Agentless Host Intrusion Detection Using Machine Learning Techniques

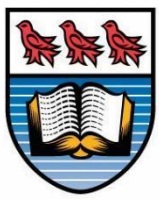
by

Jianfeng Liu

A Report Submitted in Partial Fulfillment of the
Requirements for the Degree of

MASTER OF ENGINEERING

in the Department of Electrical and Computer Engineering



University of Victoria

Agentless Host Intrusion Detection Using Machine Learning Techniques

by

Jianfeng Liu

Supervisory Committee

Dr. Issa Traore, Department of Electrical and Computer Engineering
Supervisor

Dr. Mihai Sima, Department of Electrical and Computer Engineering
Departmental Member

Abstract

With the rise in the frequency and sophistication of cyberattacks, host intrusion detection systems (HIDSs) have become an essential component in monitoring and protecting endpoints in the network security perimeter. Current HIDSs rely on a local software agent deployed on the monitored host that collects and processes or pre-processes required data. However, this architecture has adverse effects such as increased attack surface, and high maintenance cost and overhead.

Recently, a generic agentless endpoint framework that collects transparently raw data from the monitored host was proposed by Ghaleb et al [1] along with a basic threshold-based statistical model for intrusion detection as an initial proof of concept.

This report extends the generic agentless framework by collecting a new dataset with more attack vectors and developing and comparing six machine learning models, including k-nearest neighbors, logistic regression, naïve Bayes, decision tree, random forest, and support vector machine.

The experimental evaluation using the collected dataset confirmed the feasibility of agentless host intrusion detection, with increased detection efficiency and effectiveness.

Table of Contents

List of Figures

List of Tables

Glossary

ISOT: Information security and object technology Research lab, at the University of Victoria

IDS: Intrusion Detection System

HIDS: Host Intrusion Detection System

GRIDS: Graph-based Intrusion Detection System

SSH: Secure Shell Protocol

SFTP: Secure File Transfer Protocol

ARP: Address Resolution Protocol

FTP: File Transfer Protocol

Dedication

I would like to express my greatest thanks to my supervisor, Prof. Issa Traoré, for all his support, patience and guidance throughout my program.

Special thanks to my wife for all the unconditional support, caring and understanding. Also, thanks to my parents and my friends for encouraging me during the program.

Chapter 1 - Introduction

1.1 Background

In 1980, James P. Anderson creatively introduced the idea of using audit trail data to monitor intrusions and used it as a basis for proposing different intrusion detection methods [2]. From 1984 to 1986, Dorothy Denning and Peter Neumann introduced and implemented a statistical model for real-time intrusion detection [3]. In 1988, Teresa Lunt improved Dorothy Denning's model by creating the Intrusion Detection Expert System (IDES） model and platform [4], a system for detecting intrusions on a single host. After this, the GRIDS system, which is more suitable for dealing with large-scale host detection and automated detection, was created and developed [5]. In recent years, network security has evolved rapidly, and more and more solutions incorporating machine learning techniques have been implemented in intrusion detection, such as support vector machine (SVM), neural network approaches, etc.

The detection engines of current intrusion detection systems can generally be divided into misuse detection and anomaly detection.

The misuse detection approach is based on known attack characteristics. A library of attack models is created for existing attacks so that the monitoring data can be matched with the model library. The correct representation of attack features in the model library is the key to misuse detection, and it is also important to keep it up to date with new vulnerabilities. In addition, it has an obvious disadvantage in that only known attacks can be detected, so it would miss novel attacks.

The anomaly detection approach is behavior-based detection, which involves establishing the normal behavior profiles of the authorized users and then comparing the newly collected data with the normal behavior profiles to determine whether there is an attack. The anomaly detection approach is general in the sense it is not dependent on prior knowledge of known attacks. So, it has greater potential to detect novel attack patterns. However, it has the disadvantage that it may judge the special behavior of some normal users as attacks, which may easily generate false positives.

Given this, machine learning techniques provide a proven engineering way to achieve robust and effective detection systems. Since machine learning algorithms can explore deeper connections between input features and make fuller use of information, machine learning-based detection often shows improved accuracy and can automate the analysis of

unknown attacks.

Intrusion detection systems can also be categorized into host-based and network-based depending on whether the data used for detection is local to the endpoint or host under monitoring, or whether it is sourced from the network traffic logs. The former is referred to as host IDS (HIDS) while the latter is called network IDS (NIDS). The project presented in this report focused on HIDS.

1.2 Context

HIDS acts as a monitor and analyzer of the computer system; it does not act on the external interface, but focuses on the internal host, monitoring the dynamic behavior of all or part of the host, as well as the state of the entire computer system.

The traditional HIDS is installed on the host to be protected, relying on the inherent logging and monitoring capabilities of the host. This creates significant overheads on the host, reducing the efficiency of the application and its reliability, causing unstable performance impact on the running business systems. In addition, there is a serious possibility that agent-based HIDS could be compromised, e.g., the agent could become a Trojan, resulting in the loss of the entire network.

To address the limitations of agent-based HIDS, some efforts have been made recently toward developing and deploying agentless alternatives.

The first advantage of Agentless HIDS is that it is easy to set up, you only need to add the target machine to the configuration file [6]. Secondly, it requires low server performance, the data is collected and sent back to the server for analysis, and there would be almost no impact on the monitored hosts. Thirdly, there is no adaptation problem. Because there is no need to install agents or develop specific agents for different types of hosts.

1.3 Project Objectives and Approach

One of the few works available on agentless HIDS was contributed by Ghaleb et al. at the Information Security and Object Technology (ISOT) Lab [1]. In this work, the data

collection architecture and implementation for a novel agentless HIDS were proposed. The architecture provided a framework in which different machine learning algorithms can be implemented and integrated for intrusion detection. However, in the initial proof of concept proposed by Ghaleb et al. [1], only a bare-bone proof of concept analytical model was developed. The model is a simple statistical threshold-based model.

The purpose of the project presented in this report was to extend the initial framework by exploring and implementing selected machine-learning models for agentless host intrusion detection.

Using the framework, a new dataset was collected by executing different attacks. Using the dataset, six different machine learning models were developed and compared, including k-nearest neighbors, logistic regression, naïve Bayes, decision tree, random forest, and support vector machine. The obtained evaluation performance in terms of accuracy and overhead helped support the claim that agentless host intrusion is feasible.

1.4 Report Outline

The remaining chapters in this report are structured as follows.

Chapter 2 gives an overview of the agentless HIDS architecture, as well as the criteria for data collection and the process.

In Chapter 3, the collected datasets, feature model, and trained machine learning techniques are presented. The evaluation results for each technique are also presented and discussed.

Chapter 4 contains concluding remarks and shortcomings, as well as an outline of future work.

Chapter 2 - Feature Model and Data Collection

2.1 Existing Agentless HIDS Framework

The project leverages the agentless HIDS framework developed by Ghaleb et al. [1]. The framework provides an infrastructure for host data collection without relying on an agent. The collected data is processed by machine learning models deployed on a remote server to detect intrusions in the host or endpoint under monitoring.

Figure 2.1 describes the architecture and workflow of the framework.

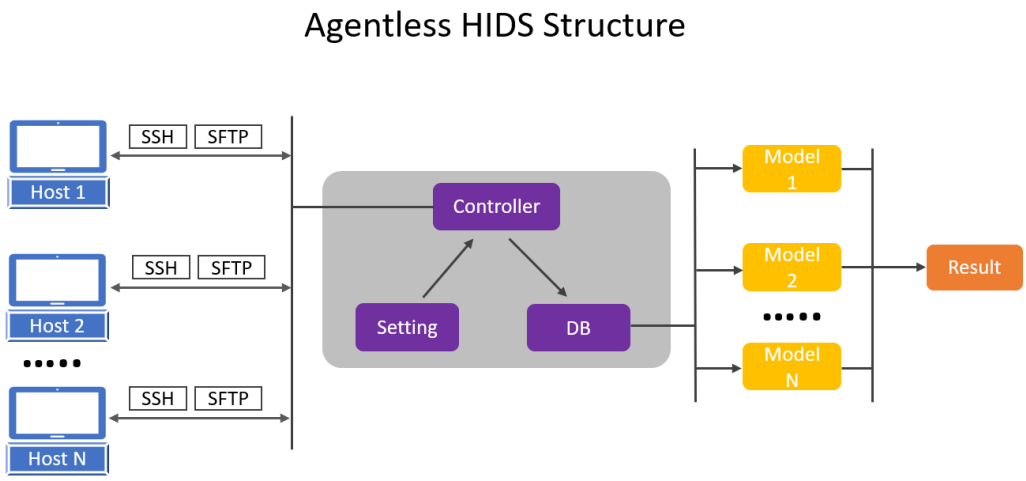


Figure 2.1 Agentless HIDS Architecture [1]

The framework supports both SSH and SFTP protocols. The SSH protocol is used to connect the intrusion detection server to the monitored host, and the SFTP protocol is used to transfer the commands and script, and then transfer the collected data back. Firstly, the IP address of the host is specified in a configuration file called hosts. conf. SSH connects to the monitored host, provides two authentication methods using user ID and password or using a private key, and then transmits the collected data back to the server securely over the network after passing the commands to the monitored host. The data is stored in a MongoDB database.

The controller is the heart of the framework. It handles core settings such as data

collection frequency, interval time, host configuration, etc. The resulting dataset is then passed to the machine learning algorithm models, which make intrusion decisions.

The underlying data collection workflow is described by Figure 2.1. Firstly, we need to specify some host IP addresses in the configuration file hosts.conf, which are the monitored machines for the whole architecture and the source of data. The second step is to specify what data is to be collected, which is specified in the configuration file collect.py. The third step is to select the type of authentication to be used, username password login or by using SSH keys. In the fourth step, execute the command script to collect the data. Finally, store the data in the database.

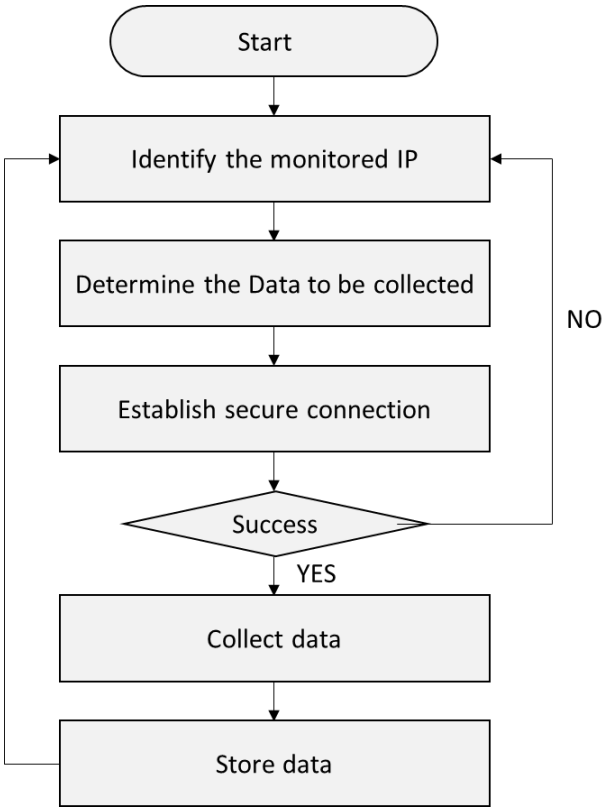


Figure 2.2 Data Collection Flow

2.2 Data Source and Feature Model

Many different types of data can be collected on the host being monitored, such as audit records (e.g., syslog), system calls, and commands issued at a terminal. In this project, we focused on the latter type of data.

The commands executed on the monitoring system were obtained by a python script, using the psutil library, changing the original way of storing bash commands into a sh file and executing it after passing to the target host. The data extracted from the commands fall into five commonly used dimensions, namely, CPU, memory, I/O, disk, and network.

Psutil is a cross-platform library that makes it easy to obtain information about the processes running on the system and system utilization (including CPU, memory, disk, network, etc.) [7]. It is mainly used for system monitoring, analysis of system resources and process management. It can implement functions equivalent to those provided by command line tools such as ps, top, lsof, netstat, ifconfig, who, df, kill, free, nice, ionice, iostat, iotop, uptime, pidof, tty, taskset, pmap, etc.

The library supports mainstream operating systems, such as Linux, Windows, Mac OS X, etc. The advantage of the psutil approach is that the data are based on calculated values, which already have practical analysis value, and can be applied directly to the data model without further processing of the raw data after acquisition. Each data item is a feature, and our proposed feature model derived directly from the data items consists of 21 features shown in Table 2.1.

Features	**Description**
virtual_memory().percent	Statistics about system memory usage percentage
swap_memory().percent	Statistics about system memory usage percentage
cpu_percent(interval=5)	A float representing the current system-wide CPU utilization as a percentage
cpu.user	Time spent by normal processes executing in user mode; on Linux this also includes **guest** time
cpu.system	Time spent by processes executing in kernel mode
cpu.iowait	Time spent waiting for I/O to complete
cpu.softirq	Time spent for servicing software interrupts
cpu_freq().current	CPU frequency as a named tuple including *current*, *min* and *max* frequencies expressed in Mhz
pid_sleeping	Number of sleeping processes
pid_idle	Number of idle processes
pid_running	Number of running processes
pid_stopped	Number of stopped processes

Features	Description
pid_blocked	Number of blocked processes
bytes_sent	System-wide network I/O statistics in number of bytes sent
bytes_recv	System-wide network I/O statistics in number of bytes received
packets_sent	System-wide network I/O statistics in number of packets sent
packets_recv	System-wide network I/O statistics in number of packets received
disk_usage('/').percent	Disk usage percentage statistics about the partition
disk_io_counters().read_time	System-wide disk I/O statistics in time spent reading from disk (in milliseconds)
disk_io_counters().write_time	System-wide disk I/O statistics in time spent writing from disk (in milliseconds)
disk_io_counters().busy_time	System-wide disk I/O statistics in time spent doing actual I/Os (in milliseconds)

Table 2.1. Proposed features and their descriptions.

2.3 Data Collection

2.3.1 Normal Activities

To compare the attacked state, we need some time to obtain the normal usage state. I simulated the regular use of the computer, including editing files, browsing web pages, opening multiple software, and so on to obtain data in this category. The corresponding performance indicators were recorded during this process.

In addition to data of normal usage state, in the process of browsing web pages, some performance fluctuations which are somehow similar to the state of being attacked may occur. For example, slow website loading and system lagging will result in high memory usage of the machine. Such a phenomenon is a common state in regular usage as well. I also simulated this type of computer performance fluctuation to collect relevant data.

2.3.2 Simulated Attacks

In addition to normal data, this project also uses simulated attacks to capture changes in monitoring data. The simulated attacks were launched against the monitored host using the kali platform.

Kali is a distribution of Linux derived from Debian and specifically designed for computer forensics and penetration testing [8]. Several commonly used attack and digital forensics tools are pre-installed in Kali Linux.

To collect sample attack data, I executed 3 different types of attacks against the target network. All three attacks are resource-consuming attacks, which are carried out against the target from different ports and protocols. This enables the target performance to change and facilitates data collection.

DOS Attack

A Denial of Service (DoS) attack, depicted in Figure 2.3, is a network attack in which a malicious actor makes a computer or other device unavailable to authorized users by interrupting the normal functionality of the device. A DoS attack typically denies service to other users by attempting to overwhelm or flood the target computer until it cannot handle normal traffic. This type of attack is also the best choice for detecting performance changes. In this project, the attacked port is based on the docker open web port 8080. I executed a syn flood attack which is a well-knows dos attack. After the attack is launched, the target being attacked will keep trying to respond to the received syn packet. But the target will not get any response after that, and finally, run out of resources, indicating that the purpose of dos was achieved. This attack was launched for a total of 42 minutes yielding CPU and memory consumptions of 89% and 100%, respectively.

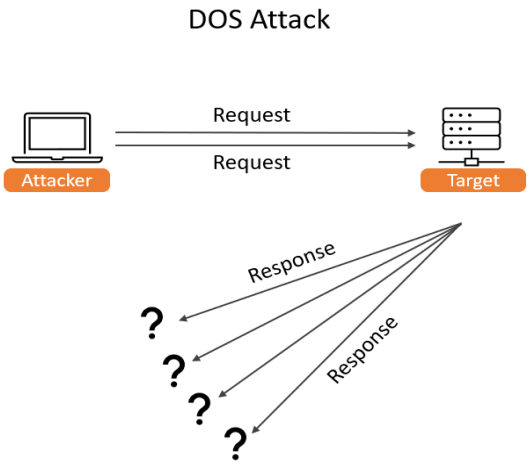


Figure 2.3 DOS Attack Outline

ARP Spoofing Attack

ARP spoofing attack, depicted in Figure 2.4, consists of forging IP and MAC addresses, which can generate a large amount of ARP traffic in the network to block the network. The attacker can change the IP-MAC entries in the ARP cache of the target host by continuously sending out forged ARP response packets, causing network outages or man-in-the-middle attacks. In this project, I used the arpspoof tool in kali as an implementation of the attack, and this attack lasted about 10 minutes. The outcome of the attack was that the target machine could not access the Internet normally.

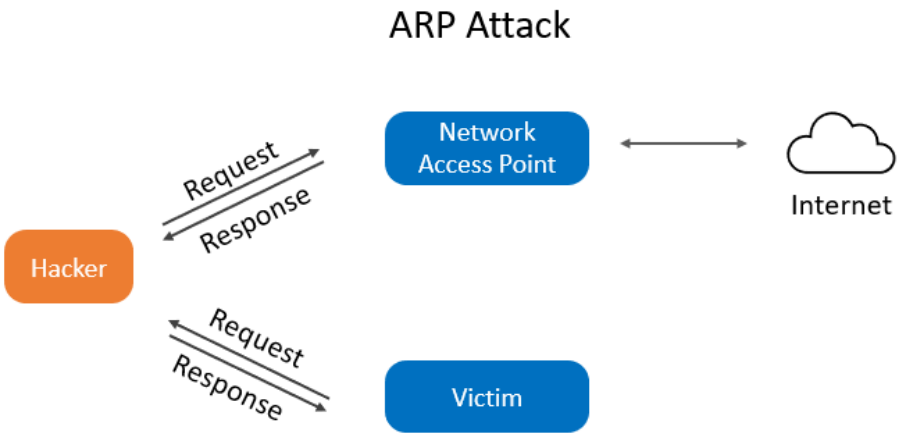


Figure 2.4 ARP Attack Outline

Password Guessing Attack Through Brute Force

A brute force attack (outlined in Figure 2.6) uses trial-and-error to guess login information. Hackers work through all possible combinations hoping to guess correctly. In this project, the hydra tool was used to perform a blast attack on both SSH and FTP protocols, respectively. The loaded dictionary type is one of the default dictionaries called common.txt in kali, the size of the dictionary is shown in figure 2.6, and the duration of this attack was about 14 minutes.

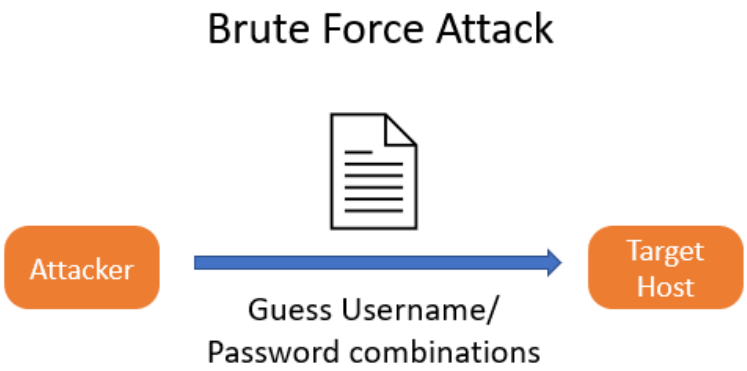


Figure 2.5 Brute Force Attack Outline

```
root@kali:/usr/share/wordlists/dirb# cat common.txt |wc
   4614    4617   35849
```

Figure 2.6 Brute Force Dictionary

2.3.3 Data Collection Environment and Process

The setup components for the network environment used for data collection and attack simulation are depicted in Figure 2.7.

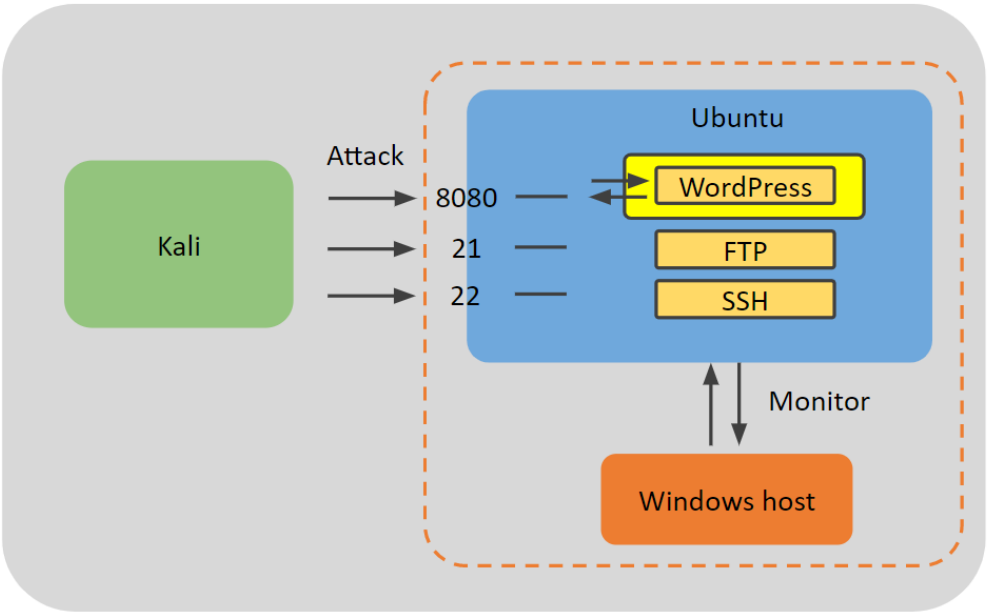


Figure 2.7 Experimental Setup Components

The following components are involved:

- The monitored host Operating System: Ubuntu
  - On this machine, we install the WordPress server through docker.
  - To get started with the Docker engine on Ubuntu, we chose the special version 64-bit 22.04 for Ubuntu.
  - Also, SSH and FTP services were running on this host.
- Attack Operating System: Kali
- HIDS server Operating System: Windows

The data collection is outlined in Figure 2.8.

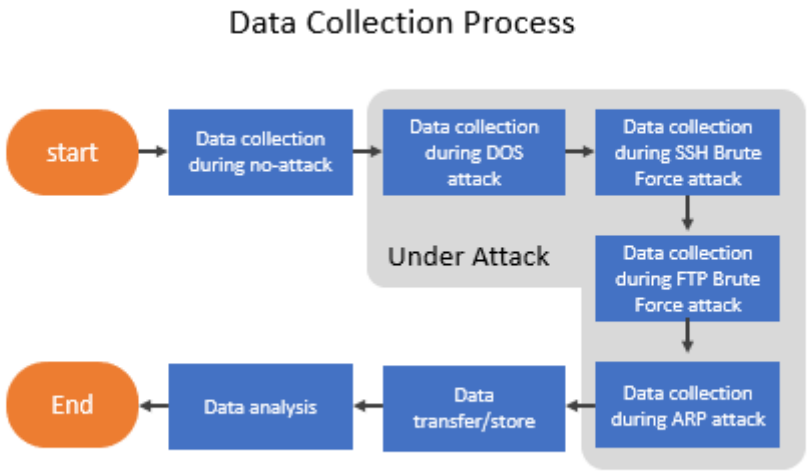


Figure 2.8 Data Collection Process

As the figure shows, data was collected initially under normal behavior, and then under four types of attacks. After a period of collection, the data was transferred to the server and then analyzed. The data collection for the whole process lasted for a total of 180 minutes, and a total of 366 blocks or samples of data were recorded.

2.4 Collected Data

To acquire data, the framework's server passes the python script to the monitored host via SFTP, and this acquisition is set to execute once every minute, and then pass the data back to the server.

The collected data from the different activities and scenarios were labeled and merged into one data set.

For labeling, a value of 1 was assigned to the data samples produced during the attack period and a value of 0 to the data samples produced during the normal period.

The distribution of the merged dataset into normal and attack samples is depicted in Figure 2.9.

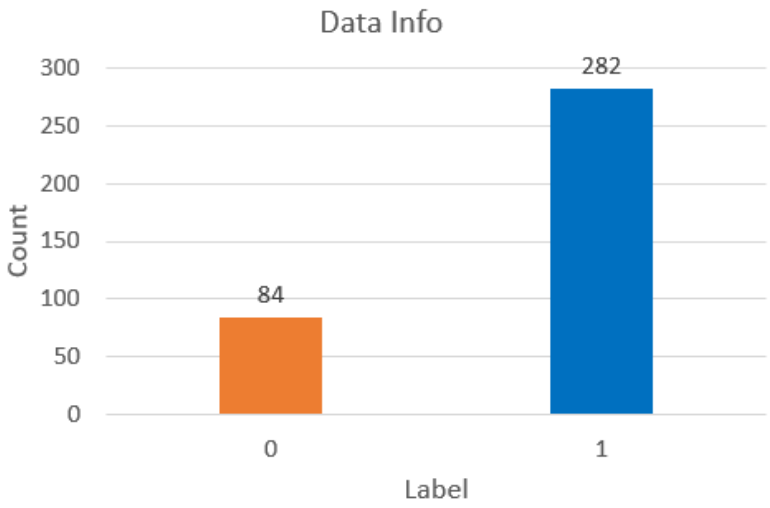


Figure 2.9 Collected Data Breakdown

Chapter 3 - Detection Models Design and Evaluation

3.1 Performance Metrics

A true positive (TP) occurs when an attack sample is correctly predicted as an attack. A false positive is when a normal instance is falsely classified as an intrusion. A false negative (FN) occurs when an actual attack is missed by the detection model. A true negative (TN) is when a normal sample is correctly categorized as normal.

Confusion matrices are visualization tools, which provide a visual breakdown of the classification performance based on the different classes involved in the data.

Table 3.1 shows the structure of a confusion matrix.

Confusion Matrix		Real Value	
		Positive	Negative
Predict Value	Positive	TP	FP (Type II)
	Negative	FN (Type I)	TN

Table 3.1 Confusion Matrix Structure

From the confusion matrix, other metrics can be calculated, including the following:

True Positive Rate (**TPR**): This is the percentage of correctly predicted attack samples over all attack samples, and naturally, the larger the percentage, the better. TPR is also known as recall.

$$TPR = \frac{TP}{TP + FN}$$

False Positive Rate (**FPR**): This is the percentage of normal instances that are falsely

categorized as attacks out of all normal instances. This percentage is expected to be as small as possible.

$$FPR = \frac{FP}{FP + TN}$$

Precision: This is the percentage of correctly predicted attacks out of all the predicted attacks.

$$Precision = \frac{TP}{TP + FP}$$

F-Score helps to measure Recall and Precision at the same time. It uses the Harmonic Mean in place of the Arithmetic Mean by punishing the extreme values more. **F1** is calculated by the following formula:

$$F1 = \frac{2 \times Recall \times Precision}{Recall + Precision}$$

A receiver operating characteristic (ROC) curve plots TPR vs. FPR at different classification thresholds. Lowering the classification threshold classifies more items as positive, thus increasing both False Positives and True Positives.

3.2 Classification Models

We explored 6 different machine learning classification techniques, namely, naïve Bayes (NB), k nearest neighbors (k-NN), logistic regression (LR), decision tree (DT), random forest (RF), and support vector machine (SVM). Each of the classifiers was trained using 80% of the dataset while the remaining 20% was used for testing.

3.2.1 Using Naïve Bayes

Based on Bayes' theorem, several Naive Bayes methods have been proposed and

implemented in supervised learning problems, especially in text classification and spam filtering [9]. In this project, the standard Bayesian algorithm, which assumes that the data attributes are independent of each other, was used. By training the NB classifier on 80% of the dataset and testing it on the remaining 20%, the performance measures were computed. Figure 3.1 and Table 3.2 show the obtained confusion matrix and performance measures, respectively.

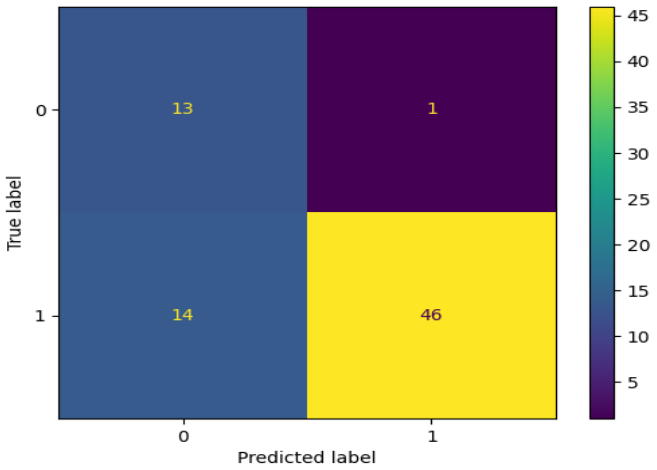


Figure 3.1 Confusion Matrix of NB Model

Model	Accuracy	Precision	TPR (Recall)	FPR	F1	Time
NB	79.73%	97.87%	76.67%	7.14%	85.98%	0.011992s

Table 3.2 Performance of NB Model

3.2.2 Using k-NN

The K-Nearest Neighbor (k-NN) classification algorithm is one of the most straightforward data mining techniques and is a well-known statistical method for pattern recognition, which occupies a significant place in machine learning classification. For this report, I also tried using k-NN for computing the classification.

Figure 3.2 shows the confusion matrix obtained by training and running k-NN on the dataset. Table 3.3 shows the obtained performance measures.

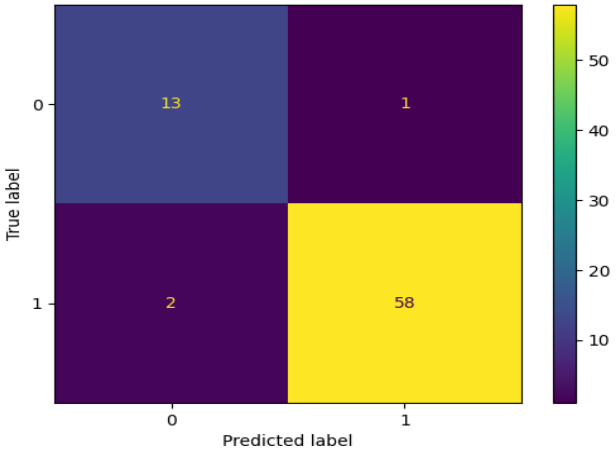


Figure 3.2 Confusion Matrix of k-NN Model

Model	Accuracy	Precision	TPR (Recall)	FPR	F1	Time
k-NN	95.95%	98.31%	96.67%	7.14%	97.48%	0.242154s

Table 3.3 Performance of k-NN Model

3.2.3 Using Logistic Regression

Logistic Regression is a very widely used machine learning algorithm that fits data to a logistic function, which enables to accomplish prediction of the probability of an event occurring. Logistic regression is not a regression model, but a model used to deal with problems labeled as dichotomous. The essence of the classification problem transforms the features and labels of the training data into a decision surface.

Figure 3.3 shows the confusion matrix, obtained by training and running logistic regression on the dataset. Table 3.4 summarizes the corresponding performance measures.

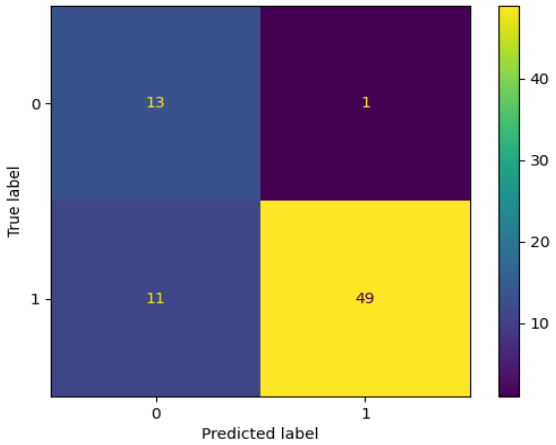


Figure 3.3 Confusion Matrix of Logistic Regression Model

Model	Accuracy	Precision	TPR (Recall)	FPR	F1	Time
LR	83.78%	98.00%	81.67%	7.14%	89.09%	0.029499s

Table 3.4 Performance of Logistic Regression Model

3.2.4 Using Decision Tree

Decision Tree is one of the most widely used machine learning algorithms for solving supervised learning problems. It is a distribution-free or non-parametric method, which does not depend upon probability distribution assumptions [11]. Decision trees can handle high-dimensional data with good accuracy. For this classification, I used the CART decision tree which is the default. Figure 3.4 shows the confusion matrix, respectively, obtained by training and running decision tree on the dataset. Table 3.5 summarizes the computed performance measures.

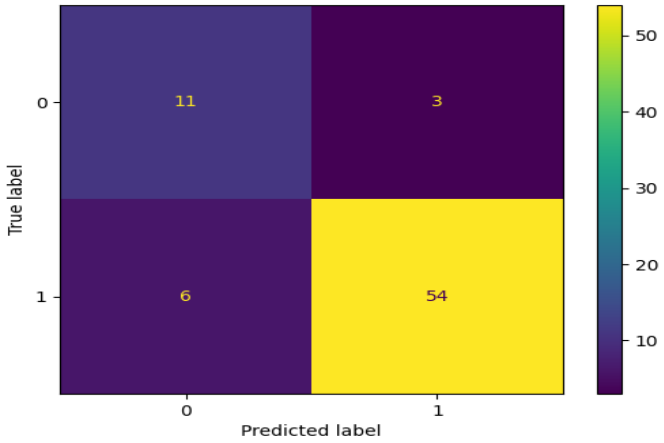


Figure 3.4 Confusion Matrix of Decision Tree model

Model	Accuracy	Precision	TPR (Recall)	FPR	F1	Time
DT	87.84%	94.74%	90.00%	21.43%	93.22%	0.006990s

Table 3.5 Performance of Decision Tree model

3.2.5 Using Random Forest

Random Forest is an algorithm that integrates multiple decision trees through the idea of integrated learning, and its basic unit is the decision tree. When a classification task is performed, new input samples are entered, and each decision tree in the forest is allowed to judge and classify them separately, and each decision tree will get a classification result of its own. Figure 3.5 shows the confusion matrix, obtained by training and running random

forest on the dataset. Table 3.6 presents the corresponding performance measures.

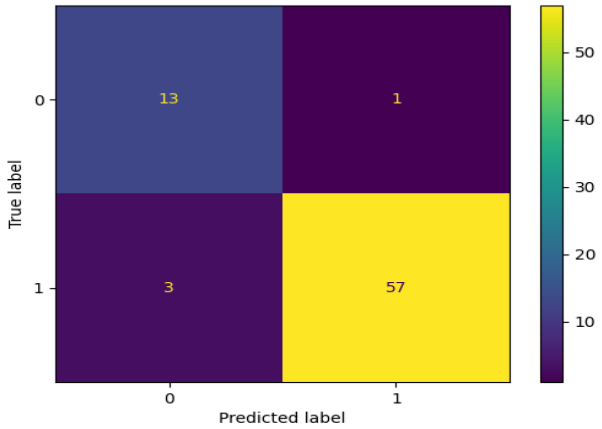


Figure 3.5 Confusion Matrix of Random Forest model

Model	Accuracy	Precision	TPR (Recall)	FPR	F1	Time
RF	94.59%	98.28%	95.00%	7.14%	95.73%	0.189889s

Table 3.6 Performance of Random Forest model

3.2.6 Using Support Vector Machine

SVM is a machine learning method based on statistical theory, which is suitable for algorithms in the case of finite samples [12]. It improves the generalization ability by the principle of structural risk minimization, uses kernel functions to avoid the complex computation of high-dimensional feature space, and can solve the problems of small samples, nonlinearity and high dimensionality better. Applying this algorithm to intrusion detection has high detection accuracy and dominates in training speed.

Figure 3.6 shows the confusion matrix, obtained by training and running SVM on the dataset. Table 3.7 shows the computed performance measures.

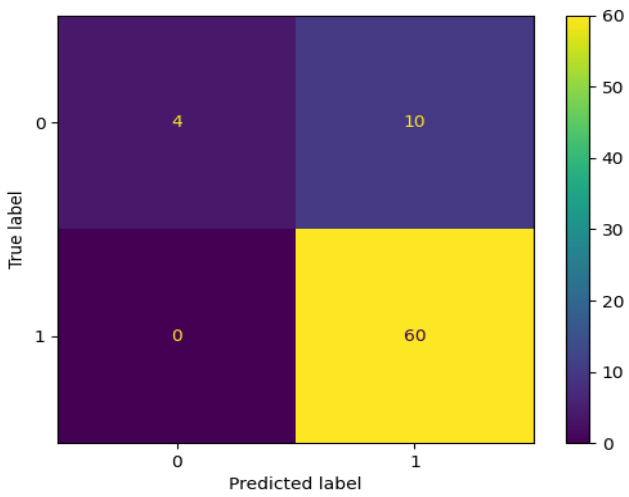


Figure 3.6 Confusion Matrix of SVM Model

Model	Accuracy	Precision	TPR (Recall)	FPR	F1	Time
SVM	86.49%	85.71%	100.00%	71.43%	92.31%	0.049971s

Table 3.7 Performance of SVM Model

3.3 Comparison of the Selected Models

Table 3.8 presents the accuracy and evaluation metrics of precision, recall and F1 for each model. Time overhead for each model is also included.

Model	Accuracy	Precision	TPR (Recall)	FPR	F1	Time
NB	79.73%	97.87%	76.67%	7.14%	85.98%	0.011992s
k-NN	95.95%	98.31%	96.67%	7.14%	97.48%	0.242154s
LR	83.78%	98.00%	81.67%	7.14%	89.09%	0.029499s
RF	94.59%	98.28%	95.00%	7.14%	95.73%	0.189889s
DT	87.84%	94.74%	90.00%	21.43%	93.22%	0.006990s
SVM	86.49%	85.71%	100.00%	71.43%	92.31%	0.049971s

Table 3.8 Comparison of Model Performance

By comparing the results of the different models in the table, we can see that the k-NN model outperformed the other models, achieving the highest accuracy rate, although it also took the longest time. Furthermore, k-NN also achieved the highest TPR at 96.67% with the lowest FPR of 7.14%. The random forest model showed second performance, achieving

almost the same high accuracy and TPR as the former, but with less time. Decision tree model, logistic regression model and Support Vector Machine model showed similar performance at accuracy, with an accuracy rate of about 85%, while DT and SVM had higher FPR. Naive Bayes model whose accuracy was the worst (just below 80%) among the six models, showed worst performance.

Chapter 4 - Conclusion

With the rapid development of Internet technology, information security is receiving more and more attention. Intrusion detection is an indispensable part of information security.

Although it is challenging to implement in practice, agentless HIDS represents the future of IDS technology. Agentless deployment of HIDS helps to address some of the limitations of existing traditional detection solutions, such as increased attack surface and high maintenance costs.

In this project, we have developed and compared six different machine-learning models for agentless host intrusion detection. The focus of the study was the comparison between the different models. We aimed to provide a perspective for choosing a better model. The k-NN classifier turned out to be the most effective in terms of detection accuracy, however, it is the least efficient in terms of overhead.

Despite the encouraging results obtained in the project, there are still some aspects of the detection system that need to be improved. This is also the direction of future work.

Firstly, the current approach of reading the configuration file to obtain the authentication information of the monitored machine faces the risk of data leakage. In the future, we will upgrade the architecture so that the configuration information can be read by using a configuration center and the authentication information will be no longer hard-coded (in case of password-based authentication). This makes authentication information more secure and less likely to be leaked.

Secondly, in the current industrial environment, access to the servers needs to go through a bastion host, thus the servers cannot access each other by SSH protocol, i.e., the SSH ports of servers are only accessed by the bastion host. However, agentless communication in the current architecture is based on PSSH. PSSH, which stands for parallel-ssh, is a non-blocking parallel SSH client library, and requires SSH access, which may conflict with the actual industrial environment. Thus, a whitelist needs to be set specifically for the monitoring server in the actual application. This enables the monitoring server to have direct access to the SSH port and services of the monitored machine.

Finally, since the hosts were simulated with virtual machines on my personal computer in this experiment, the performance of these virtual machines is limited, and the performance metrics vary for smaller changes. The monitored data are more sensitive to fluctuations,

normal activities may also lead to performance measures becoming like the state when being attacked. In the future, we could use high-performing servers in the production environment, richer information can be collected, and more precise models can be trained.

In conclusion, the agentless host intrusion detection system with machine learning detection engine is a more intelligent, accurate and flexible intrusion detection solution that can effectively improve network security. Furthermore, optimization and tuning of each algorithm are also required to achieve better performance in the future. More information about the monitored host can be added to the preset data features, and the type of simulation attack is another direction that can be explored.

References

[1] Asem Ghaleb, Issa Traore, Karim Ganame (2019). A Framework Architecture for Agentless Cloud Endpoint Security Monitoring. 2019 IEEE Conference on Communications and Network Security (CNS) Communications and Network Security (CNS). :1-9 June, 2019

[2] JAMES P ANDERSON.Co. (1980). Computer Security Threat Monitoring and Surveillance[C]. Technical Report, Fort Washington: Pennsylvania, pp. 4-5

[3] DOROTHY E DENNING. (1987). An Intrusion Detection Model[P]. IEEE Transactions on Software Engineering, 13(2), pp. 229-232.

[4] LUNT T, JAGANNATHAN R. (1998). A Prototype Real-time Intrusion-detection System[P]. IEEE Symposium on Security and Privacy, Oakland, CA, pp. 108-113

[5] Staniford-Chen, S., Cheung, S., Crawford, R., Dilger, M., Frank, J., Hoagland, J., Levitt, K., Wee, C., Yip, R. and Zerkle, D., 1996, October. GrIDS-a graph based intrusion detection system for large networks. In Proceedings of the 19th national information systems security conference, Vol. 1, pp. 361-370.

[6] Joe “monitoring-network-devices-wazuh-hids”. Accessed on: Oct 17th 2022 [online] https://wazuh.com/blog/monitoring-network-devices-wazuh-hids/

[7] psutil documentation, Accessed on: Nov 4th 2022 [online] https://psutil.readthedocs.io/en/latest/

[8] Kali Linux, Accessed on: Nov 4th 2022 [online] https://en.wikipedia.org/wiki/Kali_Linux

[9] Lars Buitinck, Gilles Louppe, Mathieu Blondel, Fabian Pedregosa, Andreas Mueller, Olivier Grisel, Vlad Niculae, Peter Prettenhofer, Alexandre Gramfort, Jaques Grobler, Robert Layton, Jake VanderPlas, Arnaud Joly, Brian Holt, and Gaël Varoquaux. Complement Naive Bayes, Accessed on: Oct 21th 2022 [online] https://scikit-learn.org/stable/modules/naive_bayes.html#complement-naive-bayes

[10] Trstenjak, B., Mikac, S. and Donko, D. (2014). KNN with TF-IDF based framework for text categorization. Procedia Engineering, 69, pp.1356-1364.

[11] Lei Shi, Mei Weng, Xinming Ma, and Lei Xi. (2010). Rough set-based decision tree ensemble algorithm for text classification. Journal of Computational Information Systems, 6(1), pp. 89–95

[12] Cortes, C. and Vapnik, V. (1995). Support vector networks[J]. Machine Learning, 20, pp. 273-297