8-15-2018 10:30 AM

INSecS: An Intelligent Network Security System

Nadun Rajasinghe, *The University of Western Ontario*

Supervisor: Samarabandu Jagath, *The University of Western Ontario*
Co-Supervisor: Wang Xianbin, *The University of Western Ontario*
A thesis submitted in partial fulfillment of the requirements for the Master of Engineering Science degree in Electrical and Computer Engineering

Abstract

There are new challenges in network security, introduced by the nature of modern networks like IoT systems, Cloud systems, and other distributed systems. System resource limitations in IoT, delays in processing the large stream of data from Cloud and distributed system, incapability to handle multi-step attacks due to delay in updates, limited datasets used for Intrusion Detection System (IDS) training which impacts the system performance are some of the pressing issues. To address these challenges, the author proposes Intelligent Network Security Systems, a framework that can handle these issues and also be as accurate as a commercial grade IDS. The proposed framework consists of three components: a Dataset Creation Software (DCS), an Intrusion Detection System and a Learning module. This thesis presents implementation details and validation results for DCS and IDS.

The first component is a highly customizable software framework capable of generating labeled network intrusion datasets on demand. This software is able to collect data from a live network as well as from a pre-recorded packet capture file. The output can be either Raw packet capture (PCAP) with selected attributes per packet or a processed dataset with customized attributes related to both individual packet features and overall traffic behavior within a time window. The abilities of this component are compared with a state-of-the-art dataset creation system through a feature comparison.

The proposed Intrusion Detection System is a novel, distributed IDS that is able to perform in real-time in a distributed system. Hierarchical decision making is used to reduce traffic overhead on the IDS and allow faster Intrusion Detection. The IDS also detects multi-step attacks faster by updating the system rules when a reconnaissance attack is detected, without any human intervention. Internal attacks are also detected easily because of the distributed nature of the IDS. The performance tests show that the IDS performs 8 times faster on averages with the hierarchical decision-making structure and still maintains the same level of accuracy as Snort.

Keywords: Intrusion Detection, Intrusion Datasets, Distributed System Security, IoT and

Acknowlegements

The work presented in thesis is mainly possible due to the help, support and guidance from quite a number of people. I thank my Supervisor, Dr. Jagath Samarabandu, who provided me with the opportunity to work under him with constant guidance and support in many situations connected to my work and in my life. My Co-Supervisor, Dr. Xianbin Wang, was always helpful in providing great ideas and feedback in addition to the support provided for my graduate studies. It was a great privilege to work with both of them.

I am thankful to my group mates, Gobi and Wafaa, with whom I shared some wonderful times at the lab. I would like to specially thank Gobi for providing me some timely and valuable help when my project was going through a rough patch.

I also thank Professors at Western including, Dr. Serguei Primak, Dr. Olga Veksler, Dr. Aleksander Essex, Dr. Vijay Parsa and Dr. Pirathayini Srikantha for sharing their knowledge in multiple domains and providing me with a graduate level learning experience.

My Wife, Achini Abayakoon, has been a great partner in life without whom I would not have had the same ability to dedicate my time for work. I thank her for all the support and companionship given to me while completing graduate studies herself. My parents and sister, Chandra Rajasinghe, Ranjith Rajasinghe and Nirmi Rajasinghe, made me into who I am today and none of this would have been possible without them.

I have always considered myself lucky to have known friends from London, who made my time at Western much more interesting. The list is too long to mention but they will always have my utmost gratitude.

Contents

List of Figures

List of Tables

List of Appendices

List of Abbreviations

IDS - Intrusion Detection System

IPS - Intrusion Prevention System

NIDS - Network Intrusion Detection System

HIDS - Host Intrusion Detection System

DCS - Dataset Creation Software

PCAP - Packet Capture

IoT - Internet of Things

DARPA - Defense Advanced Research Projects Agency

KDD - Knowledge Discovery and Data Mining

INSecS - Intelligent Network Security Systems

INSecS-IDF - Intelligent Network Security Systems Intrusion Detection Framework

INSecS-DCS - Intelligent Network Security Systems Dataset Creation Software

INSecS-IDS - Intelligent Network Security Systems Intrusion Detection System

MIT - Massachusetts Institute of Technology

CSV - Comma-Separated Values

JSON - JavaScript Object Notation format

PDN - Primary Decision Node

CDE - Complex Decision Engine

RNN - Recurrent Neural Network

NIDES - Next generation real-time Intrusion Detection Expert System

DNS - Domain Name Service

ARP - Address Resolution Protocol

TCP - Transmission Control Protocal

UDP - User Datagram Protocol

SQL - Structured Query Language

IP - Internet Protocol

OS - Operating System

CPU - Central Processing Unit

ML - Machine Learning

HTTP - Hyper Text Transfer Protocol

SMTP - Simple Mail Transfer Protocol

SSH - Secure Shell

IMAP - Internet Message Access Protocol

POP - Post Office Protocol

FTP - File Transfer Protocol

SSL - Secure Socket Layer

DHCP - Dynamic Host Configuration Protocol

CIA - Confidentiality Integrity Availability

SVM - Support Vector Machines

GUI - Graphical User Interface

PPS - Packets Per Second

CEP - Complex Event Processor

Chapter 1

Introduction

1.1 Motivation

Intrusion detection and prevention has become an important part of cybersecurity in recent times due to the increase in network-based attacks and intrusions happening on computer systems across the world. Systems that have been thought of as being very secure have ended up being hacked by intruders whose identity, in some cases, is just limited to a country name. [74].

Network intrusion techniques are continuously evolving and intrusion detection algorithms must evolve alongside to keep pace. Network Intrusion Detection System (NIDS) community builds and tests security algorithms based on standard datasets. A common problem with standard datasets is that they are a snapshot (often simulated) of intrusion and normal traffic at the time of creation and quickly becomes outdated when newer types of network intrusions occur [69]. Another issue with datasets is the limitation of the attacks that are presented in them [69, 12].

Intrusion Detection Systems (IDS) encounter different types of traffic depending on the type of network and the nature of services offered. However, standard datasets are created on a specific network under a specific scenario. For example in the case of DARPA dataset,

they created their own network and collected intrusion traffic for the attacks they simulated [10]. The other popular datasets that were derived from DARPA were KDD 99 and NSL-KDD which also suffer from the same drawback [40].

There have been attempts to create more recent datasets by different researchers with improvements in the number and quantity of attacks, the quality of networks where data is captured from, attack simulation methods etc. [9]. Still, when a user tries to train their IDS with these newer datasets, they will be training on a subset of scenarios the creator of the data set faced even though the user of the IDS will not be facing the same threats locally [20]. It is also likely that the user runs different versions of software and services than what was used by the dataset creators, causing the user to have a different set of vulnerabilities.

Another factor to consider is the security features that were present in the network that hosted the platforms used for creating the data sets. Firewalls and encrypted communication channels are common security features in a network and traffic from such a network is different from traffic from a network without those features. This acts as further justification for the need of a method to create custom data sets that are specialized for local traffic.

One factor affecting the intrusion prevention efforts is that the volume of data that needs to be protected has increased exponentially. Thus, network traffic, which is generated as a result of people accessing all that data, has also increased. This increase in traffic is responsible for causing delays in the intrusion detection/prevention efforts due to its volume per unit time [65]. Some systems drop packets as a solution to this while others use distributed approaches to optimize intrusion detection.

Another trend is the increased use of Cloud computing throughout many fields. Cloud services now offer infrastructure, software and platform services all bundled together at very affordable rates. Due to the improvements made in communication and related infrastructure, many commercial entities find Cloud services a better alternative to housing and managing all their computing resources themselves. While accepting all the advantages of Cloud computing systems we also need to accept the risks involved when it comes to security. Cloud computing

has introduced new challenges that did not exist with in-house computing resources and thus, intrusion detection/prevention has to improve to keep up [42, 7].

Internet of Things (IoT) is another field that has emerged as an important part of modern technology in this age of connectivity. Traditional systems are being replaced by smart devices that are networked together to be more efficient and work without human intervention. This is a distributed computing challenge that requires new levels of security because of the inherent problems with system resources and communication in most IoT devices [47],[56]. In recent times, many IoT systems have been under attack from DoS and Botnet attacks, which makes them a liability and often renders them useless in many cases. This has prompted new security research directions such as vehicular security, smart home security, smart-grid security, etc.

Traditional intrusion detection/prevention systems are particularly vulnerable in today's environment because of the speed at which threat models and attack strategies are changing. The attack that was used today is not an attack that a system would face tomorrow. The traditional IDS/IPS needs to be updated by a system admin to keep up with the challenges introduced by constantly improving hacking tools, attack scripts and knowledge about the latest vulnerabilities. This whole process of manual updating causes a delay that makes the production systems (systems protected by the Intrusion Detection Systems) vulnerable to complex attack patterns (multi-step attacks). Detecting the reconnaissance attacks first and then, preparing for more harmful attacks to immediately follow them is absolutely crucial in defending against such attacks.

Given these challenges, modern intrusion detection/prevention systems should have the capability to handle these challenges and offer reasonable assurances that they are capable of providing security for modern information systems. Another factor that hinders the performance of Intrusion Detection Systems is incomplete or unsuitable datasets. Any IDS trained using such a dataset would inherit the associated issues. After careful consideration of the issues presented above, I propose Intelligent Network Security Systems-Intrusion Detection Framework (INSecS-IDF) as a system capable of providing a comprehensive solution to Intru-

sion Detection. Figure 1.1 shows how the proposed framework work in an example situation where one mobile device is connected to the network. Even with multiple devices connected to the network, the intrusion detection process would be the same.

INSecS Intrusion Detection System (INSecS-IDS) is one of the three components of the proposed Intrusion Detection Framework. The other two components- INSecS Dataset Creation Software (INSecS-DCS) [53] and the Learning module add critical functionality to the IDS, thus completing the INSecS Intrusion Detection Framework collectively. INSecS-IDF is to be deployed in a network in such a way that INSecS-DCS creates datasets at regular intervals, the learning module extracts rules out of the datasets using an unsupervised learning technique and finally, INSecS-IDS performs intrusion detection based on both the newly generated and existing rules.

1.2 Contributions

In this document, the author is presenting INSecS-Dataset Creation Software and INSecS-Intrusion Detection System while claiming the following contributions,

1. INSecS-DCS, an on-demand dataset creation software that can be run on a network of choice and gives the user the ability to fully customize the dataset according to their requirements. This includes the ability to make a dataset with new attributes, make attributes with temporal properties and collective properties, change attributes, etc. This provides a solution for the weakness in intrusion detection research related to inadequate or unsuitable datasets. This software is available under MIT [1] license so that the research community is able to create many new and relevant data sets by customizing every aspect of the dataset.

2. INSecS-IDS, a distributed IDS fit for modern networks due to the design and architecture, which is aimed at supporting IoT, Cloud and distributed systems. It detects intru-

[1] Massachusetts Institute of Technology

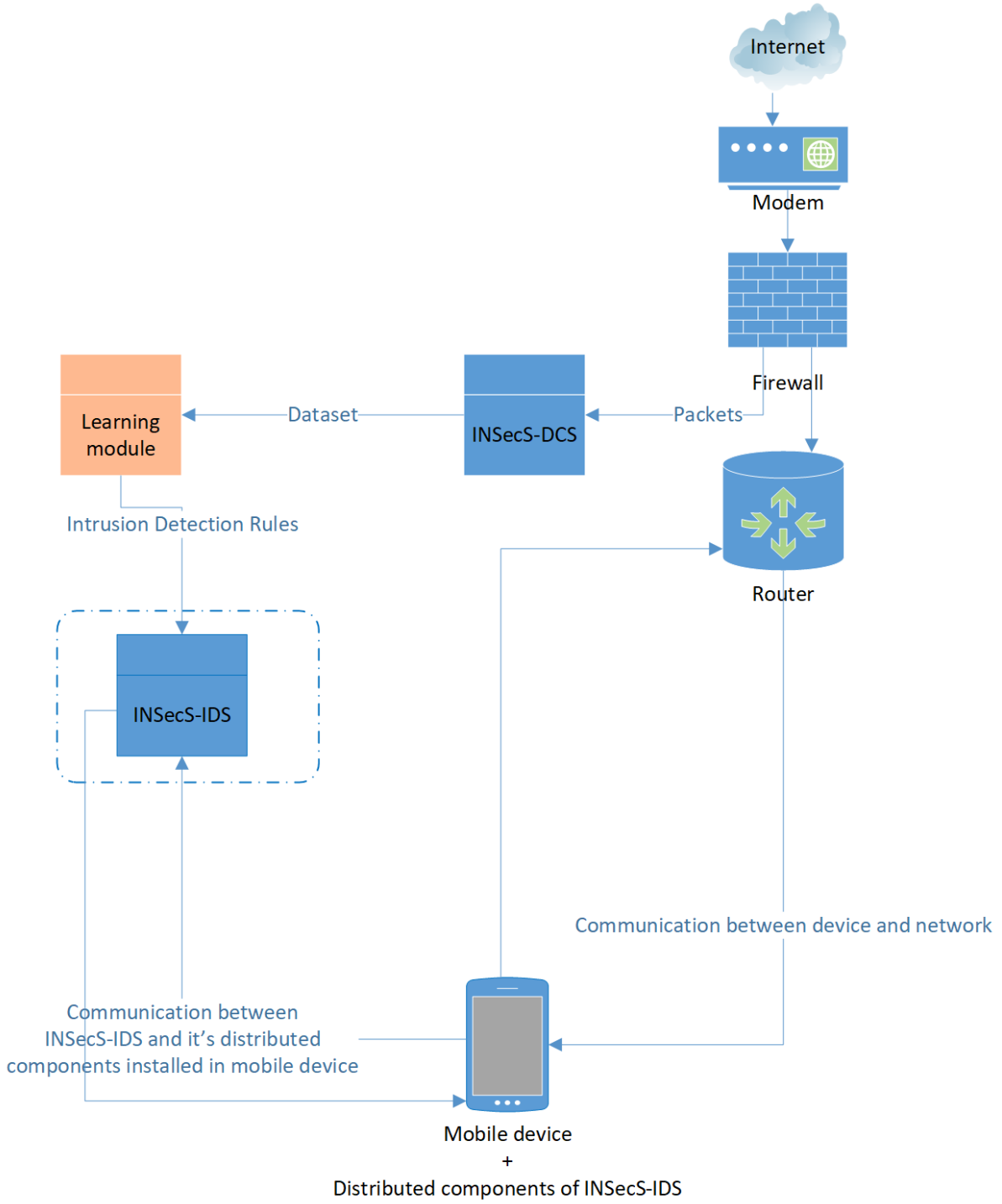

Figure 1.1: Design and Architecture of INSecS Intrusion Detection Framework (INSecS-IDF) with one device in the network.

sions faster by using a hierarchical decision making process, works in distributed setups, detects complex attacks (multi-step) faster due to the system being ready for the second attack, updates intruder information automatically enabling IDS to detect attacks from known intruders faster and accurately and detect internal and external attacks. This approach is different from the conventional IDS approach and results show that compared to the conventional approach, the proposed IDS performs 8 times faster on average in a test scenario.

3. In the absence of a single survey covering security of all modern distributed systems, this thesis provides a survey of network attacks and existing IDS solutions in the context of IoT, Cloud and distributed computing setups.

4. Provide flexibility within the IDS enabling users to replace the proposed Complex Decision Engine (CDE) with their choice of Decision Engines.

5. A comparison of the performance of INSecS-IDs compared to Snort [2].

6. Possible example setups for INSecS-IDS in IoT and Cloud applications.

A dataset generated by the proposed software framework includes not only the raw PCAP [3] files for each packet but also a processed dataset file in CSV [4] or JSON [5], that can be fed into a machine learning tool easily.

1.3 Document Structure

The remainder of this thesis consists of the following chapters. Chapter 2 provides an overview and details of related work on Intrusion Detection Systems and Intrusion datasets. Chapter 3 explains the implementation details of INSecS-Dataset Creation Software and INSecS-

[2] A popular intrusion detection system. More information is provided in section 2.1.3 under Chapter 2
[3] Packet Capture
[4] comma-separated values
[5] JavaScript Object Notation format

Intrusion Detection System. Chapter 4 provides test results and how our implementations compare with the competition. Chapter 5 on Conclusions and Future work gives example implementations of INSecS-IDS for hypothetical IoT and Cloud systems and an overview of how the proposed INSecS Intrusion Detection Framework incorporates the dataset creation software with the intrusion detection system.

Chapter 2

Background and Related work

2.1 Intrusion Detection Systems

Intrusions affecting computer systems are quite numerous and their attack strategies, behavior and consequences are different from one another. Due to this, different defense techniques need to be employed by security personnel for comprehensive protection.

The most common network-based attacks are given in Figure 2.1. Active attacks are where an intruder would use the attack as a means to change private information. In passive attacks, the intruder is observing to either learn more about the system, to steal private information or even to plan the next attack. Advance attacks are mostly done on routers or on neighboring nodes of the real target in order to make an attack possible. Given these different types of attacks, the most popular defense techniques are,

1. Firewalls
2. Intrusion Detection/Prevention Systems (IDS/IPS)
3. Encryption Techniques
4. Anti-virus software

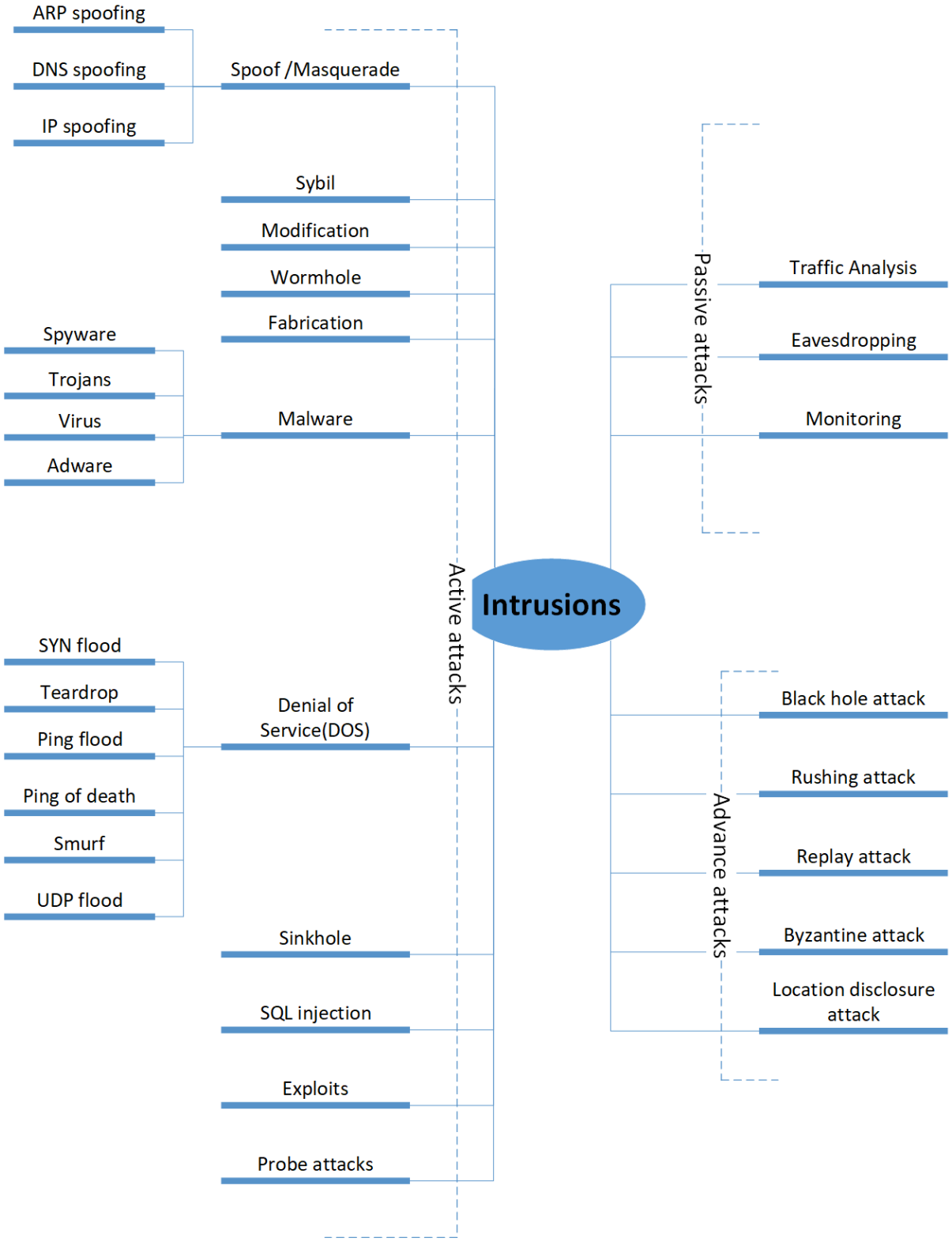

Figure 2.1: Taxonomy of some of the common Network-based attacks

Not all of the above-mentioned defenses are suitable for all situations and attacks. With the advancements in the field of cyber-security, anti-virus and host-based IDSs have developed very similar characteristics and have quite a bit of overlap. Some anti-virus software now comes with inbuilt IDS/IPS systems.

Intrusion detection systems have different classifications based on different qualities such as detection strategy, function and capabilities, the location of IDS, etc. There are different ideas expressed by different researchers on proper classification. However, Some of the popular classifications are given in subsection 2.1.1. Subsection 2.1.2 describes evaluation metrics used in IDS literature and subsection 2.1.3 gives a brief overview of some of the widely used Intrusion Detection Systems. There have also been attempts at addressing the challenges faced by modern intrusion detection systems in different forms.

2.1.1 Common IDS taxonomies

Intrusion detection systems are classified into the following categories based on the detection strategy [34].

Anomaly-based detection

As the name suggests, anomaly-based detection systems are built on the principle that legitimate users have a certain pattern of system usage and that a deviation from that is a possible intrusion. Failed login attempts, processor and memory usage, network connections, etc are examples of behavior trends used to build a behavior pattern. Due to this reason anomaly detection is also known as behavior-based detection. Anomaly detection can be used to detect attacks such as Denial of service (DoS), Trojans, exploits, masquerading etc. Since an anomaly is detected if there is a deviation from normal behavior, anomaly-based detection systems have the capability to detect unknown attack patterns. This means that we can detect new attacks without having prior knowledge of the nature of the attack. One drawback of this approach

is that in case of a misidentification, an unknown or new attack will be considered as normal traffic, which will increase the risk to the entire network unless there are additional security measures in place for such a situation.

To implement anomaly detection, several techniques are widely used.

- Machine Learning - By using labeled data for normal behavior and training a model, anomalous traffic can be identified using machine learning techniques[43]. K-Nearest Neighbor, Bayesian networks, Neural networks, Decision trees and support vector machines are some of the machine learning techniques found in the literature [49]. Currently, Recurrent Neural Networks (RNN) are quite popular among researchers for this purpose and they produce good results in identifying previously unknown attack sequences [64, 18].

- Statistical Models - Statistical methods are among some of the earliest methods to be used in intrusion detection and possibly triggered the move towards machine learning techniques. In this technique, statistical models are made for normal behavior and decisions are made under the assumption that intruder behavior is noticeably different from that of a normal user. Examples of effective statistical models in anomaly-based intrusion detection are,

    1. Stanford Research Institute's next-generation real-time intrusion detection expert system statistical component (NIDES/STAT) which collects data in the host over time and adaptively creates a profile for normal behavior [46].

    2. Haystack creates four vectors related to behavior and uses them in a weighting system to calculate an intrusion score based on which it determines how suspicious the current session is [46].

Signature-based detection (Misuse detection)

In signature-based detection, knowledge about attacks is used to detect intrusions. Each attack would have its own signature [5]. A port scan, an IP sweep, a DoS would have a unique signature, which can be used to detect it. This attack is also known as misuse detection in security literature. This method can only be used for known attacks and would fail if confronted with a new attack because of its reliance on prior knowledge about the attack. Common techniques used to implement a signature based intrusion detection system are listed below.

- Rule-based approach - This is the most widely used approach for misuse detection because it is quite straightforward and requires no training due to the fact that all information about the attack is known [24]. Implementation is simpler compared to machine learning and detection is faster overall because there is no training involved [17].

- Machine learning - Use of machine learning for misuse detection has proven to be effective according to most researchers [54], while misuse detection on KDD dataset has been reported to be unsuccessful based on tests [57]. Although popular algorithms used in anomaly detection are also popular in misuse detection, there are differences in the implementation. Traffic containing known attacks are used to train the model for various attacks, and test data are classified into different classes based on how close they are to a specific attack in comparison to the model.

Most practical intrusion detection methodologies use a hybrid model involving both anomaly detection and misuse detection combined, due to the fact that they complement each other. Anomaly detection would identify a behavior that is different from normal usage behavior and when attack signatures are used to match that anomalous behavior, an intrusion can be detected with much more precision and ease.

Intrusion Detection Systems (IDS) can also be classified into different categories based on the function and capabilities (scope of protection) [60].

Network-based Intrusion Detection Systems (NIDS)

In Network-based Intrusion Detection Systems, intrusion detection is done based on the data collected about network traffic [52]. To get network data, network traffic is mirrored to the IDS by a router or some other device with the capability to capture network traffic. NIDS can be placed at different locations in a network.

1. Between the Border Firewall and the Internet. This allows the IDS to listen to all incoming and outgoing traffic. This placement usually results in data overload conditions which most likely leads to delays in intrusion detection.

2. Between the Border Firewall and the Internal Network. This is a better placement because the firewall will block some traffic that is already known to be harmful. This allows the IDS to focus on traffic that truly needs to be checked.

3. Inside the Local Network. Placing the IDS at crucial points within the network is a strategy that would work well if the servers or systems which are most likely to be attacked are known. This will ensure that intrusion detection is faster than the previous two setups and that any internal attacks would be detected as well.

The main drawback of basing intrusion detection only on network data is that it would not work very well with encrypted traffic. Since encryption can be done at any layer, information required for an IDS to make a decision (regardless of detection strategy) would be inaccessible, if the traffic is encrypted.

Host-based Intrusion Detection Systems (HIDS)

HIDS use system logs, login attempts, system calls and other host-based information to detect intrusions [21]. There are known patterns of behavior related to certain attacks involving specific system commands and file creations which can be identified using the host information. This type of IDS can only be used in a single host and is more effective at doing so because

of all the additional data from the host activity. A host-based IDS would collect information under the following three categories.

1. Protocol Stack monitoring - This is similar to having an NIDS on a single host.

2. Operating System monitoring - OS monitoring gives access to information like login attempts, system calls and commands, unusual file creations, system registry modifications, file deletions, unusual communication attempts, etc.

3. Application monitoring - In this monitoring, the IDS looks at logs generated by individual applications.

Information collected under all three categories are used together to detect attacks. The main problem with these is that if the host is compromised, all the logs may be changed or deleted and thus make it impossible to detect the intrusion anymore.

Distributed Intrusion Detection Systems

Distributed IDSs gather data from multiple host-based detectors in order to optimize resource usage [15]. It also allows for faster detection and faster preventive action thus being an ideal solution for intrusion prevention. Another unique feature of distributed IDSs is that they can detect attacks occurring internally. Internal attacks can occur from malicious software installed somewhere inside the network or because of intruders who has access to the network. Distributed IDS have detectors at most network nodes within the network and localized detection is achieved faster and with precision because of the internal switching and routing information availability. The main disadvantage with distributed IDS is that the performance is directly affected by the communication between the distributed components. Network delays, security issues due to public networks, unreliable connections are some of the known issues.

2.1.2 Evaluation Metrics

To analyze the performance of intrusion detection several metrics are used and the most commonly used evaluation metrics given below [30].

1. True positive rate (TP) - Percentage of traffic identified as intrusions and are actual intrusions.

2. True negative rate (TN) - Percentage of traffic identified as non-intrusion (normal) and are actual non-intrusion (normal).

3. False positive rate (FP) - Percentage of traffic identified as intrusions but are actually non-intrusion (normal) traffic.

4. False negative rate (FN) - Percentage of traffic identified as non-intrusion (normal) but are actually intrusion traffic.

5. Precision - The number of true positives divided by the total of true positives and false positives (see equation 2.1).

6. Sensitivity - The number of true positives divided by the total of true positives and false negatives (see equation 2.2).

7. accuracy - The number of true positives and true negatives divided by all 4 possibilities (see equation 2.3).

8. Processing speed which is the amount of time it takes an IDS to classify all of the attacks in a dataset.

9. System utilization: the amount of memory and CPU required to run the IDS.

Precision, Sensitivity and Accuracy are defined in the following manner,

$$Precision = TP/(TP + FP) \tag{2.1}$$

$$Sensitivity = TP/(TP + FN) \tag{2.2}$$

$$Accuracy = (TP + TN)/(TP + FP + TN + FN) \tag{2.3}$$

Positive and Negative indicate the verdict of the IDS while the terms True and False indicate whether the verdict is right or wrong. Not all of them are used to evaluate a single IDS because different IDS have different purposes.

2.1.3 Widely used Intrusion Detection Systems

There are multiple intrusion detection systems that are freely available with some of them even being open source. The most popular ones and their features are given below.

- Snort: A free, widely used Intrusion Detection/ Intrusion Prevention System (IDS/IPS) [29]. It is a rule-based IDS/IPS which relies on the Snort company to provide the latest rules. It also has a set of community updated rules. Some of the advanced rules are only offered for paid subscribers. A comparison of Snort vs INSecS-IDS is given in section 4.2.1.

- Bro: An open-source, passive, Unix based network intrusion detection system that works well with large traffic loads [50]. A key advantage Bro has is its deep analysis involving a broad range of protocols. It has its own script which is used to write new rules if required.

- Suricta: Considered to be a close competitor to Snort, with an advantage over Snort in detection speed [60]. According to literature, Suricta supports Snort rules which makes it a strong performer as Snort is the most widely supported IDS.

- OSSEC: An Open Source, host-based Intrusion Detection System that works for most Operating Systems. It performs log analysis, file integrity checking, Windows registry monitoring, etc, within the host and communicates using encrypted channels with the OSSEC server for intrusion detection [1].

2.2 Background and Related work of INSecS-Dataset Creation Software

This section provides a background into intrusion dataset creation and related work. Section 2.2.1 explains the key factors to be considered in dataset creation together with comparisons on various strategies used in the standard datasets.

2.2.1 Dataset requirements

A dataset should possess certain qualities to be deemed fit to be used in the field of network intrusion detection and these have been identified and verified by researchers [79]. This section highlights several key requirements and their importance, along with details on how the requirements have been addressed.

Realistic nature of Data

Simulated attacks, generated using intrusion tools, are used to create intrusion traffic in most standard datasets [10][44]. The exception is datasets created using honeypots where the traffic you get at the honeypot is always intrusion traffic [63]. Both approaches produce realistic intrusion data and thus users are allowed to pick either option. This is achieved by operating the software package at any suitable network node and a honeypot, respectively.

Proper labeling of dataset

Labeling an intrusion dataset properly is critical regardless of whether it is being used to train an IDS or used for research. Insufficient or inaccurate labeling requires users to utilize unsupervised learning methods to determine any attack classes or manually label the dataset based on anomaly identification [62] which takes a considerable amount of time and effort. Our software does automatic labeling of records provided that the attack IPs are known.

Accuracy of normal traffic

When collecting data to be included in the normal traffic portion of the dataset, there is a possibility that intrusion traffic may get included in it, unknown to the dataset creators. This is due to undetected attacks present in traffic that is assumed to be normal. This is avoided by generating normal traffic through simulation or getting a trustworthy group of volunteers to generate the normal traffic by interacting with the network habitually.

Customizability of dataset

This is a novel and yet indispensable property of the output from INSecS-DCS, as compared to other dataset creation methods and tools discussed above. What is offered in INSecS-DCS is the chance for the software user to be able to add more protocols to be tracked, filter out the undesired traffic, decide on where the data collection should happen inside the network, decide which attributes to have in the dataset, change the time window for average traffic analysis and the ability to run the software inside the user's network, server or personal computer to see what the traffic looks like and use that data to find what the possible threats are.

Payload availability

This is a debatable topic due to the loss of privacy of the original owners of the captured traffic [39]. There are anonymizing techniques to hide this information [13]. However, it is not a service provided with the software. The ability to get the payload in the raw dataset format is allowed by design but the author leaves it up to the users to anonymize the data they capture when creating datasets inside networks.

Freedom to choose input

There are two choices offered in selecting data input to create datasets using our software. One is to run the software on a network node and generate a raw (records are in PCAP format) or processed (records are saved in CSV file in a tabular form, similar to NSL-KDD dataset

[38]) dataset based on captured traffic (see figure 3.1). The other option is to use raw PCAPs provided by other dataset providers and feed it to the software and generate a raw dataset or processed dataset.

Freedom to select format of output dataset

The dataset generated from our software can be in two forms. The raw form which only allows attributes related to individual packets to be present. The alternative is the processed format which allows attributes to represent individual packets as well as average properties of network traffic for a specific time window. The added advantage of this processed dataset is that it can be fed directly into ML algorithms without any preprocessing, which in turn saves a lot of time and effort.

2.2.2 Related work

Although tools that can create custom data sets have been proposed, they can only insert attack packets into the network traffic capture files and create a Raw dataset [69][12]. A key feature of the proposed dataset creation software (INSecS-DCS) is the ability to process the data captured from the PCAP files in order to construct customized and meaningful attributes in a logical manner. The output is similar to the KDD 99 and NSL-KDD datasets in presentation style, where the dataset can be directly fed into a learning algorithm with no preprocessing because of its tabular form.

Shiravi *et al.* [62] has proposed a profile based custom dataset creation methodology where profiles are derived from observed traffic to represent certain features or events captured from network traffic. The profiles are based on HTTP, SMTP, SSH, IMAP, POP3, and FTP protocols and the output is a raw dataset. In contrast, INSecS-DCS allows the users to add any protocol they wish to track and set any attribute they wish to see in their dataset.

2.3 Background and related work of INSecS-Intrusion Detection System

Intrusion Detection Systems have existed as a solution to intrusions for a considerable amount of time. Different IDS solutions have been proposed for various applications and scenarios. When an existing solution does not adequately provide protection from intrusions, new features have to be introduced into the Intrusion Detection Systems. This section is divided into two subsections. Subsection 2.3.1 on Security of Distributed networks, IoT and Cloud gives an overview of the security needs of those networks, existing security solutions and how those systems can benefit from an IDS, while subsection 2.3.2 on Related work provides a brief introduction to similar IDS implementations while .

2.3.1 Security of Distributed systems, IoT and Cloud

With the development of Distributed systems, IoT and Cloud systems, the need for their security has become new areas of research as people tend to rely on them more and more. With the expanding need for data storage, Cloud and distributed systems host a lot of personal and sensitive data, which makes them obvious targets of intruders.

Distributed systems

Distributed systems can be considered as a precursor of modern Cloud technologies. Because the system is distributed by nature, the system as a whole has more open ports for communication. This fact alone increases the chances of an intruder hacking into the system. Different approaches have been suggested in the literature for handling intrusion detection in distributed systems. Nezarat [45], has proposed using game theory to model the scenario involving a detection agent reporting from different nodes in the distributed system and an intruder. An IDS capable of detecting DoS attacks for distributed client-server systems using separate client detectors and server detectors has been proposed by Kshirsagar *et al.* [15]. The method uses a

rule-based approach to detecting attacks but details of rule generation are not explained. Snort has also been proposed to be used in a distributed setup because of its ability in general intrusion detection. By running different instances of Snort in a distributed manner, at different nodes across the network, researchers have controlled the load on each of them by careful resources usage observation [76]. However, due to processing all the traffic instances with multiple Snort instances, the researchers were not able to complete the experiment because of resource insufficiency.

IoT

More and more devices with multiple capabilities are being connected to the internet today and this has introduced the concept of Internet of Things (IoT). These IoT devices have computing and networking power embedded in them, enabling different software operations to take place without proper monitoring. This is why IoT devices are found to be part of Botnets, without being detected by the system administrators who use these devices [8]. Due to the system resource limitations, these IoT devices do not have the ability to deploy the defenses used in other computing systems. The main vulnerabilities in IoT can be listed in the following manner [33].

- Insecure web interface - Using the default password and having an insecure password recovery method are some examples of insecure web interface problems in many IoT devices.

- Insufficient or non-existent authentication/authorization

- Insecure network services - Open ports are used for communication and if the services using those ports have known vulnerabilities they will be exploited by hackers.

- Lack of transport encryption - If data is not encrypted, it can be read while the data packets are being transported.

- Insecure software/firmware - The authenticity of software that is installed in the device has to be verified by signatures. If not, an intruder can intercept the connection and send software with malicious code in them.

- Poor physical security - If the device is physically accessible and the data storage is not secure, an intruder can access, modify or delete the data.

Given this background, there is a clear need for suitable IoT security frameworks that work well in IoT system restrictions. Noorman *et al.* [47] has proposed an embedded security system with a separate microprocessor that relies on knowing each node in the IoT system. The isolation capabilities ensure that even if one node is compromised, intruders will not be able to hack into the other nodes using the compromised node. Traditional IDS would not work well with IoT systems because of the system resource issues in the IoT devices. There have been multiple approaches suggested to overcome this issue by having the IDS outside the IoT device[58],[41].

Lee *et al.* [32] proposed a distributed IDS solution based on analyzing the energy consumption at each node, to detect anomalies. Snort attack signatures have also been used in signature-based IDS implementations because they are easily accessible [48]. Jun *et al.* [28] proposed a centralized IDS based on a Complex Event Processor, which resulted in an IDS that was CPU intensive but consumed less memory.

Cloud Setups

Cloud service providers offer resources and services to consumers, who use them for various applications. With the continuous expansion of Cloud usage, the Cloud services and resources have become targets of attacks. Some of the security issues identified in Cloud computing setups have been identified by Mishra *et al.* [42].

- Data Breaches and Loss - Release and possible loss of data that was intended to be kept private. This includes any sensitive or personal data belonging to individuals or

companies. There have been some well known examples of this happening, during the past few years [11], [51].

- Account or Service Hijacking - By getting access to user credentials, intruders can gain access to the Cloud user accounts. This would enable intruders to see all the personal data associated with the compromised accounts and also initiate new data exchanges.

- Malicious Insiders - This is a human factor rather than a technical loophole in Cloud computing. Loss of security due to malicious insiders can occur easily if the people given access to the Cloud resources by the Cloud Service Provider are not properly filtered. Even then, there is no guarantee that there will be no malicious insiders.

- Hypervisor Vulnerabilities - Hackers ca figure out vulnerabilities in a hypervisor and exploit that to gain access to it. By gaining access to the hypervisor, intruders can change system resources, change access privileges and gain total control of the virtual machines connected to the hypervisor.

- Denial of Service - Legitimate users are denied access to their accounts and resources due to Denial or Distributed Denial of Service attacks.

Different techniques and methods have been proposed for Cloud security which can be categorized under Confidentiality, Integrity, and Availability (CIA triad) [7].

Confidentiality

Different encryption strategies have been proposed to protect data confidentiality from both the Cloud service provider and intruders. Fully Homomorphic Encryption (FHE) [66], Searchable Encryption (SE) [23], Onion Encryption (OE) [14], Attribute based encryption for secure scalable fine grained access control (ABE) [75] are some of the methods proposed for data confidentiality, identified by Basu *et al.* [7].

Protection of confidentiality of virtual machines in Cloud servers is as important as protecting data confidentiality. Proposed methods for this include algorithms and rules to be used by Cloud service providers in virtual machine hosting, virtual machine migration and maintenance [19].

Integrity

Data integrity solutions include using Tree Signature scheme [59] and Public key based homomorphic authenticator with random masking [72] and other similar techniques to ensure the integrity of the data in Cloud setups. Virtualization integrity issues are handled by using hashing, private and public key encryption [78] and activity log checks within the Cloud servers [37].

Availability

The main issue concerning the availability of data and virtual machines is Denial of Service and Distributed Denial of Service attacks and Intrusion Detection System have been used effectively to provide protection.

Similar to a local computer system, an intrusion detection system can be used in the Cloud server to detect intrusions [6]. An IDS that can identify the reconnaissance attacks can even alert the client that a possible hack or system compromise might take place. Depending on the Cloud service, the IDS can be a Host-based IDS [73] or a Network-based IDS [38].

2.3.2 Related Work

INSecS-IDS has introduced multiple capabilities that enable it to function effectively in a modern network. Some of these capabilities have been introduced by others in similar or different applications and situations. In this subsection, research that is similar to INSecS-IDS are presented.

The concept of multi stage intrusion detection has been explored by multiple researchers. A binary SVM classifier identifying anomalous traffic in the first step, followed by a self organized map for identifying the attack class is proposed in [67]. Multi Stage attack detection strategies are designed based on the argument that an attack can be broken down into multiple stages. For the attack to be complete, it needs to complete the different stages in a certain order. Herwono *et al.* [25] have proposed a multi-stage intrusion detection system which uses a Hidden Markov Model to predict the probability of an attack. Each prediction is done in the form of a state transition and the transition probabilities are calculated within a training phase. An attack is defined to have a certain set of states. A similar argument has been used in the design of an intrusion detection system which uses various machine learning techniques to detect states [68]. The authors have used a resource expensive methods for the second stage of the attack detection because they manage to reduce the traffic load by a resource inexpensive classifier in the first stage.

Security models in Intrusion Detection Systems get outdated really fast and IDS should have adaptively changing models to deal with threats. Adaptively changing the security model to face continuously changing attack patterns has been proposed in [55]. The clustering models are adaptively changed based on expert input from an outside source using extreme learning machines. Wahab *et al.* [71] has presented a Cloud IDS that adaptively changes its detection strategy using a sequential Stackelberg game between the intruder and Cloud system. Game theory is a well suited method to use in this situation because of the dynamic nature of its models.

Chapter 3

INSecS-Framework

3.1 INSecS-Dataset Creation Software

INSecS-DCS was proposed with the aim of creating Network Intrusion datasets specific to individual networks, where the network managers can specify their own attributes [53]. There are very few datasets that have proper documentation and can be used effectively to train and test an IDS. Although considerably old, NSL-KDD [27] is one such example. INSecS-DCS allows network managers to create a custom dataset, with attributes they feel are the most important for their network.

The workflow used in the dataset creation process is shown in figure 3.1. In this section, the procedure is described in detail. This software was developed in Python due to the availability of many libraries related to network packet processing. This also allowed this software to work easily with our future developments.

3.1.1 Dataset creation

The steps involved in the dataset creation is given in the following subsections of the thesis. Each step has its own software component available for public use under an MIT license. Instructions on using the software are provided in Appendix B.

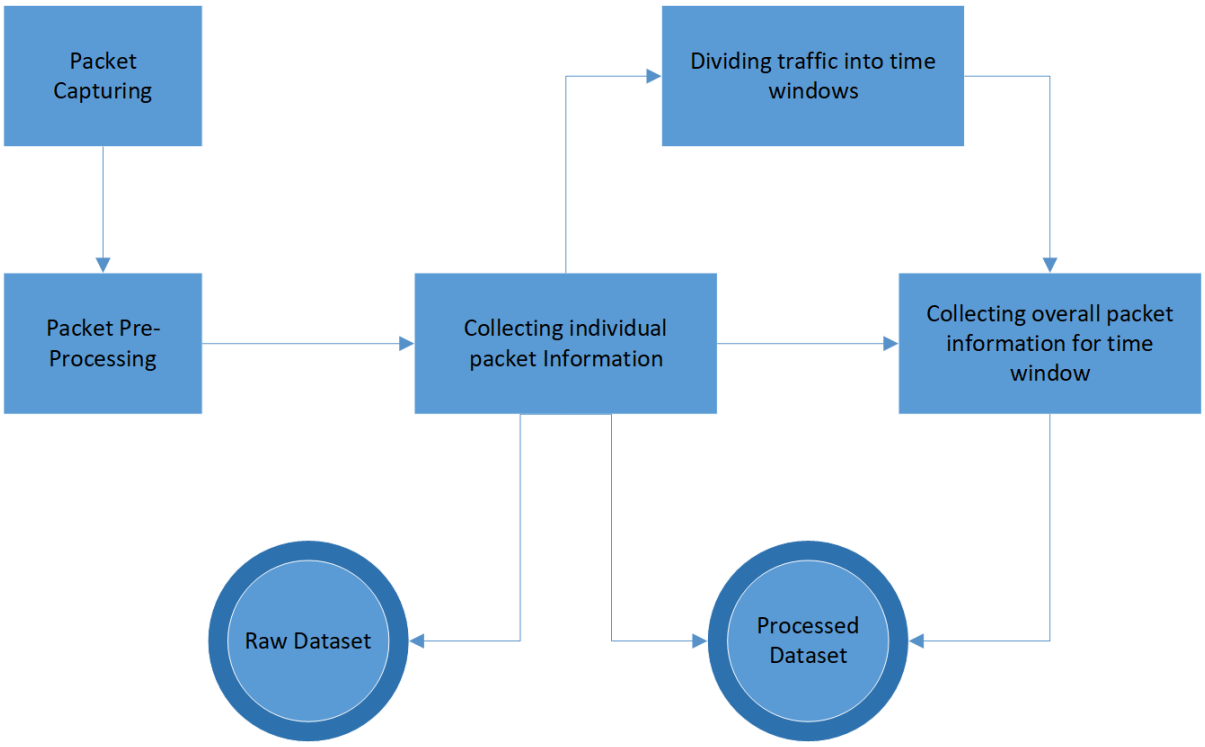

Figure 3.1: Architecture of INSecS-DCS

Packet capturing

The first step in dataset creation is capturing network packets. After evaluating several packet capture tools, *Tshark* [1] was selected as the packet capturing tool as it provides a great GUI to view the captured packets with custom filters. No filtering is done at this stage and all the information available on each packet is captured.

Packet pre-processing

Captured packets are in PCAP format and preprocessing involves converting these PCAP files to JSON format for easier processing. The rationale behind this is to connect future projects with this software without the need for type conversions. The pre-processing is done by the packet pre-processor, which converts the data in a packet into key-value pairs, where the key

[1] "TShark is a network protocol analyzer. It lets you capture packet data from a live network, or read packets from a previously saved capture file, either printing a decoded form of those packets to the standard output or writing the packets to a file. TShark's native capture file format is pcap format, which is also the format used by tcpdump and various other tools." [2].

Table 3.1: Overall attributes for time window

Feature name	Feature description
connection_pairs	the number of different source and destination pairs.
IP_addresses	the different IP addresses used in the connections
num_ports	number of different port numbers used
num_packets	total number of packets sent and received
src_bytes	the total amount of source traffic
dst_bytes	the total amount of destination traffic
tcp_frame_length	the total amount of frame bytes for TCP traffic
tcp_ip_length	the total amount of IP data for TCP traffic
tcp_length	the total amount of TCP data
udp_frame_length	the total amount of frame bytes for UDP traffic
udp_ip_length	the total amount of IP data for UDP traffic
udp_length	the total amount of UDP data
arp_frame_length	the total amount of frame bytes for ARP traffic
num_tcp	total number of tcp packets sent and received
num_udp	total number of udp packets sent and received
num_arp	total number of arp packets sent and received
num_ssl	total number of packets containing SSL traffic
num_http	total number of http requests that were sent and received
num_ftp	total number of ftp packets sent and received
num_ssh	total number of packets containing SSH traffic
num_smtp	total number packets containing SMTP traffic
num_dhcp	total number packets containing DHCP traffic
num_dns	total number packets using DNS traffic

Note 1: The data type is Discreet Integer in all of the listed attributes.
Note 2: Feature name column shows the attribute names given in the code.

is the name given by *tshark* for the value. The generated key-value pairs are then stored in a python Dictionary.

Collecting individual packet features

During the next step, the features related to individual packets are collected by selecting key-value pairs of interest. These features include the protocols used (such as TCP, UDP, IP, FTP, SMTP, SSH, SSL, ARP, DHCP, HTTP etc), the source and destination addresses (IP address) and port numbers used etc. The software comes with a list of preselected attributes but the user can customize this.

Dividing the individual traffic flow into time windows and collecting overall features

An important step in creating a processed dataset involves selecting a time window and analyzing the traffic flow during that time [3]. This will give information about the overall traffic behavior during given time interval and identify common trends in traffic; as opposed to considering information from individual packets only. For example, if both TCP and UDP packets are used in an exploit, analyzing the TCP stream alone or the UDP stream alone would not suffice. With that taken into consideration, parameters about total traffic flow, such as total length of TCP packets sent, total length of UDP packets sent etc. are collected. This is done for a time window, where the length is chosen by the user. Descriptions on setting the time window are given in related Appendix section.

3.1.2 The processed dataset

The term ”Processed dataset” in this thesis describes an intrusion dataset that is converted to a tabular format. The format is similar to that of the popular NSL-KDD dataset, which provides the dataset in a tabular manner. The processed dataset can be considered as the standard output of the INSecS-DCS. The attributes in the dataset contain information related to individual packets as well as information about overall traffic behavior during a specified time window. This is a much better dataset in terms of usability and usefulness as explained previously. Details of the attributes collected in the dataset are included later in this section. The steps in

creating the processed dataset are depicted in figure 3.1.

The attributes in the processed dataset

The inbuilt attributes in the processed data set that are related to overall traffic behavior within a time window, are given in the table 3.1. It should be noted that these attributes were picked as examples of possible attributes that capture overall information for a specific time. The purpose of presenting these attributes is to show the capability to extract important attributes for specific conditions that might serve the research efforts better than attributes from already available datasets. Attributes related to individual packets include protocols used, flags set, source and destination information etc. Since the main objective of INSecS-DCS is to provide customizability in intrusion datasets, I do not believe in dictating which attributes are to be included in the dataset. So a key factor to be considered here is the ability for the user to add more features or remove features in their dataset creation.

3.1.3 The raw dataset

The unprocessed or raw dataset is a dataset that presents data in a raw PCAP format. Each record contains attributes related to a single packet which has been preprocessed. The attributes include all the data related to the packet, including payload. This may introduce privacy concerns with encrypted traffic, as the payload may contain sensitive information. The author expects users to use this feature responsibly. Figure 3.1 shows the steps involved in creating the raw dataset.

3.2 INSecS-Intrusion Detection System

The proposed Intelligent Network Security Systems Intrusion Detection System (INSecS-IDS) is a real-time, distributed IDS that uses horizontal scaling and hierarchical decision making to deliver an adaptive intrusion detection system. Making a true real-time IDS that has fast re-

sponses is a challenge on many fronts. Several major issues faced in developing such a system have been addressed in a novel manner fitting an industry standard IDS. In the remainder of this section, the author will present the overall architecture of INSecS-IDS and the features introduced that makes it better suited as a standalone IDS for any setting.

INSecS-IDS was designed after a careful study of commonly available IDS and their strengths an weaknesses. What I gathered from that study enabled me to fuse what was learned from the existing IDS technology, with real-time stream data handling. The latest literature on new trends in IDS focuses on IDS designed for Cloud and Internet Of Things (IoT) systems, which have different requirements than the traditional IDS [77],[61]. Traditional Intrusion Detection Systems are not flexible enough to change according to these requirements. This situation has introduced a need for an IDS that would work across multiple platforms and applications, including traditional networks, Cloud environments, IoT systems and other modern distributed computing setups [22]. This subsection gives an overview of INSecS-IDS and where it stands in IDS classification.

3.2.1 Overview and capabilities of INSecS-IDS

The design of INSecS-IDS involved several important decisions that were aimed at addressing the challenges identified, in comparison to the competition.

- First, our research group identified the need for a hierarchical, but distributed, decision-making structure as the key to having an IDS architecture that works across multiple platforms. The structure is hierarchical because each node makes simple decisions while a central, dedicated complex decision maker is used to achieve the final verdict on intrusions. The distributed nature of the IDS is due to the fact that, each node connected to the network does some job related to intrusion detection while maintaining overall connectivity. Having such a structure allows us to make simple decisions at any node in the network and leave the complex decision making for a dedicated entity, while maintaining the distributed nature required by Cloud, IoT and other distributed setups. This

is further validated by surveys done on possible strategies and technologies for IoT and Cloud, where a distributed strategy has the least overhead and faster, localized detection [77].

- Intrusion detection strategy used in INSecS-IDS is an adaptive strategy that learns from already detected attacks and updates the rules by itself. This allows the proposed IDS to function as an Intrusion Detection System that has adaptive learning capabilities, allowing it to function for an extended period of time without an operator updating the rules. This also allows faster intrusion detection of attacks from known intruders.

- Another built-in capability that gives INSecS-IDS the edge, is its ability to face complex attacks (multi-step attacks). It is the usual practice of an intruder to scan for potential targets to attack before launching many of the attacks. They look for vulnerable systems, databases, useful web servers, etc, and finding the ports open for communication is an important piece of information [70]. Also, knowing which machine hosts the services inside a network makes it easier for an intruder to target attacks. Because of this, Port scans and IP sweeps act as reconnaissance attacks, leading way for more dangerous attacks like DoS, SQL injection, exploits, botnets, etc. This sort of attack pattern is considered a multi-step attack or complex attack pattern.

- Ability to detect internal attacks is a key characteristic of distributed intrusion detection systems. It is impossible for NIDS with a central architecture, to detect internal or local attacks such as ARP spoofing, spreading of botnets inside the network and man in the middle attack.

- HTTP is used as the communication protocol between the components of the IDS, making the IDS simpler to be implemented anywhere without the need for additional hardware or software.

- Tshark, which is a GUI less version of Wireshark that works on most OS, is used for

packet capturing in INSecS-IDS. This ensures that the IDS works irrespective of the OS and enables the usage of Wireshark's advanced packet analyzing features.

Classification

IDS classification has different forms, as shown in Section 2.1 on Intrusion Detection Systems. INSecS-IDS is a signature based (Misuse) detection system in terms of detection strategy and a distributed intrusion detection system based on the scope of protection. Although a part of the IDS is hosted on each node, it does not use host information like systems logs and relies only on network information.

3.2.2 Logical Architecture

A major challenge in having a real-time IDS is handling the large volume of traffic. A typical IDS would have to process each and every packet to effectively provide security and protection. Hence, the time it takes the IDS to process all the traffic generates a considerable delay in the intrusion detection process. The proposed IDS architecture handles this problem by using a hierarchical architecture. The argument behind this design is that running each and every packet through a complex filtering process consumes too much time and resources. By having a logical hierarchy in the decision-making, resource and time usage were optimized. This enables quicker intrusion detection and less resource usage. The logical hierarchy is shown in figure 3.3. Primary decision-making module is allowed to pass judgment on the oncoming stream of packets and let the complex decision maker spend it's resources only if there is a possible threat. Doing so ensures that the full intrusion detection capabilities are spent, in the form of complex event processing, only on traffic that has a probability of being associated with an intrusion. In figure 3.2, these two logical stages are shown as stage 1 and stage 2.

The primary decision-making module has been named Primary Decision Node (PDN), which is placed at each user node in a network. A PDN is responsible for checking each packet for a pre-defined simple condition. These conditions are typical behavior associated

with known attacks. The process of selecting and deciding on these conditions is given in section 3.2.4. If the traffic satisfies the conditions, there is reason to suspect that the traffic is involved in an intrusion or attack. PDNs then generate a data unit with selected information about the suspected packet, which is then forwarded to a more complex decision maker. This generates a stream of Data from each of the PDNs placed within the network.

INSecS-CDE (INSecS-Complex Decision Engine) is the complex decision maker introduced in INSecS-IDS. This decision engine has the capability to receive incoming streams of data and make a decision on whether the related packets are involved in an intrusion or not. The CDE is capable of analyzing all the packets in a defined time window and checking for complex conditions. The implementation of time windows and the process of applying complex conditions is given in detail under section 3.2.5. Once the CDE detects an attack, it updates the IDS with information which allows the IDS to form new rules and conditions. This allows the IDS to update itself and to detect multi-stage attacks with the fastest possible detection time.

Multi-step attack detection allows the IDS to deploy new PDNs in the network node which faced the original attack. These new PDNs detect known attack sequences like Port scans and IP sweeps followed by DoS, Botnets, vulnerability exploits, SQL injection, etc. CDEs that are necessary to these second stage attacks are already in place so that no additional effort is required by the system admins to deploy new CDEs.

3.2.3 Physical Architecture

The physical architecture and workflow of INSecS-IDS are given in figure 3.2. The diagram relates to an example scenario as is is practically impossible to draw a generic diagram for an entire network. Different symbols used in the diagram shows different types of functional blocks. They can be listed in the following manner.

1. IDS Controller

2. Packet Handler

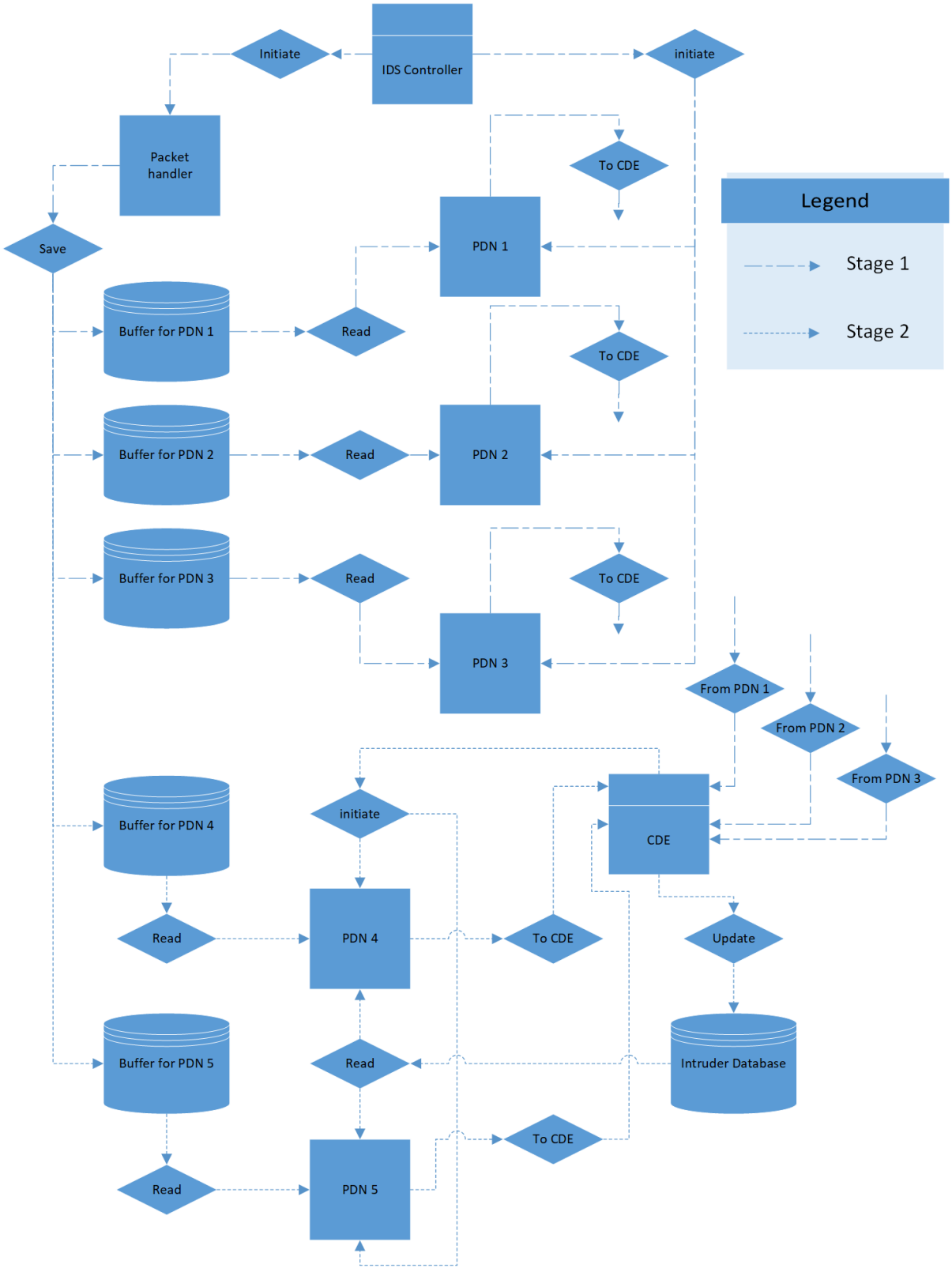

Figure 3.2: Architecture and workflow of INSecS-IDS for an example setup with 3 PDNs placed in a single network node working with the CDE placed in a central server.

3. Primary Decision Node (PDN)

4. Complex Decision Engine (CDE)

5. Storage and Buffer units

There are two types of storage and buffer units used in the setup. See Storage and Buffer units under subsection 3.2.3 for more details.

The objective of each functional block can be described in the following manner.

IDS Controller

IDS controller is the unit placed inside each network node and is responsible for initiating the Packet handler and the initial Primary Decision Nodes. This gives the IDS user the ability to use the IDS controller unit to select which PDNs are to be deployed in which network nodes.

Packet Handler

Packet handler is a unit initiated by the IDS Controller and does a few crucial tasks in handling the packet stream. There is one packet handler at each network node that needs the IDS. As soon as the packet handler is initiated, it runs Tshark, which captures the stream of packets going in and out of the node and generates a JSON object with all the packet attributes and values for each and every packet. This JSON object is then pre-processed in order to generate a Python dictionary, which is broadcast to all the active buffers in a network node. There is a buffer for each active PDN and the corresponding PDN reads the stream of dictionaries continuously.

Primary Decision Node & Complex Decision Engine

PDNs are functional units that check simple conditions that correspond with known intrusions and attacks in order to generate events. PDNs required for the initial deployment of the IDS are selected by the person deploying the IDS. Depending on the network characteristics, the PDNs

Table 3.2: Buffer and Data Storage units placement

Unit type	task	Using entity
Buffers	Act as a data buffer	Primary Decision Node
	Act as a data buffer	Complex Decision Engine
Data Storage	Store Possible Intruder IDs	Primary Decision Nodes

that are most suitable to be used in the initial deployment are listed in appendix A. Details on the architecture, functionality and design procedure of Primary Decision Nodes and Complex Decision Engines are discussed in detail in section 3.2.4 and section 3.2.5 respectively.

Storage and Buffer units

Data buffers and data storage units are required when handling the streams of data involved in several units of INSecS-IDS. Table 3.2 lists the type of unit, the expected task and where the unit is used. As illustrated in figure 3.2, the packet handler sends multiple copies of the same data stream onto data buffers. Each primary decision node then reads from the buffer that is allocated for that specific PDN. The reason for having the buffer here is to prevent the processing of the PDNs from interfering with the data flow within the IDS. If the buffer was not used, the packet handler would have to wait until the condition checking is completed for one packet by the PDN, before sending the next packet. This will slow down the packet handler, and consequently hinder the productivity of the IDS. Once the PDN suspects any traffic to be involved in an intrusion it would generate an event, which is then forwarded to the Complex Decision Engine. Inside the CDE, complex condition checks are done on the events sent from the PDNs to identify real intrusions. In some cases, this requires the CDE to break the event stream into chunks of events for specified time windows and perform condition checks on all events included in that time window. In order to stop this process affecting the event stream flow, a buffer is used here to store the events received by the CDE until the CDE is ready to process them. The data buffers used in both cases are Queues, operating on *first in, first out* (FIFO) strategy in order to maintain correct order in the data stream.

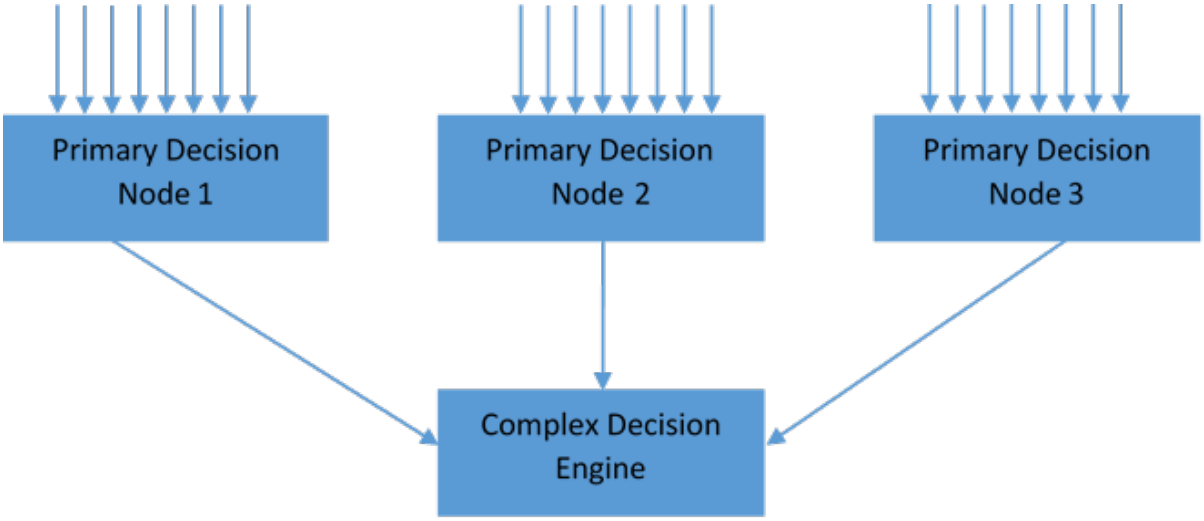

Figure 3.3: Hierarchical decision-making setup

3.2.4 Primary Decision Nodes

The idea of network event detectors in our implementation is based on the concept of hierarchical decision making. In hierarchical decision-making, there is less stress on the Primary Decision Nodes (PDN), which transfers the complex decision making workload onto a dedicated Complex Decision-making Engine (CDE). See figure 3.3. The nature of the Primary Decision Node in a hierarchical decision model would vary depending on the objective and placement.

The Primary Decision Node assumes the role of a simple event detector, in the proposed architecture. It looks for a pre-defined simple event and triggers complex action once the event is detected. The pre-defined events are the simplest possible conditions that can alert the Complex Decision Engine of a possible attack/intrusion. PDNs are placed at each of the network end-nodes including servers, personal computers, tablets etc. There are 2 main reasons behind the placement of the PDNs at each of the network end-nodes.

1. They help reduce the amount of traffic being inspected by IDS.

2. The PDNs are important components of the proposed distributed IDS.

Reduce traffic overhead on IDS

Network traffic flow in and out of a network can be analyzed using different metrics. Packets Per Second (PPS) is one such metric used to measure traffic flow volume per unit time[31]. This number can vary depending on factors like the number of users, the number of servers, the nature of services hosted on the servers, type of company the network belongs to etc. The average number of packets per second is the fairer metric to generalize traffic flow volume over a larger period of time like a day. For a typical server, this value would be stable, with circumstantial fluctuations. However, the traffic flow volume inside the network would be different from what is observed at the Gateway. As shown in figure 3.4, if we consider traffic flow volume per unit time at A and B, the two values would be different with B recording a higher value. A is just after the gateway to the network and B is inside the network covering all nodes inside the network. The difference between the PPS values is due to the local traffic that does not cross the gateway.

Given this background, one major drawback of having the IDS immediately after the firewall (at A in figure 3.4) is that the IDS needs to process each and every packet that enters the network. This is a considerable overhead for large networks which in some cases handles over a million packets per second. Under these conditions, an IDS would take a considerable amount of resources to process all packets and in the process cause a delay in producing a classification. This will cause the delay to increase with increasing traffic volume per unit time even if the IDS has dedicated hardware.

The PDNs are capable of filtering normal traffic and forwarding only suspicious traffic onto the CDEs. The performance advantage of this is verified under Chapter 4 on Results and Comparison.

Support provided for Distributed IDS

One key feature in INSecS-Intrusion Detection System is the built-in capability for distributed IDS. The need for distributed intrusion detection systems were discussed in detail in chapter 2

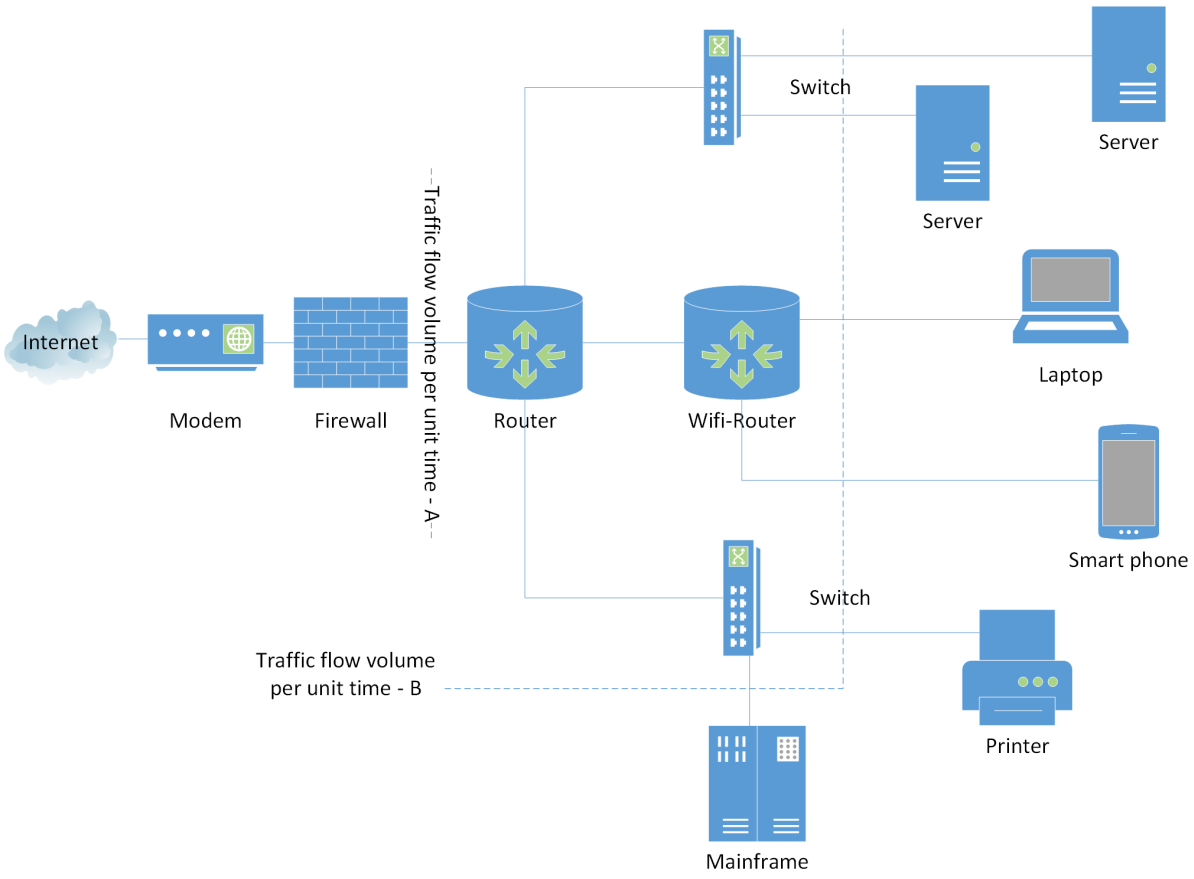

Figure 3.4: A typical network setup

under section 2.1.1. The architecture and implementation of PDNs that allow the distributed IDS will be discussed in this section.

The primary decision nodes are responsible for the detection of simple events and forwarding the detected traffic to a complex decision engine, as described earlier in this section. To detect these simple events, the PDNs use a pre-defined set of conditions pertaining to network attacks or intrusions and expert knowledge is required to identify these conditions. The INSecS-IDS comes with a set of PDNs for the most common attacks. Creation of additional PDNs, as required by users, is supported by the architecture.

Structure of PDN

An example PDN algorithm is shown in algorithm 1, which is used for the detection of potential Port scan packets. In this example, the PDN checks the incoming packet stream for packets with SYN flag set to 1. The identified packets are considered for potential threats and are sent to the CDE for further processing. The other PDNs follow a similar algorithmic pattern but have different condition checks based on the specific attacks they detect. Each and every PDN is implemented as a different thread so that there will be minimal interference between the functional blocks shown in figure 3.2.

The purpose of algorithm 1 is to monitor the packet stream in the network and detect packets that have a possibility of being involved in an intrusion. It also ensures that the suspected packets are in order, before sending the suspected packets to the Complex Decision Engine. For this particular attack, the CDE would divide the packet flow based on a defined time window and count the number of different ports in the destination, probed by the same source. If that number exceeds a certain threshold, the CDE would identify an attack. This model was made based on the packets captured during nmap Port scan attacks done on our servers using an external device.

Algorithm 1: Port scan Primary Decision Node

Input: Input and output Packet stream in the network.
Output: PDN events in the form of a JSON object consisting of predefined attributes.

```
1  StartTime = timestamp when packet receiving starts;
2  while Port scan PDN is on do
3      Data = get packet from PortsQ
4      if EndOfTransmission == True then
5          EndTime = timestamp when packet receiving stops;
6          Time= EndTime - StartTime
7      end
8      timestamp = timestamp on packet;
9      if previous timestamp <= current timestamp and tcp.syn.flag == 1 then
10         Request = send http request with selected attributes
11     end
12 end
```

Data Transfer from PDN to CDE

Our architecture supports a distributed IDS architecture, as described earlier. This means that the PDNs at each network node needs to send their selection of suspicious traffic onto the complex decision engine. The implementation of this step is shown on algorithm 1 in the line where an HTTP request is sent, given that the TCP syn flag condition and timestamp conditions are true. For the purpose of this communication, a simple HTTP client-server connection was chosen because of the wide availability of the protocol and ease of use, as opposed to developing our own communication standard. When a PDN communicates with the CDE it uses a port number unique to each attack. For example, all Portscan PDNs in the network use the same destination port number (a port number on the server hosting CDE) .

3.2.5 Complex Decision Engine

The complex decision engine (CDE), is a decision unit designed for complex decision making for data streams. It's the functional unit responsible for doing a complex analysis and make decisions based on the data streams sent by the primary decision nodes, which are located at each network node. The CDE is designed to be placed on a dedicated computing device

with considerable computing power at its disposal. This is in contrast to the PDN, which is designed for computing devices with low computing power and memory. Doing so ensures that the hardware usage is optimized throughout our IDS implementation.

Structure of CDE

Each attack type (like Portscan, Smurf, DoS, etc.) has its own complex decision making algorithm defined as a class, in the CDE. (The CDE class definition for Portscan is based off of algorithm 3). For each PDN in the network, there is a CDE module (object made using the class definition). Refer figure 3.5 for an example scenario involving three attack types and 4 PDNs. The module 1 and module 2 are made from the class definition for attack type 1. Module 3 is made from the class definition for attack type 2. Module 4 is made from the class definition for attack type 3.

The CDE is designed to act as a central decision maker, as described earlier in this section. When the CDE gets a packet from a PDN, it sends the packet to the correct CDE module. Since all PDNs belonging to the same attack type uses the same port number to communicate with CDE (example: all portscan PDNs in the network would send packets to port 8081 of the server hosting CDE) the CDE knows that all traffic on a certain port is from all the PDNs in the network related to one attack. So in order to send the packet to the correct module, the Stream Splitter (see figure 3.5) checks the incoming HTTP requests from a certain port to determine which PDN it is coming from. The HTTP request comes with a tag indicating the PDN. Using this information, the packet is sent to the correct CDE module. Algorithm 2 is used to implement the process described above.

Algorithm 3 is the complex decision-making algorithm used by the IDS to identify a Port scan attack at a specific PDN. The algorithm divides the traffic into time windows (1 second in this experiment) and counts how many unique ports in a host, received TCP packets with SYN flag set to 1. If the number exceeds 50 this is identified as an Portscan. The validity of this algorithm was verified using nmap.

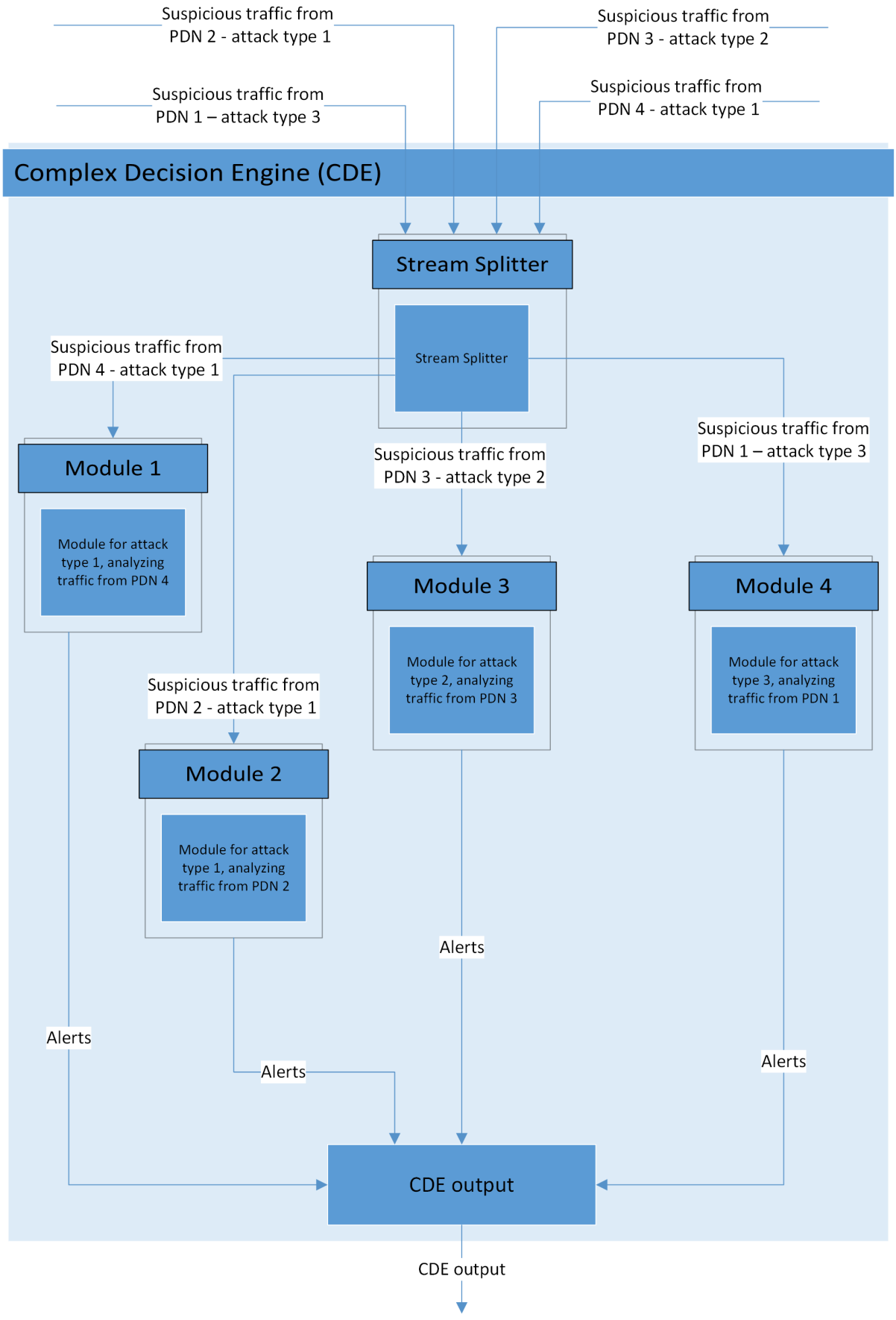

Figure 3.5: Structure of Complex Decision Engine for an example scenario

Algorithm 2: Directing traffic to the correct module

Input: Data stream from Primary Decision Nodes

1 *PostData* = read *http* request;
2 *data* = convert *PostData* to JSON object;
3 *PDNno* = get the PDN identity;
4 *PDNno.window*(*data*)

The validation of the proposed IDS is provided in the next chapter using experiments and results.

Algorithm 3: module functionality

Input: Data stream directed to the module
Output: Final verdict on traffic (Intrusion or not)

```
1  data = get data sent to the module; window = initialized Queue; icmp = object
     containing already known destination, port number pairs ;
2  packetcount = packetcount + 1;
3  if packetcount == 1 then
4      startTime = Timestamp of current packet;
5      endTime = startTime + 1000 (milliseconds); window.put(data)
6  else if endTime <Timestamp of current packet then
7      startTime = Timestamp of current packet;
8      endTime = startTime + 1000 (milliseconds);
9      while window is not empty do
10         packet = window.get();
11         key = "destination-IP" of packet;
12         port = "destination-port" of packet;
13         if key in icmp and port not in icmp[key] then
14             append.icmp[key](port);
15         else
16             ports = initialize list;
17             append.ports(port);
18             icmp[key] = ports;
19         end
20     end
21     for key,values in icmp do
22         if values >50 then
23             Port scan is detected;
24         end
25     end
26     Initialize icmp;
27 else
28     window.put(data);
29 end
```

Chapter 4

Results and Comparison

4.1 INSecS-Dataset Creation Sotware

The tool or software that is closest to the proposed dataset creation software (INSecS-DCS) is the ID2T toolkit developed by Vasilomanolakis *et al.* [69]. In order to evaluate INSecS-DCS, compared to ID2T toolkit, key advantages of INSecS-DCS are highlighted in table 4.1.

The attack generation to capture intrusion traffic is built into the software in ID2T implementation. This is done by attack scripts and PCAP modification. However, network attack generation softwares are already highly advanced and the hacker community keeps on developing better versions. Thus, I felt no need to include attack generation as an inbuilt software feature.

4.2 INSecS-Intrusion Detection System

Experiments

The following Experiments were designed to validate the detection speed improvement from hierarchical decision making and to compare intrusion detection abilities of INSecS-IDS with snort.

Table 4.1: Capabilities of dataset creators

Capability	INSecS-DCS	ID2T
Ability to Label dataset	yes	yes
Open Source	yes	yes
Raw PCAP dataset	yes	yes
Has a GUI	no	yes
Allows attack injection within the software	no	yes
Ability to divide traffic into time window and get overall traffic attributes	yes	no
Ability to select input method (packets captured on a network of choice or get a raw PCAP dataset from another source)	yes	no
Processed dataset that can be fed into WEKA and other ML tools directly	yes	no
Attribute selection for processed dataset	yes	no

Experiment 1 (Validate effectiveness of hierarchical decision making): test case 1

In order to validate the idea that using a hierarchical decision-making approach improves performance in intrusion detection, the following experiment was designed. I selected one attack (Smurf) and designed two experimental setups with one having hierarchical decision making and the other without it. Both setups had the same task and they were evaluated on accuracy and detection speed. In order to keep the experiment fair, I used PDNs and CDEs in both setups but in the setup where hierarchical decision making was absent, the PDN was just forwarding all traffic without doing any decision making.

To ensure repeatability, 3 dataset files containing Smurf attack from the DARPA 98 dataset were used. Packets from the dataset files were sent as a stream to simulate the stream of packets. Details about the number of packets in each dataset file are shown in the table 4.2.

The experiment was conducted in the same network. One Linux virtual machine contained the PDN for Smurf. This served as the node under attack and received the stream of packets.

Table 4.2: Dataset details for experiment 1:test case 1

Dataset	Number of packets
Week 1, Wednesday	875,472
Week 3, Wednesday	619,767
Week 5, Monday	2,202,859

Table 4.3: Results of hierarchical decision making performance experiment 1: Test case 1

Dataset	Setup 1/ (mins)	Setup 2/ (mins)	Percentage of ICMP traffic/ %	Percentage decrease in detection time/ %
Week 1, Wednesday	53.5	5.090	4.25	90.485
Week 3, Wednesday	34.2	6.837	13.15	80.008
Week 5, Monday	141.25	124.266	87.67	12.024

Note 1: Setup 1 has no hierarchical decision making while setup 2 has hierarchical decision making.
Note 2: Only Smurf attack is detected and test was conducted with 3 datasets containing Smurf attack signatures.

It contained a RAM of 2 Gb, hosted on a computer with a 2.9 GHz CPU with 4 cores and was connected to the network using Ethernet. The machine hosting the CDE was a windows computer with 8 Gb of RAM 2.9 GHz CPU with 4 cores which was also connected to the network using Ethernet.

Results of Experiment 1: test case 1

The results of experiment 1, given in table 4.3, shows that setup 2 (using hierarchical decision making) performs faster Intrusion Detection with all 3 datasets. Results seen in table 4.3 can be explained by considering the variation of the percentage of potential threats (ICMP traffic in the case of Smurf) in each dataset, with the percentage decrease in detection time (See graph given in figure 4.1) for the setup 1 and setup 2.

In the case of setup 1, there is a linear variation between percentage of potential threats and percentage decrease in detection time. When the percentage of potentially anomalous traffic is low, detection time is low and when the percentage of potentially anomalous traffic is higher, detection time is larger. This is because the full processing power of IDS is spent only on traffic

that has a possibility of belonging to an intrusion. This is the advantage of using hierarchical decision making.

In result for the dataset from week 5, Monday, even with hierarchical decision making, most of the traffic needs to be fully processed by both PDN and CDE (percentage of potentially anomalous traffic is 87.67%. Hence, the advantage given by hierarchical decision making depends on the nature of traffic. Imperva [26], which is an industry leader in internet traffic statistics, has reported that 28.9% of all traffic in the internet are generated by bad bots. So it is safe to assume that the percentage of anomalous traffic that a network my experience is somewhere close to this. 87.67% is an unlikely percentage of anomalous traffic for a network. So out of the 3 datasets used, the first two seems to represent a real situation. In both those situations, INSecS-IDS has achieved a large improvement in detection speed (90 and 80 percent decrease in detection time, respectively).

In setup 2, where hierarchical decision making was not used, the full processing capability of IDS is spent on the entire traffic stream. This means that in comparison to setup 1, there is no change in detection time. The contrast between the results for setup 1 and setup 2 can be seen in figure 4.1). The curve for setup 2 shows no percentage decrease in detection time.

The average detection time for the setup 1 and setup 2 are given in Figure 4.2 (the average for the 3 datasets). From the figure it is clear that hierarchical decision making improves the detection time in intrusion detection.

Experiment 1: test case 2

Another test case was designed to validate that using a hierarchical decision-making approach improves performance in intrusion detection even in the presence of multiple attacks simultaneously. So I selected Smurf, IP sweep and Port scan as our attacks. The DARPA 98 datasets were available according to weeks of data collection. Since DARPA did not have a week which contained all three attacks, I combined datasets from Monday and Wednesday of week 3 to form one dataset file. The combined file had 1,327,564 packets in it.

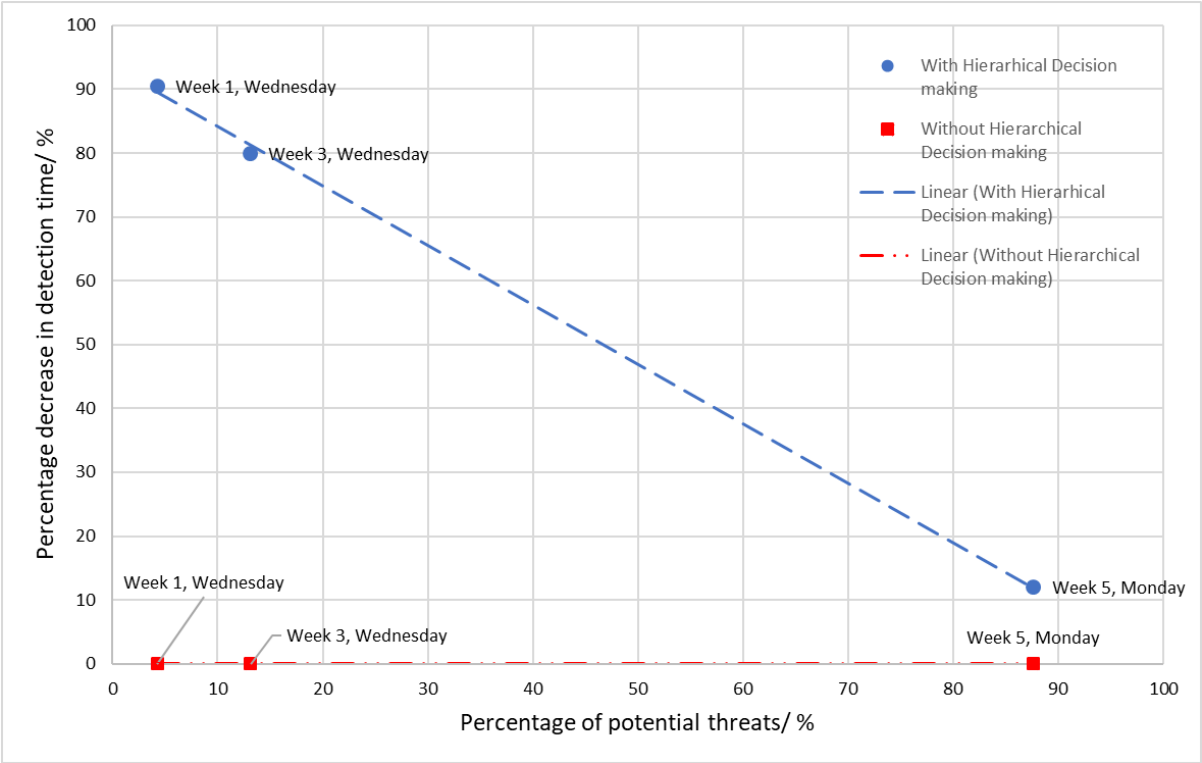

Figure 4.1: Variation of the percentage decrease in detection time with percentage of potential threats in Experiment 1: test case 1

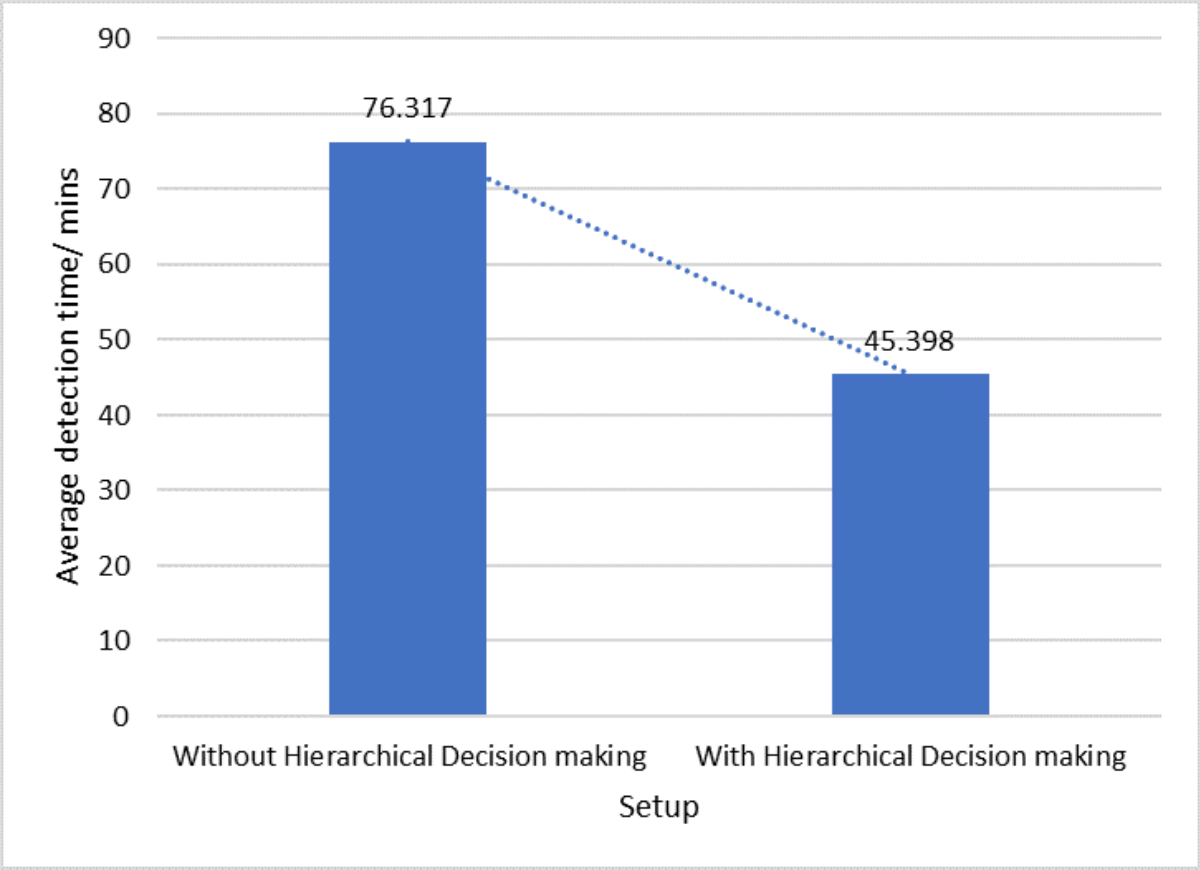

Figure 4.2: Comparison of average detection time without and with hierarchical decision making in Experiment 1: test case 1

Table 4.4: Results of hierarchical decision making performance experiment 1: Test case 2

Attack	Setup 1/ (mins)	Setup 2/ (mins)	Percentage of possible threats/ %	Percentage decrease in detection time/ %
Smurf, IP sweep, Port scan	112.55	14.202	15.49	87.7

Note 1: Setup 1 has no hierarchical decision making while setup 2 has hierarchical decision making.
Note 2: Test was conducted on 1 dataset file containing all 3 attack signatures.

The experiment was conducted in the same network. One Linux virtual server contained the 3 PDNs for Smurf, Portscan and IPsweep. This served as the node under attack and received the stream of packets. It contained a RAM of 2 Gb, hosted on a computer with a 2.9 GHz CPU with 4 cores and was connected to the network using Ethernet. The machine hosting the CDE was a windows computer with 8 Gb of RAM and also connected to the network using Ethernet.

Results of Experiment 1: test case 2

The results of this experiment are given in table 4.4. The ratio between column 1 and column 2 gives ratio between detection time without hierarchical decision making over detection time with hierarchical decision making.

$$\begin{aligned} Ratio\ of\ column\ 1\ over\ column\ 2\ from\ Table\ 4.4 &= \frac{112.55}{14.202} \\ &= 7.92 \end{aligned}$$

This test case shows that setup 2, which uses hierarchical decision making, is nearly 8 times faster in detecting intrusions.

Table 4.5: Results of hierarchical decision making performance experiment: Test case 1 with Wisdom

Dataset	Setup 1/ (mins)	Setup 2/ (mins)	Percentage of ICMP traffic/ %	Percentage decrease in detection time/ %
Week 1, Wednesday	55.2	5.474	4.25	90.08
Week 3, Wednesday	37.9	7.24	13.15	80.89
Week 5, Monday	105.65	133.58	87.67	11.33

Note: Setup 1 has no hierarchical decision making while setup 2 has hierarchical decision making.

Experiment 2 ((Validate effectiveness of hierarchical decision making with a CEP instead of CDE): test case 1

The flexibility of our IDS design allows the users to use a commercial grade Complex Event Processor, which is a software capable of tracking and analyzing streams of data related to events deriving conclusions. This capability makes it useful in a scenario such as the one presented in our IDS. The Complex Decision Engine that is inbuilt, is a case-specific complex event processor designed for this specific IDS. This allows the author to introduce a commercial grade CEP, with queries written with similar logic as our complex decision engine, to be used instead of the CDE. This was tested using Wisdom [36], a Complex Event Processor developed by a researcher in our lab at the University of Western Ontario. I repeated test case 1 from Experiment 1, but replacing the CDE with the Wisdom CEP. The results of this experiment are given in tables 4.5.

Results of Experiment 2: test case 1

By comparing table 4.3 with table 4.5 it is evident that the Complex Event processor performs slightly better than Wisdom when considering the detection time. The percentage decrease in detection time has similar trends in both experiments and shows that the advantage given by hierarchical decision making is consistent. However, it should be noted that the full capabilities of Wisdom are not used and that the advantage the INSecS-CDE has over Wisdom is due the

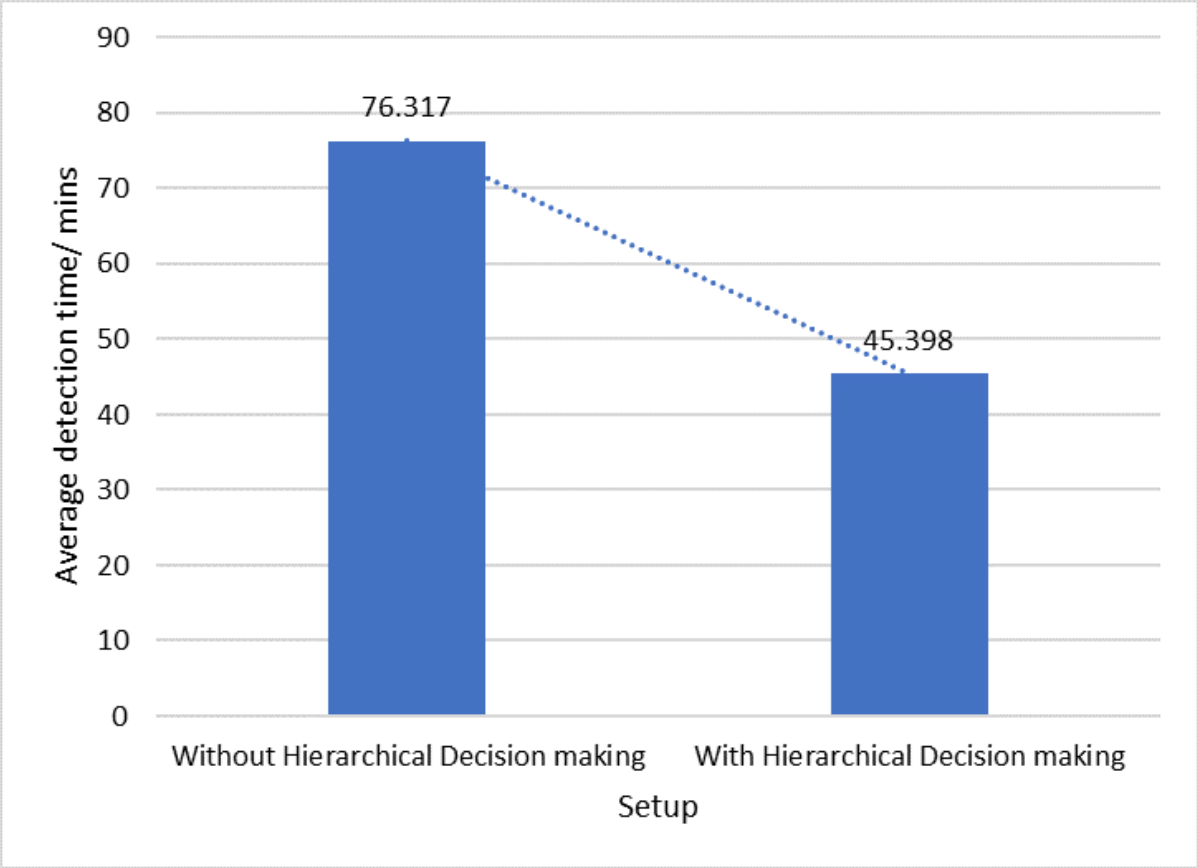

Figure 4.3: Comparison of average detection time with and without hierarchical decision making in Experiment 1: test case 1

fact that the CDE is like a stripped down version of Wisdom with only the necessary modules.

Figure 4.3 shows the difference between average detection times with and without hierarchical decision making for test case 1 in experiment 2.

Experiment 2: test case 2

Test case 2 of Experiment 1 was again repeated as test case 2 of experiment 2, with the CDE replaced by Wisdom.

Results of Experiment 2: test case 2

Results of this test are given in table 4.6. Results show similar trends as in experiment 1, but the detection times are slightly higher in comparison. The reasons for this behavior are again

Table 4.6: Results of hierarchical decision making performance experiment with 3 attacks: Test case 2 with Wisdom

Attack	Setup 1/ (mins)	Setup 2/ (mins)	Percentage of possible threats/ %	Percentage decrease in detection time/ %
Smurf, IP sweep, Port scan	124.617	14.90	15.49	88.04

Note: Setup 1 has no hierarchical decision making while setup 2 has hierarchical decision making.

the slight advantage CDE has over a CEP in this specific application.

The ratio between column 1 and column 2 of table 4.6 gives ratio between detection time without hierarchical decision making over detection time with hierarchical decision making.

$$Ratio\ of\ column\ 1\ over\ column\ 2\ from\ Table\ 4.6 = \frac{124.617}{14.90}$$
$$= 8.36$$

This test case shows that setup 2, which uses hierarchical decision making, is more than 8 times faster in detecting intrusions.

The purpose of this comparison was to show that INSecS-IDS has the capability to integrate well with other software packages, that ensures an easier learning curve for INSecS-IDS users. Since CEPs use a query structure similar to SQL, the above-mentioned capability will be useful in situations where an organization prefers to use a CEP product they are already familiar with, when setting up and using INSecS-IDS. But, the use of INSecS-Complex Decision Engine is recommended if better performance is valued over the easier learning curve.

4.2.1 Comparison with Snort

Snort is a popular, open source, real-time IDS which is widely used by researchers for its strengths compared to other intrusion detection systems [4],[16]. Snort is a rule-based IDS

(similar to INSecS-IDS) and has a large rule base which is updated periodically. It also allows community rules, which is custom rules written by users. Using this functionality, I compared the accuracy of Snort with compared to INSecS-IDS by writing Snort rules that were logically similar to the rules used in our IDS. Tests showed that INSecs-IDS has an accuracy identical to the accuracy of Snort, demonstrating that INSecS-IDS is comparable to a leading IDS in the industry.

The experiment was conducted on a windows computer with 8 Gb of RAM and a 2.9 GHz CPU with 4 cores which was also connected to the network using Ethernet.

During the course of the study, the following advantages of INSecS-IDS over a popular and widely used IDS like Snort were identified.

- Better control over rule deployment - One drawback of Snort is that it has a large number of rules in multiple repositories [16] with no central control. It's also possible to write custom rules or change the existing ones without having prior knowledge about whether that rule is already in place. This introduces a problem in system performance because there might be multiple rules which perform the same condition checks.

  In INSecS-IDS, there is only one rule base and the rule implementation is much more flexible although it requires more work since it has no application-specific script like Snort does. Conversely, INSecS-IDS is completely implemented in Python and only requires a basic knowledge of Python, to make changes to rules or implement new ones.

- Efficiency in intrusion detection - Evaluating the entire rule set on each packet is an unavoidable situation with the Snort design. As shown in section 3.2.4, this causes a significant difference in the detection time. By having a hierarchical decision-making structure, this challenge faced by many IDS including Snort, has been overcome.

- Ease of usage and flexibility - While conducting the test cases, I also noted that counting instances within user defined time windows is much more difficult to implement in Snort due to the preset options for threshold and filtering. Such preset options have an adverse

effect on the flexibility of Snort, which has to be avoided in order to face the challenge posed by varying attacks.

- Designed for distributed intrusion detection - Another characteristic that was identified when using Snort was the difficulty of using it in a distributed setting. It does not support distributed real-time usage in its design and requires instances of Snort to run on all nodes in the network if Snort is used in a distributed setup [76]. INSecS-IDS addresses these issues by having a more flexible design and by keeping track of the various intrusion detectors available for deployment.

In all the experiments presented above, the IDS was able to detect all attack instances listed in the dataset documentation.

Chapter 5

Conclusions and Future work

5.1 INSecS-Dataset Creation Software

This thesis introduced a software framework (INSecS-DCS) that is capable of creating a labeled network intrusion dataset, with the option to choose between a raw and processed dataset as the output. A real-time packet capture stream, as well as imported PCAP files, are accepted as inputs. INSecS-DCS is highly customizable and depending on the needs of the researcher or IDS user, they are able to recreate a dataset with the attributes they want, instead of the attributes selected by another researcher.

5.2 INSecS-Intrusion Detection System

INSecS-IDS has the capability to perform well as a modern Intrusion Detection System and as demonstrated by the test cases involving different setups and scenarios, it is quite effective. INSecS-IDS works well for any modern network setup including a Cloud environment or an IoT setup and shows it's true potential in a distributed setup, using it's adaptive learning capabilities, hierarchical decision making, detecting complex attack patterns (multi-step attacks) etc. The following subsections give examples of how INSecS-IDS can be used with IoT and Cloud setups.

5.2.1 IoT system

Let us consider an IoT system consisting of mobile devices that connect to a central control system. An example would be modern smart cars, which connect to central servers in order to share information, receive guidance, update travel details, carry out passenger communication, etc. A few more fitting examples would be a system of sensors and cameras working together to give guidance to a vehicle system [35] or a security system providing security to a vulnerable facility. The IoT devices in the above examples, cannot manage their own intrusion detection functions due to constraints such as,

1. Low system/computing resources (low processing power and low memory) - The IoT devices such as sensors, cameras and even the on-board computer of a smart car, have limited computing capabilities because they are designed for a certain task and the software and hardware they carry are mostly adequate for those specified tasks only.

2. Limited rule updating capabilities - Updating an IDS rule set in each IoT device would be practically impossible due to the communication and effort overhead.

Given the above issues, IoT devices have to rely on a data processing system to operate Intrusion Detection tasks along with the dedicated functions/services expected of the IoT system. A data processing system has dedicated hardware for complex processes and enough computing power to perform a thorough intrusion detection. INSecS-IDS would fit well in this scenario with PDNs placed at each IoT device or gateway devices, sending suspicious traffic to a data processing server for more processing and CDEs can work well inside the data processing servers to complete the detection tasks. No additional communication means are necessary because INSecS-IDS uses the same communication channels used in the standard operation of the IoT system, with the added benefit of using the same protocols and security features used by the IoT system to protect its communication channels.

5.2.2 Cloud computing setup

Because of the distributed nature and flexible design, INSecS-IDS can be easily deployed in a Cloud computing environment. Consider a Cloud computing setup where multiple services and resources are hosted on different Cloud platforms. A few Cloud servers host a website and a database is implemented as a Cloud service. By gaining access to at least one component, an intruder can gain access to the other components because the Cloud companies mostly provide intrusion prevention regarding traffic from unverified users. In this setup, the Cloud servers (infrastructure as a service) are possible weak links, because they allow users to interact with the OS. Since there are multiple servers with OS access, a distributed intrusion detection such as INSecS-IDS would be the most suitable choice. Primary Decision Nodes can be deployed at each server and Complex Decision Engine can be hosted in a local machine.

5.2.3 Future work

In the future, I plan to utilize the dataset creation capability to provide a Real Time Network Intrusion Detection System (Real Time NIDS), with real-time datasets. Customization options of the dataset will give the NIDS user, the ability to customize the performance of the NIDS to fit the local network. This would have a huge impact on commercial and private network security industry.

I have identified a few areas that can be improved. The communication between the distributed components of INSecS-IDS is done using HTTP, making it vulnerable to an attack from within the network. Encrypted and secure communication channels can be used to mitigate this threat. It also lacks a Graphical User Interface (GUI), which would add value to its usability. An improvement which would make this even better is integrating host-based IDS capabilities into INSecS and making it the perfect hybrid IDS that would work on any setup against any attack.

The next step is to integrate INSecS-DCS and INSecS-IDS using the machine learning

module. The expected functionality for learning module is to be able to extract rules from custom datasets and feed them to the IDS rule set. An unsupervised learning algorithm would be an ideal fit given the need for automation of the entire process. Even if that does not yield the expected results, INSecS-DCS has the ability to label the datasets it creates and we can expect to have good supervised learning algorithm to complete the task.

Bibliography

[1] OSSEC - open-source host-based intrusion detection system.

[2] tshark - The Wireshark Network Analyzer 2.6.2.

[3] Preeti Aggarwal and Sudhir Kumar Sharma. Analysis of KDD Dataset Attributes - Class wise for Intrusion Detection. *Procedia Computer Science*, 57:842–851, 2015.

[4] Ghilman Ahmed, Mehdi Hussain, and M N A Khan. Characterizing Strengths of Snort-based IDPS. *Research Journal of Recent Sciences*, 3(4):88–94, 2014.

[5] Monther Aldwairi, Ansam M. Abu-Dalo, and Moath Jarrah. Pattern matching of signature-based IDS using Myers algorithm under MapReduce framework. *EURASIP Journal on Information Security*, 2017(1):9, Dec 2017.

[6] Hifaa Bait Baraka and Huaglory Tianfield. Intrusion Detection System for Cloud Environment. In *Proceedings of the 7th International Conference on Security of Information and Networks - SIN '14*, pages 399–404, New York, New York, USA, 2014. ACM Press.

[7] Srijita Basu, Arjun Bardhan, Koyal Gupta, Payel Saha, Mahasweta Pal, Manjima Bose, Kaushik Basu, Saunak Chaudhury, and Pritika Sarkar. Cloud computing security challenges & solutions-A survey. In *2018 IEEE 8th Annu. Comput. Commun. Work. Conf.*, pages 347–356. IEEE, Jan 2018.

[8] Elisa Bertino and Nayeem Islam. Botnets and Internet of Things Security. *Computer (Long. Beach. Calif).*, 50(2):76–79, Feb 2017.

[9] S. Bhattacharya and S. Selvakumar. Ssenet-2014 dataset: A dataset for detection of multiconnection attacks. In *2014 3rd International Conference on Eco-friendly Computing and Communication Systems*, pages 121–126, Dec 2014.

[10] Van Loi Cao, Van Thuy Hoang, and Quang Uy Nguyen. A scheme for building a dataset for intrusion detection systems. *2013 3rd World Congress on Information and Communication Technologies, WICT 2013*, pages 280–284, 2014.

[11] Long Cheng, Fang Liu, and Danfeng (Daphne) Yao. Enterprise data breach: causes, challenges, prevention, and future directions. *Wiley Interdisciplinary Reviews: Data Mining and Knowledge Discovery*, 7(5):e1211.

[12] Carlos Garcia Cordero, Emmanouil Vasilomanolakis, Nikolay Milanov, Christian Koch, David Hausheer, and Max Muhlhauser. ID2T: A DIY dataset creation toolkit for Intrusion Detection Systems. *2015 IEEE Conference on Communications and NetworkSecurity, CNS 2015*, (December):739–740, 2015.

[13] S. E. Coull, F. Monrose, M. K. Reiter, and M. Bailey. The challenges of effectively anonymizing network data. In *2009 Cybersecurity Applications Technology Conference for Homeland Security*, pages 230–236, March 2009.

[14] Carlo Curino, Evan P C Jones, Raluca Ada Popa, Nirmesh Malviya, Eugene Wu, Sam Madden, Hari Balakrishnan, and Nickolai Zeldovich. Relational Cloud: A Database-as-a-Service for the Cloud. In *Bienn. Conf. Innov. Data Syst. Res.*, 2011.

[15] Deepak Deepak Kshirsagar, Suraj Suraj Sawant, Ravi Ravi Wadje, and Pravin Pravin Gayal. Distributed Intrusion Detection System for TCP Flood Attack. In *Proceeding of International Conference on Intelligent Communication, Control and Devices*, pages 951–958. Springer, Singapore, 2017.

[16] Anna Drozdova. Securing the DNS server with Snort IDS: Severen-Telecom case. 2015.

[17] J Ernst, T Hamed, and S Kremer. A Survey and Comparison of Performance Evaluation in Intrusion Detection Systems. In *Computer and Network Security Essentials*, pages 555–568. Springer International Publishing, Cham, 2018.

[18] Y. Fu, F. Lou, F. Meng, Z. Tian, H. Zhang, and F. Jiang. An intelligent network attack detection method based on rnn. In *2018 IEEE Third International Conference on Data Science in Cyberspace (DSC)*, pages 483–489, June 2018.

[19] Vinod Ganapathy. Reflections on the Self-service Cloud Computing Project. In *Int. Conf. Inf. Syst. Secur.*, pages 36–57. Springer, Cham, Dec 2015.

[20] Amirhossein Gharib, Iman Sharafaldin, Arash Habibi Lashkari, and Ali A. Ghorbani. An Evaluation Framework for Intrusion Detection Dataset. *ICISS 2016 - 2016 International Conference on Information Science and Security*, (Cic):0–4, 2017.

[21] M. Ghorbanian, B. Shanmugam, G. Narayansamy, and N. B. Idris. Signature-based hybrid intrusion detection system (hids) for android devices. In *2013 IEEE Business Engineering and Industrial Applications Colloquium (BEIAC)*, pages 827–831, April 2013.

[22] Jairo Giraldo, Esha Sarkar, Alvaro A. Cardenas, Michail Maniatakos, and Murat Kantarcioglu. Security and Privacy in Cyber-Physical Systems: A Survey of Surveys. *IEEE Design & Test*, 34(4):7–17, Aug 2017.

[23] Jyun-Yao Huang and I-En Liao. A searchable encryption scheme for outsourcing cloud storage. In *2012 IEEE Int. Conf. Commun. Networks Satell.*, pages 142–146. IEEE, Jul 2012.

[24] Neminath Hubballi and Vinoth Suryanarayanan. False alarm minimization techniques in signature-based intrusion detection systems: A survey. *Computer Communications*, 49:1–17, Aug 2014.

[25] Karl Smith Ian Herwono, Zhan Cui, Ben AZVINE, Martin Brown. Method and Apparatus for detecting a multi stage event, 2018.

[26] Inc(US) Imperva. Bot Traffic Report 2016 — Incapsula, 2018.

[27] B. Ingre and A. Yadav. Performance analysis of nsl-kdd dataset using ann. In *2015 International Conference on Signal Processing and Communication Engineering Systems*, pages 92–96, Jan 2015.

[28] Chen Jun and Chen Chi. Design of Complex Event-Processing IDS in Internet of Things. In *2014 Sixth International Conference on Measuring Technology and Mechatronics Automation*, pages 226–229. IEEE, Jan 2014.

[29] Chintan C Kacha, Kirtee A Shevade, Kuldeep S Raghuwanshi, and Asst Professor. Improved Snort Intrusion Detection System Using Modified Pattern Matching Technique. *International Journal of Emerging Technology and Advanced Engineering Website: www.ijetae.com ISO Certified Journal*, 3(7), 2013.

[30] Gulshan Kumar. Evaluation Metrics for Intrusion Detection Systems -A Study. *Int. J. Comput. Sci. Mob. Appl.*, 2(11):11–17, 2014.

[31] James F Kurose and Keith W Ross. *COMPUTER NETWORKING*. Pearson, New Jersey 07458,, sixth edition, 2013.

[32] Tsung-Han Lee, Chih-Hao Wen, Lin-Huang Chang, Hung-Shiou Chiang, and Ming-Chun Hsieh. A Lightweight Intrusion Detection Scheme Based on Energy Consumption Analysis in 6LowPAN. pages 1205–1213. Springer, Dordrecht, 2014.

[33] Shancang Li, Li D. Xu, and Imed Romdhani. *Securing the internet of things*. Elsevier, 2017.

[34] Hung-Jen Liao, Chun-Hung Richard Lin, Ying-Chih Lin, and Kuang-Yuan Tung. Intrusion detection system: A comprehensive review. *Journal of Network and Computer Applications*, 36:16–24, 2012.

[35] Jiajia Liu, Shubin Zhang, Wen Sun, and Yongpeng Shi. In-Vehicle Network Attacks and Countermeasures: Challenges and Future Directions. *IEEE Network*, 31(5):50–58, 2017.

[36] Gobinath Loganathan, Jagath Samarabandu, and Xianbin Wang. Real-time intrusion detection in network traffic using adaptive and auto-scaling stream processor. In *2018 IEEE Global Communications Conference: Communication & Information System Security (Globecom2018 CISS)*, Abu Dhabi, United Arab Emirates, December 2018.

[37] Flavio Lombardi and Roberto Di Pietro. Secure virtualization for cloud computing. *J. Netw. Comput. Appl.*, 34(4):1113–1122, Jul 2011.

[38] Hafza A. Mahmood and Hafza A. Mahmood. Network Intrusion Detection System (NIDS) in Cloud Environment based on Hidden Naïve Bayes Multiclass Classifier. *Al-Mustansiriyah Journal of Science*, 28(2):134, Apr 2018.

[39] Marek MaÅowidzki, Przemyslaw Berezinski, and MichaÅ Mazur. Network intrusion detection: Half a kingdom for a good dataset, April 2015.

[40] G. Meena and R. R. Choudhary. A review paper on ids classification using kdd 99 and nsl kdd dataset in weka. In *2017 International Conference on Computer, Communications and Electronics (Comptelix)*, pages 553–558, July 2017.

[41] Daniele Midi, Antonino Rullo, Anand Mudgerikar, and Elisa Bertino. Kalis A System for Knowledge-Driven Adaptable Intrusion Detection for the Internet of Things. In *2017 IEEE 37th International Conference on Distributed Computing Systems (ICDCS)*, pages 656–666. IEEE, Jun 2017.

[42] Nishit Mishra, Tarun Kumar Sharma, Varun Sharma, and Vrince Vimal. *Secure Framework for Data Security in Cloud Computing*. Springer, Singapore, 2018.

[43] Preeti Mishra, Emmanuel S. Pilli, Vijay Varadharajan, and Udaya Tupakula. Intrusion detection techniques in cloud environment: A survey. *Journal of Network and Computer Applications*, 77:18–47, Jan 2017.

[44] Nour Moustafa and Jill Slay. UNSW-NB15: a comprehensive data set for network intrusion detection systems (UNSW-NB15 network data set). *2015 Military Communications and Information Systems Conference (MilCIS)*, pages 1–6, 2015.

[45] Amin Nezarat. Distributed Intrusion Detection System Based on Mixed Cooperative and Non-Cooperative Game Theoretical Model. *International Journal of Network Security*, 20(1):56–64, 2018.

[46] Peng Ning and North Carolina. *Intrusion Detection Techniques*. Wiley, 3 edition, 2012.

[47] Job Noorman, Jo Van Bulck, Jan Tobias Mühlberg, Frank Piessens, Ku Leuven PIETER MAENE, Bart Preneel, Ingrid Verbauwhede, Ku Leuven JOHANNES GÖTZFRIED, Tilo Müller, Felix Freiling, Pieter Maene, and Johannes Götzfried. 7 Sancus 2.0: A Low-Cost Security Architecture for IoT Devices ACM Reference Format. *ACM Transactions on Privacy and Security ACM Trans. Priv. Secur*, 20(7), 2017.

[48] Doohwan Oh, Deokho Kim, and Won Ro. A Malicious Pattern Detection Engine for Embedded Security Systems in the Internet of Things. *Sensors*, 14(12):24188–24211, Dec 2014.

[49] Salima Omar, Asri Ngadi, and Hamid H Jebur. Machine Learning Techniques for Anomaly Detection: An Overview. *International Journal of Computer Applications*, 79(2):975–8887, 2013.

[50] Vern Paxson. Bro: A System for Detecting Network Intruders in Real-Time. *Computer Networks: The International Journal of Computer and Telecommunications Networking*, 31(23-24):2435–2463, 1999.

[51] Deba Prasead Mozumder, MdJulkar Nayeen Mahi, Md Whaiduzzaman, and Md Julkar Nayeen Mahi. Cloud Computing Security Breaches and Threats Analysis. *Int. J. Sci. Eng. Res.*, 8(1), 2017.

[52] B. R. Raghunath and S. N. Mahadeo. Network intrusion detection system (nids). In *2008 First International Conference on Emerging Trends in Engineering and Technology*, pages 1272–1277, July 2008.

[53] Nadun N Rajasinghe, Jagath Samarabandu, and Xianbin Wang. INSecS-DCS: a highly customizable network intrusion dataset creation framework. In *2018 IEEE Canadian Conference on Electrical & Computer Engineering (CCECE) (CCECE 2018)*, Quebec City, Canada, May 2018.

[54] Rohini Rajpal, Sanmeet Kaur, and Ramandeep Kaur. Improving detection rate using misuse detection and machine learning. In *2016 SAI Computing Conference (SAI)*, pages 1131–1135, London, UK., 2016.

[55] Setareh Roshan, Yoan Miche, Anton Akusok, and Amaury Lendasse. Adaptive and online network intrusion detection system using clustering and Extreme Learning Machines. *Journal of the Franklin Institute*, 355(4):1752–1779, Mar 2018.

[56] Antonino Rullo, Daniele Midi, Edoardo Serra, Elisa Bertino, D Midi, and E Bertino. Pareto Optimal Security Resource Allocation for Internet of Things. *ACM Trans. Priv. Secur. Article*, 20(15), 2017.

[57] Maheshkumar Sabhnani and Gursel Serpen. Why Machine Learning Algorithms Fail in Misuse Detection on KDD Intrusion Detection Data Set. *Why machine learning algorithms fail in misuse detection on KDD intrusion detection data set*, 8(4):403–415, 2004.

[58] Leonel Santos, Carlos Rabadao, and Ramiro Goncalves. Intrusion detection systems in Internet of Things: A literature review. In *2018 13th Iberian Conference on Information Systems and Technologies (CISTI)*, pages 1–7. IEEE, Jun 2018.

[59] Muhammad Saqib Niaz and Gunter Saake. Merkle Hash Tree based Techniques for Data Integrity of Outsourced Data General Terms. In *27th GI-Workshop Found. Databases*, Magdeburg, 2015.

[60] Bezborodov Sergey. Intrusion Detection System and Intrusion Prevention System with Snort provided by Security Onion. *MAMK school of applied Sciences, Bachelor's thesis*, 2016.

[61] Sparsh Sharma and Ajay Kaul. A survey on Intrusion Detection Systems and Honeypot based proactive security mechanisms in VANETs and VANET Cloud. *Vehicular Communications*, 12:138–164, Apr 2018.

[62] Ali Shiravi, Hadi Shiravi, Mahbod Tavallaee, and Ali A. Ghorbani. Toward developing a systematic approach to generate benchmark datasets for intrusion detection. *Computers and Security*, 31(3):357–374, 2012.

[63] Jungsuk Song, Hiroki Takakura, Yasuo Okabe, Masashi Eto, Daisuke Inoue, and Koji Nakao. Statistical analysis of honeypot data and building of Kyoto 2006+ dataset for NIDS evaluation. *Proceedings of the First Workshop on Building Analysis Datasets and Gathering Experience Returns for Security - BADGERS '11*, pages 29–36, 2011.

[64] T Tang, SAR Zaidi, D McLernon, L Mhamdi, and M Ghogho. Deep recurrent neural network for intrusion detection in sdn-based networks, March 2018. This article is protected by copyright. Personal use of this material is permitted. Permission from IEEE must be obtained for all other uses, in any current or future media, including reprinting/republishing this material for advertising or promotional purposes, creating new collective works,

for resale or redistribution to servers or lists, or reuse of any copyrighted component of this work in other works.

[65] Peiying Tao, Zhe Sun, and Zhixin Sun. An Improved Intrusion Detection Algorithm Based on GA and SVM. *IEEE Access*, 6:13624–13631, 2018.

[66] Maha Tebaa and Said El Hajii. Secure Cloud Computing through Homomorphic Encryption. *Int. J. Adv. Comput. Technol.*, 5(16), 2013.

[67] Muhammad Fahad Umer, Muhammad Sher, and Yaxin Bi. A two-stage flow-based intrusion detection model for next-generation networks. *PLOS ONE*, 13(1):e0180945, Jan 2018.

[68] Muhammad Fahad Umer, Muhammad Sher, and Imran Khan. Towards Multi-Stage Intrusion Detection using IP Flow Records. *IJACSA) Int. J. Adv. Comput. Sci. Appl.*, 7(10), 2016.

[69] Emmanouil Vasilomanolakis, Carlos Garcia Cordero, Nikolay Milanov, and Max Mühlhäuser. Towards the creation of synthetic, yet realistic, intrusion detection datasets. *Proceedings of the NOMS 2016 - 2016 IEEE/IFIP Network Operations and Management Symposium*, pages 1209–1214, 2016.

[70] Manish Verma. *A Multi-Stage Intrusion Detection Approach for Network Security*. PhD thesis, National Institute of Technology Rourkela, 2015.

[71] Omar Abdel Wahab, Jamal Bentahar, Hadi Otrok, and Azzam Mourad. I Know You Are Watching Me: Stackelberg-Based Adaptive Intrusion Detection Strategy for Insider Attacks in the Cloud. In *2017 IEEE International Conference on Web Services (ICWS)*, pages 728–735. IEEE, Jun 2017.

[72] Cong Wang, Qian Wang, Kui Ren, and Wenjing Lou. Privacy-Preserving Public Auditing for Data Storage Security in Cloud Computing. In *2010 Proc. IEEE INFOCOM*, pages 1–9. IEEE, Mar 2010.

[73] Zhijian Wang and Yanqin Zhu. A centralized HIDS framework for private cloud. In *2017 18th IEEE/ACIS International Conference on Software Engineering, Artificial Intelligence, Networking and Parallel/Distributed Computing (SNPD)*, pages 115–120. IEEE, Jun 2017.

[74] Maochao Xu, Kristin M. Schweitzer, Raymond M. Bateman, and Shouhuai Xu. Modeling and Predicting Cyber Hacking Breaches. *IEEE Transactions on Information Forensics and Security*, 13(11):2856–2871, Nov 2018.

[75] Shucheng Yu, Cong Wang, Kui Ren, and Wenjing Lou. Achieving Secure, Scalable, and Fine-grained Data Access Control in Cloud Computing. In *2010 Proc. IEEE INFOCOM*, pages 1–9. IEEE, Mar 2010.

[76] Wu Yuan, Jeff Tan, and Dung Le. Distributed Snort Network Intrusion Detection System with Load Balancing Approach. In *Proceedings of the International Conference on Security and Management (SAM)*, Athens, 2013.

[77] Bruno Bogaz Zarpelão, Rodrigo Sanches Miani, Cláudio Toshio Kawakani, and Sean Carlisto de Alvarenga. A survey of intrusion detection in Internet of Things. *Journal of Network and Computer Applications*, 84:25–37, Apr 2017.

[78] Fengzhe Zhang, Yijian Huang, Huihong Wang, Haibo Chen, and Binyu Zang. PALM: Security Preserving VM Live Migration for Systems with VMM-enforced Protection. In *2008 Third Asia-Pacific Trust. Infrastruct. Technol. Conf.*, pages 9–18. IEEE, Oct 2008.

[79] Richard Zuech, Taghi M. Khoshgoftaar, Naeem Seliya, Maryam M. Najafabadi, and Clifford Kemp. A New Intrusion Detection Benchmarking System. *Proceedings of the*

Twenty-Eighth International Florida Artificial Intelligence Research Society Conference, (McHugh):252–255, 2015.

Appendix A

Primary Decision Nodes selected for Initial Deployment

INSecS-Intrusion Detection System can be deployed in a distributed environment. As explained in the Chapters on functionality, Primary Decision Nodes (PDNs) have to be deployed in each network node. These would communicate with a Complex Decision Engine, placed on a dedicated server, in order to complete the Intrusion Detection task. For the initial deployment of the IDS the author has selected the following Primary Decision Nodes,

1. Port scan
2. Smurf
3. IP sweep.

There is flexibility to add more Primary Decision Nodes as new attacks are identified.

Appendix B

Operating instructions of INSecS-DCS

File available on Github for local dataset creation are listed below.

- dataset creator with foreign PCAP
- counts.py
- countsVariables.py
- cvar.py
- detectors.py
- main.py
- services.py
- set.py
- requirements.txt
- LICENSE
- README.md

B.1 Requirements

Wireshark/Tshark is needed to run this software. Installation method would vary depending on the OS. This software is written in python3 so python3 needs to be installed before running the code.

```
sudo apt-get update
sudo apt-get install python3.5.2
```

The requirements.txt file includes the python packages needed for the software.

```
pip3 install -r requirements.txt
```

B.2 Making datasets from local traffic

Users can start creating datasets from local traffic by running the following code on a shell.

```
python3 main.py
```

B.2.1 Setting a timer to capture files

When creating the dataset from local traffic, a user can define the duration for packet collection by setting the variable *howlong* in milliseconds. The changes are to be done on the file **set.py**.

B.2.2 Setting time window for processed attributes (attributes that correspond to overall behavior within a time window)

A user needs to define a time window in creating attributes that correspond to the overall behavior of traffic within a time window. This can be done by setting variable *time window* in the file **set.py**.

B.3 Making datasets from Captured PCAP

The same set of python files with changes inside them are found in the folder named **dataset creator with foreign PCAP**. To use this code, the user needs to specify the path to the PCAP file in **main.py** by changing the following line of code.

```
cmd = sudo tshark -r /path/filename -V -T json
```

Setting a time window is exactly as described in section B.2.

Appendix C

Intrusion Detection Logs for INSecS-IDS

The following logs are from the test case described in subsection 3.2.5 where the related results are given in Table 4.3. All the attack instances were identified with and without hierarchical decision making, with the only difference being the difference in detection time.

C.1 Attack Detection Log

smurf attack detected. icmp count: 1090 172.16.114.50

smurf attack detected. icmp count: 1084 172.16.114.50

smurf attack detected. icmp count: 1033 172.16.114.50

smurf attack detected. icmp count: 1084 172.16.114.50

smurf attack detected. icmp count: 1076 172.16.114.50

smurf attack detected. icmp count: 1082 172.16.114.50

smurf attack detected. icmp count: 1081 172.16.114.50

smurf attack detected. icmp count: 1029 172.16.114.50

smurf attack detected. icmp count: 1080 172.16.114.50

smurf attack detected. icmp count: 1072 172.16.114.50

smurf attack detected. icmp count: 1085 172.16.114.50

smurf attack detected. icmp count: 1076 172.16.114.50

smurf attack detected. icmp count: 1010 172.16.114.50

smurf attack detected. icmp count: 1045 172.16.114.50

smurf attack detected. icmp count: 1046 172.16.114.50

smurf attack detected. icmp count: 1044 172.16.114.50

smurf attack detected. icmp count: 1062 172.16.114.50

smurf attack detected. icmp count: 1003 172.16.114.50

smurf attack detected. icmp count: 1039 172.16.114.50

smurf attack detected. icmp count: 1048 172.16.114.50

smurf attack detected. icmp count: 1061 172.16.114.50

smurf attack detected. icmp count: 1060 172.16.114.50

smurf attack detected. icmp count: 912 172.16.114.50

smurf attack detected. icmp count: 1047 172.16.114.50

smurf attack detected. icmp count: 1066 172.16.114.50

smurf attack detected. icmp count: 1070 172.16.114.50

smurf attack detected. icmp count: 1069 172.16.114.50

smurf attack detected. icmp count: 994 172.16.114.50

smurf attack detected. icmp count: 1086 172.16.114.50

smurf attack detected. icmp count: 1105 172.16.114.50

smurf attack detected. icmp count: 1098 172.16.114.50

smurf attack detected. icmp count: 1095 172.16.114.50

smurf attack detected. icmp count: 1034 172.16.114.50

smurf attack detected. icmp count: 1091 172.16.114.50

smurf attack detected. icmp count: 986 172.16.114.50

Curriculum Vitae

Name:	Nadun Rajasinghe
Post-Secondary Education and Degrees:	The University of Western Ontario London, ON 2016 - 2018 MESc. University of Peradeniya Sri Lanka 2011 - 2015 BSc.Eng
Honours and Awards:	Award for Outstanding Presentation in Graduate Symposium - Software Engineering Annual Electrical & Computer Engineering Graduate Research Symposium, May 2017
Related Work Experience:	Teaching Assistant The University of Western Ontario 2016 - 2018

Publications:

INSecS-DCS: A Highly Customizable Network Intrusion Dataset Creation Framework, IEEE Canadian Conference on Electrical and Computer Engineering, Quebec City, Canada, May 2018.