Mathematical Biosciences and Engineering, 16(4): 2650–2667.
DOI: 10.3934/mbe.2019132
Received: 05 December 2018
Accepted: 13 March 2019
Published: 26 March 2019

http://www.aimspress.com/journal/MBE

Research article

TKRD: Trusted kernel rootkit detection for cybersecurity of VMs based on machine learning and memory forensic analysis

Xiao Wang[1,2], Jianbiao Zhang[1,2,*], Ai Zhang[3] and Jinchang Ren[4]

[1] Faculty of Information Technology, Beijing University of Technology, Beijing 100124, China
[2] Beijing Key Laboratory of Trusted Computing, Beijing 100124, China
[3] Department of Computer Science & Engineering, UC San Diego, CA, USA
[4] Department of Electronic and Electrical Engineering, University of Strathclyde, Glasgow, U.K.

* **Correspondence:** Email: zjb@bjut.edu.cn; Tel: +861067392873.

Abstract: The promotion of cloud computing makes the virtual machine (VM) increasingly a target of malware attacks in cybersecurity such as those by kernel rootkits. Memory forensic, which observes the malicious tracks from the memory aspect, is a useful way for malware detection. In this paper, we propose a novel TKRD method to automatically detect kernel rootkits in VMs from private cloud, by combining VM memory forensic analysis with bio-inspired machine learning technology. Malicious features are extracted from the memory dumps of the VM through memory forensic analysis method. Based on these features, various machine learning classifiers are trained including Decision tree, Rule based classifiers, Bayesian and Support vector machines (SVM). The experiment results show that the Random Forest classifier has the best performance which can effectively detect unknown kernel rootkits with an Accuracy of 0.986 and an AUC value (the area under the receiver operating characteristic curve) of 0.998.

Keywords: virtual machine; private cloud; kernel rootkit detection; memory forensic; machine learning

1. Introduction

With the popularization of cloud computing [1], cybersecurity has become a major concern [2,3]. According to the model of service, cloud computing can be divided into public, private and hybrid cloud. The private cloud is deployed for certain organizations with higher requirements in terms of performance

and security. Virtual machine (VM), as the main form providing service to users in the cloud environment, has increasingly become the target of cyber-attacks, leveraging malwares to comprise the VM system and leading to significant damages to the virtual and even physical platforms.

There are many kinds of malwares such as virus, worm, Trojan and backdoor. As one of the most dangerous malwares, kernel rootkit [4], also called Driver Trojan, obtains the system administrator privilege while hiding its existence in the OS kernel. Kernel rootkits modify the critical kernel data structures, making it much more difficult for detection than any other kinds of malwares. Traditional anti-virus solutions are based on the signatures of the already known malwares whilst installed in the same system with the malware. They can only detect known malwares and can be easily spotted and evaded by the smart malwares.

Automatic detection of known and unknown kernel rootkits on VMs is becoming an urgent issue. As the kernel rootkit is hard to detect in the user mode, we observe its tracks from the memory side. As a kind of digital investigation for detecting cybercrimes [5], memory forensic can effectively extract malicious features from the memory dumps of the monitored computer. It is especially suitable for kernel rootkits detection. In virtualization environment, the memory dump acquisition of the VMs can be conducted out of the monitored VM at the higher hypervisor level in a trusted way, so the memory acquisition operation cannot be discovered or subverted by malwares. Machine learning technology has proved effective at automatically detecting known and unknown malware in many researches [6–8]. We combine the machine learning technology with memory forensic analysis to form a novel kernel rootkit detection method TKRD for the VMs in private cloud.

The main contribution of this paper can be highlighted as follows:

- A novel TKRD method to automatically detect kernel rootkits in VMs of private cloud was proposed;
- The memory dumps of the VM were acquired from the controller node of the OpenStack platform in a trusted way;
- Memory forensic analysis along with the Volatility framework was leveraged to extract malicious features;
- Bio-inspired machine learning classifiers were trained for automatic malware detection;
- The experiment results showed that our methodology can effectively detect the known and unknown kernel rootkits for VMs.

The rest of the paper is organized as follows. Related works are introduced in Section 2. Section 3 presents the memory acquisition and machine learning methods used in our research. The experimental settings and results are discussed in Section 4, followed by some concluding remarks summarized in Section 5.

2. Related works

2.1. Rootkit

Rootkits are malwares allowing permanent or consistent, undetectable presence in a computer system [9]. Rootkits can hide specific system resources to achieve the goal of hiding the intrusion into the compromised computer. They are categorized into five classes: user mode rootkits, kernel mode rootkits, bootkits, hypervisor level rootkits and firmware rootkits. User mode rootkits run on Ring 3 of the x86 processor with the least privilege. Kernel mode rootkits execute in privileged mode

on Ring 0, making it very hard to detect. Bootkits are a variant of kernel mode rootkits, which can infect boot code such as Master Boot Record (MBR), Volume Boot Record (VBR), or boot sector. Hypervisor level rootkits exploit hardware virtualization features running in Ring -1 which hosts the VM operating system (OS). Firmware rootkits exist in the firmware of hardware such BIOS. As the integrity of firmware is not checked so frequently, it is an ideal place for firmware rootkits to hide. In this paper we focus on the kernel mode rootkits.

Rootkits leverage many techniques to subvert the OS. The flow of the program can be modified by installing hooks. For example, by manipulating function pointers in System Service Descriptor Table (SSDT), the transform from user mode API to system Native API is compromised. More advanced rootkits such as FU and Shadow Walker can launch Direct Kernel Object Manipulation (DKOM) attacks, which directly modify the core data structure of OS kernel in memory. Malicious library injection and code injection are also common means for rootkits to subvert the system.

2.2. Memory forensic for rootkit detection

Memory forensic belongs to the rank of digital forensics which collect and present digital evidence for cybercrime investigation. There are many researches on leveraging memory forensic to uncover the malicious behaviors in the memory.

Case et al. [10] presented a new kernel rootkit detection technique suitable for Mac OS X system. They used the most popular memory forensic framework Volatility to analyze the features of malwares. For each new rootkit detection technique, a Volatility plugin was developed. Evaluation of the plugins were done to illustrate their effectiveness.

For detecting malwares in Android, Yang et al. [11] proposed a general tool named "AMExtractor" for volatile memory acquisition for Android devices. Using memory forensic tools, the memory dumps acquired were further analyzed. Rootkits were successfully detected by memory forensic framework Volatility.

For malware detection in virtualization environment, Kumara et al. [12] leveraged memory forensic tool such as Volatility, Rekall to analyze the memory state of the VMs, which can address the semantic gap problem existing in Virtual Machine Introspection (VMI). LibVMI can also address the semantic gap problem, and eliminate memory dump acquire time, achieving a high efficiency of memory dump acquisition and analysis.

Using memory forensic technology, Hua et al. [13] designed and implemented a VMM(virtual machine monitor)-based hidden process detection system to investigate rootkits by identifying the lack of the critical process and the target hidden process from the aspect of memory forensics. They addressed the limitation that the tradition host-based detection tools could be deceived or subverted by malwares.

Tien et al. [14] introduced a memory data monitoring method against the running malware outside the VM, various features were observed from the memory. At last, they suggested combining memory forensic with heuristic method to promote the detection rate for malwares. From the above researches, memory forensic is an effective way to detect complex malwares, especially rootkits.

2.3. Machine learning method for malware detection

Machine learning technology has proved effective in automatically detecting known and unknown malwares in many researches. Cohen et al. [15] proposed a method for trusted detection of ransomware

in private cloud. Memory forensic framework Volatility was used to extract meta-features from the volatile memory dumps of the VM. These meta-features were used to train machine learning classifiers. The experiment results showed that the method can effectively detect unknown ransomware.

In order to automatically detect malware in cloud VMs, NirNissim et al. [16] conducted a research on memory forensic for cloud VM. System-call sequences were extracted from the memory dumps as features for malware detection. Various machine learning algorithms were leveraged to detect malwares in the cloud. They focused on the ransomware and RAT malware in IIS and Email server. The experiment results showed that the methodology can detect unknown ransomware and RAT in an average 97.9% true positive rate (TPR) and 0% false positive rate (FPR).

Kumara et al. [17] proposed an automated multi-level detection system for Rootkits and other malwares for VMs at hypervisor level. The detection system was composed of Online Malware Detector (OMD) and Offline Malware Classifier (OFMC). The OMD detected known malwares based on the local malware signature database. The OFMC classified the unknown malwares by adopting machine learning technique. The meta-data for classifier training was extracted by virtual machine introspection tools from the running VMs at the hypervisor level. The experiments achieved 100% Accuracy and zero FPR. Himanshu et al. [18] proposed an automated and real-time monitoring and forecasting system for rootkits and other malwares for the Windows virtualization architecture. It firstly extracted kernel data structures with VMI tool LibVMI. Then a prediction tool was developed using machine learning techniques. However, the authors did not tell which data structure to extract and how to train the machine learning classifier.

Mosli et al. [19] proposed an automated malware detection method using artifacts in forensic memory images. The features extracted from the memory images were registry activity, imported libraries, and API function calls. Different machine learning techniques were implemented to compare performances of malware detection. The highest Accuracy was reached by a SVM model with the feature of registry activity. Ajay Kumara et al. [20] proposed an advanced VMM-based machine learning techniques at the hypervisor. Machine learning techniques were used to analyze the executables that were mined and extracted using MFA (memory forensic analysis)-based techniques and ascertained the malicious executables. The evaluation results achieved 99.55% Accuracy and 0.4% FPR on the 10-fold cross-validation to detect unknown malware on the generated dataset. Bai et al. [21] improved malware detection using multi-view ensemble learning. Because malwares were hard to disguise itself in every feature view, so they designed two schemes to incorporate three single-view features to fully exploit complementary information to discover malwares which achieved a false alarm rate of 0%.

In our work, we combine machine learning technology with memory forensic to detect kernel rootkits in VMs. Firstly, VM memory dumps are acquired in a trusted way at the hypervisor level. Then memory forensic analysis framework Volatility is used to extract malicious features from the memory dumps. Finally, based on these features, machine learning classifiers are trained. We will describe the dataset acquisition and machine learning classifier training in detail in the next section.

3. The proposed methodology

In this section, the methods used in our research are presented. In subsection 3.1, we introduce the way to obtain the volatile memory dumps from the monitored VM. In subsection 3.2, the methods of feature extraction and representation are introduced. In subsection 3.3, we describe

machine learning algorithms used to train classifiers. The flowchart of the methods is shown in Figure 1.

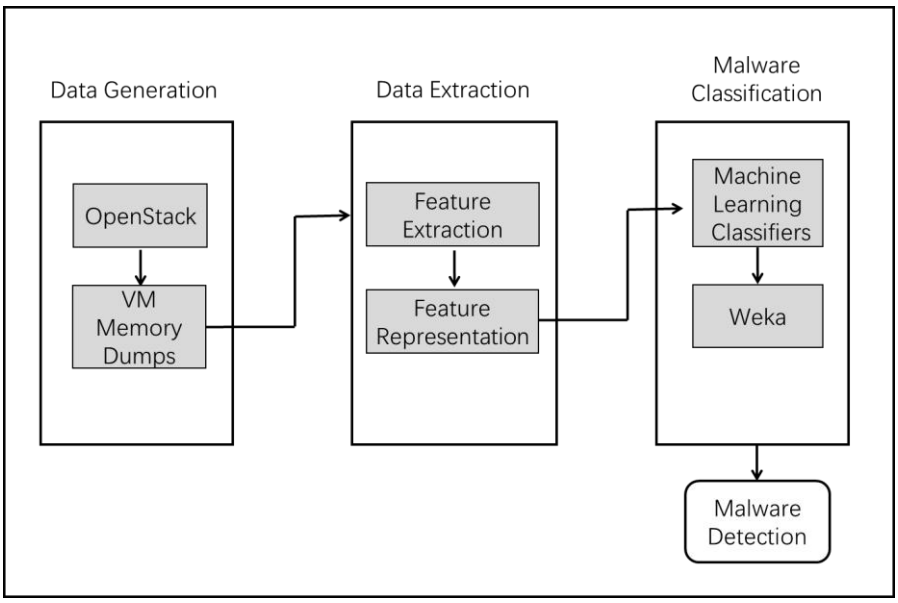

Figure 1. The flow chart of the malware detection.

3.1. Memory dump acquisition from OpenStack platform

OpenStack [22] is a popular cloud operating system, which manages the whole cloud computing environment. Users utilize OpenStack to deploy their public, private and hybrid cloud environment. In our research, we leverage OpenStack to deploy a private cloud experiment environment. OpenStack manages various resources throughout datacenter and provides dashboard or API to users to simplify the routine operations. The topological structure of the virtualization management infrastructure is shown in Figure 2.

The network node provides network service for the whole platform. The controller node is responsible for managing other nodes, providing identity service, image service and dashboard service. The storage node includes block storage service and object storage service. The virtual machine instances are created on the compute nodes, which provide computing service. By default, the kernel-based VM (KVM) hypervisor is supported by OpenStack. So we choose KVM as our hypervisor. Windows 7 Ultimate (x86) has been installed on the VM.

We use the dashboard on the controller node to create VM instances on the compute node. The dashboard supplies snapshot acquisition function. During the VM running process, we get its snapshots including volatile memory dumps and other information. We only extract the volatile memory dump documents for data analysis. Because the memory dumps are obtained out of the monitored VM, the malware running in the VM cannot evade or interfere with the memory obtaining process. Otherwise, the memory acquisition mechanism may be destroyed or evaded by the smart malwares, if it is located in the same VM with the malware. So it is a trusted way to gain the memory dumps of the VM.

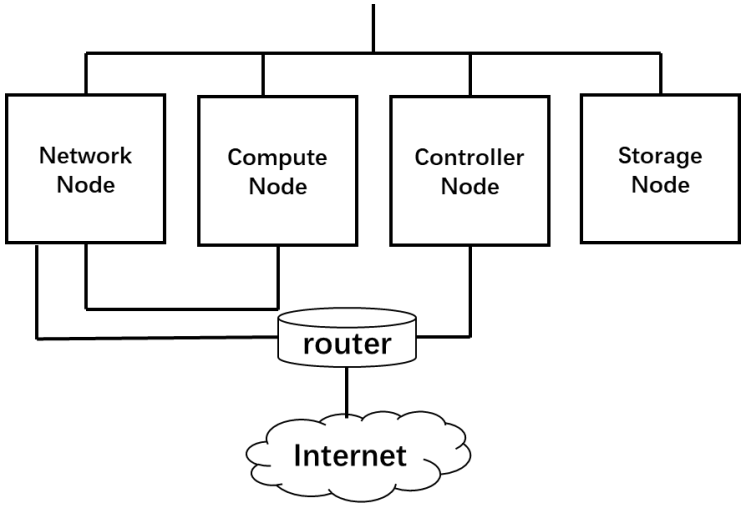

Figure 2. The topological structure of OpenStack platform.

3.2. Data collection

To collect training and testing dataset, ~100 kernel rootkit samples belonging to the family of NTRootkit, FURootkit etc. and their variants have been downloaded from the malware repository. The rootkit samples run on the VM one at a time. As the malware is dynamic, the state of the memory will be constantly changing. During the running time of each sample, we capture its memory states at appropriate intervals, for example every 10 minutes. Because there are long running samples and short running samples, the interval for memory capture are varied. For the short running samples the intervals are shorter to avoid the issue that they bypass the detection. The intervals of memory capture can be short or long depending on the requirement of the system. To this end, this sampling interval can be adaptively set as a tradeoff of efficiency and efficacy. For each sample, 100 memory dumps have been captured. The total number of captured memory dumps is 10000. Malicious features of each dump have been extracted, forming a record in the dataset. To get the benign memory samples, we captured memory state of the fresh installed Windows OS and some commonly used applications such as office, PDF, Winzip, Google browser etc.. During the running process of each application, the memory dumps have been obtained in the same way as the malicious. The total number of the benign memory dumps is 2300. An automatic tool for memory dump acquisition and feature extraction has been developed based on Python script.

3.3. Feature extraction and representation

After obtaining the memory dumps, features need to be extracted from them for abnormal behavior detection. Memory forensic framework Volatility [23] is used to extract features for memory forensic analysis.

3.3.1. The Volatility framework

As a widely used memory forensic platform, Volatility is a completely open collection of tools, implemented in Python under GNU Public License. Memory forensics for Windows, Linux and

MAC operating systems are supported by Volatility. In our research, Volatility 2.6 is leveraged to extract digital artifacts from volatile memory samples of Windows 7. Various plugins are supplied by Volatility to extract information from memory dumps. In our research, the following plugins are utilized to extract features, shown in Table 1.

Table 1. Volatility plugins used for rootkit detection.

Volatility plugin	Description
modules	Print loaded modules
modscan	Pool scanner for kernel modules
threads	Investigate _ETHREAD and _KTHREADs
OrphanThread	Investigate orphan threads
driverscan	Pool scanner for driver objects
driverirp	Pool scanner for driver objects
devicetree	Show device tree
ssdt	Display SSDT entries
callbacks	Print system-wide notification routines
timers	Print kernel timers and associated module DPCs

3.3.2. Feature extraction

According to the way rootkit attacks, the features to be extracted are as follows [24]: hidden kernel modules, orphan threads, driver objects, device tree, the SSDT function, call backs and timers.

Hidden kernel modules: The rootkit loads into the kernel as a kernel module. In order to hide its existence, rootkit hides its module. A metadata structure KLDR_DATA_TABLE_ENTRY is generated when a kernel module is loaded into the kernel. All of the metadata structures form a doubly linked list. The rootkit can hide its presence by unlinking the corresponding entry from the list, as shown in Figure 3.

In Volatility, the *modules* plugin walks through the double linked list to enumerate the kernel modules. If the metadata structure is unlinked, the module becomes stealthy for the *modules* plugin. Although the metadata entry is unlinked from the list, it is still in the memory, which is tagged with MmLd. The *modscan* plugin uses pool-scanning approach to find the metadata entry by the tag of Mmld. Through comparing the scanning results of these two plugins, the unlinked stealthy modules can be found.

Orphan threads: Orphan threads do not belong to any modules, which is an abnormal phenomenon in memory. For the purpose of hiding their presence, some rootkits unload the module when the executing thread is created. It can evade the detection of suspicious modules, but will create Orphan thread which is an obvious artifact for rootkit detection. We use *threads* and *OrphThread* plugins to find out orphan threads.

Driver objects: When a kernel module is loaded, a corresponding structure _DRIVER_OBJECT is initialized. The driver object contains information about the kernel module. Even when the tag MmLd of the metadata structure KLDR_DATA_TABLE_ENTRY is corrupted, the *driverscan* plugin can still detect the module. So through the comparison between the results of *modules* and *driverscan* plugins, we can discover the hidden module whose metadata structure has been destroyed.

Every driver object contains a major function table that handles different requests from the OS

user mode. Rootkit can hook these functions. To discover the hooks, *driverirp* plugin is needed to display the 28 values in the MajorFunction array. Of course, further analysis is needed to decide the existence of the hooks.

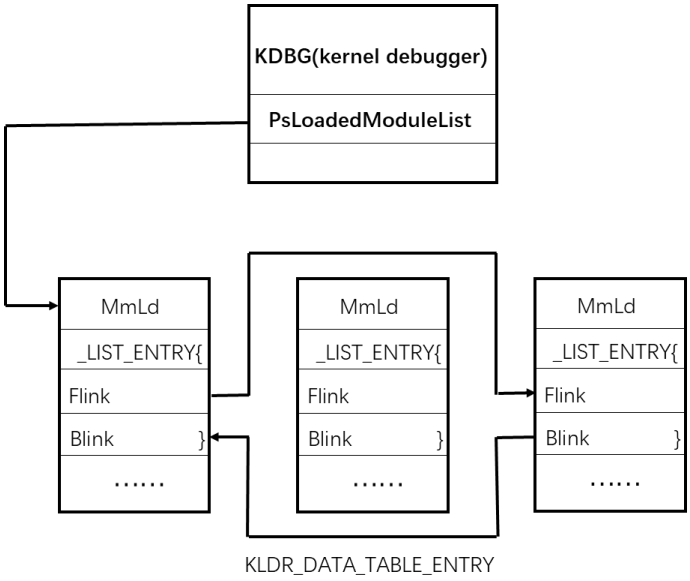

Figure 3. The doubly linked list of module entry.

Devicetree: In Windows OS, one device object is permitted to attach to the stack of another device. So they can handle the same I/O request, achieving transparent system archiving and encryption. This also provides rootkits with opportunities to intercept the legitimate I/O Request Packets (IRPs). The *devicetree* plugin can be used to audit the device tree to inspect whether there are malware drivers attaching to the legitimate device.

SSDT: SSDT containing pointers to kernel mode functions is a big target for rootkits. Many rootkits get to execute malicious modules by modifying or hooking the SSDT. So it is necessary to audit the SSDT. There are several ways to attack the SSDT functions. Some rootkits overwrite the pointers in SSDT. Normally, SSDT only points to two tables: native API table and shadow SSDT table, corresponding to the module of ntoskrnl.exe and win32k.sys respectively. The *ssdt* plugin can detect the modified pointers pointing to other malicious modules. More cunning rootkits use inline hook technique, instead of pointing out of the NT modules or win32k.sys. In this case, further static analysis is needed.

Callbacks: The *callback* plugin can show all the callbacks installed on the system, including information about the callback type, the start address of the callback function, the corresponding module of the callback function and some other information. If a rootkit has hidden its module, the corresponding module information will show *unknown*, which is an obvious abnormal artifact. Take one step back, some rootkits don't hide their modules, but use a hard-coded name which also indicates a compromise.

Timers: Timer is another feature indicating a rootkit. Each timer is associated with a module. The *timers* plugin can list all timer objects created in the memory, including the module that the timer belongs to. So if the module shows *unknown*, the timer is suspicious.

The above is all the features used to detect rootkits in our research. Different rootkits may have

different features. In Table 2, we list the features and the plugins extracting them. We develop a Python-based tool to automatically extract these features from the memory dumps.

Table 2. The features extracted and plugins used.

Feature	Description	Data type	Plugin
Hidden modules	Whether hidden modules exist	Boolean	modules modscan
Orphan thread	Whether orphan threads exit	Boolean	thread OrphanThreads
Driver object	Whether abnormal driver object exits	Boolean	driverscan modules
Hooks in the IRP table	Whether hooks exit in the major function table	Boolean	driverirp
Malicious attachment	Whether there is malware driver attaching to the legitimate device	Boolean	devicetree
SSDT hooking	Whether there are hooks on SSDT	Boolean	ssdt
Abnormal Callbacks	Whether there is malicious callback in the system	Boolean	callbacks
Abnormal timers	Whether there is malicious timer in the system	Boolean	timers

3.3.3. Feature representation

In order to improve the classification, we use a vector $\vec{v}$ to represent the features. The element of the vector consists of two parts: the Boolean value and the weight value. The Boolean value indicates whether a certain feature appears in the memory dump. In our research, there are totally eight features. If the feature shows up in the memory dump, the Boolean value of this feature is true represented by 1, otherwise it is false represented by 0.

The weight of the feature is gauged by Mutual Information (MI) which measures the statistical dependence of the feature on the class of the sample, as shown in (1).

$$I(X;Y) = \sum_{y \in Y} \sum_{x \in X} p(x,y) \log\left(\frac{p(x,y)}{p(x) \cdot p(y)} \right) \quad (1)$$

where X represents the frequency of occurrence of a feature in a memory dump and Y is the class of the sample indicating whether or not the sample is a rootkit (i.e. rootkit or benign). $p(x)$ and $p(y)$ are the marginal probability of X and Y. $p(x,y)$ is the joint probability of X and Y. The MI calculated for each feature is used as a weight value, which reduces the noise from less relevant features. The vector is expressed as (2):

$$\vec{v} = ((f_1, w_1), (f_2, w_2), \ldots, (f_{n-1}, w_{n-1}), (f_n, w_n)) \quad (2)$$

where f_n is the Boolean value of the nth feature, and w_n is the weight value computed with (1).

With the feature vector of the memory dumps, machine learning based classifiers can be trained to detect the variants of the malwares and the new emerging ones.

3.4. Machine learning algorithms

The machine learning algorithms used to train our classifiers include those based on human perception and bio-inspired models, such as Decision trees, Rule based classifiers, Bayesian and SVM.

Based on tree structures to make decisions, decision tree is a simple but widely used classification technology. Generally, a decision tree contains one root node, several internal nodes and leaf nodes. Each leaf node is assigned with a class label corresponding to the decision result. The internal node corresponds to an attribute test condition, which is used to distinguish records with different characteristics and divide the sample set into different sub-nodes. The purpose of decision tree learning is to produce a decision tree with strong generalization ability. After building the decision tree, tree-pruning can be carried out to reduce its scale. Therefore, the over fitting problem can be solved and the generalization ability is improved.

Rule based classifiers uses a set of *if…then…* rules to classify records. A rule is as follows: $\oplus \leftarrow f_1 \wedge f_2 \wedge \cdots \wedge f_{\mathrm{L}}$. The part on the right side of the symbol $\leftarrow$ is called rule body, which indicates the premise of the rule. The left part is called rule head representing the result of the rule. The quality of classification rules can be gauged with Coverage and Accuracy. Rule sets generated by rule-based classifiers have two important properties: Mutually Exclusive Rule and Exhaustive Rule which are combined to ensure that every record is covered by one and only one rule.

Bayesian is a method of modeling probabilistic relations between attribute sets and class variables. It combines prior probability of a class with new evidence collected from data. There are two implementations of Bayesian classifier: naive Bayes and Bayesian belief network (BBN). Naive Bayesian classifier assumes conditional independence between attributes when estimating conditional probability of classes. BBN is applicable to problems of classification with certain correlations between attributes.

SVM is a dual-classification model whose purpose is to find a hyperplane to segment the samples. The principle of segmentation is to maximize the interval, which eventually converts to a convex quadratic programming problem. In the sample space, the partition of hyperplane can be described with the following linear equations: $w^T x + b = 0$, in which *w* is the normal vector which determines the direction of hyperplane, and *b* is the displacement determining the distance between the hyperplane and origin. It finally comes down to a cubic programming problem, which can be solved with a general quadratic programming algorithm. SMO (Sequential Minimal Optimization) is an efficient algorithm, which can effectively solve the dual problem derived from SVM.

4. Experiments and evaluation

MalShare [25] and VirusShare [26] are websites of malware repository. In our experiment, around 100 Windows kernel rootkit samples have been collected from them, including FURootkit, NtRootkit etc., as well as their variants for 32bit Windows 7 OS. VirusTotal [27] is an online malware analysis system, through which the type of the collected samples can be confirmed. VirusTotal can also display the variant malwares family.

Weka [28] is a suite of machine learning software written in Java, containing a collection of tools and algorithms for data analysis and predictive modeling. In our research, Weka 3.9 was used for classifier training. The default file format supported by Weka is Attribute Relation File Format (ARFF). It also supports csv format in which we stored our data.

To train the classification models, various algorithms supplied by Weka are used, which are divided into four categories: Decision tree, Rule based classifiers, Bayesian and SVM. In each category, the following algorithms are selected: RandomForest, J84 algorithms for Decision tree; JRip, PART algorithms for Rule based classifiers; BayesNet, NaiveBayes for Bayesian classifiers; SMO for SVM. The default configurations of these algorithms are utilized. The machine learning models and corresponding algorithms are shown in Table 3.

Table 3. The machine learning models and corresponding algorithms.

Machine learning models	Algorithms
Decision tree	RandomForest, J84
Rule based classifiers	JRip, PART
Bayesian	BayesNet, NaiveBayes
SVM	SMO

To evaluate the effectiveness of the classifiers trained by the algorithms proposed, we need to answer the following questions:

i. Which classifier we selected has the best performance?
ii. Whether or not the scale of the dataset influences the performance of the classifier?
iii. Can the trained classifiers effectively detect new rootkits?

4.1. Experimental Settings

Based on these questions, the following experiments are designed:

Experiment #1: Ten-fold cross-validation is used to train the chosen classifiers. The extracted features of all the collected samples are used to train the classifiers. There are 12300 records in the dataset, including 10000 malicious and 2300 benign samples. All the classifiers are trained with the default parameters available in Weka. The purpose of this experiment is to verify the best classifier for rootkit detection with the dataset provided.

Experiment #2: We selected 3000 malicious records and 450 benign records from the dataset of experiment#1 to train the classifiers to verify the influence of dataset scale on the effect of the classifiers.

Experiment #3: To verify whether our trained classifier can detect unknown malware

effectively, we collected some new samples to form a testing dataset, including 500 new malicious samples and 300 benign ones to prove that the classifier can detect new emerging malwares.

4.2. Evaluation Criteria

To evaluate the performance of each classifier, we use TPR, FPR, the AUC value (the area under the receiver operating characteristic curve), F-measure and Accuracy as the evaluation criteria. The formulas are shown below, where TP and FP respectively represent the numbers of the correctly classified malwares and benign software, and FN and TN are the numbers of malwares and benign software misidentified. The definitions for these metrics are given as follows.

$$TPR = \frac{TP}{TP + FN} \tag{3}$$

$$FPR = \frac{FP}{FP + TN} \tag{4}$$

$$recall = \frac{TP}{TP + FN} \tag{5}$$

$$Precision = \frac{TP}{TN + FP} \tag{6}$$

$$F\text{-}measure = 2 \cdot \frac{Precision \cdot recall}{Precision + recall} \tag{7}$$

$$Accuracy = \frac{TP + TN}{TP + FP + TP + TN} \tag{8}$$

4.3. Experimental results

From the result of experiment #1 shown in Table 4, it can be seen that the Random Forest classifier has the best performance with TRP of 0.996, FRP of 0.079 and AUC of 0.997. The Bayes classifier has the lowest capability for malware detection: the TPR of the NaiveBayes is 0.838， and the FPR and the AUC are 0.630 and 0.922. The JRIP and PART classifier have better performance than SMO .

Table 4. The results of experiment #1.

Classifier	TPR	FPR	AUC	F-measure	Accuracy
Random Forest	0.966	0.079	0.997	0.966	0.966
J48	0.955	0.096	0.990	0.956	0.956
JRip	0.965	0.079	0.967	0.965	0.965
PART	0.963	0.079	0.996	0.964	0.964
BayesNet	0.847	0.630	0.922	0.815	0.827
NaiveBayes	0.838	0.729	0.950	0.785	0.824
SMO	0.925	0.016	0.954	0.930	0.947

Table 5. The results of experiment #2.

Classifier	TPR	FPR	AUC	F-Measure	Accuracy
Random Forest	0.946	0.096	0.983	0.946	0.946
J48	0.931	0.100	0.967	0.932	0.934
JRip	0.903	0.154	0.951	0.904	0.906
PART	0.953	0.071	0.980	0.953	0.954
BayesNet	0.776	0.630	0.912	0.801	0.813
NaiveBayes	0.819	0.603	0.938	0.775	0.825
SMO	0.935	0.019	0.958	0.938	0.950

Table 5 is the result of experiment #2. Comparing Table 5 with Table 4, the performance of classifiers declines as a whole. For example, the TPR of the Random Forest classifier decreases from 0.966 to 0.946, and FRP increases from 0.079 to 0.096. It indicates that the quantity of the training samples does affect the classifier performance. The classifier trained with mass data would have a better performance.

To verify the unknown malware detection ability of the classifier, we collected 500 new malware samples and 300 benign to form a new testing dataset. The classifiers trained in experiment#1 were tested to classify the new samples. From the result of the experiment shown in Table 6, the detection performances are good for most classifiers. It indicates that the classifier can detect unknown malware effectively. The Random Forest classifier has the best performance which can detect unknown kernel rootkits with an Accuracy of 0.986 and an AUC value of 0.998.

Table 6. The results of experiment #3.

Classifier	TPR	FPR	AUC	F-measure	Accuracy
Random Forest	0.984	0.076	0.998	0.986	0.986
J48	0.961	0.084	0.991	0.962	0.965
JRip	0.967	0.094	0.974	0.971	0.971
PART	0.969	0.064	0.987	0.966	0.965
BayesNet	0.856	0.329	0.932	0.891	0.894
NaiveBayes	0.839	0.583	0.958	0.795	0.864
SMO	0.965	0.010	0.978	0.958	0.968

The comparison of TPR, FPR, AUC, F-Measure and Accuracy achieved by all the classifiers in the three experiments is shown in Figures 4–8.

Figures 4 and 5 show the TPR and FPR of the classifiers in the three experiments, respectively. Random Forest has the highest TPR on the whole, which reaches as high as 0.984 in experiment #3. The average value of the TPR of Random Forest in the three experiment is 0.965. The average TPR value of J84, JRip, PART, BayesNet, NaiveBayes and SMO are respectively 0.949, 0.945, 0.962, 0.826, 0.832, 0.942. The classifier of Random Forest has the highest average value of TPR. For FPR, Random forest, J47, PART and SMO seem to produce much lower FPR than BayesNet and NaiveBayes, where SMO has the least FPR among all these approaches.

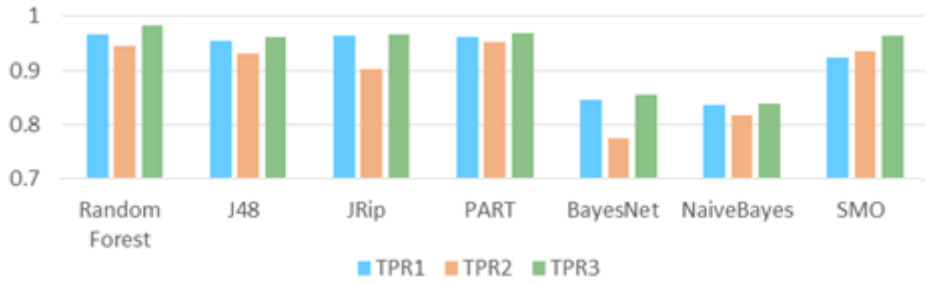

Figure 4. The TPR of the three experiments.

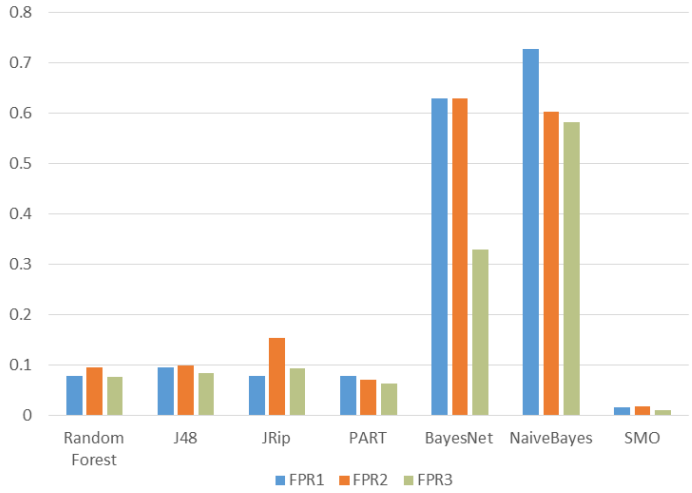

Figure 5. The FPR of the three experiments.

Figures 6–8 are respectively the AUC, F-Measure, and Accuracy of the classifiers in the three experiments. As seen, the highest average values of AUC, F-Measure, Accuracy are achieved by the Random Forest classifier, which are 0.992, 0.966 and 0.966. As a result, the Random Forest classifier is the most excellent classifier in our experiment for kernel rootkits detection for VMs.

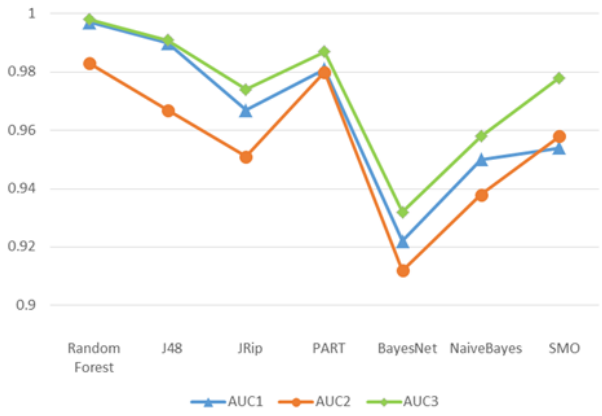

Figure 6. The AUC of the three experiments.

Figure 7. The F-Measure of the three experiments.

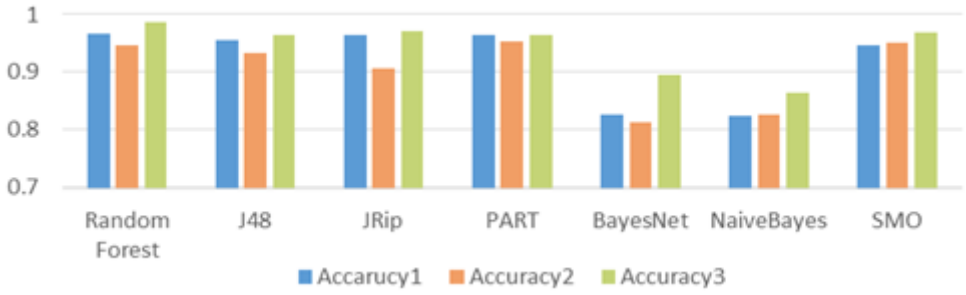

Figure 8. The Accuracy of the three experiments.

4.4. Comparison with other related works

The comparison with other related works is summarized in Table 7. For each approach, several classifiers are compared, where the classifier to produce the best results among each group is underlined in the table.

Nissim et al. [16] presented a methodology for trusted detection of unknown malwares for VMs. Experiment #4 in Nissim et al. [16] was aiming at detecting new emerged unknown malwares which is most relevant with our work. Classifiers of Decision Tree, Random Forest, Naïve Bayes, Bayesian network and SVM were trained. The best performance was achieved by Random Forest with the TPR of 0.979, FPR of 0 and AUC of 1.The dataset contained 4000 samples, half of which were benign and the others were malicious. They used 80% of the data for training and 20% for testing.

Cohen et al. [15] conducted trusted analysis of memory dumps for VMs using Volatility framework, and classifiers of Random Forest, BayesNet, AdaBoostM1 etc. were trained. Experiment 2 in Cohen et al. [15] focused on detecting the unknown infected states which is most relevant with our research. The best performance was reached by Random Forest with the TPR of 0.958, FPR of 0 and AUC of 1. The training dataset consisted of 400 malicious instances and 400 benign ones, the testing dataset consisted of 100 malicious instances and 100 benign ones.

Bai et al. [21] proposed a malware detection method using ensemble method. Scheme II in Bai et al. [21] leveraged multi features to train base classifiers which is most interrelated with our work. Classifiers of Random Forest, J48 and BayesNet were trained on dataset D2. Its best performance was reached by Random Forest with the TPR of 0.988, FRP of 0.001 and AUC of 0.998. The training dataset consisted of 5202 malicious instances and 3918 benign instances, and the testing dataset was composed of 109245 malicious instances and 3953 benign instances.

The best performance of our proposed work reached as TPR of 0.984, FPR of 0.076 and AUC of 0.998 by Random Forest, which achieved a relatively satisfactory result for VM kernel rootkits detection.

Table 7. The Comparison with other related works.

Related work	Classifiers	TPR	FPR	AUC
Nissim et al.	**RF**,DT,NB,BN,SVM	0.979	0	1
Cohen et al.	**RF**,BN,AM,LB J48,NB,SMO, Bagging,LR	0.958	0	1
Bai et al.	**RF**,J48,BN	0.988	0.001	0.998
Proposed work	**RF**,J48,PART,JRip,NB, BN,SMO	0.984	0.076	0.998

Random Forest (RF), Decision Trees (DT), Naïve Bayes (NB), Bayesian network (BN), Support Vector Machine (SVM), AdaBoostM1(AM), LogitBoost(LB), Logistic Regression (LR), Partial Decision Tree (PART), where the best classifier within each group is underlined and **bold** for attention.

5. Conclusions

Aiming at solving the cybersecurity problem of virtual machine in private cloud computing environment, a TKRD method is proposed to detect known and unknown kernel rootkits based on machine learning method. We leveraged memory forensic technology to obtain the malicious tracks in the VM memory. Machine learning technology is used to analyze the malicious behavior of the kernel rootkits and detect malware attacks in the VMs. Experimental results show that our TKRD method can effectively detect kernel rootkits with a best Accuracy of 0.986 and AUC of 0.998.

The major innovation of our work is summarized as following. First, a trusted way for memory dump acquisition was proposed, where the memory dumps of the VM were collected on the compute nodes at the controller node of the cloud OS OpenStack. For the short and long time running rootkits, the memory capture interval can be adjusted, which can be adaptively set as a tradeoff of efficiency and efficacy. As far as we know, it is the first time that such a method is proposed in VM memory forensics. Second, comprehensive features representing the abnormal behaviors of the kernel rootkits were extracted from the memory dumps, from which we could identify the kernel rootkits effectively. At last, in order to find the classifier with the best performance, various machine learning models were evaluated with the best one i.e. the random forest classifier recommended.

In future work, deep learning methods [29] such as S-SAE [30] can be used to train malware classifiers. It can not only reduce complexity but also improve the efficacy of feature extraction and the accuracy of data classification. Other classifiers such as SVM-GRBF and CNNs [31] as an extended ANN [32] will be investigated to find the optimal classifying way for malware detection. In addition, we will also explore combined strategies for malware detection as it has shown great potential in Kumara et al. [17], and this will be extended for improved detection of rootkits in future.

Acknowledgments

This research was sponsored by the International Research Cooperation Seed Fund of Beijing University of Technology (No. 2018A01).

Conflict of interest

The authors declare there is no conflict of interest.

References

1. Y. Ding, H. M. Wang, P. C. Shi, et al., Trusted cloud service, *Chin. J. Comput.*, **38** (2015), 133–149.
2. M. Ali, S. U. Khan and A. V. Vasilakos, Security in cloud computing: Opportunities and challenges, *Inform. Sciences.*, **305** (2015), 357–383.
3. Y. Q. Zhang, X. F. Wang, X. F. Liu, et al., Survey on cloud computing security, *J. Software*, **27** (2016), 1328−1348.
4. J. Wilhelm and T. C. Chiueh, A forced sampled execution approach to kernel rootkit identification, *In: International Workshop on Recent Advances in Intrusion Detection;* 2007 Sept 5–7; Gold Goast, Australia. Berlin: Springer; 2007: 219–235.
5. N. Zhang, R. Zhang, K. Sun, et al., Memory Forensic Challenges Under Misused Architectural Features, *IEEE T. Inf. Foren. Sec.*, **13** (2018), 2345–2358.
6. A. Cohen, N. Nissim, L. Rokach, et al., SFEM: Structural feature extraction methodology for the detection of malicious office documents using machine learning methods, *Expert Syst. Appl.*, **63** (2016), 324–343.
7. N. Nissim, R. Moskovitch, O. BarAd, et al., ALDROID:Efficient update of Android anti-virus software using designated active learning methods, *Knowl. Inf. Syst.*, **49** (2016), 795–833.
8. N. Nissim, A. Cohen, C. Glezer, et al., Detection of malicious PDF files and directions for enhancements: A state-of-the art survey, *Comput. Secur.*, **48** (2015), 246–266.
9. G. Hoglund and J. Butler, *Rootkits: subverting the Windows kernel*, Addison-Wesley Professional, New Jersey, 2006.
10. A. Case and G. G. Richard III, Advancing Mac OS X rootkit detection, *Digit. Invest.*, **14** (2015), S25–S33.
11. H. Yang, J. Zhuge, H. Liu, et al., A tool for volatile memory acquisition from Android devices, *In: IFIP International Conference on Digital Forensics*; 2016 Jan 4-6; New Delhi, India. Cham: Springer; 2016: 365–378.
12. A. Kumara and C. D. Jaidhar, Execution time measurement of virtual machine volatile artifacts analyzers, *In: 2015 IEEE 21st International Conference on Parallel and Distributed Systems (ICPADS);*2015 Dec14-17; Melbourne, VIC, Australia. IEEE; 2015: 314–319.
13. Q. Hua and Y. Zhang, Detecting Malware and Rootkit via Memory Forensics, *In:2015 International Conference on Computer Science and Mechanical Automation (CSMA)*; 2015 Oct 23–25; Hangzhou, China. IEEE; 2015: 92–96.
14. C. W. Tien, J. W. Liao, S. C. Chang, et al., Memory forensics using virtual machine introspection for Malware analysis, *In:2017 IEEE Conference on Dependable and Secure Computing;* 2017 Aug 7-10; Taipei, Taiwan. IEEE; 2017: 518–519.
15. A. Cohen and N. Nissim, Trusted detection of ransomware in a private cloud using machine learning methods leveraging meta-features from volatile memory, *Expert Syst. Appl.*, **102** (2018), 158–178.

16. N. Nissim, Y. Lapidot, A. Cohen, et al., Trusted system-calls analysis methodology aimed at detection of compromised virtual machines using sequential mining, *Knowl.-Based Syst.*, **153** (2018), 147–175.
17. A. Kumara and C. D. Jaidhar, Automated multi-level malware detection system based on reconstructed semantic view of executables using machine learning techniques at VMM, *Future Gener. Comp. Sy.*, **79** (2018), 431–446.
18. H. Upadhyay, H. A. Gohel, A. Pons, et al., Windows Virtualization Architecture For Cyber Threats Detection. *In:2018 1st International Conference on Data Intelligence and Security (ICDIS).* 2018 Apr 8-10; South Padre Island, TX, USA.IEEE; 2018: 119–122.
19. R. Mosli, R. Li, B. Yuan, et al., Automated malware detection using artifacts in forensic memory images. *In:2016 IEEE Symposium on Technologies for Homeland Security (HST).* 2016 May 10-11; Waltham, MA, USA. IEEE; 2016: 1–6.
20. M. A. Kumara and C. D. Jaidhar, Leveraging virtual machine introspection with memory forensics to detect and characterize unknown malware using machine learning techniques at hypervisor, *Digit. Invest.*, **23** (2017), 99–123.
21. J. Bai and J. Wang, Improving malware detection using multi view ensemble learning, *Secur. Commun. Netw.*, **9** (2016), 4227–4241.
22. OpenStack. Available from: https://docs.openstack.org/rocky/.
23. Volatility. Available from: https://www.volatilityfoundation.org/.
24. M. H. Ligh, A. Case, J. Levy, et al., *The art of memory forensics: detecting malware and threats in windows, linux, and Mac memory*, John Wiley & Sons, New Jersey, 2014.
25. Malshare. Available from: http://www.malshare.com
26. T. Kim, B. Kang, M. Rho, et al., A Multimodal Deep Learning Method for Android Malware Detection Using Various Features. *IEEE T. Inform. Foren. Sec.*, **14** (2019), 773–788.
27. Virustotal. Available from: https://www.virustotal.com/
28. M. Hall, E. Frank, G. Holmes, et al., The WEKA data mining software: an update. *ACM SIGKDD explorations newsletter,* **11** (2009): 10–18.
29. Z. Wang, J. Ren, D. Zhang, et al., A deep-learning based feature hybrid framework for spatiotemporal saliency detection inside videos, *Neurocomputing*, **287** (2018), 68–83.
30. J. Zabalza, J. Ren, J. Zheng, et al., Novel segmented stacked autoencoder for effective dimensionality reduction and feature extraction in hyperspectral imaging, *Neurocomputing*, **185** (2016), 1–10.
31. S. Md Noor, J. Ren, S. Marshall, et al., Hyperspectral Image Enhancement and Mixture Deep-Learning Classification of Corneal Epithelium Injuries, *Sensors*, **17** (2017), 2644.
32. J. Ren, D. Wang and J Jiang, Effective recognition of MCCs in mammograms using an improved neural classifier, *Eng. Appl. Artif. Intel.*, **24** (2011), 638–645.

AIMS Press