Ελληνικό Μεσογειακό Πανεπιστήμιο

Τμήμα Ηλεκτρολόγων Μηχανικών και Μηχανικών Υπολογιστών

Πρόγραμμα Σπουδών Μηχανικών Πληροφορικής ΤΕ

Τίτλος:

Δημιουργία Συνόλου Δικτυακών Δεδομένων με Σκοπό την Εκπαίδευση Αλγορίθμων Μηχανικής Μάθησης

Title:

Machine Learning-based Active Operating System Fingerprinting Dataset

{Νίκας Ευάγγελος-Πολυνίκης(4258)}

Επιβλέπων εκπαιδευτικός : Μαρκάκης Ευάγγελος

Επιτροπή Αξιολόγησης :

- **Μαρκάκης Ευάγγελος**
- **Παπαδάκης Νικόλαος**
- **Παναγιωτάκης Σπυρίδων**

Ημερομηνία παρουσίασης: 26/9/2022

Ευχαριστίες

Ευχαριστώ θερμά τον καθηγητή μου, Δρ. Ευάγγελο Μαρκάκη, για την ευκαιρία να πραγματοποιήσω την πτυχιακή μου εργασία μαζί του.

Επίσης, ευχαριστώ θερμά τον Research Associate, Στυλιανό Κλάδο, για την πολύτιμη βοήθεια του κατά τη διάρκεια των πειραμάτων καθώς και για τις διορθώσεις της συγκεκριμένης πτυχιακής.

Περίληψη

Οι επιθέσεις στον κυβερνοχώρο έχουν επεκταθεί σημαντικά και έχουν αυξηθεί σε αποτελεσματικότητα τα τελευταία χρόνια. Ως αποτέλεσμα, τα τρωτά σημεία των στοχευμένων συστημάτων υπολογιστών και δικτύων μπορούν να αξιοποιηθούν εξ αποστάσεως. Πριν από την εκτέλεση μιας κυβερνοεπίθεσης, πραγματοποιείται ψηφιακή αναγνώριση του στόχου. Κατά τη διάρκεια αυτής της φάσης, εκτελούνται σαρώσεις του συστήματος, προκειμένου να βρεθούν πληροφορίες όπως το λειτουργικό σύστημα (OS) του στόχου, οι ανοικτές θύρες και οι υπηρεσίες. Οι σαρώσεις συστημάτων κατηγοριοποιούνται σε παθητικές και ενεργές . Η παρούσα πτυχιακή εργασία προτείνει δύο σύνολα δεδομένων σάρωσης λειτουργικού συστήματος σε δύο από τα πιο πρόσφατα πρότυπα ροής δικτύου της Cisco (NetFlow v9 και IPFIX), επιτρέποντας σε ένα εργαλείο παρακολούθησης δικτύου με μηχανική μάθηση να εντοπίζει άμεσα τη συγκεκριμένη εν εξελίξει σάρωση που υφίσταται ένα δίκτυο και αυτοματοποιώντας τη διαδικασία ανίχνευσης ενεργών σαρώσεων λειτουργικού συστήματος.

Abstract

Cyber-attacks have expanded significantly and progressed in effectiveness over the past few years. As a result, vulnerabilities of targeted computer and network systems can be exploited remotely. Before the initialization of a cyber-attack, a digital reconnaissance of the target is conducted. During this phase, fingerprinting scans are executed, in order to found information such as Operating System (OS) of the target, open ports and Services. The fingerprinting scans are categorized to passive and active fingerprinting scans. This thesis suggests two OS fingerprinting scan datasets in two of the most recent network flow standards from Cisco (NetFlow v9 and IPFIX), allowing a machine learning network monitoring tool to instantly identify the specific on-going scan that a network is undergoing and automating the active OS fingerprint scan detection process.

Table of Contents

List of Figures

List of Tables

Chapter 1 - Introduction

A cyber-attack [1] is the attempt to obtain unauthorized access to a computer, computing system, or computer network with the intention of harming them. The goal of a cyber-attack is to disable, disrupt, destroy, or take control of a computer system. In recent years, the volume of sensitive information transferred every day over the internet has significantly increased as a result of the rapid advancement in technology. This advancement is also leveraged by attackers too, as it offers the opportunity for remote attacks[2] to be conducted.

In order for a cyber-attack to be conducted, attackers need first to orchestrate their way through before starting exploiting vulnerabilities. Firstly, a reconnaissance phase[3] takes place. With this procedure, an attacker can gather information about the targeted system or network. This information could be port status, Operating System (OS), and Services of the target. In order to get this information, a fingerprinting scan[4] needs to be executed. Fingerprinting scans are split into two (2) categories, namely: passive and active. With the passive fingerprinting, the captured data could be analyzed while being offline. The capturing process of this method is undetectable. However, the active fingerprinting is sending and receiving packets in real time meaning that the results are presented in real-time. However, this procedure is traceable because of some network-related indicators of intrusion, like slow network connections, or inbound traffic from unusual geographic locations. As a result, a cybersecurity analyst that thoroughly monitors the network may understand that a cyber-attack is occurring. Still, this procedure is rather hard, even for experienced network analysts.

To automate the process of active OS fingerprint scan detection, this thesis proposes an OS fingerprinting scan dataset in two of the most recent Cisco network flow standards (NetFlow v9[5] and IPFIX[6]), which enables a machine learning network monitoring tool to instantly identify the specific on-going scan that a network in undergoing. The datasets are comprised of bidirectional flows. As a result, a machine learning network monitoring software could successfully predict an active OS fingerprinting scan solely from the attacker’s inbound network traffic.

This thesis is structured as demonstrated:

- Chapter 1 showcases the Introduction,

- Chapter 2 represents the State of the Art,
- Chapter 3 presents the dataset implementation
- Chapter 4 showcases the evaluation process of the datasets,
- Chapter 5 presents the conclusion and the future work.

Chapter 2 - State of the Art

In this section, a brief overview of the State-Of-The-Art will showcase the research studies that have been conducted in the Operating System (OS) fingerprinting field of study.

A passive network flow-based framework for OS fingerprinting was proposed by Tomas Jirsik et al. [7]. Large network infrastructures can be inspected by the framework because of the nature of the flows and observation points that can be placed in the main network traffic hub and borders, meaning that all connected hosts can be monitored simultaneously.

In [8], Rohit Tyagi et al. suggest a passive OS fingerprinting method using TCP packet information for unauthorized OS detection. Their method conducts a TCP header analysis with time-to-live (TTL), total length, window size, and option fields of the IP packet utilized. The algorithm implemented for the identification of the OS is a Euclidean distance estimation algorithm that specifies the TCP header fields and options.

Additionally, Ahmet Aksou et al. [9] proposed a machine learning-based passive method of OS fingerprinting, utilizing a dataset composed of TCP/IP protocol headers. Network data was collected with Wireshark from a local network connection, emulating an as realistic scenario as possible. For the dataset testing, the included protocols TCP, IP, UDP, HTTP, DSN, ICMP, SSL, FTP and SSH were accurately classified. Several machine learning algorithms were used for the created dataset performance evaluation. WEKA, a machine learning evaluation tool, was utilized. Results showed that the protocol headers average classification performance accuracy was 50.94%.

Following the passive OS fingerprinting method, Martin Lastovicka et al. [10] proposed a system architecture, for large network monitoring. Monitoring of the networks in fulfilled with the utilization of in the IPFIX network format. The collected IPFIX network data was stored in a database. The stored network traffic was parsed from the database to be processed for OS identification.

Continuing their research, Martin Lastovicka et al. [11] created three more passive OS fingerprinting methods based on HTTP User-agents parsing, which is contains information about the sender's OS and browser. The network data type they utilized was IPFIX network flows.

In another adaptation of passive OS fingerprinting, Martin Lastovicka et al. [12] introduced a methodology for OS fingerprinting based on identifying a particular combination of TCP/IP packet options. Data was collected from the institution's network in the form of IPFIX network flows. After this procedure, three (3) specific TCP packet parameters were selected for the OS detection: IP Time-To-Live (TTL), Window Size, and the size of the first TCP SYN packet. The IPFIX dataset was evaluated with several machine learning algorithms. The average results of the machine learning dataset accuracy were 89%.

In addition, Desta H. Hagos et al. [13] proposed a novel approach for boosting the classification performance of a passive OS fingerprinting method utilizing machine and deep learning techniques. The data utilized originated from a commercial database. A tool was created for TCP fingerprinting scan prediction, utilizing passive traffic trails. For evaluation purposes of their approach, they synthetically created network traffic and assessed the prediction performance. Results of evaluation performance revealed an improvement of up to 94% across all scenarios of validation.

Some drawbacks can be identified from the aforementioned studies. First of all, regarding the type of data they utilized, most of the proposals utilize network traffic data that needs to be actively processed in order to be used. Moreover, even though the network data gathering was executed, those data were not utilized as a defensive mean to precisely predict if an active OS fingerprinting scan is happening in real time. Moreover, most of the studies that were machine learning-evaluated their datasets, achieved an average of 77% prediction accuracy, which is not considered high accuracy value. Additionally, in order to utilize the proposed set of data or methods, you need to assume that the attacker has already access to the network of the target devices. In order to mitigate those drawbacks, this thesis proposed two (2) datasets that:

1. are created by capturing the network traffic of an active fingerprinting OS scan
2. include bidirectional traffic captured during the scans
3. enable machine learning algorithms to be trained with and precise predict an imminent active OS fingerprinting scan

4. are in the most recent Cisco standards, namely: NetFlow v9 and IPFIX

Chapter 3 - Technology Enablers

Virtual Box

Oracle VM Virtual Box [14] is a cross-platform virtualization application that works with Intel and AMD-based computers. Furthermore, it improves the capability of your current computer so that it may run many OSs within multiple virtual machines at the same time. It was utilized in this thesis to host and virtualize every system that was part of the network topology, which were used in order to capture network traffic. Moreover, the systems' network was isolated from the rest of the network, even without having access to the internet, so that the traffic generated from the various Nmap scans would not be interrupted by scan-unrelated data.

Kali Linux

Kali Linux [15] is a Linux distribution that is free and open-source, and it is intended for advanced penetration testing a security audit. It accomplishes this by offering common tools, configurations, and automations that allow the user to focus on the task at hand rather than the surrounding activities. Kali Linux comes with numerous tools for performing a wide range of information security tasks, such as penetration testing, vulnerability management, security research, and red team testing. The specific operating system was used for the traffic generation and data capture, which was used for the creation of the datasets.

Ubuntu 20.04

Ubuntu [16] is a free and open-source Debian Linux distribution by Canonical, a popular and user-friendly operating system. In this thesis, the latest updated version of Ubuntu (20.04) was used as one of the targeted operating systems by various fingerprinting scans. The OS fingerprinting scans conducted in diverse security levels and services of Ubuntu.

Windows 10

Windows 10[17], is the most popular and user-friendly operating system, distributed by Microsoft. In this thesis, Windows 10 was used as another targeted operating system by various fingerprinting scans. The scans were carried out various security statuses and services of Windows.

Wireshark

Wireshark[18] is a, free and open-source network analyzer and packet capture tool. This tool was used with Kali Linux, capturing the network traffic during the OS fingerprinting scans. Wireshark has a user interface that is compatible with Kali Linux, so it was not necessary to use Wireshark's command-line version, tcpdump.

nProbe

NProbe[19] is a standalone command-line software application used for the monitoring, collection, and conversion of network data to network flows. NetFlow is a network protocol system that is implemented to Cisco routers to gather IP network traffic as it enters or exits an interface. The NetFlow protocol is used to analyze network traffic, to discover its origin, destination, volume and network pathways. In this thesis, the Wireshark captured data was analyzed by nProbe and exported flows to NetFlow v9 and IPFIX. However, the nProbe exported data needed to be analyzed and pre-processed.

Python Programming Language

The Python language has a wide variety of libraries, making it easier for data analysis. In this thesis, we used Python extended with the Pandas library[20]. The library focuses specifically on data analysis. In our case, it was used for dataset creation, analysis and label classification using the data provided by nProbe.

Weka

Weka[21] is a free and open-source tool that is used for dataset assessment. Weka was used for the datasets' features importance assessment and dataset evaluation with machine learning algorithms.

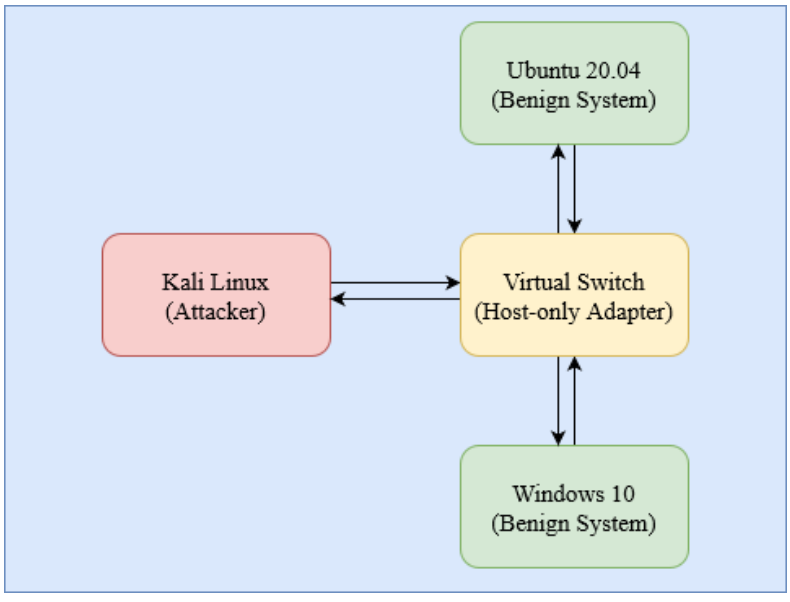
Chapter 4 - Implementation

In this section, the creation of the dataset is showcased. The network topology, software used, and dataset pre-processing procedure are also presented so that other researchers can replicate the process.

Network Topology

In this thesis, three (3) Virtual Machines (VMs) were created with: Kali Linux, Windows 10, and Ubuntu 20.04 . The attacker VM had Kali Linux 2022.1 because of the already installed active fingerprinting tools that this system provides. The network traffic produced by the fingerprinting attacks was captured from the attackers' system. The VMs that were attacked had Windows 10 Pro 21H2 and Ubuntu 20.04. These systems were deployed one at a time, because each OS produces different network traffic footprint when responding to such attacks. The targeted systems will be referred to as "benign". The local network of the attacker and benign systems were isolated. The reason for the isolation of the network was to capture the network traffic of the reconnaissance attack packets without being affected by irrelevant network flow.

Figure 1. Network Topology



Tools and method for data collection

Kali Linux provides an open-source tool called Nmap, which offers a variety of functions, such as network and operating system scans. In this study, Nmap was utilized for its capability of OS scans, to gather active OS fingerprinting network traffic.

Nmap scans are separated into four (4) types of scans. These types of scans are: OS scan, TCP SYN - Stealth scan, ACK scan, and Aggressive scan. The three (3) last scan types were conducted in combination with the OS scan. When the OS scan is executed, Nmap sends TCP and UDP packets to the victim and examines every response thoroughly. Nmap compares the responses with its OS database of known operating systems and provides a possible OS estimation. The TCP SYN-Stealth scan can be conducted swiftly, probing thousands of ports free from obtrusive firewalls because it never completes the TCP connection, making its operation undetectable, and provides a consistent distinction between open, closed, and filtered states. The ACK scan is used for organizing firewall rules sets, identifying which ports are filtered and whether or not they are stateful. When the Aggressive scan is executed, it conducts simultaneous port scanning, OS fingerprinting scanning, script scanning, and traceroute. Script scanning is a Nmap service mechanism which find vulnerabilities and malware, and gather data from databases and other network services. Additionally, traceroute service is an enhanced traceroute implementation of Nmap.

There are two (2) categories of firewall operations, stateful and stateless. Stateful firewalls are monitoring all angles of network traffic, focusing on the communication channels, characteristics, and all elements of the traffic. Stateless firewalls are utilizing the source, destination, and other information in a data packet to evaluate if the data represents a threat. The stateless firewall protocol will analyze the threat and then constrain or block the data containing it if a data packet deviates from the firewall's accepted boundaries. These firewalls have the ability to incorporate encryption or tunnels, recognize TCP connection phases, packet state, and other crucial status updates.

In this thesis, the purpose of the reconnaissance phase was to collect the network traffic produced by an active fingerprinting OS scan. To be as realistic as possible, the benign systems were up-to-date before the reconnaissance attacks were conducted. For the OS fingerprinting attacks, the aforementioned four (4) types of scans were utilized. Also, a variety of firewall and service states were applied during the fingerprinting scans. These states were separated into four (4) different categories, namely: firewall enabled, firewall disabled, SSH[22] port opened, and SSH port closed. These categories, and every possible combination of them, were chosen to record their different responses produced by the scans. One convenient way to check the response differences was the status of the firewall. When a firewall is active, it blocks malicious network traffic, in our case, the OS fingerprinting packets. Another, again, convenient way to check the response difference was the Service

status. That's why SSH was chosen. Secure Shell is a cryptographic network protocol that allows the secure operation of network services over a non-secure network. Besides the fact, that it is easy to use, it was also chosen for its popularity. These four (4) categories were tested in all possible combinations during the fingerprinting.

Table 1. Type of scans

Type of Scan	Dataset Labels
Operating System scan	Ubuntu_O
	Windows10_O
TCP SYN and OS scan	Ubuntu_sS_O
	Windows10_sS_O
TCP ACK and OS scan	Ubuntu_sA_O
	Windows10_sA_O
Aggressive scan	Ubuntu_Agg_O
	Windows10_Agg_O

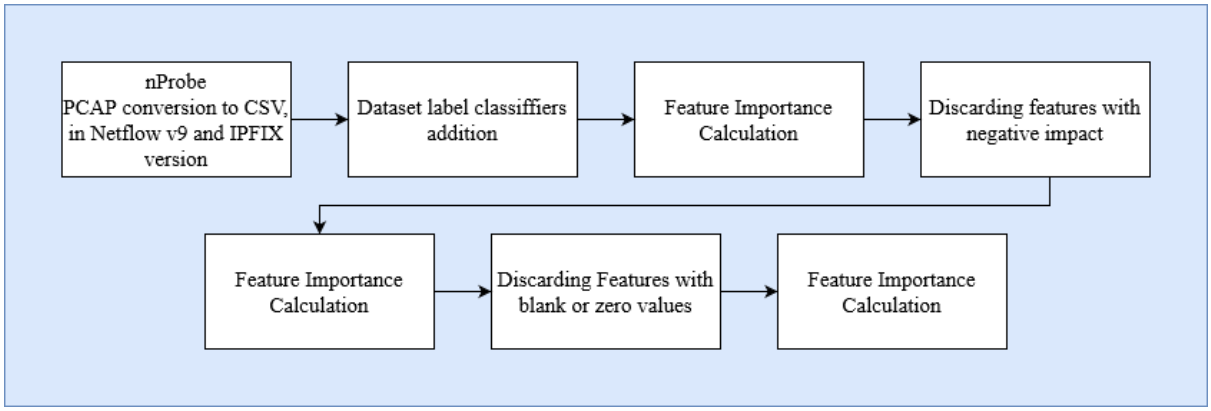
Data collection

To compose an as realistic and complete dataset as possible, the network traffic capture process occurred at the side of the attacker. Consequently, both the active fingerprinting OS scan, as well as the target's responses were captured, resulting a bidirectional network traffic capture. Wireshark was utilized for the network traffic capture during the scans.

Network Data

The captured network data was converted to NetFlow v9 and IPFIX network flows with nProbe. The NetFlow v9 dataset contained 194 network-related features, whereas the IPFIX dataset contained 182. Afterwards, each record in this dataset was given a label as an extra feature. Each fingerprinting scan that was conducted received a unique label. Both datasets had 8 labels, according to the OS and the type of active OS fingerprinting scan. **Table 1** displays these labels and the fingerprinting scans that are linked to. The steps that were performed to construct the datasets final versions are detailed below and shown as a flow diagram in **Figure 2**.

Figure 2. Pre-process Procedure



After the datasets' creation, the feature importance was calculated- which is a score. Assigned to each of the input features and signifies their importance[23]. A higher score indicates that the particular characteristic will have a larger impact when used on a machine learning algorithm. The Info Gain Attribute evaluator [24], which is one of the most well-known and effective methods for attribute evaluation, was used to evaluate the datasets features. According to the evaluation's conclusion, 26 out of 194 features from the NetFlow v9 dataset and 26 out of 182 features from the IPFIX dataset had negative influence on the datasets and were therefore excluded. Table displays the features that were removed.

Table 2. Removed Netflow and IPFIX features

Removed Netflow and IPFIX Features	
IN_SRC_MAC	BITTORRENT_HASH
OUT_DST_MAC	FLOW_SERVER_NAME
IN_DST_MAC	PLUGIN_NAME
OUT_SRC_MAC	UNTUNNELED_IPV6_SRC_ADDR
INTERFACE_NAME	UNTUNNELED_IPV6_DST_ADDR

IPV6_SRC_ADDR	IN_SRC_OSI_SAP
IPV6_DST_ADDR	OUT_DST_OSI_SAP
IPV6_NEXT_HOP	SRC_AS_MAP
APPLICATION_NAME	DST_AS_MAP
FLOW_END_REASON	DST_TO_SRC_SECOND_BYTES
UNTUNNELED_IPV4_SRC_ADDR	TLS_CIPHER
L7_INFO	TLS_UNSAFE_CIPHER
TLS_SERVER_NAME	PAYLOAD_HASH

Furthermore, by removing the features that were negatively impacting the dataset, the features that only produced the value zero (0) were removed, since they do not provide any kind of information. The number of the features that remained were 58 network-related features for the IPFIX dataset and 62 network-related features for the NetFlowV9 dataset. Both datasets feature importance scores are presented in **Table 3** and **Table 4**, and their removed features are presented in **Table 2** and **Table 5**.

Table 3. IPFIX Feature Importance score

Feature	Feature Importance Score
FLOW_START_MILLISECONDS	4.9206534
LAST_SWITCHED	4.9206534
FLOW_END_SEC	4.9206534
FLOW_START_SEC	4.9206534
FLOW_END_MILLISECONDS	4.9206534
FLOW_START_MICROSECONDS	4.9206534
FLOW_END_MICROSECONDS	4.9206534
FIRST_SWITCHED	4.9206534
ENGINE_ID	4.878642
L4_SRC_PORT	3.5996226
L4_DST_PORT	1.5599703
L4_SRV_PORT	1.3341943
DST_TO_SRC_AVG_THROUGHPUT	1.2151887
SERVER_TCP_FLAGS	1.1739528
OUT_BYTES	1.1379104
MIN_IP_PKT_LEN	1.1342636
TCP_FLAGS	1.0628229
CLIENT_TCP_FLAGS	1.0619841
TCP_WIN_MIN_OUT	0.8802321
TCP_WIN_MAX_OUT	0.8801647
NUM_PKTS_UP_TO_128_BYTES	0.8628888
OUT_PKTS	0.8626444
TCP_WIN_MSS_OUT	0.7783296
SRC_TO_DST_AVG_THROUGHPUT	0.6329838
LONGEST_FLOW_PKT	0.5033947
MAX_IP_PKT_LEN	0.5033947
IN_BYTES	0.5032326

OCTET_TOTAL	0.5032326
SRC_TO_DST_SECOND_BYTES	0.5032326
TCP_WIN_MSS_IN	0.5023772
SHORTEST_FLOW_PKT	0.5009578
NUM_PKTS_TTL_96_128	0.3068354
NUM_PKTS_TTL_32_64	0.3064899
FLOW_ID	0.2579624
TOTAL_FLOWS_EXP	0.2579624
TCP_WIN_MAX_IN	0.2479909
TCP_WIN_MIN_IN	0.2479909
TLS_VERSION	0.1897217
FLOW_DURATION_MICROSECONDS	0.1804083
FLOW_DURATION_MILLISECONDS	0.1804083
L7_PROTO_NAME	0.0009097
L7_PROTO	0.0008841
application_id	0.0008841
L7_PROTO_CATEGORY	0.0007187
TCP_WIN_SCALE_OUT	0.0004868
L7_CONFIDENCE	0.0003919
TCP_WIN_SCALE_IN	0.0003452
DST_TO_SRC_MIN_THROUGHPUT	0.0002951
DST_TO_SRC_MAX_THROUGHPUT	0.0002951
DURATION_OUT	0.0002018
NUM_PKTS_TTL_EQ_1	0.0000765
PROTOCOL_MAP	0.0000717
PROTOCOL	0.0000717
L4_DST_PORT_MAP	0.0000647
L4_SRV_PORT_MAP	0.0000647
L4_SRC_PORT_MAP	0.0000647
SEQ_PLEN_HASH	0.0000555
NUM_PKTS_1024_TO_1514_BYTES	0.0000238

Table 4. Netflow V9 Feature Importance Score

Features	Feature Importance Score
FLOW_END_MILLISECONDS	4.9230382
FLOW_END_MICROSECONDS	4.9230382
FLOW_START_MICROSECONDS	4.9230382
FLOW_START_MILLISECONDS	4.9230382
FLOW_END_SEC	4.9230382
FIRST_SWITCHED	4.9230382
FLOW_START_SEC	4.9230382
LAST_SWITCHED	4.9230382
L4_SRC_PORT	4.9225145
ENGINE_ID	4.7224902
TCP_FLAGS	1.6826871
L4_SRV_PORT	1.3518739
SERVER_TCP_FLAGS	1.1446056

IPV4_DST_ADDR	0.9999305
SHORTEST_FLOW_PKT	0.9599949
NUM_PKTS_UP_TO_128_BYTES	0.8771052
DST_TO_SRC_AVG_THROUGHPUT	0.87678
OUT_BYTES	0.8767699
OUT_PKTS	0.8746178
MIN_IP_PKT_LEN	0.8742214
SRC_TO_DST_AVG_THROUGHPUT	0.8085139
OCTET_TOTAL	0.8084765
SRC_TO_DST_SECOND_BYTES	0.8084765
IN_BYTES	0.8084765
CLIENT_TCP_FLAGS	0.8082988
MAX_IP_PKT_LEN	0.8081082
LONGEST_FLOW_PKT	0.8081082
TCP_WIN_MSS_IN	0.806602
TOTAL_FLOWS_EXP	0.2491907
FLOW_ID	0.2491907
TCP_WIN_MIN_OUT	0.000807
TCP_WIN_MAX_OUT	0.000807
NUM_PKTS_TTL_32_64	0.000701
IN_PKTS	0.0006011
PACKET_TOTAL	0.0006011
TCP_WIN_MSS_OUT	0.0005516
TCP_WIN_SCALE_OUT	0.0005156
MAX_TTL	0.0004944
TCP_WIN_SCALE_IN	0.000493
MIN_TTL	0.0004775
TCP_WIN_MIN_IN	0.0004773
TCP_WIN_MAX_IN	0.0004773
L7_PROTO_NAME	0.0003538
L4_DST_PORT	0.0002926
SEQ_PLEN	0.0002249
L7_CONFIDENCE	0.000162
SEQ_TDIFF	0.0001462
NUM_PKTS_128_TO_256_BYTES	0.0001167
TLS_VERSION	0.0000795
SEQ_TDIFF_HASH	0.0000745
NUM_PKTS_TTL_EQ_1	0.0000692
SEQ_PLEN_HASH	0.000057
IPV4_SRC_ADDR	0.0000521
IP_PROTOCOL_VERSION	0.0000379
FLOW_DURATION_MILLISECONDS	0.0000284
FLOW_DURATION_MICROSECONDS	0.0000284
SRC_TO_DST_IAT_MIN	0.0000231
NUM_PKTS_1024_TO_1514_BYTES	0.0000231
DURATION_IN	0.0000221

Table 5. IPFIX and NetFlow v9 removed irrelevant affect features

IPFIX	NetFlow v9
DOT1Q_DST_VLAN	BIFLOW_DIRECTION
MPLS_LABEL_8	POST_NAT_SRC_IPV4_ADDR
DOT1Q_SRC_VLAN	MPLS_LABEL_2
EXPORTER_IPV4_ADDRESS	MPLS_LABEL_1
FORWARDING_STATUS	MPLS_LABEL_7
EXPORTER_IPV6_ADDRESS	DIRECTION
IP_PROTOCOL_VERSION	MPLS_LABEL_8
MPLS_LABEL_6	EXPORTER_IPV6_ADDRESS
MPLS_LABEL_9	EXPORTER_IPV4_ADDRESS
MPLS_LABEL_5	MPLS_LABEL_5
DIRECTION	MPLS_LABEL_6
MPLS_LABEL_10	MPLS_LABEL_9
MPLS_LABEL_4	MPLS_LABEL_4
MPLS_LABEL_3	MPLS_LABEL_3
MPLS_LABEL_2	SAMPLED_PACKET_ID
MPLS_LABEL_1	BIFLOW_DIRECTION
MPLS_LABEL_7	PACKET_SECTION_OFFSET
SAMPLING_INTERVAL	APPLICATION_ID
PACKET_TOTAL	MPLS_LABEL_10
SRC_AS	SAMPLED_PACKET_SIZE
IPV4_NEXT_HOP	FLOW_ACTIVE_TIMEOUT
DST_AS	DOT1Q_DST_VLAN
IPV4_DST_MASK	BGP_NEXT_ADJACENT_ASN
BGP_PREV_ADJACENT_ASN	IPV4_DST_MASK
OUTPUT_SNMP	OUTPUT_SNMP
IPV4_DST_ADDR	IPV4_NEXT_HOP
DST_VLAN	SRC_AS
SRC_TOS	DST_AS
IN_PKTS	L4_SRV_PORT_MAP
IPV4_SRC_ADDR	L4_DST_PORT_MAP
INPUT_SNMP	INPUT_SNMP
IPV4_SRC_MASK	PROTOCOL_MAP
BGP_NEXT_ADJACENT_ASN	PROTOCOL
IPV4_BGP_NEXT_HOP	SRC_TOS
IPV6_SRC_MASK	IPV4_SRC_MASK
MIN_TTL	L4_SRC_PORT_MAP
TOTAL_PKTS_EXP	BGP_PREV_ADJACENT_ASN
MAX_TTL	IPV4_BGP_NEXT_HOP
IPV6_DST_MASK	DOT1Q_SRC_VLAN
SRC_VLAN	IPV6_SRC_MASK
TOTAL_BYTES_EXP	TOTAL_PKTS_EXP
ENGINE_TYPE	DST_TOS
FLOW_INACTIVE_TIMEOUT	SRC_VLAN
FLOW_ACTIVE_TIMEOUT	DST_VLAN
ICMP_TYPE	FORWARDING_STATUS
POST_NAT_SRC_IPV4_ADDR	TOTAL_BYTES_EXP

SAMPLING_ALGORITHM	ENGINE_TYPE
biflow_direction	FLOW_INACTIVE_TIMEOUT
DST_HOST_LABEL	ICMP_TYPE
POST_NAT_DST_IPV4_ADDR	IPV6_DST_MASK
DOWNSTREAM_TUNNEL_ID	SAMPLING_INTERVAL
UNTUNNELED_IPV4_DST_ADDR	POST_NAPT_SRC_TRANSPORT_PORT
UNTUNNELED_L4_DST_PORT	SAMPLING_ALGORITHM
DOWNSTREAM_SESSION_ID	POST_NAT_DST_IPV4_ADDR
NUM_PKTS_TTL_128_160	DST_TO_SRC_IAT_STDDEV
NUM_PKTS_TTL_2_5	POST_NAPT_DST_TRANSPORT_PORT
NUM_PKTS_TTL_5_32	NUM_PKTS_TTL_2_5
UNTUNNELED_L4_SRC_PORT	DOWNSTREAM_SESSION_ID
UNTUNNELED_PROTOCOL	NAT_ORIGINATING_ADDRESS_REALM
OOORDER_OUT_PKTS	NUM_PKTS_TTL_5_32
OOORDER_IN_PKTS	NUM_PKTS_TTL_64_96
UPSTREAM_SESSION_ID	NUM_PKTS_TTL_96_128
RETRANSMITTED_IN_BYTES	NUM_PKTS_TTL_128_160
RETRANSMITTED_IN_PKTS	NUM_PKTS_TTL_160_192
RETRANSMITTED_OUT_BYTES	DOWNSTREAM_TUNNEL_ID
RETRANSMITTED_OUT_PKTS	L7_PROTO_CATEGORY
NUM_PKTS_TTL_64_96	L7_PROTO
NUM_PKTS_TTL_160_192	OOORDER_OUT_PKTS
POST_NAPT_SRC_TRANSPORT_PORT	RETRANSMITTED_OUT_PKTS
L7_PROTO_RISK_NAME	OOORDER_IN_PKTS
ENTROPY_SERVER_BYTES	UNTUNNELED_PROTOCOL
L7_PROTO_RISK	UNTUNNELED_L4_DST_PORT
L7_RISK_SCORE	UNTUNNELED_L4_SRC_PORT
NUM_PKTS_TTL_192_224	UNTUNNELED_IPV4_DST_ADDR
L7_ERROR_CODE	NUM_PKTS_TTL_192_224
FLOW_VERDICT	NUM_PKTS_TTL_224_255
ENTROPY_CLIENT_BYTES	DURATION_OUT
HASSH_SERVER	SRC_TO_DST_IAT_AVG
HASSH_CLIENT	DST_HOST_LABEL
PKT_VECTOR	SRC_TO_DST_IAT_MAX
NUM_PKTS_TTL_224_255	SRC_TO_DST_IAT_STDDEV
DURATION_IN	FLOW_VERDICT
SEQ_PLEN	DST_TO_SRC_IAT_MIN
SEQ_TDIFF	DST_TO_SRC_IAT_MAX
SEQ_TDIFF_HASH	SRC_HOST_LABEL
UPSTREAM_TUNNEL_ID	L7_ERROR_CODE
FLOW_PROTO_PORT	PKT_VECTOR
CUMULATIVE_ICMP_TYPE	ENTROPY_CLIENT_BYTES
SRC_HOST_LABEL	HASSH_CLIENT
IPFIX_SAMPLING_ALGORITHM	HASSH_SERVER
SAMPLING_SIZE	ENTROPY_SERVER_BYTES
FRAME_LENGTH	L7_RISK_SCORE
NUM_PKTS_OVER_1514_BYTES	L7_PROTO_RISK
PACKETS_OBSERVED	L7_PROTO_RISK_NAME

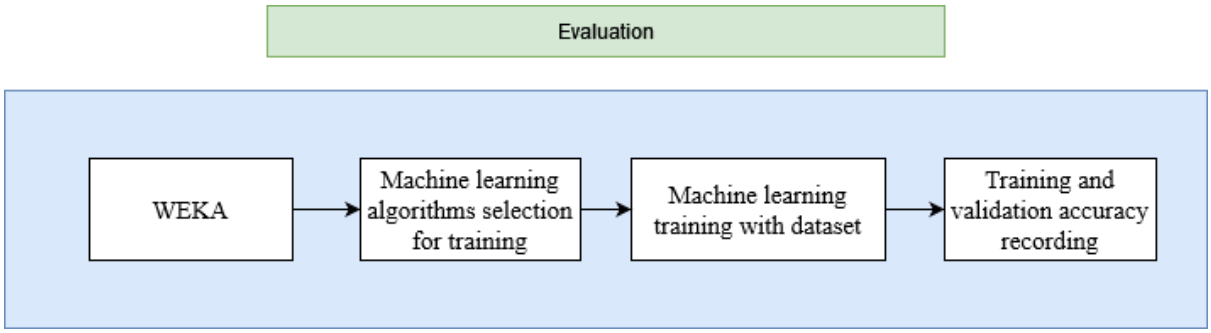
PACKETS_SELECTED	RETRANSMITTED_OUT_BYTES
SELECTOR_ID	RETRANSMITTED_IN_PKTS
OBSERVATION_POINT_ID	RETRANSMITTED_IN_BYTES
OBSERVATION_POINT_TYPE	PACKETS_SELECTED
ICMP_IPV4_CODE	FRAME_LENGTH
POST_NAPT_DST_TRANSPORT_PORT	DST_TO_SRC_IAT_AVG
NAT_ORIGINATING_ADDRESS_REALM	INGRESS_VRFID
NAT_EVENT	SAMPLING_SIZE
FIREWALL_EVENT	EGRESS_VRFID
ICMP_IPV4_TYPE	SELECTOR_NAME
INGRESS_VRFID	SAMPLING_POPULATION
EGRESS_VRFID	IPFIX_SAMPLING_ALGORITHM
SELECTOR_NAME	PORT_RANGE_END
SRC_TO_DST_MAX_THROUGHPUT	ICMP_IPV4_TYPE
SRC_TO_DST_MAX_EST_THROUGHPUT	NAT_EVENT
DST_TO_SRC_MAX_EST_THROUGHPUT	FIREWALL_EVENT
NUM_PKTS_128_TO_256_BYTES	ICMP_IPV4_CODE
NUM_PKTS_256_TO_512_BYTES	SELECTOR_ID
NUM_PKTS_512_TO_1024_BYTES	OBSERVATION_POINT_TYPE
SRC_TO_DST_MIN_THROUGHPUT	OBSERVATION_POINT_ID
NPROBE_IPV4_ADDRESS	PORT_RANGE_START
PORT_RANGE_START	SRC_FRAGMENTS
APPL_LATENCY_MS	UPSTREAM_SESSION_ID
PORT_RANGE_END	NUM_PKTS_OVER_1514_BYTES
SRC_FRAGMENTS	NUM_PKTS_256_TO_512_BYTES
DST_FRAGMENTS	NUM_PKTS_512_TO_1024_BYTES
CLIENT_NW_LATENCY_MS	CUMULATIVE_ICMP_TYPE
SERVER_NW_LATENCY_MS	SRC_TO_DST_MAX_EST_THROUGHPUT
SAMPLING_POPULATION	FLOW_PROTO_PORT
	UPSTREAM_TUNNEL_ID
	DST_TO_SRC_MAX_EST_THROUGHPUT
	DST_TO_SRC_MIN_THROUGHPUT
	DST_FRAGMENTS
	APPL_LATENCY_MS
	CLIENT_NW_LATENCY_MS
	SERVER_NW_LATENCY_MS
	NPROBE_IPV4_ADDRESS
	DST_TO_SRC_MAX_THROUGHPUT
	SRC_TO_DST_MAX_THROUGHPUT
	SRC_TO_DST_MIN_THROUGHPUT
	PACKETS_OBSERVED

Chapter 5 - Evaluation

Aim of the Experiment

According to the literature, evaluating datasets with machine learning algorithms is the most common and realistic way of evaluation [25]. The objective of this evaluation is to show that both datasets, when used as a training dataset for machine learning algorithms, provide high accuracy results. **Figure 3** presents how the evaluation process progressed. The already pre-processed datasets were used in this evaluation.

Figure 3. Evaluation



Method

The initial stage of the datasets' evaluation was to train machine learning algorithms with both datasets. The machine learning algorithms that were chosen were Naïve Bayes [26], Adaptive Boosting (AdaBoost) with Naïve Bayes classifier [27], and Cost Sensitive with Naïve Bayes classifier [28]. Weka's machine learning algorithm implementations were utilized.

Due to datasets' large size, the machine learning algorithms need to be configured accordingly. For this reason, both datasets were trained with the percentage split option enabled. Every time a dataset is analyzed, the percentage split option randomly divides the dataset into training and testing segments, estimating the performance rapidly. The IPFIX dataset was trained with an 85% percentage split and the NetFlow v9 dataset with 25%. The results can be showcased in the Results section below.

Variables

Dependent Variables

The variables that include an evaluation's results are known as dependent variables. The most common machine learning metrics were utilized in this study to assess each dataset performance. The performance of machine learning models can be evaluated based on interpretable validation and training accuracy values. They provide the prediction accuracy of the models at the time of training. On the other hand, the total of the errors that happen throughout the training stage represents the validation and training loss values. Better performance is shown by lower loss numbers and higher accuracy values [29].

The following four machine learning metrics were used in order to compare and evaluate how well the trained machine learning models performed following the training:

- Accuracy: is the machine learning evaluation percentage of correctly predicted outcomes
- Precision: presents the percentage of the correctly classification of datasets
- Recall: presents the percentage of the true positives found
- F-measure: presents the percentage of models' accuracy on a dataset

$$Precision = \frac{True\ Positive}{\text{True Positive} + \text{False Positive}}$$

$$Recall = \frac{True\ Positive}{\text{True Positive} + \text{False Negative}}$$

$$F_{measure} = 2 * \frac{Precision * Recall}{\text{Precision} + \text{Recall}}$$

$$Accuracy = \frac{True\ Positive + True\ Negative}{\text{True Positive} + \text{True Negative} + \text{False Positive} + \text{False Negative}}$$

Independent Variables

The variables that are continuously changing to track changes in the dependent variables are known as independent variables. The datasets and the machine learning models are the independent variables in this experiment because adjustments to at least one of them resulted in different values for the training accuracy and loss, as well as the validation accuracy and loss values. The dataset that is given to the machine learning models after their training in to evaluate their performance is also an independent variable because its features were altered to enable each model to use it, leading to diverse outcomes for the dependent variables accuracy, precision, recall, and f-measure score.

Fixed Variables

Fixed variables are those that remain the same over the course of an experiment. The network architecture, attacker and benign systems, fingerprinting scans, and machine learning techniques are all kept the same during the evaluation phase.

Prediction

First, prior to evaluating the datasets with machine learning algorithms, the results of the OS scans of benign systems were obtained using Nmap, so there is a tough estimation of expected results. Nmap has an OS database, where it compares the target's OS fingerprint with the database's fingerprint. However, this procedure is not always successful and is not working properly because of the first-match mechanism, where the OS detection tool chooses the first OS fingerprint that matches from its OS database and return faulty guesses. For example, if two (2) fingerprints match with the target fingerprint, it will choose the one that is first in the OS database stack. Secondly, for the machine learning part, high accuracy on Windows labels is expected. The reason is that Windows responded with more packets during the scans and needed more time for a scan to complete than the Linux OS.

Results

In **Table 6 Error! Reference source not found.**and **Table 7** the average results of every machine learning algorithm is presented, for both datasets.

Table 6. IPFIX Dataset Average Scores

IPFIX Dataset Average Scores	***Accuracy***	***Precision***	***Recall***	***F-measure***
Naïve Bayes	0.849	0.889	0.849	0.837
AdaBoost	0.915	0.915	0.915	0.912
Cost Sensitive	0.849	0.889	0.849	0.837

Table 7. NetFlow v9 Dataset Average Scores

NetFlow v9 Dataset Average Scores	***Accuracy***	***Precision***	***Recall***	***F-measure***
Naïve Bayes	0.958	0.968	0.958	0.957
AdaBoost	0.997	0.998	0.998	0.998
Cost Sensitive	0.958	0.968	0.958	0.957

In **Table 8** and **Table 9** are presented the details of the Naïve Bayes accuracy and confusion matrix results of the IPFIX dataset.

Table 8. Naïve Bayes IPFIX dataset metrics

Labels	***Precision***	***Recall***	***F-measure***
Ubuntu_Agg_O	0.599	0.999	0.749
Ubuntu_O	0.794	0.976	0.876
Ubuntu_sA_O	0.982	0.999	0.991
Ubuntu_sS_O	0.942	0.826	0.880
Windows10_Agg_O	0.991	0.314	0.476
Windows10_O	0.968	0.742	0.840
Windows10_sA_O	1.000	0.999	1.000
Windows10_sS_O	0.834	0.929	0.879
Average	0.889	0.849	0.837

Table 9.Naïve Bayes IPFIX dataset confusion matrix

a	***b***	***c***	***d***	***e***	***f***	***g***	***h***	***Classified***
58903	0	0	0	75	0	0	0	***a= ubuntu_Agg_O***
1	53765	0	0	3	1387	0	0	***b = Ubuntu_O***
3	0	58991	0	12	0	26	0	***c = Ubuntu_sA_O***
1	0	0	48763	26	0	0	10266	***d = Ubuntu_sS_O***
39370	0	0	0	18221	1	0	513	***e = Windows10_Agg_O***
0	14856	0	1	2	42635	0	0	***f = Windows10_O***
0	0	1	0	21	0	59191	11	***g = Windows10_sA_O***
2	0	1050	3020	34	0	0	54025	***h = Windows10_sS_O***

In **Table 10** and **Table 11** are presented the details of the Ada Boost accuracy and confusion matrix results of the IPFIX dataset.

Table 10. Ada Boost IPFIX dataset metrics

Labels	***Precision***	***Recall***	***F-measure***
Ubuntu_Agg_O	0.815	0.581	0.678
Ubuntu_O	0.993	0.999	0.996
Ubuntu_sA_O	0.947	0.999	0.972
Ubuntu_sS_O	0.999	0.999	0.999
Windows10_Agg_O	0.785	0.813	0.799
Windows10_O	0.999	0.993	0.996
Windows10_sA_O	0.790	0.957	0.866
Windows10_sS_O	0.995	0.979	0.987
Average	0.915	0.915	0.912

Table 11. Ada Boost IPFIX confusion matrix

a	***b***	***c***	***d***	***e***	***f***	***g***	***h***	***Classified***
34166	0	745	18	12901	5	10951	32	***a= ubuntu_Agg_O***
0	58833	0	1	0	28	1	0	***b = Ubuntu_O***
0	0	58822	0	0	0	57	0	***c = Ubuntu_sA_O***
1	0	6	58979	0	31	5	21	***d = Ubuntu_sS_O***
7765	0	1	13	47065	10	2823	195	***e = Windows10_Agg_O***
1	384	0	4	0	57498	1	3	***f = Windows10_O***
4	1	2527	0	0	0	56561	14	***g = Windows10_sA_O***
0	0	2	34	0	5	1161	57101	***h = Windows10_sS_O***

In **Table 12** and **Table 13** are presented the details of the Cost Sensitive accuracy and confusion matrix results of the IPFIX dataset.

Table 12. Cost Sensitive IPFIX dataset Metrics

Labels	***Precision***	***Recall***	***F-measure***
Ubuntu_Agg_O	0.599	0.999	0.749
Ubuntu_O	0.794	0.976	0.876
Ubuntu_sA_O	0.982	0.999	0.991
Ubuntu_sS_O	0.942	0.826	0.880
Windows10_Agg_O	0.991	0.314	0.476
Windows10_O	0.968	0.742	0.840
Windows10_sA_O	1.000	0.999	1.000
Windows10_sS_O	0.834	0.929	0.879
Average	0.889	0.849	0.837

Table 13. Cost Sensitive IPFIX dataset confusion matrix

a	***b***	***c***	***d***	***e***	***f***	***g***	***h***	***Classified***
58903	0	0	0	75	0	0	0	a= ubuntu_Agg_O
1	57365	0	0	3	1387	0	0	b = Ubuntu_O
3	0	58991	0	12	0	26	0	c = Ubuntu_sA_O
1	0	0	48763	26	0	0	10266	d = Ubuntu_sS_O
39370	0	0	0	18221	1	0	513	e = Windows10_Agg_O
0	14856	0	1	2	42635	0	0	f = Windows10_O
0	0	1	0	21	0	59191	11	g = Windows10_sA_O
2	0	1050	3020	34	0	0	54025	h = Windows10_sS_O

In **Table 14** and **Table 15** are presented the details of the Naïve Bayes accuracy and confusion matrix results of the NetFlow v9 dataset.

Table 14. Naïve Bayes NetFlow v9 dataset Metrics

Labels	***Precision***	***Recall***	***F-measure***
Ubuntu_Agg_O	1.000	0.999	1.000
Ubuntu_O	1.000	0.666	0.800
Ubuntu_sA_O	0.999	1.000	1.000
Ubuntu_sS_O	1.000	1.000	1.000
Windows10_Agg_O	0.999	0.999	0.999
Windows10_O	0.754	1.000	0.859
Windows10_sA_O	1.000	1.000	1.000
Windows10_sS_O	0.999	1.000	0.999
Average	0.968	0.958	0.957

Table 15. Naïve Bayes NetFlow v9 dataset confusion matrix

a	***b***	***c***	***d***	***e***	***f***	***g***	***h***	***Classified***
294972	0	13	0	158	0	11	4	***a= ubuntu_Agg_O***
0	196629	28	0	0	98567	0	7	***b = Ubuntu_O***
0	0	295018	0	0	0	67	6	***c = Ubuntu_sA_O***
0	0	21	294919	0	0	2	69	***d = Ubuntu_sS_O***
1	0	13	0	302042	0	27	158	***e = Windows10_Agg_O***
1	1	28	0	0	301341	3	17	***f = Windows10_O***
12	0	23	0	1	0	294493	28	***g = Windows10_sA_O***
1	0	24	0	0	0	6	301341	***h = Windows10_sS_O***

In **Table 16** and **Table 17** are presented the details of the Ada Boost accuracy and confusion matrix results of the NetFlow v9 dataset.

Table 16. AdaBoost NetFlow v9 dataset metrics

Labels	***Precision***	***Recall***	***F-measure***
Ubuntu_Agg_O	1.000	0.999	1.000
Ubuntu_O	0.983	1.000	0.992
Ubuntu_sA_O	1.000	1.000	1.000
Ubuntu_sS_O	1.000	1.000	1.000
Windows10_Agg_O	0.999	1.000	1.000
Windows10_O	1.000	0.983	0.992
Windows10_sA_O	1.000	1.000	1.000
Windows10_sS_O	1.000	1.000	1.000
Average	0.998	0.998	0.998

Table 17. AdaBoost NetFlow v9 dataset confusion matrix

a	***b***	***c***	***d***	***e***	***f***	***g***	***h***	***Classified***
294983	0	2	0	158	0	13	2	***a= ubuntu_Agg_O***
0	295180	9	0	0	36	3	3	***b = Ubuntu_O***
0	0	295078	0	0	0	7	6	***c = Ubuntu_sA_O***
0	0	4	294996	0	0	1	10	***d = Ubuntu_sS_O***
2	0	1	0	302184	0	44	10	***e = Windows10_Agg_O***
0	4963	5	2	0	296400	9	12	***f = Windows10_O***
0	0	31	0	1	0	294511	14	***g = Windows10_sA_O***
0	0	17	16	0	0	10	301329	***h = Windows10_sS_O***

In **Table 18** and **Table 19** are presented the details of the Cost Sensitive accuracy and confusion matrix results of the NetFlow v9 dataset.

Table 18. Cost Sensitive NetFlow v9 dataset metrics

Labels	Precision	Recall	F-measure
Ubuntu_Agg_O	1.000	0.999	1.000
Ubuntu_O	1.000	0.666	0.800
Ubuntu_sA_O	0.999	1.000	1.000
Ubuntu_sS_O	1.000	1.000	1.000
Windows10_Agg_O	0.999	0.999	0.999
Windows10_O	0.754	1.000	0.859
Windows10_sA_O	1.000	1.000	1.000
Windows10_sS_O	0.999	1.000	0.999
Average	0.968	0.958	0.957

Table 19. Cost sensitive NetFlow v9 dataset confusion matrix

a	***b***	***c***	***d***	***e***	***f***	***g***	***h***	***Classified***
294972	0	13	0	158	0	11	4	***a= ubuntu_Agg_O***
0	196629	28	0	0	98567	0	7	***b = Ubuntu_O***
0	0	295018	0	0	0	67	6	***c = Ubuntu_sA_O***
0	0	21	294919	0	0	2	69	***d = Ubuntu_sS_O***
1	0	13	0	302042	0	27	158	***e = Windows10_Agg_O***
1	1	28	0	0	301341	3	17	***f = Windows10_O***
12	0	23	0	1	0	294493	28	***g = Windows10_sA_O***
1	0	24	0	0	0	6	301341	***h = Windows10_sS_O***

Discussion

In this section, the machine learning training and testing evaluation results of the datasets are showcased in the tables above.

The average machine learning scores for the IPFIX datasets are showcased in Table 6, with the performance results for Naïve Bayes, Ada Boost, and Cost Sensitive algorithms being 84.9%, 91.5%, and 84.6%, respectively. Table 8 presents the Naïve Bayes results of the IPFIX dataset, with the average score of label classification for Ubuntu and Windows to be 82.9 % and 94.8 %, respectively. Table 10 presents the Ada Boost results of the IPFIX dataset, with the average score of label classification for Ubuntu and Windows to be 93.8 % and 89.2 %, respectively. Table 12 presents the Cost Sensitive results of IPFIX dataset, with the average score of label classification for Ubuntu and Windows to be 82.9 % and 94.8 %, respectively.

The average machine learning scores for the Netflow v9 datasets are showcased in Table 7 with the performance results for Naïve Bayes, Ada Boost, and Cost Sensitive algorithms being 95.8 %, 99.7%, and 95.8%, respectively. Table 14 presents the Naïve Bayes results of Netflow v9 dataset, with the average score of label classification for Ubuntu and Windows to be 99.7 % and 93.3 %, respectively. Table 16 presents the Ada Boost results of Netflow v9 dataset, with the average score of label classification for Ubuntu and Windows to be 99.5 % and 99.7 %, respectively. Table 18 presents the Cost Sensitive results of Netflow v9 dataset, with the average score of label classification for Ubuntu and Windows to be 99.7 % and 93.3 %, respectively.

Both datasets have high accuracy results with all three (3) machine learning evaluation algorithms. However, as mentioned in the prediction section , it was expected that

the Ubuntu label percentages would be lower because of the constrained responses but their results was also highly performed.

Chapter 6 - Conclusion

In this thesis, we present the creation and assessment procedures of two OS fingerprinting datasets, that assisted in the automated and precise active OS fingerprinting scan detection. The raw captured data was captured and converted to NetFlow v9 and IPFIX network flow versions, with the creation of two (2) datasets. Prior to the machine learning evaluation, pre-processing of the datasets was conducted. The raw captured data was analyzed and converted to NetFlow v9 and IPFIX network flow versions, creating two (2) datasets. Prior to the machine learning evaluation, pre-processing was conducted with the utilization of the Info Gain Attribute evaluator, a Feature Importance Ranking measure. Following the feature ranking, the machine learning evaluation was conducted. For the training and evaluation of the datasets, three (3) algorithms were utilized: Naive Bayes, AdaBoost, and CostSensitive algorithms. For the IPFIX dataset, the results showcased 84.9%, 91.5%, and 84.9% accuracy, respectively. As for the NetFlow v9 dataset, the results showcased 95.8%, 99.7%, and 95.8% accuracy, respectively.

Future Work

Regarding the future steps of this thesis, a machine learning-based network monitoring software could be built, in order to utilize this dataset, for precise active OS fingerprinting scan detection, and even mitigation. This software could be utilizing the nProbe software, since it provides real-time conversion of network traffic to Cisco’s latest network flow standards, NetFlow v9 and IPFIX. Furthermore, the chance of adding more diverse data, instead of only network-related, is considered as a next step procedure. Moreover, more network traffic could be added to the dataset to support the detection of more OSs (IOs, MacOS, Windows 11, etc).

Bibliography

[1] "Common cyber attacks." https://www.cisco.com/c/en/us/products/security/common-cyberattacks.html

[2] "Types of remote attacks." https://support.eset.com/en/kb2907-types-of-remote-attacks

[3] "Reconnaissance." https://www.usna.edu/Users/cs/wcbrown/courses/si110AY13S/lec/l32/lec.html

[4] "OS fingerprinting." https://resources.infosecinstitute.com/topic/must-know-os-fingerprinting/

[5] "Netflow v9." https://www.cisco.com/en/US/technologies/tk648/tk362/technologies_white_paper09186a00800a3db9.html

[6] "IPFIX." https://www.ciscopress.com/articles/article.asp?p=2812391&seqNum=4

[7] T. Jirsík and P. P. Pavelčeleda, "Identifying Operating System Using Flow-based Traffic Fingerprinting." [Online]. Available: http://is.muni.cz/th/359565/fi_b

[8] R. Tyagi, T. Paul, B. S. Manoj, and B. Thanudas, "Packet Inspection for Unauthorized OS Detection in Enterprises," *IEEE Secur. Priv.*, vol. 13, no. 4, pp. 60–65, Jul. 2015, doi: 10.1109/MSP.2015.86.

[9] A. Aksoy and M. H. Gunes, "Operating System Classification Performance of TCP/IP Protocol Headers," 2016, doi: 10.1109/LCNW.2016.37.

[10] M. Lastovicka and D. Filakovsky, "Passive os fingerprinting prototype demonstration," in *NOMS 2018 - 2018 IEEE/IFIP Network Operations and Management Symposium*, Apr. 2018, pp. 1–2. doi: 10.1109/NOMS.2018.8406128.

[11] M. Lastovicka, T. Jirsik, P. Celeda, S. Spacek, and D. Filakovsky, "Passive os fingerprinting methods in the jungle of wireless networks," in *NOMS 2018 - 2018 IEEE/IFIP Network Operations and Management Symposium*, Apr. 2018, pp. 1–9. doi: 10.1109/NOMS.2018.8406262.

[12] M. Lastovicka, A. Dufka, and J. Komarkova, "Machine Learning Fingerprinting Methods in Cyber Security Domain: Which one to Use?," in *2018 14th International Wireless Communications & Mobile Computing Conference (IWCMC)*, Jun. 2018, pp. 542–547. doi: 10.1109/IWCMC.2018.8450406.

[13] D. H. Hagos, M. Loland, A. Yazidi, O. Kure, and P. E. Engelstad, "Advanced Passive Operating System Fingerprinting Using Machine Learning and Deep Learning," in *2020 29th International Conference on Computer Communications and Networks (ICCCN)*, Aug. 2020, pp. 1–11. doi: 10.1109/ICCCN49398.2020.9209694.

[14] "Virtual Box." https://www.virtualbox.org

[15] "Kali Linux." https://www.kali.org/

[16] "Ubuntu." https://ubuntu.com/

[17] "Windows 10." https://www.microsoft.com/en-us/windows/windows-10-specifications

[18] "Wireshark." https://www.wireshark.org/

[19] “nProbe.” https://www.ntop.org/products/netflow/nprobe/

[20] “Pandas.” https://pandas.pydata.org/

[21] “WEKA.” https://www.cs.waikato.ac.nz/ml/weka/

[22] “SSH”, [Online]. Available: https://www.ssh.com/academy/ssh

[23] A. Zien, N. Krämer, S. Sonnenburg, and G. Rätsch, “The feature importance ranking measure,” *Lect. Notes Comput. Sci. (including Subser. Lect. Notes Artif. Intell. Lect. Notes Bioinformatics)*, vol. 5782 LNAI, no. PART 2, pp. 694–709, 2009, doi: 10.1007/978-3-642-04174-7_45.

[24] D. Gnanambal, D. Thangaraj, Meenatchi V T, and D. Gayathri, “Classification Algorithms with Attribute Selection: an evaluation study using WEKA,” *Int. J. Adv. Netw. Appl.*, vol. 09, no. 06, pp. 3640–3644, 2018.

[25] S. Raschka, “Model Evaluation, Model Selection, and Algorithm Selection in Machine Learning,” Nov. 2018, [Online]. Available: http://arxiv.org/abs/1811.12808

[26] J. Janssen and W. Laatz, “Naive Bayes,” in *Statistische Datenanalyse mit SPSS*, vol. 4, no. 1, Berlin, Heidelberg: Springer Berlin Heidelberg, 2017, pp. 557–569. doi: 10.1007/978-3-662-53477-9_25.

[27] W. Li and Q. Li, “Using Naive Bayes with AdaBoost to Enhance Network Anomaly Intrusion Detection,” in *2010 Third International Conference on Intelligent Networks and Intelligent Systems*, Nov. 2010, pp. 486–489. doi: 10.1109/ICINIS.2010.133.

[28] L. Vinet and A. Zhedanov, “A ‘missing’ family of classical orthogonal polynomials,” *Proc. Seventeenth Int. Jt. Conf. Artif. Intell.*, p. 7, Nov. 2010, doi: 10.1088/1751-8113/44/8/085201.

[29] J. Dj Novakovi, A. Veljovi, S. S. Ili, ˇ Zeljko Papi, and M. Tomovi, “Evaluation of Classification Models in Machine Learning,” *Theory Appl. Math. Comput. Sci.*, vol. 7, no. 1, pp. 39–46, 2017.