Proceedings of the 28th SMART-iSTEAMS
Interteriary Multidisciplinary Conference
American International University West Africa
The Gambia

Full Research Paper

A Deception Based Intelligent Intrusion Detection System for Detecting Threats of Exploits in Cloud Based Environments

[1] Oluoha Onyekware U.
[2] Okereke George E.
[3] Udanor Collins N.
[4] Bakpo Francis S.

Dept of Computer Science
University of Nigeria,
Nsukka, Nigeria.

E-mails
[1]d_blackdiamond@yahoomail.com
[2]george.okereke@unn.edu.ng
[3]Collins.udanor@unn.edu.ng
[4]francis.bakpo@unn.edu.ng

Phones
[1]+2348181320874
[2]+2348037784613
[3]+2348061338765
[4]+2348034401289

ABSTRACT

Despite its numerous advantages, cloud computing faces major security threats with constantly evolving digital prints and attack-like patterns. Unfortunately, due to the share size and complexity of cloud computing, traditional approaches to Intrusion Detection Systems (IDS) have been shown to be rather defective in adapting to, identifying and mitigating threat in cloud based environment. While, anomaly-based IDS are plagued with misidentifying legitimate network activities or sometimes permitting sophisticated malicious traffic patterns, signature-based IDS on the other hand are less adaptive and practically ineffective against sophisticated attacks and advanced persistent threat (APT). This paper presents a unique design approach for deception-based intelligent Intrusion Detection Systems, which are better suited for operations in cloud based environments. Modelling and simulation was conducted using Application Characterization Engine and Flow Modelling Engine within OPNET modular to create runtimes of known attack types in a deception based environment. The machine learning scripts, attack codes and embedded socket and API integration scripts are presented in Python. The security framework was modelled with machine learning to further enhance its adaptability and predictive capabilities.

Keywords: Cybersecurity, Intrusion Detection System, Deception techniques, Machine Learning.

Proceedings Reference Format
Oluoha, U.O., Okereke, G.E., Udanor, N.C. & Bakpo, F.S. (2021): A Deception Based Intelligent Intrusion Detection System for Detecting Threats of Exploits in Cloud Based Environments. Proceedings of the 28th iSTEAMS Intertertiary Multidisciplinary Conference. American International University West Africa, The Gambia. October, 2021. Pp 7-36 www.isteams.net/gambia2021.
DOI - https://doi.org/ 10.22624/AIMS/iSTEAMS-2021/V28P2x

1. INTRODUCTION

There is no gainsaying the fact that the advent of the Internet and World Wide Web infrastructure has revolutionized the entire Information Technology landscape, bringing about the metamorphosis of other disruptive technological advancements such as grid computing, virtualization, cloud computing, sensor networks and the Internet of Things (IoT). The ascendency of such innovative technologies have brought in its wings major gains to the IT world and the general public.

Virtualization and cloud computing are arguably one of the most widely used of these innovation, and its use is projected to increase astronomically in the future [1]. Despite the advantages which cloud computing presents, it is also beset by major security issues, which seems to be on the increase globally as shown in the 2019 WhiteHat Security Statistics Report [2] and the studies undertaken by Symantec in 2017 [3,4]. Such incidences and security breaches come with attendant losses running into billions of dollars annually.

One of the front-line defence of choice in the fight against cyber-attacks and exploits in the cloud is the Intrusion Detection System (IDS). Simply put, an IDS could be described as a contraption/tool (presented as a software or hardware), which is used by cyber security administrators in monitoring a system or network to guard against security breaches, suspicious activities or set policy violations [5]. When such breaches or violations occur, the IDS is expected to detect and flag such, while appropriate counter-action(s) are taken by the security administrator. IDS can operate at network-level and host-level [6,7].

Detection of attacks in IDS may be broadly classified as signature-based detection and anomaly detection [7] [8]. It can also be a hybrid system, possessing the characteristics of both signature-based and anomaly-based IDS [9]. Signature-based detection generates alarms with significant details, like type of attacks, attack sources, attack ports, and list of victims. In this method, a database of attack signatures is created and updated regularly by security vendors [10]. Anomaly-based detection is a different approach to signature-based detection. The normal traffic patterns are considered in a baseline and considerable deviations from the normal patterns are considered as anomaly, which are raised as alarms [11,12,13,14]. Definitive decision-making on attacks is not automated in anomaly-based intrusion detection, as well. Figure 1 gives an abstraction of a traditional IDS System.

Traditional IDSs have over the years been shown to have grave problems and security lapses, which could be exploited to undermine the integrity of the entire system [15,11,7]. These issues are indeed worse in cloud environment. For instance, anomaly-based IDS while showing great prospects in identifying new and evolving threats, are notorious in misidentifying legitimate traffic patterns as malicious, while possibly allowing malicious traffic as legitimate traffic. In a similar vein, while signature-based IDS are very effective in stopping all attacks documented in its signature database, they are grossly ineffective in identifying new evolving attacks and day-0 attacks. Also, building and maintaining a meaningful, dynamic and relevant signature database remains a major challenge.

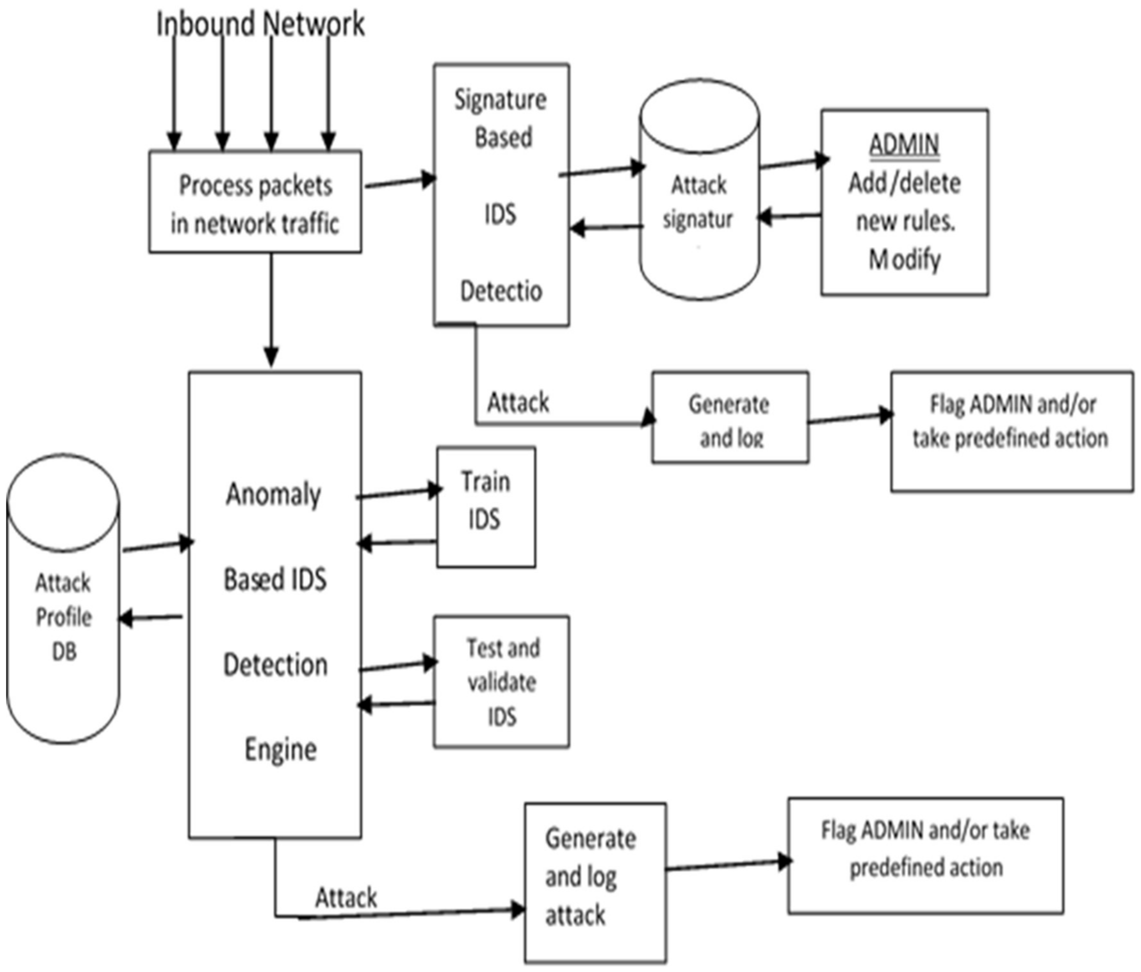

Inbound Network
Process packets in network traffic
Signature Based IDS Detectio
Attack signatur
ADMIN Add/delete new rules. Modify
Attack
Generate and log
Flag ADMIN and/or take predefined action
Attack Profile DB
Anomaly Based IDS Detection Engine
Train IDS
Test and validate IDS
Attack
Generate and log attack
Flag ADMIN and/or take predefined action

Figure 1: Abstraction of a Traditional IDS System

These issues greatly exposed the entire system to potential malicious attacks, self-inflicted denial of service amongst others [41]. Furthermore, these issues are amplified and exasperated when IDS operate in cloud computing environment. This study sets out to propose a solution to these current problems with Intrusion Detection Systems (IDS). Furthermore, the processes of signature or anomaly detection is more cumbersome on cloud computing [16,17]. Merely publishing alarms to security administrators is highly ineffective for cloud computing. Clouds have massive attack surfaces because they are constructed using shared computing, storage, and networking resources for thousands of businesses and have numerous entry points. A successful attack on cloud computing can affect multiple businesses simultaneously. Given the volumes and frequency of attacks possible on cloud computing, automated intrusion detection and prevention methods with high accuracy is mandatory.

Also, there is a serious lack of data sets for training machine learning algorithms. The training databases for detecting attacks and anomalies on cloud computing should be much larger and comprehensive than those in smaller self-hosted networks [16,18]. Creating such training databases is a challenge because it may be large but not comprehensive. Using ready to use databases (existing data mines), like KD99, UNSW-NB15, ISOT, TUIDS, CTU-13 and SSHCure is also prone to multiple types of errors in decision-making as regards true positive intrusions [19]. Such databases may not capture the dynamics of modern types of attacks on cloud computing. Existing data mines are therefore insufficient to build an IDS model to protect cloud computing from the dynamically changing attack patterns by groups of collaborating attackers and Advanced Persistent Treats (APTs) [16,20,21].

In addition, the traditional method of alarms generation for the network administrators to act on them cannot serve the security needs of cloud computing. A high level of automation is required for a Network Intrusion Detection System (NIDS) to function adequately on an enterprise cloud platform. Majority of the attacks detected by the anomaly NIDS are false positives [10]. This implies that automated prevention of attacks, which is essential on cloud platforms cannot be implemented with trustworthiness given these challenges. A session block caused by a false positive may be as expensive as a session permitted by a false negative.
To solve these challenges encountered by NIDS on cloud computing, we propose an IDS model better suited for cloud computing platforms, using deception-based techniques. This design is powered by a hybrid classification module engine, with Machine Learning Algorithm (MLA) to aid in its anomaly decision making. The proposed model is capable of capturing massive traffic behaviours in a running network and build its own anomaly database within a short time, thus increasing its overall efficiency and more accurate outputs.

2. LITERATURE REVIEW

2.1 Network Intrusion Detection Systems

A Network Intrusion Detection System (NIDS) needs to be designed with multiple performance specifications [7,8]. It is expected that they are able to gather data related to suspected attack-like behaviours from the network, store and analyse the collected data on the network or locally, and raise relevant alerts and alarms [8]. The performance of a NIDS in carrying out these tasks is characterised by its hardware capacity, overheads, accuracy of detection of attacks, coverage of attacks (content, aspect, and form of attacks), ability to resist techniques of evading detection, speed of detection and reporting, and capacity to process the workloads assigned in a network [7].

Signature-based systems can be deployed as active or passive sensors [22,10,23]. The active sensors (also called in-line sensors and flow-based sensors) are deployed after the firewall and before the high-security zone comprising the hosts and other systems protected by the NIDS [23,24]. All user traffic is allowed to pass through it such that the inspections can be conducted in near-real-time. Some of the major vendors manufacture single hybrid devices capable of acting as a firewall and a NIDS such as Cisco Advanced Security Appliance (ASA). The anomaly detection NIDS comprises sensors capable of identifying anomalies among ongoing events, analysis systems that confirm an anomaly as a possible hostile (attack-like) behaviour, and a response system capable of generating detailed alarms with alarm metadata [11].

Unlike the signature-based detection technique, anomaly detection is not definitive but is effective in detecting unconventional exploits and attacks, like zero day attacks and insider attackers [12]. There are no universally applicable standards for anomaly detection because the designs may vary with different network technologies and environments. Further, the designs detection strategies depend upon the strategies followed by the actual attackers in a network environment. For example, the detection strategies in banks may be different from those in manufacturing organisations. In order to detect an event as any form of anomaly, the concept of normality needs to be defined within the network environment [13]. An anomaly needs to be defined based on the degree of variation of the network attributes in an event from the normality accepted on the network.

This degree of variation can be estimated using parametric or non-parametric statistical techniques or using classification-based machine learning methods. Other methods may also be used, such as expert systems (knowledge-based), genetic and fuzzy algorithms, ant colony, and ensemble/fusion methods [42]. The classification module is the core of a NIDS as it selects the most appropriate alarm metadata based on carefully observed network attributes in different attack types [25]. NIDS can be configured to monitor and detect attacks based on three architectures: deep packet inspection, flow-based inspection and stateful protocol analysis. This research will focus on and use the flow-based inspection architecture due to its numerous advantages.

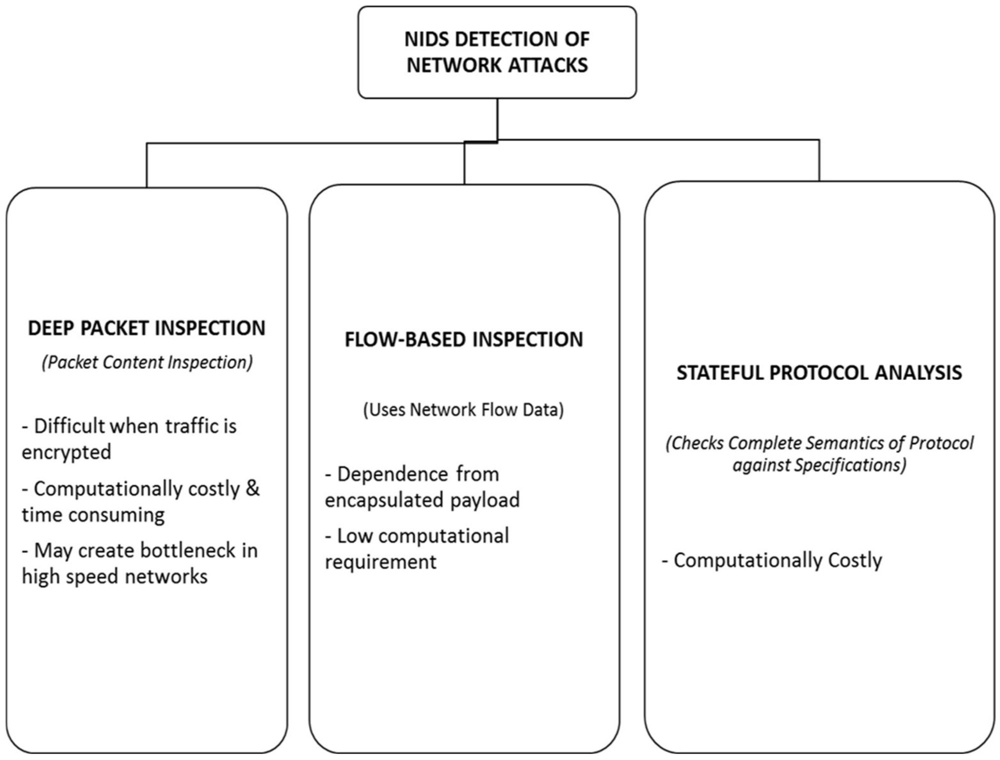

NIDS DETECTION OF NETWORK ATTACKS
DEEP PACKET INSPECTION
(Packet Content Inspection)
- Difficult when traffic is encrypted
- Computationally costly & time consuming
- May create bottleneck in high speed networks
FLOW-BASED INSPECTION
(Uses Network Flow Data)
- Dependence from encapsulated payload
- Low computational requirement
STATEFUL PROTOCOL ANALYSIS
(Checks Complete Semantics of Protocol against Specifications)
- Computationally Costly

Figure 2: NIDS Monitoring and Detection Architecture

2.2 Employment of Machine Learning Algorithms (MLA) in Network Intrusion Detection Systems

Machine Learning has the ability to detect patterns of similarities between two data sets with definitive distance measures [26]. In NIDS, the patterns of attacks in the data flows passing through a network port can be detected by employing an appropriate Machine Learning Algorithm (MLA) [26,27,28,29]. The MLA need to be trained on the patterns (based on optimised values and interrelationships of network attributes called classifiers) that it is expected to detect in the data flows used for testing. The accuracy and effectiveness of MLA depends upon the quality, relevance, and accuracy of the training data set used to train the MLA. MLAs can recognise highly complex data patterns in massive voluminous data flows. In this quest, they can be effectively used to detect new forms of attacks if they have any similarities of patterns with the prior known attacks [43]. MLAs can be designed with two major training approaches: supervised trained MLAs and unsupervised trained MLAs [26].

In supervised training, the inputs and their expected outputs are mapped appropriately within the training data set such that the MLA can generate an output model clearly showing the interrelationships between inputs and outputs. The input data is organised in defined classes for predicting high regression outputs. In real world NIDS, the inputs may be defined with classifiers of network traffic attributes and their interrelationships closely related with different known forms of denial of service attacks, exploits, malware, Trojans, and similar malicious attempts by hackers. In unsupervised training, there are no mappings between the inputs and outputs as the goal of such MLAs is to detect groups of patterns similar to the inputs provided in the data flows. Formulating the training data for machine learning involves transforming the high-level applications traffic into class attributes using classifiers at the network and transport layers representing multiple network attributes [27]. The attributes needs to be simplified through reduction of dimensionality in a pre-processing stage. The resulting data set can be used for supervised, semi-supervised, or unsupervised training of MLAs.

2.3 Cloud Computing

Cloud computing is one of the major revolutionary technologies in the field of information technology, bringing in its wake ideas such as Virtualization, resource pooling, scalability, speedy elasticity and services on-demand [31]. The concept of cloud computing is continuously evolving. Authors in [30] defines cloud computing as “a model for enabling ubiquitous, convenient, on-demand network access to a shared pool of configurable computing resources that can be rapidly provisioned and released with minimal management effort or service provider interaction”. In other words, cloud computing provides shared scalable infrastructure for computing resources such as, servers, storage, networks, services and applications, in a highly dynamic and flexible manner.

Cloud computing in all its forms should present the following major characteristics [32]:

i. Shared computing infrastructure
ii. Dynamic provisioning of resources
iii. Ease of network access
iv. Managed metering services

However, cloud computing is faced with major challenges, which are fast becoming major obstacle in widespread cloud implementations. Chief amongst them being data protection, security and privacy issues [33, 34].

Cloud computing systems are massive infrastructures with IaaS, PaaS, and SaaS service layers [35]. The systems comprise of the service layers using virtualisation technology that enables creation of Virtual Machine Instances (VMIs) under Virtual Private Cloud Environments (VPCEs) and facilitates orchestration of processor, memory, storage, and network resources for them. At the physical level, these resources are drawn dynamically from thousands of physical servers running hypervisors and the guest operating systems on the top of them allocated to individual VMIs [16,17].

The exploits can attack the hypervisors and the guest operating systems directly. This means that thousands of VPCEs and VMIs can be compromised through a large scale single attack. The attack surface may be so large that thousands of businesses can be taken down by a single group of hackers. It is worthy to note here however that our knowledge of cloud computing security vulnerabilities and attack methods are currently limited and constantly evolving [36].

2.4 Network Intrusion Detection Systems (NIDS) on Cloud Computing

NIDS suffer several major challenges when deployed in cloud computing environment [37] such as the ever evolving cloud environment, complex architecture and heterogeneous nature of cloud, massive attack surface in cloud makes security, more susceptible to insider attacks, automated actions cannot be easily taken, etc. In fact, cloud computing is vulnerable to all the known classes of intrusions that have historically troubled self-hosted IT systems. A massive coordinated attack can breach the virtual boundaries affecting the security and privacy of thousands of businesses together. Hence, the stakes of security and privacy controls on cloud computing are significant, requiring both internal and perimeter security controls [38]. Authors in [39] argued that a single solution for NIDS on cloud computing cannot be effective. NIDS on cloud computing should be a hybrid of signature-based, statistics-based, and MLA-based techniques.

The three techniques have their limitations. Signature-based detection requires numerous NIDS probes distributed throughout the cloud to cover the massive volumes of traffic. However, distributed NIDS cannot detect collaborative attacks reliably unless the alerts and alarms of all of them are correlated through a collaborative system. Same limitation applies to statistics-based NIDS as well. MLAs require reliable and complete training data sets of attacks on cloud computing. Because of lack of latest training data sets, MLA technique for NIDS on the clouds is a research-in-progress [16]. For example, attacks like VM escape, VM-to-VM side-channel attacks, VM-to-VMM (virtual machine monitors) attacks, and Guest DoS are new types of attacks [20]. Furthermore, NIDS on cloud platforms cannot be designed with separate systems for intrusion detection and intrusion prevention [21].

They need to be combined. It is impractical to design a system requiring human intervention and decision-making for a network having massive volumes of traffic generated by thousands of organisations. The NIDS designs on the clouds need to have automated decision-making with minimal delays between detection and enforcement of policies. There is a need for comprehensive threat modelling for NIDS operations on cloud computing [20].

This requires a distributed architecture of cloud agents collaborating with controllers to feed information such that anomalies detected can be correlated before pre-processing. The approach of honey pots is an effective way to build comprehensive attack databases to conduct threat modelling [40].

2.5 Deception Based Technology in Network Intrusion Detection

In recent times, Honey pots have gained significant research attention. Authors in [26] presented an extensive classification framework based on bots collected from botnet-based honey pots. They also presented a framework for collaborative analysis of attack traces from multiple traces on an attack network, where multiple MLAs need to be used on the cloud computing to arrive at a final comprehensive classification of attack patterns. As presented by [44], mobile honey pots can be used to collect distributed attack patterns throughout the cloud network that can dynamically roam on the cloud and position themselves intelligently on the propagation paths of ongoing attacks. This research is similar to the dynamic Markov chain formation using intelligent dynamic honey pot agents presented by [45].

The data collected by dynamically distributed mobile honey pot agents need to be collaborated at the analysis engine to create new forms of attack classifiers prevailing on the cloud computing networks. In another study involving mobile intelligent honey pots, [46] designed DNS honey tokens, web server honey tokens, and fake social network avatars to create network and application layer deception models such that attackers believe the victims as real social network users. Authors in [47] presented a technique based on the innovative deception-based defense mechanism referred to as the moving target defense (MTD) technique. This novel defense mechanism is based on the frequent migration of VMs follows a signaling game technique. In addition, [48] presented a novel real-time threat monitoring system centred on the Cloudera platform.

A Flume module was designed and implemented, which helped to reduce and distribute real time data streams form numerous sources into the data analysis mode. Apache Spark 2015 (an implementation of MapReduce) was used to further design the analysis mode. In order to detect abnormalities in network activities and alert the network administrators, the fuzzy *c*-means algorithm and *k*-means. The system could also incorporate and combine the use of Artificial Neural Networks (ANN) and Support Vector Machines (SVM). This threat monitoring system was further trained and evaluated using the relatively new CAIDA Dataset from Chicago Equinix data centre (CAIDA Data 2015), with promising results.

In [49], the researchers presented a deception-based approach to security where a honeypot server was combined with an Intrusion detection and prevention system, which carried out real-time analysis of network traffic. The honeypot was a hybrid deployment, with both high interaction and low interaction honeypots, which were virtually segregated from the intrusion detection and prevention system. The proposed system was setup and tested in a simulated campus network environment. The authors in [50] presented a cloud IDS based on the novel Spiking Neural Network (SNN) architecture (also termed the NeuCube algorithm). The NeuCube algorithm with SNN (core processing module) can easily manage huge data traffic thereby improving performance in classification and identification of various malicious attacks.

It also used two machine learning algorithms including; classification and clustering algorithms. This security architecture was trained and tested using the NSL-KDD dataset. It was shown that the proposed system exhibited high performance in high-speed real-world networks. [37] presented an IDS security framework based on the Support Vector Machine (SVM) for the detection of Denial of Service (DoS) attacks in a virtualized cloud environment. The main aim of this security framework was to identify DoS attacks and other attack types were not evaluated. Furthermore, the CAIDA "DDoS Attack 2007" dataset and 1998 FIFA World Cup' datasets were used in training and evaluation of the presented model. Similarly, [51] presented an IDS design which leveraged on Support Vector Machine (SVM) for classification, while using firefly algorithm for optimization. The firefly algorithm is a meta-heuristic method derived from the behavioural patterns of fireflies. It helps in identifying the best features in a given feature set. The Support Vector Machine (SVM) is trained using the features extracted with the optimized firefly algorithm. This security model was tested in CLOUDSIM virtualized environment.

[52] proposed a novel IDS model which featured a mixture of Artificial Bee Colony (ABC) algorithm, MultiLayer Perceptron (MLP) network and fuzzy clustering algorithm. In this model, while the fuzzy clustering algorithm is used to create numerous training subsets, the ABC algorithm is used to train the multilayer perceptron network by ensuring the optimization of biases values and linkage weight values. The MultiLayer Perceptron (MLP) network on the other hand aids in identifying normal and abnormal network traffic patterns in network traffic flow. The unique combination of ABC, ANN and fuzzy clustering algorithm gives the proposed IDS very great capabilities. The proposed model was simulated using the CloudSim simulator, while the NSL-KDD dataset is used in training, testing and evaluation of the said model, with attacks grouped into four major categories.

Authors in [53] proposed a hypervisor based cloud IDS, where an IDS was deployed at hypervisor level and leverages on data and communications at the hypervisor level, which it uses for anomaly detection. This system employs a mixture of the gradient descent algorithm and the E-Div algorithm to identify anomalous cloud behaviour by observing and noting statistical changes and multivariate sequential change discovery. In other to address the paucity of publicly available datasets, the researchers in conjunction with a cloud service provider (CSP) generated a new cloud intrusion dataset which comprised of a large assortment of attack types. This new dataset was used in the training, testing and evaluation of the model.

Researchers in [54] demonstrated a novel and complex deception-based security architecture which relied on a proxy system for misery digraphs in cloud-based virtual networks. Misery digraphs are systems which have been developed to evolve and change their fundamental structures over a period of time, thereby increasing the entropy in the cloud platform for would be attackers. These misery digraphs (which were developed based on Apache's reverse proxy module) acted by greatly obfuscating and complicating the attack paths of a malicious intruder. This it achieves by introducing endlessly repositioning decoys, while enlarging the pathway to the attacker's target. The misery digraphs as proposed were composed of two major parts: (i). Several identical and bloated paths to a given attack target, (ii). a timetable of relocating/resetting hosts on arbitrarily chosen paths to attack target. Similarly, [55] proposed an attack detection model which introduced intrusion detection for layers of the IOS model.

The presented model is divided into two zones; Host IDS (HIDS)/VM-IDS and Network IDS (NIDS) are located in the first zone and are used as signature-based detectors, while Web-IDS (WIDS)/Application IDS presented as anomaly-based detectors are used in the second zone. Also, [56] presented a novel security framework for an innovative system defense based on dynamic location of honeypots. Here, a distributed honeypot network scheme is configured so as to periodically and randomly change its services. An active attacker can therefore not differentiate between honeypot services and real services, thereby making the malicious network flow more readily recognizable. In other to validate their proposed system and illustrate it's effectiveness, the authors used game theoretic reasoning (Bayesian system game model) and conducted gambit simulations using MATLAB. The service allocation algorithm introduces uncertainty into the system by periodically changing services and keeping the occurrence of honeypots in high probability. Due to the uncertainty introduced to the security system, intending attackers are inevitably forced to abandon lunching any attacks.

3. METHODOLOGY

3.1 Data Collection

In this research, a deception based model was developed using a honeypot network comprising of honeypot-based hosts and networking system on cloud computing and attacks were simulated. The attacks were programmed manually in OPNET Application Characterization Engine (ACE) in similar ways as expected from real world exploit interaction patterns.

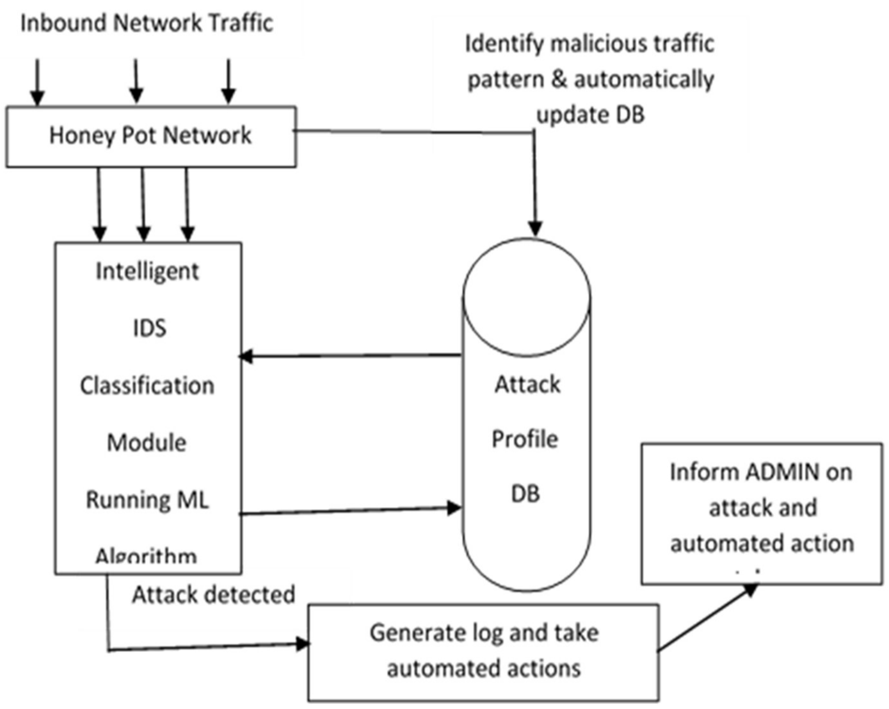

Inbound Network Traffic
Identify malicious traffic pattern & automatically update DB
Honey Pot Network
Intelligent IDS Classification Module Running ML Algorithm
Attack Profile DB
Inform ADMIN on attack and automated action
Attack detected
Generate log and take automated actions

Figure 1: Abstraction of the Proposed IDS

The programming in ACE is visual as well as based on OPNET's proprietary Proto-C C++ blocks codes. The phases of interactions and their interdependencies were programmed in ACE and later packaged as runtimes. The attack profiles of hundreds of attacks were captured to build an attack knowledge database for future analysis and training of the machine learning algorithms. The data collection in modelling and simulation was carried out from the Discrete Events Simulation (DES). An abstraction of the proposed IDS is given in Figure 3. :

The scenario planned in this model comprises a honey pot cloud of ten servers, which will be Virtual Machine Instances (VMIs) in a Virtual Private Cloud Environments (VPCE) simulated on Amazon's Elastic Compute Cloud (EC2), interacting with 80 active attackers. In addition, ten data storage servers (similar to Amazon's Block Store) are configured within the VPCE. An additional five servers were configured for running the algorithmic interactions of machine learning. The attackers are configured to launch the exploits while the honey pot servers are configured to respond actively to the attacks.

This research focused on detecting major attack exploit types including;

- i. Denial of Service (DoS) attacks and Distributed Denial of Service (DDoS) attacks.
- ii. User to root (privilege shift) attacks (U2R attacks)
- iii. Remote (unauthorised) to user (authorised) attacks (R2U attacks)
- iv. Probing attacks (like, port scans and host sweeping).

3.2 Programming Attack Patterns

For interacting with the attackers, two types of python scripts have been created: attack scripts and the embedded socket and API interaction scripts within the attack scripts. Both these scripts have been converted into OPNET ACE interactions such that they can be executed on an OPNET network model. The third type of Python scripts deployed are of Support Vector Machine (SVM) and selected deep learning algorithms for machine learning. An attack exploit may have multiple variants, hence seven hacking sequences were consolidated for programming in C-ACE, as summarized in Table 1. :

Table 1: Consolidated Hacking Sequences for Proposed Model

	Exploit Name	Exploit Description/Purpose
1	Network sniffing using python sniffer.py	To capture and sniff useful information from IP packets to be used for planning an attack. This is typically the first step by Internet hackers to identify potential target hosts out of millions over the Internet.
2	ARP Cache poisoning using Python Scapy	To force fake address resolution protocol entries in the ARP cache of the target machines.
3	Port scanning using python-nmap-0.2.4.tar.gz	To detect all open and vulnerable TCP and UDP ports.
4	Mechanizing a browser session to establish multiple unique connections	To establish multiple unique connections to a web server masquerading as web access sessions. This can also be used as a denial of service attack.
5	Distributed Denial of Service (DDoS) attack	To flood multiple web hosts with TCP and UDP packets to choke the available bandwidth, the network interface card processing capacity, and the processor and the memory of the target host. It may results in hanged or suspended sessions to a web server denying access to genuine users.
6	Key logger hooking and activation	To capture keys entered by operators using local keyboards into the web server consoles.
7	Injecting memory pointers and manipulating call back functions (buffer overflow attack) in Windows machines	To inject fake contents in the buffer memory and then write the buffer contents into the process memory of the running application process. This exploit can poison and crash a running application. It also helps in stealing data.

It may be noted that authors in [57,58,59,60] discussed numerous alternatives for executing specific stage of attacks. For the purpose of valid C-ACE programming in this research, a specific scenario of each stage and the possible next stage based on an example set of outcomes has been chosen. The idea was to build a valid attack sequence and not exploring all the ways to run the particular stage of an exploit. In this way, a traffic profile closer to the practical exploits possible on a honey pot network was consolidated.

3.3 Design of Machine Learning Algorithm (MLA)

The machine learning sequences were designed first to run the training and test sequences. Thereafter, the data processing and classification was done following support vector classifiers, deep belief classifiers (deep machine learning), and extreme learning classifiers (extreme machine learning). Normally, only one classifier is sufficient to detect the distance vectors between data planes. However, as observed from the attack patterns, exploits can behave differently depending upon the exploit type, payload, and attack strategy used by the hackers. Generalising attack detection based on just one classifier may cause false positives and negatives. It is better to be doubly or triply sure before categorising a traffic sequence as an attack type and recording the observations in the archives for future machine learning training.

To be triply sure about the false or true positives, support vector classifiers, deep belief classifiers (deep machine learning), and extreme learning classifiers (extreme machine learning) have been configured to reduce the chances of error to near negligible.
This research presents a simple logic to use the three classifiers called the three digit binary logic. If any one of the classifiers reports a positive, the probability of detected exploit shall be 33.33%, if any two of the classifiers report a positive, the probability of detected exploit shall be 66.66%, and all the three of the classifiers report a positive, the probability of detected exploit shall be 100%. The table below presents all the possible occurrences of a true positive using the three classifiers:

Table 2: True Positive Detection Based on the Three Classifiers of Machine Learning

Support Vector Machine Classifier	Deep Machine Learning Classifier	Extreme Machine Learning Classifier	Outcome Record in the Training Archive
Negative	Negative	Negative	No exploit
Negative	Negative	Positive	True positive with 33.33% probability of exploit
Negative	Positive	Negative	True positive with 33.33% probability of exploit
Negative	Positive	Positive	True positive with 66.66% probability of exploit
Positive	Negative	Negative	True positive with 33.33% probability of exploit
Positive	Negative	Positive	True positive with 66.66% probability of exploit
Positive	Positive	Negative	True positive with 66.66% probability of exploit
Positive	Positive	Positive	True positive with 100% probability of exploit

3.4 Architectural Design of Proposed System

A logical representation of the proposed model is presented in below:

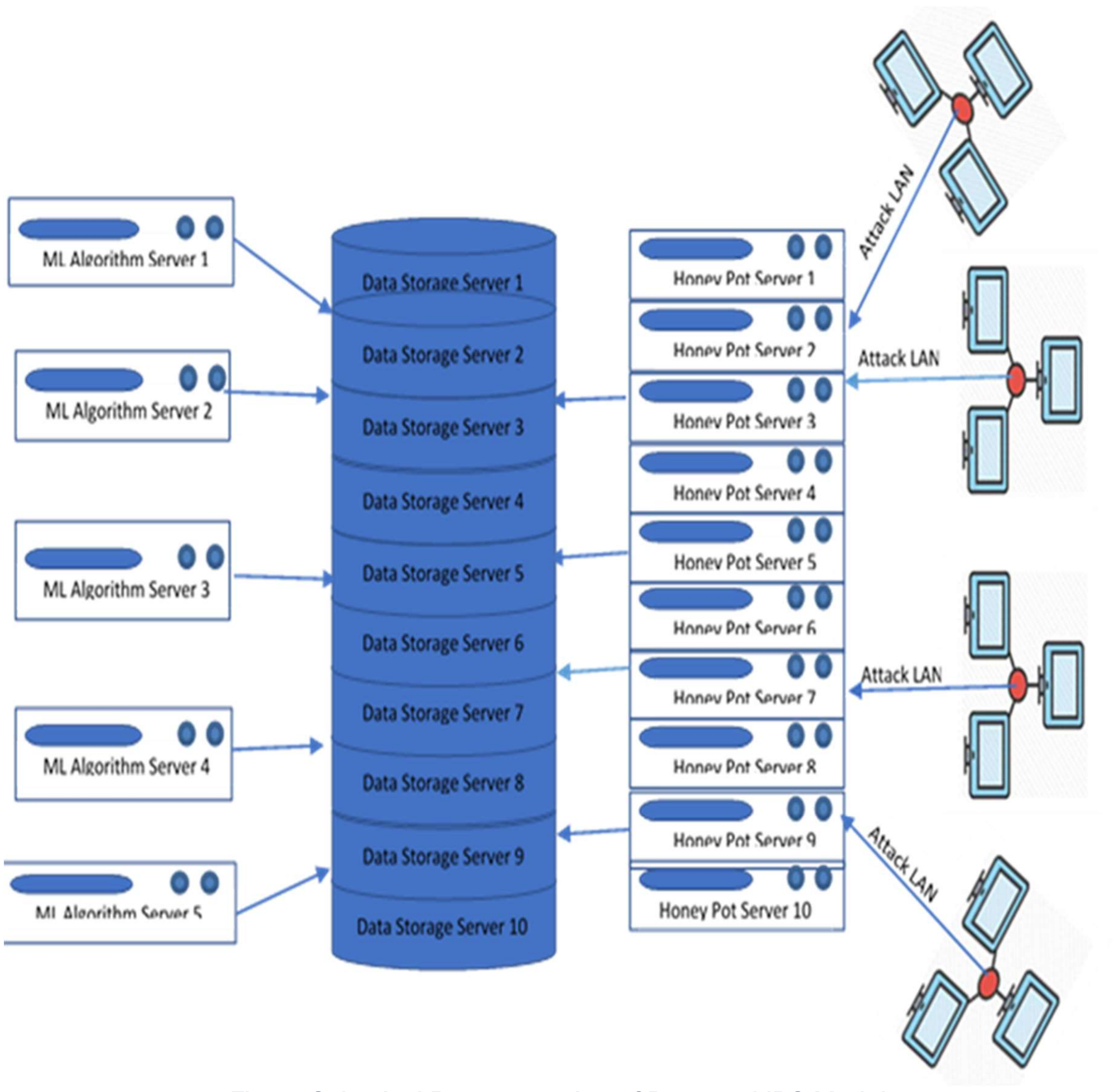


Figure 2: Logical Representation of Proposed IDS Model

The hackers were shown as connecting from 4 local area networks, each having 20 attacking machines. The hacker LANs were assigned 1000 Mbps of Internet bandwidth and their workstations were assigned 100 Mbps dedicated switching capacity in the 2 Gbps (shared) switched backplane. The performance of this network capacity with 80 concurrent hackers was studied in the simulation. A honey pot network was modelled using four Layer-3 Cisco 7000 series switches and two AWS firewalls configured using Cisco PIX 535 10 AE advanced model.

The switches were capable of supporting 10/100/1000 Mbps 1000 Base X links. Each server was assigned 1000 Mbps in the honey pot network. A set of fake web server and database applications were configured within the honey pot servers. The hacker machines were given destination preferences to gain access to these applications and databases. The machine learning servers were kept out of the reach of the attackers in a private LAN facing only the storage devices connected with honey pot servers. The traffic of all the hacking attempts, honey pot interactions, and machine learning interactions were observed from a series of long simulation of the OPNET ACE interactions and documented. The schematics of the proposed IDS model is given below:

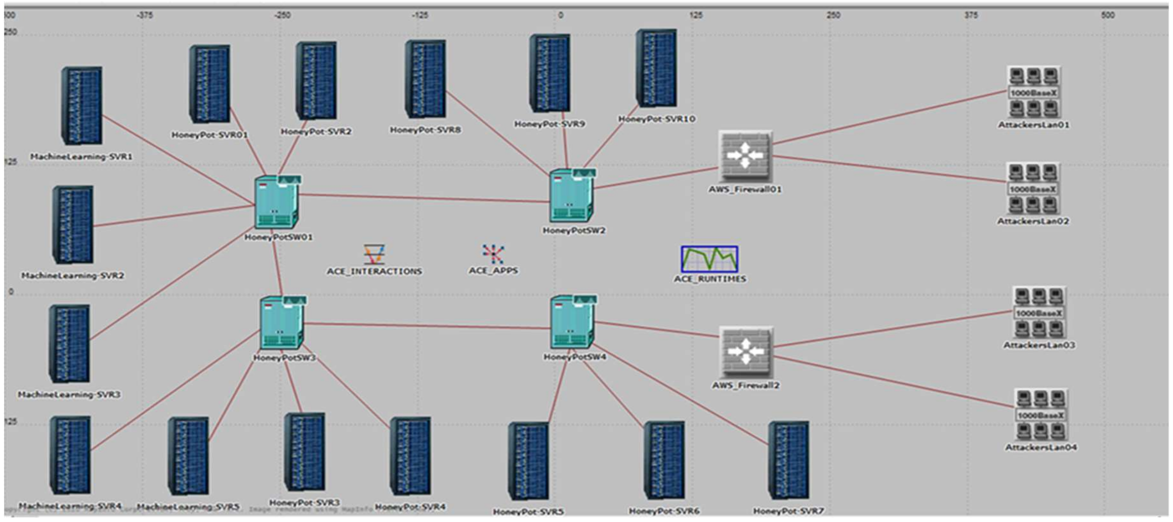


Figure 3: Schematics of Proposed IDS Model

Servers used in this experiment are very large scale HP servers of model "HPAlpha GS 320 32 CPU". Given that the servers will operate as an integrated virtualised cloud, the processors of the servers can be combined forming a pool, thus resulting in a massive scale cloud deployment.

4. RESULTS

The testing of the proposed model was carried out using the discrete events simulation engine (DESE) of OPNET. In this research, DESE output setting was configured to simulate only C-ACE having seven runtimes of the exploits and one runtime of Machine Learning. DESE was executed for two hours in every test session. Two key statistics related with the exploits, machine learning, and their interactions are discussed: Packet Network Delay (PND) and Phase Response Times (PRT). PND and PRT provide detailed view into the performance of the exploits and their interactions, and also the performance of machine learning process collecting data from these interactions.

4.1 Statistics related to the Exploits, Machine Learning, and their Network Interactions

PND statistics of all the exploits and of the machine learning process engaged in collecting data from the logs generated of the running attacks and shipping them to the machine learning servers showed that the highest PND recorded was for DDoS attack and the minimum PND recorded was for the buffer overflow attack. The machine learning PND was negligible for some time and then it started shooting up as spikes. The second highest PND is recorded for port scanning. However, the actual average response times of each of the phases where much higher. As every exploit had multiple phases of execution (algorithmic interactions), the reports where therefore generated for each exploit separately.

The average PRT of "ARP Cache Poisoning using Python Scapy" showed that the response times of all the phases except one are similar: varying between 0 to 40 seconds. However, one phase stands out. Overall, it appears that the ARP cache poisoning attacks' attempts may take about 10 minutes to an hour. Buffer overflow attack works quite fast compared with ARP cache poisoning because only a few bytes of memory pointers injected through a payload into the open socket can result in an effective attack outcome. All the exploit steps are completed within six seconds of phase response time except the return value specification of the functions under exploit. The response time varies significantly as ACE was programmed to generate random payload injection sizes over the exploit period.

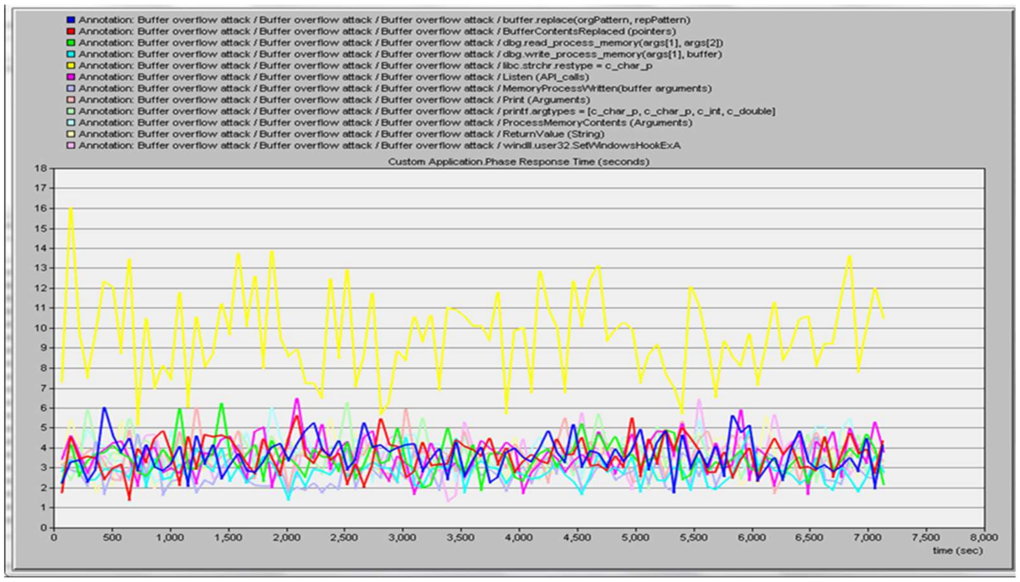


Figure 4: Application PRT of all the Interactions in the Exploit Buffer Overflow Attack.

As observed from Figure 6, the other steps also follow this step in terms of phase response times – the lower the payload injection, the faster is the buffer overflow attack. In real world, the payload content and sizes of replacement memory pointers are decided manually by the hackers [58]. PRT for DDoS showed that once DDoS has started, none of the interactions can perform smoothly. The response times of some of the interactions exceeded 5000 seconds.

There were long chocking periods when no interactions occurred on the network. It appears that managing multiple DDoS attacks was a problem for the honeypot servers after the payloads of the first attack has been injected. With subsequent attacks reaching the payload injection points, there is a cascade of chocking events that may at some stage make the network fully non-responsive causing a completely hanged state.

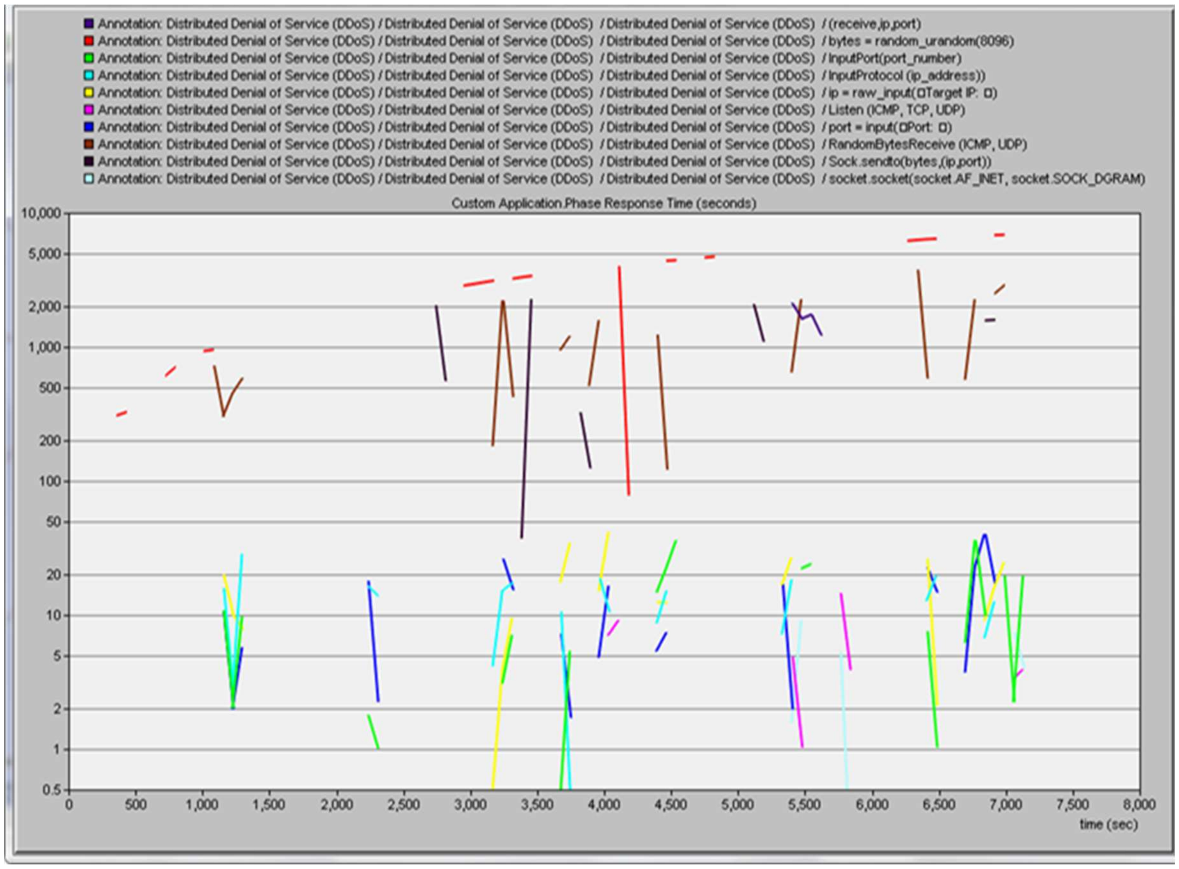


Figure 5: Application PRT of all the Interactions in the Exploit DDoS

Phase Response Times of all the Interactions in the Exploit "Key logger Hooking and Activation" shows continuous flow of interactions with a phase response time not exceeding seven seconds. None of the interactions are highlighted separately in this exploit as they operate within the same range of phase response times. The response time of mechanisation of browser session are quite similar to a mini-level denial of service attack. The server did not choke because the attacks were not as intense as the DDoS. The steps for setting up the mechanising were completed within 10 seconds of phase response times. However, the actual mechanising steps of browsers, creation of hundreds of user agents, and addition of headers and cookies in each browser session took longer times.

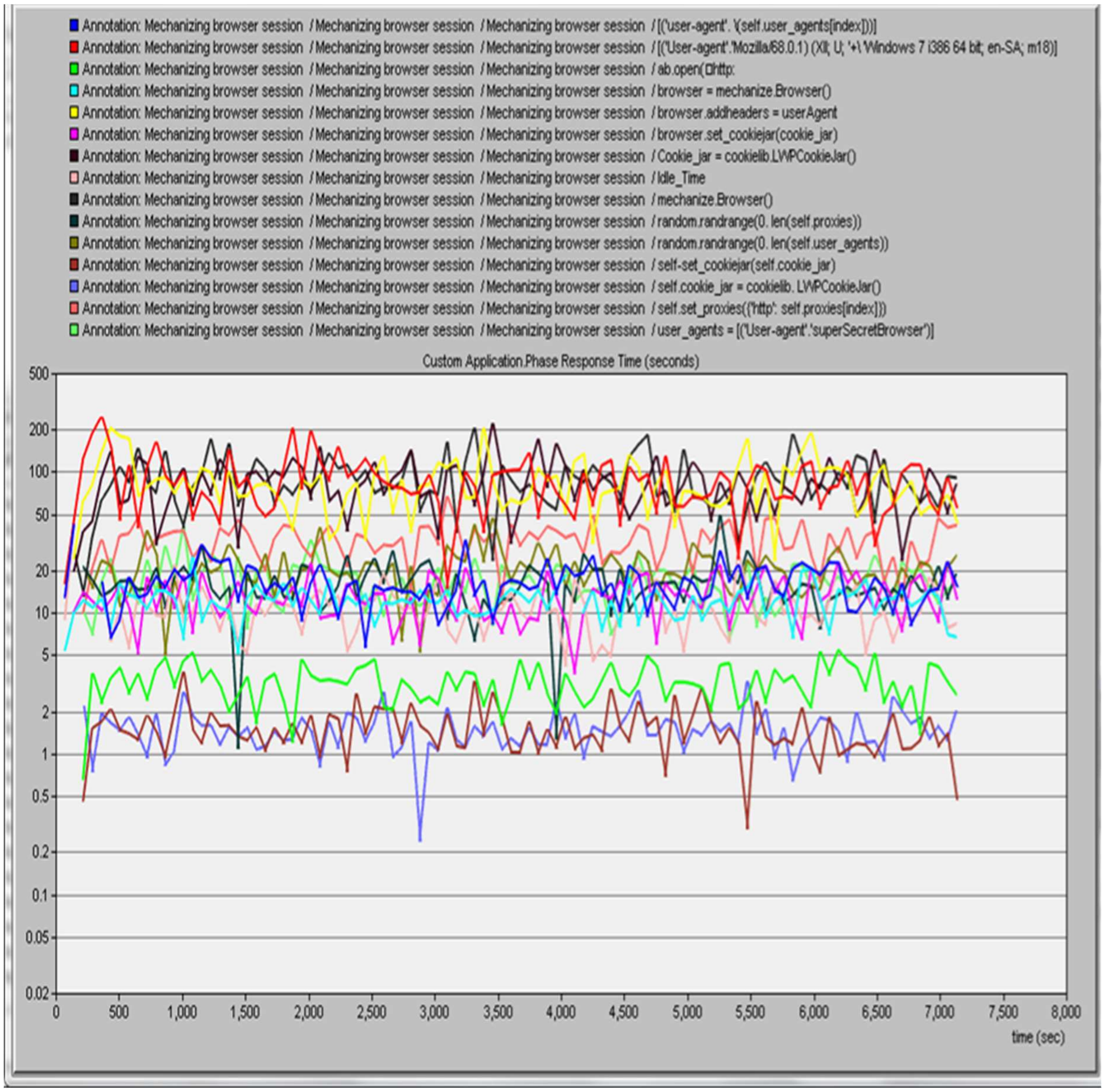


Figure 6: Application PRT of all the Interactions in the Exploit "Mechanising Browsing Session"

Network sniffing was executed for a brief period with phase response times reaching 80 seconds. OPNET simulation showed an abrupt stopping of all the four phases indicating that the system is now ready to be used as a launch pad of the subsequent exploits. It is also observed that the port scanning exploit process has continued through the simulation window because this step also tests the socket connections to the open ports to detect their vulnerabilities.

4.2 Statistics related to Machine Learning and its Network Interactions

The statistics related to machine learning and its network interactions are presented using a plot of phase response times of all its phases. From Figure 9, it may be observed that the machine learning begins with training and testing sequencing of interactions. In the proposed honey pot network, the machine learning has to begin somewhere using the historical flow databases to build the baseline. Either the company may purchase a flow database from a security vendor or use one of the openly available flow-based attack signature libraries. Thus, the training and testing sequences configured may be viewed as conducted by the baseline flow database plus all the appended flow sequences that the honey pot network could add after it was started the first time.

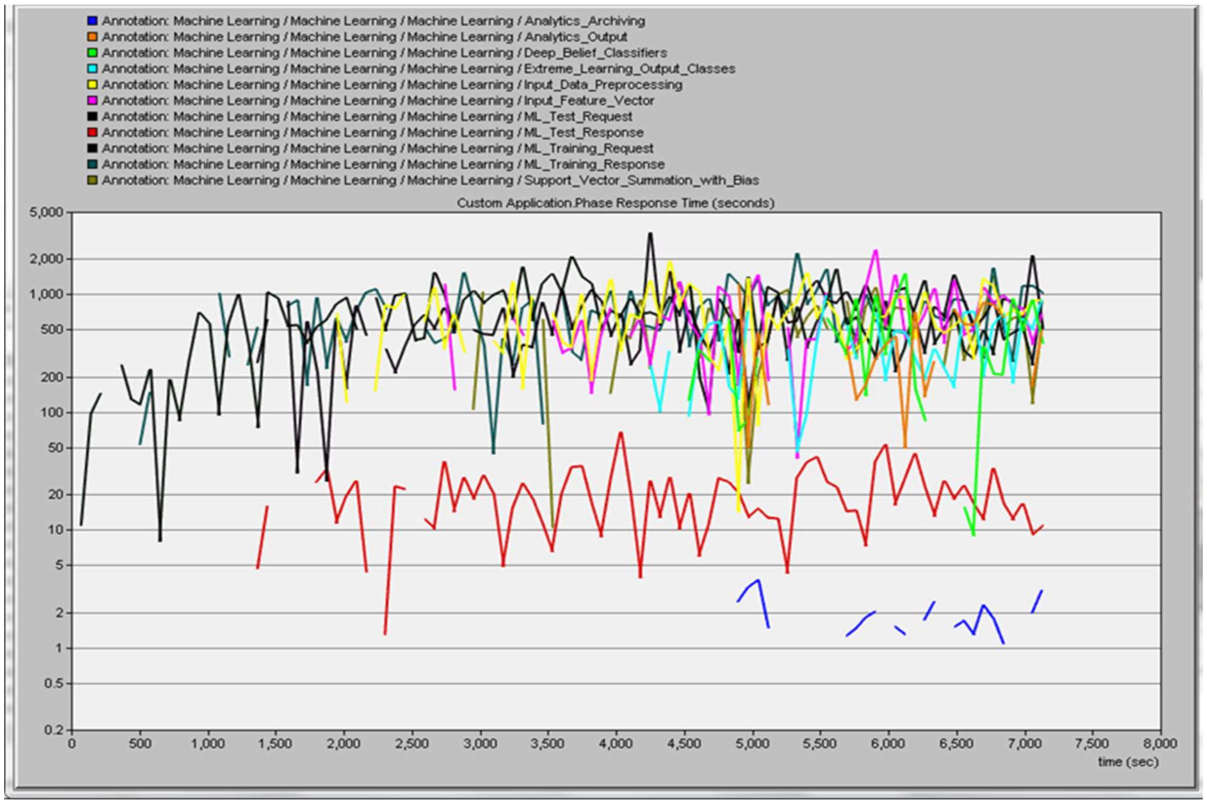


Figure 7: Application PRT of all the Interactions in the "Machine Learning Interactions"

It may be observed that the training requests and responses are overlapping but the test responses are delayed after the test requests were made. This is because the server first waited for the live data to be loaded before running the test sequences. The interactions were configured in this way assuming that this behaviour shall exist in the live honey pot servers as well. The three classifiers started acting almost simultaneously as the test data started flowing in. It is expected that over a prolonged period, the honey pot will become a highly accurate engine to detect an attack thus allowing automatic blocking of exploits with 100% accuracy. This is the purpose of investing in the honey pot network.

4.4 Central Processing Unit (CPU) utilisation reports

The CPU capacity of used firewall was not indicated in the OPNET's model library. However, CPU utilisation of 1 to 1.2% indicates that the firewalls probably had higher CPU capacity than the servers. There may be a possibility that the exploits do not load the firewall CPUs given the high packet forwarding capacity of the firewall's switching backplane (typically configured at 500000 packets per second in default configuration). Furthermore, they simply acted as packet forwarders in this design. This may be the reason why the CPU utilisation of AWS firewalls was miniscule. The CPU utilisation patterns of both the firewalls are identical indicating that the load balancing configuration is working. Different utilisation levels are reflected in each of the servers indicating that they may be processing different exploits at the same time. The attackers and the exploits will be different thus causing a randomised pattern of utilisation as reflected in the results in Figure 10.

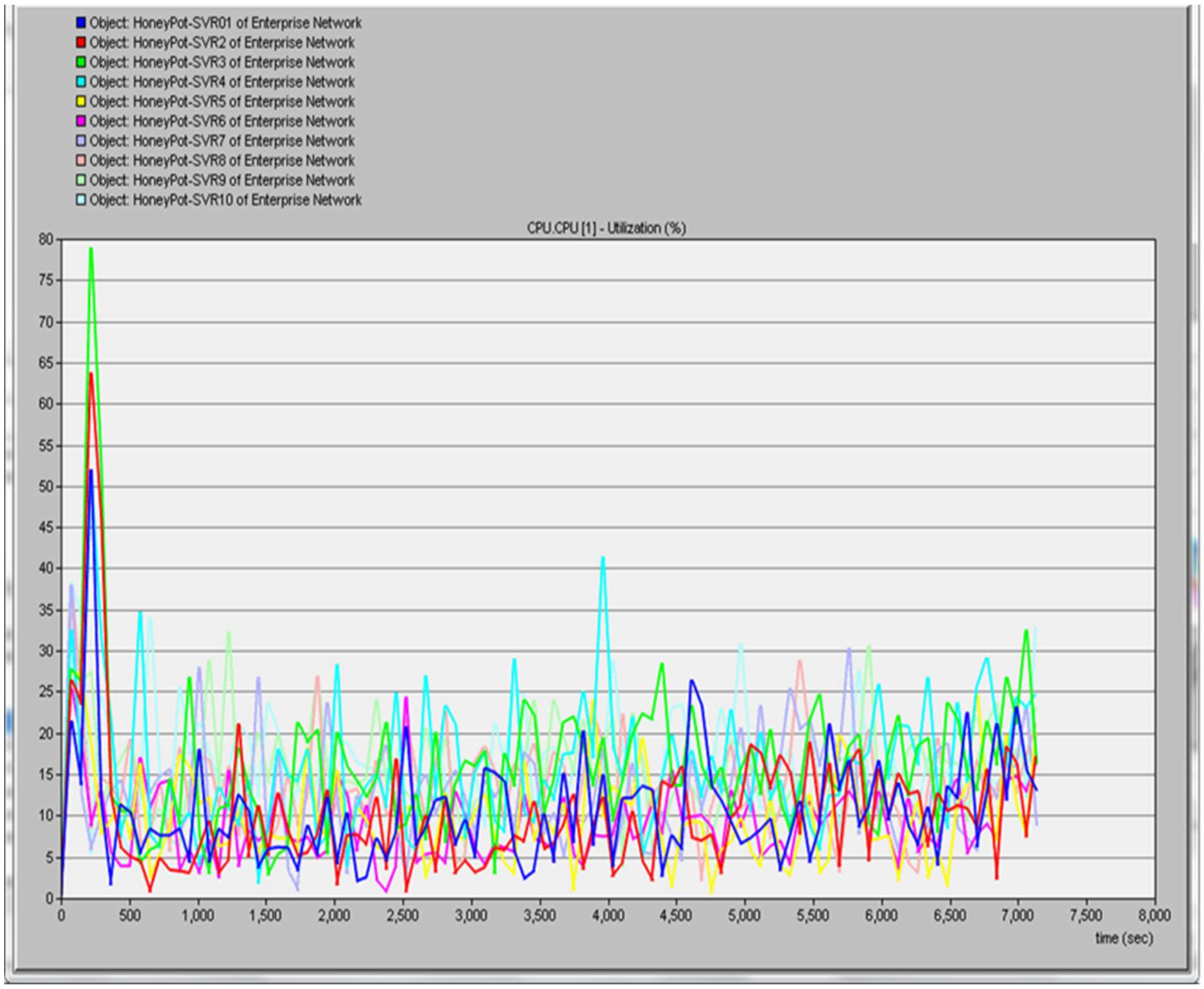


Figure 8 CPU Utilization of the Honeypot Servers

The initial CPU utilisation spikes in each honeypot server were quite high (50 to 80%). However, when the exploits are in progress, the utilisation largely remained at less than 30% barring a few spikes. This may appear as too high for just 80 hackers attacking the system. However, the performance is expected to be much better when modern servers are deployed. Be that as it may, the utilisation levels were stable and didn't increase linearly or exponentially indicating their capability to sustain prolonged exploit sessions. Perhaps, these servers can handle up to 1000 hackers attacking using different exploits. The CPU utilisation patterns of the four honey-pot switches suggest pairing of switches for routing different exploits but the magnitude of utilisation is miniscule again. This may be because of high packet switching capacity in the backplanes. Figure 11 presents the CPU utilisation of the machine learning servers. It showed an interesting pattern; starting from zero and increased steadily at a rapid pace. During the two hours of simulation, the utilisation spikes had reached 40%.

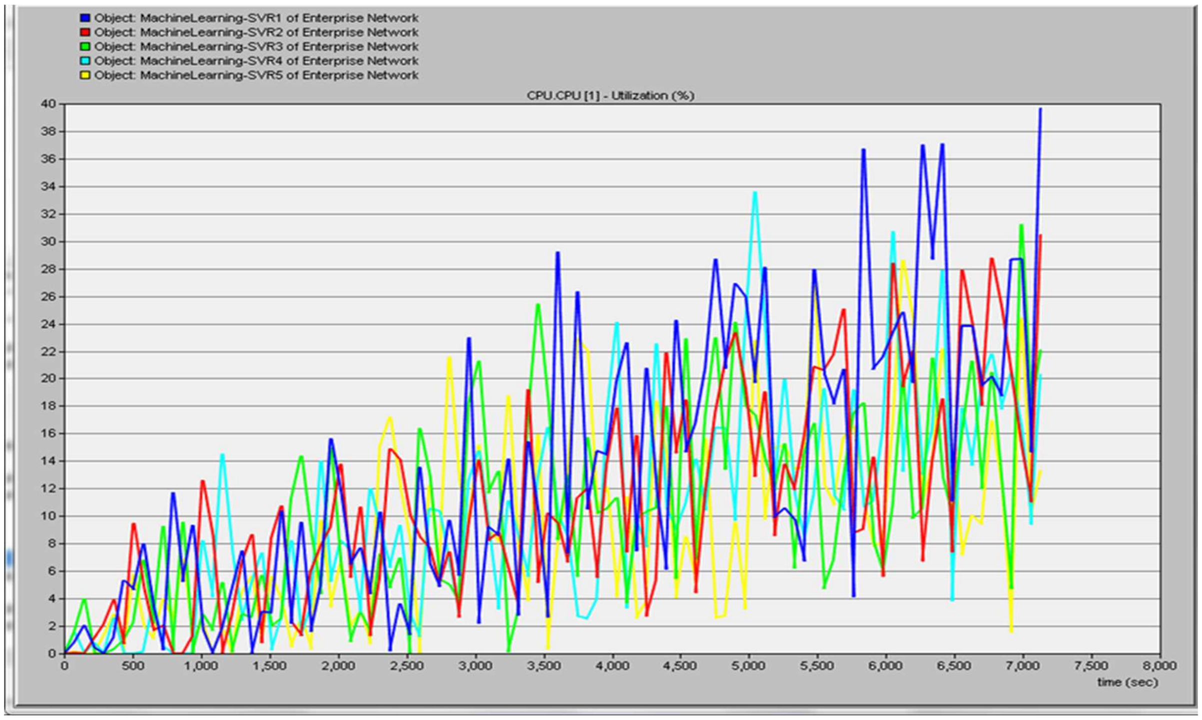
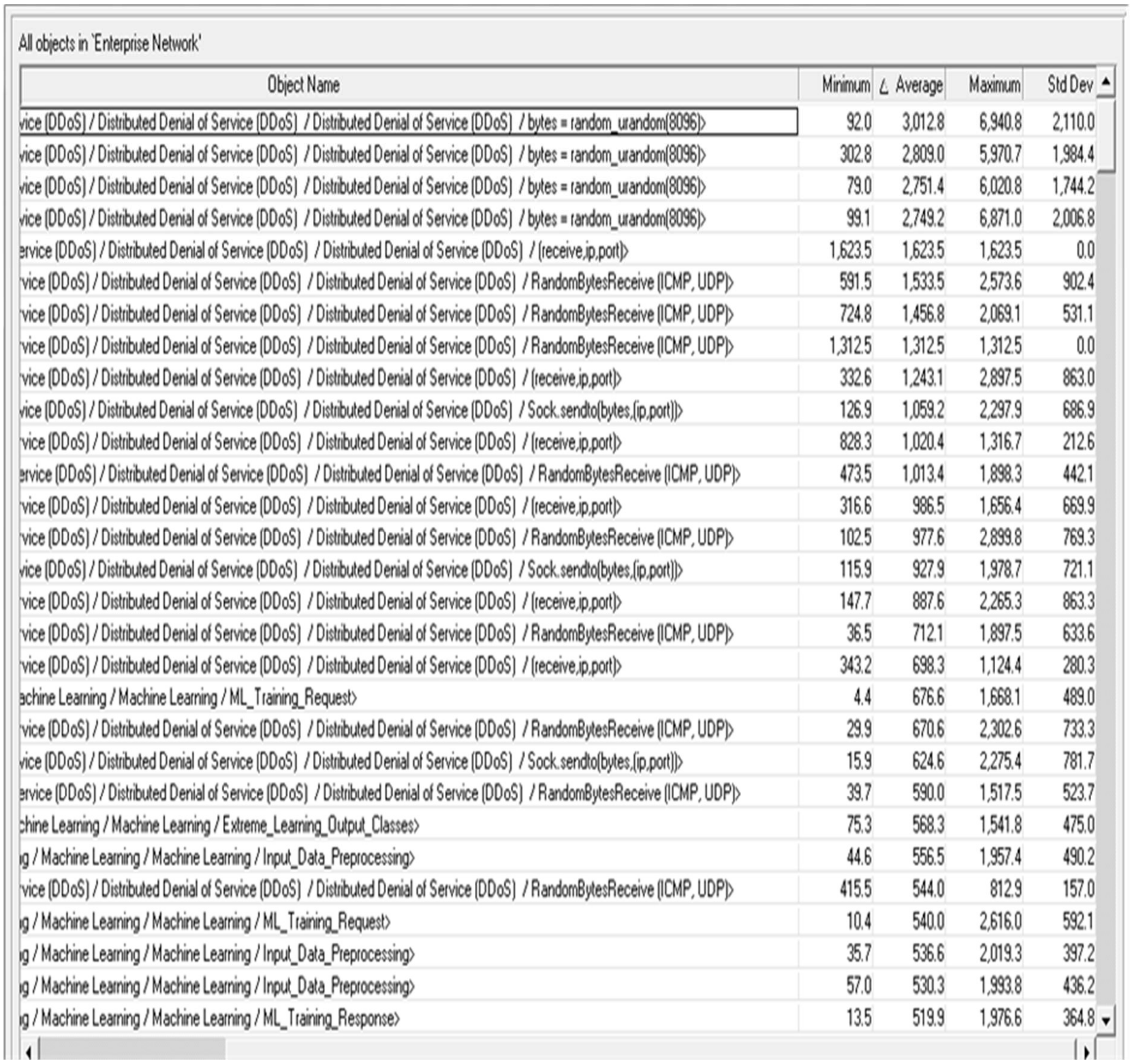


Figure 9: CPU Utilization of the Machine Learning Servers

4.5 Top objects reports

The top objects reports are significant in observing the events that occurred during the simulations runtime in the OPNET's internal engine. Their relative values provide useful information for setting the expectations approximately before the actual network is commissioned. For example, the "receive.ip.port" interaction had taken the highest average round trip time with nil standard deviation. Although their numerical values do not matter, this result prepares a designer to design hardware and network resources' capacities on the cloud accordingly.

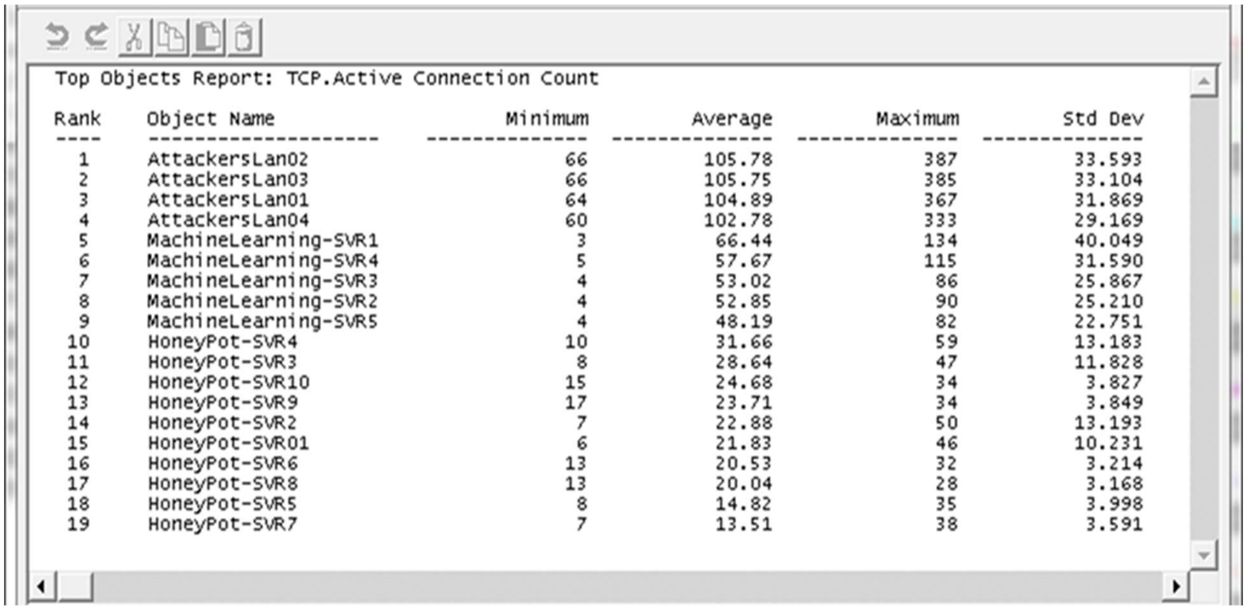


All objects in 'Enterprise Network'

Object Name	Minimum	Average	Maximum	Std Dev
vice (DDoS) / Distributed Denial of Service (DDoS) / Distributed Denial of Service (DDoS) / bytes = random_urandom(8096)>	92.0	3,012.8	6,940.8	2,110.0
vice (DDoS) / Distributed Denial of Service (DDoS) / Distributed Denial of Service (DDoS) / bytes = random_urandom(8096)>	302.8	2,809.0	5,970.7	1,984.4
vice (DDoS) / Distributed Denial of Service (DDoS) / Distributed Denial of Service (DDoS) / bytes = random_urandom(8096)>	79.0	2,751.4	6,020.8	1,744.2
vice (DDoS) / Distributed Denial of Service (DDoS) / Distributed Denial of Service (DDoS) / bytes = random_urandom(8096)>	99.1	2,749.2	6,871.0	2,006.8
ervice (DDoS) / Distributed Denial of Service (DDoS) / Distributed Denial of Service (DDoS) / (receive,ip,port)>	1,623.5	1,623.5	1,623.5	0.0
vice (DDoS) / Distributed Denial of Service (DDoS) / Distributed Denial of Service (DDoS) / RandomBytesReceive (ICMP, UDP)>	591.5	1,533.5	2,573.6	902.4
vice (DDoS) / Distributed Denial of Service (DDoS) / Distributed Denial of Service (DDoS) / RandomBytesReceive (ICMP, UDP)>	724.8	1,456.8	2,069.1	531.1
vice (DDoS) / Distributed Denial of Service (DDoS) / Distributed Denial of Service (DDoS) / RandomBytesReceive (ICMP, UDP)>	1,312.5	1,312.5	1,312.5	0.0
vice (DDoS) / Distributed Denial of Service (DDoS) / Distributed Denial of Service (DDoS) / (receive,ip,port)>	332.6	1,243.1	2,897.5	863.0
vice (DDoS) / Distributed Denial of Service (DDoS) / Distributed Denial of Service (DDoS) / Sock.sendto(bytes,(ip,port))>	126.9	1,059.2	2,297.9	686.9
vice (DDoS) / Distributed Denial of Service (DDoS) / Distributed Denial of Service (DDoS) / (receive,ip,port)>	828.3	1,020.4	1,316.7	212.6
ervice (DDoS) / Distributed Denial of Service (DDoS) / Distributed Denial of Service (DDoS) / RandomBytesReceive (ICMP, UDP)>	473.5	1,013.4	1,898.3	442.1
vice (DDoS) / Distributed Denial of Service (DDoS) / Distributed Denial of Service (DDoS) / (receive,ip,port)>	316.6	986.5	1,656.4	669.9
vice (DDoS) / Distributed Denial of Service (DDoS) / Distributed Denial of Service (DDoS) / RandomBytesReceive (ICMP, UDP)>	102.5	977.6	2,899.8	769.3
vice (DDoS) / Distributed Denial of Service (DDoS) / Distributed Denial of Service (DDoS) / Sock.sendto(bytes,(ip,port))>	115.9	927.9	1,978.7	721.1
vice (DDoS) / Distributed Denial of Service (DDoS) / Distributed Denial of Service (DDoS) / (receive,ip,port)>	147.7	887.6	2,265.3	863.3
vice (DDoS) / Distributed Denial of Service (DDoS) / Distributed Denial of Service (DDoS) / RandomBytesReceive (ICMP, UDP)>	36.5	712.1	1,897.5	633.6
vice (DDoS) / Distributed Denial of Service (DDoS) / Distributed Denial of Service (DDoS) / (receive,ip,port)>	343.2	698.3	1,124.4	280.3
achine Learning / Machine Learning / ML_Training_Request>	4.4	676.6	1,668.1	489.0
vice (DDoS) / Distributed Denial of Service (DDoS) / Distributed Denial of Service (DDoS) / RandomBytesReceive (ICMP, UDP)>	29.9	670.6	2,302.6	733.3
vice (DDoS) / Distributed Denial of Service (DDoS) / Distributed Denial of Service (DDoS) / Sock.sendto(bytes,(ip,port))>	15.9	624.6	2,275.4	781.7
ervice (DDoS) / Distributed Denial of Service (DDoS) / Distributed Denial of Service (DDoS) / RandomBytesReceive (ICMP, UDP)>	39.7	590.0	1,517.5	523.7
chine Learning / Machine Learning / Extreme_Learning_Output_Classes>	75.3	568.3	1,541.8	475.0
g / Machine Learning / Machine Learning / Input_Data_Preprocessing>	44.6	556.5	1,957.4	490.2
vice (DDoS) / Distributed Denial of Service (DDoS) / Distributed Denial of Service (DDoS) / RandomBytesReceive (ICMP, UDP)>	415.5	544.0	812.9	157.0
g / Machine Learning / Machine Learning / ML_Training_Request>	10.4	540.0	2,616.0	592.1
g / Machine Learning / Machine Learning / Input_Data_Preprocessing>	35.7	536.6	2,019.3	397.2
g / Machine Learning / Machine Learning / Input_Data_Preprocessing>	57.0	530.3	1,993.8	436.2
g / Machine Learning / Machine Learning / ML_Training_Response>	13.5	519.9	1,976.6	364.8

Figure 10: Top Objects Report – Request to Response Round Trip Times

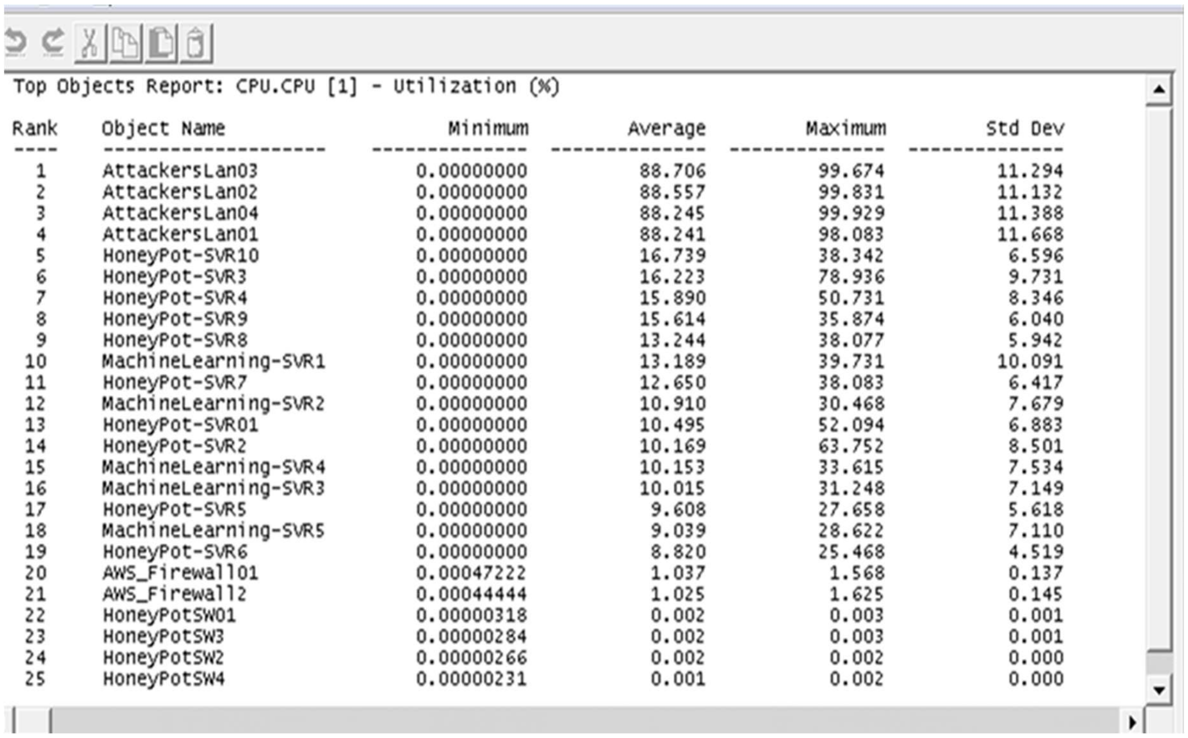
The report on top objects with TCP load (bytes per second) is again interesting. The HoneyPot-SVR9 has the highest average TCP loading honey pot server and the MachineLearning-SVR1 has the highest TCP loading. The reasons cannot be straightforward as the TCP connection profiles similar to those presented in Figures 13 are required to be investigated. However, a designer can safely assume that these loading profiles are arbitrary and can happen with any of the servers. This means, that the highest average TCP loading and highest maximum TCP loading should be taken into account while designing the server capacities.



```
Top Objects Report: TCP.Active Connection Count

Rank    Object Name                   Minimum          Average          Maximum          Std Dev
----    --------------------    --------------   --------------   --------------   --------------
  1     AttackersLan02                     66           105.78              387           33.593
  2     AttackersLan03                     66           105.75              385           33.104
  3     AttackersLan01                     64           104.89              367           31.869
  4     AttackersLan04                     60           102.78              333           29.169
  5     MachineLearning-SVR1                3            66.44              134           40.049
  6     MachineLearning-SVR4                5            57.67              115           31.590
  7     MachineLearning-SVR3                4            53.02               86           25.867
  8     MachineLearning-SVR2                4            52.85               90           25.210
  9     MachineLearning-SVR5                4            48.19               82           22.751
 10     HoneyPot-SVR4                      10            31.66               59           13.183
 11     HoneyPot-SVR3                       8            28.64               47           11.828
 12     HoneyPot-SVR10                     15            24.68               34            3.827
 13     HoneyPot-SVR9                      17            23.71               34            3.849
 14     HoneyPot-SVR2                       7            22.88               50           13.193
 15     HoneyPot-SVR01                      6            21.83               46           10.231
 16     HoneyPot-SVR6                      13            20.53               32            3.214
 17     HoneyPot-SVR8                      13            20.04               28            3.168
 18     HoneyPot-SVR5                       8            14.82               35            3.998
 19     HoneyPot-SVR7                       7            13.51               38            3.591
```

Figure 11: Top Objects Report – TCP Active Connection Counts



```
Top Objects Report: CPU.CPU [1] - Utilization (%)

Rank    Object Name                   Minimum          Average          Maximum          Std Dev
----    --------------------    --------------   --------------   --------------   --------------
  1     AttackersLan03             0.00000000           88.706           99.674           11.294
  2     AttackersLan02             0.00000000           88.557           99.831           11.132
  3     AttackersLan04             0.00000000           88.245           99.929           11.388
  4     AttackersLan01             0.00000000           88.241           98.083           11.668
  5     HoneyPot-SVR10             0.00000000           16.739           38.342            6.596
  6     HoneyPot-SVR3              0.00000000           16.223           78.936            9.731
  7     HoneyPot-SVR4              0.00000000           15.890           50.731            8.346
  8     HoneyPot-SVR9              0.00000000           15.614           35.874            6.040
  9     HoneyPot-SVR8              0.00000000           13.244           38.077            5.942
 10     MachineLearning-SVR1       0.00000000           13.189           39.731           10.091
 11     HoneyPot-SVR7              0.00000000           12.650           38.083            6.417
 12     MachineLearning-SVR2       0.00000000           10.910           30.468            7.679
 13     HoneyPot-SVR01             0.00000000           10.495           52.094            6.883
 14     HoneyPot-SVR2              0.00000000           10.169           63.752            8.501
 15     MachineLearning-SVR4       0.00000000           10.153           33.615            7.534
 16     MachineLearning-SVR3       0.00000000           10.015           31.248            7.149
 17     HoneyPot-SVR5              0.00000000            9.608           27.658            5.618
 18     MachineLearning-SVR5       0.00000000            9.039           28.622            7.110
 19     HoneyPot-SVR6              0.00000000            8.820           25.468            4.519
 20     AWS_Firewall01             0.00047222            1.037            1.568            0.137
 21     AWS_Firewall2              0.00044444            1.025            1.625            0.145
 22     HoneyPotSW01               0.00000318            0.002            0.003            0.001
 23     HoneyPotSW3                0.00000284            0.002            0.003            0.001
 24     HoneyPotSW2                0.00000266            0.002            0.002            0.000
 25     HoneyPotSW4                0.00000231            0.001            0.002            0.000
```

Figure 12: Top Objects Report – CPU Utilization

The distribution of TCP connections show that the attacker LANs initiated the highest number of connections because they are the ones driving all these activities and the honey pot network is simply responding to their interactions smartly. The machine learning servers have experienced more connection counts than the honey pot servers.

5. DISCUSSION

Except the DDoS attacks and to some extent the buffer overflow attacks, the payloads of the exploits were within the capacities of the ten honey pot servers. The packet network delay and phase response times of all the exploits reflected minor performance issues except the DDoS and buffer overflow. The completion of phases in almost each attack took a lot of time. For example, ARP cache poisoning attacks' attempts may take about 10 minutes to an hour. The buffer overflow attacks may take much longer times as they depend upon the payload that the attacker may plan to inject into the program execution memory of the target host. As buffer overflow can be used to exploit a vulnerability of the program running in the memory, the payload content and sizes of replacement memory pointers depends upon the size of replacement program segments injected by the attacker. This is one of the oldest and most dangerous and effective exploits used by attackers, especially when the vulnerabilities of a newly built program have not been realised in a production environment.

Organisations can use honey pots to not only train their AI-based intrusion detectors for making prevention decisions but also carefully study the type of injected programs sent by the hackers enabling production of quick security patches to be applied on the running programs. The most difficult exploit to manage in a honey pot network is the DDoS. DDoS needs to be managed manually (as the intrusion detector may not get enough time to decide before the network choking happens) and insert policy updates for blocking the sources based on traces of attackers before restarting the honey pot. DDoS will require active management and blocking to keep the network running. DDoS attack traces are not as complex as buffer overflow, but may be very expensive for honey pot operators. It is advised that all forms of exploits that can finally lead to DDoS-like behaviours need to be managed with a similar strategy. This conclusion is derived from the experiences in simulating the exploit capable of mechanising of the browser sessions.

The pattern of response times of mechanisation of browser exploit appears similar to a regular and genuine yet heavy duty web services application running on the cloud web server. If the commands are hidden, it is very difficult for an IDS to differentiate between regular web services traffic and the mechanisation exploit traffic. Even the idle time response times are between 10 to 50 seconds indicating long durations of no activity on the server, which may appear genuine when no users are browsing. The response times during actual browser mechanisation (red curve) after the command for mechanisation has been issued (blue curve) is between 50 to 200 seconds indicating heavy duty client traffic. In real world hacking scenarios, the hacker can create multiple client types through separate and parallel mechanisation commands. An overloading of the web server by mechanised browsing sessions beyond its processing and memory capacities can crash the server. On cloud computing, enormous CPU and memory resources may be drawn thus escalating the usage bills of the honey pot network.

A DDoS is a simple exploit, but the mechanisation of browser exploit is much more sophisticated. The payload sizes in the simulation were configured to increase as per exponential distribution. The simulation results showed peaked phase response times of 200 seconds indicating heavy duty web traffic. A hacker can run parallel session mechanisations and cause secondary DDoS attempts on the running web services and applications, such as cookie overloading, overloading through web services scripting, overloading by massive voting strings, overloading by running video and audio streams available on the web server, overloading by filling millions of forms, overloading through SQL injections, and overloading through junk file uploading in cloud storages. Recording the interactions of browser mechanising through the three machine learning algorithms is very useful for training the intrusion detector.

However, active honey pot administration is needed to kill sessions periodically to keep the loading and hence cloud usage costs under control. The machine learning sequences suffered session breaks in the test responses and analytics archiving interactions. Most probably, these sessions were occurring at the time of DDoS execution because all the interactions were executed in a single runtime. Hence, it may be safely concluded that all interactions of machine learning will run effectively on the honey pot server. The operator should ensure that DDoS or DDoS-like attacks (such as browser mechanising) should be killed timely to ensure that the machine learning interactions run without any disruptions and the cloud billing is kept as low as possible.

The honey pot network was found to be sufficient to face 80 concurrent attackers for all the exploit types except the distributed denial of services (DDoS) attacks. The CPU utilisation statistics of honey pot servers showed that initial CPU utilisation spikes in each server were from 50 to 80% but later settled at below 30% (however, this research does not claim any benchmarking). The utilisation will depend upon the attack types, engagement lengths, and payload sizes. Also, too many DDoS events will boost the utilisation to significant levels. A good knowledge of black hat hacking sequences is needed to program the vulnerabilities in the honeypot network carefully such that they can effectively interact with the incoming exploits with reasonably good level of denial and blocking to make the hackers struggle through them.

It is important that the vulnerabilities should not appear as deliberate to maintain interest among the hackers. For example, there should be no default port opened on any server and the sessions should be encrypted. However, there may be some hash keys hidden in folders that hackers can somehow access to break the encryption. Thus the services should be unreal but offer enough business sense for attackers to take interest in them. Furthermore, the services should match the business of the organisation to eliminate any chances of doubt in the minds of the attackers. The observation as a cause of concern was of the CPU utilisation of machine learning servers. It does seem that sizing of the machine learning servers was severely underestimated. For ten honey pot servers, the capacity of five machine learning servers running the three algorithmic sequences (support vector machine, deep machine learning, extreme machine learning) is rather inadequate. During the simulation period of two hours, the CPU utilisation spikes had reached 40% increasing steadily from zero percent. This means, it could have reached 100% in another three hours of simulation or even earlier if the distribution was exponential.

It is apparent that the number of machine learning servers may need to be larger than the honey pot servers. However, it should be noted that the traffic sizing for each interaction was done as per the black hat books. In real world, they may differ depending upon the programs used and the interactions between hackers and target hosts. Hence, the conclusions presented here are not performance benchmarks.

6. CONCLUSION

This study presented an alternative strategy of creating accurate databases of attack patterns for improving machine learning performance and increasing true positives and true negatives in detection of attacks and anomaly behaviours in NIDS operating on cloud platforms. Fundamentally, well configured honeypot networks are very feasible and useful solution for building large yet highly relevant and focussed data marts of logs of exploit interactions. These logs can train the artificial intelligence-enabled machine learning processors for identifying false positives, true positives, false negatives, and true negatives about the exploit versus realistic business transactions. Thus, when the intrusion prevention blocks are enabled based on the trained AI knowledge, the organisation can be protected from exploits while reducing or eliminating blocking of genuine business transactions and communications, thus enabling the proposed model to achieve remarkable levels of accuracy.

Furthermore, this research also indicated that it is feasible to use cloud computing for hosting organisation-specific honey pot networks. Hosting large capacity data mart servers on cloud computing is economical and feasible in terms of performance, capacities, and reliability. Honey pot networks on cloud computing can be configured using virtual machine images in a virtual private cloud environment. The storage systems for data mining can be deployed in the electronic block storage systems, which are service-oriented storage blocks published for enterprise storage usage on the top of virtualised storage systems combining network attached storage systems and storage area networks. AI-enabled deception-based intrusion prevention may be a major breakthrough in cyber security and this research presented one of the ways to achieve this unique capability.

REFERENCES

[1] Columbus L. (2018, Sep 23) Roundup of cloud computing forecasts and market estimates 2018, Forbes magazine. Online. Retrieved from: https://www.forbes.com/sites/louiscolumbus/2018/09/23/roundup-of-cloud-computing-forecasts-and-market-estimates-2018/#935204d507b0

[2] WhiteHat Security (2019) 2019 Application Security Statistics Report. Online. Retrieved from: https://info.whitehatsec.com/Content-2019-StatsReport_LP.html?utm_source=website&utm_medium=0819-Website-WhiteHat2019StatisticsReport

[3] Symantec (2017, a), "Internet Security Threat Report, ISTR." 22: 17.

[4] Symantec (2017, b), "Internet Security Threat Report, ISTR." 22: 14 and 16-18.

[5] Agarwal N & Hussain Z. S. (2018) "A Closer Look at Intrusion Detection System for Web Applications". Security and Communication Networks, Volume 2018, Article ID 9601357. DoI: https://doi.org/10.1155/2018/9601357

[6] Al-Jarrah, O. & Arafat, A. (2014), "Network Intrusion Detection System using Attack Behaviour Classification", *In 2014 5th International Conference on Information and Communication Systems (ICICS)*, 1-3 April 2014, Irbid, Jordan, p. 1-6, IEEE Xplore.

[7] Milenkoski, A., Viera, M., Kounev, S., Avritzer, A., & Payne, B. D. (2015), "Evaluating Computer Intrusion Detection Systems: A Survey of Common Practices", *ACM Computing Surveys*, 48 (1): p. 12:1-12:41, ACM.

[8] Mitchell, R. & Chen, I. (2014), "A Survey of Intrusion Detection Techniques for Cyber-Physical Systems", *ACM Computing Surveys*, 46 (4): 55:1-55:21, ACM.

[9] Chetan, R. & Ashoka, D. V. (2012), "Data Mining Based Network Intrusion Detection System: A Database Centric Approach", *In 2012 International Conference on Computer Communication and Informatics (ICCCI -2012)*, 10-12 January, 2012, Coimbatore, India, p. 1-6, IEEE Xplore.

[10] Hubballi, N. & Suryanarayanan, V. (2014), "False alarm minimization techniques in signature-based intrusion detection systems: A survey", *Computer Communications*, 49: p. 1–17, Elsevier.

[11] Garcia-Teodoro, P., Diaz-Verdejo, J., Macia-Fernandez, G., & Vazquez, E. (2009), "Anomaly-based network intrusion detection: Techniques, systems and challenges", *Computers & Security*, 28: p. 18-28, Elsevier.

[12] Ahmed, M., Mahmood, A. N., & Hu, J. (2016), "A survey of network anomaly detection techniques", *Journal of Network and Computer Applications*, 60: p. 19-31, Elsevier.

[13] Bhuyan, M. H., Bhattacharya, D. K., & Kalita, J. K. (2014), "Network Anomaly Detection: Methods, Systems and Tools", *IEEE Communication Surveys & Tutorials*, 16 (1): p. 308-336, IEEE Xplore.

[14] Song, C. & Ma, K. (2009), "Design of Intrusion Detection System Based on Data Mining Algorithm", *In 2009 International Conference on Signal Processing Systems*, 15-17 May 2009, Singapore, p. 370-373, IEEE Xplore.

[15] Debar, H., Dacier, M., & Wespi, A. (1999), "Towards a taxonomy of intrusion-detection systems", *Computer Networks*, 31: p. 805-822, Elsevier.

[16] Kumar, R. S. S., Wicker, A., & Swann, M. (2017), "Practical Machine Learning for Cloud Intrusion Detection: Challenges and the Way Forward", *In AISec '17 Proceedings of the*

10th ACM Workshop on Artificial Intelligence and Security, 03 November 2017, Dallas, Texas, USA, p. 81-90, ACM.

[17] Yassin, W., Udzir, N. I., Muda, Z., Abdullah, A., & Abdullah, M. T. (2012), "A Cloud-based Intrusion Detection Service framework", *In Proceedings Title: 2012 International Conference on Cyber Security, Cyber Warfare and Digital Forensic (CyberSec)*, 26-28 June 2012, Kuala Lumpur, Malaysia, p. 213-218, IEEE Xplore.

[18] Matija, S. (2016), "Machine learning for network-based malware detection", Aalborg University, Denmark, p. 1-90.

[19] Moustafa, N. & Slay, J. (2016), "The evaluation of Network Anomaly Detection Systems: Statistical analysis of the UNSW-NB15 data set and the comparison with the KDD99 data set", *Information Security Journal: A Global Perspective*, 25 (1-3): p. 18-31, Taylor & Francis.

[20] Mishra, P., Pilli, E. S., Varadharajan, V., & Tapukula, U. (2017), "Intrusion detection techniques in cloud environment: A survey", *Journal of Network and Computer Applications*, 77: p. 18–47, Elsevier.

[21] Kene, S. G. & Theng, D. P. (2015), "A Review on Intrusion Detection Techniques for Cloud Computing and Security Challenges*", In IEEE 2nd International Conference on Electronics and Communication Systems (ICECS' 2015),* 26-27 February 2015, Coimbatore, India, p. 227-232, IEEE Xplore.

[22] Garofalo, M. (2017), "Big Data Analytics for Flow-Based Anomaly Detection in High-Speed Networks", Published PHD Thesis, Universita Degli Studi Di Napoli Federico II, p. 1-108.

[23] Scarfone, K. & Mell, P. (2012), "Guide to Intrusion Detection and Prevention Systems (IDPS)", Recommendations of the National Institute of Standards and Technology, Special Publication 800-94, Revision 1, 1-129.

[24] Umer, M. F., Sher, M., & Bi, Y. (2017), "Flow-based intrusion detection: Techniques and challenges", *Computers & Security*, 70: p. 238-254, Elsevier.

[25] Qassim, Q. S., Zin, A. M., & Aziz, M. J. A. (2016), "Anomalies Classification Approach for Network-based Intrusion Detection System", *International Journal of Network Security*, 18 (6): p. 1159-1172.

[26] Stevanovic, M. (2016), "Machine learning for network-based malware detection", Published PHD Thesis, Aalborg Universitetsforlag, p. 1-90, PhD Series: Faculty of Engineering and Science, Aalborg University, Aalborg Universitet, Denmark.

[27] Buczak, A. L. & Guven, E. (2016), "A Survey of Data Mining and Machine Learning Methods for Cyber Security Intrusion Detection", *IEEE Communications Surveys & Tutorials*, 18 (2): p. 1153-1176, IEEE Xplore.

[28] Hodo, E., Bellekens, X., Hamilton, A., Tachtatzis, C., & Atkinson, R. (2017), "Shallow and Deep Networks Intrusion detection System: A Taxonomy and Survey", Department of Electronic & Electrical Engineering, University of Strathclyde, p. 1-43.

[29] Salem, M. (2014), "Adaptive Real-time Anomaly-based Intrusion Detection using Data Mining and Machine Learning Techniques", Published PHD Thesis, Faculty of Electrical Engineering / Computer Science, University of Kassel, Germany, p. 1-195.

[30] Azad, C. & Jha, V. K. (2016), "A Novel Fuzzy Min-Max Neural Network and Genetic Algorithm-Based Intrusion Detection System", Book Chapter: *Advances in Intelligent Systems and Computing*, Vol. 380: p. 429-439, New Delhi: Springer.

[31] Alkhald M. F. (2017) "Value characteristics of cloud computing and big data attributes". European Journal of Business and Management, Volume 9, Number 30, 2017.

[32] Sriniva J., Reddy S. V. K. & Qyser M. A. (2012) "Cloud computing basics". International Journal of Advanced Research in Computer and Communication Engineering, Vol. 1, Issue 5, July 2012.

[33] Morsy A. M., Grundy J. & Müller I. (2010) "An analysis of the cloud computing security problem". In Proceedings of APSEC 2010 Cloud Workshop, Sydney, Australia, 30th Nov 2010.

[34] Odun-Ayo I., Ajayi O. and Misra S. (2018) "Cloud computing security: issues and developments". Proceedings of the World Congress on Engineering 2018, Vol I, WCE 2018, July 4-6, 2018.

[35] Puthal D., Sahoo B. P. S., Mishra S. & Swain S. (2015) "Cloud computing features, issues and challenges: a big picture". 2015 International Conference on Computational Intelligence & Networks (CINE 2015). DoI: 10.1109/CINE.2015.31

[36] Aldribi A., Traore I., Moa B., Nwamuo O. (2019) "Hypervisor-based cloud intrusion detection through online multivariate statistical change tracking". Computers & Security 88 (2020) 101646. DoI: https://doi.org/10.1016/j.cose.2019.101646

[37] Abusitta A., Bellaiche M & Dagenais M. (2018) "An SVM-based framework for detecting DoS attacks in virtualized clouds under changing environment". Journal of Cloud Computing: Advances, Systems and Applications (2018) 7:9. DoI: https://doi.org/10.1186/s13677-018-0109-4

[38] Farroha, B. S. & Farroha, D. L. (2011), "An Investigative Analysis Into Security in the Clouds and the Impact of Virtualization on the Security Architecture", *In the 2011 Military Communications Conference - Track 3 - Cyber Security and Network Operations*, US Department of Defense, p. 1369-1374, IEEE Xplore.

[39] Keegan, N., Ji, S., Chaudhary, A., Concolato, C., Yu, B., & Jeong, D. H. (2016), "A survey of cloud-based network intrusion detection analysis", *Human-Centric Computing & Information Sciences*, 6 (19): p. 1-16, Springer Open.

[40] Saadi, C. & Chaoui, H. (2016), "Cloud Computing Security Using IDS-AM-Clust, Honeyd, Honeywall and Honeycomb", *Procedia Computer Science*, 85: p. 433 – 442, Elsevier.

[41] Yange, S. T., Oluoha, O. & Abdulmuminu, M. Y. (2020) "A data analytics system for network intrusion detection using decision tree". Journal of Computer Sciences and Applications, 2020, Vol. 8, No. 1, 21-29. DoI: DOI:10.12691/jcsa-8-1-4

[42] Khraisat, A., Gondal, I., Vamplew, P. and Kamruzzaman, J. (2019) "Survey of intrusion detection systems: techniques, datasets and challenges". Cybersecurity, (2019) 2:20. DoI: https://doi.org/10.1186/s42400-019-0038-7

[43] Ahmad, Z., Khan, S. A., Shiang, W. C., Abdullah, J. & Ahmad F. (2020) "Network intrusion detection system: A systematic study of machine learning and deep learning approaches". Transactions on Emerging Telecommunications Technologies. Volume 32, Issue1, 2021. DoI: https://doi.org/10.1002/ett.4150

[44] Vasilomanolakis, E. (2016), "On Collaborative Intrusion Detection", Published PHD Thesis, Technische Universität Darmstadt, p. 1-233.

[45] Bar, A., Shapira, B., Rokach, L., & Unger, M. (2016), "Identifying Attack Propagation Patterns in Honeypots using Markov Chains Modeling and Complex Networks Analysis", *In 2016 IEEE International Conference on Software Science, Technology and Engineering*, 23-24 June 2016, Beer-Sheva, Israel, p. 28-36, IEEE Xplore.

[46] Virvilis-Kollitiris, N. (2015), "Detecting Advanced Persistent Threats through Deception Techniques", Published PHD Thesis, Information Security and Critical Infrastructure Protection (INFOSEC) Laboratory, p. 1-174, Department of Informatics, Athens University of Economics & Business, Greece.

[47] Adili, T. M. T., Mohammadi, A., Manshaei, H. M., & Rahman, A. M. (2017) "A Cost-Effective Security Management for Clouds: A Game-Theoretic Deception Mechanism". 2017 IFIP/IEEE International Symposium on Integrated Network Management (IM2017)

[48] Chen, Z., Wei, S., Yu, W., Nguyen, J.H. & Hatcher, W.G. (2018) "A cloud/edge computing streaming system for network traffic monitoring and threat detection", Int. J. Security and Networks, Vol. 13, No. 3, pp.169–186

[49] Baykara M. & Das R. (2018) "A novel honeypot based security approach for real-time intrusion detection and prevention systems". Journal of Information Security and Applications 41 (2018) 103–116.

[50] Almomani A., Alauthman M., Albalas F., Dorgham O. & Obeidat A. (2018) "An Online Intrusion Detection System to Cloud Computing Based on Neucube Algorithms". International Journal of Cloud Applications and Computing Volume 8 • Issue 2 • April-June 2018

[51] Shrivastav S. & Dhawan G. (2018) "Detection of Intrusion Detection System in Cloud Using Artificial Intelligence". International Journal of Advanced Computronics and Management Studies (IJACMS), Volume 3, Issue 2, April-May - 2018, pp.43-50

[52] Hajimirzaei, B. & Navimipour, N.J. (2018) "Intrusion detection for cloud computing using neural networks and artificial bee colony optimization algorithm". ICT Express (2018). DoI: https://doi.org/10.1016/j.icte.2018.01.014

[53] Aldribi A., Traore I., Moa B. & Nwamuo O. (2019) "Hypervisor-based cloud intrusion detection through online multivariate statistical change tracking". Computers & Security 88 (2020) 101646. DoI: https://doi.org/10.1016/j.cose.2019.101646

[54] Qasem M. & Almohri M. J. H. (2019) "An efficient deception architecture for cloud-based virtual networks". Kuwait J. Sci. 46 (3) pp. 40-52, 2019

[55] Jelidi M., Ghourabi A. & Gasmi K. (2019) "A Hybrid Intrusion Detection System for Cloud Computing Environments". 978-1-5386-8125-1/19/$31.00 ©2019 IEEE

[56] Li Y., Shi L. & Feng H. (2019) "A Game-Theoretic Analysis for Distributed Honeypots". Future Internet 2019, 11, 65; doi:10.3390/fi11030065

[57] Leo, E. W. (2015), "Python Hacking Essentials", Copyright by Earnest Wish Leo, 1-265.

[58] Seitz, J. (2017), "Black Hat Python: Python Programming for Hackers and Pentesters", No Starch Press, 1-193.

[59] Tale, S. (2017), "Hacking with Python: The Ultimate Beginner's Guide", Copyright by Steve Tale, 1-94

[60] Sinha, S. (2017), "Beginning Ethical Hacking with Python", NY: APress, Springer Science+Business Media.