Future Generation Computer Systems 79 (2018) 431–446

Contents lists available at ScienceDirect

Future Generation Computer Systems

journal homepage: www.elsevier.com/locate/fgcs

# Automated multi-level malware detection system based on reconstructed semantic view of executables using machine learning techniques at VMM

Ajay Kumara M.A. [*], Jaidhar C.D.

*Department of Information Technology, National Institute of Technology Karnataka, Surathkal, India*

A R T I C L E I N F O

*Article history:*
Received 3 October 2016
Received in revised form 30 April 2017
Accepted 5 June 2017
Available online 19 June 2017

*Keywords:*
Cyber–Physical system
Hypervisor
Feature selection technique
Machine learning
Memory forensic analysis
N-gram feature extraction
Virtual machine introspection

A B S T R A C T

In order to fulfill the requirements like stringent timing restraints and demand on resources, Cyber–Physical System (CPS) must deploy on the virtualized environment such as cloud computing. To protect Virtual Machines (VMs) in which CPSs are functioning against malware-based attacks, malware detection and mitigation technique is emerging as a highly crucial concern. The traditional VM-based anti-malware software themselves a potential target for malware-based attack since they are easily subverted by sophisticated malware. Thus, a reliable and robust malware monitoring and detection systems are needed to detect and mitigate rapidly the malware based cyber-attacks in real time particularly for virtualized environment. The Virtual Machine Introspection (VMI) has emerged as a fine-grained out-of-VM security solution to detect malware by introspecting and reconstructing the volatile memory state of the live guest Operating System (OS) by functioning at the Virtual Machine Monitor (VMM) or hypervisor. However, the reconstructed semantic details by the VMI are available in a combination of benign and malicious states at the hypervisor. In order to distinguish between these two states, extensive manual analysis is required by the existing out-of-VM security solutions. To address the foremost issue, in this paper, we propose an advanced VMM-based guest-assisted Automated Multilevel Malware Detection System (AMMDS) that leverages both VMI and Memory Forensic Analysis (MFA) techniques to predict early symptoms of malware execution by detecting stealthy hidden processes on a live guest OS. More specifically, the AMMDS system detects and classifies the actual running malicious executables from the semantically reconstructed process view of the guest OS. The two sub-components of the AMMDS are: Online Malware Detector (OMD) and Offline Malware Classifier (OFMC). The OMD recognizes whether the running processes are benign or malicious using its Local Malware Signature Database (LMSD) and online malware scanner and the OFMC classify unknown malware by adopting machine learning techniques at the hypervisor. The AMMDS has been evaluated by executing large real-world malware and benign executables on to the live guest OSs. The evaluation results achieved 100% of accuracy and zero False Positive Rate (FPR) on the 10-fold cross-validation in classifying unknown malware with maximum performance overhead of 5.8%.

## 1. Introduction

Rapid improvement in technology of sensor devices, computing and communication devices and high-speed transmission media etc., are supporting the development of CPS. The CPS has evolved as a most crucial infrastructure that integrates devices like computation, networking, and physical elements together to facilitate and communicate various applications [1]. Primary goals of the CPS is to enable intelligent monitoring as well as controlling of the various applications running on the computing devices. In other words, CPS main objective is to assist quick extraction of information, analysis of the data, decision making and data transmission in real time. The CPS has been widely adopted in various fields like robotics, health care, military, industrial control, power systems, avionics systems, intelligent building, smart electrical power grids, smart medical systems, etc., [2].

The central processing core of the CPS receives a massive amount of data from various cyber-enabled devices as well as other physical devices through communication networks. For example, in a smart and reliable transportation system, a vehicle equipped with the sensor device sends data to the roadside unit deployed on the roads which in turn transmits the received data to central

[*] Corresponding author.
*E-mail addresses:* ajayit13f01@nitk.edu.in (Ajay Kumara M.A.), jaidharcd@nitk.edu.in (Jaidhar C.D.).

http://dx.doi.org/10.1016/j.future.2017.06.002
0167-739X/© 2017 Elsevier B.V. All rights reserved.

processing core of the CPS over a high-speed network. Since a large number of devices sends the data, inbound traffic at the central processing core of the CPS is massive. Generally, CPSs are real-time systems, thus, the significance of computational latency of their critical components are as same as their correctness of their functional modules. In order to mitigate loss the caused due to the violation of the real-time property of a perilous function, the best solution is to deploy central processing core of the CPS in the cloud computing (virtualization) environment. The main aim of development of the cloud computing is to offer fast computational speed. Moreover, the cloud computing can offer a vast pool of resources to store, process, and analyze the data, which creates precise data information. Elasticity property of the cloud computing offers adequate amount resources to central processing core of the CPS as and when the demand arises [3]. The virtualization enables a number of benefits such as reduced operation and maintenance cost as well as setup cost, easy the procurement process, more importantly, dependability and availability.

Much of the works have been done in the context of hardware virtualization and fault-tolerance on virtualization [4,5]. Recent work [6] provides the aspects of integration and consolidation of CPS with virtualization. Since central processing core of the CPS frequently communicates with the other systems in the physical world through communication media, malware can be used as a weapon by an attacker or malignant user who intentionally desire to create havoc [7].

Malware is a malicious program developed with an intention to launch malignant tasks. Generally, malware uses the stealthy technique to exploit the system and network vulnerabilities in order to gain control of the user system to achieve unauthorized activities [7]. Its prime target is not only restricted to destroy the single system or group of systems, it also targets to disrupt the normal functions of the computer networks [8]. This results in increasing threat to the information systems that are used in day to day activities. Malware not only makes use of zero-day exploits to acquire the control of vulnerable machine but also Stealthily achieve its intended job by hiding in an infected system and cause contaminations over time. The proliferation of new variants or a particular class of malware constantly uses code obfuscation technique [9] or rootkit functionality [10] to subvert most of the existing in-host security solutions to gain access to the targeted machine.

The virtualized environment can also be targeted by an attacker due to the reason that VMs can easy to avail through the cloud service provider [11]. With an intention of destroying the entire virtualized platform, an attacker or malignant user may use the rented virtual machine to launch malware based attacks or cyber-attacks. To mitigate such kinds of attacks quickly and efficiently in a timely manner, a real-time powerful tamper resistant malware detection system is highly essential.

To tackle this issue, VMI [12] has emerged as a promising out-of-VM security solution that operates at the VMM and facilitates in constructing a semantic view of the live guest OS in real-time by introspecting low-level details of the volatile memory state of the introspected guest OS without the consent of one being monitored. In addition, it allows to view and acquire running process details, system calls invoked by the process, kernel modules etc., of the Monitored VM. However, obtaining meaningful run state details such as network connections, the system call details, process list, and module list, etc., from the primary memory of the introspected guest OS is a biggest challenging task for the VMI and it is named as the semantic gap [13,14]. To fill this semantic gap issue, various research works have been proposed over the last few years by considering a number of different constraints of the guest OS [15,16].

Other significant challenges in precisely detecting the malicious executables, mainly in a virtualized environment are:

- In order to provide real-time protection for CPS which is operating within the guest OS in a virtualized cloud environment, the traditional VM-based security solution are inadequate to protect guest OS resources against the sophisticated malware. Therefore, VMI techniques provide out-of-VM security solution for the introspected guest OS while operating at VMM. However, introspected information (e.g., processes) was available in dubious forms.[1] For example, the proliferation of the Kelihos malware on the guest OS spawned a number of malicious child processes before exiting from the main process [17]. In such an instance, manually distinguishing, detecting, and preventing the running malicious processes from hundreds of benign processes was time-consuming for a security administrator, as it required a wide knowledge of the malicious executables.
- The CPS functioning on the virtualized environment (e.g., guest OS) are targeted by malicious executables use the code obfuscation technique [18,19] and other stealthy malware attacks [20]. Hence, performing early symptoms of malware execution and accurately estimating the stealthy hidden, dead and dubious malicious processes under the dynamic nature of the process creation and expires on introspected live guest OS is challenging task.

To address the aforementioned challenges, we propose an intelligent and guest-assisted AMMDS that leverages both the VMI and MFA techniques to perform three levels of the investigation to secure the critical infrastructure of the virtualized cloud environment. As a first level of investigation, it performs introspection and precisely detects the semantic view of the hidden and malicious process to estimate the perfect infection state of the live introspected guest OS. It seizes the execution state of the introspected guest OS by capturing the memory dump of the monitored guest OS soon after it identifies the unusual behavior and then instantly reconstructs and extracts executables from the acquired memory dump to carry out next level of investigations. As a second level of investigation, the OMD component of the AMMDS examines the extracted executables to ascertain the malicious one. The OFMC component of AMMDS analyzes the extracted executables in order to identify unknown or zero-day malware using machine learning techniques as the third level of the investigation. The AMMDS is evaluated by using real malware datasets on the virtualized environment established using Xen hypervisor. Our empirical results show that AMMDS is robust in detecting and classifying unknown malware that can evade VM-based security solution, and it only incurs acceptable moderate run-time overhead.

The key contributions of the present work are as follows:

1. We have designed, implemented, and evaluated a consistent, real-time VMM-based guest-assisted AMMDS that periodically examines the state of the live guest OS system while defending the CPS to detect the running malicious processes from the forensically reconstructed executables.
2. We have implemented an Intelligent Cross-View Analyzer (ICVA) as proof-of-concept and implanted it into the AMMDS to intelligently cross-examine the internally (VM-level) and externally (VMM-level) gathered state information to detect hidden, dead, and dubious processes and also to predict the early symptoms of malware execution using the novel Time Interval Threshold (TIT) technique.
3. OMD and OFMC are two prime sub-components of the AMMDS. These are practically implemented and implanted into AMMDS to distinguish an actual malware from semantic view processes that are reconstructed as dubious

[1] Dubious process is a process that is currently running on a guest OS, and it may or may not be a malicious process (not hidden).

executables at VMM. The OMD performs a malicious check on hidden and dubious processes by cross-verifying with its LMSD and online scanner. On the other hand, OFMC uses the extracted features recommended by feature selection techniques in the form of Final Feature Vector (FFV) to perform malware analysis in offline.

4. To the best of our knowledge, our proposed AMMDS is the first to adopt machine learning techniques from a VMI perspective at the VMM, and to perform runtime detection of unknown malware from the introspected-cum-forensically extracted executables of the introspected guest OS. This idea opens the door for researchers to leverage other scientific techniques at the VMM to perform automatic detection of malware.
5. The robustness of the AMMDS was practically evaluated by injecting large samples of real-world Windows malware and rootkits on a live Windows guest OS. The OMD of the AMMDS is powerful enough to distinguish between malicious and benign executables. Similarly, the OFMC achieved malware detection rate of 100%, and 0% FPR with maximum performance overhead of 5.8%.

## 2. Background and related work

The related works specific to our proposed technique are outlined from a VMI, MFA, and machine learning perspective. The VMI was pioneered by Garfinkel and Rosenblum by developing a prototype called Liveware [12] for detection of an intrusion by examining the low-level state of the guest OS from outside the VM. Since then, numerous efforts have focused on the significant adoption of the VMI for malware detection and analysis [21,20], process monitoring [22], rootkit detection [23], etc.

Several out-of-VM security approaches have also proposed to monitor and ensure the protection of the introspected system, while addressing specific security problems of the guest OS. The Antfarm [24] approach tracks and implicitly obtains the execution of the guest processes' information, while functioning at the VMM. However, the traced processes may sometimes be incorrect or noisy. Lycosid [25] extended the Antfarm approach aimed to detect and identify the hidden processes at the VMM based on the obtained trusted and untrusted view of the guest OS processes. However, the employed cross-view analysis technique [26] by the Lycosid depends on manual analysis that is incapable of detecting the disguised processes[2] that appear due to the malware. The VMwatcher [20] uses the guest view casting technique to externally reconstruct the internal semantics view of the guest OS, while functioning at the VMM. It uses the view comparison-based method to detect elusive malware based on the discrepancy obtained between the internal and external views. However, the results still do not help in detecting particular variants of the stealthy malware, which use code obfuscation technique [28]. Other VMI-based solutions such as XenAccess [29] and Patagonix [30] ensure the security of the introspected VM at the hypervisor by performing manual analysis of the reconstructed semantic view.

From another perspective, a number of VMM-based in-guest monitor techniques [31,32] have also been explored to achieve better robustness while leveraging the security advantages from an out-of-VM security approach. Lare [33] pioneered the active monitoring approach by placing a hook into an introspected system using the VMM-protected address space. These hooks trap and analyze the events inside the guest OS and trigger the security application of the trusted VM to take suitable action against the attacks. The main limitation of this approach is its high overhead. The processes-out-grafting [22] inserts the kernel modules into the guest OS to relocate the suspect processes from the introspected VM on to a secure VM, whereby the parallel running VMI tools can access the data structure of the guest OS without an intervention of the untrusted introspected VM. However, the proposed approach is limited to out-graft a single process. The SYRINGE [34] uses a guest-assisted function-call injection technique to implant a function on to the introspected VM along with a localized shepherding technique to confirm the execution of the invoked guest code on to the guest OS. However, this approach does not ensure the integrity of the data delivered by the infected guest OS that was tampered by kernel-level attacks.

Meanwhile, leveraging the MFA with the VMI at the hypervisor offers many advantages [35,36] such as examination of the rich semantic view of the critical kernel data structure of all OS [37] manipulated by operating system specific rootkit or malware. More importantly, it significantly reduces the development of an introspection program of a large kernel data structure by addressing the semantic gap problem [38]. As the forensic examination begins with the captured live physical memory dump, a number of host [39,40] and hardware assisted-virtualization [41,42] based anti-forensic acquisition techniques evolve to perform memory acquisition of the live VM. There have been efforts [43,44] to perform live forensic analysis by leveraging virtualization extension on a potentially compromised system without modification or termination of the guest OS state. However, digital forensic involves extensive manual analysis to gather empirical evidence of real malicious executables from the infected memory dump.

Machine learning perspective: Over the past decades, several research works have witnessed the use of machine learning techniques to detect unknown malware executables. Schultz et al. [45] pioneered machine learning for detection of malicious executables by understanding the features of the malware and benign executables. Generally, there are two kinds of methods that can be utilized to detect and classify malware, namely, static malware analysis and dynamic malware analysis.

In static malware analysis, the detection of malware is performed by examining the malicious executables without its execution. The traditional malware detection and classification approaches use static properties of malicious executables that include header details, embedded strings details, packer signature, checksum or MD5 hash, metadata etc. Zubair et al. [46] presented a malware detection framework with the name PE-Miner that has the potency to extract individual features from a Portable Executables (PE) file to spot unknown or zero-day malware in real-time. The proposed approach is based on a three-fold research methodology that consists of identifying structural features of the PE file, reduction of feature by pre-processing and classifying based on the data mining algorithms. The authors experimented with large data samples and achieved detection rate greater than 99% with less than 0.5% FPR.

A Hellal and LB Romdhane [47] proposed a graph mining algorithm, called, minimal contrast frequent subgraph miner that extracts malicious behavioral patterns from malware executables. It based on the graph mining approach with static malware analysis. The proposed method is evaluated by considering 1083 malware and 1000 benign Windows samples and achieved highest detection rate with low FPR on detection of new variants malware and obfuscated malware. Ahmadi et al. [48] proposed a malware classification approach. The extraction of the features was based on the structure and content of the malware samples as multi-feature. The authors used an ensemble based XGBoost learning algorithm to validate their classification methodologies and achieved 98.80%

[2] Disguised processes [27]: These processes may appear as legitimate ( eg, svchost.exe) by attaching themselves to existing benign processes based on their injected malicious code or by originating from a wrong directory path on the guest OS.

detection accuracy on a large malware dataset. A number of classic approaches use Bytecode N-gram [49–51] and Opcode N-gram [52–54] based feature extraction techniques to detect and classify malware using the static analysis method. However, the main limitation of this analysis is that it is susceptible to inaccurate detection of the malware that uses strong evasion and obfuscation techniques [55]. As a consequence, static malware analysis techniques are inadequate to detect unseen malware [56].

To address the limitation of static malware analysis, dynamic or behavioral analysis based techniques are widely used, it detects and classifies the malware by observing the behavior of the malicious executables while it is actually running on the controlled and monitoring environment. It can use API calls [57] or system call [58] or any other function-based [59] features to observe the behavior of the executables during their run time on the OS. Therefore, these approaches are well suited for capturing new and unseen malware variants.

Moskovitch et al. [8] have proposed an approach for the detection of worm activity on a monitored computer system. They managed to extract over 323 computer features using their developed agent on the monitored system. The obtained features were reduced by using four feature selection techniques, while four machine learning classifiers were used to evaluate the results. The evaluation results showed 99% accuracy for specific unknown computer worm activity. Shahzad et al. [60] proposed a novel genetic footprint concept by mining information from the process control block of a process. The footprint consists of the semantic behavior of each executing process that can be used to discover malicious processes at run-time. The proposed approach is capable of discovering a malicious process rapidly (less than 100 ms) and achieved an accuracy of 96% with 0% false alarm rate. Similarly, Miao et al. [61] proposed bilayer abstraction method, it utilized discriminant and stable behavior feature from semantic analysis of dynamic API call sequences that are extracted during in execution of samples in a controlled system. Finally, the authors have experimented using improved version of one-class Support Vector Machine (SVM) algorithm to detect a new variant of unknown malware with low FPR.

Islam et al. [62] presented classification method that uses static and dynamic features to classify benign and malware samples. The static feature vector was formed by extracting a function length frequency and printable string information from each executable (benign and malware). Similarly, the dynamic features include API function names and parameters execution of each benign and malware. Finally, these three features are combined to construct integrated feature vector. Authors have used four machine learning classifiers namely, SVM, Random Forest, Decision Trees and Instance-Based (IB) classifier to evaluate the classification methodologies and achieved 97.05% of detection accuracy. Recently, A.Kumar et al. [63] also used an integrated feature vector that was comprised of each PE files of header fields raw value and derived values. Authors have used various machine learning classifiers and achieved 98.4% classification accuracy.

In contrast, very few approaches were concentrated to protect CPS while detecting and performing classification of malware from a VMM perspective. In order to protects real-time protection of CPS, recently, Huda et al. [64] proposed a semi-supervised approach that protects the CPS against unknown malware attacks. It uses an automatic malware database update strategy that helpful to extract patterns of dynamic changes of malware attacks. The proposed approach has been evaluated against a real-world malware sample of both static and dynamic malware features by using four supervised machine learning classifiers, such as Random Forest, SVM, J48 and IB and achieved higher malware detection rate. Recently, one such research effort [65] leveraged a one-class SVM technique at the hypervisor for detection of malware on a live VM. The utilization of the features was at both system and network level. In addition, custom volatility plug-in was used to extract features from every resident process that impacted on the generating training dataset. Overall, this approach achieved more than 90% detection accuracy for two types of malware used in the work.

While the behavioral analysis based method achieve promising results, it has shortcomings. For example, when a sophisticated stealthy malware, which uses rootkit functionality, is executed on the sandbox or VM, the majority of execution path will not execute or it is hard to retrieve full execution path and behavior of such malware by the existing dynamic malware analysis approach [66,67].

In contrast, our AMMDS uses introspected and forensically extracted hidden and dubious executables at the VMM to measure the detection accuracy of the malware on live introspected VM where the CPS is functioning.

## 3. Assumption and threat model

In this work, we firstly assume that both the VMM and the monitoring VM are to be trusted and are operating under trusted computing base [68] that sufficiently enforces physical security control, while resisting hardware-based attacks on the virtualized cloud infrastructure [69]. Secondly, the malware leverages guest OS vulnerabilities that cannot jeopardize the security of the AMMDS operating in the privileged domain (Dom0) of the Xen hypervisor. These assumptions are consistently shared by most of the VMM-based previous security researches [12,33,22,20]. Thirdly, we do not attempt to hide the fact that the established communication channel between the State Information Requester (SI-Requester) of the AMMDS and the Guest Assisted Module (GAM) is secure during the lifetime of the live introspected VM. It cannot be modified by any kind of attack or security threats. Furthermore, some previous researches [70] have highlighted the successful compromising of the VMM and Dom0, but that is beyond the scope of this paper.

## 4. Overview of AMMDS

The AMMDS is a VMM-based guest-assisted introspection system that advances the current out-of-VM security approach in an automated, isolated, real-time, scientific manner, while functioning at the secure Monitoring Virtual Machine (Monitoring VM) or Dom0. Fig. 1 shows an overview of the AMMDS, its major components are: Guest Virtual Machine Introspector (GVM-Introspector), ICVA, and Malware detector. It is introduced to efficiently investigate and detect any hidden, dead, and dubious processes, while predicting early infection of the malware symptoms by internally gathering and externally introspecting the volatile memory of the live untrusted Monitored Virtual Machine (Monitored VM). The AMMDS achieves this goal by using its integral component called ICVA. Furthermore, Malware detector of the AMMDS consisting of two sub-components that are OMD and OFMC. The OMD identifies whether the detected hidden and dubious process is malicious or benign by cross-comparing with its LMSD and online malware scanner based on the computed hash digest for each of the extracted dubious executable. The OFMC leverages a practical machine learning techniques to classify the execution of the unknown malware from the reconstructed dubious semantic view of the processes (i.e., noticed from a VMI perspective) forensically extracted as executables from the seized live memory dump of the introspected guest OS.

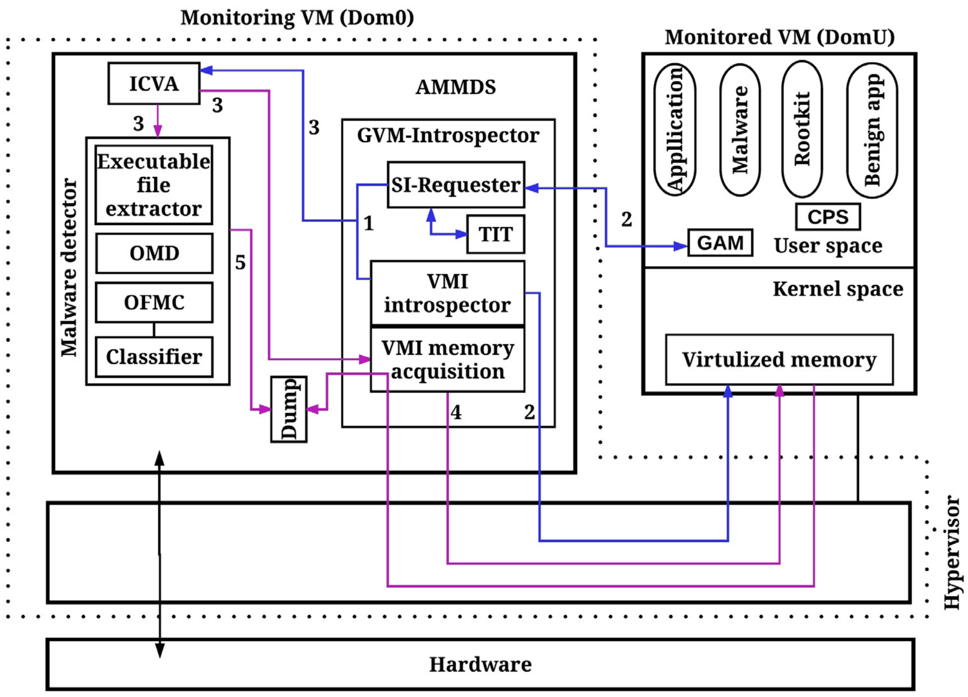


**Fig. 1.** The proposed VMI-based AMMDS.

### 4.1. GVM-introspector

The prime function of the GVM-Introspector is to introspect and extract the running processes information of the Monitored VM. Its sub-components are: SI-Requester, TIT, VMI introspector, and VMI memory acquisition. To start with, the GVM-Introspector initiates the procedure of investigation by signaling to both the SI-Requester and VMI introspector (step 1) to introspect and acquire the current execution of the process state information of the Monitored VM. The SI-Requester (step 2) triggers the GAM via a secure communication channel[3] to internally enumerate the executing process state of the Monitored VM. Upon receiving the internally acquired process state information, the SI-Requester verifies whether the reply arrived within TIT. The time interval between the state information request sent and the reply received by the SI-Requester from the GAM is known as TIT. If the time gap between the state information request and the reply lies within the TIT, it continues the operation. At the same time, the VMI introspector (step 2) introspects and reconstructs the memory state of the Monitored VM from the hypervisor (externally) to get currently running process details. The procedure followed by the VMI introspector to reconstruct and obtain the semantic view of the processes (including hidden processes) by traversing the _EPROCESS Windows kernel data structure of the introspected VMs is discussed in Section 4.1.1. In addition, the TIT, as shown in Fig. 2, efficiently addresses the time synchronization problem, which impacts on the detection of the malicious processes under the dynamic creation and destroy of processes as discussed in the second challenge of Section 1.

For example: Let $T_1$ be the date and time at which the state information request is sent to the Monitored VM, and $T_2$ be the date and time at which the reply is received by the SI-Requester from

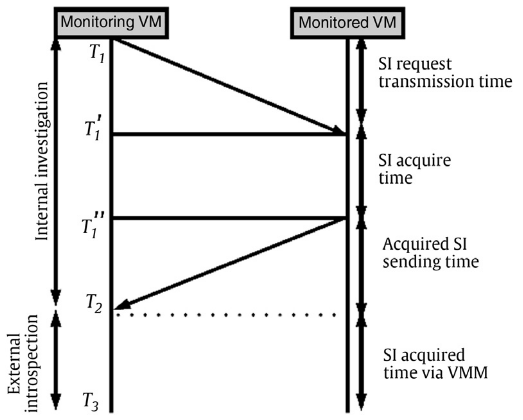


**Fig. 2.** Time interval threshold used by AMMDS.

the GAM. After receiving the state information, the SI-Requester checks the time interval between $T_2$ and $T_1$; $T_2 - T_1 > \Delta T$. Then, the SI-Requester instantly resends the request (maximum 3 times), if the response is *delayed* or not received within the predefined $\Delta$T (where $\Delta T$ denotes the predefined threshold time that ranges between 2 to 3 second). Then, the AMMDS confirms that the Monitored VM is in an infected state and immediately informs the VMI memory acquisition (step 3) to pause and perform memory acquisition (step 4), and then, to resume running the Monitored VM.

If $T_2 - T_1 \leq \Delta T$ then, both the SI-Requester and VMI introspector continue periodic introspections of the Monitored VM by extracting and relocating the process run-state information to the ICVA (Step 2 and Step 3). Further, the ICVA (will be shortly introduced) cross-examines the acquired process state information. This process continues throughout the lifetime of the live Monitored VM. The novel time synchronization technique, the TIT helps the ICVA to detect hidden and malicious state information of the Monitored VM. For instance, the processes $P_1, P_2, P_3, \ldots, P_N$

[3] A secure communication channel is established between the SI-Requester and the GAM by the AMMDS as soon as the Monitored VM is launched by the VMM. The SI-Requester holds native information (IP address and Port number) of the Monitored VM and it triggers the GAM in the form of state information request to receive internally acquired process information of the Monitored VM.

currently being run at the Monitored VM and their details are extracted internally between the time intervals $T_1$ and $T_1''$. If any process expires or dies after $T_1''$ and before $T_2$, the process details will not appear in the state information caught externally by the VMI introspector, but can be found in the internally captured state information, such processes are treated as `dead processes`.

In contrast, if a new process, $P_{N+1}$ is created between $T_1''$ and $T_3$, then the process details will appear only in the externally captured state information and not in the internally captured state information. As a result, process $P_{N+1}$ is recognized as a `hidden process`, even though process $P_{N+1}$ is not concealed. Thus, a disparity emerges between the internally and externally captured state information. Further, to check the maliciousness of the new process $P_{N+1}$, the ICVA immediately signals to the VMI memory acquisition (step 4) to pause and accomplish memory acquisition of the Monitored VM. The executable file extractor extracts the entire executable file of the corresponding process from the seized memory dump of the Monitored VM.

#### 4.1.1. *Memory state reconstruction*

Since the VMM view of the Monitored VM is available in raw memory state, the AMMDS performs memory state reconstruction before it collects and analyzes the process run state information of the Monitored VM at the VMM. This can be done with the VMI introspector by leveraging the address translation mechanism [29]. The use of the *xc_map_foreign_range()* function, provided in the Xen Control Library (libxc) helps the VMI introspector of AMMDS to understand and reconstruct the volatile memory artifacts of the live Monitored VM without its consent. Later, the same function accesses the RAM artifacts, and finally, converts the page frame number to memory frame number [71].

In the Windows system, each process associated with a data structure is called as `_EPROCESS`; each `_EPROCESS` has many data fields including one Forward Link (`FLINK`) pointer and one Backward Link (`BLINK`) pointer. The `FLINK` contains the address of the next `_EPROCESS`, while the BLINK stores the address of the previous `_EPROCESS`. The first field of the `_EPROCESS` is a process control block, which is a structure of the type Kernel Process (`KPROCESS`). The `KPROCESS` is used to provide data related to scheduling and time accounting. The other data fields of the `_EPROCESS` are `PID`, Parent PID (`PPID`), exit status, etc., [71]. The field position of the `PID` and the `PPID` in the `_EPROCESS` structure may differ from one OS to another, and the series of `FLINK` and `BLINK` systematizes the `_EPROCESS` data structure in a doubly linked list. The Windows symbol, such as the `PsActiveProcessHead` (head of the doubly linked list) traverses the whole of the doubly linked list `_EPROCESS` from the beginning to the end providing all the running process details. In order to identify the hidden processes at user and kernel-mode of the Monitored VMs, our proposed VMI introspector scans the raw memory by looking at the `_EPROCESS` structure patterns of the Monitored VM, as similar to the previous approach [20,72].

### 4.2. *Guest assisted module*

The GAM is a lightweight component that is placed inside the Monitored VM (soon after the guest OS are created by the VMM). It is controlled and operated by the SI-Requester on-demand via an established secure communication channel. During introspection by the AMMDS, the GAM will not create its own processes but will make use of its built-in `tasklist` command of the native Windows to acquire the running processes of the live Monitored VM and forward it to the SI-Requester in the form of a text file. The GAM can be tampered by the malware as it is placed in the untrustworthy Monitored VM. Under these circumstances, the AMMDS estimates the symptoms of malware execution when *delay* or *modification* occurs, while forwarding the internally acquired state information by the GAM (see Section 4.1).

**Table 1**
Notation and description.

| Symbol | Meaning | Symbol | Meaning |
|---|---|---|---|
| BFV | Benign Feature Vector | *ps* | Process |
| MFV | Malware Feature Vector | PN | Process Name |
| PID | Process Identifier | DO | Descending Order |
| $INT_{psc}$ | Internal process count | MG | Malware Group |
| $EXT_{psc}$ | External process count | BG | Benign Group |

### 4.3. *Intelligent cross-view analyser*

The ICVA is an integral component of the AMMDS and its prime function is to perform an intelligent examination of the internally captured and externally introspected execution state information to recognize hidden, dead, and dubious running processes of the Monitored VM. The notations used by the AMMDS are shown in Table 1.

The process details of the Monitored VM introspected from the hypervisor undergo preprocessing operation, and are then stored as $EXT_{ps} = \{PID \parallel PN_1, PID \parallel PN_2, PID \parallel PN_3\ldots.., PID \parallel PN_m\}$, where, the $PID \parallel PN_m$ represents the $m^{th}$ process. The internally captured process details after the preprocessing operation are represented as $INT_{ps} = \{PID \parallel PN_1, PID \parallel PN_2, PID \parallel PN_3\ldots.., PID \parallel PN_n\}$, where, $PID \parallel PN_n$ represents the $n^{th}$ process.

The ICVA conducts the preprocessing operation while removing unimportant state information and sorts the elements of both the $EXT_{ps}$ and $INT_{ps}$ in ascending order based on the PID. The total number of processes gathered from the $EXT_{ps}$ are symbolized as $EXT_{psc}$

$$EXT_{psc} = \mid EXT_{ps} \mid \tag{1}$$

Similarly, the total number of processes gathered from $INT_{ps}$ are symbolized as $INT_{psc}$

$$INT_{psc} = \mid INT_{ps} \mid \tag{2}$$

where, $\mid INT_{ps} \mid$ and $\mid EXT_{ps} \mid$ represent the cardinality of $INT_{ps}$ and $EXT_{ps}$, respectively. The cardinality of $INT_{ps}$, $EXT_{ps}$ is the total number of elements (processes) in the $INT_{ps}$ and $EXT_{ps}$.

The ICVA proficiently detects the hidden, dead, and dubious running processes and predicts the symptoms of the malware execution on the Monitored VM based on the decision function as follows:

$$ICVA(EXT_{ps}, INT_{ps}) = \begin{cases} Hidden & \text{if } (PID/PN \in EXT_{ps} \text{ and } PID/PN \notin INT_{ps}) \\ Dead & \text{if } (PID/PN \notin EXT_{ps} \text{ and } PID/PN \in INT_{ps}) \\ Dubious & \text{if } (PID/PN \in EXT_{ps} \text{ and } PID/PN \in INT_{ps}) \end{cases} \tag{3}$$

However, at the end of the scrutiny, it is not feasible for the ICVA to check whether these processes are malicious or not. To resolve this ambiguity, it commands (on confirmation of Eq. (3)) the VMI memory acquisition (step 4) to pause and perform memory acquisition of the Monitored VM. At the same time, the executable file extractor (step 5) component extracts all the hidden and active dubious executables (.exe).

### 4.4. *Malware detector*

The Malware detector of the AMMDS consists of three subcomponents that are Executable file extractor, OMD, and OFMC.

#### 4.4.1. *Executable file extractor*

The major function of the executable file extractor is to extract complete executables that correspond to detected hidden and dubious processes as indicated by the ICVA. The executable file extractor accomplishes this task by utilizing the *procdump* plugin of an open source volatility tool[4] based on the consistent state of seized memory dump of the Monitored VM. The executable file reconstruction is achieved by parsing the PE header data structure [73,74] from the obtained VM memory dump. Once the hidden and dubious executables are extracted, the OMD and OFMC investigate them individually to ascertain any malicious substance is present in the reconstructed executables.

#### 4.4.2. *Online malware detector*

The OMD computes a hash digest for the extracted executables soon after receiving the confirmation from the executable file extractor. The OMD computes three distinct hash digests such as Secure Hash Algorithm-256 (SHA-256), Secure Hash Algorithm 1 (SHA-1), and Message Digest (MD5), for each of the extracted executables. Further, these computed hash digests checked with LMSD[5] to identify the malware which are known in the digital world. If the OMD does not find any match, then it sends the generated hash digests to powerful publicly available free online malware scanner[6] to obtain an analysis report of the examination that provides whether the tested executable file is malware or benign. The left side of the Fig. 3 depicts the sequence of operations followed by the OMD to check maliciousness of the given input executables. However, the main limitation of the online malware scanner is that it is unable to detect the new variants of malware due to unavailability of a new malware signature in its database. Meanwhile, malware detector uses the OFMC to detect and classify unknown malware using machine learning techniques.

#### 4.4.3. *Offline malware classifier*

The OFMC is another important subcomponent of the malware detector and it addresses the limitation of the OMD, while accurately classifying any kind of executables as benign or malware by employing machine learning techniques. For any machine learning based malware detection approach, first, the classifier model should be trained with a sufficient number of benign and malware executables so that the classifier model can easily and quickly distinguish malware executables from benign executables. The training phase of OFMC comprises of feature extraction and feature selection techniques (will be shortly introduced) and FFV generation which in turn needed to perform malware classification. In the evaluation phase we perform detection of unknown malware which are forensically reconstructed as detected hidden and dubious executables based on the trained classifier. In this evaluation phase, the OFMC functions on the reconstructed executables by extracting N-grams as features and then prepares the testing file by using FFV with extracted N-grams of an executable file to be verified. The right side of the Fig. 3 depicts the steps followed by the OFMC to identify the given test input executables as benign and malicious.

**Feature extraction technique**: The executables are used as input files in the first step of feature extraction to extract the hexadecimal dump, and then pre-process the hexadecimal dump to remove any irrelevant information. After the pre-processing operation, only the byte sequences that represent a snippet of the machine code of the executable received. The extracted byte sequences are grouped in the form of N-gram [75], which represents contiguous bytes sequences, where N represents the number of bytes. In this work, we have chosen N-gram of size 4 bytes for the OFMC experimental analysis to achieve best malware detection rate. The steps involved to obtain the N-grams from the

**Algorithm 1:** Feature extraction

```
Input  : Binary files (.exe) F = {$f_1, f_2, f_3, \ldots f_M$}
Output: N-gram files $f_N$ = {$f_{N_1}, f_{N_2}, \ldots f_{N_M}$}

1  foreach file $f_i \in F$ do
2      Extract hexadump, N-grams
3      $hd_i \longleftarrow$ hexadump($f_i$)   // hd - hexadecimal dump
4      $g_i \longleftarrow$ preprocess($hd_i$)   // g - temporary file
5      Create N-gram file $f_{N_i}$
6      while not EOF($g_i$) do
7          N-gram $\longleftarrow$ N-gram ($g_i$)
8          $f_{N_i}$.append(N-gram)
9      end
10 end
11 $f_N$ = {$f_{N_1}, f_{N_2}, \ldots f_{N_M}$}
```

executables is shown in Algorithm 1. Each individual N-gram is considered as a feature and all N-grams of all executables in the training dataset are treated as original feature vector.

The N-gram based technique produces a large number of N-grams that include duplicate N-grams. All the N-grams cannot be used as final features to generate a training file as well as a testing file as needed by the classifier because it may result in memory consumption and performance overhead. In addition, it may also reduce the predictive performance of the classifier with more FPR. In order to address these issues, the feature selection technique is employed.

**Feature selection technique:** The approach for selecting the best features from the original feature vector plays an imperative role in classifying the input files accurately. The feature selection step identifies which features are highly crucial and which are noisy from the original feature vector by generating a score for each feature. The noisy features are ignored because they degrade the predictive accuracy of the classifier. In this work, OFMC comprising of two statistical techniques such as NGL Correlation Coefficient (CC) and Odds Ratio to rank each feature individually and recommend the prominent features based the procedure described in Algorithm 2.

**(a). NGL Correlation Coefficient**: NGL Correlation Coefficient (CC) is a variant of the chi-square test [76]. It selects the features (N-grams) that are correlate with class $C_i$ and does not select features that are correlated with other classes. The NGL CC score for a given N-gram of class $C_i$ is computed as follows:

$$NGL(\text{N-gram}, C_i) = \frac{\sqrt{N}(PS - RQ)}{\sqrt{(P+R)(Q+S)(P+Q)(R+S)}} \tag{4}$$

Where, $N$ represents the total number of N-gram files in the dataset, $P$ indicates the number of N-gram files in class $C_i$ that contains N-gram, $Q$ is the number of N-gram files other than class $C_i$ that contains N-gram, R is the number of N-gram files in class $C_i$ that does not contain the N-gram, and $S$ is the number of N-gram files that does not contain the N-gram, other than class $C_i$. The class $C_i$ = {benign, malware}.

**(b). Odds Ratio**: It is one of the popular feature selection techniques. A positive score of Odds Ratio indicates that the given N-gram often appears in a given class as compared to other class. A negative score represents that the given N-gram presence is more in the other class. Odds Ratio for binary classification is defined as follows [77]:

$$OddsRatio(\text{N-gram}, C_i) = \log \frac{P(\text{N-gram} \mid benign)(1 - P(\text{N-gram} \mid malware))}{P(\text{N-gram} \mid malware)(1 - P(\text{N-gram} \mid benign))} \tag{5}$$

[4] http://www.volatilityfoundation.org/.

[5] LMSD consists of 107520 MD5, SHA1, and SHA256 hash digests for known malware which were downloaded from https://virusshare.com/ malware repository.

[6] https://www.virustotal.com/.

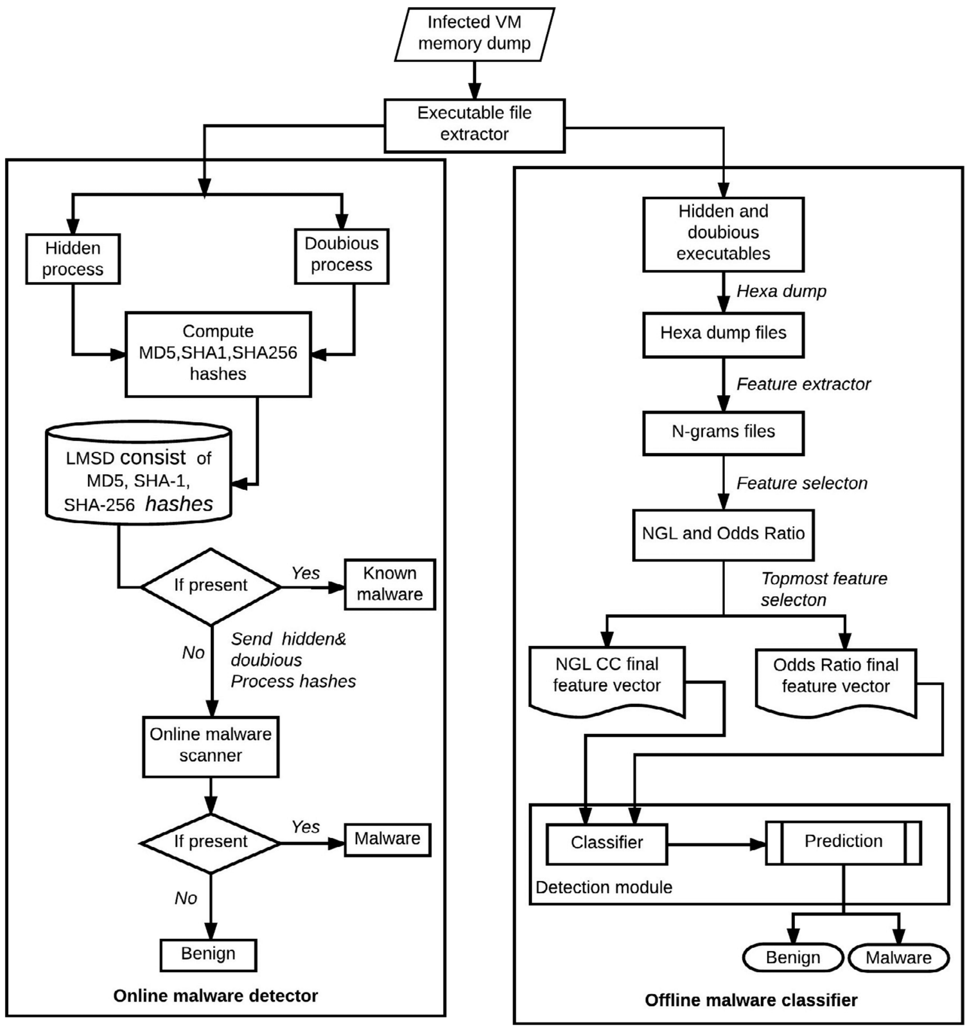


**Fig. 3.** Flow chart of the AMMDS for detection of malware using OMD and OFMC components.

P(N-gram | *benign*) is the probability of occurrence of N-gram in benign class. Similarly, P(N-gram| *malware*) is the probability of occurrence N-gram in malware class. The class $C_i$ = {benign, malware}.

## 5. Implementation and evaluation

### 5.1. Experimental setup

To evaluate the efficiency of our proposed AMMDS, the host system that had the following specifications: Ubuntu 14.04 (Trusty Tahr) 64-bit operating system, 8 GB RAM, Intel(R) core(TM) i7-3770 CPU@3.40 GHz, was utilized to conduct experiments. The popular open-source bare metal hypervisor such as Xen 4.4.l was used to set-up a virtualized environment. The Windows XP SP3 32 bit Monitored VM was created as DomU to act as CPS and it was controlled by the Monitoring VM (Dom0) of the Xen hypervisor. The AMMDS was installed on the Monitoring VM and the popular open source VMI tool,[7] such as the LibVMI version 0.10.1 to acquire the RAM dump of the Monitored VM. An open source forensic memory analyzer such as Volatility was applied to construct executables from the acquired infected memory dump. The machine learning algorithms' suite such as WEKA [78] was employed to achieve offline malware detection and classification at VMM level.

[7] http://libvmi.com/.

### 5.2. Implementation

Our proposed AMMDS is implemented using the Python programming language and its implementation was structured at three levels: (i) the AMMDS functions as an advanced automated VMI-based security solution by using open source VMI tool to introspect semantic view of run state of the monitored VM. Further, it is also acquiring the memory dump of the Monitored VM when symptoms of malware are detected at the introspected guest OS without the knowledge of one being monitored. (ii) ICVA is implemented as PoC and induced into the AMMDS, which detects hidden, dead, and dubious processes. (iii) Implementation of the malware detection component includes both OMD and OFMC. The OMD checks the detected hidden and dubious process as benign and malware by cross-comparing with both LMSD and online malware scanner. At the same time, implementation of OFMC uses feature extraction and feature selection techniques (see Algorithms 1 and 2) facilitate to construct FFV in order to detect actual unknown malware from introspection-cum-forensically extracted hidden and dubious executables using trained machine learning classifiers at VMM.

**Algorithm 2:** Feature selection

```
Input  : $\triangle = \{B = \{f_{N_{B_1}}, f_{N_{B_2}}, \dots f_{N_{B_M}}\} \cup M = \{f_{N_{M_1}}, f_{N_{M_2}}, \dots f_{N_{M_M}}\}\}$
Output: Selected features (N-grams)

1  Create files BOR, MOR, BNGL and MNGL
2      // B(M)OR - benign (malware) Odds Ratio
3      // B(M)NGL - benign (malware) NGL
4  foreach file $f_{N_i} \in \triangle$ do
5      foreach N-gram $\in f_{N_i}$ do
6          Compute Odds-Ratio as per equation(5)
7          OR_score $\longleftarrow$ Odds-Ratio(N-gram)
8          if ($f_{N_i} \in B$) then
9              Store N-gram and OR_score into a file
10             BOR.append(N-gram, OR_score)
11         else
12             MOR.append(N-gram, OR_score)
13         end
14         Compute NGL score as per equation(4)
15         NGL_score $\longleftarrow$ NGL(N-gram)
16         if ($f_{N_i} \in B$) then
17             Store N-gram and NGL_score into a file
18             BNGL.append(N-gram, NGL_score)
19         else
20             MNGL.append(N-gram, NGL_score)
21         end
22     end
23 end
24 Sort BOR and MOR in descending order based on OR_score
25 Sort BNGL and MNGL in descending order based on NGL_score
26 Select top 'L' number of N-grams from BOR and MOR
   separately
27 Select top 'L' number of N-grams from BNGL and MNGL
   separately
```

### 5.3. Dataset creation and uses

About 3375 Windows malware samples (executables) were collected from the VX Heaven[8] malware repository by directly connecting the Windows Monitored VM to an external network. In addition, 675 Windows benign executables were also collected from a freshly installed Windows Monitored VM along with other data source[9] that included both Windows native utilities and application executables. These executables were invoked by our developed program on the Windows Monitored VM with different experimental scenarios, including the installation of 2 rootkits such as `Hacker Defender` and `FU Rootkit`. These rootkits used to explicitly hide the running benign and malware processes on the guest OS. The experimental results depicted in Table 2 illustrate the execution of different categories of malware samples on a live Monitored VM.

### 5.4. Evaluation and results discussion

In this section, we discussed the evaluation and results analysis of proposed approach based on VMM-based generated dataset that was generated by executing benign and malicious executables on live Monitored VM. In order to prepare the training and testing files that are required to evaluate the performance of the OFMC, the collected malware, and benign executables were divided into two parts: Set-A and Set-B. The Set-A consisted of 60% of total samples and these were used to train the classifier and the remaining 40% of samples were grouped into Set-B. In the testing phase, malware and benign samples from Set-B were executed on live Windows XP Monitored VM in different experimental cases (as shown in Table 2) while measuring the detection proficiency of the AMMDS to detect hidden, dead and dubious processes. Finally, the semantic view of these detected processes were forensically reconstructed as executables from the infected memory dump of the guest OS. These injected malware and benign samples were forensically extracted in the form of dubious executables to detect actual malicious executables by following the procedure discussed in Section 4.1.

The experiments were performed at different stages to evaluate the malware detection proficiency of the AMMDS. The procedure employed by the OMD to detect whether the extracted executables is malware or benign is discussed in Section 4.4.2. As part of the experimental observations, we have performed six different tests using different types of malware and benign executables on a freshly installed Windows Monitored VM for each test as shown in Table 2. More precisely, in test I, we executed 160 Trojan and 45 benign files, totaling 205 executables on a clean live Windows guest OS. Once all the executables were injected, some Trojan executables hid or disappeared on the Windows guest OS. At the same time, we also injected a Hacker Defender user-mode rootkit to explicitly hide two benign (*explore.exe* and *chrome.exe*) and one malware (*Trojan.Win32.exe*) process running on the Monitored VM. These experiments lasted 4 min. A periodic introspection of the AMMDS system helped in identifying the symptoms of malware execution by recognizing the disparity of the processes that emerged between the internal and external view of the processes state information of the Monitored VM (see Section 4.3).

As seen in test I of Table 2, the AMMDS system ascertained and counted the introspected processes, namely, internal, external, hidden, dead, and dubious processes as 221, 225, 5, 1, and 220, respectively. However, there were variations in the detected processes counted on the internally captured and externally introspected process state information including 21 (additional) running default processes of the clean Windows guest OS. More specifically, the internal view of the total process visible is 221, i.e., from the 205 total injected malicious and benign executables, 5 processes (3 were self-hidden by the Trojan and rootkit and 2 were benign processes that are *explore.exe* and *chrome.exe* explicitly hidden by the `Hacker Defender` rootkit) are hidden at the guest OS and 21 are native running processes of the guest OS. Finally, 221 are internally gathered processes. Note that these 5 hidden processes are not gathered by the GAM (i.e., internal view of guest OS) as it does not appear in the `tasklist` command of the GAM (See Section 4.2). Finally, the VMI introspector of the AMMDS externally (i.e., VMM level) introspect and ascertain these 5 hidden processes.

Similarly, in the II experiment, we injected 185 Backdoor and 45 benign files, totaling 230 executables on the clean Windows guest OS. Meanwhile, in this test we have injected direct kernel object modification based kernel-mode, `FU Rootkit` to explicitly hide a few benign and malware running processes by directly removing or unlinking it from the `_EPROCESS` data structure of the process list at the kernel-mode. Finally, the disparity of the processes is identified by the ICVA of the AMMDS similar to test I. Likewise, experiment III, IV, V, and, VI were conducted by executing other types of malware and benign executables on the guest OS. Finally, six different experimental test cases, the AMMDS reconstructed over 21 hidden and 1725 dubious executable as VMM-based generated malware dataset from six infected memory dumps of the guest OS at VMM as shown in Table 2.

In each test cases, the OMD computes hashes digest for all the reconstructed executables from the captured memory dump and

8 http://vxheaven.org/.
9 http://download.cnet.com/windows/, accessed on July 2016.

**Table 2**
Execution of malware and benign executables on live Monitored VM in different experimental test cases.

| Test #. | Malware types | # Executables used | | | # ps visible | ps introspected | # ps detected by AMMDS | | | |
|---|---|---|---|---|---|---|---|---|---|---|
| | | Malware | Benign | Total | by internal view | from external view | Hidden | Dead | Dubious | Time (in second) |
| I | Trojan | 160 | 45 | 205 | 221 | 225 | 5 | 1 | 220 | 2.45 |
| II | Backdoor | 185 | 45 | 230 | 249 | 250 | 2 | 1 | 248 | 2.25 |
| III | Worm | 210 | 45 | 255 | 273 | 274 | 3 | 2 | 271 | 2.81 |
| IV | Virus | 230 | 45 | 275 | 292 | 294 | 4 | 2 | 290 | 2.55 |
| V | Adware | 270 | 45 | 315 | 333 | 345 | 3 | 1 | 342 | 2.62 |
| VI | Spyware | 295 | 45 | 340 | 357 | 358 | 4 | 3 | 354 | 2.75 |
| Total # of executables | | **1350** | **270** | | | | **21** | | **1725** | |

**Table 3**
Identifying an actual malicious process from test-I of detected hidden processes by OMD of AMMDS.

| Malware type | No. of hidden process to be checked | MD5 hash digest for classified hidden process | Predicted as | PS name | Detection rate |
|---|---|---|---|---|---|
| Trojan | 5 | 55cc1769cef44910bd91b7b73dee1f6c | Malicious | *hxdef073.exe* | 51/53 |
| | | 6cf0acd321c93eb978c4908deb79b7fb | Benign | *chrome.exe* | 0/53 |
| | | bf4177e1ee0290c97dbc796e37d9dc75 | Benign | *explore.exe* | 0/53 |
| | | 1338dfc088a24a477dd3c6d65fe71b9b | Malicious | *stubd.exe* | 33/43 |
| | | 5fcfe2ca8f6b8d93bda9b7933763002a | Malicious | *kelihos dec.exe* | 36/54 |

**Table 4**
Identifying an actual malicious process from test-I of detected dubious executables by OMD of AMMDS.

| Malware type | No. of dubious process to be checked | MD5 hash digest for classified dubious process | Predicted as | PS name |
|---|---|---|---|---|
| Trojan | 220 | 0bf067750c7406cf3373525dd09c293c | Known malware | *EFMTnkTm-216.exe* |
| | | 376121485bee9e8885d879d5407388c3 | Known malware | *Win32.Hidedoor.exe* |

then performed cross-examination with LMSD and online malware scanner based on the computed hash digest. The time elapsed to generate the hash digest for malware of different types malware and benign (i.e., reconstructed executables of all six test cases) is shown in Fig. 4a. Similarly, the time taken by the OMD to detect the known malware by cross-comparing with LMSD (based on computed hashes) is shown in Fig. 4b. Table 3 represents the results of online malware scanner for the detected 5 hidden processes that included 1 rootkit self-hidden process (hxdef073.exe), 2 explicitly hidden benign processes by the `Hacker Defender` rootkit (*chrome.exe* and *explore.exe*), and other 2 are self-hidden by the Trojan malware on the monitored VM. The Fig. 5 represents the snapshot of execution of the *Kelihos dec.exe* malware (not hidden) detected by the online malware scanner with a detection rate of 36/54. Table 4 represents the AMMDS classified 220 as dubious executables and cross-compared with both LMSD and online malware scanner to detect any known malware. Finally, Fig. 6 represents a snapshot of OMD detected the known malware named *EFMTnkTm-216.exe* by cross-checking with LMSD.

## 5.5. Experimental methods

The prime aim of the OFMC is to explore the accurate detection and classification of malicious executables that are semantically reconstructed as hidden and dubious executables on live Monitored VMs using machine learning techniques. It has been seen in many researches [49,53,54,65,79,80] that the overall process of classifying unknown executables as benign or malware using machine learning techniques consists of two phases, training, and testing phase. In *training phase*, 60% of the benign and malware samples of the training set (i.e., Set-A) are used to prepare a training file. The first step in the training phase is to pre-process the training samples to derive the N-gram features using the approaches explained in Section 4.4.3. It has been seen in previous researches that N-gram feature of size 4 bytes exhibits promising results [49,50]. Therefore, we have decided to perform feature construction using N-gram of size 4 bytes during the evaluation of our proposed approach. Since the constructed features size is quite large, it is impractical to use all the extracted features to prepare a training file required to train the classifier to attain the

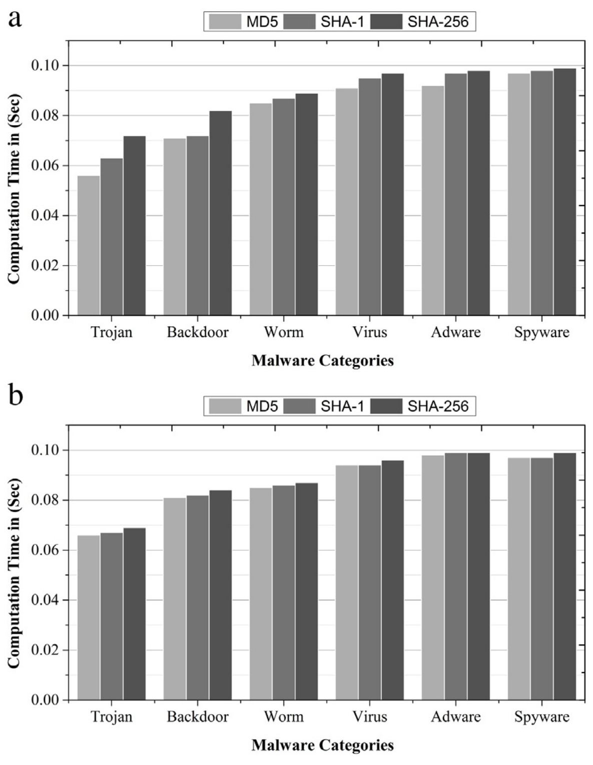
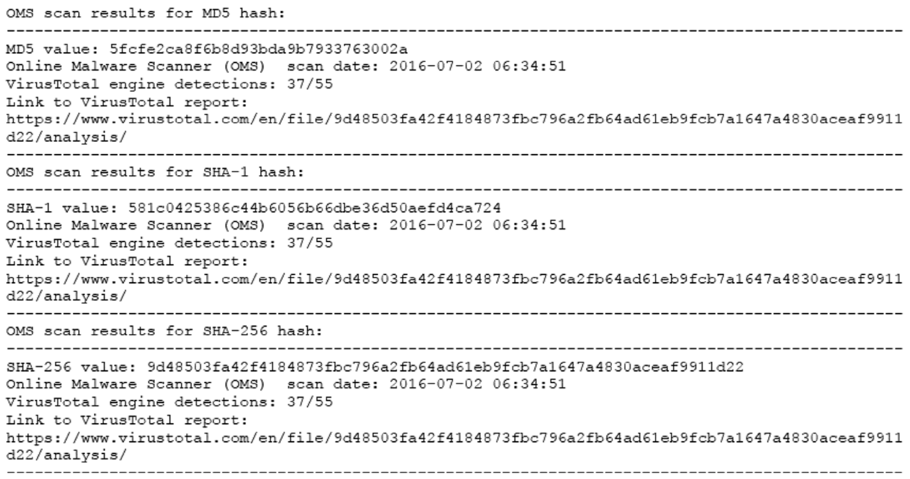


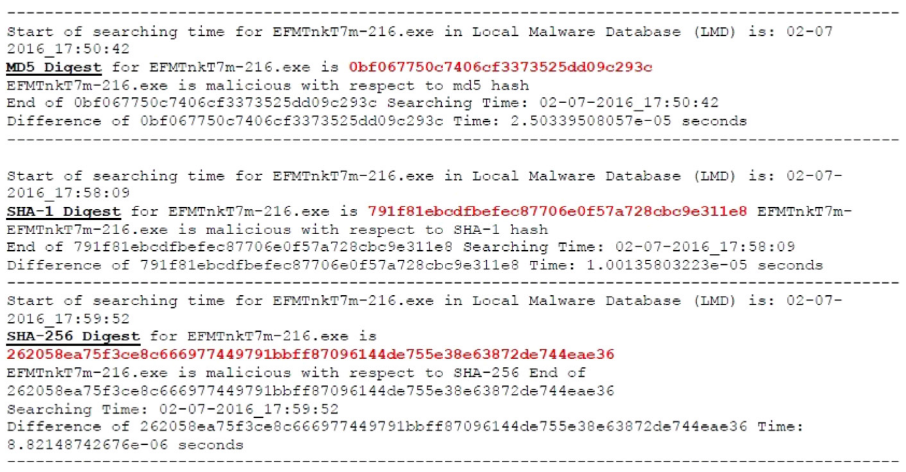
**Fig. 4.** The average time consumed by the OMD to generate SHA-256, SHA-1, and MD5 hash digest for execution of different types of malware (4a). Time taken by the OMD to identify the known malware by cross-checking with LMSD based on computed hash digest (4b).

real-time detection of the malware. Therefore, only the crucial topmost features were selected on the basis of the rank assigned by the feature selection techniques (as discussed in Algorithm 2). For each of the feature, two separate feature score (rank) are computed using two different feature selection techniques namely NGL CC and Odds Ratio. Based on the highest feature score, the



```
OMS scan results for MD5 hash:
------------------------------------------------------------------------------------------------
MD5 value: 5fcfe2ca8f6b8d93bda9b7933763002a
Online Malware Scanner (OMS)  scan date: 2016-07-02 06:34:51
VirusTotal engine detections: 37/55
Link to VirusTotal report:
https://www.virustotal.com/en/file/9d48503fa42f4184873fbc796a2fb64ad61eb9fcb7a1647a4830aceaf9911
d22/analysis/
------------------------------------------------------------------------------------------------
OMS scan results for SHA-1 hash:
------------------------------------------------------------------------------------------------
SHA-1 value: 581c0425386c44b6056b66dbe36d50aefd4ca724
Online Malware Scanner (OMS)  scan date: 2016-07-02 06:34:51
VirusTotal engine detections: 37/55
Link to VirusTotal report:
https://www.virustotal.com/en/file/9d48503fa42f4184873fbc796a2fb64ad61eb9fcb7a1647a4830aceaf9911
d22/analysis/
------------------------------------------------------------------------------------------------
OMS scan results for SHA-256 hash:
------------------------------------------------------------------------------------------------
SHA-256 value: 9d48503fa42f4184873fbc796a2fb64ad61eb9fcb7a1647a4830aceaf9911d22
Online Malware Scanner (OMS)  scan date: 2016-07-02 06:34:51
VirusTotal engine detections: 37/55
Link to VirusTotal report:
https://www.virustotal.com/en/file/9d48503fa42f4184873fbc796a2fb64ad61eb9fcb7a1647a4830aceaf9911
d22/analysis/
------------------------------------------------------------------------------------------------
```

**Fig. 5.** Snapshot of OMD for identification of malicious (not hidden) process *kelihos_dec.exe* from online malware scanner.



```
------------------------------------------------------------------------------------------------
Start of searching time for EFMTnkT7m-216.exe in Local Malware Database (LMD) is: 02-07
2016_17:50:42
MD5 Digest for EFMTnkT7m-216.exe is 0bf067750c7406cf3373525dd09c293c
EFMTnkT7m-216.exe is malicious with respect to md5 hash
End of 0bf067750c7406cf3373525dd09c293c Searching Time: 02-07-2016_17:50:42
Difference of 0bf067750c7406cf3373525dd09c293c Time: 2.50339508057e-05 seconds
------------------------------------------------------------------------------------------------

Start of searching time for EFMTnkT7m-216.exe in Local Malware Database (LMD) is: 02-07-
2016_17:58:09
SHA-1 Digest for EFMTnkT7m-216.exe is 791f81ebcdfbefec87706e0f57a728cbc9e311e8 EFMTnkT7m-
EFMTnkT7m-216.exe is malicious with respect to SHA-1 hash
End of 791f81ebcdfbefec87706e0f57a728cbc9e311e8 Searching Time: 02-07-2016_17:58:09
Difference of 791f81ebcdfbefec87706e0f57a728cbc9e311e8 Time: 1.00135803223e-05 seconds
------------------------------------------------------------------------------------------------
Start of searching time for EFMTnkT7m-216.exe in Local Malware Database (LMD) is: 02-07-
2016_17:59:52
SHA-256 Digest for EFMTnkT7m-216.exe is
262058ea75f3ce8c666977449791bbff87096144de755e38e63872de744eae36
EFMTnkT7m-216.exe is malicious with respect to SHA-256 End of
262058ea75f3ce8c666977449791bbff87096144de755e38e63872de744eae36
Searching Time: 02-07-2016_17:59:52
Difference of 262058ea75f3ce8c666977449791bbff87096144de755e38e63872de744eae36 Time:
8.82148742676e-06 seconds
------------------------------------------------------------------------------------------------
```

**Fig. 6.** Snapshot of OMD for the detection of known malware by cross-checking with LMSD.

length (L) of the topmost features 250, 500, and 750 are selected to verify which feature length achieves more accuracy. The symbol L represents the number of topmost features selected based on the highest feature score from each class separately used to generate FFV. Finally, a training file is built using the FFV with the N-gram files corresponding to the training samples. Lastly, the classifier is trained using the constructed training file.

Next, during the *testing phase*, executables excluded from the training set are used to construct a testing file. To evaluate the testing phase, binary files belong to Set-B are executed on a live Windows Monitored VM in different experimental scenarios for each type of malware and benign samples, and finally, those executables are semantically detected (VMI perspective) and forensically extracted at the VMM level are in dubious form (as discussed in Section 4.1). The reconstructed hidden and dubious executables are first parsed and then representative feature vector is extracted as a training instance. Based on this feature vector, the classifier categorizes the testing file as either benign or malware in real time at the hypervisor.

The overall malware detection rate of the OFMC is measured in the testing phase by following the $K/N$-fold cross-validation approach. Here, the independent dataset is randomly divided into $N$ equal-sized subparts (samples). Out of these N subparts, a single subpart is retained as validation data, and the remaining $N - 1$ subparts are used as training data. The cross-validation process is reiterated $N$ times (i.e., N-folds) and the final results are presented as an average of all the folds. This approach helps in systematically evaluating the robustness of our OFMC to detect and classify unknown malware from extracted executables.

**Table 5**
Confusion matrix.

| Class | Predicted malware | Predicted benign |
|---|---|---|
| Malware | True Positive(TP) | False Negative(FN) |
| Benign | False Positive(FP) | True Negative(TN) |

## 5.6. Evaluation metrics

The classifier detection performance can be measured by computing the difference between the predicted class for a given input and the actual class that the input belongs to. For instance, if the test input data is of the benign class and the classifier predicts it as benign, then, it is a correct classification. To quantify the detection performance of the classifier, the $2 \times 2$ confusion matrix is used (shown in Table 5) as it provides all the possible outcomes of a prediction and has the form TP, TN, FP, and FN of the classifier. The detection of the classifier is considered as TP when a malware file is properly identified as a malware otherwise, it is treated as FN. Any benign file classified as malware is treated as FP otherwise, it is measured as TN. Six performance metrics such as True Positive Rate (TPR), FPR, Precision, Recall, Accuracy, and F-measure were used in this work to measure the performance of the classifier as shown in Eq. (6). Finally, the weighted average result of the performance metrics was considered.

$$\text{TPR} = \frac{\text{TP}}{(\text{TP}+\text{FN})};\ \text{FPR} = \frac{\text{FP}}{(\text{FP}+\text{TN})}$$

$$\text{Precision} = \frac{\text{TP}}{(\text{TP}+\text{FP})};\ \text{Recall} = \frac{\text{TP}}{(\text{TP}+\text{FN})}$$

$$\text{Accuracy} = \frac{\text{TP}+\text{TN}}{(\text{TP}+\text{TN}+\text{FP}+\text{FN})}$$

$$F-\text{Measure} = 2 * \left(\frac{\text{Precision} * \text{Recall}}{\text{Precision} + \text{Recall}}\right) \tag{6}$$

### 5.6.1. *Results analysis*

Six popular machine learning technique such as Logistic Regression, Random Forest, Naives Bayes, Random Tree, and Sequential Minimal Optimization (SMO), and, J48 were used to measure the effectiveness of the proposed approach individually. All the chosen classifiers were initially trained with the default parameters available in the WEKA. Next, during the evaluation of each classifier, we selected three different FFV of length $L = 250$, $L = 500$, and $L = 750$ were selected from the training set to train the classifier. The evaluation performed by following the 10-fold cross-validation. It randomly splits the original input file into 10 equal subparts, where 9 subparts are used as the training dataset and the remaining 1 subpart is used as the validation data to measure the detection efficiency of the classifier. The cross-validation process is reiterated 10 times (the folds) for every combination with the condition that each subpart is used once as testing data. Finally, the outcome of each fold is averaged to estimate the overall efficiency of the classifier. The same steps are repeated separately for different FFVs of different sizes from two different feature selection techniques. This approach helps in systematically evaluate the feasibility of our proposed OFMC of the AMMDS to measure detection and the classification accuracy of malware from semantically reconstructed dubious executables at the VMM.

In the initial experiments, we observed that the Random Forest classifier achieved the highest accuracy on all three different L for two feature selection techniques, as compared to the other classifiers. Fig. 7 and Table 6 provides details of the malware detection accuracy and TPR and FPR achieved by different classifiers for 10-fold cross-validation evaluation. In particular, the Random Forest classifier achieved an accuracy of 99.75% with 0.003 FPR, 99.83% with 0.002 FPR, and 100% with 0 FPR for $L = 250$, $L = 500$ and $L = 750$ features respectively for suggested feature of NGL feature selection technique. Similarly, the same classifier performs pretty well on the features that are suggested by Odds Ratio feature selection technique while yielding an accuracy of 99.78% with 0.002 FPR, 99.87% with 0.001 FPR and, 100% with 0 FPR for $L = 250$, $L = 500$, and $L = 750$ respectively. The main reason to achieve better accuracy by the Random Forest classifier is that, it uses multiple decision trees that are randomly chosen to vote for overall classification of the given input file, where each decision tree classifies the new instance of features with a majority of the vote [81]. In this work, the Random Forest classifier achieved highest accuracy under the default parameters (tree size, $T = 10$), where, $T$ represents the number of decision trees in the ensemble.

It can also be seen from Fig. 7 is that the second highest accuracy yielded is by Logistic regression classifier ranging from 99.71% to 99.82% followed by Navies Bayes, Random Tree, SMO and J48 classifiers for $L = 250$, $L = 500$ and $L = 750$ feature of NGL based feature selection techniques. Similarly, for Odds Ratio based feature selection technique, Navies Bayes achieved second highest accuracy ranging from 99.55% to 99.75% followed by Logistic Regression, SMO, Random Tree, and J48 classifiers for $L = 250$, $L = 500$, and $L = 750$ respectively.

The performance of the each classifier was also evaluated using other performance metrics such as Precision, Recall, F-Measure, and Receiver Operating Curve (ROC) area separately for the both the feature section techniques. Fig. 8(a), (b), and (c) represents different performance metric results for three different feature lengths 250, 500, and 750, which are selected based on the highest feature score recommended by NGL CC selection technique. We notice that the maximum detection proficiency of the malware was achieved by random forest classifier for $L = 750$ with 0.998, 0.999, 1, 1 of precision, Recall, F-Measure and ROC respectively. Similarly, Fig. 8(d), (e), and (f) represents performance metric results of all classifier for $L = 250$, $L = 500$, and $L = 750$ as recommended by the Odds Ratio feature selection technique. It can also be notice that, for $L = 750$ of Odds Ratio feature selection techniques the same Random forest classifier yielded 0.998, 1, 0.999, 1 of precision, Recall, F-Measure and ROC respectively. Furthermore, the malware detection performance of all other classifiers are depicted in Fig. 8 with appropriate feature lengths of feature selection techniques. The lower accuracy shown by the J48 classifier for different feature length of both NGL CC and Odds Ratio feature selection techniques.

## 5.7. *Performance overhead*

In order to measure performance overhead induced by the AMMDS, a series of tests were performed by running the PCMark05 industry standard benchmark suite on Windows XP SP3 Monitored VM. Tests such as the Memory, HDD, and CPU of the PCMark05 were executed separately on the guest OS in two different settings. Each test was reiterated five times and an average five-time execution of each test was considered. In the first setting, the AMMDS was deactivated (not functioning), and in the second setting, the AMMDS was activated (running). Fig. 9 represents the overall performance overhead caused by AMMDS. Tests such as File Decryption, HDD-Text Startup, and HDD-File-Write induced maximum performance overheads of 4.9%, 5.8%, and 4.6%, while other test performance overheads were less than 4.5% on the Windows XP SP3 Monitored VM. These were noticed when the AMMDS abstracted the process semantic view during explicit detection of the hidden, dead, and dubious processes and pause and perform acquisition of the memory dump of the live Monitored VM.

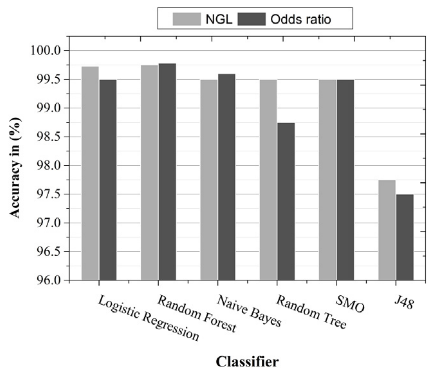
## 6. Discussion

The current development of the proposed AMMDS included extended functionalities for detection and estimation of the symptoms of malware execution on the live Monitored VM. In addition, the incorporation of machine learning techniques emphasized as the first scientific in-guest assisted VMI introspection technique to precisely detect and classify the running processes on the Monitored VM as benign or malicious at the VMM. The AMMDS performed this task from the semantically reconstructed executables that were introspected and forensically extracted at the VMM. The current demonstration of this approach is specific to the Windows guest OS to automatically detect the execution of large malware on the live Monitored VM by eliminating manual analysis.

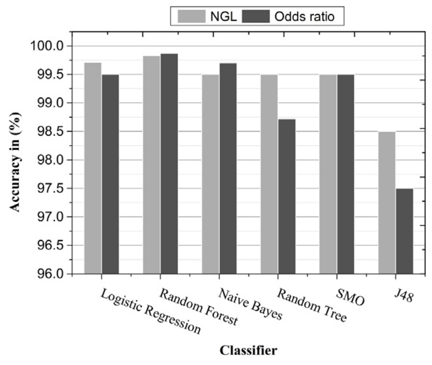
The AMMDS successfully addresses all the challenges discussed in Section 1. The different categories of real world malware executables used in this work include self-hidden behavior malware, which hides on execution on the guest OS. In addition, we also used both user-mode and kernel-mode rootkits to explicitly hide

**Table 6**
TPR and FPR of different classifiers on different feature length.

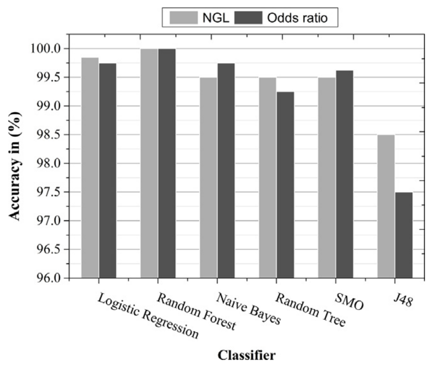
| Feature Length | | $L = 250$ | | $L = 500$ | | $L = 750$ | |
|---|---|---|---|---|---|---|---|
| Classifier | Metrics | NGL CC | Odds ratio | NGL CC | Odds ratio | NGL CC | Odds ratio |
| Logistic Regression | TPR | 0.998 | 0.995 | 0.998 | 0.998 | 0.998 | 0.998 |
| | FPR | 0.002 | 0.005 | 0.002 | 0.002 | 0.002 | 0.002 |
| Random Forest | TPR | 0.998 | 0.997 | 0.998 | 0.998 | 1 | 1 |
| | FPR | 0.003 | 0.002 | 0.002 | 0.001 | 0 | 0 |
| Naives Bayes | TPR | 0.995 | 0.995 | 0.995 | 0.995 | 0.995 | 0.998 |
| | FPR | 0.005 | 0.005 | 0.005 | 0.005 | 0.005 | 0.002 |
| Random Tree | TPR | 0.995 | 0.988 | 0.995 | 0.99 | 0.995 | 0.993 |
| | FPR | 0.005 | 0.012 | 0.005 | 0.01 | 0.005 | 0.007 |
| SMO | TPR | 0.995 | 0.995 | 0.995 | 0.995 | 0.995 | 0.998 |
| | FPR | 0.005 | 0.005 | 0.005 | 0.005 | 0.005 | 0.002 |
| J48 | TPR | 0.978 | 0.975 | 0.985 | 0.975 | 0.985 | 0.975 |
| | FPR | 0.022 | 0.025 | 0.015 | 0.025 | 0.015 | 0.025 |



(a) $L = 250$.

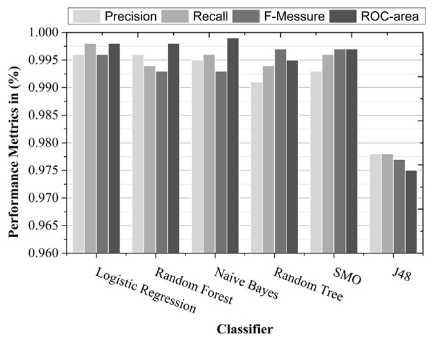


(b) $L = 500$.

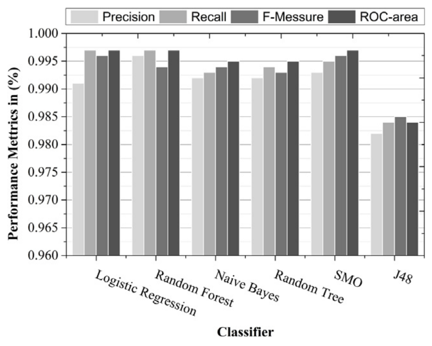


(c) $L = 750$.

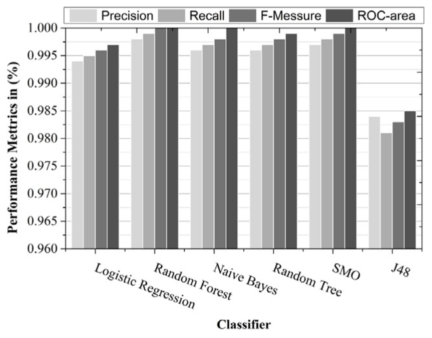
**Fig. 7.** Malware detection accuracy achieved by different classifiers based on NGL CC and Odds Ratio feature selection techniques for three different feature length.



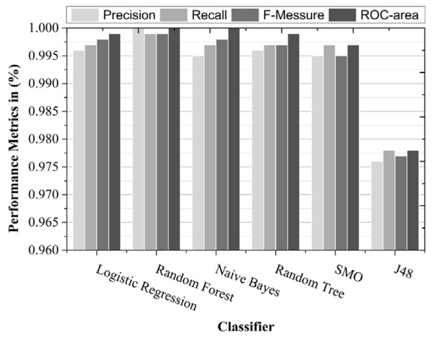
(a) $L = 250$.



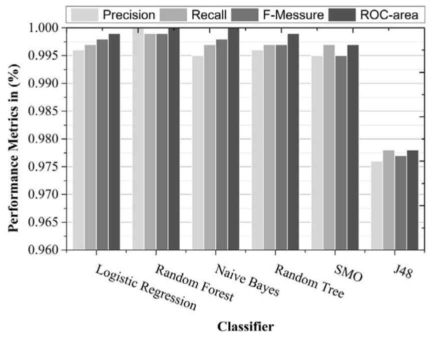
(b) $L = 500$.



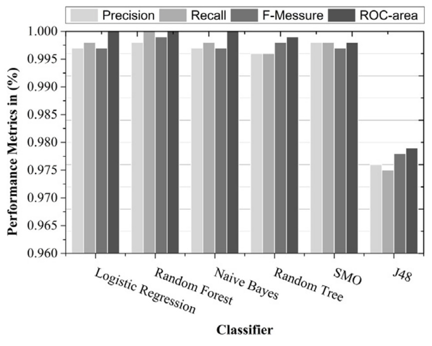
(c) $L = 750$.



(d) $L = 250$.



(e) $L = 500$.



(f) $L = 750$.

**Fig. 8.** Comparison of performance of the classifier under different performance metrics for the different feature length recommended by NGL CC and Odds Ratio feature selection techniques.

**Table 7**
Comparison of results with other VMM-based and non-introspection based malware detection approaches that used machine learning techniques.

| Related work | Feature types | Approaches | Accuracy | FPR | VMM |
|---|---|---|---|---|---|
| Masud et al. [50] | N-gram | SVM, Boosted J48 | 96.88% | □ | × |
| Shabtai et al. [53] | Opcode | SVM, LR, RF, ANN, DT, BDT, NB, BNB | >96% | 0.02 | × |
| Bai and Wang [54] | Byte N-gram + Opcode N-gram + Format feature | J48, RF, AdboostM1(J48), Bagging(J48) | 99% | 0 | × |
| Zhanget et al. [67] | Multi-feature | XGBoost, ExtraTreeClassifier, GradientBoost | 99.72% | □ | × |
| Bai and Wang [54] | N-gram, opcode | J48, RF, AdboostM1(J48), Bagging(J48) | 99% | 0 | × |
| Huda et al. [64] | String feature | RF, SVM, J48, NB and IB | 100% | 0 | × |
| Watson et al. [65] | Statistical meta-features | One-class SVM | >90% | □ | √ |
| Proposed work | N-gram | LR, RF, NB, RT, SMO, J48 | 100% | 0 | √ |

Logistic Regression (LR), Random Forest (RF), Artificial Neural Networks (ANN), Decision Trees (DT), Boosted Decision Trees (BDT) Naive Bayes (NB), Boosted Naive Bayes (BNB), Random Tree (RT), Simple Logistic (SL), □: indicates information not available in the previous work.

**Table 8**
Comparison of AMMDS with previous VMI-based malware detection approaches.

| Functionality | Lycosid [25] | Lare [33] | VMwatcher [20] | Process-out-grafting [22] | SYRINGE [34] | AMMDS |
|---|---|---|---|---|---|---|
| Hidden process detection | √ | □ | √ | □ | □ | √ |
| Time synchronization | × | × | × | × | × | √ |
| Incorporation of MFA | × | × | × | × | × | √ |
| Malicious check on hidden and dubious process | × | × | × | × | × | √ |
| Guest-assisted | × | √ | × | √ | √ | √ |
| Manual analysis | √ | □ | √ | □ | □ | × |
| Fully automated | × | × | × | × | □ | √ |
| Machine learning techniques | × | × | × | × | × | √ |
| Performance overhead | 6% | Varied | □ | Varied | Varied | 5.8% |

□: Information are not explicitly mentioned in the previous work

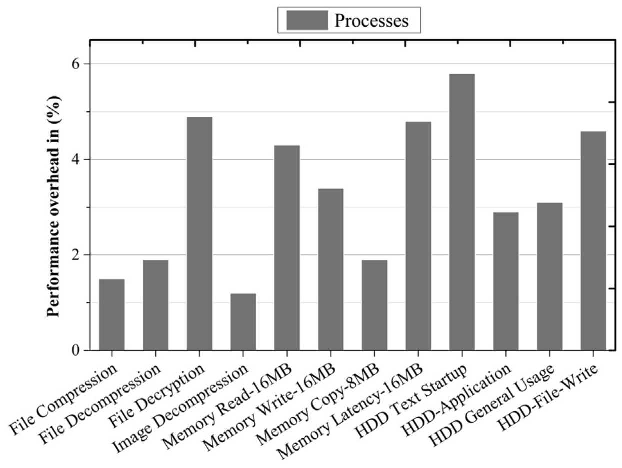


**Fig. 9.** Performance overhead of AMMDS on PCMark05 in detecting hidden and dubious state information of Monitored VM.

some benign and malicious running processes to test the detection feasibility of our proposed AMMDS. We practically confirmed that VMI introspector of the AMMDS as being proficient in detecting hidden and malicious processes caused by stealthy malware and rootkits by traversing the semantic view of the _EPROCESS data structure of the live Monitored VMs.

## 6.1. Comparison with existing work

In order to highlight the significance of our proposed AMMDS, a comparison was undertaken in two phases. In the first phase, the extended functionality of the AMMDS was compared with other previous in-guest assisted and out-of-VM VMM-based malware detection techniques. Table 8 we can see that the AMMDS is able to detect and estimate the symptoms of malware with enhanced functionality, which was not addressed in previous approaches. Furthermore, functionalities such as the incorporation of MFA with VMI, malicious check on detected hidden and dubious process at the VMM, and fully automated and leveraging machine learning techniques for detection of known and unknown malware were not presented in any of the previous VMI-based relevant approaches.

In the second phase, the systematic evaluation of the proposed AMMDS was compared with other VMM-based introspection and non-introspection malware detection and classification approaches that used machine learning techniques. Table 7 summarizes the comparison of the AMMDS with other related works. To the best of our knowledge, except Watson et al. [65], none of the VMM-based malware detection approaches used machine learning techniques to detect malware. In particular, none of the VMI-based malware detection approaches used the machine learning techniques to detect and classify malware on semantically reconstructed high-level state information of live introspected system from VMI perspective. Authors [65] proposed VMM-based malware detection and classification system using one-class SVM machine learning technique. The vectorial (feature) representation of this approach not only considered the features related to the processes, but also the network activity of the introspected VM as statistical meta-features. In addition, this was not compared with any of the public benchmarked datasets to quantify the classification accuracy of their system. Maximum malware detection accuracy of this work was highlighted as more than 90%. Our proposed AMMDS achieved up to an accuracy of 100% with 0 FPR on the generated dataset at VMM-level.

To substantiate the effectiveness of the results, the proposed AMMDS was also compared with non-VMM or non-introspection based malware detection approaches. As a number of static and dynamic (non-introspection) based works are presented in the literature, we have chosen a relevant research with principal targets as feature extraction type and 96% or above accuracy and FPR. Masud et al. [50] proposed hybrid feature selection technique to detect malicious portable executables. The construction of the hybrid feature set was based on the N-grams of the executables feature, assembly instructions, and a dynamic link library. Overall, this approach achieved 96.88% accuracy.

Additionally, Shabtai et al. [53] used an opcode based feature extraction methods to detect unknown malware using several machine learning classifiers. Their approach was evaluated by considering a large number of benign and malware datasets and achieved greater than 96% of detection accuracy. Recently, in [54] authors used Bytecode N-gram, opcode N-gram, and format features as multi-view features to detect unseen malware using three ensemble learning methods. They used two static malware datasets to validate the proposed classification methodologies with 0% false alarm rate.

Similarly, Zhanget et al. [67] have proposed a lightweight malware classification system using ensemble tree-based XGBoost, ExtraTree, and GradientBoost as machine learning classification algorithms to detect new and real-world malware. They combined multiple categories of features that were extracted from malicious executables whereby the proposed approach would work even on obfuscated and packed malware sample of different families. The authors experimented their approach with Microsoft provided "Kaggle" malware challenge dataset and achieved 99.72% detection accuracy of malware.

In order to protect CPS against new malware variants recently, Huda et al. [64] proposed semi-supervised approach. They have used four supervised machine learning classifiers based on the static and dynamic (run-time) malware executables feature to evaluate classification methodologies. Finally, this approach yielded up to 100% accuracy with zero FPR as similar to our proposed approach. Some of the non-introspection based related work mentioned in Table 7 achieved equivalent accuracy than our proposed AMMDS. However, our approach is unique for the detection of unknown malware from a VMI perspective at the VMM with 100% detection accuracy.

## 7. Conclusion and future work

In this work, we have presented the design, implementation, and evaluation of the AMMDS as an advanced VMI-based guest assisted out-of-VM security solution that leverages both VMI and MFA techniques to estimate symptoms of malware execution and also able to accurately detect unknown malware (malicious executables) running on the CPS-based Monitored VM. The OMD of the AMMDS is able to recognize known malware whereas the OFMC is capable of detecting and classifying unknown malware by using machine learning techniques. The proposed AMMDS extensively reduces the manual effort required to accurately identify the malware from the semantically reconstructed and forensically extracted executables as compared to other existing VMI and MFA based out-of-VM approaches. Finally, the AMMDS was evaluated against a large number of real-world Windows malware as well as benign executables to measure the malware detection rate. Our empirical results demonstrate that AMMDS is capable of recognizing malware with an accuracy of 100%. Further, the observed experimental results showed that the maximum performance of overhead induced by the AMMDS is 5.8% under evaluation of the Windows benchmark suite.

In future work, we aim to evaluate the AMMDS for the Linux-based operating system to evaluate its detection rate against the propagation of the sophisticated Linux malware.

## References

[1] Z. Chen, G. Xu, V. Mahalingam, L. Ge, J. Nguyen, W. Yu, C. Lu, A cloud computing based network monitoring and threat detection system for critical infrastructures, Big Data Res. 3 (2016) 10–23.

[2] J. Ma, K.-K.R. Choo, H.-h. Hsu, Q. Jin, W. Liu, K. Wang, Y. Wang, X. Zhou, Perspectives on cyber science and technology for cyberization and cyber-enabled worlds, in: Dependable, Autonomic and Secure Computing, 14th Intl Conf on Pervasive Intelligence and Computing, 2nd Intl Conf on Big Data Intelligence and Computing and Cyber Science and Technology Congress (DASC/PiCom/DataCom/CyberSciTech), 2016 IEEE 14th Intl C, IEEE, 2016, pp. 1–9.

[3] R. Chaâri, F. Ellouze, A. Koubâa, B. Qureshi, N. Pereira, H. Youssef, E. Tovar, Cyber-physical systems clouds: a survey, Comput. Netw. 108 (2016) 260–278.

[4] A.B. Nagarajan, F. Mueller, C. Engelmann, S.L. Scott, Proactive fault tolerance for HPC with Xen virtualization, in: Proceedings of the 21st Annual International Conference on Supercomputing, ACM, 2007, pp. 23–32.

[5] B. Jablkowski, O. Spinczyk, CPS-Xen: A virtual execution environment for cyber-physical applications, in: International Conference on Architecture of Computing Systems, Springer, 2015, pp. 108–119.

[6] B. Jablkowski, U.T. Gabor, O. Spinczyk, Evolutionary planning of virtualized cyber-physical compute and control clusters, J. Syst. Archit. 73 (2017) 17–27.

[7] Y.B. Reddy, Security and design challenges in cyber-physical systems, in: Information Technology-New Generations, ITNG, 2015, 12th International Conference on, IEEE, 2015, pp. 200–205.

[8] R. Moskovitch, Y. Elovici, L. Rokach, Detection of unknown computer worms based on behavioral classification of the host, Comput. Stat. Data Anal. 52 (9) (2008) 4544–4566.

[9] D. Lin, M. Stamp, Hunting for undetectable metamorphic viruses, J. Comput. Virol. 7 (3) (2011) 201–214.

[10] Goudey, Threat report: rootkits, https://www.microsoft.com/en-in/download/details.aspx?id=34797, 2012.

[11] M. Pearce, S. Zeadally, R. Hunt, Virtualization: issues, security threats, and solutions, ACM Comput. Surv. (CSUR) 45 (2) (2013) 17.

[12] T. Garfinkel, M. Rosenblum, et al., A virtual machine introspection based architecture for intrusion detection, in: NDSS, vol. 3, 2003, pp. 191–206.

[13] B. Dolan-Gavitt, T. Leek, M. Zhivich, J. Giffin, W. Lee, Virtuoso: narrowing the semantic gap in virtual machine introspection, in: 2011 IEEE Symposium on Security and Privacy, IEEE, 2011, pp. 297–312.

[14] B. Jain, M.B. Baig, D. Zhang, D.E. Porter, R. Sion, Sok: introspections on trust and the semantic gap, in: 2014 IEEE Symposium on Security and Privacy, IEEE, 2014, pp. 605–620.

[15] Y. Fu, Z. Lin, Bridging the semantic gap in virtual machine introspection via online kernel data redirection, ACM Trans. Inform. Syst. Sec. (TISSEC) 16 (2) (2013) 7.

[16] A. Saberi, Y. Fu, Z. Lin, HYBRID-BRIDGE: Efficiently bridging the semantic gap in virtual machine introspection via decoupled execution and training memoization, in: Proceedings of the 21st Annual Network and Distributed System Security Symposium, NDSS14, 2014.

[17] M. Garnaeva, Kelihos/Hlux botnet returns with new techniques, Securelist, http://www.Securelist.Com/En/Blog/655/Kelihos_Hlux_botnet_Returns_with_new_techniques, 2012.

[18] M. Sharif, A. Lanzi, J. Giffin, W. Lee, Automatic reverse engineering of malware emulators, in: Security and Privacy, 2009, 30th IEEE Symposium on, IEEE, 2009, pp. 94–109.

[19] U. Bayer, I. Habibi, D. Balzarotti, E. Kirda, C. Kruegel, A view on current malware behaviors, in: LEET, 2009.

[20] X. Jiang, X. Wang, D. Xu, Stealthy malware detection through vmm-based out-of-the-box semantic view reconstruction, in: Proceedings of the 14th ACM Conference on Computer and Communications Security, ACM, 2007, pp. 128–138.

[21] A. Dinaburg, P. Royal, M. Sharif, W. Lee, Ether: malware analysis via hardware virtualization extensions, in: Proceedings of the 15th ACM Conference on Computer and Communications Security, ACM, 2008, pp. 51–62.

[22] D. Srinivasan, Z. Wang, X. Jiang, D. Xu, Process out-grafting: an efficient out-of-vm approach for fine-grained process execution monitoring, in: Proceedings of the 18th ACM Conference on Computer and Communications Security, ACM, 2011, pp. 363–374.

[23] J. Rhee, R. Riley, D. Xu, X. Jiang, Defeating dynamic data kernel rootkit attacks via vmm-based guest-transparent monitoring, in: Availability, Reliability and Security, 2009, ARES'09. International Conference on, IEEE, 2009, pp. 74–81.

[24] S.T. Jones, A.C. Arpaci-Dusseau, R.H. Arpaci-Dusseau, Antfarm: tracking processes in a virtual machine environment, in: USENIX Annual Technical Conference, General Track, 2006, pp. 1–14.

[25] S.T. Jones, A.C. Arpaci-Dusseau, R.H. Arpaci-Dusseau, VMM-based hidden process detection and identification using lycosid, in: Proceedings of the Fourth ACM SIGPLAN/SIGOPS International Conference on Virtual Execution Environments, ACM, 2008, pp. 91–100.

[26] Y.-M. Wang, D. Beck, B. Vo, R. Roussev, C. Verbowski, Detecting stealth software with strider ghostbuster, in: 2005 International Conference on Dependable Systems and Networks, DSN'05, IEEE, 2005, pp. 368–377.

[27] A. Shevchenko, The evolution of self-defense technologies in malware, Available from Webpage: http://www.Viruslist.Com/Analysis, 2007.

[28] M. Saleh, E.P. Ratazzi, S. Xu, Instructions-Based detection of sophisticated obfuscation and packing, in: 2014 IEEE Military Communications Conference, IEEE, 2014, pp. 1–6.

[29] B.D. Payne, M. De Carbone, W. Lee, Secure and flexible monitoring of virtual machines, in: Computer Security Applications Conference, ACSAC 2007, Twenty-Third Annual, IEEE, 2007, pp. 385–397.

[30] L. Litty, H.A. Lagar-Cavilla, D. Lie, Hypervisor support for identifying covertly executing binaries, in: USENIX Security Symposium, 2008, pp. 243–258.

[31] Z. Gu, Z. Deng, D. Xu, X. Jiang, Process implanting: a new active introspection framework for virtualization, in: Reliable Distributed Systems, SRDS, 2011, 30th IEEE Symposium on, IEEE, 2011, pp. 147–156.
[32] Y. Fu, J. Zeng, Z. Lin, HYPERSHELL: a practical hypervisor layer guest OS shell for automated in-VM management, in: 2014 USENIX Annual Technical Conference, USENIX ATC 14, 2014, pp. 85–96.
[33] B.D. Payne, M. Carbone, M. Sharif, W. Lee, Lares: an architecture for secure active monitoring using virtualization, in: 2008 IEEE Symposium on Security and Privacy, Sp 2008, IEEE, 2008, pp. 233–247.
[34] M. Carbone, M. Conover, B. Montague, W. Lee, Secure and robust monitoring of virtual machines through guest-assisted introspection, in: International Workshop on Recent Advances in Intrusion Detection, Springer, 2012, pp. 22–41.
[35] B. Hay, K. Nance, Forensics examination of volatile system data using virtual introspection, Oper. Syst. Rev. 42 (3) (2008) 74–82.
[36] A. Case, L. Marziale, G.G. Richard, Dynamic recreation of kernel data structures for live forensics, Digital Invest. 7 (2010) S32–S40.
[37] R. Poisel, E. Malzer, S. Tjoa, Evidence and cloud computing: the virtual machine introspection approach, JoWUA 4 (1) (2013) 135–152.
[38] B. Dolan-Gavitt, B. Payne, W. Lee, Leveraging forensic tools for virtual machine introspection, 2011.
[39] B. Schatz, Bodysnatcher: Towards reliable volatile memory acquisition by software, Digital Invest. 4 (2007) 126–134.
[40] J. Stüttgen, M. Cohen, Anti-forensic resilient memory acquisition, Digital Invest. 10 (2013) S105–S115.
[41] M. Yu, Q. Lin, B. Li, Z. Qi, H. Guan, Vis: virtualization enhanced live acquisition for native system, in: Proceedings of the Second Asia-Pacific Workshop on Systems, ACM, 2011, p. 13.
[42] X. Zhong, C. Xiang, M. Yu, Z. Qi, H. Guan, A virtualization based monitoring system for mini-intrusive live forensics, Int. J. Parallel Program. 43 (3) (2015) 455–471.
[43] L. Martignoni, A. Fattori, R. Paleari, L. Cavallaro, Live and trustworthy forensic analysis of commodity production systems, in: International Workshop on Recent Advances in Intrusion Detection, Springer, 2010, pp. 297–316.
[44] Y. Cheng, X. Fu, X. Du, B. Luo, M. Guizani, A lightweight live memory forensic approach based on hardware virtualization, Inform. Sci. (2016).
[45] M.G. Schultz, E. Eskin, F. Zadok, S.J. Stolfo, Data mining methods for detection of new malicious executables, in: Security and Privacy, S&P 2001. Proceedings, 2001 IEEE Symposium on, IEEE, 2001, pp. 38–49.
[46] M.Z. Shafiq, S.M. Tabish, F. Mirza, M. Farooq, Pe-miner: mining structural information to detect malicious executables in realtime, in: International Workshop on Recent Advances in Intrusion Detection, Springer, 2009, pp. 121–141.
[47] A. Hellal, L.B. Romdhane, Minimal contrast frequent pattern mining for malware detection, Comput. Secur. 62 (2016) 19–32.
[48] M. Ahmadi, D. Ulyanov, S. Semenov, M. Trofimov, G. Giacinto, Novel feature extraction, selection and fusion for effective malware family classification, in: Proceedings of the Sixth ACM Conference on Data and Application Security and Privacy, ACM, 2016, pp. 183–194.
[49] J.Z. Kolter, M.A. Maloof, Learning to detect and classify malicious executables in the wild, J. Mach. Learn. Res. 7 (Dec) (2006) 2721–2744.
[50] M.M. Masud, L. Khan, B. Thuraisingham, A scalable multi-level feature extraction technique to detect malicious executables, Inform. Syst. Front. 10 (1) (2008) 33–45.
[51] N. Nissim, R. Moskovitch, L. Rokach, Y. Elovici, Detecting unknown computer worm activity via support vector machines and active learning, Patt. Anal. Appl. 15 (4) (2012) 459–475.
[52] I. Santos, F. Brezo, X. Ugarte-Pedrero, P.G. Bringas, Opcode sequences as representation of executables for data-mining-based unknown malware detection, Inform. Sci. 231 (2013) 64–82.
[53] A. Shabtai, R. Moskovitch, C. Feher, S. Dolev, Y. Elovici, Detecting unknown malicious code by applying classification techniques on opcode patterns, Security Inform. 1 (1) (2012) 1.
[54] J. Bai, J. Wang, Improving malware detection using multi-view ensemble learning, Security Commun. Netw. 9 (17) (2016) 4227–4241.
[55] A. Moser, C. Kruegel, E. Kirda, Limits of static analysis for malware detection, in: Computer Security Applications Conference, ACSAC 2007, Twenty-Third Annual, IEEE, 2007, pp. 421–430.
[56] Z. Shan, X. Wang, Growing grapes in your computer to defend against malware, IEEE Trans. Inform. Foren. Secur. 9 (2) (2014) 196–207.
[57] C. Willems, T. Holz, F. Freiling, Toward automated dynamic malware analysis using cwsandbox, IEEE Secur. Priv. 5 (2) (2007).
[58] K. Rieck, P. Trinius, C. Willems, T. Holz, Automatic analysis of malware behavior using machine learning, J. Comput. Secur. 19 (4) (2011) 639–668.
[59] E. Menahem, A. Shabtai, L. Rokach, Y. Elovici, Improving malware detection by applying multi-inducer ensemble, Comput. Statist. Data Anal. 53 (4) (2009) 1483–1494.
[60] F. Shahzad, M. Shahzad, M. Farooq, In-execution dynamic malware analysis and detection by mining information in process control blocks of Linux OS, Inform. Sci. 231 (2013) 45–63.
[61] Q. Miao, J. Liu, Y. Cao, J. Song, Malware detection using bilayer behavior abstraction and improved one-class support vector machines, Internat. J. Inform. Secur. 15 (4) (2016) 361–379.
[62] R. Islam, R. Tian, L.M. Batten, S. Versteeg, Classification of malware based on integrated static and dynamic features, J. Netw. Comput. Appl. 36 (2) (2013) 646–656.
[63] A. Kumar, K. Kuppusamy, G. Aghila, A learning model to detect maliciousness of portable executable using integrated feature set, J. King Saud University-Computer and Information Sciences (2017).
[64] S. Huda, S. Miah, M.M. Hassan, R. Islam, J. Yearwood, M. Alrubaian, A. Almogren, Defending unknown attacks on cyber-physical systems by semi-supervised approach and available unlabeled data, Inform. Sci. 379 (2017) 211–228.
[65] M.R. Watson, A.K. Marnerides, A. Mauthe, D. Hutchison, et al., Malware detection in cloud computing infrastructures, IEEE Trans. Dependable Secure Comput. 13 (2) (2016) 192–205.
[66] T.K. Lengyel, S. Maresca, B.D. Payne, G.D. Webster, S. Vogl, A. Kiayias, Scalability, fidelity and stealth in the drakvuf dynamic malware analysis system, in: Proceedings of the 30th Annual Computer Security Applications Conference, ACM, 2014, pp. 386–395.
[67] Y. Zhang, Q. Huang, X. Ma, Z. Yang, J. Jiang, Using multi-features and ensemble learning method for imbalanced malware classification, in: Trustcom/BigData SE/ISPA, 2016 IEEE, IEEE, 2016, pp. 965–973.
[68] Intel, Intel trusted exceution technology, Accessed on September 2016, http://www.intel.com/technology/security/, 2016.
[69] R. Wojtczuk, J. Rutkowska, Attacking intel trusted execution technology, Black Hat DC, 2009.
[70] R. Wojtczuk, Subverting the xen hypervisor, Black Hat USA, 2008.
[71] M.E. Russinovich, D.A. Solomon, A. Ionescu, Windows internals, Pearson Education, 2012.
[72] J. Lamps, I. Palmer, R. Sprabery, WinWizard: expanding xen with a LibVMI intrusion detection tool, in: Cloud Computing (CLOUD), 2014 IEEE 7th International Conference on, IEEE, 2014, pp. 849–856.
[73] M. Pietrek, Peering inside the pe: a tour of the win32 (r) portable executable file format, Microsoft Syst. J.-US Edition (1994) 15–38.
[74] M.H. Ligh, A. Case, J. Levy, A. Walters, The Art of Memory Forensics: Detecting Malware and Threats in Windows, Linux, and Mac Memory, John Wiley & Sons, 2014.
[75] D.K.S. Reddy, A.K. Pujari, N-gram analysis for computer virus detection, J. Comput. Virol. 2 (3) (2006) 231–239.
[76] K. Dave, Study of feature selection algorithms for text-categorization, 2011.
[77] D. Mladenic, M. Grobelnik, Feature selection for unbalanced class distribution and naive bayes, in: ICML, vol. 99, 1999, pp. 258–267.
[78] M. Hall, E. Frank, G. Holmes, B. Pfahringer, P. Reutemann, I.H. Witten, The weka data mining software: an update, ACM SIGKDD Explor. Newslet. 11 (1) (2009) 10–18.
[79] M.I. Sharif, W. Lee, W. Cui, A. Lanzi, Secure in-vm monitoring using hardware virtualization, in: Proceedings of the 16th ACM Conference on Computer and Communications Security, ACM, 2009, pp. 477–487.
[80] M. Valipour, Optimization of neural networks for precipitation analysis in a humid region to detect drought and wet year alarms, Meteorol. Appl. 23 (1) (2016) 91–100.
[81] T.M. Oshiro, P.S. Perez, J.A. Baranauskas, How many trees in a random forest?in: International Workshop on Machine Learning and Data Mining in Pattern Recognition, Springer, 2012, pp. 154–168.

**Ajay Kumara M.A.** received the B.E and M.Tech degrees in computer science and engineering from Visvesvaraya Technological University, Karnataka, India, in 2009, and 2012, respectively. He is currently pursuing the Ph.D. degree with the Department of Information Technology, National Institute of Technology, Karnataka, Surathkal. His research interests include virtualization security and cloud computing, dynamic malware analysis, memory forensic and machine learning.

**Jaidhar C.D.** received Ph.D. degree in computer science and engineering from NIT Trichy Tamilnadu, in 2008. He is currently with department of Information Technology, National institute of Technology, Karnataka, Surathkal, India, where he is working an Assistant Professor, since 2013. His research interests include computer networks, cloud computing and virtualization, cryptography and network security, malware analysis, machine learning and Ad hoc networks.