

Introduction

In our Advanced Networking final project, we explored and compared two different approaches to intrusion detection: a classic rule-based system (Snort) and our own AI-based Model (AIIDS). Our goal was to understand how well each system could detect and handle different types of cyberattacks. Our team used the CIC-IDS2017 dataset, which provides realistic network traffic in both PCAP (packet capture) format for Snort and labeled CSV flow files for training and testing our AI model. We built our testing environment on a Linux virtual machine, where we installed Snort and tools like tcpreplay and Wireshark to simulate and inspect network traffic. Snort allowed us to write or use predefined detection rules, such as identifying brute-force attacks by tracking repeated login attempts. In contrast, our AI-based IDS used machine learning techniques to analyze patterns in the data and classify traffic as either normal or malicious without relying on manually written rules. Throughout the project, we aimed to assess which system was more effective in detecting threats, how easy each one was to deploy, and how accurate they were when analyzing both clean and attack traffic from the CIC-IDS2017 dataset.

Project Overview and Objectives

Intrusion Detection Systems (IDS) play a crucial role in protecting computer networks against malicious activities. These systems are essential components in the defense infrastructure of businesses, governments, and other organizations. Traditionally, IDS tools such as Snort have relied on hardcoded rules to detect and flag known cyber threats. While effective in identifying predefined attacks, these rule-based systems are often unable to detect novel or obfuscated threats. Given the increasing sophistication of cyberattacks, there is growing interest in evaluating how artificial intelligence (AI), particularly machine learning (ML)-based intrusion detection systems, can complement or even replace legacy systems like Snort.

The goal of this project is to evaluate the performance, ease of use, and effectiveness of a traditional rule-based IDS (Snort) in comparison with an ML-based IDS approach using the CIC-IDS2017 dataset.

Although we initially intended to use the Stratosphere IPS Suite, due to technical difficulties, we opted

for a flow-based ML evaluation using labeled CSV data from the same dataset. Our objective is to determine whether AI-enhanced IDS systems offer a tangible improvement in detection capabilities and operational efficiency.

Methodology

To perform our analysis, we used a combination of open-source tools and simulated network traffic environments. The dataset employed was CIC-IDS2017, a comprehensive and widely accepted dataset that contains a mix of benign and malicious traffic, including brute force attacks, DoS/DDoS, infiltration, and botnet traffic.

Snort was installed on a virtual machine running Ubuntu Linux. Traffic from the CIC-IDS2017 dataset was replayed using tcpreplay, while Scapy was used to simulate specific attacks to trigger Snort alerts. To analyze packet-level behavior, we used Wireshark, and all configurations and alerts were monitored using system logs.

We configured Snort with several custom detection rules, including:

- **Detection of SSH brute force attacks:**

```
alert tcp any any -> any 22 (msg:"SSH Brute-force detected"; flags:S;  
threshold:type threshold, track by_src, count 60, seconds 60; sid:1000004;  
rev:1;)
```

- **Detection of unauthorized MySQL root login attempts:**

```
alert tcp any any -> 192.168.0.102 3306 (msg:"Unauthorized MySQL root login  
attempt"; content:"root"; sid:1000002; rev:1;)
```

- **Detection of unencrypted MySQL password usage:**

```
alert tcp any any -> 192.168.0.102 3306 (msg:"Unencrypted MySQL password  
detected"; content:"password="; sid:1000003; rev:1;)
```

```
ubuntu@ubuntu-VirtualBox: ~  
GNU nano 7.2 /etc/snort/rules/local.rules  
$Id: local.rules,v 1.11 2004/07/23 20:15:44 bmc Exp $  
# .....  
# LOCAL RULES  
# .....  
# This file intentionally does not come with signatures. Put your local  
# additions here.  
  
# To Detect SSH Brute-Force Attacks  
alert tcp any any -> any 22 (msg:"Possible SSH brute-force attack"; flags:S; threshold:type threshold, track by_src, count 5, seconds 60; sid:1000001; rev:1;)  
#Detect Unauthorized MySQL Root Login Attempts  
alert tcp any any -> 192.168.0.102 3306 (msg:"Unauthorized MySQL root login attempt"; content:"root"; sid:1000002; rev:1;)  
#Detect Unencrypted MySQL Passwords  
alert tcp any any -> 192.168.0.102 3306 (msg:"Unencrypted MySQL password detected"; content:"password="; sid:1000003; rev:1;)  
#Snort Rule for SSH Brute Force Detection  
alert tcp any any -> any 22 (msg:"SSH Brute-force detected"; flags:S; threshold:type threshold, track by_src, count 60, seconds 60; sid:1000004; rev:1;)  
  
● snort.service - LSB: Lightweight network intrusion detection system  
   Loaded: loaded (/etc/init.d/snort; generated)  
   Active: active (running) since Mon 2025-03-03 07:37:39 EST; 15s ago  
     Docs: man:systemd-sysv-generator(8)  
  Process: 14546 ExecStart=/etc/init.d/snort start (code=exited, status=0/SUCCESS)  
    Tasks: 4 (limit: 4608)  
   Memory: 159.0M (peak: 176.0M)  
      CPU: 1.384s  
   CGroup: /system.slice/snort.service  
           └─ 14567 /usr/sbin/snort -n 027 -D -d -l /var/log/snort -u snort -g snort --pid-path /run/snort/ -c /etc/snort/snort.conf -S "\HOME_NET=[192.168.0.0/16]" -i enp0s3  
             14582 /usr/sbin/snort -n 027 -D -d -l /var/log/snort -u snort -g snort --pid-path /run/snort/ -c /etc/snort/snort.conf -S "\HOME_NET=[192.168.0.0/16]" -i enp0s3  
  
Mar 03 07:37:39 ubuntu-VirtualBox snort[14582]: Preprocessor Object: SF_DCEP2C Version 1.0 <Build 3>  
Mar 03 07:37:39 ubuntu-VirtualBox snort[14582]: Preprocessor Object: SF_DNP3 Version 1.1 <Build 1>  
Mar 03 07:37:39 ubuntu-VirtualBox snort[14582]: Preprocessor Object: SF_SDF Version 1.1 <Build 1>  
Mar 03 07:37:39 ubuntu-VirtualBox snort[14582]: Preprocessor Object: SF_DNS Version 1.1 <Build 4>  
Mar 03 07:37:39 ubuntu-VirtualBox snort[14582]: Preprocessor Object: SF_REPUTATION Version 1.1 <Build 1>  
Mar 03 07:37:39 ubuntu-VirtualBox snort[14582]: Preprocessor Object: SF_SMTP Version 1.1 <Build 9>  
Mar 03 07:37:39 ubuntu-VirtualBox snort[14582]: Preprocessor Object: SF_POP Version 1.0 <Build 1>  
Mar 03 07:37:39 ubuntu-VirtualBox snort[14582]: Commencing packet processing (pid=14582)  
Mar 03 07:37:39 ubuntu-VirtualBox snort[14546]: ...done.  
Mar 03 07:37:39 ubuntu-VirtualBox systemd[1]: Started snort.service - LSB: Lightweight network intrusion detection system.  
~  
~  
~  
~  
~  
~  
~  
~  
~  
~
```

Using Scapy, we generated traffic to trigger these alerts. Sample scripts used include:

- For SSH brute force:

```
from scapy.all import *  
for i in range(65):  
    send(IP(dst="192.168.0.102")/TCP(dport=22, flags="S"))
```

[illegible]

```
ubuntu@ubuntu-VirtualBox:~$ sudo cat /var/log/snort/snort.alert
4g?vjK?v
?vG?vP4g?v~-~?
?vf4g?v~?#?
?vf4g?v:~s?
?vf4g?v|T?
?vf4g?v?tBA
?vf4g?v?t?
?vfg?v?v?v
?vf4? ?g?vS.?
?vf4
g?v*?
g?vTrash?
?vf4? ?g?v]??
?vf4? ?g?v?vBA
g?v?v?
?vf4g?v?v
?vf4g?vvy? ?
?vf4g?v?v?v
?vf4g?v?vz?v
?vf4g?v?vBA
?vf4g?v?v?v
?vf4g?v?v?v?v
?vf4g?v?v"?
?vf4g?v?vl`?
?vf4g?v?v;?
?vf4g?v?4?vBA
?vf4g?v?4?v
?vf4g?v?"?
?vf4?v <?
?vf4g?v ?t?
?vf4g?v
2?
?vf4g?v
?vBA
?vf4g?v
```

File Edit View Go Capture Analyze Statistics Telephony Wireless Tools Help
tempus

Time Filter: tcp
100%

No.	Time	Source	Destination	Protocol	Length	Info
166	1790.1533191	192.168.48.1	239.255.255.250	SSDP	179	M-S-SEARCH > HTTP/1.1
167	1984.054076	192.168.48.1	224.0.0.251	MDNS	409	Standard query response 0x0000 PTR Yama..._dsvoc.tcp.local SRV 0 0 6880 Yama... Local T...
168	1984.054080	192.168.48.1	224.0.0.251	MDNS	409	Standard query response 0x0000 PTR Yama..._dsvoc.tcp.local SRV 0 0 6880 Yama... Local T...
169	1984.054086	192.168.48.1	239.255.255.250	SSDP	179	M-S-SEARCH > HTTP/1.1
170	1984.054094	192.168.48.1	239.255.255.250	SSDP	179	M-S-SEARCH > HTTP/1.1
171	1984.054104	192.168.48.3	192.168.48.3	DHCP	437	DHCP Request - Transaction ID 0x1d228c2c
172	1984.0757874	192.168.48.2	192.168.48.3	DHCP	990	DHCP ACK Transaction ID 0x1d228c2c
173	1984.075794	192.168.48.2	239.255.255.250	M-S-SEARCH	179	M-S-SEARCH > HTTP/1.1
174	1984.2967776	192.168.48.1	239.255.255.250	M-S-SEARCH	179	M-S-SEARCH > HTTP/1.1
175	1989.3774707	192.168.48.1	239.255.255.250	M-S-SEARCH	179	M-S-SEARCH > HTTP/1.1
176	2151.2684846	192.168.48.1	239.255.255.250	SSDP	179	M-S-SEARCH > HTTP/1.1
177	2151.268486	192.168.48.1	239.255.255.250	SSDP	179	M-S-SEARCH > HTTP/1.1
178	2151.6745816	192.168.48.1	239.255.255.250	SSDP	179	M-S-SEARCH > HTTP/1.1
180	2151.680580	192.168.48.1	239.255.255.250	SSDP	179	M-S-SEARCH > HTTP/1.1
181	2151.543747	f686::c6e6:7375:f6	ff02::1	MDNS	101	Standard query 0x0000 PTR _nmea-0183_.tcp.local, "Qm" question
182	2151.523905	192.168.48.3	224.0.0.251	MDNS	81	Standard query 0x0000 PTR _nmea-0183_.tcp.local, "Qm" question
183	2151.51375	192.168.48.3	ff02::1	MDNS	101	Standard query 0x0000 PTR _nmea-0183_.tcp.local, "Qm" question
184	2151.5180574	192.168.48.3	224.0.0.251	MDNS	81	Standard query 0x0000 PTR _nmea-0183_.tcp.local, "Qm" question
185	2151.5174	192.168.48.3	ff02::1	MDNS	101	Standard query 0x0000 PTR _nmea-0183_.tcp.local, "Qm" question
186	2151.510889	192.168.48.3	224.0.0.251	MDNS	81	Standard query 0x0000 PTR _nmea-0183_.tcp.local, "Qm" question
187	2151.523377	f686::c6e6:7375:f6	ff02::1	MDNS	101	Standard query 0x0000 PTR _nmea-0183_.tcp.local, "Qm" question
188	2151.520250	192.168.48.3	ff02::1	MDNS	81	Standard query 0x0000 PTR _nmea-0183_.tcp.local, "Qm" question
189	2186.531764	f686::c6e6:7375:f6	ff02::1	MDNS	101	Standard query 0x0000 PTR _nmea-0183_.tcp.local, "Qm" question
190	2186.532075	192.168.48.3	224.0.0.251	MDNS	81	Standard query 0x0000 PTR _nmea-0183_.tcp.local, "Qm" question
191	2202.5349213	f686::c6e6:7375:f6	ff02::1	MDNS	101	Standard query 0x0000 PTR _nmea-0183_.tcp.local, "Qm" question
192	2202.5362403	192.168.48.3	224.0.0.251	MDNS	81	Standard query 0x0000 PTR _nmea-0183_.tcp.local, "Qm" question
193	2204.060623	192.168.48.3	192.168.48.2	DHCP	337	DHCP Request - Transaction ID 0xb6ac1f
194	2204.072977	192.168.48.2	192.168.48.3	DHCP	500	DHCP ACK Transaction ID 0xb6ac1f
195	2209.000434	PCSSystemtec.48:15:	PCSSystemtec.48:15:	ARP	42	who has 192.168.48.2? Tell 192.168.48.3
196	2209.0004815	PCSSystemtec.48:15:	PCSSystemtec.48:15:	ARP	68	192.168.48.2 is at 08:00:27:71:a0:0e
197	2209.869446	f686::c6e6:7375:f6	ff02::1	ICMPv6	118	Multicast Listener Report Message v2
198	2209.869466	f686::c6e6:7375:f6	ff02::1	ICMPv6	118	Multicast Listener Report Message v2
199	2504.066334	PCSSystemtec.48:15:	Broadcast	ARP	42	who has 192.168.48.2? Tell 192.168.48.3
200	2504.066334	PCSSystemtec.48:15:	Broadcast	ARP	68	192.168.48.2 is at 08:00:27:71:a0:0e
201	2504.066376	192.168.48.3	192.168.48.2	DHCP	337	DHCP Request - Transaction ID 0xf49cc50
202	2504.0773407	192.168.48.2	192.168.48.3	DHCP	500	DHCP ACK Transaction 0xf49cc50
203	4424.0491131	f686::c6e6:7375:f6	ff02::1	MDNS	167	Standard query 0x0000 PTR _ipps_.tcp.local, "Qm" question
204	4424.037448	192.168.48.3	224.0.0.251	MDNS	87	Standard query 0x0000 PTR _

- **For unauthorized MySQL root login:**

```
from scapy.all import *
for i in range(65):
    send(IP(dst="192.168.0.102")/TCP(dport=3306)/Raw(load="root"))
```

[illegible]

- **For unencrypted password detection:**

```
from scapy.all import *
for i in range(65):
send(IP(dst="192.168.0.102")/TCP(dport=3306)/Raw(load="password=12345"))
```


We have also begun analyzing the labeled CSV files from CIC-IDS2017 to establish a comparative benchmark for ML-based detection systems. This work is ongoing and will contribute to our final comparison in terms of detection rates, false positives, and scalability.

Snort (Rule-Based IDS)

Pros:

- **Established and Reliable:** Snort has been around for many years and is trusted by professionals in the cybersecurity field. It's a well-tested tool used for both basic and complex intrusion detection tasks.
- **Signature-Based Detection:** It identifies threats using predefined rules or "signatures." This makes it very effective at detecting well-known types of attacks, such as brute-force logins, malware infections, and network scans.
- **Real-Time Analysis:** Snort is capable of monitoring and analyzing network traffic as it happens, which allows it to detect live threats in real-time, especially useful for active, high-traffic networks.
- **Flexible Rule Creation:** Users can write custom detection rules tailored to specific threats in their environment, giving it flexibility in how threats are identified.
- **Free and Open-Source:** Being open-source means Snort is completely free to use and can be modified or improved by anyone in the community.

Cons:

- **Only Detects Known Threats:** Snort can only identify attacks that match existing rules. If an attacker uses a new method not yet defined in the rule set, Snort won't detect it unless the rule is manually added or updated.
- **Prone to False Positives:** If the rules are not precisely tuned, Snort might incorrectly flag harmless traffic as malicious, which can lead to unnecessary alerts and confusion.

- **Ongoing Rule Maintenance:** The rule set must be frequently updated to stay current with emerging threats. Managing this, especially in large or complex environments, can become time-consuming and challenging.
- **Performance Limitations:** As the network traffic increases, Snort has to check every packet against its growing list of rules. This can slow it down and affect overall performance.
- **No Learning Ability:** Snort does not adapt or learn on its own. It relies entirely on rules, so it cannot detect brand-new or evolving attack patterns without manual input.

Challenges:

- **Maintaining Accurate Rules:** It's difficult to keep Snort's rule set both comprehensive and accurate, especially since attackers constantly change tactics.
- **Performance Bottlenecks:** With a large number of rules and high network traffic, Snort may slow down or miss packets.
- **Complex Configuration:** Setting up and fine-tuning Snort properly requires deep knowledge of network behavior and attack patterns, which can be hard for beginners.

AI-IDS (AI-Based Intrusion Detection System)

Pros:

- **Can Detect New Attacks:** Unlike Snort, AI-IDS does not depend on predefined rules. It uses machine learning to analyze network behavior, allowing it to detect attacks that have never been seen before.
- **Self-Improving:** AI-IDS can learn from new data over time, meaning it gets better at recognizing attack patterns as it sees more traffic.
- **Fewer False Alarms (When Trained Well):** If properly trained on good quality data, AI-IDS can more accurately tell the difference between normal and malicious traffic, reducing the chances of false positives.

- **Good for Large Networks:** Since it relies on learned models rather than matching each packet to rules, AI-IDS can handle bigger datasets and traffic volumes more efficiently.
- **Automated Threat Detection:** Once the model is trained, the system can automatically detect suspicious behavior without needing someone to constantly update or fine-tune rules.

Cons:

- **Needs High-Quality Data:** The success of AI-IDS heavily depends on the dataset used to train it. If the training data is incomplete or biased, the system might give inaccurate results.
- **Complex to Build:** Designing, training, and deploying a machine learning model requires expertise in data science and cybersecurity, as well as powerful computing resources.
- **Hard to Interpret:** Many AI models, especially deep learning ones, don't offer clear explanations of how they reach a decision. This "black box" nature makes it harder to understand or trust certain results.
- **Possible Missed Threats (False Negatives):** If the model isn't trained well, it might fail to catch some attacks or mistakenly mark them as safe. Resource Intensive: Training AI models takes significant computing power, time, and memory, especially when working with large datasets like CIC-IDS2017.

```
(env) salman@Salmans-MacBook-Pro-2 network_anomaly_detector_updated % sudo pip install .
Password:
WARNING: The directory '/Users/salman/Library/Caches/pip' or its parent directory is not owned or is not writable by the current user. The cache has been disabled. Check the permissions and owner of that directory. If executing pip with sudo, you should use sudo's -H flag
Processing /Users/salman/Documents/Learning/UWindsor/COMP4680/network_anomaly_detector_updated
Installing build dependencies ... done
Getting requirements to build wheel ... done
Preparing metadata (pyproject.toml) ... done
Requirement already satisfied: numpy in /Users/salman/Documents/Learning/UWindsor/COMP4680/NTLFlowLyzer/env/lib/python3.13/site-packages (from network_anomaly_detector==0.1) (2.2.5)
Collecting pandas (from network_anomaly_detector==0.1)
  Downloading pandas-2.2.3-cp313-cp313-macosx_11_0_arm64.whl.metadata (89 kB)
Collecting scikit-learn (from network_anomaly_detector==0.1)
  Downloading scikit_learn-1.6.1-cp313-cp313-macosx_12_0_arm64.whl.metadata (31 kB)
Collecting joblib (from network_anomaly_detector==0.1)
  Downloading joblib-1.4.2-py3-none-any.whl.metadata (5.4 kB)
Collecting scipy (from network_anomaly_detector==0.1)
  Downloading scipy-2.6.1-py3-none-any.whl.metadata (5.6 kB)
Collecting python-dateutil>=2.8.2 (from pandas->network_anomaly_detector==0.1)
  Downloading python_dateutil-2.9.0.post0-py2.py3-none-any.whl.metadata (8.4 kB)
Collecting pytz>=2020.1 (from pandas->network_anomaly_detector==0.1)
  Downloading pytz-2025.2-py2.py3-none-any.whl.metadata (22 kB)
Collecting tzdata>=2022.7 (from pandas->network_anomaly_detector==0.1)
  Downloading tzdata-2025.2-py2.py3-none-any.whl.metadata (1.4 kB)
Collecting six>=1.5 (from python-dateutil>=2.8.2->pandas->network_anomaly_detector==0.1)
  Downloading six-1.17.0-py2.py3-none-any.whl.metadata (1.7 kB)
Requirement already satisfied: scipy>=1.6.0 in /Users/salman/Documents/Learning/UWindsor/COMP4680/NTLFlowLyzer/env/lib/python3.13/site-packages (from scikit-learn->network_anomaly_detector==0.1) (1.15.2)
Collecting threadpoolctl>=3.1.0 (from scikit-learn->network_anomaly_detector==0.1)
  Downloading threadpoolctl-3.6.0-py3-none-any.whl.metadata (13 kB)
Downloaded joblib-1.4.2-py3-none-any.whl (301 kB)
Downloaded pandas-2.2.3-cp313-cp313-macosx_11_0_arm64.whl (11.3 MB)
11.3/11.3 MB 23.0 MB/s eta 0:00:00
Downloaded python_dateutil-2.9.0.post0-py2.py3-none-any.whl (229 kB)
Downloaded pytz-2025.2-py2.py3-none-any.whl (509 kB)
Downloaded six-1.17.0-py2.py3-none-any.whl (11 kB)
Downloaded tzdata-2025.2-py2.py3-none-any.whl (347 kB)
Downloaded scipy-2.6.1-py3-none-any.whl (2.4 MB)
2.4/2.4 MB 23.2 MB/s eta 0:00:00
Downloaded scikit_learn-1.6.1-cp313-cp313-macosx_12_0_arm64.whl (11.1 MB)
```

```
(env) salman@Salmans-MacBook-Pro-2 network_anomaly_detector_updated % sudo scan-live-traffic
[INFO] Sniffing for 10 seconds on interface: en0
[INFO] Done sniffing.
[INFO] Captured packets saved to captured_traffic.pcap
[RESULT] Live Traffic Prediction Results:
Source IP Destination IP Source Port Destination Port Protocol label
0 149.154.175.57 10.72.101.247 443 58058 TCP BENIGN
1 10.72.101.247 149.154.175.57 58058 443 TCP BENIGN
2 10.72.101.247 34.120.195.249 58105 443 TCP BENIGN
3 34.120.195.249 10.72.101.247 443 58105 TCP BENIGN
4 10.72.101.247 35.186.224.9 58106 443 TCP BENIGN
5 35.186.224.9 10.72.101.247 443 58106 TCP BENIGN
6 10.72.101.247 140.82.113.22 58882 443 TCP BENIGN
7 140.82.113.22 10.72.101.247 443 58882 TCP BENIGN
8 10.72.101.247 35.186.224.30 58053 443 TCP BENIGN
9 35.186.224.30 10.72.101.247 443 58053 TCP BENIGN
10 10.72.101.247 35.186.224.24 58086 443 TCP BENIGN
11 35.186.224.24 10.72.101.247 443 58086 TCP BENIGN
12 10.72.101.247 4.249.131.160 58887 443 TCP BENIGN
13 4.249.131.160 10.72.101.247 443 58887 TCP BENIGN
14 140.82.112.25 10.72.101.247 443 58047 TCP BENIGN
15 10.72.101.247 140.82.112.25 58047 443 TCP BENIGN
16 10.72.101.247 31.13.80.53 58048 443 TCP BENIGN
17 31.13.80.53 10.72.101.247 443 58048 TCP BENIGN
18 10.72.101.247 142.250.31.108 58897 585 TCP BENIGN
19 10.72.101.247 35.186.224.30 56168 443 TCP BENIGN
20 35.186.224.30 10.72.101.247 443 56168 TCP BENIGN
21 10.72.101.247 142.250.31.109 58898 585 TCP BENIGN
(env) salman@Salmans-MacBook-Pro-2 network_anomaly_detector_updated %
```

[INFO] Evaluation Report:

	precision	recall	f1-score	support
BENIGN	1.00	1.00	1.00	454620
MALICIOUS	0.99	1.00	1.00	111529
accuracy			1.00	566149
macro avg	1.00	1.00	1.00	566149
weighted avg	1.00	1.00	1.00	566149

[INFO] Model saved to models/xgboost_model.pkl

Challenges:

- **Extracting valuable data from PCAP captures:** We needed flow-based data to train our model. We found an application called CICFlowMeter that helped us convert the PCAP files into usable CSV format. However, we faced errors using CICFlowMeter at times, especially with large PCAPs, and had to find workarounds.
- **Choosing an appropriate machine learning model:** We researched different ML models to see which one had better accuracy. At first, we tried Isolation Forest, but later switched to XGBoost because it gave us better results.
- **Coding and debugging:** While building the model, we ran into several coding issues. We had to keep debugging and fixing errors to make sure the model worked properly.
- **Displaying only the relevant information in the results:** The output sometimes had too much data, so we had to clean it up and only show the important stuff like attack type, traffic source, and timestamps.
- **Extracting relevant features from PCAP files:** During training, we used CICFlowMeter to extract features easily. But during testing and execution, we couldn't use CICFlowMeter, so we had to manually extract the needed features from raw PCAPs.
- **Long training time for the model:** Training the AI model took a long time. We didn't know exactly when it would finish, so we had to wait and keep checking.
- **Finding a suitable dataset:** First we needed to find a dataset. We looked at many different sources but decided to go with the CIC dataset. The problem with that is that it included a ton of features (78 to be precise) and we were debating whether to look for another dataset that had less features to easiness.
- **Large dataset size:** We decided to go with the CIC dataset (almost 50GB of PCAP data that got extracted into CSV files), which was quite heavy and required time and space to process effectively.

	Destination	Port	Flow Duration	Total Fwd Packets	Total Backward Packets	...	Idle Std	Idle Max	Idle Min	label
0		443	0.026381	2	0	...	0	0	0	BENIGN
1		62795	0.000000	1	0	...	0	0	0	BENIGN
2		62821	1.129716	2	0	...	0	0	0	BENIGN
3		443	1.129283	2	0	...	0	0	0	BENIGN
4		57621	0.000000	1	0	...	0	0	0	BENIGN
5		0	0.000544	2	0	...	0	0	0	BENIGN
6		443	0.290074	4	0	...	0	0	0	BENIGN
7		63782	0.255747	3	0	...	0	0	0	BENIGN
8		443	0.498975	17	0	...	0	0	0	BENIGN
9		63816	0.470386	16	0	...	0	0	0	BENIGN
10		443	0.034640	3	0	...	0	0	0	BENIGN
11		63817	0.031881	2	0	...	0	0	0	BENIGN
[12 rows x 79 columns]										

The screenshot displays a VS Code workspace with a file explorer on the left showing files like `predict_live.py`, `entry_points.txt`, and `model_trainer.py`. The main editor area shows the `network_anomaly_detector.py` file with the following code:

```
def capture_live_traffic(output_pcap, duration=10, iface=None):
    packets = sniff(timeout=duration, ifaces=[iface])
    wrpcap(output_pcap, packets)
    print(f"[INFO] Traffic saved to {output_pcap}")

def extract_features_with_cicflowmeter(input_pcap, output_csv):
    """
    Use CICFlowMeter to extract features from a PCAP file.
    """
    print(
        f"[INFO] Extracting features from {input_pcap} using CICFlowMeter..."
    )
    command = f"cicflowmeter -f {input_pcap} -c {output_csv}"
    result = subprocess.run(command, shell=True,
                           capture_output=True, text=True)
    if result.returncode == 0:
        print(f"[INFO] Features saved to {output_csv}")
    else:
        print(f"[ERROR] Failed to extract features: {result.stderr}")

def predict_traffic(output_csv, model_path):
    """
    Predict traffic labels using the trained model.
    """
```

The terminal window at the bottom shows the execution of the script with the following error:

```
(3.12.0) salman@Salmans-MacBook-Pro-2 network_anomaly_detector_updated % cicflowmeter -f live_traffic.pcap -c extracted_features.csv
PermissionError: [Errno 13] Permission denied: 'extracted_features.csv'
Exception ignored in: <function CSVWriter._del_ at 0x1084bd4e0>
Traceback (most recent call last):
  File "/Users/salman/.pyenv/versions/3.12.0/lib/python3.12/site-packages/cicflowmeter/writer.py", line 27, in _del_
AttributeError: "CSVWriter" object has no attribute 'file'
```

A sidebar on the right contains a chat interface with a message about debugging the CSV file issue.

Challenges and Resolutions

The project encountered several technical challenges, the most significant being difficulties with deploying the Stratosphere IPS Suite due to environmental compatibility issues. This led us to pivot towards analyzing the CSV portion of the dataset using an ML approach.

Another challenge was ensuring proper configuration of Snort and its associated network interfaces, especially when using tcpreplay to simulate traffic. We resolved these issues through careful documentation, use of official guides, and frequent testing using tools like tail, grep, and Wireshark.

Writing effective and accurate rules for Snort also presented a challenge, especially in ensuring that alerts were triggered correctly and consistently. We addressed this by iteratively testing each rule with specific simulated traffic using Scapy and confirming results through multiple logs and packet captures.

Conclusion

The preliminary results of this project highlight the strengths and limitations of Snort as a rule-based IDS. Snort is effective in detecting well-defined and known threats, making it a reliable tool in environments with predictable traffic and known vulnerabilities. It is also well-documented, relatively lightweight, and easy to deploy.

However, Snort's primary limitation lies in its dependence on predefined rules. It cannot detect novel threats unless they have been explicitly defined, which poses a risk in modern, dynamic threat environments. Additionally, rule creation and maintenance require significant manual effort, making scalability a concern for large or fast-changing networks.

In contrast, machine learning-based IDS systems (which we are currently exploring using CIC-IDS2017 CSV data) aim to overcome these limitations by learning patterns from historical traffic and identifying anomalies without the need for manually defined rules. These systems promise improved detection for zero-day and previously unseen attacks, and they may offer reduced operational overhead in the long run.

We expect our final results to show that while rule-based systems like Snort remain useful, they are best suited for hybrid deployments where AI systems can provide broader, more adaptive detection capabilities.

References

- ASIS International. (2021, April). *Breaking down the pros and cons of AI in cybersecurity*. Security Management. <https://www.asisonline.org/security-management-magazine/monthly-issues/security-technology/archive/2021/april/breaking-down-the-pros-and-cons-of-ai-in-cybersecurity/>
- Bulajoul, W., James, A., & Pannu, M. (2017). Network intrusion detection systems in high-speed traffic in computer networks. *Future Generation Computer Systems*, 79, 1–9.
<https://doi.org/10.1016/j.future.2017.09.058>
- EC-Council. (2023). *Getting started with Snort IDS: Examining the pros and cons for network security*. EC-Council Cybersecurity Exchange. <https://www.eccouncil.org/cybersecurity-exchange/whitepaper/getting-started-snort-ids-network-security/>
- Shaik, D. A. (2024, January 12). *AI-based intrusion detection systems*. Insights2Techinfo.
<https://insights2techinfo.com/ai-based-intrusion-detection-systems/>
- University of New Brunswick, Canadian Institute for Cybersecurity (CIC). (2017). *CICIDS 2017 dataset*.
<https://www.unb.ca/cic/datasets/ids-2017.html>
- Scapy. (n.d.). *Scapy – Packet crafting for Python*. <https://scapy.net/>
- Snort. (n.d.). *Snort users manual and rule guides*. <https://www.snort.org/>
- Wireshark Foundation. (n.d.). *Wireshark: Go deep*. <https://www.wireshark.org/>