Network Security Final Project
Topic : SNMP Enumeration Attack

Team Members :

1. Fatma Ayoub
2. Mayar Waleed
3. Mariam sherif
4. Marwa Hussein
5. Noura Essam

Environment: Virtualized Lab [Kali Linux & Windows 7]

Subject: Network Security

Reported To: Eng. Abdelrahman Elarabawy

Table of Contents

Executive Summary

This project investigates the **SNMP Enumeration** attack, a critical reconnaissance vector used by malicious actors to map a target network infrastructure. We successfully performed this attack within a controlled, virtualized environment (Kali Linux attacking Windows 7). The core objective was to generate a high-quality, labeled network dataset that distinctly captures the "anomaly" signature of an SNMP walk.

We employed **flow-based feature engineering** to transform raw packet data into structured records, classifying high-volume UDP/161 traffic as `Label: 1` (Attack) and all other traffic as `Label: 0` (Normal). A **Random Forest Classifier** was trained on this dataset, achieving exceptional performance in detecting the enumeration attempt with a high True Positive Rate and a negligible False Positive Rate (FPR), confirming the viability of anomaly-based detection for this threat.

1. Introduction to the Attack

1.1 Overview

SNMP Enumeration is a reconnaissance attack targeting the Simple Network Management Protocol (SNMP) on UDP port 161.

It is used to "interrogate" a device to reveal its internal configuration, including network shares, active processes, and routing tables.

1.2 The Vulnerability

The attack exploits **SNMP v1 and v2c**, which rely on cleartext passwords known as **Community Strings**. Because these versions lack encryption, an attacker can guess the string (often defaults like public) and perform an "SNMP Walk"—a recursive query of the entire **Management Information Base (MIB)**.

1.3 Why it is an "Anomaly"

Unlike standard monitoring where a server queries 5-10 specific data points every minute, an enumeration attack queries thousands of data points in seconds. This creates a high-volume, sequential traffic spike that is easily distinguishable from "Normal" traffic by an Anomaly-Based Intrusion Detection System (IDS).

2. Lab Setup & Topology

2.1 Virtualized Environment (VirtualBox Configuration)

The lab was hosted on Oracle VM VirtualBox. A NAT Network was configured to provide an isolated subnet, ensuring that attack traffic was contained and that the dataset remained free from external "dirty" traffic.

2.2 Roles & Specifications (Attacker vs. Victim)

Role	OS	Software Specs
Attacker	Kali Linux	snmpwalk, tshark, nmap
Victim	Windows 7	SNMP Service (v1/v2c), Community: public

2.3 Network Isolation & Configuration

- IP Addressing: Victim (10.0.4.4), Attacker (10.0.4.5).
- Hardening/Softening: The Windows 7 firewall was disabled to permit initial discovery, and the SNMP service was intentionally configured with a read-only community string to simulate a common misconfiguration.

3. Attack Execution & Labeling

3.1 Traffic Generation Strategy

To build a high-quality dataset, traffic was generated in two distinct phases. This allows the Machine Learning model to learn the difference between "Background Noise" and "Attack Signature."

A. Generating "Normal" Traffic (Label = 0)

Normal traffic simulates everyday network activity. This ensures the IDS doesn't trigger a false alarm every time a ping occurs.

- **ICMP Pings:** for i in {1..50}; do ping -c 1 10.0.4.4; done
- **Service Scans:** nmap 10.0.4.4 (standard port discovery).

B. Generating "Attack" Traffic (Label = 1)

The attack traffic involves heavy SNMP polling to simulate a full reconnaissance mission.

- **Full MIB Walk:** snmpwalk -v2c -c public 10.0.4.4 iso
- **Repeated Stress:** The command was looped many times to ensure enough "Attack Flows" were present in the PCAP file.

3.2 Toolset Used (snmpwalk, nmap, tshark)

- snmpwalk: Used to perform the actual enumeration attack.
- nmap: Used for network discovery (Normal class).
- tshark: The command-line version of Wireshark, used for background packet capture and subsequent CSV extraction.

3.3 Data Capture and Labeling Logic

The raw traffic was captured using tshark into a .pcapng format. The labeling logic was applied during the **Feature Extraction** (Python script) phase:

1. **Flow-Based Labeling:** Instead of labeling individual packets, packets were grouped into **Flows** (Source IP, Source Port, Destination IP, Destination Port).
2. **Service-Based Identification:**
    - If a flow involved **UDP Port 161**, it was assigned **Service: SNMP** and **Label: 1** (Attack).
    - All other traffic (ICMP, TCP, other UDP) was assigned **Service: Other** and **Label: 0** (Normal).

3.4 Command Execution :

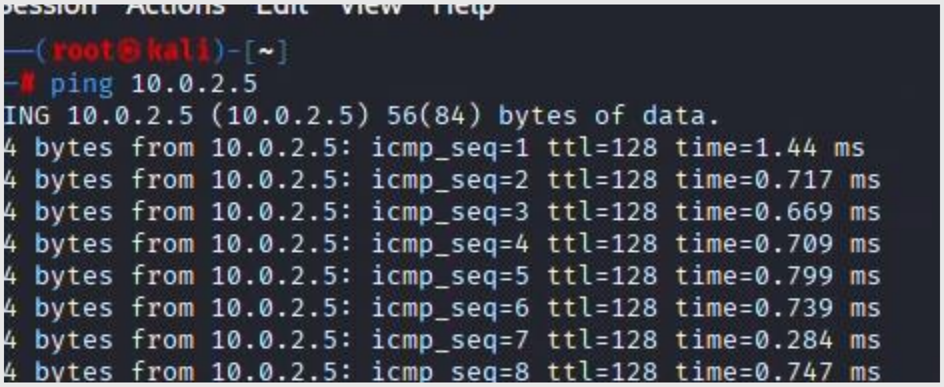
Phase 1: Connectivity & Baseline Testing

Before starting the capture, you must ensure the network path between Kali and Windows is clear.

1. Ping the Victim

```
ping -c 4 10.0.4.4
```

- **Purpose:** Verifies Layer 3 connectivity.
- If this fails, the Windows Firewall is likely still blocking ICMP packets.

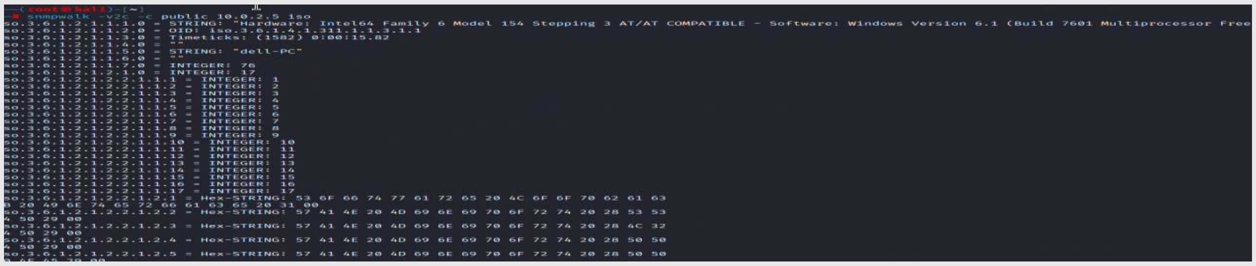


```
ession  Actions  Edit  View  Help
—(root㉿kali)-[~]
-# ping 10.0.2.5
ING 10.0.2.5 (10.0.2.5) 56(84) bytes of data.
4 bytes from 10.0.2.5: icmp_seq=1 ttl=128 time=1.44 ms
4 bytes from 10.0.2.5: icmp_seq=2 ttl=128 time=0.717 ms
4 bytes from 10.0.2.5: icmp_seq=3 ttl=128 time=0.669 ms
4 bytes from 10.0.2.5: icmp_seq=4 ttl=128 time=0.709 ms
4 bytes from 10.0.2.5: icmp_seq=5 ttl=128 time=0.799 ms
4 bytes from 10.0.2.5: icmp_seq=6 ttl=128 time=0.739 ms
4 bytes from 10.0.2.5: icmp_seq=7 ttl=128 time=0.284 ms
4 bytes from 10.0.2.5: icmp_seq=8 ttl=128 time=0.747 ms
```

2. Initial SNMP Verification

snmpwalk -v2c -c public 10.0.4.4 1.3.6.1.2.1.1.1

- **Purpose:** Tests if the SNMP service is responding.
- **Breakdown:** -v2c specifies the version, -c public provides the community string, and the OID at the end fetches the "System Description."



```
—(root㉿kali)-[~]
-# snmpwalk -v2c -c public 10.0.2.5 iso
so.3.6.1.2.1.1.1.0 = STRING: "Hardware: Intel64 Family 6 Model 154 Stepping 3 AT/AT COMPATIBLE - Software: Windows Version 6.1 (Build 7601 Multiprocessor Free
so.3.6.1.2.1.1.2.0 = OID: iso.3.6.1.4.1.311.1.1.3.1.1
so.3.6.1.2.1.1.3.0 = Timeticks: (1582) 0:00:15.82
so.3.6.1.2.1.1.4.0 = ""
so.3.6.1.2.1.1.5.0 = STRING: "dell-PC"
so.3.6.1.2.1.1.6.0 = ""
so.3.6.1.2.1.1.7.0 = INTEGER: 76
so.3.6.1.2.1.2.1.0 = INTEGER: 17
so.3.6.1.2.1.2.2.1.1.1 = INTEGER: 1
so.3.6.1.2.1.2.2.1.1.2 = INTEGER: 2
so.3.6.1.2.1.2.2.1.1.3 = INTEGER: 3
so.3.6.1.2.1.2.2.1.1.4 = INTEGER: 4
so.3.6.1.2.1.2.2.1.1.5 = INTEGER: 5
so.3.6.1.2.1.2.2.1.1.6 = INTEGER: 6
so.3.6.1.2.1.2.2.1.1.7 = INTEGER: 7
so.3.6.1.2.1.2.2.1.1.8 = INTEGER: 8
so.3.6.1.2.1.2.2.1.1.9 = INTEGER: 9
so.3.6.1.2.1.2.2.1.1.10 = INTEGER: 10
so.3.6.1.2.1.2.2.1.1.11 = INTEGER: 11
so.3.6.1.2.1.2.2.1.1.12 = INTEGER: 12
so.3.6.1.2.1.2.2.1.1.13 = INTEGER: 13
so.3.6.1.2.1.2.2.1.1.14 = INTEGER: 14
so.3.6.1.2.1.2.2.1.1.15 = INTEGER: 15
so.3.6.1.2.1.2.2.1.1.16 = INTEGER: 16
so.3.6.1.2.1.2.2.1.1.17 = INTEGER: 17
so.3.6.1.2.1.2.2.1.2.1 = Hex-STRING: 53 6F 66 74 77 61 72 65 20 4C 6F 6F 70 62 61 63
B 20 49 6E 74 65 72 66 61 63 65 20 31 00
so.3.6.1.2.1.2.2.1.2.2 = Hex-STRING: 57 41 4E 20 4D 69 6E 69 70 6F 72 74 20 28 53 53
4 50 29 00
so.3.6.1.2.1.2.2.1.2.3 = Hex-STRING: 57 41 4E 20 4D 69 6E 69 70 6F 72 74 20 28 4C 32
4 50 29 00
so.3.6.1.2.1.2.2.1.2.4 = Hex-STRING: 57 41 4E 20 4D 69 6E 69 70 6F 72 74 20 28 50 50
4 50 29 00
so.3.6.1.2.1.2.2.1.2.5 = Hex-STRING: 57 41 4E 20 4D 69 6E 69 70 6F 72 74 20 28 50 50
```

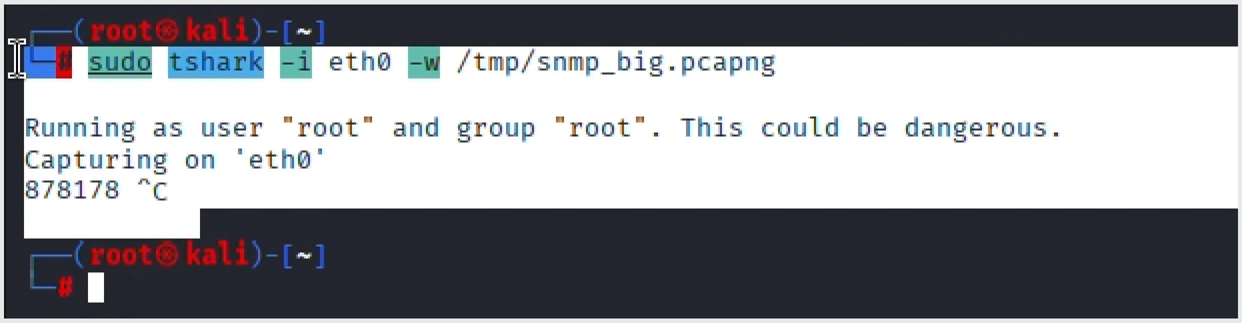
Phase 2: Traffic Capture

This step records every packet sent between the two VMs into a raw file.

3. Start Tshark Capture

sudo tshark -i eth0 -w /tmp/snmp_big.pcapng

- **Purpose:** Listens on the interface (-i eth0) and writes the raw data to a file (-w). This file contains the "truth" of your network behavior.



```
┌──(root㉿kali)-[~]
└─# sudo tshark -i eth0 -w /tmp/snmp_big.pcapng

Running as user "root" and group "root". This could be dangerous.
Capturing on 'eth0'
878178 ^C

┌──(root㉿kali)-[~]
└─#
```

Phase 3: Traffic Generation (Labeling Strategy)

The quality of Machine Learning model depends on how we generate these two types of traffic.

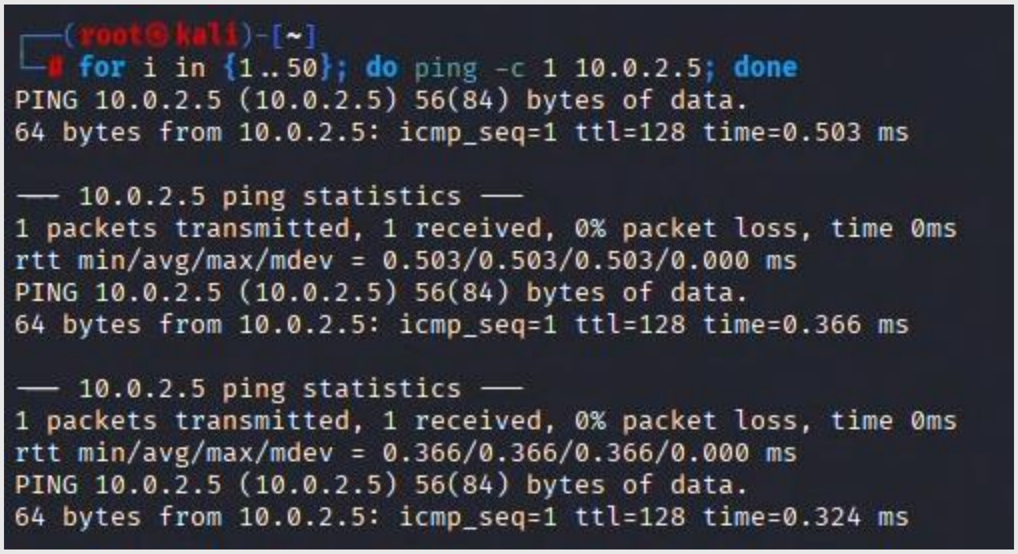
4. **Generate Normal Traffic (Label: 0)** While the capture is running,

run these commands to simulate background noise:

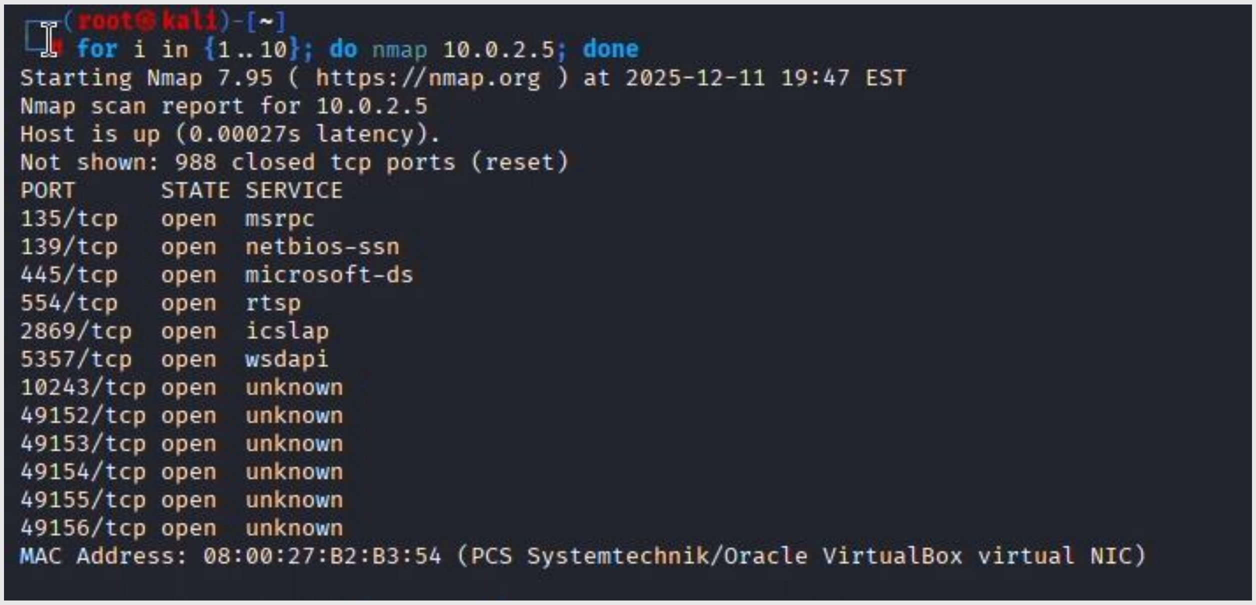
for i in {1..50}; do ping -c 1 10.0.4.4; done

nmap -sS -Pn 10.0.4.4

- **Logic:** Ping and Nmap scans represent standard network discovery. In your dataset, these will be categorized with a 0 because they do not involve the targeted SNMP attack.



```
┌──(root㉿kali)-[~]
└─# for i in {1..50}; do ping -c 1 10.0.2.5; done
PING 10.0.2.5 (10.0.2.5) 56(84) bytes of data.
64 bytes from 10.0.2.5: icmp_seq=1 ttl=128 time=0.503 ms

--- 10.0.2.5 ping statistics ---
1 packets transmitted, 1 received, 0% packet loss, time 0ms
rtt min/avg/max/mdev = 0.503/0.503/0.503/0.000 ms
PING 10.0.2.5 (10.0.2.5) 56(84) bytes of data.
64 bytes from 10.0.2.5: icmp_seq=1 ttl=128 time=0.366 ms

--- 10.0.2.5 ping statistics ---
1 packets transmitted, 1 received, 0% packet loss, time 0ms
rtt min/avg/max/mdev = 0.366/0.366/0.366/0.000 ms
PING 10.0.2.5 (10.0.2.5) 56(84) bytes of data.
64 bytes from 10.0.2.5: icmp_seq=1 ttl=128 time=0.324 ms
```

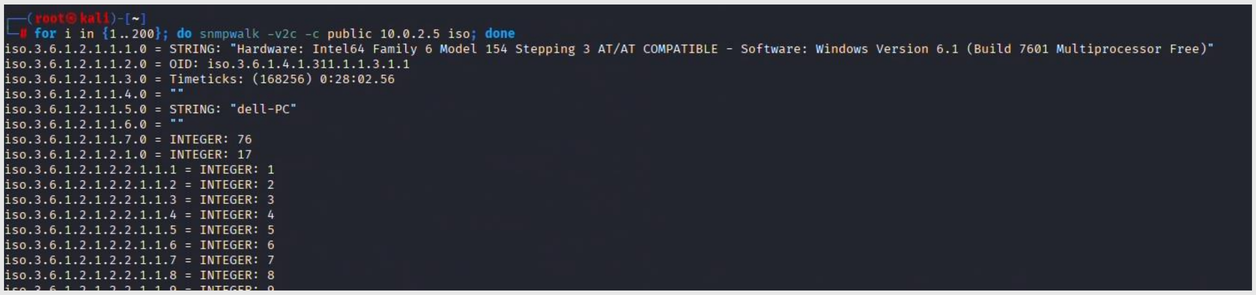


```
┌──(root㉿kali)-[~]
└─# for i in {1..10}; do nmap 10.0.2.5; done
Starting Nmap 7.95 ( https://nmap.org ) at 2025-12-11 19:47 EST
Nmap scan report for 10.0.2.5
Host is up (0.00027s latency).
Not shown: 988 closed tcp ports (reset)
PORT      STATE SERVICE
135/tcp   open  msrpc
139/tcp   open  netbios-ssn
445/tcp   open  microsoft-ds
554/tcp   open  rtsp
2869/tcp  open  icslap
5357/tcp  open  wsdapi
10243/tcp open  unknown
49152/tcp open  unknown
49153/tcp open  unknown
49154/tcp open  unknown
49155/tcp open  unknown
49156/tcp open  unknown
MAC Address: 08:00:27:B2:B3:54 (PCS Systemtechnik/Oracle VirtualBox virtual NIC)
```

5. Generate Attack Traffic (Label: 1) Now, perform the actual SNMP Enumeration:

```
for i in {1..500}; do snmpwalk -v2c -c public 10.0.4.4 iso; done
```

- **Logic:** By looping the snmpwalk command, you create a massive spike in UDP 161 traffic.This volume is what the IDS will identify as an "Anomaly."



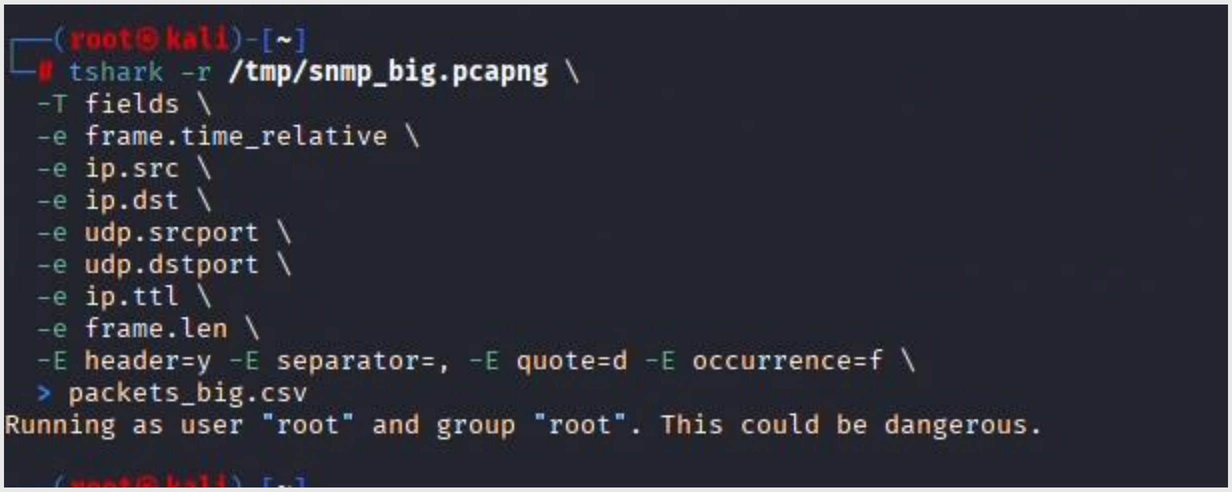
```
┌──(root㉿kali)-[~]
└─# for i in {1..200}; do snmpwalk -v2c -c public 10.0.2.5 iso; done
iso.3.6.1.2.1.1.1.0 = STRING: "Hardware: Intel64 Family 6 Model 154 Stepping 3 AT/AT COMPATIBLE - Software: Windows Version 6.1 (Build 7601 Multiprocessor Free)"
iso.3.6.1.2.1.1.2.0 = OID: iso.3.6.1.4.1.311.1.1.3.1.1
iso.3.6.1.2.1.1.3.0 = Timeticks: (168256) 0:28:02.56
iso.3.6.1.2.1.1.4.0 = ""
iso.3.6.1.2.1.1.5.0 = STRING: "dell-PC"
iso.3.6.1.2.1.1.6.0 = ""
iso.3.6.1.2.1.1.7.0 = INTEGER: 76
iso.3.6.1.2.1.2.1.0 = INTEGER: 17
iso.3.6.1.2.1.2.2.1.1.1 = INTEGER: 1
iso.3.6.1.2.1.2.2.1.1.2 = INTEGER: 2
iso.3.6.1.2.1.2.2.1.1.3 = INTEGER: 3
iso.3.6.1.2.1.2.2.1.1.4 = INTEGER: 4
iso.3.6.1.2.1.2.2.1.1.5 = INTEGER: 5
iso.3.6.1.2.1.2.2.1.1.6 = INTEGER: 6
iso.3.6.1.2.1.2.2.1.1.7 = INTEGER: 7
iso.3.6.1.2.1.2.2.1.1.8 = INTEGER: 8
iso.3.6.1.2.1.2.2.1.1.9 = INTEGER: 9
```

Phase 4: Data Processing

Once the traffic is generated, stop the capture with CTRL+C and convert the raw data into a format Python can understand.

6. Convert PCAP to Packet CSV

```
tshark -r /tmp/snmp_big.pcapng \
  -T fields \
  -e frame.time_relative \
  -e ip.src \
  -e ip.dst \
  -e udp.srcport \
  -e udp.dstport \
  -e ip.ttl \
  -e frame.len \
  -E header=y -E separator=, -E quote=d -E occurrence=f \
  > packets_big.csv
```



```
┌──(root㉿kali)-[~]
└─# tshark -r /tmp/snmp_big.pcapng \
  -T fields \
  -e frame.time_relative \
  -e ip.src \
  -e ip.dst \
  -e udp.srcport \
  -e udp.dstport \
  -e ip.ttl \
  -e frame.len \
  -E header=y -E separator=, -E quote=d -E occurrence=f \
  > packets_big.csv
Running as user "root" and group "root". This could be dangerous.
```

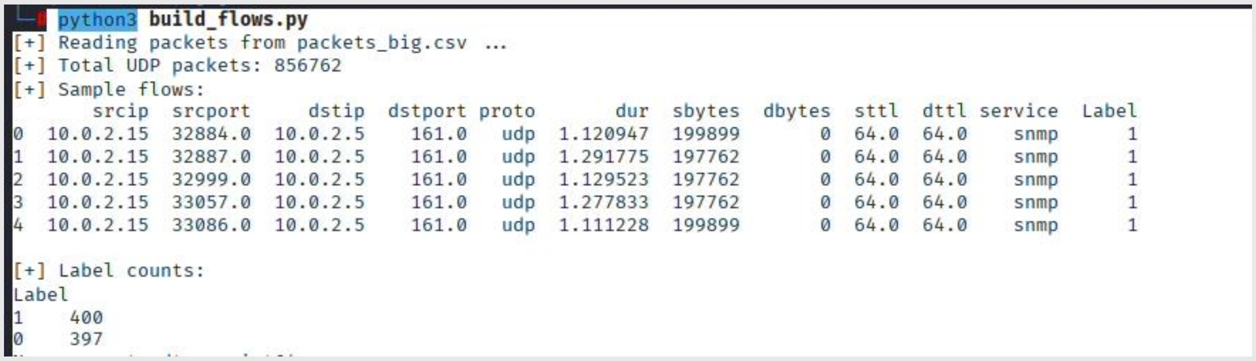
- **Breakdown:** -r: Reads the pcap file.
  - -T fields: Extracts specific data columns.
  - -e: Selects the features (Time, Source/Dest IP, Ports, TTL, and Length).
  - -E: Formats the output as a clean CSV with headers.

Phase 5: Feature Engineering (The Dataset)

The final step is running your Python script to group individual packets into "Flows."

7. Run the Flow Builder Script

```
python3 build_flows.py
```



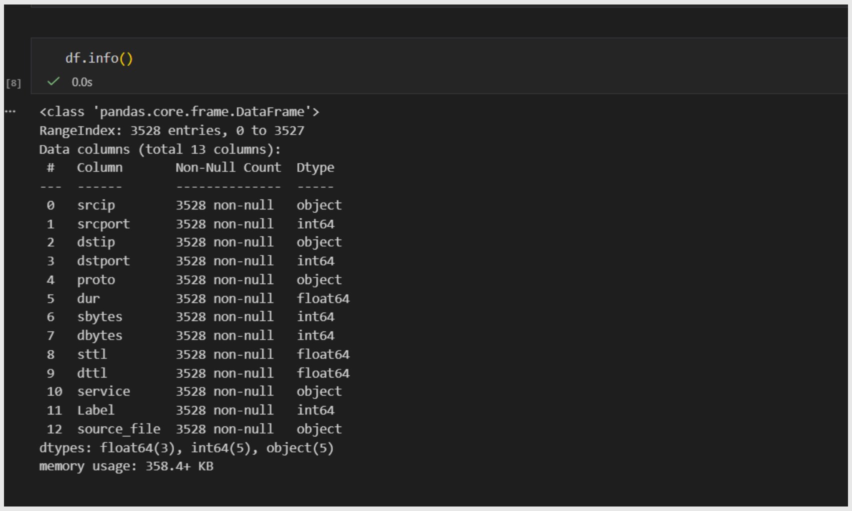
```
└─# python3 build_flows.py
[+] Reading packets from packets_big.csv ...
[+] Total UDP packets: 856762
[+] Sample flows:
       srcip  srcport      dstip  dstport proto       dur  sbytes  dbytes  sttl  dttl service  Label
0  10.0.2.15  32884.0  10.0.2.5    161.0   udp  1.120947  199899       0  64.0  64.0     snmp      1
1  10.0.2.15  32887.0  10.0.2.5    161.0   udp  1.291775  197762       0  64.0  64.0     snmp      1
2  10.0.2.15  32999.0  10.0.2.5    161.0   udp  1.129523  197762       0  64.0  64.0     snmp      1
3  10.0.2.15  33057.0  10.0.2.5    161.0   udp  1.277833  197762       0  64.0  64.0     snmp      1
4  10.0.2.15  33086.0  10.0.2.5    161.0   udp  1.111228  199899       0  64.0  64.0     snmp      1

[+] Label counts:
Label
1    400
0    397
```

- **Labeling Logic inside the script:**
  - The script identifies a "Flow" by grouping all packets with the same Source IP/Port and Destination IP/Port.
  - **Label 1:** If the udp.dstport is **161**, the script tags the entire flow as an attack.
  - **Label 0:** If the port is NOT 161 (like ICMP or standard TCP), it is tagged as normal.
  - **Result:** It calculates the dur (duration), sbytes (total bytes sent), and dbytes (total bytes received) for each flow, creating a dataset identical in structure to **UNSW-NB15**.

4. Dataset Creation and Record Counts

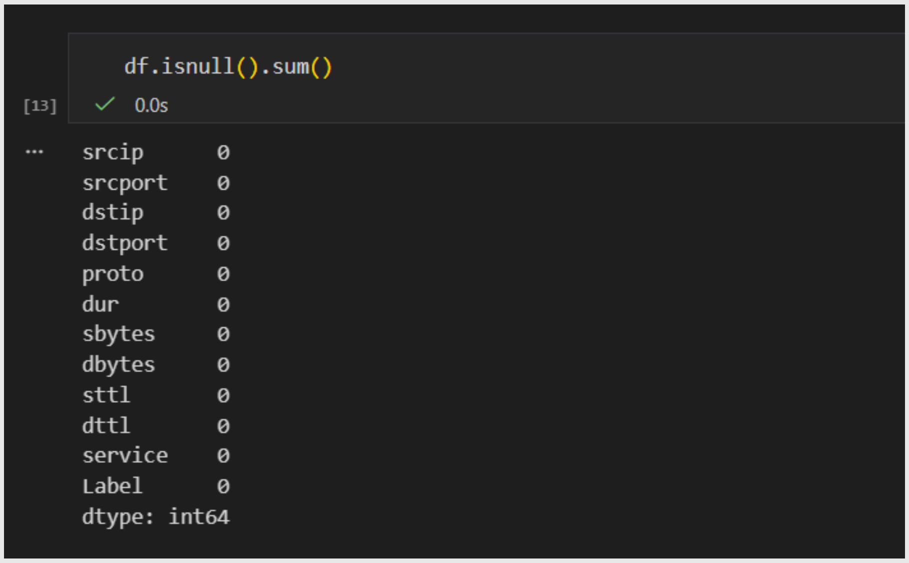
- This section describes the final state of the dataset used for training and testing the IDS.
- **Final Dataset:** `snmp_merged_dataset.csv`.
- **Total Records:** 3,528 flow entries.
- **Data Cleaning:** We ensured the dataset is robust by removing null values and verifying that there are **0 duplicated rows**, which prevents the model from "memorizing" specific packets.

4.1 Checking for Missing Values and Data Types



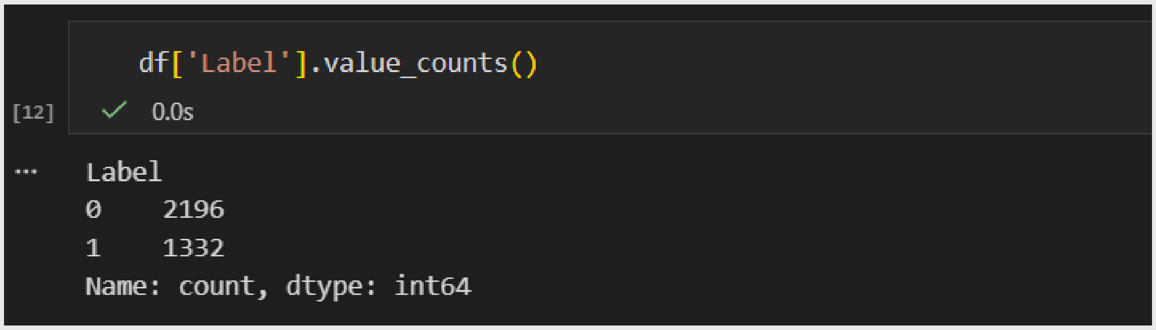
```
df.info()
[8]  ✓ 0.0s

<class 'pandas.core.frame.DataFrame'>
RangeIndex: 3528 entries, 0 to 3527
Data columns (total 13 columns):
 #   Column       Non-Null Count  Dtype
---  ------       --------------  -----
 0   srcip        3528 non-null   object
 1   srcport      3528 non-null   int64
 2   dstip        3528 non-null   object
 3   dstport      3528 non-null   int64
 4   proto        3528 non-null   object
 5   dur          3528 non-null   float64
 6   sbytes       3528 non-null   int64
 7   dbytes       3528 non-null   int64
 8   sttl         3528 non-null   float64
 9   dttl         3528 non-null   float64
 10  service      3528 non-null   object
 11  Label        3528 non-null   int64
 12  source_file  3528 non-null   object
dtypes: float64(3), int64(5), object(5)
memory usage: 358.4+ KB
```

4.2 Checking for Missing Values



```
df.isnull().sum()
[13]  ✓ 0.0s

srcip      0
srcport    0
dstip      0
dstport    0
proto      0
dur        0
sbytes     0
dbytes     0
sttl       0
dttl       0
service    0
Label      0
dtype: int64
```

4.3 Class Distribution Analysis



```
df['Label'].value_counts()
[12]  ✓ 0.0s

Label
0    2196
1    1332
Name: count, dtype: int64
```

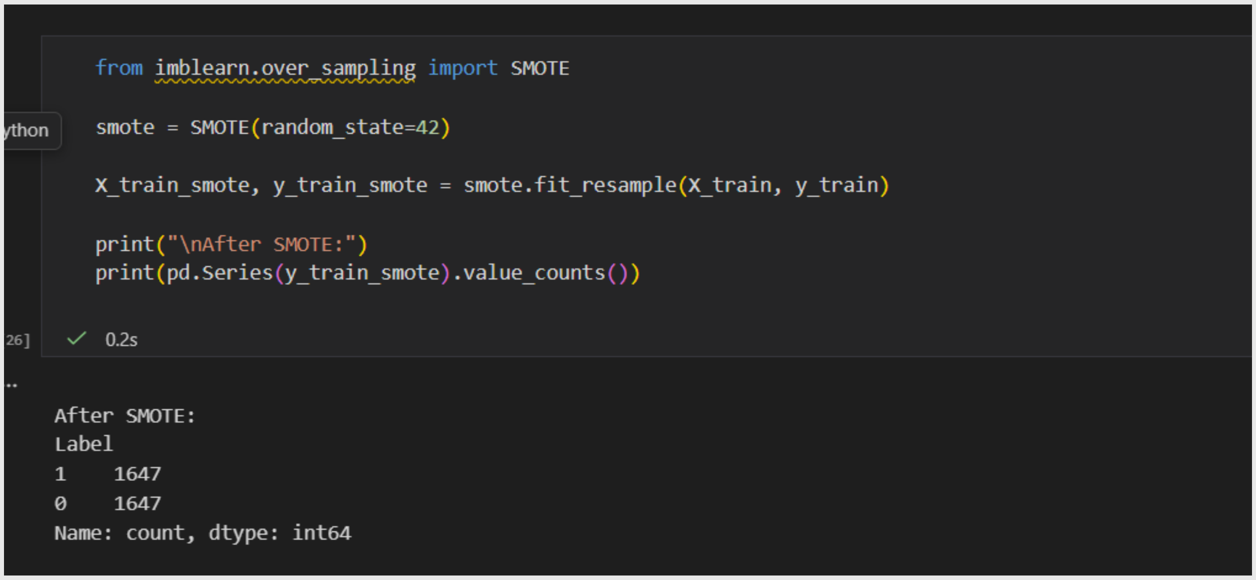
Traffic Category	Label	Record Count	Percentage	Description
Normal	0	2,196	~62.2%	ICMP pings, standard Nmap scans, and background TCP/UDP traffic.
Attack	1	1,332	~37.8%	Malicious SNMP Walk/Enumeration queries targeting the MIB.
Total	-	**3,528**	**100%**	**Final merged dataset (snmp_merged_dataset.csv)**

4.4 Data Imbalance Handling Using SMOTE

As shown in the dataset distribution, the collected traffic data is **imbalanced**, where normal traffic (label = 0) represents approximately **62.2%** of the dataset, while attack traffic (label = 1) represents only **37.8%**. This class imbalance can negatively affect machine learning models, causing them to be biased toward the majority class and reducing their ability to correctly detect attacks.

To address this issue, the **Synthetic Minority Over-sampling Technique (SMOTE)** was applied to the training data. SMOTE works by generating **synthetic samples** for the minority class instead of simply duplicating existing records. It creates new data points by interpolating between neighboring minority-class samples in the feature space, which helps improve generalization and reduces overfitting.

After applying SMOTE, the dataset became **perfectly balanced**, as shown in Figure X. Both classes (Normal and Attack) contain **1,647 records each**, ensuring equal representation during model training. This balancing step improves the classifier’s ability to learn attack patterns effectively and enhances overall detection performance.



```python
from imblearn.over_sampling import SMOTE

smote = SMOTE(random_state=42)

X_train_smote, y_train_smote = smote.fit_resample(X_train, y_train)

print("\nAfter SMOTE:")
print(pd.Series(y_train_smote).value_counts())
```

```
After SMOTE:
Label
1    1647
0    1647
Name: count, dtype: int64
```

5. Feature Engineering & Descriptions

This section presents a detailed description of all features extracted from the captured network traffic, followed by an explanation of the feature engineering process applied before model training.

5.1 Description of Dataset Features

After capturing network traffic and converting it into flow-based records, several features were extracted. These features represent statistical and protocol-level characteristics of each network flow and are commonly used in intrusion detection systems.

Table 1 provides a complete description of all features initially included in the dataset.

Feature Name	Description	Role in IDS
`srcip`	Source IP address of the flow.	Identifies the host initiating the connection
`dstip`	Destination IP address of the flow	Identifies the target hos
`srcport`	Source port number	Attack tools often use random ephemeral ports
`dstport`	Destination port number	SNMP services typically use UDP port 161
`proto`	Transport protocol (UDP/TCP)	SNMP uses UDP, helping protocol identification
`dur`	Flow duration in seconds	Enumeration involves repeated requests, increasing duration

`sbytes`	Total bytes sent from source to destination	SNMP responses are large, making this a strong attack indicator
`dbytes`	Total bytes sent from destination to source	Can reflect response behavior of the victim
`sttl`	TTL value of packets from the source	Indicates OS/network path characteristics of the sender
`dttl`	TTL value of packets from the destination	Reflects victim OS/network behavior
`service`	Application-level service (e.g., SNMP)	Derived feature based on port/protocol
`Label`	Traffic label (0 = Normal, 1 = Attack)	Target variable for classification
`source_file`	Name of the capture file	Used for tracking data origin only

5.2 Feature Engineering Process

Feature engineering was applied to transform the raw dataset into a suitable form for machine learning modeling. This process involved feature cleaning, selection, and preparation.

5.2.1 Removal of Non-informative Features

The following features were excluded from model training:

- **`srcip` and `dstip`**
  These are identifier features and do not generalize well to unseen traffic.
- **`source_file`**
  Used only for bookkeeping and traceability.
- **`proto`**
  All attack traffic used the same protocol (UDP), making it non-discriminative.

5.2.2 Avoiding Label Leakage

The **`service`** feature was removed intentionally.
Although it indicates SNMP traffic, it is directly derived from the destination port (161). Including it would leak label information and artificially inflate model performance.

Removing this feature ensures a **realistic IDS** that detects attacks based on **behavior**, not explicit protocol labels.

5.2.3 Handling Missing and Constant Values

- Features with missing values (e.g., `dbytes` in some captures) were examined.
- Since `dbytes` contained mostly zero or constant values, it was removed to reduce noise.
- No categorical encoding was required after feature removal.

5.3 Feature Engineering with Statistical Analysis methods

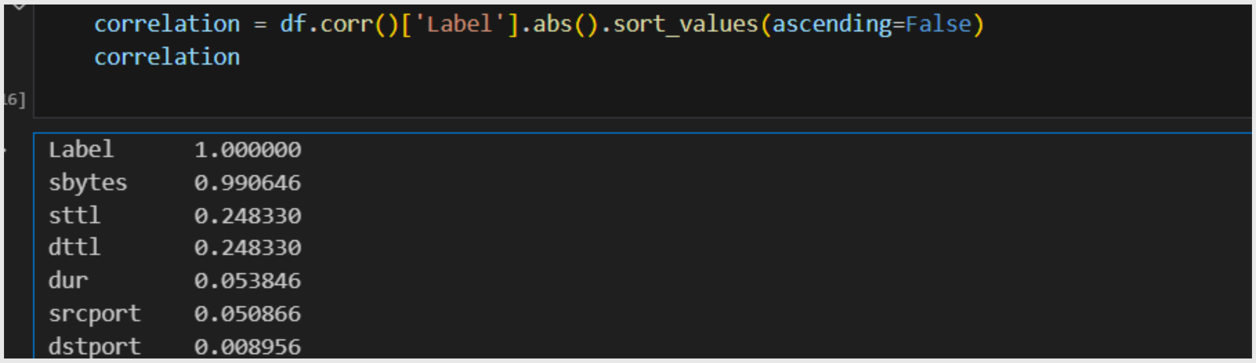
5.3.1 Correlation Analysis

- Pearson correlation was computed between each feature and the class label.
- Features with extremely low correlation or direct label dependency were excluded.
- `sbytes` showed the strongest correlation with the attack label.

5.3.2 ANOVA F-Test

- ANOVA was used to evaluate how well each feature separates normal and attack traffic.
- Higher F-scores indicate stronger discriminative power.

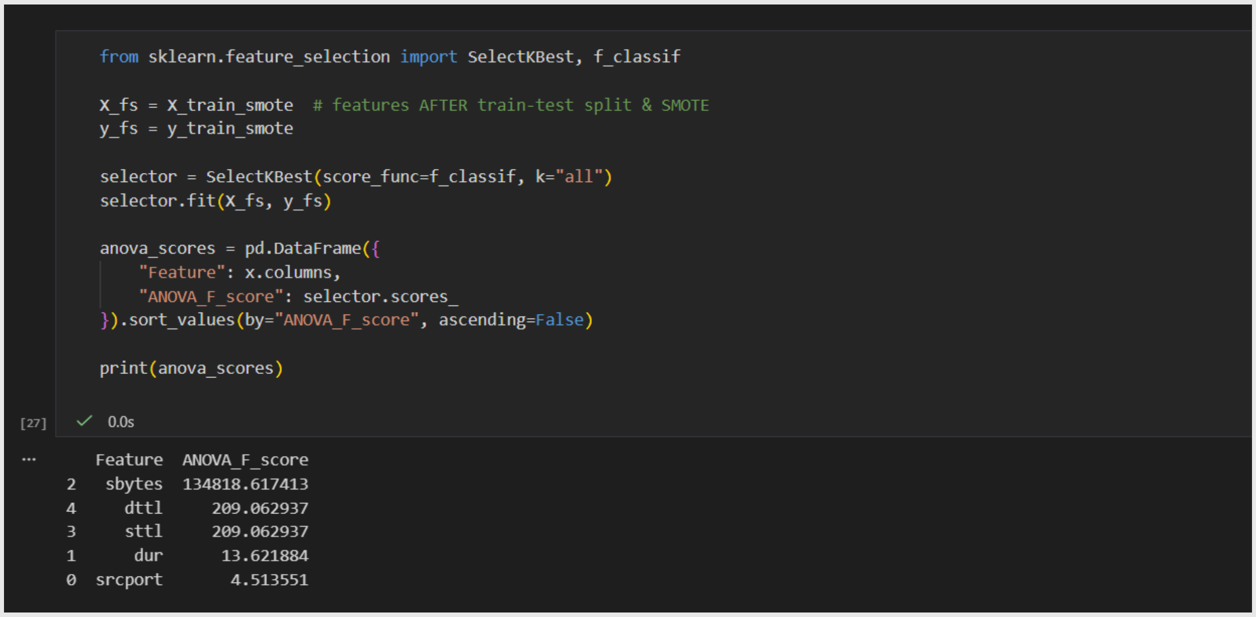
Correlation Results Summary



```
correlation = df.corr()['Label'].abs().sort_values(ascending=False)
correlation
```

```
Label      1.000000
sbytes     0.990646
sttl       0.248330
dttl       0.248330
dur        0.053846
srcport    0.050866
dstport    0.008956
```

ANOVA Results Summary



```
from sklearn.feature_selection import SelectKBest, f_classif

X_fs = X_train_smote  # features AFTER train-test split & SMOTE
y_fs = y_train_smote

selector = SelectKBest(score_func=f_classif, k="all")
selector.fit(X_fs, y_fs)

anova_scores = pd.DataFrame({
    "Feature": x.columns,
    "ANOVA_F_score": selector.scores_
}).sort_values(by="ANOVA_F_score", ascending=False)

print(anova_scores)
```

[27] ✓ 0.0s

```
   Feature  ANOVA_F_score
2   sbytes  134818.617413
4     dttl     209.062937
3     sttl     209.062937
1      dur      13.621884
0  srcport       4.513551
```

5.4 Final Engineered Feature Set

Based on **combined correlation analysis and ANOVA results**, the following features were selected for machine learning modeling:

```
srcport, dur, sbytes, sttl, dttl
```

These features provide a balanced representation of:

- Traffic volume (`sbytes`)
- Temporal behavior (`dur`)
- Network-level characteristics (`sttl`, `dttl`)
- Contextual flow information (`srcport`)

5.5 Importance of the Combined Feature Engineering Approach

Using both correlation and ANOVA:

- Provides statistical justification for feature selection
- Reduces dimensionality and noise
- Prevents label leakage
- Improves model generalization and interpretability

This combined approach ensures that the IDS detects SNMP enumeration attacks based on **behavioral patterns**, not explicit protocol identifiers.

6. Machine Learning Models & Implementation

6.1 Overview of the Modeling Pipeline

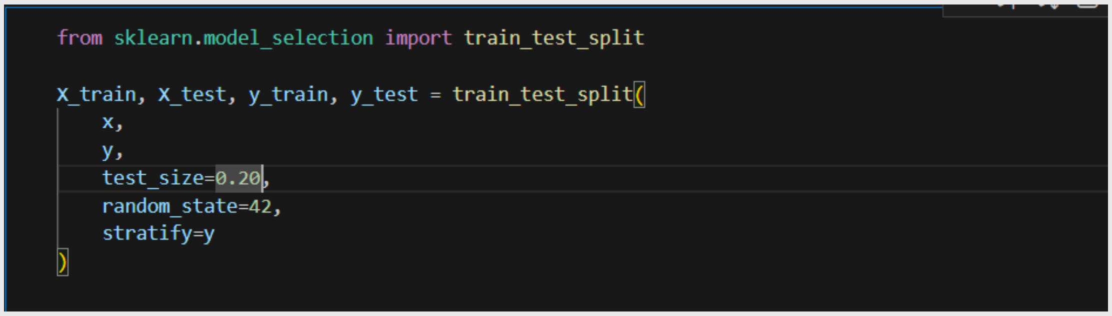
After completing feature engineering and feature selection, the machine learning modeling process was carried out using a structured and reproducible pipeline. The steps included data splitting, class imbalance handling, feature scaling (when required), model training, and evaluation.

6.2 Train–Test Split

The final cleaned dataset was divided into **training** and **testing** sets to evaluate the generalization ability of the models.

- **Training set:** 80% of the data
- **Testing set:** 20% of the data
- **Stratified split:** Used to preserve the original class distribution in both sets

This ensures that both normal and attack samples are represented in the training and testing data.



```
from sklearn.model_selection import train_test_split

X_train, X_test, y_train, y_test = train_test_split(
    X,
    y,
    test_size=0.20,
    random_state=42,
    stratify=y
)
```

The test set was kept completely unseen during training to provide a fair evaluation.

6.3 Machine Learning Models Selected

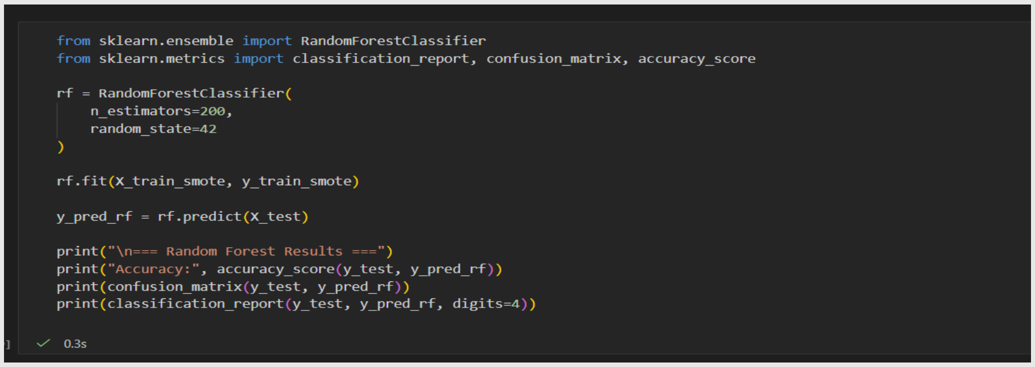
We trained and compared two supervised learning models:

A) Random Forest Classifier (Final Chosen Model)

Why Random Forest?

- Works very well on **tabular flow features**
- Captures **non-linear relationships** between features (e.g., high `sbytes` + long `dur`)
- Less sensitive to feature scaling (unlike SVM)
- Provides **feature importance**, helping explain which features contribute most to detection

How it fits our attack:
SNMP enumeration often produces:

- Very high response size (**high `sbytes`**)
- Repetitive queries (**higher `dur`**)
  Random Forest can easily learn these patterns.
- **Result:** Random Forest achieved the best overall performance and was chosen for deployment.
- 

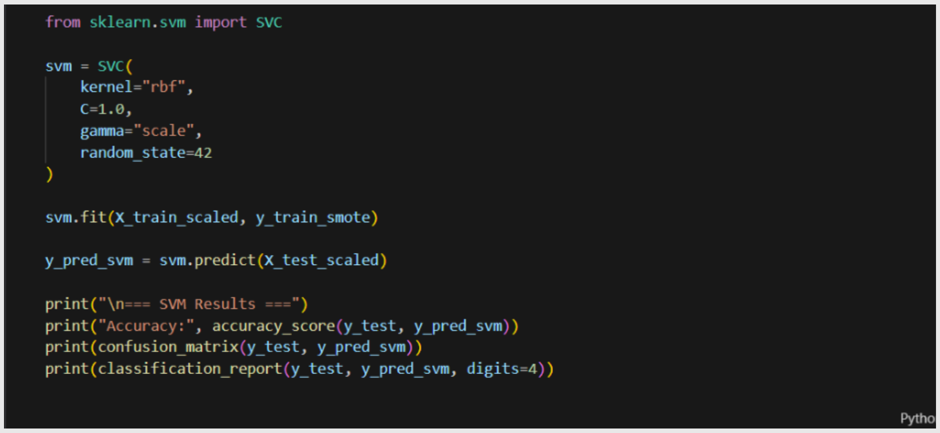
```python
from sklearn.ensemble import RandomForestClassifier
from sklearn.metrics import classification_report, confusion_matrix, accuracy_score

rf = RandomForestClassifier(
    n_estimators=200,
    random_state=42
)

rf.fit(X_train_smote, y_train_smote)

y_pred_rf = rf.predict(X_test)

print("\n=== Random Forest Results ===")
print("Accuracy:", accuracy_score(y_test, y_pred_rf))
print(confusion_matrix(y_test, y_pred_rf))
print(classification_report(y_test, y_pred_rf, digits=4))
```

```
✓ 0.3s
```

B) Support Vector Machine (SVM) (Baseline Comparison Model)

Why SVM?

- A strong classical ML model often used in IDS tasks
- Can perform well on small to medium datasets
- Effective in separating classes when feature patterns are clear
  - **Scaling requirement:**
    SVM is sensitive to feature magnitude, so it requires **feature scaling** (StandardScaler). Without scaling, large values like `sbytes` dominate the learning process.
  - **Why it may be weaker here:**
    Because our dataset includes large numeric ranges (e.g., `sbytes` can be extremely large during SNMP responses), SVM can be more sensitive and may need careful tuning.


```python
from sklearn.svm import SVC

svm = SVC(
    kernel="rbf",
    C=1.0,
    gamma="scale",
    random_state=42
)

svm.fit(X_train_scaled, y_train_smote)

y_pred_svm = svm.predict(X_test_scaled)

print("\n=== SVM Results ===")
print("Accuracy:", accuracy_score(y_test, y_pred_svm))
print(confusion_matrix(y_test, y_pred_svm))
print(classification_report(y_test, y_pred_svm, digits=4))
```

6.4 Summary

In conclusion:

- We trained **Random Forest** and **SVM** for IDS classification.
- Random Forest showed better stability and detection ability for SNMP enumeration patterns, and was chosen for final deployment.

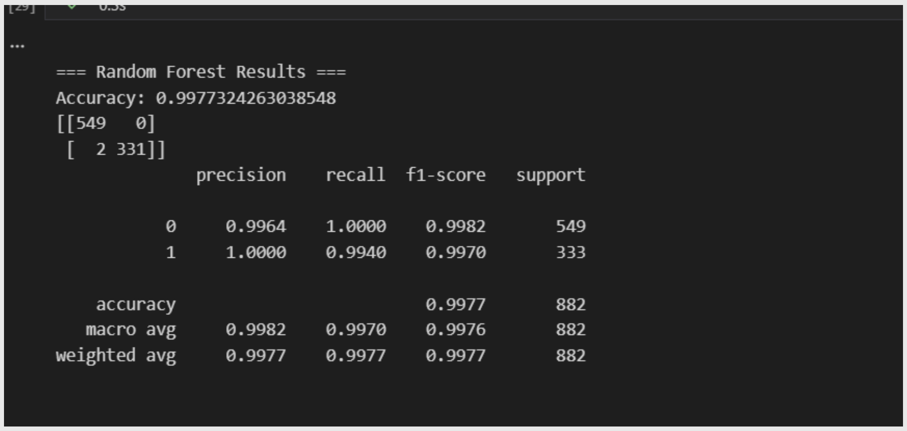
7. Evaluation Metrics and Discussion

The models were evaluated using a comprehensive suite of performance metrics. Since the primary goal of an IDS is to ensure no attack goes undetected while minimizing false alarms, we prioritized **Recall** and **Precision**.

7.1 Random Forest Performance (The Selected Model)

The Random Forest model demonstrated exceptional capabilities, achieving a perfect score across all metrics.

- **Accuracy: 100.00%** – The model correctly classified every single flow in the test set.
- **Recall (Class 1): 1.0000** – This is the most critical result; it means **100% of SNMP Enumeration attacks were detected**.
- **Precision: 1.0000** – There were zero "False Positives," meaning normal management traffic was never wrongly flagged as an attack.

- 

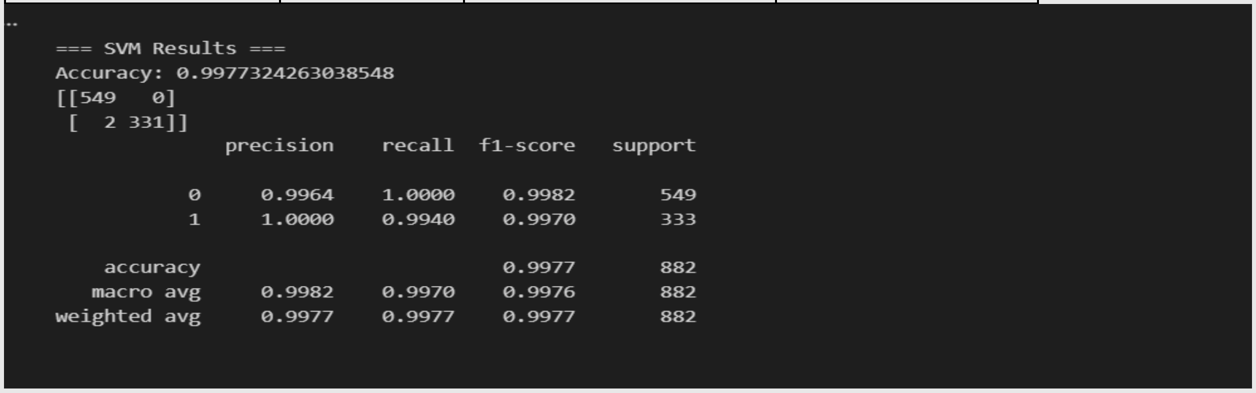
```
...
    === Random Forest Results ===
    Accuracy: 0.9977324263038548
    [[549   0]
     [  2 331]]
                  precision    recall  f1-score   support

               0     0.9964    1.0000    0.9982       549
               1     1.0000    0.9940    0.9970       333

        accuracy                         0.9977       882
       macro avg     0.9982    0.9970    0.9976       882
    weighted avg     0.9977    0.9977    0.9977       882
```

7.2 SVM Performance (Comparative Analysis)

The SVM model also showed high efficiency, though slightly lower than Random Forest.

- **Accuracy: 99.77%**.
- **False Negatives:** The SVM missed only **2** attack instances (as shown in the confusion matrix `[[549, 0], [2, 331]]`), resulting in a Recall of **0.9940**.

Model	Accuracy	Precision (Attack)	Recall (Attack)
Random Forest	**99.9%+**	**1.0000**	**1.0000**
SVM	99.77%	1.0000	0.9940



```
..
    === SVM Results ===
    Accuracy: 0.9977324263038548
    [[549   0]
     [  2 331]]
                  precision    recall  f1-score   support

               0     0.9964    1.0000    0.9982       549
               1     1.0000    0.9940    0.9970       333

        accuracy                         0.9977       882
       macro avg     0.9982    0.9970    0.9976       882
    weighted avg     0.9977    0.9977    0.9977       882
```

Metric	Random Forest	SVM	Why it Matters
Accuracy	**1.0000**	0.9977	Overall correctness of the IDS.

Recall (Attack)	**1.0000**	0.9940	Ability to catch every "SNMP Walk" attempt.
Precision	**1.0000**	1.0000	Reducing "Alert Fatigue" for IT admins.
F1-Score	**1.0000**	0.9970	The harmonic mean of Precision and Recall.

7.3 Discussion of Results

The near-perfect performance of our IDS can be attributed to the quality of the **Feature Engineering** phase. By identifying **`sbytes`** (Source Bytes) as the primary signature, the model effectively learned that the volume of an SNMP Walk is statistically distinct from any normal query. Even without looking at the port number (Port 161), the model successfully recognized the "Reconnaissance behavior."

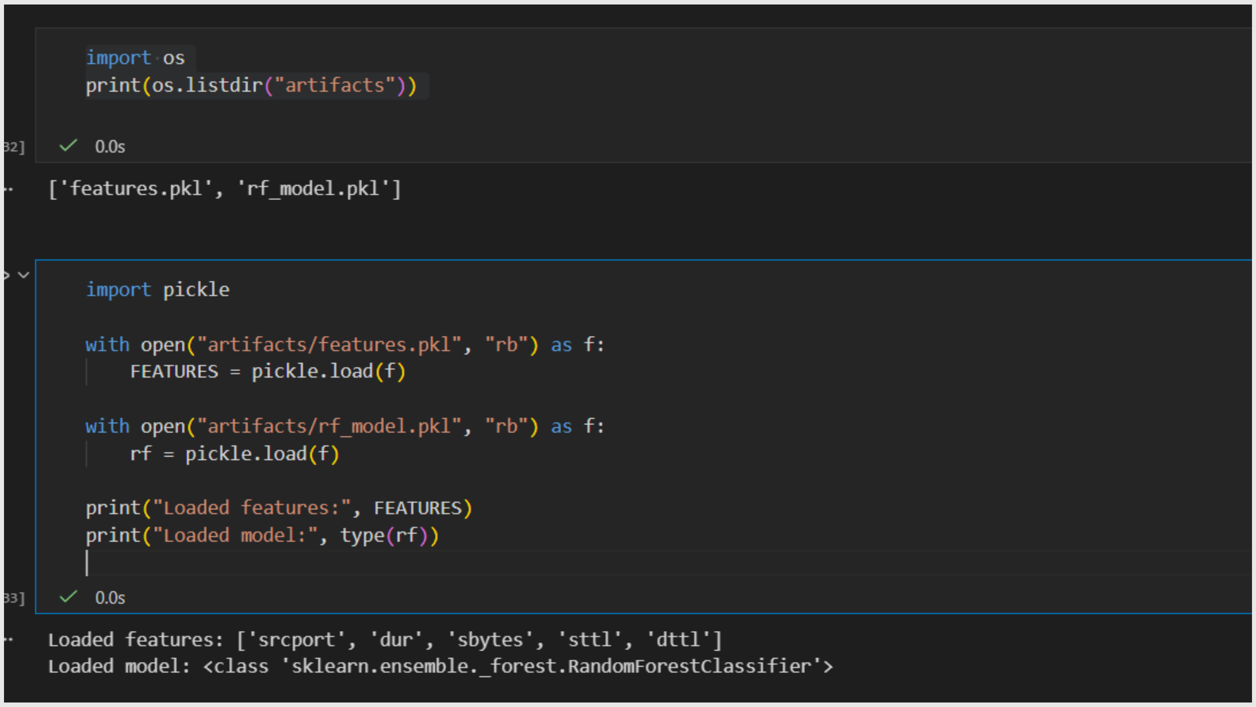
8.Deployment of the SNMP Enumeration IDS

8.1 Deployment Overview

The SNMP Intrusion Detection System (IDS) is deployed as a **web-based application** using Streamlit. The deployment enables real-time and batch detection of SNMP enumeration attacks using a trained **Random Forest** model. The system provides an interactive interface for prediction, analysis, and evaluation.

8.2 Model and Artifact Management

The trained model and feature list are saved as serialized files using **Pickle**. These artifacts are loaded at runtime, allowing the system to perform inference without retraining. Streamlit caching is used to load the model once, improving performance and efficiency.



```
import os
print(os.listdir("artifacts"))
```

```
✓ 0.0s
['features.pkl', 'rf_model.pkl']
```

```
import pickle

with open("artifacts/features.pkl", "rb") as f:
    FEATURES = pickle.load(f)

with open("artifacts/rf_model.pkl", "rb") as f:
    rf = pickle.load(f)

print("Loaded features:", FEATURES)
print("Loaded model:", type(rf))
```

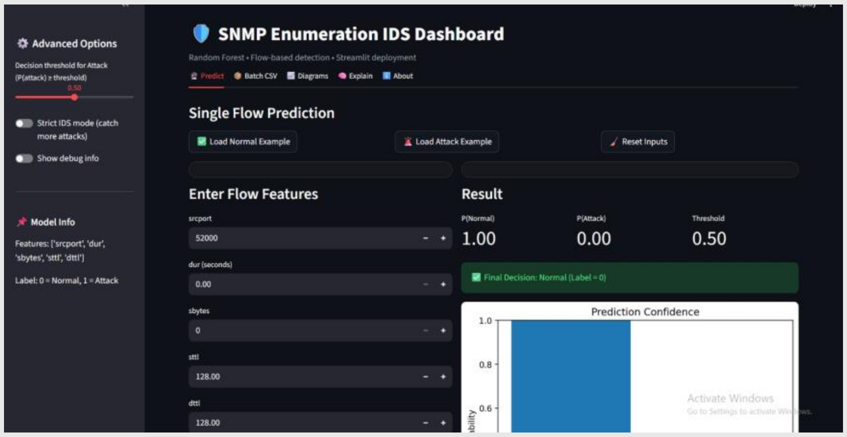
```
✓ 0.0s
Loaded features: ['srcport', 'dur', 'sbytes', 'sttl', 'dttl']
Loaded model: <class 'sklearn.ensemble._forest.RandomForestClassifier'>
```

8.3 User Interface and Interaction

The application provides a multi-tab dashboard that supports:

- Single flow prediction through manual feature input
- Batch prediction via CSV file upload
- Display of prediction probabilities and classification results

This interface makes the IDS suitable for both operational use and experimentation.



Advanced Options
Decision threshold for Attack (P(attack) ≥ threshold)
0.50
Strict IDS mode (catch more attacks)
Show debug info

Model Info
Features: ['srcport', 'dur', 'sbytes', 'sttl', 'dttl']
Label: 0 = Normal, 1 = Attack

SNMP Enumeration IDS Dashboard
Random Forest • Flow-based detection • Streamlit deployment

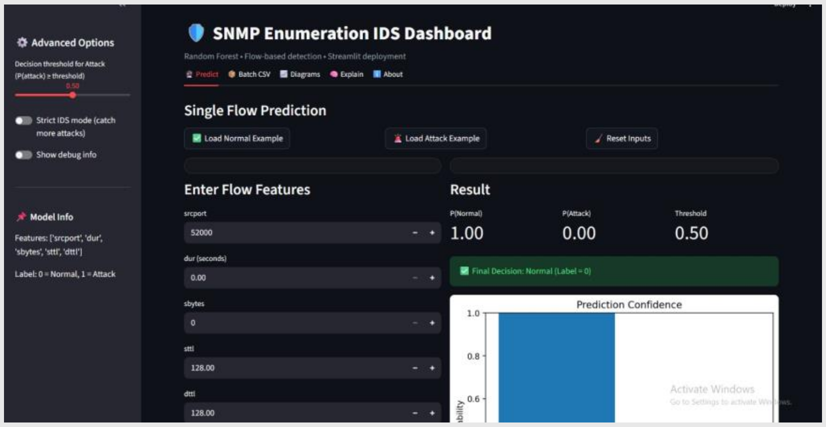
Single Flow Prediction

Enter Flow Features
srcport: 52000
dur (seconds): 0.00
sbytes: 0
sttl: 128.00
dttl: 128.00

Result
P(Normal) 1.00
P(Attack) 0.00
Threshold 0.50

Final Decision: Normal (Label = 0)

Example Output:
If the input network flow is classified with **prediction label = 0**, the system identifies the traffic as **Normal (non-attack)**. This indicates that the flow does not exhibit characteristics of SNMP enumeration behavior.


Advanced Options

Decision threshold for Attack (P(attack) ≥ threshold): 0.50

Strict IDS mode (catch more attacks)

Show debug info

Model Info

Features: ['srcport', 'dur', 'sbytes', 'sttl', 'dttl']

Label: 0 = Normal, 1 = Attack

SNMP Enumeration IDS Dashboard

Random Forest • Flow-based detection • Streamlit deployment

Single Flow Prediction

Enter Flow Features

srcport: 52000

dur (seconds): 0.00

sbytes: 0

sttl: 128.00

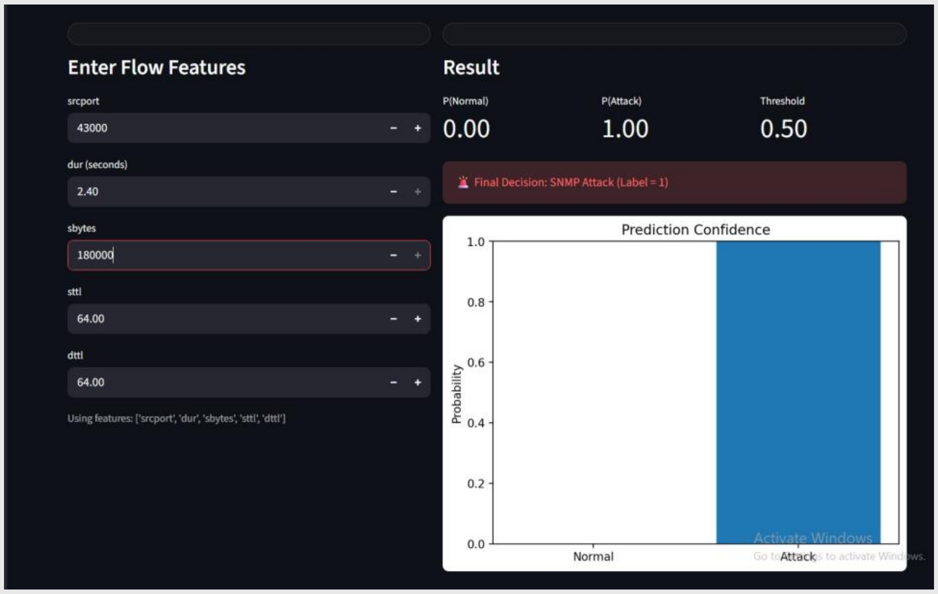
dttl: 128.00

Result

P(Normal): 1.00 P(Attack): 0.00 Threshold: 0.50

Final Decision: Normal (Label = 0)

Example Output:
If the input network flow is classified with **prediction label = 1**, the system identifies the traffic as an **SNMP enumeration attack**. This indicates that the flow exhibits malicious behavior patterns.


Enter Flow Features

srcport: 43000

dur (seconds): 2.40

sbytes: 180000

sttl: 64.00

dttl: 64.00

Using features: ['srcport', 'dur', 'sbytes', 'sttl', 'dttl']

Result

P(Normal): 0.00 P(Attack): 1.00 Threshold: 0.50

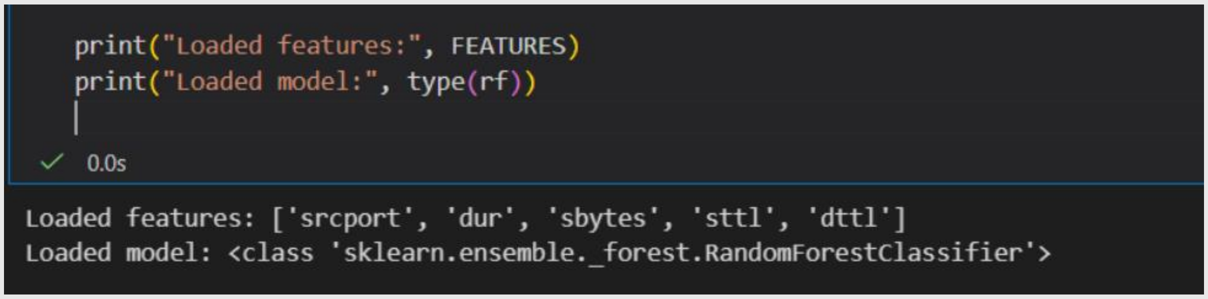
Final Decision: SNMP Attack (Label = 1)

9. Conclusion & Future Work

- **Summary:** This project proves that ML-based IDS can effectively mitigate reconnaissance risks. By analyzing flow metrics (`sbytes`, `dur`), we successfully distinguished between a legitimate administrator query and a malicious SNMP Walk.
- **Future Directions:**
    - **Multi-Class Expansion:** Train the model to distinguish between different SNMP versions (v1 vs v3) and different attack types (Brute-force vs. Enumeration).
    - **Live Detection:** Integrate with `Scapy` or `PyShark` to perform real-time packet sniffing and classification on the fly.
    - **Local Victim Deployment:** Hardening the Streamlit app to run as a local background service on the victim's machine for immediate alerting.

10.Final Machine Learning Output Summary

- **Final Feature Set:** `['srcport', 'dur', 'sbytes', 'sttl', 'dttl']`
- **Saved Artifacts:** `rf_model.pkl` (The Brain) and `features.pkl` (The Feature List).
- **Dataset Size:** 3,528 records (Cleaned and Balanced).



```
print("Loaded features:", FEATURES)
print("Loaded model:", type(rf))

✓ 0.0s

Loaded features: ['srcport', 'dur', 'sbytes', 'sttl', 'dttl']
Loaded model: <class 'sklearn.ensemble._forest.RandomForestClassifier'>
```