big data and cognitive computing

MDPI

Article

Hardening the Security of Multi-Access Edge Computing through Bio-Inspired VM Introspection

Huseyn Huseynov [1,*], Tarek Saadawi [1] and Kenichi Kourai [2]

[1] Department of Electrical Engineering, City University of New York, City College, New York, NY 10031, USA; saadawi@ccny.cuny.edu
[2] Department of Computer Science and Networks, Kyushu Institute of Technology, Fukuoka 820-8502, Japan; kourai@csn.kyutech.ac.jp
* Correspondence: hhuseynov@ccny.cuny.edu

Abstract: The extreme bandwidth and performance of 5G mobile networks changes the way we develop and utilize digital services. Within a few years, 5G will not only touch technology and applications, but dramatically change the economy, our society and individual life. One of the emerging technologies that enables the evolution to 5G by bringing cloud capabilities near to the end users is *Edge Computing* or also known as *Multi-Access Edge Computing (MEC)* that will become pertinent towards the evolution of 5G. This evolution also entails growth in the threat landscape and increase privacy in concerns at different application areas, hence security and privacy plays a central role in the evolution towards 5G. Since MEC application instantiated in the virtualized infrastructure, in this paper we present a distributed application that aims to constantly introspect multiple virtual machines (VMs) in order to detect malicious activities based on their anomalous behavior. Once suspicious processes detected, our IDS in real-time notifies system administrator about the potential threat. Developed software is able to detect keyloggers, rootkits, trojans, process hiding and other intrusion artifacts via agent-less operation, by operating remotely or directly from the host machine. Remote memory introspection means no software to install, no notice to malware to evacuate or destroy data. Experimental results of remote VMI on more than 50 different malicious code demonstrate average anomaly detection rate close to 97%. We have established wide testbed environment connecting networks of two universities Kyushu Institute of Technology and The City College of New York through secure GRE tunnel. Conducted experiments on this testbed deliver high response time of the proposed system.

Keywords: security and privacy; virtualization; intrusion detection; artificial immune system; virtual machine introspection; cloud security

Citation: Huseynov, H.; Saadawi, T.; Kourai, K. Hardening the Security of Multi-Access Edge Computing through Bio-Inspired VM Introspection. *Big Data Cogn. Comput.* **2021**, *5*, 52. https://doi.org/10.3390/bdcc5040052

Academic Editor: Min Chen

Received: 4 August 2021
Accepted: 23 September 2021
Published: 8 October 2021

1. Introduction

The main idea of the MEC initiative is bringing the clouds closer to the edge of the network as well as to the users. According to the estimation by Cisco Annual Internet Report [1], 5G devices and IoT connections will be over 10% of global mobile devices by 2023. Utilizing MEC architecture as a forefront for 5G mobile networks (Figure 1) enables technological advancement to both cloud service providers (CSPs) and to businesses. However, these remarkable benefits are not offered without cost [2]. Due to its decentralized computational architecture, resources in MEC environment expanded across different geographical regions [3]. Usage of edge servers and cloud computing poses a number of security risks in various areas such as in Application-Programming Interfaces (APIs), Virtualization and Containerization, Physical machines and others [4].

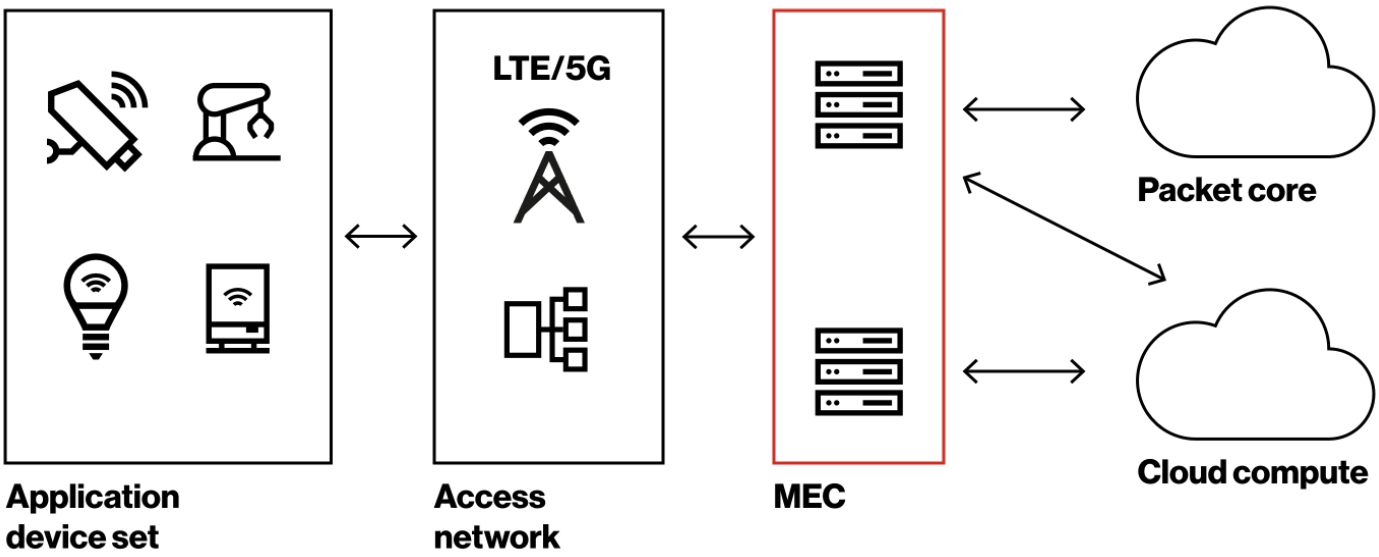

Figure 1. Multi-access Edge Computing (MEC) in 5G.

One of the most critical type of attacks in such a decentralized environment are *insider attacks*. In this context, insiders can be a CSP employees having access to the physical servers on which user data are stored. Mitigating the risk of potential insider attacks might require CSP to ensure well coordinated routine background checks for their employees, but this will not be enough if someone already uploaded a *keylogger* or installed a *rootkit*. Tackling these types of attacks requires "out of the box" intrusion detection system (IDS), which we are presenting in this paper by implementing remote virtual machine introspection (VMI). Proposed IDS operates outside of the infected VM, which lowers to zero chances of being compromised by any sophisticated malware.

In MEC, the adversary compromising Virtual Machines (VMs) are mostly malicious insiders that have administrative privileges or an application that operates with escalated privileges [4]. In this sort of situation, multiple set of attacks can be performed to the VM including attacks to communication links by continuous data eavesdropping between edge nodes and IoT devices [5]. These types of threats open up the affected VMs to numerous other potential attacks such as *logic bomb*, *trojans*, *spyware* and other malicious applications that could compromise the security of other data centers when the infected VM migrates to different physical location. Data encryption is a common mechanism that can be applied here to protect data confidentiality, but typically this cause a significant reduction in computational resources [6].

Proposed work is a real-time, artificial immune system (AIS) based intrusion detection and mitigation solution for MEC servers, which aims to provide autonomous security and constant virtual machine introspection through a high degree of detection accuracy. Developed IDS constantly introspects multiple VMs by tracking the events such as system calls, interrupts, memory reads/writes, number of open files, network activities and other logs. Built-in VMI tool—*KVMonitor*, remotely accesses VM memory and gathers data eliminating needs to install any software in VM. Remote VMI along with network traffic monitoring provides efficient malware protection mechanism against many new or existing attacks such as spyware, keylogger, worm, rootkit or trojan. Fully automated, real-time, IR-like discovery task directly into cloud fabric through volatile VM introspection. In this paper, we are demonstrating experimental results of our testbed that connects networks between Kyushu Institute of Technology and The City College of New York through secure GRE tunnel. This project considered as part of the worldwide development of an international networking and wireless platform by federating US research testbeds including COSMOS, ORBIT, FABRIC and PEERING with exploratory facilities in Ireland, Greece, Brazil, and Japan [7–10]. The global platform will enhance experimental research

on a wide range of optical, wireless, SDN and NFV, blockchain, inter-domain routing and edge computing experiments at a global scale. Conducted experiments deliver high response time of the proposed system and efficient detection rate.

This paper is organized as follows: Section 2 provides overview of related work in intrusion detection for MEC servers and security of Virtual Machines. Section 3 delivers a brief background on MEC and outlines potential threat vectors in this domain. Section 4 explains the artificial immune system based IDS and the way a negative selection algorithm (NSA) is applied in this work. Section 5 presents our intrusion detection system for MEC environment. Section 6 illustrates a comprehensive performance evaluation of the proposed security approach. Section 7 draws conclusion and discusses future work.

2. Related Work

With the increasing number of cyber security incidents, caused by the large attack surfaces, intrusion detection and mitigation becomes an even more important topic of research. Existing articles in Fog Computing and MEC security mainly focuses on analyzing the aspects of authentication, access control, intrusion detection, and user privacy. Dsouza et al. [11] first described the advantages of Fog Computing as a new paradigm and underlined the security issues in several scenarios, including Man-In-The-Middle (MITM) attacks and privacy issues. Pacheco, Benitez et al. [12] propose a methodology to develop an intrusion detection system based on anomaly behavior analysis to detect when a Fog node has been compromised. The crucial component of their architecture is Artificial Neural Networks (ANN), which requires the dataset of features and constant offline training. In [13] Evmorfos et al. suggested a technique that uses Long Short-Term Memory (LSTM) and Random Neural Networks (RNN) to tackle SYN flooding type attacks in large cloud-based networks. Their detection tactics includes capturing and saving network traffic as pcap files for further processing.

Since MEC application instantiated in the virtualized infrastructure of an MEC host, therefore, research in development of intrusion detection systems for Virtual Machines (VMs) is paramount in this domain. To prevent information leakage from virtual devices and tampering with their I/O data, Futagami et al. [14] proposed a nested virtualization application (VSBypass) that runs the entire virtualized system in an outer VM. Another paper, written by Inokuchi and Kourai [15], addresses user privacy in virtualized environment. They propose a strong user binding to VMs by decrypting its encrypted disk inside the trusted hypervisor. Sethi et al. [16] proposed Intrusion Detection System for cloud infrastructure based on Deep Reinforcement Learning. They have performed extensive experimentation using the benchmark *UNSW-NB15* dataset [17]. This publicly available dataset consists of normal traffic and nine types of attack traffic that includes DoS, DDoS, fuzzing, backdoor, analysis, exploit, worm, and shellcode. The limitation of this dataset is the lack of sufficiently many samples for some attack types.

Bio-inspired algorithms' adaptability allows many researchers and practitioners to utilize these techniques in solving many security-related cloud computing issues. In general, biologically inspired algorithms can be classified into four categories, such as *Evolutionary algorithms*, *Swarm algorithms*, *Immune algorithms*, and *Neural algorithms* [18]. Chiba et al. [19] describe network intrusion detection system in cloud environment by applying Back Propagation Neural Network and Genetic Algorithm. The anomaly detection technique builds a model from normal behavior and any deviation from the normal model is considered to be an outlier/attack. Obinna et al. [20] proposed a Denial-of-Service attack detection based on Artificial Immune System. Their method is based on implementation of negative selection algorithm that allows us to classify data as self-nonself, providing clear distinction between normal profile and abnormal network activity (i.e., DoS attack).

Some of the existing works on cloud-based IDS and VM introspection were used publicly available datasets. However, these datasets were not designed for Software Defined Networks (SDN). Many existing datasets described here were created by recording and processing pcap files utilizing different tools. Some other listed works focused on

application of bio-inspired algorithms only in detection network based attacks through local VM introspection. Therefore, there is a high possibility that those systems will not be able to detect anomalies in already compromised MEC servers. An important idea of remote VMI presented in this work is that IDS residing in a client machine is able to access multiple VMs running on different hosts across the world. Therefore, the proposed system can introspect many VMs and at the same time cannot be compromised by malicious application launched on remote host or guest machines.

3. Security Threats in Multi-Access Edge Computing

MEC technology aims to reduce communication latency between IoT devices and centralized cloud by storing data at the edge of network rather than at some distant data centers [21]. The key aspects of MEC implementation will be massive bandwidth, compute and storage availability at remote locations, reduced latency, minimized network traffic and less compute on the device. Shifting computing requirements from devices to the edge nodes at MEC will reduce energy consumption and deliver new devices, such as portable augmented/virtual reality (AR/VR) headsets. In addition, organizations will benefit from MEC by developing a scalable and efficient IoT capabilities known as Mobile Internet of Things (MIoT) [22].

MEC architecture is designed for optimal softwarization of functions and efficient infrastructure utilization. As shown in Figure 2, all MEC applications and application platform services are software applications running on hardware components that consist of multiple virtual machines. This design allows us to lower the cost of hardware components by combining off-the-shell elements with function virtualization. For example, the MEC virtualization manager layer shown in Figure 2 provides *Infrastructure as a Service (IaaS)* facilities, which will provision for flexible and efficient multi-tenancy, run-time and hosting environment for MEC application platform services.

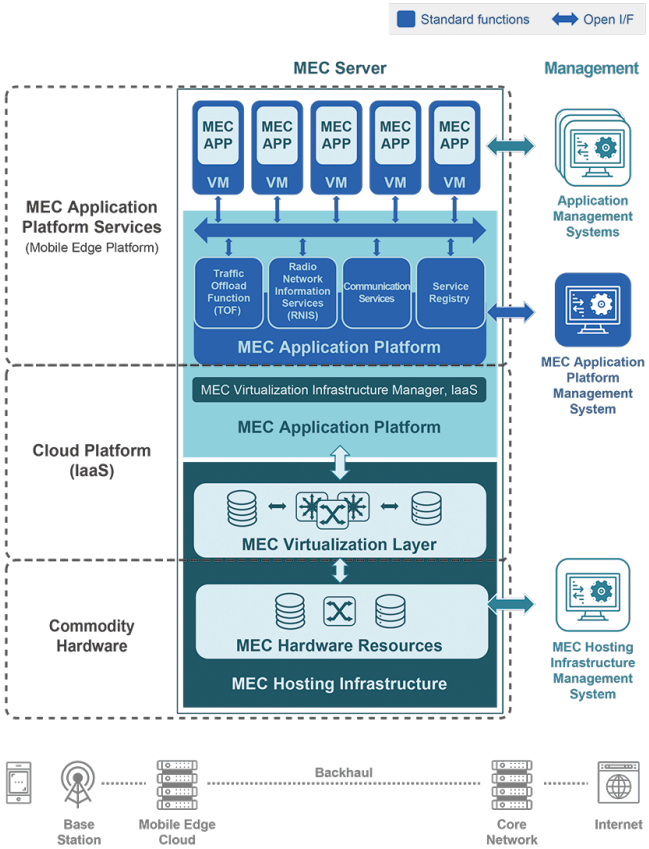

Figure 2. MEC server platform [23].

Compared to conventional on-premises data centers, the diversity of MEC architecture and its distributed nature awakes different types of vulnerabilities and privacy concerns. We have classified these threats into three main categories based on attack vectors: virtualization, application and network based threats. Table 1 illustrates a summary of categorized data security challenges in edge computing.

Table 1. Categorization of attack vectors in edge computing paradigm.

Asset	Virtualization	Application	Networking
Challenges	• VM Escape • Hyperjacking • VM Manipulation • Malicious Insiders • Zero-day Exploit	• Data Tampering • Privacy Leakage • API Abuse • Security Misconfiguration • Zero-day Exploit • Injection	• Denial of Service • Man-in-the-Middle (MitM) • Spoofing • Zero-day Exploit • Eavesdropping

- **Virtualization threats.** Virtualization is an integral attribute of MEC architecture. The hypervisor being a controlled mechanism of virtualization layer, controls access to hardware resources. This allows each VM to run on a shared hardware and at the same time to hide the presence of others. While virtual machines provides an isolated secure environment there are several types of vulnerabilities they can be exposed to. If an attacker succeeds in taking control of the hypervisor either being a malicious insider or through injection of a fraudulent hypervisor then this considered as a *hyperjacking attack*. Spurious hypervisor will manage the entire system and regular security measures will be ineffective in detecting this adversary. *VM escape* is another threat factor when the process is "breaking out" of the VM where it is running and gets access to the host machine [24]. MEC infrastructure can be affected by *VM manipulation* attack, where software with escalated privileges or a malicious insider can take control of VM and perform necessary modifications. In addition, arbitrary container access manipulation can lead to a control takeover attack on the container, and there is a possibility of data manipulation or data leakage through open API vulnerabilities in MEC applications.
- **Application threats.** Third party applications running in MEC servers can pose a fatal security threats by exposing virtual machines to different malicious applications. With current AI-driven technologies cybercriminals can develop more sophisticated malware to perform *data-tampering attacks* by injecting a malevolent client. *Injection attacks* occur when malicious code is embedded into unsecured software. SQL injections and XSS (cross-site scripting) are well known examples of this type attack. Attacks targeting hypervisor software or container engines can lead to data leakage within MEC applications [21]. Often, indirectly launched attacks through remotely controlled software or other malware-infected applications can spread to many MEC applications. *Keyloggers*, *rootkits*, *spyware/adware*, *worms*, *ransomware*, *trojans* and other villainous threats poses potential risks for virtual machines. Often hackers can exploit the vulnerability in software before developers can detect it, that exploit becomes known as a *zero-day attack* and all three layers in Table 1 can be vulnerable to zero-day attack.
- **Networking threats.** *Denial of Service (DoS) attack* is one of the most common and age-long threats in network infrastructure. DoS can shut down a machine or network by flooding the target with traffic, or sending it information that triggers a crash [20]. An additional type of DoS attack is the Distributed Denial of Service (DDoS) attack. A DDoS happens when multiple systems orchestrate a synchronized DoS attack to a single target. The essential difference is that instead of being attacked from one location, the target is attacked from multiple locations at once. In MEC systems, DoS attacks can carry only limited damage to network as described in [4]. Consequently, localized architecture of edge data centers prevents major damage to its core components. MEC systems are also vulnerable to *Man-in-the-Middle (MitM)* attacks, when a hacker or malicious agent intercepts and alters communication of two or more parties while they believe that

they are communicating with each other directly [25,26]. Software Defined Network (SDN) opens up an avenue for *DNS spoofing* attacks, when adversary divert traffic to an IP address other than where it was originally directed. Therefore, since the MEC architecture relies on virtualization, this type of attacks can disrupt not only multiple connected VMs but also affect all other elements of infrastructure.

Proposed work mainly focuses on detecting security threats in MECs from application and networking perspective. Our approach is based on implementation of remote virtual machine introspection (VMI) followed by applying screening through artificial immune system (AIS) algorithms. Inspired by theoretical immunology, AIS found its application in various Computer Science problems [27]. Utilizing evolutionary algorithms provide efficient anomaly detection mechanism through continuous training on normal models and producing a set of patterns (detectors).

4. Artificial Immune System Based Intrusion Detection

IDSes are usually classified by their approach of detecting attacks. Two main categories are signature-based detection and anomaly-based detection. One of the largest drawbacks of signature-based IDSes is that they mainly rely on signature database in order to detect attacks. Therefore, they may not recognize a new type of attack if its not listed in their database. On the other hand, anomaly-based IDSses typically work by taking into account a baseline of the normal traffic and any deviation from the normal considered as a threat [4]. Conversely, there can be a large number of false positives from anomaly-based IDSes compared to signature-based detection techniques. Therefore, efficiency of anomaly-based IDSes depends on multiple factors such as types of implemented algorithms, targeted systems, amount of generated input data, type of selected features, application complexity, response time and so on.

Artificial Immune System (AIS) is an area of artificial intelligence that focuses on algorithms abstracted from the models that exist in immunology [28]. These computational models of algorithms inspired by the principles of human immune system (HIS) and have characteristics of learning, adaptation, self-organization, memory and scalability. By imitating HIS these algorithms have developed as an effective solution for scientific computing and engineering applications [27]. Among various applications are data mining, pattern recognition, anomaly detection, predictive analytics, industrial control systems and IoT.

4.1. Negative Selection Algorithm

There are several class of algorithms inspired by various immunological theories. The most common of them are Clonal Selection Algorithm, Immune Network Algorithm, Negative Selection Algorithm (NSA) and Dendritic Cell Algorithm. This class of algorithms are generally used for classification and pattern recognition problems, where the problem space is modeled based on available knowledge. NSA is inspired by positive (self) and negative (nonself) selection process that resembles analogy of the human immune system (HIS). Forest et al. [29] describe initial steps of NSA as randomly generated detectors, such as B cells in HIS. These detectors later in the process are used to match with incoming set of data for anomaly detection. Many alternatives of Negative Selection Algorithm have been developed since the original version was first introduced [29], but despite this the original NSA is still popular. The main idea in NSA is that given shape-space U is divided into two sets: a self set S and a nonself set N, as shown below

$$U = S \cup N \; and \; S \cap N = \emptyset \tag{1}$$

NSA consist of two phases: *detector generation* phase and *nonself detection* phase. The generation of detectors Figure 3 involves screening the entire system to obtain its normal profile. This considered one of the challenges in NSA because self elements do not remain unchanged through the whole time. Therefore, continuous learning and building a self pro-

file is an important factor of the algorithm. Once the normal profile is obtained, we utilize Genetic Algorithm to generate candidate detectors that differ from normal set [27]. Nonself detection phase is a separate process that constantly utilizes previously built set of detectors to determine potential anomaly. Algorithm 1 illustrates a pseudocode of a basic negative selection algorithm. A detector is defined as $d = (C, r_d)$, where $C = \{c_1, c_2, ..., c_m\}, c_i \in \mathbb{R}$, is an m-dimensional point that corresponds to the center of a unit hyper-sphere with $r_d \in \mathbb{R}$ as its unit radius. For the generic NSA shown in Algorithm 1, $r_d = r_s$.

Algorithm 1 A Generic Negative Selection Algorithm.

function GENERICNSA(S, T_{max}, r_s)
▷ Where S - set of normal/self profiles, T_{max} - max. number of detectors, r_s - matching threshold.
    $D \leftarrow \emptyset$
    while $|D| < T_{max}$ **do**
        Generate a random detector (d)
        if d does not match any element in S **then**
            $D \leftarrow D \cup d$
        end if
    end while
    for All new incoming samples $\nu \in \cup$ **do**
        if ν matches any element in D **then**
            Classify ν as a nonself sample
        end if
    end for
    return D
end function

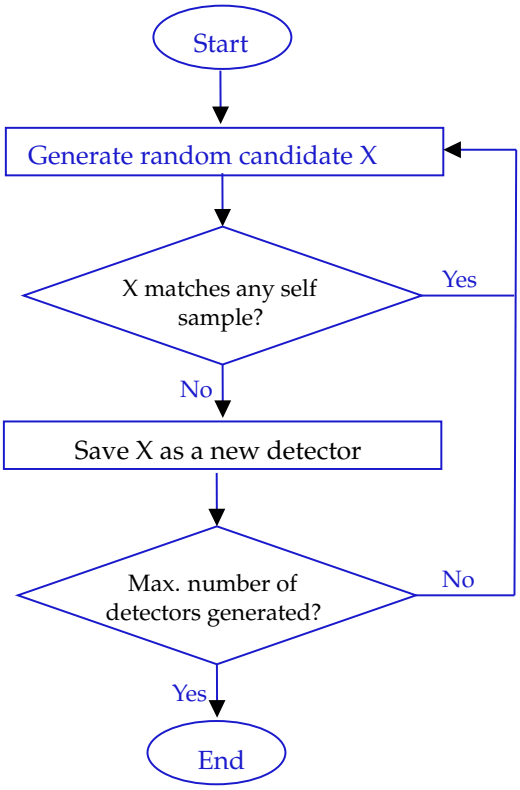

Figure 3. NSA detector generation phase.

A block-diagram in Figure 4 illustrates two NSA phases. During the detectors generation process any candidates that matched with self samples are removed. Once the number of obtained detectors is sufficient the algorithm terminates generation process [27].

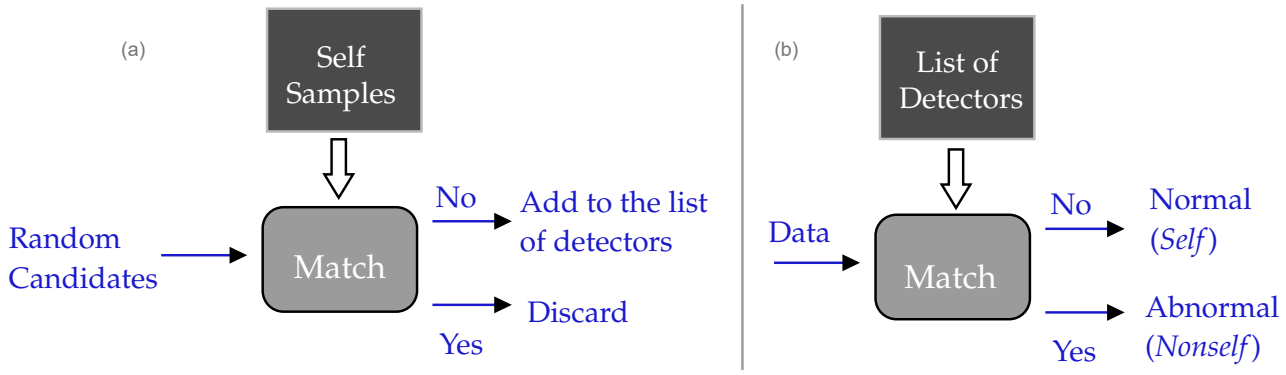

Figure 4. Detector generation process on the left (**a**) and nonself detection on the right (**b**).

The distance between detectors and self features was calculated using *Squared (Euclidean) distance*. This can also be derived using any real valued distance measures.

$$d(c,x) = \sum_{i=1}^{m} (c_i - x_i)^2 \tag{2}$$

In proposed work, we are constantly applying NSA to the data obtained from a VM through virtual machine introspection. If at any point IDS detects a match then this counts as a potential anomaly. Given a set S that has a subset of $[0,1]^m$, we can describe a feature vector as $x = (x_1, x_2, ..., x_m)$ in $[0,1]^m$. By employing Genetic Algorithm initial set of candidates are being generated randomly and called candidate detectors. These detectors later in the process evolve to cover more areas around the self set. This happens during each iteration, when the radius of each detector is calculated as $r_d = dValue - r_s$, where r_s is the variable distance around a self, Figure 5.

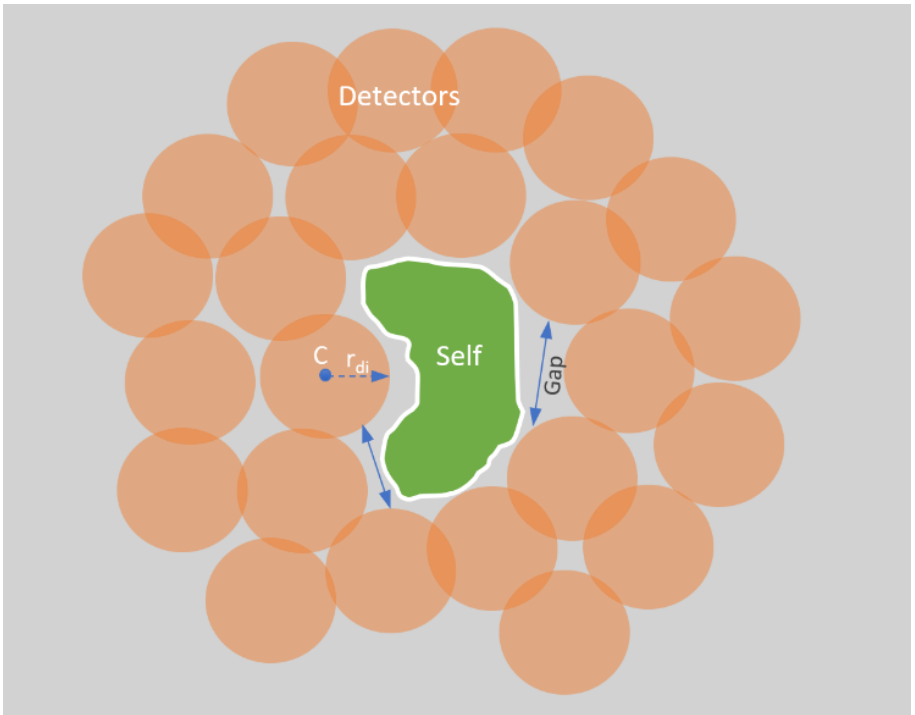

Figure 5. Self and nonself regions during detector generation process.

During an iterative process, detectors are moved away from self data and the other generated detectors. Depending on coverage, to eliminate the gap between self and nonself data, different sized detectors were produced and evaluated on each generation. A clone of detector is generated by moving center of the original detector by a fixed distance to its proximity. In addition, new random detectors are introduced to explore new areas of the nonself space. The overlap between two detectors are also computed in terms of the distance $dValue$ between their centers and radii. The detector generation process terminates when a set of mature detectors evolved that can provide significant coverage of nonself space. Figure 6 shows the flow diagram of the *Genetic Algorithm* process for generating variable-sized negative detectors.

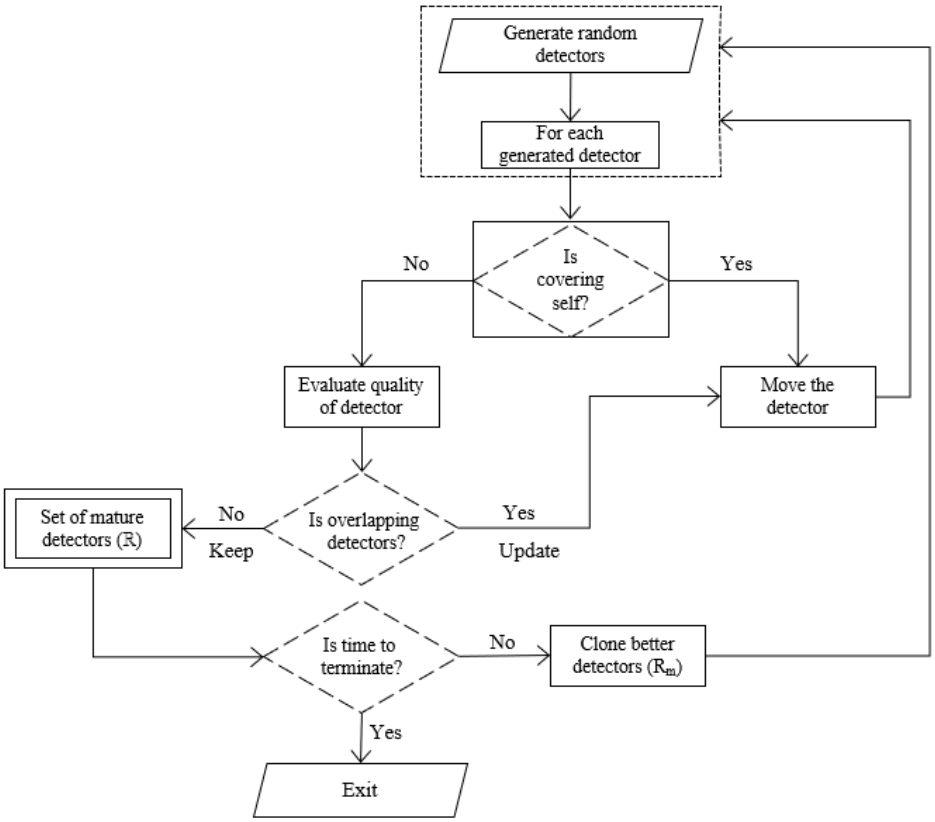

Figure 6. Genetic Algorithm for variable-sized detector generation process.

In the detection stage, the list of stored detectors are used to check whether new incoming samples correspond to self or nonself instances. If an input sample matches a detector, then it is identified as part of nonself, which refers that anomaly/change has occurred (see Figure 4b).

5. Proposed Security Approach

Proposed IDS provides security in MEC environments through automated, bio-inspired analysis of network flows, VM system calls and memory readings. Comprehensive intrusion detection process is based on two main components: *KVMonitor*—a lightweight and secure VMI module that gathers data from virtual machines and Artificial Immune System based IDS that employs *KVMonitor* and performs analysis of collected data. Composed information is being constantly compared to the list of detectors, any match directly reported as anomaly.

First step of provided approach lies on detailed study of many different malicious applications in order to discover what features of the system have been affected. Every process, whether this is a normal system application or hidden malicious script, creates a fingerprint in the system by constantly changing several system-wide parameters. As

a simple example, every running process has some memory consumption that can be tracked. At the same time, we can count how many times a particular process accessed the network, communicated to any drivers, how many child processes it has and so on. Table 2 illustrates eight different features and their values for four normal and abnormal processes running in Linux based VM. First column represent process ID numbers (PID) of running processes. All other columns are system/network based features: system calls *Write()* and *Read()*, *RssFile*—size of resident file mappings, *Open Files*—number of open files, *Socket*—number of attached file descriptors, amount of *TCP* and *UDP* connections followed by a system call *Send()*—number of bytes sent by the process.

Table 2. Example of raw features for normal (green) and abnormal (red) processes.

PID	Write	Read	RssFile	Open Files	Sockets	TCP	UDP	Send
237	0	18,472,960	6788	167	126	0	0	0
1124	0	86,016	716	8	3	0	0	0
3618	0	233,472	5396	33	8	0	1	0
578	24,576	35,786,752	30,084	40	23	0	0	0
1344	4096	0	0	3	0	6	0	0
1876	57,344	36,864	0	2	0	0	0	32,768
3151	32,768	16,384	0	5	0	8	12	12,288
895	24,576	8192	0	4	0	4	3	8192

A person can differentiate processes listed in Table 2 without knowledge of normal/abnormal, only based on represented values. We automated this process using *Negative Selection Algorithm* (NSA) giving prior knowledge about benign and malignant behavior. To generate detectors, features are being converted to the binary tuples in accordance with predetermined string matching rules [27].

Algorithm 2 demonstrates converting process for values of system call *Write()* to the binary form. Depending on the feature, conversion process can be limited until the value reaches certain defined constant. Applying similar transformation rules for every other feature, final representation of binary form will be as shown on Table 3. Feature column on this table is a concatenation of all features after converting them to the binary form. Lines colored red are detectors since they belong to abnormal processes. On the contrary, green lines are the features that belong to normal processes. In this example, length of detectors $l = 20$ and $PID = 1876$ (nonself) consists of the following tuples: $C_1[1110], C_2[1001], C_3[0], C_4[011], C_5[00], C_6[0], C_7[0], C_8[1110]$. For any $C_i[s]$ and partial matching threshold r:

$$1 \leq i \leq (l - r + 1)$$

Algorithm 2 Converting Value of System Call *Write()* to the Binary Form.

```
const C = 15
var binary_value = 0000
var X = WriteSystemCall mod 4096
if X ⩽ C then
    binary_value = convertToBinary(X)
end if
return binary_value
```

Table 3. Example of self (green) and nonself (red) features in binary format.

PID	Feature
237	00000000100011000000
1124	00000000100011000000
3618	00000000100011010000
578	01100000100011000000
1344	00010000001100101000
1876	11101001001100001110
3151	10000100010100110011
895	01100010010000110010

Having a list of detectors and given a collection of self-strings S as an input along with a matching rule and a partial matching threshold, present algorithm is able to recursively compare all the bits on each tuple and return whether a match occurs or not—*nonself detection phase*, Figure 4b.

To generate binary detectors, in other words, a set of features that correspond to abnormal processes, *DEAP (Distributed Evolutionary Algorithms in Python)* package has been utilized in the proposed application [30,31].

The overall architecture of proposed application shown in Figure 7 implements security in MEC environment through remote analysis of VM's network and obtaining system level process information. Using *Evolutionary Algorithm* within the *DEAP* framework and feeding it in advance with a set of normal features, application generates sufficient amount of detectors. During this process, Squared (Euclidean) distance (Equation (2)) is implemented as a fitness function to measure the distance between self and randomly generated nonself features.

KVMonitor is a pivotal component of proposed IDS that provides continuous and remote VMI. VM memory introspection is performed through accessing a memory file withing the host machine without any interruption to running system. By executing cr3 command using QEMU monitor protocol (QMP), KVMonitor connects to QEMU-KVM and then can send necessary QMP commands as shown in Figure 8 [32]. Through the KVMonitor proposed IDS can also introspect network of a VM by constantly capturing packets from virtual NICs. To introspect virtual disk, *KVMonitor* links an available loopback device (e.g., /dev/loop0) to a disk image by the *losetup* command, followed by producing device maps and mounting them in a similar way to a disk image with qcow2 [32].

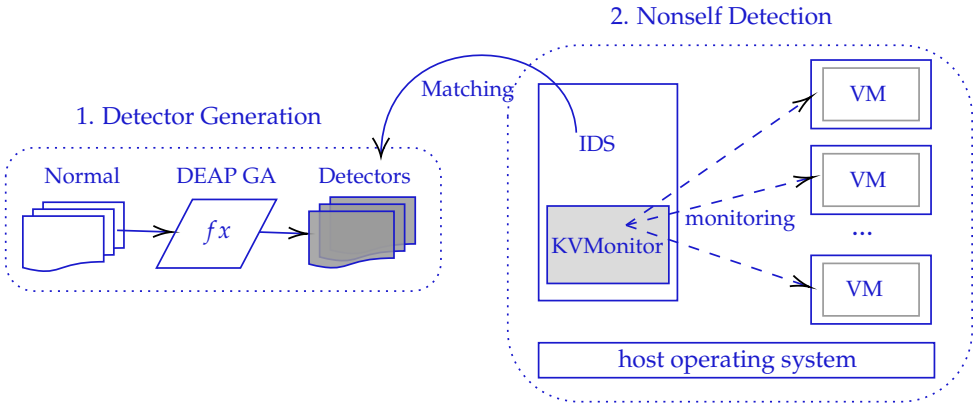

Figure 7. MEC-based Security Solution Architecture.

```
{  "execute":   "cr3"  }
{  "return":   {  "CR3":   "0x000000001f96e000"  }  }

{  "execute":   "xaddr",
   "arguments":   {  "addr":   "0xffffffff814a8340"  }  }
{  "return":   {  "paddr":   "0x00000000014a8340"  }  }
```

Figure 8. Example of execution CR3 and XADDR commands using QMP [32].

5.1. Detector Generation Using Genetic Algorithm

A detection rule considered sufficient if it is not covering any positive samples (the self features) and it covers a large area of nonself space. Considering self-space S as a subset of $[0,1]^n$ and a feature vector $x = (x_1, x_2, ..., x_n)$ in $[0,1]^n$, then a detector can be represented as a "detector rule" in the form

$$R_i : \textbf{\textit{if}}\ cond_i\ \textbf{\textit{then}}\ nonself,\ for\ i = 1, ..., m$$

where $cond_i = X_1 \in [low_1^i, high_i^i]$ and ... and $X_n \in [low_n^i, high_n^i]$. Here m is the number of detection rules and n the dimension of the Euclidean space [27]. Pseudocode for the Genetic Algorithm on generating detectors illustrated in Algorithm 3. In this approach we assume that all detectors had the same shape and size; particularly, hyper-spheres of a fixed size radius r in an n-dimensional space and size. The fitness function for a rule R defined as

$$fitness(R) = volume(R) - \alpha \times numberOfSelfSamples(R), where\ volume(R) = \left(\frac{2r}{\sqrt{n}}\right)$$

Algorithm 3 Genetic Algorithm for Generating Detectors.

```
Initialize population by selecting random individuals from the space S
for i = numberOfGenerations do
    for j = populationSize/2 do
        select two individuals (with uniform probability) as parent1 and parent2
        apply crossover to generate an offspring (child)
        mutate child
        ▷ calculate distances between child and both parents
        d1 = dist(child, parent1)
        d2 = dist(child, parent2)
        ▷ calculate fitness values for child and both parents
        fc = fitness(child)
        fp1 = fitness(parent1)
        fp2 = fitness(parent2)
        if (d1 < d2)  &  (fc > fp1) then
            replace parent1 with child
        else if (d2 ⩽ d1)  &  (fc > fp2) then
            replace parent2 with child
        end if
    end for
end for
return best (highly-fitted) individuals
```

Volume of the effective coverage of a detector was approximated as the volume of the inscribed hypercube. The parameter α denotes a coefficient of sensitivity, which for a specific rule determines the trade-off between the volume covered by it and its interception

with the self-set. Therefore, fitness is calculated as a sum of the fitness values of all evolved rules minus the overlaps between hypercubes defined by the rules.

5.2. Nonself Detection

List of generated detectors is integral part of nonself detection phase. Results of constant VM introspection are passed to the IDS as a hash table consisting process ids as keys and features as values. Application on the fly starts matching of received data with the list of nonself detectors. Simplified version of matching algorithm is shown below (Algorithm 4).

Algorithm 4 Matching Function in Nonself Detection Phase.

```
1:  function MATCHDETECTORS(F, D, r_s)
    ▷ Where F - set of features received from KVMonitor as tuples (key=pid, value=feature),
    D - list of generated detectors, r_s - matching threshold.
2:      result ← ∅
3:      for detectors ∈ D do
4:          for features ∈ F do
5:              x = (detectors ∩ features.value)
6:              if len(x) ⩾ r_s & features.key ∉ Result then
7:                  result.append(features.key, features.value)
8:              end if
9:          end for
10:     end for
11:     return result
12: end function
```

Algorithm compares set of incoming features with the set of nonself detectors by taking intersection between two sets. Result of intersection x holds list of features being matched. If the length of matched features are equal or greater than the matching threshold r_s then it is being added to the resulting dictionary. By introducing matching threshold as an argument, we create flexibility for users to manually increase or decrease detection accuracy. For instance, setting it lower than half number of overall features, increases amount of false positives returned by application. This is useful to observe how certain normal processes are close to abnormal based on their anomalous behavior.

6. Experimental Evaluation

In this section, we provide an experimental evaluation of the proposed security approach using simulation environment similar to SDN-managed IoT network. We performed experiments to evaluate the distributed intrusion detection and mitigation model in terms of its effect on VM system and network parameters during various attack scenarios. The experiments were conducted on a client machine with Intel Core i7-8750H @ 2.20 GHz processor and 16 GB RAM to introspect remote VM with Intel Xeon Silver 4114 Processor @ 2.20 GHz and 8 GB RAM running on Ubuntu 18.04 LTS based remote host with the same processor on 8 cores and 131 GB RAM.

For deployment of the proposed IDS, the testbed setup illustrated in Figure 9 was used. Secure GRE tunnel was established between the labs of two universities Kyushu Institute of Technology in Japan, where the host OS and VM have been deployed, and The City College of New York, where the client machine with IDS has been launched. The maximum bandwidth of each link in the network was limited to 100 Mb per second. Modern CISCO Gigabit Smart Switch with 50 ports has been used to manage the network.

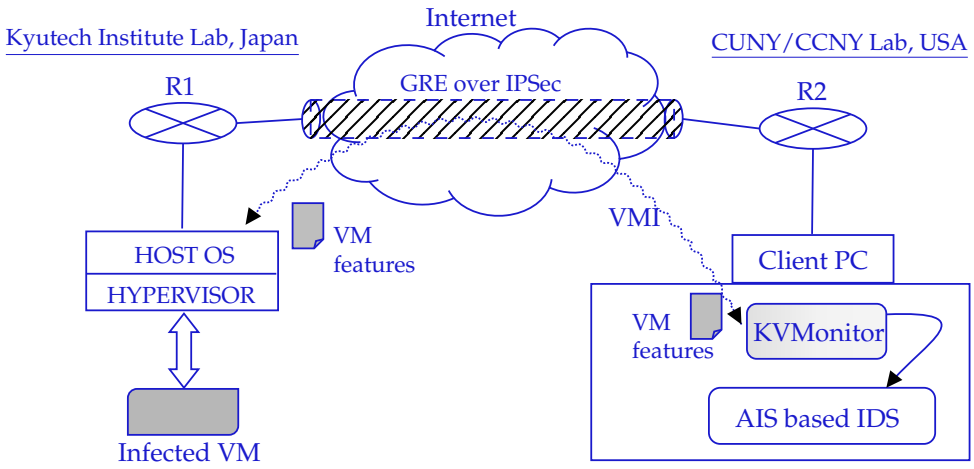

Figure 9. Testbed environment.

To evaluate the detection rate and system/network performance we conducted experiments with more than 50 different malicious applications: trojans, adware, worms, keyloggers, spyware, hyperjacking attack using preinstalled rootkit on a VM. In this paper we provide results of experiments conducted on four different open-source malicious applications listed in Table 4.

Table 4. Malware used in experiments and detection results.

Malware	F1 Score (%)	Description
Logkeys	99	Multi functional GNU/Linux keylogger. Logs all common character and function keys [33].
Rootkit	94	Linux based rootkit that listens to certain ports, gives escalated privileges to user, has built-in keylogger and able to hide from common scanners [34].
Stitch	97	Cross-platform keylogger with remote administrative tool. User can select payload to bind into specific IP and port, listens for a connection on that port, has option to send an email of system info when the system boots, and option to start keylogger on boot [35].
TrojanX	94	Basic Trojan application written in Swift with minimal GUI client on Mac OS. Uses shell scripts to set up the system to use SOCKS proxy [36].

Proposed application successfully detected all listed vulnerabilities and classified every flow entry based on triggered features. VM introspection time for *KVMonitor* in this experiment was set for 10 s. Within this time frame intrusion detection application periodically received data from *KVMonitor* and performed nonself detection process. In case if anomaly is detected, application triggered push notifications and sent email with detailed information about potential vulnerability. Figure 10 shows the action upon detection of *Logkeys* keylogger [33] running on remote VM. Depending on different threat models the output information varies.

```
VM is connected. keyboard device: /dev/input/event1
*Potential Threat Detected:
 Process ID: 1913 (user-space keylogger)

Elapsed time: 0.59375 sec.
```

Figure 10. Detection of Logkeys Keylogger.

In the following subsections, performance measurements and detection rate of our IDS system are reported.

6.1. Network Performance Results

The maximum available bandwidth of all the links between the switch and hosts in our network were set to 100 Mb per second. We have utilized *perfSONAR* toolkit (Performance Service-Oriented Network monitoring Architecture) [37], to measure, identify and isolate network problems in established testbed. Built-in *iPerf3* tool was used to measure the available bandwidth between two ends of GRE tunnel. The packet sending rate was 1000 packets per second and the payload of the packets was 1000 bytes.

We measured the feature retrieval time taken by *KVMonitor* from remote host machine with respect to data flow in the switch using our IDS application. Figure 11 corresponds to retrieval of 8 best features up to 20,000 flow entries. The feature collection of all features for 20,000 flow entries was 416.4 milliseconds, whereas it was 280 milliseconds for retrieving the 8 best features for the same number of flows. It is fairly low and does not affect the performance of the network.

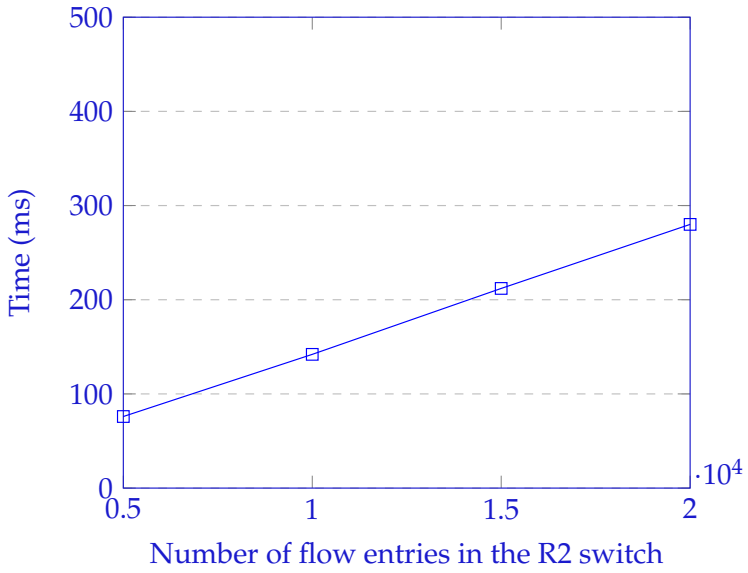

Figure 11. Feature retrieval and processing time of 8 best features up to 20,000 flow entries.

One of the important measurement we have conducted is determining the time during which application retrieves features from the VM. It is paramount to retrieve features quickly in order to detect potential attacks. It is also important that the process of retrieving features will not affect productivity on the client machine. Figure 12 shows the flow entry collection by *KVMonitor* up to 20,000 flow entries in the R2 switch and despite that IDS application collected features for all of the flows in 416.4 milliseconds, which does not cause much overhead for the application on client side.

We observe that the feature retrieval time increases linearly with the number of flow entries in the switch. However, our IDS performs feature processing on the fly and does not wait to finish every flow entry in the switch before taking action. Once data received, application calculates feature vector converting raw values into binary forms followed by classification and all takes 54 milliseconds when the switch has 1000 flow entries, shown on Figure 11.

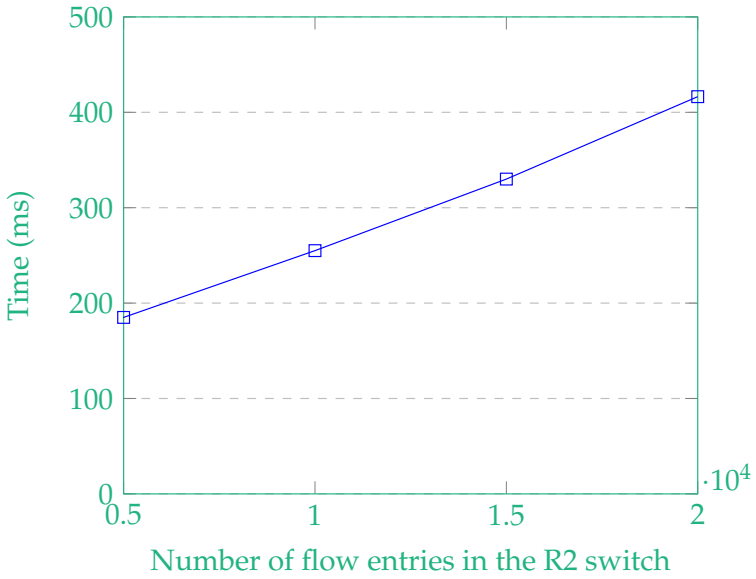

Figure 12. Feature retrieval and processing time of all features up to 20,000 flow entries.

6.2. Memory and CPU Measurements

Most of the listed vulnerabilities developed on purpose to not waste system resources and act quietly. Especially keyloggers and rootkits are behaving as unprivileged programs and surreptitiously eavesdropping all the keystrokes typed by the user. Performance comparison of *KVMonitor* with Xen described in [32,38] shows that *KVMonitor* was 48 times faster than Xen in accessing VM memory and during introspection. This is mainly because Xen is repeating mapping and unmapping for each memory page of VM. *KVMonitor*, in contrast, can map the whole memory at first and access it like heap memory [32]. Our remote host has eight cores and to measure average CPU utilization for different processes we used the Linux *iostat* and *top* commands. Under normal conditions, most of the time CPU usage of the host were 0%.

Figure 13 shows the CPU utilization on the host machine during the normal state and while KVMonitor accesses the VM memory file. Remote offloading was 8.5% slower than local due to network delay and additional booting time using VNC and SSH. During intervals of VM introspection, CPU of the host machine correlated around 2.8–3.4 %, which is normal because *KVMonitor* is only accessing QEMU-KVM memory file stored in the host in *qcow2* format. The way how KVMonitor acquire access to the VM is implemented by utilizing a cr3 command that comes with QEMU Monitor Protocol (QMP). The latter provides user-friendly JSON-based structure that enables access to certain functionality. This provides fast and reliable way to access memory of a virtual machine and extract necessary data [32].

Main processing power is balanced on the client machine, where IDS constantly receives raw data from *KVMonitor* and implements nonself detection phase. In this experiment, we used client machine with six cores and two threads per core. As shown on Figure 14 CPU utilization of feature retrieval on the client machine was around 35–45% during the period of active processing.

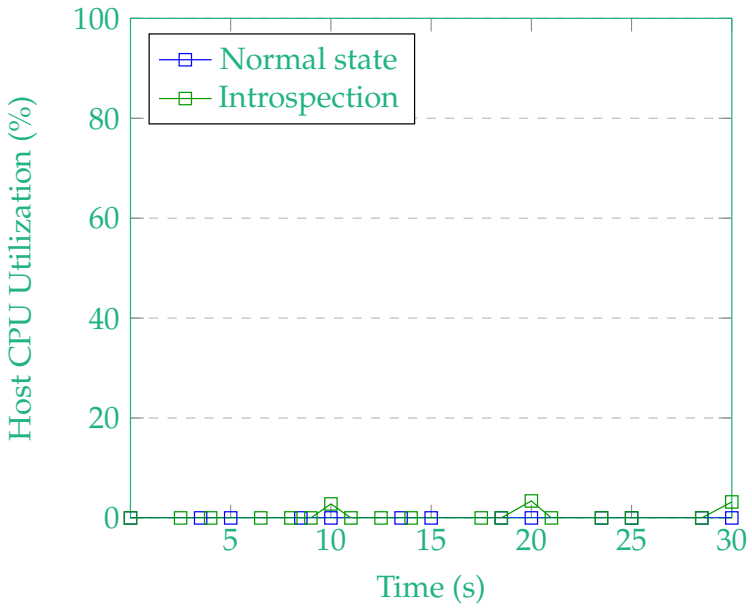

Figure 13. CPU utilization in the remote Host during normal state and VMI periods.

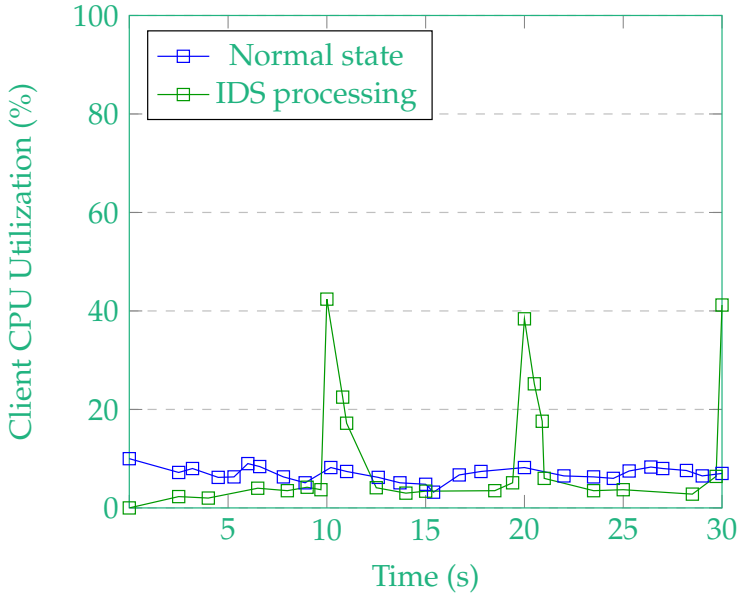

Figure 14. CPU utilization in the Client machine during normal state and IDS activity periods.

6.3. Detector Generation and Nonself Detection

For detector generation process we used set of 200 records—self samples, from different category as input to generate nonself records. Using Genetic Algorithm (GA) within Python DEAP framework [30] we generated close to 61,000 detectors, sample shown in Figure 15.

Application responsible for generating detectors utilize multiprocessing package that offers both local and remote concurrency, effectively side-stepping the Global Interpreter Lock by using sub-processes instead of threads [30]. This significantly reduces time of evolutionary algorithm, which takes on average 4–6 s to generate list of 61,000 detectors. Constant parameters for GA provided in Figure 16 represent size of generated detectors 24, initial population of random detectors 500, number of generations 200, amount of pool workers in multiprocessing 4 and constant memory page size 4096.

```
...
60884  000100000011001010010011
60885  111111011011010010001011
60886  000110110000000100101101
60887  101100010011000010110100
60888  001111010011110011011011
60889  011101011010110111110101
60890  011111110011011111100111
60891  100111010000110010110100
60892  010101110010100010101000
...
```

Figure 15. Sample of generated detectors as a result of Genetic Algorithm.

```
...
INDIVIDUAL_SIZE   24
POPULATION_SIZE   500
GENERATION_SIZE   200
POOL_WORKERS       4
MMAP_MIN_SIZE     4096
...
```

Figure 16. Genetic Algorithm parameter tuning for detectors generation.

The average F1 score (detection rate) of the nonself detection of listed in Table 4 malware was 96.86%. We divided experiments into two parts, first by exposing remote VM separately to each of the listed malicious applications and measuring performance along with the detection time. Second, we exposed remote VM to all four listed malware simultaneously and then launched our IDS. In both cases anomalies were detected with almost similar rate and IDS successfully responded on time. Figure 17 shows the results of detection rate with respect to each malicious application.

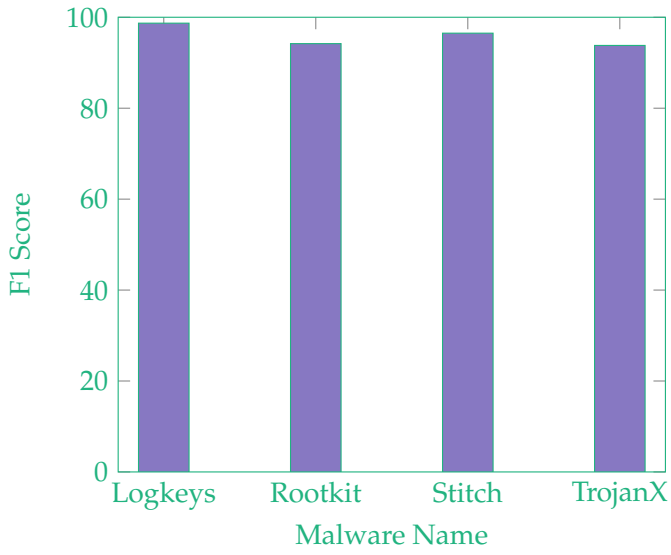

Figure 17. Detection rate of listed malware by utilizing all features.

Experimental results with only 8 primarily system-wide features after disabling network features (total number of sent packets, source/destination ports, TCP/UDP usage, and so on) speeds up detection process four times with average F1 score of 82.48%.

7. Conclusions

In this work, we proposed an automated, intelligent intrusion detection and mitigation approach for MEC servers, which aims to provide explainable security in the IoT networks of the 5G era. Proposed approach relies on Artificial Immune System based intrusion detection with built-in automated virtual machine introspection module. It has been successfully tested on remote VM over established GRE-based secure testbed between Kyushu Institute of Technology and The City College of New York. Presented results show performance evaluation as well as accuracy of intrusion detection on various types of malicious applications. Residing outside of the guest machine, described IDS cannot be subverted by any malware running in VM. This also provides significantly low performance impact on VM and the host machine.

The proposed security approach is promising for achieving real-time, highly accurate detection and mitigation of attacks in MEC-based servers, which will be in widespread use in the 5G and beyond era. Our future work will include an extension of current classification mechanism to more attack types and network topologies. We aim to run more experiments by increasing number of remote virtual machines within our testbed environment. Through continuous collaboration with different universities we plan to expand our testbed across the world creating cloud-based simulation environment with multiple availability zones. Continuous research on Linux kernel to explore more features will increase detection accuracy. In parallel, we continuously working on improving performance by developing more secure and reliable application.

Author Contributions: T.S. devised the project, developed the main conceptual ideas and proof outline. K.K. designed overall model and developed KVMonitor, the VMI module. T.S. and K.K. administered the project. The experiments were planned and separately implemented by H.H. and K.K. Software development of IDS and its integration with KVMonitor performed by H.H. T.S. and K.K. supervised H.H. in carrying out experimentation and numerical calculations. All authors have read and agreed to the published version of the manuscript

Funding: This research received no external funding.

Institutional Review Board Statement: Not applicable.

Informed Consent Statement: Not applicable.

Data Availability Statement: The data supporting the reported results in the present study will be available on request from the corresponding author or the first author.

Acknowledgments: This work is supported in part by NSF JUNO2 (Japan-US Network Opportunity 2) (Award No. 1818884) and NSF IRNC (Award No. 2029295).

Conflicts of Interest: The authors declare no conflict of interest.

References

1. *White Paper: Cisco Annual Internet Report (2018–2023)*; Technical Report C11-741490-01; 2020. Available online: https://www.cisco.com/c/en/us/solutions/collateral/executive-perspectives/annual-internet-report/white-paper-c11-741490.html (accessed on 14 September 2021).
2. Ahmad, I.; Kumar, T.; Liyanage, M.; Okwuibe, J.; Ylianttila, M.; Gurtov, A. Overview of 5G Security Challenges and Solutions. *IEEE Commun. Stand. Mag.* **2018**, *2*, 36–43. [CrossRef]
3. Ali, B.; Gregory, M.A.; Li, S. Multi-Access Edge Computing Architecture, Data Security and Privacy: A Review. *IEEE Access* **2021**, *9*, 18706–18721. [CrossRef]
4. Roman, R.; Lopez, J.; Mambo, M. Mobile edge computing, Fog et al.: A survey and analysis of security threats and challenges. *Future Gener. Comput. Syst.* **2018**, *78*, 680–698. [CrossRef]
5. Ranaweera, P.; Jurcut, A.D.; Liyanage, M. Realizing Multi-Access Edge Computing Feasibility: Security Perspective. In Proceedings of the 2019 IEEE Conference on Standards for Communications and Networking (CSCN), Granada, Spain, 28–30 October 2019; pp. 1–7. [CrossRef]

6. Khan Saad, S.P.; Qin, Y. Fog computing security: a review of current applications and security solutions. *J. Cloud Comput.* **2017**, *6*, 107593. [CrossRef]
7. Raychaudhuri, D.; Seskar, I.; Zussman, G.; Korakis, T.; Kilper, D.; Chen, T.; Kolodziejski, J.; Sherman, M.; Kostic, Z.; Gu, X.; et al. *Challenge: COSMOS: A City-Scale Programmable Testbed for Experimentation with Advanced Wireless*; MobiCom '20; Association for Computing Machinery: New York, NY, USA, 2020. [CrossRef]
8. Raychaudhuri, D.; Seskar, I.; Ott, M.; Ganu, S.; Ramachandran, K.; Kremo, H.; Siracusa, R.; Liu, H.; Singh, M. Overview of the ORBIT radio grid testbed for evaluation of next-generation wireless network protocols. In Proceedings of the IEEE Wireless Communications and Networking Conference, Trento, Italy, 23–25 February 2005; Volume 3, pp. 1664–1669. [CrossRef]
9. Baldin, I.; Nikolich, A.; Griffioen, J.; Monga, I.I.S.; Wang, K.C.; Lehman, T.; Ruth, P. FABRIC: A National-Scale Programmable Experimental Network Infrastructure. *IEEE Internet Comput.* **2019**, *23*, 38–47. [CrossRef]
10. Schlinker, B.; Arnold, T.; Cunha, I.; Katz-Bassett, E. PEERING: Virtualizing BGP at the Edge for Research. In Proceedings of the ACM CoNEXT, Orlando, FL, USA, 9–12 December 2019.
11. Dsouza, C.; Ahn, G.J.; Taguinod, M. Policy-driven security management for fog computing: Preliminary framework and a case study. In Proceedings of the 2014 IEEE 15th International Conference on Information Reuse and Integration (IEEE IRI 2014), Redwood City, CA, USA, 13–15 August 2014; pp. 16–23. [CrossRef]
12. Pacheco, J.; Benitez, V.H.; Tunc, C.; Grijalva, C. Anomaly Behavior Analysis for Fog Nodes Availability Assurance in IoT Applications. In Proceedings of the 2019 IEEE/ACS 16th International Conference on Computer Systems and Applications (AICCSA), Abu Dhabi, United Arab Emirates, 3–7 November 2019; pp. 1–6. [CrossRef]
13. Evmorfos, S.; Vlachodimitropoulos, G.; Bakalos, N.; Gelenbe, E. Neural Network Architectures for the Detection of SYN Flood Attacks in IoT Systems. In Proceedings of the 13th ACM International Conference on PErvasive Technologies Related to Assistive Environments, Corfu, Greece, 30 June–3 July 2020; Association for Computing Machinery: New York, NY, USA, 2020. [CrossRef]
14. Futagami, S.; Unoki, T.; Kourai, K. Secure Out-of-Band Remote Management of Virtual Machines with Transparent Passthrough. In Proceedings of the 34th Annual Computer Security Applications Conference, San Juan, PR, USA, 3–7 December 2018; Association for Computing Machinery: New York, NY, USA, 2018; pp. 430–440. [CrossRef]
15. Inokuchi, K.; Kourai, K. UVBond: Strong User Binding to VMs for Secure Remote Management in Semi-Trusted Clouds. In Proceedings of the 2018 IEEE/ACM 11th International Conference on Utility and Cloud Computing (UCC), Zurich, Switzerland, 17–20 December 2018; pp. 213–222. [CrossRef]
16. Sethi, K.; Kumar, R.; Prajapati, N.; Bera, P. Deep Reinforcement Learning based Intrusion Detection System for Cloud Infrastructure. In Proceedings of the 2020 International Conference on COMmunication Systems NETworkS (COMSNETS), Bangalore, India, 7–1 January 2020; pp. 1–6. [CrossRef]
17. The UNSW-NB15 Dataset. Available online: https://www.unsw.adfa.edu.au/unsw-canberra-cyber/cybersecurity/ADFA-NB15-Datasets/ (accessed on 20 February 2021).
18. Brownlee, J. Clever Algorithms: Nature-Inspired Programming Recipes, 1st ed.; Lulu.com: 2011. Available online: https://dl.acm.org/doi/book/10.5555/1983645 (accessed on 14 September 2021)
19. Chiba, Z. New Anomaly Network Intrusion Detection System in Cloud Environment Based on Optimized Back Propagation Neural Network Using Improved Genetic Algorithm. *Int. J. Commun. Netw. Inf. Secur. (IJCNIS)* **2019**, *11*, 61–84.
20. Igbe, O.; Ajayi, O.; Saadawi, T. Detecting Denial of Service Attacks Using a Combination of Dendritic Cell Algorithm and the Negative Selection Algorithm. In Proceedings of the 2017 IEEE International Conference on Smart Cloud (SmartCloud), New York, NY, USA, 3–5 November 2017; pp. 72–77. [CrossRef]
21. Kim, Y.; Park, J.G.; Lee, J.H. Security Threats in 5G Edge Computing Environments. In Proceedings of the 2020 International Conference on Information and Communication Technology Convergence (ICTC), Jeju Island, Korea, 21–23 October 2020; pp. 905–907. [CrossRef]
22. Alnahdi, A.; Liu, S.H. Mobile Internet of Things (MIoT) and Its Applications for Smart Environments: A Positional Overview. In Proceedings of the 2017 IEEE International Congress on Internet of Things (ICIOT), Honolulu, HI, USA, 25–30 June 2017.
23. 5G MEC Networking Platform by Gigabyte. Available online: https://www.gigabyte.com/Solutions/Networking/5g-imec-networking-platform (accessed on 5 February 2021).
24. CVE-2008-0923. U.S. National Vulnerability Database (NVD), CVE-ID CVE-2008-0923, 2008. Available online: https://nvd.nist.gov/vuln/detail/CVE-2008-0923 (accessed on 14 September 2021).
25. Stojmenovic, I.; Wen, S.; Huang, X.; Luan, H. An Overview of Fog Computing and Its Security Issues. *Concurr. Comput. Pract. Exp.* **2016**, *28*, 2991–3005. [CrossRef]
26. Zhang, L.; Jia, W.; Wen, S.; Yao, D. A Man-in-the-Middle Attack on 3G-WLAN Interworking. In Proceedings of the 2010 International Conference on Communications and Mobile Computing—Volume 01, Shenzhen, China, 12–14 April 2010; pp. 121–125. [CrossRef]
27. Dasgupta, D.; Nino, F. *Immunological Computation: Theory and Applications*, 1st ed.; Auerbach Publications: Boca Raton, FL, USA, 2008.
28. Igbe, O.; Saadawi, T.; Darwish, I. Digital Immune System for Intrusion Detection on Data Processing Systems and Networks, U.S. Patent 10,609,057, 31 March 2020.
29. Forest, S.; Perelson, A.; Allen, L.; Cherukuri, R. Self-Nonself Discrimination in a Computer. In Proceedings of the 1994 IEEE Computer Society Symposium on Research in Security and Privacy, Oakland, CA, USA, 16–18 May 1994; pp. 202–212.

30. DEAP. Distributed Evolutionary Algorithms in Python. Available online: https://github.com/deap/deap (accessed on 8 January 2021).
31. Fortin, F.A.; De Rainville, F.M.; Gardner, M.A.; Parizeau, M.; Gagné, C. DEAP: Evolutionary Algorithms Made Easy. *J. Mach. Learn. Res.* **2012**, *13*, 2171–2175.
32. Kourai, K.; Nakamura, K. Efficient VM Introspection in KVM and Performance Comparison with Xen. In Proceedings of the 2014 IEEE 20th Pacific Rim International Symposium on Dependable Computing, Singapore, 18–21 November 2014; pp. 192–202. [CrossRef]
33. Logkeys. GNU/Linux Keylogger. Available online: https://github.com/kernc/logkeys (accessed on 30 July 2021).
34. Multifunctional Linux Rootkit. Available online: https://github.com/maK-/maK_it-Linux-Rootkit (accessed on 30 July 2021).
35. Python Based Keylogger with Remote Administration Tool. Available online: https://github.com/nathanlopez/Stitch (accessed on 30 July 2021).
36. Minimal Swift Based Trojan Application Targeting Mac OS. Available online: https://github.com/RCD-Y/TrojanX (accessed on 30 July 2021).
37. Zurawski, J.; Balasubramanian, S.; Brown, A.; Kissel, E.; Lake, A.; Swany, M.; Tierney, B.; Zekauskas, M. perfSONAR: On-board diagnostics for big data. In Proceedings of the IEEE International Conference on Big Data, Silicon Valley, CA, USA, 6–9 October 2013.
38. Kourai, K.; Juda, K. Secure Offloading of Legacy IDSes Using Remote VM Introspection in Semi-trusted Clouds. In Proceedings of the 2016 IEEE 9th International Conference on Cloud Computing (CLOUD), San Francisco, CA, USA, 27 June–2 July 2016; pp. 43–50. [CrossRef]