FYP Final Report

Online Machine Learning-based Framework for Network Intrusion Detection

by

Atif Khurshid and Ghous Ali Khan

Advised by

Prof. LEE Pak Ching Patrick

ESTR 4999 Final Year Thesis

Department of Computer Science and Engineering

The Chinese University of Hong Kong

2018-19

[Blank Page]

Abstract

Intrusion detection systems are an essential part of any network in the era of ever-growing internet use and misuse. Machine Learning algorithms have been shown to be useful in intrusion detection because of their expertise in pattern recognition and ability to generalize from small number of attack pattern samples. However, such implementations suffer from the lack of real-time protection or high false positive rates. We propose an anomaly-signature hybrid, network intrusion detection system employing online machine learning algorithms. This implementation allows for real-time operation while enabling the system to learn to defend against novel attacks on-the-go.

This report introduces and discusses the significance of intrusion detection systems in the context of ever-increasing malicious actors on the internet and the theoretical framework of intrusion detection. It presents background information regarding several machine learning algorithms considered for use in the system. Furthermore, it contains detailed design and evaluation of the system and its components.

Table of Contents

Table of Figures

1. Introduction

1.1 Motivation

Since the advent of modern computing and the internet, we have become more and more reliant on the use of computers for data storage and processing. Emergence of ground-breaking technologies such as cloud computing has further amplified this inclination. The data transmitted over the internet and stored in such remote storages is extremely valuable. The rapid inflation in the value associated with data has attracted a large number of cyber criminals launching cyber-attacks all over the globe. With every passing day, these attacks are becoming more pervasive and more sophisticated in their nature.

Figure 1 shows the remarkable increase in the intensity of Distributed Denial of Service (DDoS) attacks between 2007 and 2016. [1] The tremendous growth in attack volume during the year 2016 is quite prominent. Furthermore, the 2018 attack on GitHub, a popular software development platform, recorded an astonishing attack bandwidth of 1.35 Terabytes per second, which was more than 50% larger than the greatest attack in 2016. [26]

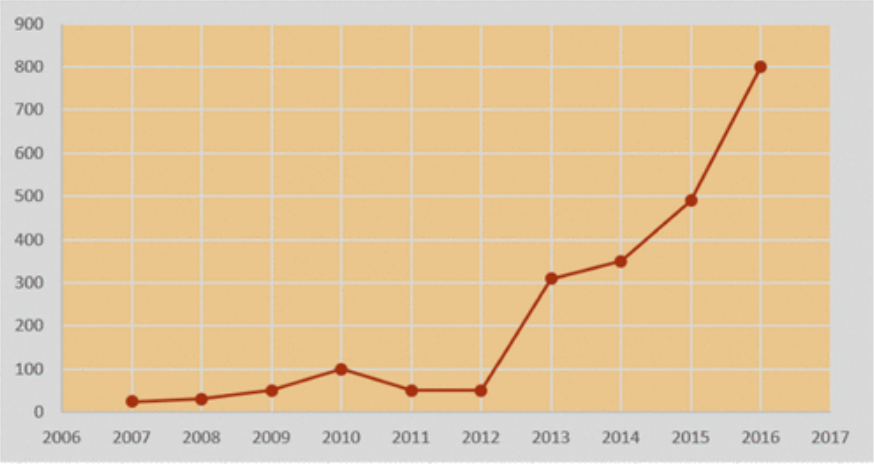

Figure 1. Volume sizes of DDoS attacks in gigabits per second, 2007-2016 [1]

It has now become a matter of utmost importance for enterprises and individuals to take measures to protect their networks. The most common approach taken by most institutions, is the development of perimeter-based network defense systems which involve surrounding the vulnerable internal network components with layers of defensive mechanisms. Figure 2 shows the layout of a typical multi layered perimeter defense system. [2]

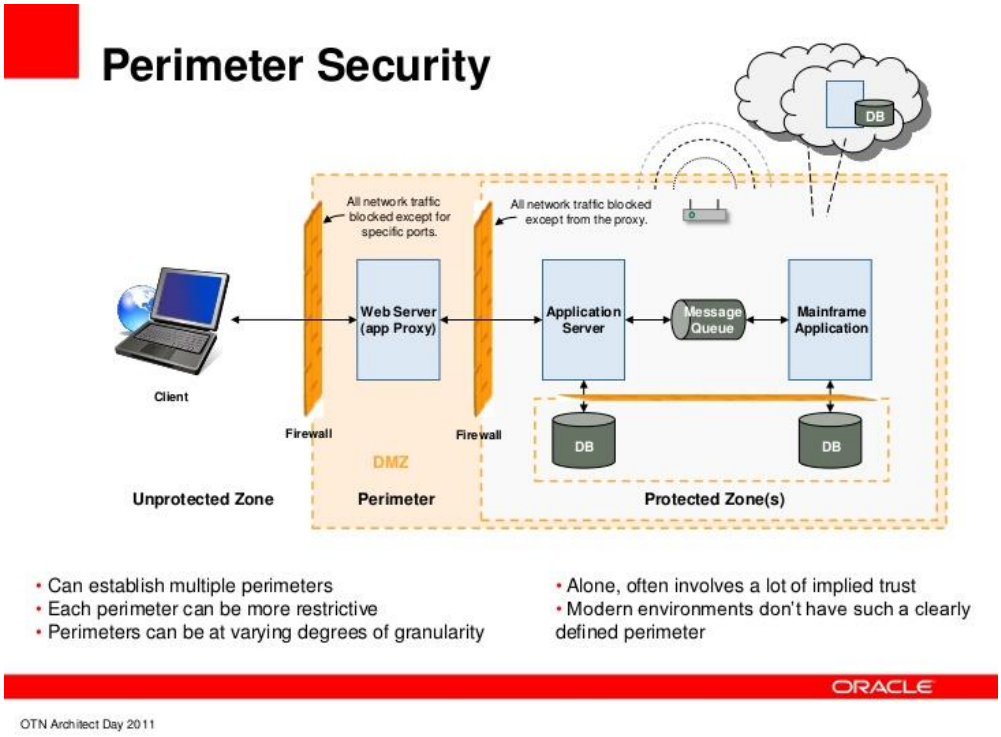

Figure 2. A typical perimeter defense system [2]

Unfortunately, it is extremely difficult, if not impossible, to develop an impregnable network. As long as a network is connected to the internet, it will always remain vulnerable to some form of cyber-attack. And even if it is not, there is always the possibility of an internal user or program violating the network's security policies and protocols. This is where network intrusion detection systems become useful. These systems monitor the network and alert security administrators to malicious activities inside their security perimeter so that timely mitigatory measures may be taken.

1.2 Objective

The main objective of the project is the implementation of a network-based intrusion detection system using machine learning algorithms. We also aim to incorporate online learning in the system so that it can improve over time by automatically learning to detect new attacks.

1.3 Structure of Report

The report is structured into four sections. In the first section, we provide background information regarding intrusion detection and some machine learning algorithms. We then discuss recent research in the field in order to gain insight into the main challenges of developing such a system. In the third section, we explain the design of our system. Finally, we describe the setup and results of the experiments designed to evaluate the system.

1.4 Division of Work

The work of this project is divided along the lines of the two major components of the system: feature extraction and attack detection. Ghous Ali Khan is responsible for the implementation of the former and Atif Khurshid for the latter. The following is the distribution of work in this report:

- Ghous Ali Khan: Section 1-3 (except Section 2.2) and Section 4.2
- Atif Khurshid: Sections 4-6 (except Section 4.2) and Section 2.2

1.5 Programming Code

The Python code for this project is available at https://github.com/khisht/nids-framework.

2. Background

2.1 Intrusion detection

Intrusion detection is the procedure of detecting the activities of users or programs that are neither a part of the network, nor do they have the required authorization to become a part of the network. Intrusion detection also involves detecting internal entities who are misusing their privileges, for example a client using more resources than allocated, eavesdropping on network communication, or a student trying to use their university's GPUs to mine cryptocurrencies.

2.1.1 Formal Definition

We will now formally define an intrusion detection system.

2.1.1.1 Intrusion

Intrusion can be defined as the "malicious violation of a security policy by an unauthorized agent." [15]

2.1.1.2 Intrusion Detection

Intrusion detection is the "automated detection and alarm of any situation where an intrusion has taken place." [15]

2.1.1.3 Intrusion Detection System

A typical intrusion detection system (IDS), as shown in the figure 3, consists of at least the following components:

a. **Site Security Officer (SSO)**

   SSOs are responsible for overseeing the IDS and responding to its alarms. They manipulate configuration data to control the actions of the IDS such as its response to intrusions. They may also update reference data to introduce the system to new attacks. [15]

b. **Audit Collection**

   Audit collection is the process of accumulating data which acts as a basis for all decisions made by the IDS. The most common form of audit data, and the type used in this project, is features extracted from network logs. [15]

c. **Audit Storage**

   Audit storage encompasses the method and location of storing audit data. A good audit storage system is of great importance to an IDS because of the sheer volume of audit data. This problem is generally tackled by converting the raw, packet-based logs into connection-based feature matrices which condenses audit data substantially. [15]

d. Processing

Intrusion detection algorithms are implemented in the processing block and their intermediate results are stored in the Active/Processing data block. This component is the heart and soul of an intrusion detection system because it is responsible for the detection of attacks. [15]

e. Reference Data

Reference data forms a knowledge base used by the processing algorithm to determine the nature of a single data point. This data may include the attack signatures used for detection in a signature-based IDS or the pattern of normal communication used to allow non-malicious traffic in an anomaly-based IDS. [15]

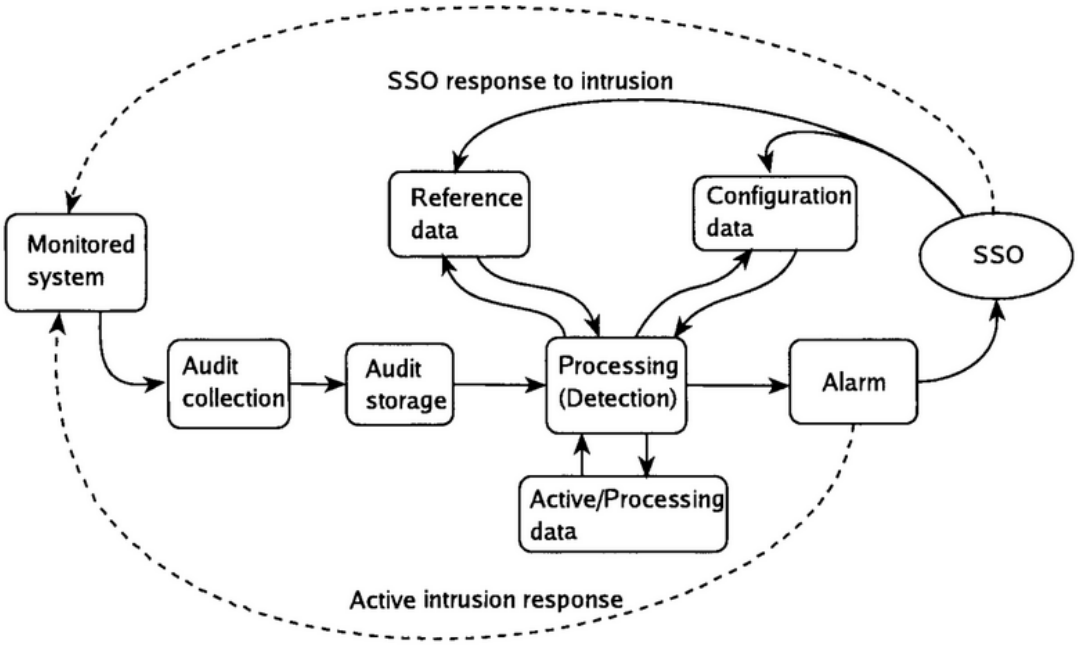

Figure 3. A typical Intrusion Detection System [15]

2.1.2 Goals

The main goals of an IDS are to ensure the following properties of communication over the network.

2.1.2.1 Confidentiality

Confidentiality literally means secrecy. It aims to prevent the data or communication inside a network from being accessed by unauthorized agents. It is the responsibility of the IDS to ensure the confidentiality of communications on a network.

2.1.2.2 Integrity

Integrity involves maintaining the consistency and trustworthiness of the data in a network. Data should not be modified as it travels across the network. Integrity is usually violated by Man in The Middle (MITM) attack, where a man, woman or even a program intercepts the data packets sent between two or more hosts, modifies its content, and sends it to the original recipients.

Integrity can also be violated when a legitimate user is allowed to modify a file that he or she doesn't have the institutional permission to modify. It is the responsibility of the IDS to detect violations of file permissions and access control policies and to detect the presence entities acting as "Man in the Middle".

2.1.2.3 Availability of resources

Availability refers to ensuring that the available system resources such as the bandwidth, processing capacity and memory resources do not fall below a certain specified threshold. Availability of the resources can be strongly affected by attacks such as Denial of Service (DoS). These attacks greatly limit a system's resources or even make them completely inaccessible to legitimate users. These attacks are sometimes used as cover to distract security administrators while malicious entities steal valuable data from the network. It is the job of an IDS to detect such attacks, preferably in the early stages, so that security administrators can respond accordingly.

Mukherjee et al. have highlighted some additional essential objectives of an effective IDS. [3] These include detection of:

- Unauthorized modification of system files or system/user information
- Unauthorized modification of network configurations such as IP tables in routers or Content Addressable Memory (CAM) tables in bridges.
- Unauthorized use of computing resources e.g. through the creation of new unauthorized accounts or unauthorized use of existing accounts.

2.1.3 Types of Intrusion Detection Systems

There are two main types of intrusion detection systems: Network-based Intrusion Detection System (NIDS) and Host-based Intrusion Detection System (HIDS). [27]

2.1.3.1 Network-based Intrusion Detection System

NIDS are concerned with network traffic. They are used to monitor internal and external network traffic to detect intrusion. They are primarily concerned with network and transport layer packet headers. But in some cases, they might analyze the packets down to application layer data, checking for states like whether the communicating user is logged in or not.

There are several advantages to using NIDS. Since NIDS run on the network instead of host systems, they don't degrade the performance of other programs. NIDS are extremely portable as they are not affected by the type of operating systems used by hosts in the network.

There are, however, several disadvantages to using NIDS as well. NIDS are not very scalable as they have to scan every packet coming into and going out of the network and in some cases, they might also have to analyze every packet exchanged within the network by internal hosts as well. Encryption can also be major problem for NIDS, as it limits the amount of information that can be extracted from the packets. [27]

2.1.3.2 Host-based Intrusion Detection System:

HIDS are used in computer systems instead of computer networks. They are primarily used to detect malicious activities within the computer. For example, a user may try to modify the files belonging to other users, a virus in the computer may try to delete all the files or a program may try to acquire the root privileges. These IDS work by analyzing system calls, CPU usage, memory usage, system logs and audit trails.

HIDS have several advantages of their use as well. Firstly, they can easily trace back the detected attack to a specific user, so that the administrator may take necessary action. HIDS remain unaffected by encryption in the network as they are deployed on hosts that have the authority to decrypt received messages.

HIDS have some disadvantages as well. These systems are not very portable since they run on top of computer architecture and operating system. Therefore, HIDS must be compatible with the operating system of the machine they are installed in. Since HIDS run on top the operating

system, they also become exposed to the security vulnerabilities of the operating system. Thirdly, since HIDS must run on a computer system to detect attacks, they limit the available system resources. [27]

2.1.4 Detection Methodologies

There are two main types of detection methodologies employed in network intrusion detection systems: anomaly-based and signature-based.

2.1.4.1 Anomaly-based NIDS

These intrusion detection systems first learn the normal behavior of a network, and anything that deviates from it is considered abnormal and therefore an attack. There can be several metrics or features used to define the norm of a computer networks. These metrics include the average volume of data traffic over a specified time, the number of packets exchanged in a connection, the number of TCP connection initiated, the average duration of a TCP connection and so on.

The normal thresholds of these metrics can be defined using complex statistical analysis making use of linear regression, Fourier analysis and time series analysis or even simply by the whim of the security administrator. It is difficult to define the normal behavior of a network, and it is even more difficult to distinguish between an innocent abnormal connection and a full-fledged attack. Therefore, Anomaly based NIDS generally have a higher false positive rate than their signature-based counterparts. They, however, are very effective in detecting zero-day attacks: attacks that have never been registered before. [6][28]

2.1.4.2 Signature based NIDS

These NIDS, as the name suggest, use unique signature or patterns to detect attacks. Most attacks have unique signatures and attributes to them. For example, the abnormal fragmentation attack used to bypass firewall rules on the Transport Layer, works by sending several small IP packets with abnormal fragmentation offset. By instructing NIDS to search for packets resembling the signature of a known attack, we can hope to successfully detect intrusion.

Signature based NIDS generally have a much better false positive rate than their Anomaly-based counterparts, but they are incapable of detecting attacks that haven’t been registered before, and sometimes even a small modification to existing attack signature is enough to trick a signature-based NIDS. [28]

2.2 Machine Learning

Due to the marked advancement in machine learning algorithms, availability of data and more sophisticated hardware, machine learning is being extensively employed in a wide array of environments. It is used to predict market trends, control pacemakers, drive cars, recognize images and to process natural languages among other things. The role of machine learning in NIDS has been extensively studied and the results so far have been quite promising. Using the KDD dataset, S. Mukkamala et al, evaluated the performance of neural networks in intrusion detection. Their best performing model was able to achieve an accuracy of 99.25% on testing data which is far better than the performance of previously discussed technologies. [4][5][7][8]

2.2.1 Supervised learning

Supervised learning is a machine learning paradigm which involves the approximation of a function based on previously seen input-output pairs. The most common application of supervised learning is in classification problems where the output is a label representing the class of the input instance. [29] Hence, supervised learning is employed in signature-based intrusion detection systems to classify benign and different types of malicious communications.

Generally, Machine learning algorithms are proficient in pattern recognition which leads to a highly accurate IDS. However, supervised learning has two major drawbacks:

- It requires an expert to provide signatures of known attacks or to label data instances for training and this is a resource-intensive and time-consuming process. [9]
- It cannot reliably detect novel attacks because of the lack of such instances in the training dataset and, therefore, requires constant feedback from the expert. [29]

2.2.2 Unsupervised learning

Unsupervised machine learning algorithms identify patterns in unlabeled input data. [30] This makes unsupervised learning ideal for anomaly detection in IDS based on the presence or absence of aforementioned patterns. The lack of the need for labelling allows IDS based on unsupervised algorithms to function without an expert and to detect novel attacks. However, these systems suffer from a high False-Positive rate since they consider all abnormal communication patterns as malicious even if they are benign. [28]

2.2.3 Offline Leaning

Offline learning is the most common method of training machine learning algorithms. It

involves a one-time training of the model which is then used for prediction. However, offline learning is inappropriate for our system because it does not allow the system to improve over time.

2.2.4 Online Learning

Online learning is a machine learning paradigm where a model is trained incrementally. Unlike other machine learning model training methods, training data is provided sequentially. This allows for the use of large training datasets and continuous improvement of a live model.

2.2.5 Batch learning

Batch learning is a compromise between offline and online training of machine learning models. The training data is divided into smaller batches and fed to the model. This allows models to be trained on huge datasets using limited computation power.

2.2.6 Naive Bayes

Naive Bayes algorithms are a set of classifiers based on Bayes Rule.

$$P(y \mid x_1, x_2, \dots, x_n) = \frac{P(y)P(x_1, x_2, \dots, x_n \mid y)}{P(x_1, x_2, \dots, x_n)}.$$

It is titled naive because of the assumption of pairwise conditional independence between all x_i, i.e.

$$P(x_i \mid y, x_1, x_2, \dots, x_n) = P(x_i \mid y).$$

For classification of an example $X = (x_1, x_2, \dots, x_n)$ into class C_k, we assign a probability

$$P(C_k \mid X)$$

for all k. By Bayes Theorem, we get

$$P(C_k \mid X) = \frac{P(C_k)P(X \mid C_k)}{P(X)}.$$

Since $P(X)$ is same for all k classes, the equation is essentially

$$P(C_k \mid X) \propto P(C_k) \prod_{i=1}^{n} P(x_i \mid C_k).$$

The class is then given by

$$\hat{y} = P(C_k) \prod_{i=1}^{n} P(x_i \mid C_k). \qquad [19]$$

The $P(x_i \mid C_k)$ is estimated by assuming that the features x_i follow a certain probability distribution. The most common distributions are Gaussian, when features are continuous, Multinomial, when features are discrete, and Bernoulli, when features are Booleans.

Naive Bayes is a simple, yet effective algorithm and most importantly, it is capable of online learning which is why we have chosen to incorporate it in our system.

2.2.7 PCA Anomaly Detection

2.2.7.1 PCA

Principal Component Analysis is essentially the projection of a k-dimensional dataset onto a p-dimensional space while conserving most of the original information in the form of variance. Mathematically, the projection is given by

$$Y = XW_p$$

where W_p contains p eigenvectors of the covariance matrix of X, Σ, corresponding to the largest eigenvalues. [24]

$$\Sigma = X^T X$$

Here, X is a n x k matrix which , after projection, results in a n x p matrix Y where $p \leq k$. The value of p is determined by the Proportion of Variance (POV) which is defined as

$$POV = \frac{\lambda_1 + \lambda_2 + \cdots + \lambda_p}{\lambda_1 + \lambda_2 + \cdots + \lambda_p + \cdots + \lambda_k}$$

where λ_i is the i-th eigenvalue of Σ arranged in descending order. [24]

POV represents the fraction of variance in Σ which has been preserved by p eigenvectors of Σ. The idea is to account for maximum amount of variance while keeping the value of p small. This allows for dimensionality reduction in the input space and improves the efficiency of ML algorithms.

Figure 4 shows the relation between POV and p for some X. This graph is useful in determining the value of p required for accurate dimensionality reduction. In this case, a value between 20 and 40 would be a good choice as almost 90% of variance is accounted for by the 20 eigenvectors of Σ corresponding to largest eigenvalues, while the smallest 30 eigenvectors provide no new information.

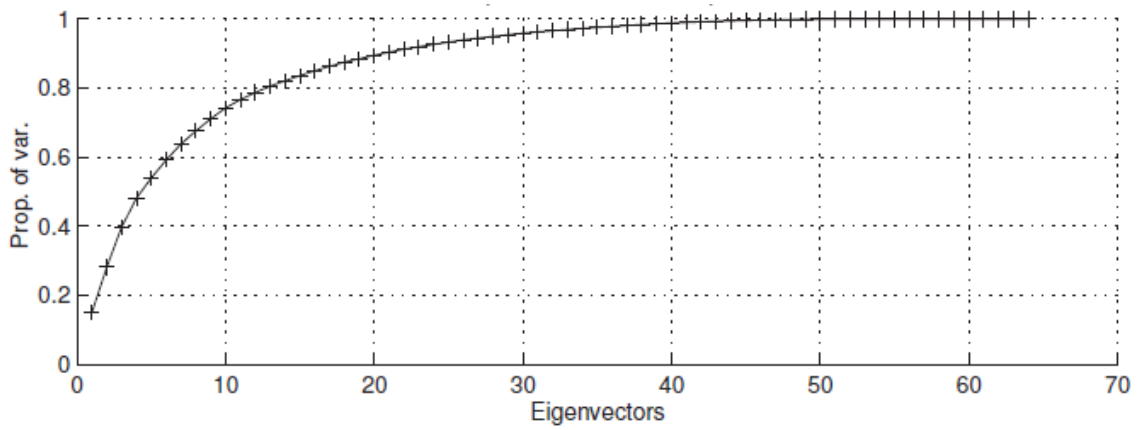

Figure 4. Graph of POV – p relationship, Source: Prof. John C.S. Lui, CUHK

2.2.7.2 Outlier Detection

As shown by Shyu et al, PCA can also be used for outlier detection. [16] It involves the reconstruction of the original higher-dimensional dataset from its lower-dimensional projection. The following is the implementation of PCA Outlier detection in Veeramachaneni et al which is based on Shyu's model. [9] As shown in figure 5, PCA is performed first using j eigenvectors corresponding to largest eigenvalues of the covariance matrix in order to reduce the k-dimensional input X to j-dimensional Y_j where

$$Y_j = XW_j.$$

Then, the reverse of the projection is applied to Y_j to create an estimation of the original matrix X, R_j which is given by

$$R_j = (W_j Y_j^T)^T$$

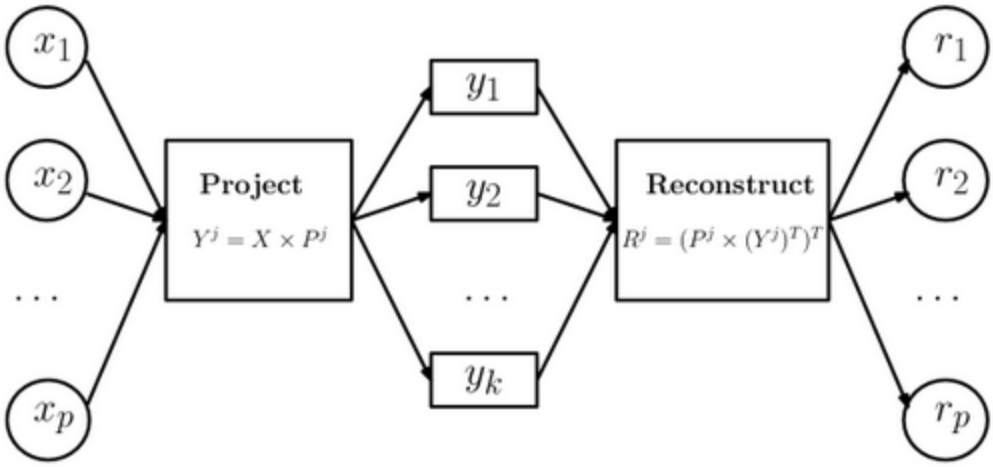

Figure 5. Graphical representation of PCA-based outlier detector [9]

The outlier score for a single instance of X is then defined as

$$score\ (X_i) = \sum_{j=1}^{p} \left|X_i - R_i^j\right| \times POV_p(j)$$

where

$$POV_p(j) = \frac{\lambda_1 + \lambda_2 + \cdots + \lambda_j}{\lambda_1 + \lambda_2 + \cdots + \lambda_j + \cdots + \lambda_p}.$$

The POV_p defined here differs from the previously introduced POV in that it only includes the largest p eigenvalues in the denomination instead of all eigenvalues of Σ.

The score is weighted by POV_p because it is a monotonically increasing function over j and hence, as we use more eigenvectors of the covariance matrix and account for more of the variance, the weight of errors increases, and significant outliers are penalized harshly. [9]

Furthermore, this method allows for the detection of outliers at low cost because of the added benefit of dimensionality reduction.

3. Related Research

3.1 AI^2

AI^2 is a network intrusion detection system developed by Veeramachaneni et al. [9] This is an analyst-in-the-loop security system that “puts together analyst intuition with state-of-the-art machine learning”. [9] AI^2 very neatly combines analyst insight, outlier detection, and supervised learning to produce an effective intrusion detection that has a demonstrably high detection accuracy and very low false positive alarms. [9]

The model is built to detect new and evolving attacks with supervised learning based on analyst insight. The algorithm has three main phases: training, deployment and feedback collection/updating. [9] The model cycles through these stages every day. The system trains supervised and unsupervised models to detect a certain number of extreme events or attacks, the analyst then picks out events that could actually be attacks and labels the data. This labelled data is used as the basis for supervised learning for the next training cycle. [9] This learning cycle has three main advantages as highlighted by the paper:

- **Overcomes limited Analyst bandwidth**

  There is a very limited number of network logs that a human analyst can successfully analyze and label. AI^2 use outlier detection to sharply limit the volume that the analyst has to label manually. [9]

- **Overcomes Weakness of unsupervised learning**

  Unsupervised learning has demonstrably lower accuracy than supervised learning when it comes to intrusion detection. According to the paper, unsupervised learning achieved an accuracy of 7.9% compared to the 86.8% accuracy of AI^2 based on supervised learning, even when the amount of daily training data was increased, unsupervised machine learning was only able to achieve an accuracy of 73.7% compared to > 85% accuracy of AI^2. The labeling of data by the analyst ensures supervised learning and in turn leads to higher detection rates than unsupervised learning. [9]

- **Actively adapts and synthesizes new models**

  Analyst feedback provides labelled data to the model on a daily basis and therefore, the total amount of training data available to AI^2 increases and the model becomes more and more accurate with time. [9]

This continuous training cycle employed by AI^2 is not only sustainable due to selective supervised learning but is also highly effective. This training cycle is a major inspiration behind the design of our system.

3.2 Datasets

The quality and quantity of data used to train a model is pivotal in determining the accuracy of a model in real life environment. But unfortunately, many enterprises lack the necessary training data required for supervised learning. [9] Researchers, over the time, have developed several comprehensive datasets to simulate actual network environment for AI training. We analyze some publicly available datasets that have been used in the training of intelligent intrusion detection systems.

3.2.1 KDD Cup 1999

The KDD Cup 1999 dataset has been a benchmark for intrusion detection research for decades. The dataset is a collection of simulated raw TCP data over a period of nine weeks on a LAN. The training data contains 22 different attack types while the test data contains 39 attack types in order to test intrusion detection for novel attacks. [21][25] The attack types belong to four categories:

- Denial of Service, DOS e.g. SYN flood
- Probing e.g. port scan
- Unauthorized access to local root privileges, U2R e.g. buffer overflow
- Unauthorized access from remote machine, R2L e.g. password guess

The dataset contains 494,021 training records, of which 19.69% are normal, 79.24% are DOS, 0.83% are probes, 0.23 are R2L while the remaining 0.01% are U2R. Each record contains 41 features and a classification label. [21][25]

3.2.2 CICIDS2017

CICIDS2017 dataset was developed by a group of researchers from Canadian Institute of Cybersecurity at University of New Brunswick. [13] This is a very comprehensive dataset that successfully integrates the background traffic characteristics of a modern network with a large number of attack instances. [13] The dataset made use of complete network topologies including modem, firewalls, routers and switches along with a collection of hosts and servers running multiple operating systems. [13] The data set includes six common attack profiles.

A. Brute force attack

The most common types of attack, brute force is simply a trial and error approach to extract information e.g. repeatedly guessing a password. Moreover, brute force can also be used to determine the existence of hidden web pages. [13]

B. Heartbleed attack

Heartbleed attack exploits security vulnerability in OpenSSL cryptography library, widely used in Transport Layer Security protocol. This vulnerability results from improper input validation that allows for buffer overread in TLS heartbeat extension. [13]

C. Botnet

Botnet attack refers to the class of attack conducted by multiple IoT devices whose security has been compromised and they are controlled by malicious agents. [13]

D. DoS

Denial of Service attacks are aimed at limiting or even disabling the resources and services of the target system. These attacks are conducted by overwhelming the target system by excessive payload, excessive number of packets, or excessive usage of ports, to the extent that legitimate users can't avail the intended services of the system or the network. [13]

E. DDoS

DDoS or Distributed Denial of Service attack is a combination of Botnets and DoS where multiple compromised devices are used to launch a DoS attack on the victim.

F. Web attack

There are several types of Web attacks. SQL injection is one of the most common attack where the attacker tries to bypass the authorization of a Web server and tries to get access to sensitive data in the target's SQL database. Another type of web attack called Cross Site Scripting (XSS) attack tries to exploit code vulnerabilities of legitimate website to modify its code and exploit the visitors of the site. [13]

G. Infiltration attack

These attacks aim to bypass a network's defensive mechanisms by creating a backdoor in one of the internal machines. [13]

In addition to the large number of attacks, CICIDS2017 also stands out because of the large number of features extracted from network logs. These features (more than 80) are generated using CICFlowMeter [14] and were shown to be very effective when used to predict attacks. [13]

We intend on using CICIDS2017 to evaluate our system. There are several reasons for this selection. Firstly, the dataset has a substantial amount of background traffic which provides a more realistic network environment. [13] Secondly, the dataset, released in 2017, is very recent compared to datasets like KDD ‘99, which many criticize for being too outdated. [25] Thirdly, the seven attack classes used in the dataset are very prevalent in today’s cyber security landscape and thus there is a high probability that our system will have to frequently detect such attacks after deployment.

4. Design

We implement a framework for a hybrid network-based intrusion detection system. The system contains three major components: feature extraction, outlier detection and attack classification. It utilizes a two-phase approach to detect network attacks. In the first phase, an outlier detection algorithm, acting as an anomaly-detection IDS, detects Y_t outliers from the input X_t. The outliers, Y_t, are then vetted by a signature-based IDS which decides the type of attack.

Anomaly-based IDS are better at detecting novel attacks, but they suffer from a high false-positive rate. [9] Signature-based IDS, on the other hand, are highly accurate at detecting traditional attacks but require extensive and incessant labelling of data to maintain their accuracy against new attacks. [13][28] This method combines the best of both types of intrusion detection systems. The attack classification algorithm offsets the high false positive rates of outlier detection while the latter reduces the workload of labelling data since only outliers need to be labelled.

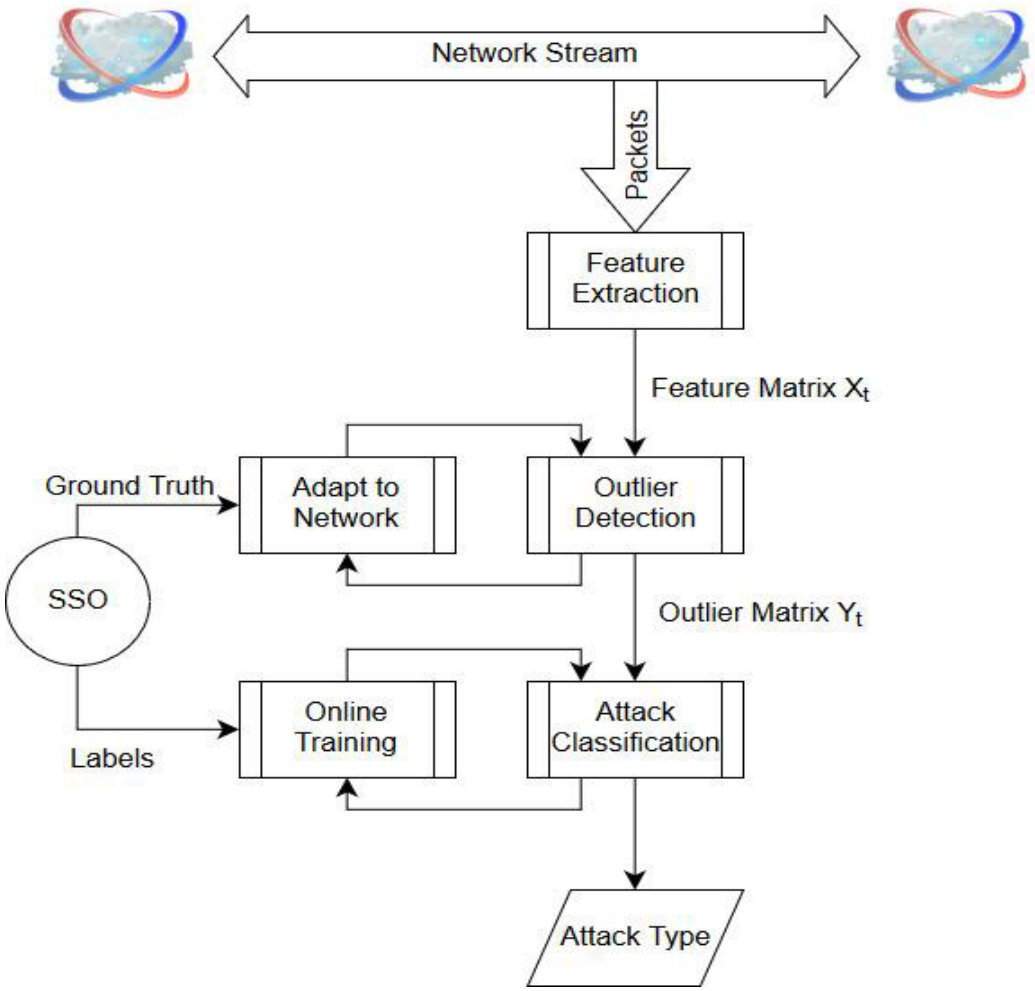

Figure 6. System Design Overview: A Hybrid Intrusion Detection System

Figure 6 contains an overview of the entire system. The major components of the system are the following.

4.1 Audit Data Collection

4.1.1 Raw Data

The most basic unit of data in the system is a raw packet. We denote a single packet as i_p.

4.1.2 Audit data

A single instance of audit data, x, is a vector consisting of features F extracted from the headers of all packets in a single session s.

4.1.2.1 Session

A session is defined as a communication between two IP addresses, srcIP and dstIP, over a time interval t.

4.1.2.2 Features

Appendix I lists the set of all features, F.

4.2 Feature Extraction

4.2.1 Preprocessing

The following is a pseudo-code definition of the pre-processing algorithm that converts a stream of packets into audit data matrix X_t.

```
1  PROCEDURE Extract Features
2      INPUT:
3          stream of packets, I
4          time interval, t
5      OUTPUT:
6          Audit data matrix, Xt
7      PROCESS:
8          REPEAT Until no more packets in I
9              WHILE time elapsed <= t
10                 Read packet i from I
11                 Extract features F, srcIP, dstIP from headers
12                 IF session S with (srcIP, dstIP) does not exist
13                     Create session S
14                     Create x[S]
15                 Update x[S] using values from F
16             END WHILE
17             FOR all sessions S:
18                 Append x[S] to Xt
19             ENDFOR
20             Output Xt
21             Reset time elapsed
22         END REPEAT
23 END PROCEDURE
```

Figure 7. Pseudo-code for Feature Extractor

4.2.2 Feature Extractor

Feature Extractor is our implementation of CICFlowMeter by researchers from University of New Brunswick . The original implementation of CICFlowMeter was developed using Java, whereas we have developed in Python to allow for greater reusability and maintainability for future researchers who could use easily use or modify it to extract the desired flow features from PCAP files or live network interfaces.

The original implementation extracts packet information from IPv4, IPv6, and L2TP packets, whereas our implementation only supports IPv4 and IPv6 packets at the Network layer. At Transport layer both implementations have been limited to TCP and UDP protocols only. Our implementation allows for both online and offline flow extractions. The online flow extraction feature was added to make it easier to bootstrap Feature Extractor with our NIDS framework. Bootstrapping the two modules together can have significant impact in reducing the anomaly

detection and classification delay.

We extract at total of 83 features from every network flow. Further details about these features can be found in the appendix at the end of this document. Before we discuss the implementation of Feature Extractor in any more detail, let's first try to understand what constitutes a flow in our program.

4.2.2.1 Flow

A flow is classified as a 4-tuple of source and destination IP addresses and port numbers. A flow ID is the unique identifier allocated to each flow and it follows the syntax:

{ IP Address 1}-{ Port Number 1}-{ IP Address 2}-{ Port Number 2}-{Protocol Number}

The suffix '1' and '2' are allocated by byte wise comparison of the two source and destination IP addresses of a packet. This comparison is performed by the generateFlowId function of our BasicPacketInfo class. The function is defined in figure 8 below. Once the flow ID has been allocated, the source and destination IP addresses of the first packet determine the source and destination of the flow. A flow is considered terminated and removed from the list of active flows if:

1. Flow duration exceeds flow timeout value provided by the user.
2. A flow packet carries a TCP FIN flag.

Furthermore, the user also has the flexibility of determining whether the flows should be treated as bidirectional flows or unidirectional flows. As an example, if user opts for bidirectional flow then the packets from {Host A:Port A} to {Host B:Port B} and from {Host B:Port B} to {Host A:Port A} will be considered as part of the same flow, if packets from both hosts belong to the same transport layer protocol. On the other hand, if the user opts for unidirectional flows, then packets from {Host A:Port A} and packets from {Host B:Port B} will be treated as independent flows.

```
def generateFlowId(self):
    forward = True
    for i in range(0,len(self.__src)):
        print('')
        if self.__src[i] != self.__dst[i]:
            if self.__src[i] > self.__dst[i]:
                forward = False
            break
    if forward:
        self.__flowId = self.getSourceIp() + "-" + self.getDestinationIp() + "-" + str
        (self.__srcPort) + "-" + str(self.__dstPort) + "-" + str(self.__protocol)
    else:
        self.__flowId = self.getDestinationIp() + "-" + self.getSourceIp() + "-" + str
        (self.__dstPort) + "-" + str(self.__srcPort) + "-" + str(self.__protocol)
    return self.__flowId
```

Figure 8. Function generateFlowId( )

4.2.2.2 Design

The Feature Extractor module has 7 main components:

1. Module Extractor
2. Flow Meter
3. Flow Processor
4. Flow Generator
5. Basic Flow.
6. Basic Packet Info
7. Summary Statistics

We will now present the more intrinsic details of the Feature Extractor program. We will start off with a high level description of the program using a UML class diagram and then move towards a more detailed textual description of these classes and some of the more important functions implemented within them. Diligent code refactoring has ensured that all of these classes are easily modifiable. In fact, by simply modifying the Basic Flow class, one can easily modify the number, type and ordering of the features extracted.

4.2.2.3 UML Class Diagram

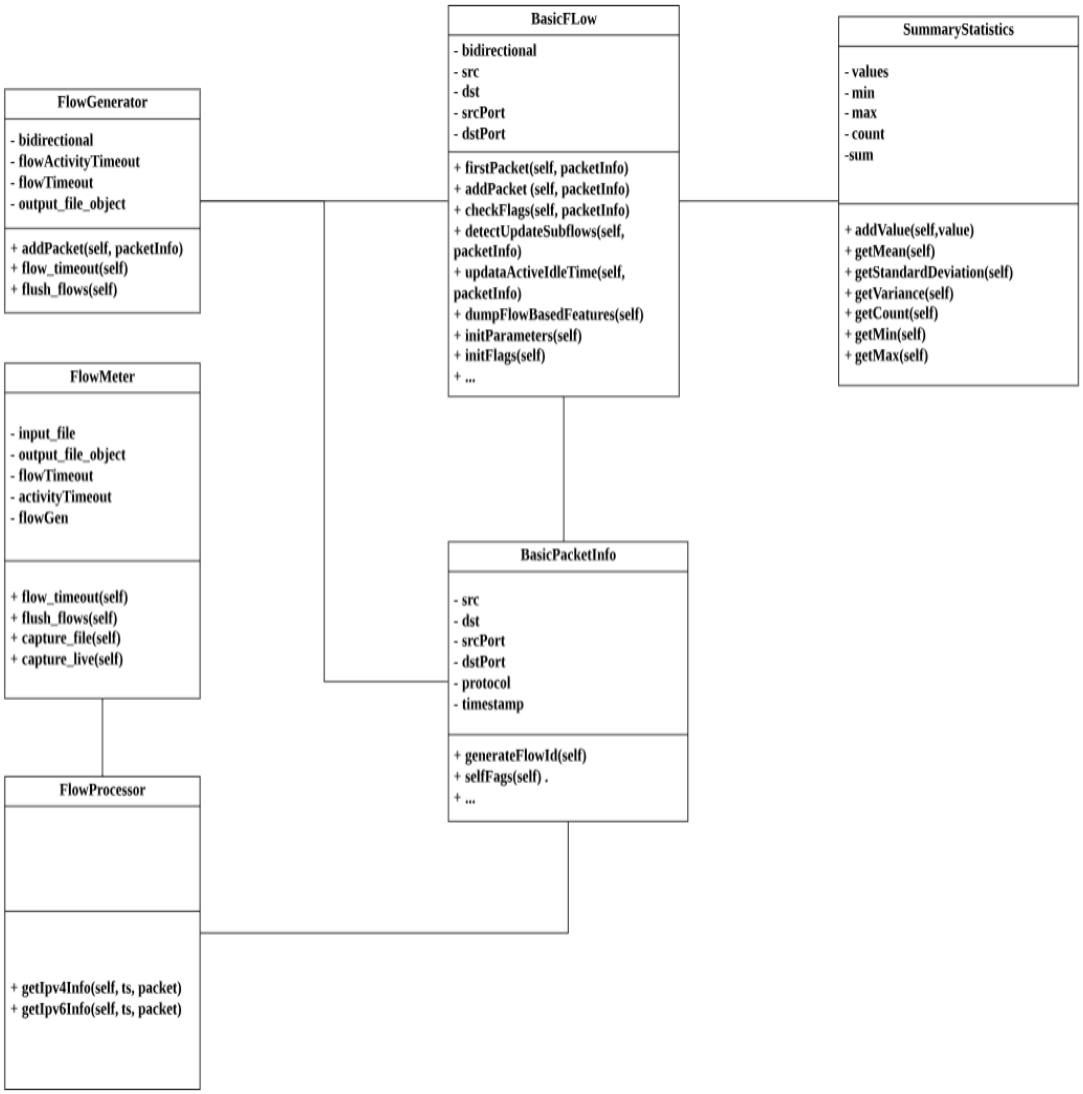

Figure 9. Feature Extraction: UML Class Diagram

4.2.2.4 Feature Extractor

Feature Extractor is the main module of our feature extraction program. This module is responsible for processing command line arguments and for verifying their correctness. This module creates the output directory for storing the csv files produced after the features have been extracted. It also creates the appropriate instances of Flow Meter module. This module also handles operating system exceptions and registers signals for live detection functionality of Flow Meter.

1. **set_signal( ):** Registers the appropriate signal handler for the main program
2. **main( ):** Main function that is called when the Feature Extractor script is executed it processes and verifies command line arguments and sets exception handlers

4.2.2.5 Flow Meter

This class is called directly by the main( ) module of the Feature Extractor class. It initializes the Flow Generator class instance according to the command line arguments provided by the user. This class contains the file reader for the input PCAP files and opens a promiscuous network interface sniffer if the user wishes to perform live intrusion detection. This class makes use of our Flow Processor class instance to extract the desired packet information from raw network layer packets.

1. **flow_timout( ) :**

   A handler to call the flow_timout function of Flow Generator object. This function is called by the signal handlers of the main( ) module when the flow activity timeout is triggered.

2. **flush_flows( ):**

   A handler to call the flush_flows function of the Flow Generator object.

3. **capture_file( ):**

   This function is called by the main( ) module of the Feature Extractor object if it receives the command line argument to process a PCAP file instead of a live network interface. It creates an instance of pcap object from Python's pcap library. It reads the ethernet frame to determine the Network layer protocol type (IPv4 or IPv6) and calls the appropriate function to extract packet information. Once packet information has been extracted, it is used to create an instance of the Basic Packet Info class which is then passed to the Flow Generator object.

4. **capture_live( ):**

   This function is called by the main( ) module of the Feature Extractor object if it receives the command line argument to process a live network interface. It creates an instance of a pcapy object from Python's pcapy library. Then pcapy's open_live ( ) function is called to sniff network traffic from the specified network interface. It reads the ethernet frame sniffed by pcapy to determine the network layer protocol type (IPv4 or IPv6) and calls the appropriate function to extract packet information. Once packet information has been extracted, it is used to create an instance of the Basic Packet Info class which is then passed to the Flow Generator instance.

4.2.2.6 Flow Processor

A simple helper class used by Flow Meter to extract desired information from IP packets. This class is composed of two functions: getIpv4Info( ) and getIpv6Info( ). Both of these functions

take timestamp and ethernet payloads as arguments to extract packet information. The extracted information is returned as an instance of Basic Packet Info. The extracted information is composed of:

1. source IP address
2. destination IP address
3. source port number
4. destination port number
5. transport layer protocol number
6. transport layer header size
7. transport layer payload size

If the transport layer protocol is TCP, then the functions also extract the TCP window size and flag bits.

4.2.2.7 Flow Generator

An important class that serves as the coordinator of all active flows in memory. It maintains a list of active flows and creates, modifies and deletes them accordingly.

1. **flow_timout( ):**

   This function is usually called by the flow timeout signal. It iterates through the list of active flows and checks for flows that have lasted longer than the specified flow activity timeout. If it finds such flows, the flow features are flushed to output file and the flow is removed from the list.

2. **flush_flows( ):**

   This function iterates through the list of active flows and calls the dumpFlowBasedFeatures( ) function for all them. The dumpFlowBasedFeatures( ) function simply write all the extracted flow features to the output file, divided by the separators provided.

3. **addPacket( ):**

   This is one of the most important functions of our module. It takes a Basic Packet Info instance as argument and inserts it into its list of flows accordingly. If a flow with the same flow ID already exists, that flow is updated according to the information encapsulated by the Basic Packet Info instance. If the instance contains a FIN flag, the flow is updated and flushed before being removed from the list. If the packet is received after the flow timeout, the existing flow is flushed, and a new Basic Flow instance is

created using the Basic Packet Info instance. This Flow instance is then added to the list of active flows. For existing flows, this function also updates the flow activity time by calling the updateActiveIdleTime( ) function of the target flow.

4.2.2.8 Basic Flow

An important class that defines every flow in our Feature Extractor. It contains important functions to update every flow based on the processed network packet.

1. **firstPacket( ):**
   This function is called by the class constructor to initialize fundamental flow identifiers. It sets the source and destination IP addresses and port numbers for the flow instance. It is also responsible for setting the protocol and the flow ID.
2. **addPacket( ):**
   This function is called by the flow generator as soon as it receives a new packet and decides to update a flow. If the flow is bidirectional this function determines the direction of the flow and updates both common and direction specific feature parameters of the flow instance. If the flow is not bidirectional it simply updates the common and forward direction flow features of the flow instance.
3. **updateActiveIdleTime( ):**
   This function is called every time an active flow is updated. This function updates the flow activity parameters according to flow activity timeout argument passed to the main program by the user.
4. **detectUpdateSubflows( ):**
   This function is called every time an active flow is updated. This function updates the sub flow parameters in the flow instance. A sub flow is essentially a classification of sudden bursts of packets. It is important to track these short, heavy bursts to detect a variety of Denial of Service attacks.

4.2.2.9 Basic Packet Info

This is the most primitive class used in our program. Each instance of this class corresponds to the information extracted from an IP packet. This class has several setter and getter functions that have been used sparingly by higher level classes to extract flow related information. An important function implemented in this class is generateFlowId( ). As the name suggest it is used to generate flow IDs based on the source and destination IP addresses and port numbers of the packet. The correct implementation of this function is critical to ensure that a unique and

useful flow ID is allocated to each flow deterministically.

4.2.2.10 Summary Statistic

A helper class, whose instances are used sparingly to calculate the minimum, maximum and mean values of certain parameters. Each class instance maintains a list of all the values passed to it, so that it can also calculate the standard deviations and variances when required.

4.2.3 Audit Storage

The system extracts all useful information from packets and then deletes the raw packets to ensure that the space required to store data is small. Audit data is stored as csv files which are then passed to the processing algorithms that delete most files after training. Therefore, the storage cost for the system is minimal.

4.3 Detection

4.3.1 Outlier Detection

The outlier detection system has been designed specifically for online learning. Figure 10 is a graphical representation of the system.

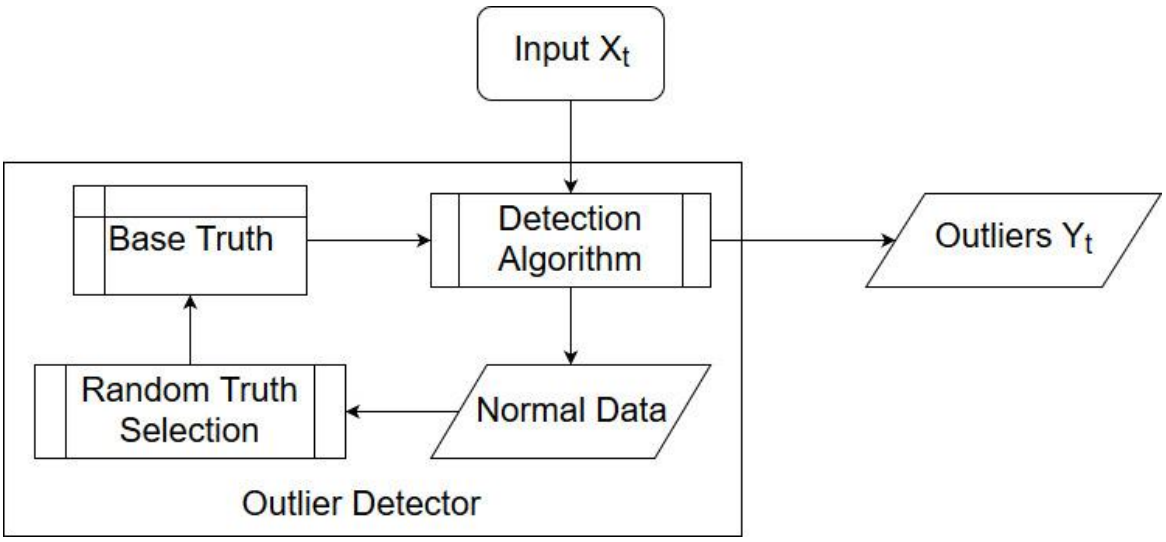

Figure 10. Online Outlier Detection

Even though the number of normal instances is generally greater than the number of anomalous instances, there is still a possibility that the input X_t contains a majority of malicious communications. Therefore, in order to ensure that the algorithm does not train on abnormal data, it requires a set of verified normal instances known as the base truth. The size of this set is m whereas the size of input X_t is n. By setting $m > n$, we ensure that abnormal data is not

learned by the algorithm.

The base truth is only instantiated once on the first iteration of the algorithm. During further iterations, the system randomly selects j instances from X_t which have been determined as normal by the algorithm and replaces them with j random instances in the base truth. This allows the algorithm to adapt to changes in network traffic. The ratio $\frac{j}{m}$ determines the speed at which the algorithm evolves alongside the network.

As shown in the figure below, the system also supports the use of an ensemble instead of a single algorithm as the outlier detector.

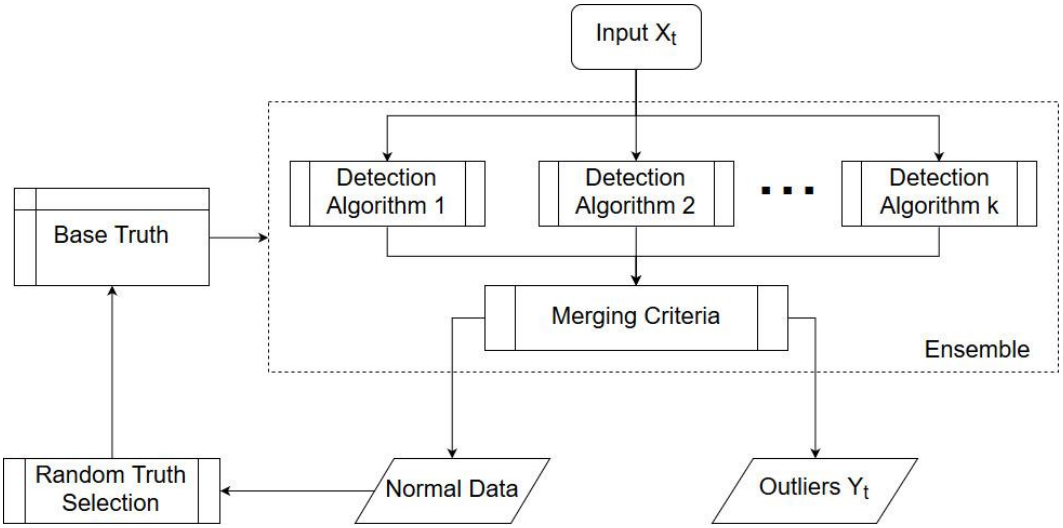

Figure 11. Outlier Detection using ensemble methods

4.3.2 Attack Classification

The attack classification algorithm has also been designed with online learning as the primary motivation. Figure 10 is a flow chart of this algorithm.

First, a classification model is trained using signatures of known attacks. It is then used to classify all outliers according to the type of attack. If the classifier fails to recognize any attack signature, the site security officer (SSO) is required to label the instance and the classifier is re-trained on the labelled instance.

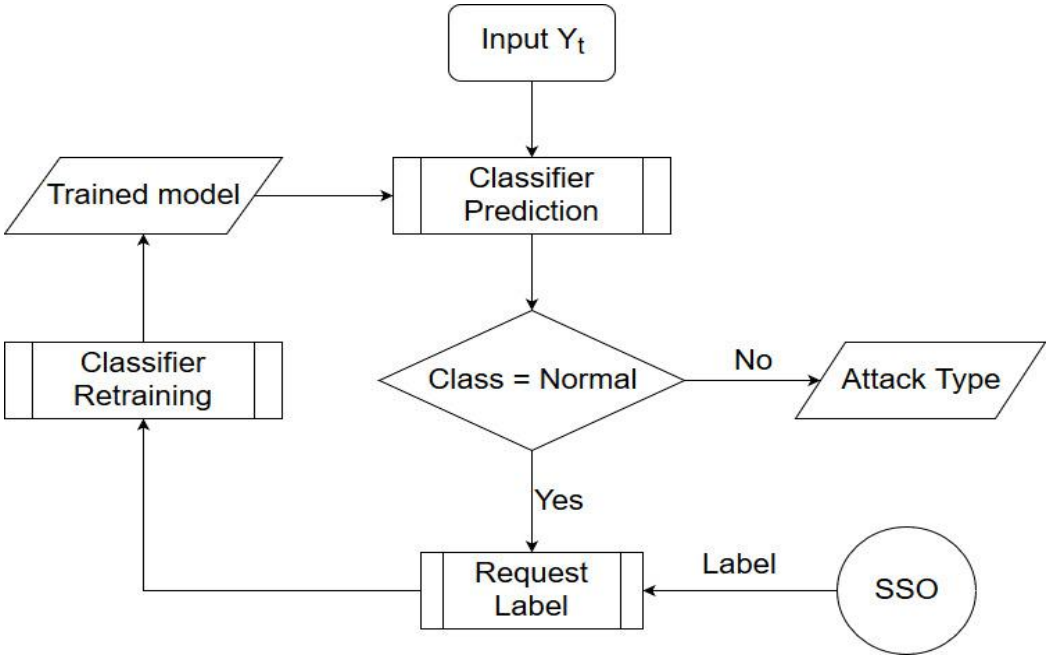

Figure 12. Online Attack Type Classification

The classification system also supports the use of an ensemble as an attack classifier. The results of individual classifiers in a classification ensemble are considered as votes for each class and the class with more votes than a threshold is the final prediction of the ensemble. If no class crosses the threshold, a label is randomly assigned.

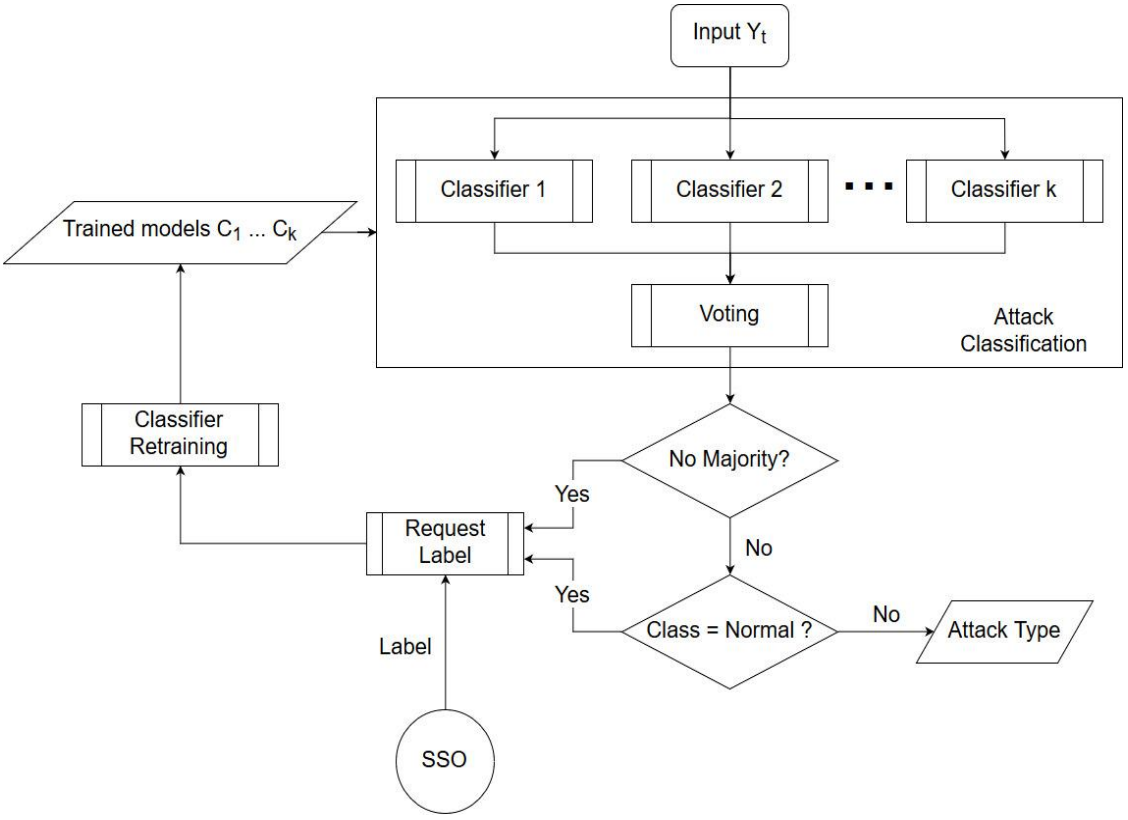

Figure 13. Attack Type Classification with ensemble methods

4.4 Deployment

The framework is capable of deployment at any router inside the network. Figure 14 shows two deployment scenarios which are of interest.

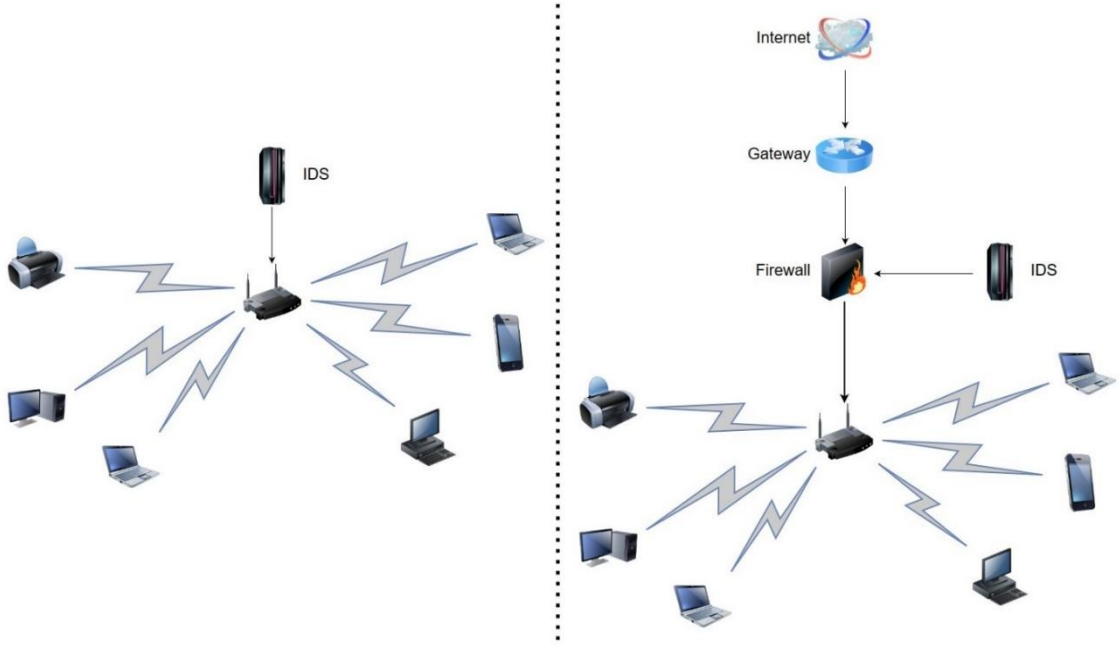

Figure 14. System deployment scenarios: Internal Router (left) and Gateway router (right)

The first scenario shows a network where a single router manages all communications. This allows the IDS to monitor all the traffic in the network to detect malicious use.

The second scenario depicts a network where the IDS is deployed at the gateway firewall. This configuration enables the detection of attacks originating outside the network. However, this configuration may not be able to detect attacks launched from within the network as inter-network communication does not pass the gateway.

However, if inter-network security is a significant concern, the following deployment scenario may be considered. It contains a central router that manages all inter-network and intra-network communication. This configuration, however, is not without drawbacks as it introduces a single point of failure in the network. This drawback can be rectified by multiple deployments at the cost of increased resource requirement.

Thus, deployment location is an important decision which must be made according to the network topology and security requirements.

5. Experiments and Results

5.1 System

The experimental model has the following configuration:

5.1.1 Outlier detection algorithm

The primary outlier detector is the PCA-based model described in section 2. This model has previously been used by Veeramachaneni et al. and Shyu et al. and has been very successful in anomaly-based intrusion detection. [9][16]

5.1.2 Attack Detection Algorithm

Naive Bayes is used as the classification algorithm because of its simplicity and support for online learning.

5.2 Datasets

The system is tested on two different datasets: KDD'99 and CICIDS'17. KDD'99 continues to be a benchmarking favorite in intrusion detection research despite its many inadequacies while CICIDS'17 is noteworthy for its novelty and large number of features. Moreover, these two datasets do not share any significant number of features and a rough comparison of performance may demonstrate the independence of the feature extraction and detection components of the system.

5.3 Overall Performance

5.3.1 Experiment

We run the system on the dataset with three different configurations:

1. Batched
    - This configuration simulates the actual execution of the system.
    - All data is fed to the system in batches of random sizes (between l=50 and h=2000) to simulate the arrival of a batch of flows after time interval *t*.
    - Batches of training data are used to train the system.
    - When a batch of testing data arrives at the system, it is first filtered for outliers and the outliers are classified into attack types. Normal data (non-outlier) is then used to update the base truth for outlier detection.
    - The system is provided the correct labels for outliers, simulating the actions of an

SSO, which are used to retrain the classifiers.

2. Complete Unseen
    - This configuration simulates the standard offline detection algorithm whereby the system is first trained using the entirety of the training data and then tested on unseen data.
3. Complete Pre-trained
    - This configuration serves to provide an upper-bound on the performance of the system.
    - The system is trained on both training and testing data so that its testing performance can be optimized.

5.3.2 Results

5.3.2.1 KDD'99

The following shows the performance of the outlier detection system of each configuration over all testing data.

	Outlier Precision	**Outlier Recall**	**Outlier F1 Score**
Batched	71.1	99.9	83.1
Complete Unseen	74.0	99.9	85.0
Complete Pre-trained	74.5	99.9	85.5

The following shows the performance of the classification system of each configuration over all outliers.

	Weighted Precision	**Weighted Recall**	**Weighted F1 Score**
Batched	75.0	52.0	49.0
Complete Unseen	48.0	40.0	38.0
Complete Pre-trained	76.0	54.0	52.0

The following shows the overall performance of the configurations over all testing data.

	Weighted Precision	**Weighted Recall**	**Weighted F1 Score**
Batched	82.0	62.0	66.0
Complete Unseen	67.0	54.0	58.0
Complete Pre-trained	82.0	65.0	70.0

It is clear from the results above that the batched configuration performs better than the complete unseen system. This is the expected result since the former involves more training than the latter. Indeed, the batched configuration re-trains after every prediction while the complete unseen configuration only involves training at the start.

Moreover, the performance of the batched configuration is very close to the rough upper limit defined by the complete pre-trained system even though the batch configuration always performs predictions on unseen data while the pre-trained configuration that has access to the entire dataset.

5.3.2.2 CICIDS'17

The following shows the performance of the outlier detection system of each configuration over all testing data.

	Outlier Precision	**Outlier Recall**	**Outlier F1 Score**
Batched	24.7	100	34.6
Complete Unseen	31.2	97.7	40.0
Complete Pre-trained	31.7	98.4	40.5

The following shows the performance of the classification system of each configuration over all outliers.

	Weighted Precision	Weighted Recall	Weighted F1 Score
Batched	98.3	66.4	75.7
Complete Unseen	60.6	56.8	56.7
Complete Pre-trained	97.9	76.8	84.2

The following shows the overall performance of the configurations over all testing data.

	Weighted Precision	Weighted Recall	Weighted F1 Score
Batched	98.6	75.4	77.0
Complete Unseen	68.7	67.6	67.2
Complete Pre-trained	97.8	84.0	89.8

The results seen above are similar to those attained in the KDD'99 experiment and confirm the performance enhancement attained by the batched configuration.

5.4 Classifier Re-training

5.4.1 Experiment

We test the following three mechanisms of re-training classifiers in the batched configuration:

1. Outliers
    - The classifiers are re-trained on all outliers detected by the system including misclassified and correctly classified outliers.
2. Mistakes
    - The classifiers are re-trained on misclassified outliers only.
3. Active
    - The classifiers are re-trained on all outliers alongside duplicated misclassified outliers. This amounts to doubling of the weights of mistakes.

5.4.2 Results

5.4.2.1 KDD'99

The following shows the performance of the classifier using the three different update mechanisms.

	Weighted Precision	Weighted Recall	Weighted F1 Score
Outliers	75.0	52.0	49.0
Mistakes	73.0	51.0	48.0
Active	75.0	51.0	49.0

It can be seen that re-training on all outliers is better than re-training on misclassified outliers only. Although active re-training has a similar performance, it also increases the training time due to the increase in the number of data points. Therefore, training on all outliers seems to be the most suitable mechanism for the system. The experiments defined in section 5.3 also use this mechanism.

5.4.2.2 CICIDS'17

The following shows the performance of the classifier using the three different update mechanisms.

	Weighted Precision	Weighted Recall	Weighted F1 Score
Outliers	98.3	66.4	75.7
Mistakes	98.2	62.8	73.4
Active	98.6	75.4	77.0

Unlike the previous results, active learning outperforms outlier-only training in the CIC dataset. Therefore, active learning can be fairly useful if a short training time is less critical.

5.5 Ensemble

5.5.1 Experiment

We compare the performance of an ensemble of machine learning algorithms to the performance of the standard detection set-up.

1. Standard
    - Outlier Detection: PCA
    - Attack Classification: Naïve Bayes
2. Ensemble
    - Outlier Detection: PCA and KMeans
    - Attack Classification: Naïve Bayes, Linear SVM and Passive Aggressive Classifier

5.5.2 Results

5.4.2.1 KDD'99

The following shows the overall performance of each set-up using the KDD dataset.

	Weighted Precision	**Weighted Recall**	**Weighted F1 Score**
Standard	82.0	62.0	66.0
Ensemble	79.0	64.0	68.0

Interestingly, the ensemble only shows a slight improvement in performance over the standard set-up.

5.4.2.2 CICIDS'17

The following shows the overall performance of each set-up using the CIC dataset.

	Weighted Precision	**Weighted Recall**	**Weighted F1 Score**
Standard	98.6	75.4	77.0
Ensemble	94.9	94.9	94.9

Unlike the former, the ensemble completely outperforms the standard set-up in this experiment. This is because most machine learning algorithms are suited to only certain datasets. Therefore, the weight of each model in the ensemble must be determined carefully so that the outputs of algorithms suited to the dataset are given more importance than those of unsuitable algorithms.

This further enhances the significance of an ensemble since a single algorithm will always fail in the face of an unsuitable dataset while a group of algorithms are less likely to experience a complete failure.

5.6 Evaluation

The aforementioned experiments and their results show that online training considerably improves the performance of the detection system. Although the performance of the outlier detection system leaves more to be desired, the low precision is a tradeoff meant to increase the recall of the system. Recall is more important in the outlier detection component because we do not want attacks to slip through the filter. However, the outlier threshold can be loosened based on the ground realities of the network in order to reduce the number of false positives.

These experiments also show that the detection component is independent of the features since the results of experiments involving different datasets are not very dissimilar. Moreover, it is also independent of the machine learning algorithms since the system shows a substantial performance improvement even though all of the above experiments involve the same algorithms. This allows the detection engine to support any number of feature extraction components and machine learning algorithms and to act as a plug-and-play framework.

6. Conclusion

In this project, we have implemented a framework to enable the use of machine learning in a network intrusion detection system capable of accurate, real-time detection. In the first term, we conducted extensive research of the broad topics of intrusion detection and machine learning. We employed this research to narrow down the scope of our project in terms of the form of intrusion detection system, types of intrusions and machine learning algorithms. In this term, our efforts were focused on the design, implementation and evaluation of the system.

7. Appendix

7.1 Table of Features

Feature Name	Description
Feduration	Duration of the flow in Microsecond
total_fpackets	Total packets in the forward direction
total_bpackets	Total packets in the backward direction
total_fpktl	Total size of packet in forward direction
total_bpktl	Total size of packet in backward direction
Mmin_fpktl,max_fpktl, mean_fpktl,std_fpktll	Minimum, maximum, mean and standard deviation of packet size of packet in forward direction
min_bpktl,max_bpktl,mean_bpktl ,std_bpkt	Minimum, maximum, mean and standard deviation of packet size of packet in backward direction
total_fiat	Total time between two packets sent in the forward direction
total_biat	Total time between two packets sent in the backward direction
min_fiat,max_fiat,mean_fiat,std_ fiat	Minimum, maximum, mean and standard deviation of time between two packets sent in the forward direction
min_biat,max_biat,mean_biat,std _biat	Minimum, maximum, mean and standard deviation of time between two packets sent in the backward direction
fpsh_cnt	Number of times the PSH flag was set in packets travelling in the forward direction (0 for UDP)
bpsh_cnt	Number of times the PSH flag was set in packets travelling in the backward direction (0 for UDP)
furg_cnt	Number of times the URG flag was set in packets travelling in the forward direction (0 for UDP)
burg_cnt	Number of times the URG flag was set in packets travelling in the backward direction (0 for UDP)
total_fhlen	Total bytes used for headers in the forward direction
total_bhlen	Total bytes used for headers in the forward direction
fPktsPerSecond	Number of forward packets per second

bPktsPerSecond	Number of backward packets per second
flowPktsPerSecond	Number of flow packets per second
flowBytesPerSecond	Number of flow bytes per second
Min_flowpktl, max_flowpktl, mean_flowpktl, std_flowpktl	Minimum, maximum, mean and standard deviation of length of a flow
Min_flowiat,max_flowiat, mean_flowiat, std_flowiat	Minimum, maximum, mean and standard deviation of inter-arrival time of packet
flow_fin	Number of packets with FIN
flow_syn	Number of packets with SYN
flow_rst	Number of packets with RST
flow_psh	Number of packets with PUSH
flow_ack	Number of packets with ACK
flow_urg	Number of packets with URG
flow_cwr	Number of packets with CWE
flow_ece	Number of packets with ECE
downUpRatio	Download and upload ratio
avgPacketSize	Average size of packet
fAvgSegmentSize	Average size observed in the forward direction
fAvgBytesPerBulk	Average number of bytes bulk rate in the forward direction
fAvgPacketsPerBulk	Average number of packets bulk rate in the forward direction
fAvgBulkRate	Average number of bulk rate in the forward direction
bAvgSegmentSize	Average size observed in the backward direction
bAvgBytesPerBulk	Average number of bytes bulk rate in the backward direction
bAvgPacketsPerBulk	Average number of packets bulk rate in the backward direction
bAvgBulkRate	Average number of bulk rate in the backward direction
sflow_fpacket	The average number of packets in a sub flow in the forward direction
sflow_fbytes	The average number of bytes in a sub flow in the forward direction

sflow_bpacket	The average number of packets in a sub flow in the backward direction
sflow_bbytes	The average number of bytes in a sub flow in the backward direction
Min_active, max_active, mean_active, std_active	Minimum, maximum, mean and standard deviation of time a flow was active before becoming idle
Min_idle, max_idle, mean_idle, std_idle	Minimum, maximum, mean and standard deviation of time a flow was idle before becoming active
Init_Win_bytes_forward	The total number of bytes sent in initial window in the forward direction
Init_Win_bytes_backward	The total number of bytes sent in initial window in the backward direction
Act_data_pkt_forward	Count of packets with at least 1 byte of TCP data payload in the forward direction
min_seg_size_forward	Minimum segment size observed in the forward direction

8. References

1. Mahjabin, T., Xiao, Y., Sun, G., & Jiang, W. (2017). A survey of distributed denial-of-service attack, prevention, and mitigation techniques. International Journal of Distributed Sensor Networks. https://doi.org/10.1177/1550147717741463
2. Chapelle, D. 2011. Rationalization and Defense in Depth - Two Steps Closer to the Clouds. Oracle Technology Network Architect Day. Retrieved on 23 Nov, 2018 from : https://www.slideshare.net/OTNArchbeat/rationalization-and-defense-in-depth-two-steps-closer-to-the-clouds
3. Mukherjee, B. Heberlein, L.T. Levitt, K.N. 1994. Network Intrusion Detection. IEEE Network.
4. Mahoney, M.V. Chan, P. (2002). PHAD: Packet Header Anomaly Detection for Identifying Hostile Network Traffic.
5. Roesch, M. 1999. Snort - Lightweight Intrusion Detection for Networks. In *Proceedings of the 13th USENIX conference on System administration* (LISA '99). USENIX Association, Berkeley, CA, USA, 229-238.
6. Lazarevic, A. Ertöz, L. Kumar, V. Ozgur, A. Srivastava, J. (2003). A Comparative Study of Anomaly Detection Schemes in Network Intrusion Detection. 3. 10.1137/1.9781611972733.3.
7. Miguel A. Calvo Moya, 2008. Analysis and evaluation of the snort and bro network intrusion detection systems. Universidad Pontifica Comillas.
8. Mukkamala, S. Janoski, G. Sung, A. (2002). Intrusion detection using neural networks and support vector machines. Proceedings of the International Joint Conference on Neural Networks. 2. 1702 - 1707. 10.1109/IJCNN.2002.1007774.
9. Veeramachaneni, K. Arnaldo, I. Korrapati, V. Bassias C. and Li K. 2016. AI^2: Training a Big Data Machine to Defend. *IEEE 2nd International Conference on Big Data Security on Cloud (BigDataSecurity), IEEE International Conference on High Performance and Smart Computing (HPSC), and IEEE International Conference on Intelligent Data and Security (IDS)*, New York, NY, 2016, pp. 49-54. 10.1109/BigDataSecurity-HPSC-IDS.2016.79
10. Can O. Turguner, C. Sahingoz, O.K. A Neural Network Based Intrusion Detection System for Wireless Sensor Networks.
11. Moustafa, N. & Slay, J. 2015. UNSW-NB15: a comprehensive data set for network intrusion detection systems (UNSW-NB15 network data set). 10.1109/MilCIS.2015.7348942.

12. Moustafa, N. & Slay, J. 2015. The Significant Features of the UNSW-NB15 and the KDD99 Data Sets for Network Intrusion Detection Systems. 25-31. 10.1109/BADGERS.2015.014.
13. Sharafaldin, I. Habibi Lashkari, A. Ghorbani, A. 2018. Toward Generating a New Intrusion Detection Dataset and Intrusion Traffic Characterization. 108-116. 10.5220/0006639801080116.
14. CICFlowMeter, Retrieved on November 23, 2018 from http://netflowmeter.ca/
15. Axelsson, S. Sands, D. 2005. Understanding Intrusion Detection Through Visualization
16. Shyu, M.L. Chen, S.C. Sarinnapakorn, K. Chang, L. 2003. A Novel Anomaly Detection Scheme Based on Principal Component Classifier. Proceedings of International Conference on Data Mining.
17. Cortes, C. and Vapnik, V. 1995. Support-Vector Networks. Mach. Learn. 20, 3 (September 1995), 273-297. DOI: https://doi.org/10.1023/A:1022627411411
18. Ho, T.K. 1995. Random decision forests. In Proceedings of the Third International Conference on Document Analysis and Recognition (Volume 1) - Volume 1 (ICDAR '95), Vol. 1. IEEE Computer Society, Washington, DC, USA, 278-.
19. Zhang, H. 2004. The Optimality of Naive Bayes. Proceedings of the Seventeenth International Florida Artificial Intelligence Research Society Conference, FLAIRS 2004. 2.
20. Liu, Y. He, Q. Chen, Q. Incremental batch learning with support vector machines, Fifth World Congress on Intelligent Control and Automation (IEEE Cat. No.04EX788), Hangzhou, China, 2004, pp. 1857-1861 Vol.2. doi:10.1109/WCICA.2004.1340997
21. KDD Cup 1999 Dataset, Retrieved on November 21, 2018 from http://kdd.ics.uci.edu/databases/kddcup99/kddcup99.html
22. Mirsky, Y. Doitshman, T. Elovici, Y. Shabtai, A. 2018. Kitsune: An Ensemble of Autoencoders for Online Network Intrusion Detection. 10.14722/ndss.2018.23211.
23. Opitz, D. Maclin, R. 1999. Popular Ensemble Methods: An Empirical Study. 11. 10.1613/jair.614.
24. Jolliffe, I. T. Cadima, J. 2016. Principal component analysis: a review and recent developments. Philosophical transactions. Series A, Mathematical, physical, and engineering sciences, 374(2065), 20150202.
25. Olusola, A.A., Oladele, A.S., & Abosede, D.O. (2009). Analysis of KDD ’ 99 Intrusion Detection Dataset for Selection of Relevance Features.

26. February 28th DDoS Incident Report, 2018, Retrieved on November 23, 2018 from https://githubengineering.com/ddos-incident-report/
27. Singh, A.P. Singh, M.D. 2014. Analysis of Host-Based and Network-Based Intrusion Detection System, I.J. Computer Network and Information Security, 2014, 8, 41-47. Published Online July 2014 in MECS
28. Jyothsna, V. Rama Prasad V.V. 2011. A Review of Anomaly based Intrusion Detection Systems. International Journal of Computer Applications (0975 – 8887) Volume 28 – No. 7, August 2011
29. Kotsiantis, S.B. 2007. Supervised Machine Learning: A Review of Classification Techniques. In Proceedings of the 2007 conference on Emerging Artificial Intelligence Applications in Computer Engineering: Real Word AI Systems with Applications in eHealth, HCI, Information Retrieval and Pervasive Technologies, Ilias Maglogiannis, Kostas Karpouzis, Manolis Wallace, and John Soldatos (Eds.). IOS Press, Amsterdam, The Netherlands, The Netherlands, 3-24.
30. Ghahramani, Z. 2004. Unsupervised Learning. Gatsby Computational Neuroscience Unit, University College London, UK