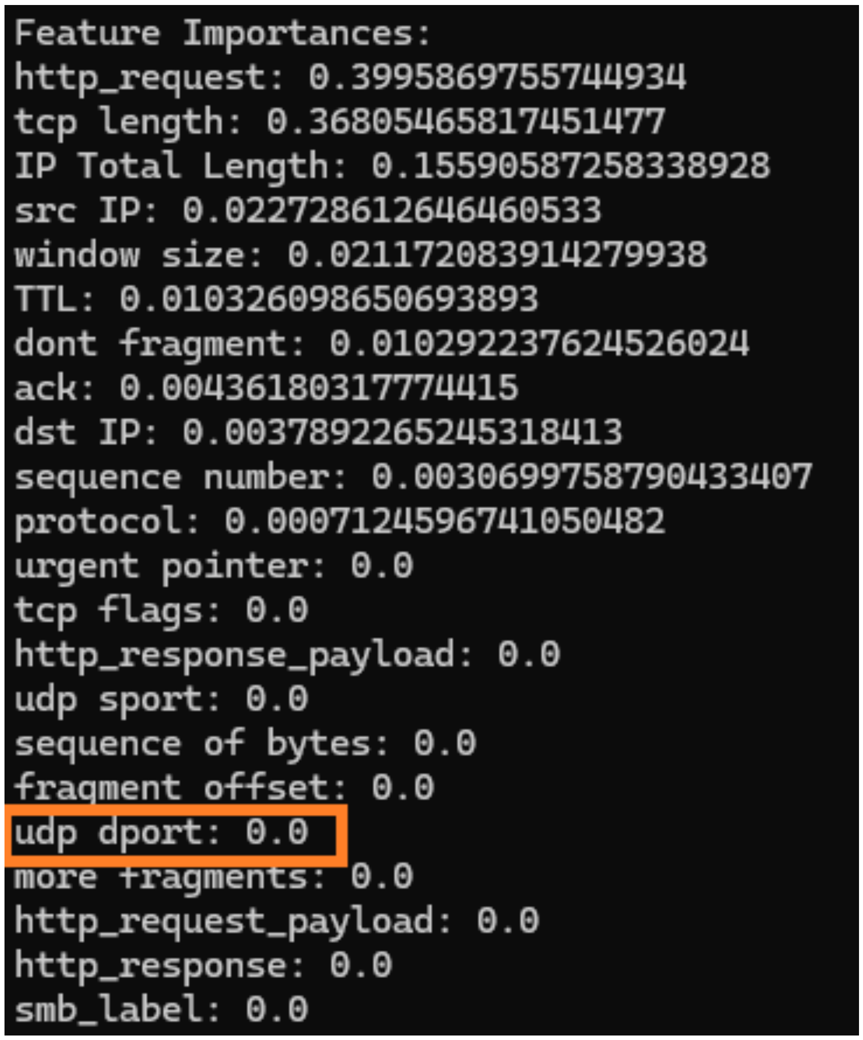
The amount of malicious and benign packets did not change between Tests 2 and 3. The accuracy of the model remained the same across all three tests. The importance of features between Tests 2 and 3 remained the same.



```
PS C:\Users\Chad\research> python.exe .\ML5.py .\2017-05-02_kali-normal.pcap
WARNING: Wireshark is installed, but cannot read manuf !
Opening PCAP: .\2017-05-02_kali-normal.pcap
Total Malicious Packets: 678386
Total Benign Packets: 1938604
Total Packets: 2616990

Accuracy: 0.9189221204513582
```

Figure 10: Labelled Malicious and Benign Packets for Test 3



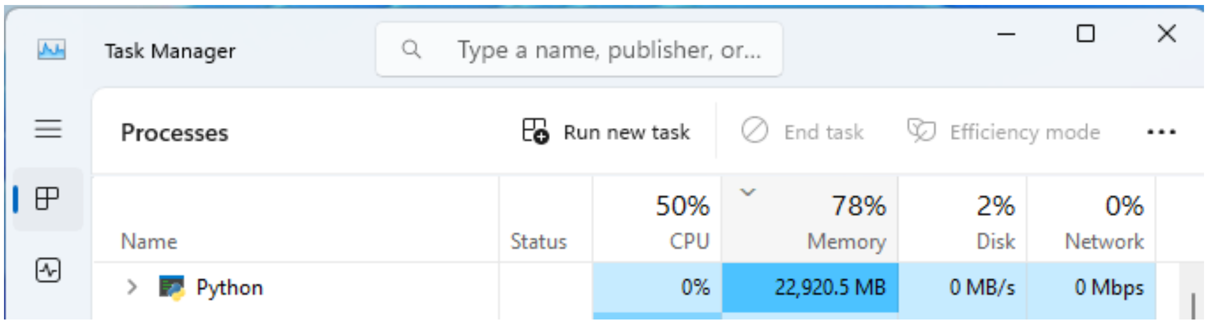
```
Feature Importances:
http_request: 0.3995869755744934
tcp length: 0.36805465817451477
IP Total Length: 0.15590587258338928
src IP: 0.022728612646460533
window size: 0.021172083914279938
TTL: 0.010326098650693893
dont fragment: 0.010292237624526024
ack: 0.00436180317774415
dst IP: 0.0037892265245318413
sequence number: 0.0030699758790433407
protocol: 0.0007124596741050482
urgent pointer: 0.0
tcp flags: 0.0
http_response_payload: 0.0
udp sport: 0.0
sequence of bytes: 0.0
fragment offset: 0.0
udp dport: 0.0
more fragments: 0.0
http_request_payload: 0.0
http_response: 0.0
smb_label: 0.0
```

Figure 11: Feature Importance for Test 3

The machine learning model's results between Test 2 and Test 3 validate that measuring system resource utilization was controlled.

4.4.2. Evaluating System Resource Utilization

Python.exe consistently consumed 24% to 27% CPU utilization during Test 3 to maintain a consistent data point for CPU utilization from the research across all three tests. Test 3 proved that removing a condition to label DNS as malicious or benign did not affect Python.exe's memory utilization. In Test 3, Python.exe consumed 22,920.5 MB of memory. In Test 2, Python.exe consumed 22,919.4 MBs of memory. In test 1, Python.exe consumed 22,918.5 MBs of memory.



Name	Status	50% CPU	78% Memory	2% Disk	0% Network
Python		0%	22,920.5 MB	0 MB/s	0 Mbps

Figure 12: Memory Utilization After Test 3

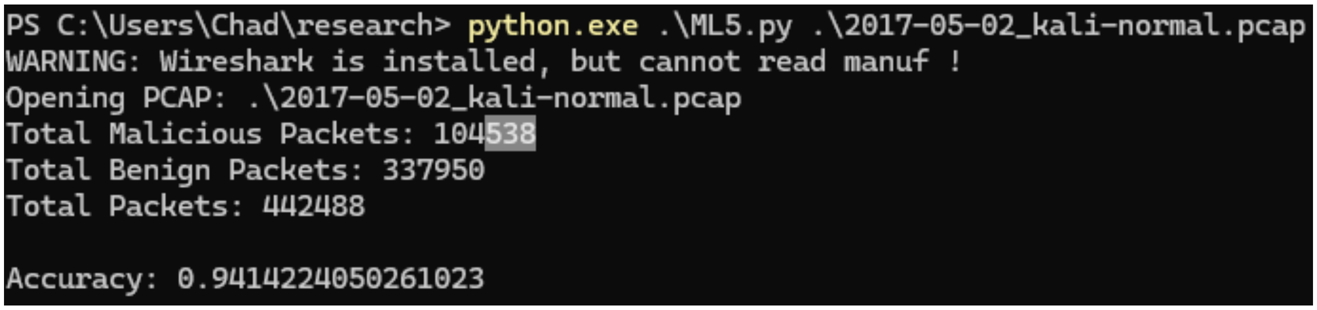
Between the three tests, memory consumption remained consistent for the benign dataset. Since the resource utilization remained the same for CPU and memory across all three tests, the size of the dataset is the sole factor to reduce the amount of system resources required for the machine learning model.

4.5. Test 4

The final test changes the dataset between Tests 3 and 4. The conditions to label a dataset will remain the same between Test 3 and Test 4 to validate that the dataset size solely affects the system's resource utilization. Since the dataset changes with Test 4, the false positives and true negatives results will differ between Tests 1, 2, 3, and 4.

4.5.1. Evaluating False Positive and True Negative Rates

The machine learning model identified 104,538 malicious packets and 337,950 benign packets and had an accuracy of 94.1%. The primary purpose of Test 4 was to determine the primary factor for system resource utilization and not necessarily the results of false positives and true negatives.

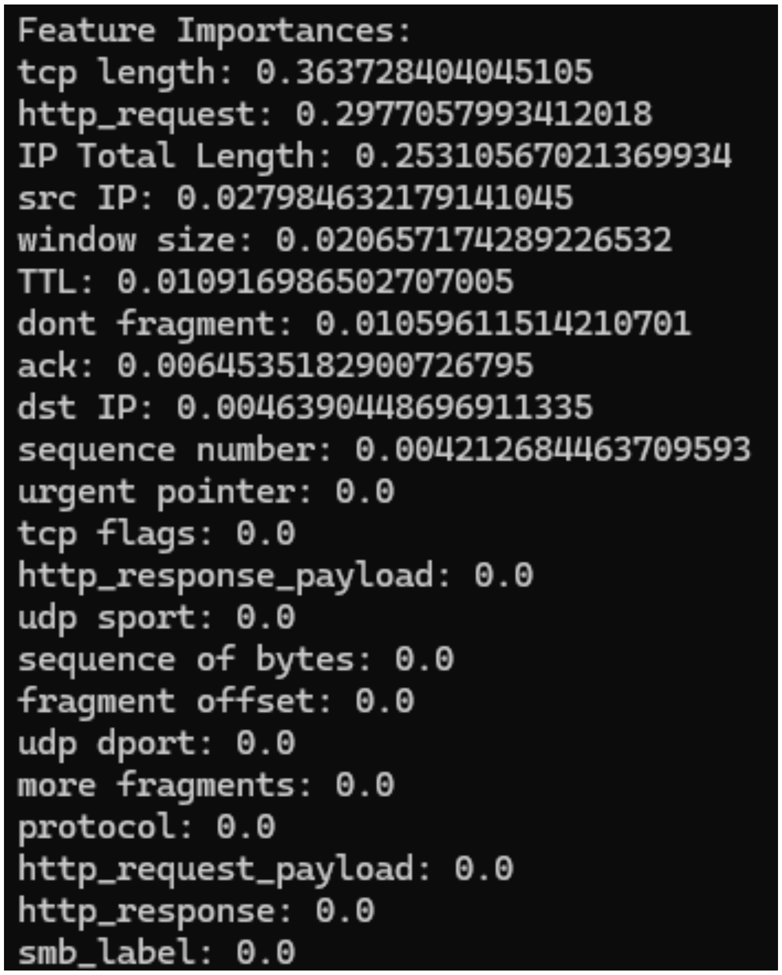


```
PS C:\Users\Chad\research> python.exe .\ML5.py .\2017-05-02_kali-normal.pcap
WARNING: Wireshark is installed, but cannot read manuf !
Opening PCAP: .\2017-05-02_kali-normal.pcap
Total Malicious Packets: 104538
Total Benign Packets: 337950
Total Packets: 442488

Accuracy: 0.9414224050261023
```

Figure 13: Labelled Malicious and Benign Packets for Test 4

Since the dataset changed, the machine learning model weighed the importance of features differently from Tests 1, 2, and 3. The most important feature of the new data set was the TCP Length, which assisted with identifying packets as benign or malicious.

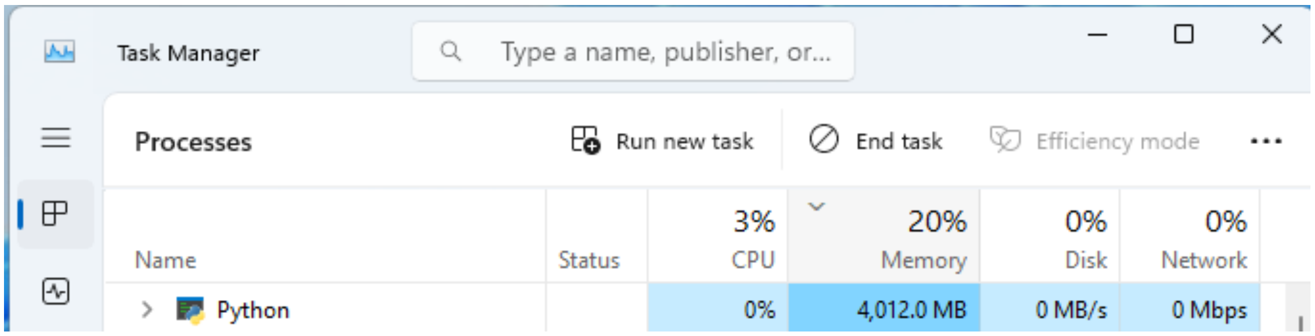


```
Feature Importances:
tcp length: 0.363728404045105
http_request: 0.2977057993412018
IP Total Length: 0.25310567021369934
src IP: 0.027984632179141045
window size: 0.020657174289226532
TTL: 0.010916986502707005
dont fragment: 0.01059611514210701
ack: 0.0064535182900726795
dst IP: 0.0046390448696911335
sequence number: 0.004212684463709593
urgent pointer: 0.0
tcp flags: 0.0
http_response_payload: 0.0
udp sport: 0.0
sequence of bytes: 0.0
fragment offset: 0.0
udp dport: 0.0
more fragments: 0.0
protocol: 0.0
http_request_payload: 0.0
http_response: 0.0
smb_label: 0.0
```

Figure 14: Feature Importance for Test 4

4.5.2. Evaluating System Resource Utilization

During Test 4, Python.exe consumed approximately 26.5% of the CPU. The dataset size significantly affected Python.exe's memory utilization. In Test 4, Python.exe only consumed 4,012.1 MB of memory. The smaller dataset required less memory space for the machine learning model to analyze and label malicious and benign datasets.



Name	Status	3% CPU	20% Memory	0% Disk	0% Network
Python		0%	4,012.0 MB	0 MB/s	0 Mbps

Figure 15: Memory Utilization After Test 4

The system's resource allocation will affect the efficiency and performance of the machine learning model in generating Windows Defender Firewall rules. The dataset's size will affect the system's memory requirement. The efficiency of the machine learning program may affect the percentage of CPU utilization on the system.

4.6. Results

Tests 1, 2, and 3 relied on the same dataset for each test. The data labeling methodology was the variable that changed between the first three tests to reduce the number of malicious packets and to determine the effects on the system's resources. Test 4 used a different dataset to determine if the size of the dataset affected the system's resources. Test 4 determined that the size of the dataset affects the amount of memory that Python needs to extract and label the dataset. The dataset size did not affect the system's CPU utilization. Table 1 visualizes the results for all four tests

Test #	CPU	Memory	Dataset Size	Malicious Packets	Benign Packets
Test 1	26.3%	22,918.5 MB	1,627 MB	730,448	1,886,542
Test 2	26.3%	22,919.4 MB	1,627 MB	678,386	1,938,604
Test 3	24%-27%	22,918.5 MB	1,627 MB	678,386	1,938,604
Test 4	26.5%	4,012.1 MB	311 MB	104,538	337,950

Table 1: Test Results

The results may influence future research into PCAP analysis in the Python script developed for this project. The study advocates for the integration of advanced machine learning and automation capabilities into existing security operations, urging security practitioners to harness these technologies to fortify defenses, anticipate threats, and respond with speed and precision to cyber-based incidents.

5. Recommendations for Security Practitioners

Machine Learning is an effective solution to automate network analysis, but the model needs appropriately labeled datasets for the environment. ML still requires research into practical methods for implementing and operating solutions that reduce system resource utilization, false positive, false negative detection rates, and provide an effective user interface. To detect and prevent malicious activity, security practitioners may embrace machine learning to minimize manual analysis of network traffic, packets, and PCAP files.

5.1. Recommendations for ML Research

Testing and training a supervised ML model to analyze malicious network traffic did not require large datasets. The ML research is easily replicated in various environments using various programming languages and associated libraries. Researching the implications, operations, and utilization of machine learning models is necessary to provide security practitioners with effective capabilities to detect and prevent malicious organizational activities.

5.2. Implications for Future Research

Although the ML model predicted a high degree of false positives and false negatives, better methods for the research exist:

1. **Data Labelling:** Future research should test labeling datasets based on authorized and approved network traffic instead of focusing on malicious indicators of compromise.
2. **Coding Practices:** Future research should optimize their scripts and code to reduce the amount of system resources required to process and analyze PCAP datasets. Optimized code could enable endpoints to host machine learning capabilities as detection and prevention solutions.
3. **Automate ML Analysis:** Future research should develop scripts that can act as a network tap directly on an endpoint’s Network Interface Card (NIC) to analyze live packets across the wire to simulate an inline

Intrusion Prevention System (IPS) or to modify an endpoint's firewall in real-time to achieve a high degree of security automation.

4. **User Interface:** Future research could develop a user interface for an ML model to enable security practitioners to study machine learning concepts without requiring programming experience.

The following immediate research could expand the concepts by developing a virtualized laboratory environment that provides standard network services to endpoints, defining security policies that authorize specific network traffic for the environment, and then incorporating the authorized policies as rules to label datasets to enable the ML model to learn from the authorized traffic. The method to train an ML model from authorized policies instead of malicious indicators of compromise could allow the model to highlight misconfigurations and malicious network traffic at the endpoint device.

6. Conclusion

The research aimed to leverage a supervised machine learning model built in Python to extract specific features from PCAP datasets, label them based on malicious indicators of compromise, and automatically deploy Windows Defender Firewall rules for isolated networks. The study found that implementing such models depends on system resources, accurate dataset labeling, and dataset size. Further research is needed to develop an effective model for automated firewall rule deployment. The model must be efficient and capable of handling normal, malicious, and background noise data.

Future research should focus on developing a lab environment, applying effective network traffic policies, and using machine learning to analyze deviations from authorized traffic. Properly labeled datasets are crucial for deploying effective firewall rules. Vendors should include user interface features that allow organizations to define authorized activities for the datasets analyzed by ML models.

The DoD's Zero Trust Reference Architecture aims to utilize ML and AI to secure defense networks. Understanding ML at all organizational levels is essential for effective implementation without hindering authorized access. ML can reduce costs associated with manual network traffic analysis, allowing security practitioners to focus on advanced threats.

Commercial security vendors are developing ML capabilities as software-as-a-service, requiring internet connectivity. Organizations with isolated networks need high-fidelity security controls that leverage custom ML models and automation to mitigate risks of lateral movement and unauthorized data access in real time to prevent compromise before, during, and after an incident.

References

Computer Security Resource Center (CSRC), Rose, S. W., Borchert, O., Mitchell, S., & Connelly, S., *NIST Special Publication 800-207: Zero trust Architecture* 9–10 (2020). Gaithersburg, MD; U.S. Dept. of Commerce, National Institute of Standards and Technology (NIST).

Defense Information Systems Agency (DISA), & National Security Agency (NSA), *Department of Defense Zero Trust Reference Architecture* (2022). Chief Information Officer U.S. Department of Defense. Retrieved February 14, 2024, from https://dodcio.defense.gov/Portals/0/Documents/Library/(U)ZT_RA_v2.0(U)_Sep22.pdf.

Garcia, S. (2017, May 2). Malware Capture Facility Project, *CTU-Normal-21*. Prague, Czech Republic; https://stratosphereips.org.

Garcia, S. (2017, May 2). Malware Capture Facility Project, *CTU-Normal-22*. Prague, Czech Republic; https://stratosphereips.org.

Garcia, S. (2017, May 2). Malware Capture Facility Project, *Normal Captures*. Prague, Czech Republic; https://stratosphereips.org.

Iandoli, M., Perez, J., Kothari, V., & Zhang, J. (2021, May 6). *Generating Firewall Rules Through a User Interface and Machine Learning* (thesis). Worcester Polytechnic Institute. Retrieved February 14, 2024, from https://digital.wpi.edu/concern/student_works/4q77fv116?locale=zh.

Nauni, M. (2021, October 13). *Basic Machine Learning: Network Anomaly Detection*. Cloud Whisperer. February 17, 2024, https://mayanknauni.com/?p=4392

Office of the Chief Information Officer, *Department of Defense Zero Trust Overlays* (2024). Retrieved June 15, 2024, from https://dodcio.defense.gov/Portals/0/Documents/Library/ZeroTrustOverlays-2024Feb.pdf.

Verbruggen, R. (2014). *Creating Firewall Rules with Machine Learning Techniques* (thesis). Radboud University. Retrieved February 14, 2024, from https://www.cs.ru.nl/masters-theses/2014/R_Verbruggen___Creating_firewall_rules_with_machine_learning_techniques.pdf.

xgboost developers. (2022). XGBoost documentation. XGBoost Documentation - xgboost 2.1.0 documentation. https://xgboost.readthedocs.io/en/stable