LETTER

On the Detection of Malicious Behaviors against Introspection Using Hardware Architectural Events*

Huaizhe ZHOU[†], **Haihe BA**[†], **Yongjun WANG**[†a)], ***Nonmembers***, ***and*** **Tie HONG**[†], ***Member***

SUMMARY The arms race between offense and defense in the cloud impels the innovation of techniques for monitoring attacks and unauthorized activities. The promising technique of virtual machine introspection (VMI) becomes prevalent for its tamper-resistant capability. However, some elaborate exploitations are capable of invalidating VMI-based tools by breaking the assumption of a trusted guest kernel. To achieve a more reliable and robust introspection, we introduce a practical approach to monitor and detect attacks that attempt to subvert VMI in this paper. Our approach combines supervised machine learning and hardware architectural events to identify those malicious behaviors which are targeted at VMI techniques. To demonstrate the feasibility, we implement a prototype named HyperMon on the Xen hypervisor. The results of our evaluation show the effectiveness of HyperMon in detecting malicious behaviors with an average accuracy of 90.51% (AUC).

key words: *virtual machine introspection, hardware architectural events, supervised machine learning*

1. Introduction

The proliferation of cloud computing also makes it an attractive target for malicious attacks. Attackers benefit from the flexibility and scalability features of the cloud to deploy and propagate malware. For instance, a corrupted virtual machine image can infect all the virtual machines built with that image. Although many significant efforts continue to detect and protect cloud from corruption, sophisticated adversaries frustrate the efforts by using advanced malicious code.

In addition to provisioning and managing resources effectively, virtualization also has the potential to improve guest system security significantly. By combining isolation, inspection, and interposition provided by the hypervisor, the method referred to as virtual machine introspection (VMI) [1] can analyze and manipulate the guest system operating from the hypervisor. It enables the tamper-resistant monitoring of guest VMs. So in recent years, the increasing solutions resort to VMI to deal with security issues in the cloud environment [2]. VMI has become a relatively mature technique with a range of valuable researches on bridging the semantic gap [3]. However, existing solutions are brittle since they implicitly assume the guest OS is benign [4]. The fragile assumption could be violated by elaborate malware or rootkits to trick an introspection-based security monitor [4].

Moreover, the most of existing methods always require specialized knowledge about guest OS to reconstruct the high-level rich semantic view of the introspected VM. They depend on source code or debugging symbols of a specific version of the guest OS, which makes them hard to generalize. To abstract data structures relevant to OS information accurately, they also have to keep up with an update of the OS version, which inevitably increases the complexity of development.

Given the shortcomings of existing VMI methods, our research focuses on the reliable and unified monitoring to identify malicious activities in the VM without introspection. The most significant challenge is how to characterize program behavior factually out of the VM. This process consists of two tightly coupled phases: *tracing* and *profiling*. In the *tracing* phase, we concern about the capture of relevant system events with high fidelity. And in the *profiling* phase, we focus on analyzing the obtained data to classify the state of the system effectively. Motivated by recent work [5], which leverage hardware performance events and machine learning methods to detect malware, we exploit the available hardware features to characterize program behavior.

In this paper, we present HyperMon — a hardware-based monitoring system that uses low-level hardware information available at the VMM layer to identify malicious activities against VMI techniques inside guest VMs. The presented system traces the information generated by the program interacting with the hardware during its execution. To achieve reliable monitoring, HyperMon traces the hardware architectural events, which cannot be manipulated by attackers in the guest VM. Then we identify the program's behaviors in a derivation pattern, which is different from the previous method based on OS knowledge. Our derivation pattern employs machine learning methods to analyze the captured data and build a statistical model for a specific program. To demonstrate the feasibility of HyperMon to identify malicious activities in the VM, we implement a prototype system on the Xen hypervisor and evaluate it with a set of malware. In summary, we make the following contributions:

- A reliable approach to monitor program behaviors in

Manuscript received August 9, 2019.
Manuscript revised September 9, 2019.
Manuscript publicized October 9, 2019.

[†]The authors are with the College of Computer, National University of Defense Technology, Changsha, Hunan, China.

*This research was funded by the National Natural Science Foundation of China under Grant No. 61402508, No. 61303191 and No. 61472439.

a) E-mail: wangyongjun@nudt.edu.cn

DOI: 10.1587/transinf.2019EDL8148

VMs. We present our derivation-based method which utilizing supervised machine learning and architectural events to build statistical models for programs.
- A portable monitoring system. We implement HyperMon on Xen hypervisor and make it rely on common architectural events which are available on most processors. It is OS-independent and can cover different platforms as it works by hardware.
- A practical supplement to existing VMI-based solutions. We demonstrate that HyperMon's ability can be used to detect kernel rootkits which can circumvent VMI-based method.

The rest of the paper is organized as follows: Sect. 2 presents the problems that motivate our work. Section 3 introduces our approach, and Sect. 4 evaluates the feasibility. Conclusions are given in Sect. 5.

2. Problem Statement

Despite their advantages over other monitoring systems, VMI-based solutions have to bridge the semantic gap between the behaviors of the guest system and the information observed at the hypervisor. The existing VMI-based monitoring tools use data structures contained in the guest OS as "templates" to cast low-level information to VM's semantic entities. However, current VMI techniques are not sufficient to reconstruct the high-level rich semantic view of the introspected VM. Since current VMI-based solutions assume that the kernel data of the guest OS is conforming to a specific pattern. It is an inadvisable assumption with respect the guest OS kernel could be compromised by sophisticated malware, which is most often the case in security issues. Once compromised, the semantics of kernel data structures become questionable.

There are three major classes of threats against to VMI techniques, including Kernel Object Hooking (KOH) attacks, Direct Kernel Object manipulation (DKOM) attacks, and Direct Kernel Structure Manipulation (DKSM) attacks. They can disrupt the system information obtained by VMI techniques. Because a successful DKSM attack hinges on kernel control flow hijacking such as KOH attacks [4], we focus on the first two classes of attacks:

Kernel Object Hooking (KOH). A KOH attack attempts to modify function pointers located in various structures of the kernel. By overwriting the function pointer with malicious address, an attacker is allowed to interpose on desired kernel operations to hide information. The common structures exploited by attackers include Interrupt Descriptor Table (IDT) and system call table. For instance, an attacker can hook the service function called `NtQueryDirectoryFile()` in the System Service Dispatch Table (SSDT), which is the Windows equivalent of the Linux system call table, to hide malicious files in guest OS.

Direct Kernel Object Hooking (DKOM). A DKOM attack tries to modify kernel data structures in the memory directly through a loadable module or user-level accessing. DKOM attacks are distinct from hooking based attacks described above since they only modify data values. A typical example of such attacks is hiding a malicious process from the process list. An attacker can achieve this by altering the values of `FLINK` and `BLINK` in the struct `EPROCESS` under Windows.

Additionally, the kernel update or variant of OS may invalidate previous VMI-based systems which build on the semantic knowledge of the former OS. The administrators must keep up with all the internal changes of the guest even though there are no attacks above.

Given the fragility and mutability of the data structures of guest OS, we aim to explore other data available in the virtualized environment for extracting security-relevant information. The intuition motivating our research is that the operations in the guest system should be accomplished by hardware support. The architectural events of related hardware components can be used as the indicators of behaviors inside the guest VM. For example, a page fault trap can be used to analyze memory access behavior. It is inefficient for inferring high-level behavior by interpreting architectural events. We employ a derivation pattern which builds a statistical model of the guest system by learning instead of inspecting the content of architectural events. Our goals in this paper are: (1) to exploit available features of hardware to derive program behavior in the VM, (2) to strengthen existing VMI-based solutions by identifying malicious activities against VMI techniques.

3. Proposed Approach

Our overall approach comprises two steps. The first step is to trace the execution of the program inside the VM and extract related events. The second step is to analyze the captured data to profile the behavior pattern for identifying malicious activities against VMI techniques. In this section, we describe each step in detail.

3.1 Architectural Events Tracing

To identify malicious activities inside the guest VM, we need to trace a set of architectural events which can indicate specific behaviors of programs. According to the goals of this work, the requirements we set out for events tracing are defined as follows:

(R1) Non-intrusive manner The tracing scheme should work in a non-intrusive manner to keep the transparency of VMI techniques.
(R2) Reliability The traced events should not be tampered or forged by adversaries in the guest VMs.
(R3) Portability The tracing scheme is independent of the guest OS and can be deployed on the most virtualized platforms with little effort.

The designed events tracing scheme in our approach focuses on low-level events instead of high-level behavior such as system calls. There are abundant events that can be

collected during the execution of programs, such as instruction mix and I/O requests. The prior works have shown that these events can provide sufficient information for malware detection [5]. To meet the requirement *R1*, we select the events that can be collected in the hypervisor. The events such as instruction mix are out of the scope of our work.

The selected events include privileged operations, interrupts, control register (CR) accesses, virtual machine extensions (VMX) instructions, I/O requests, and hypercall events. These events derive from guest OS execution and can be used to infer malicious behaviors. For instance, we can identify a hidden process by analyzing events of CR3 register writing. In the virtualized environment, the events above are enforced by hardware-assisted virtualization and can not be circumvented by attackers in the VM, which meets the requirement *R2*.

The hypervisor lies beneath the guest OS and guarantees its execution correctly by intervening in its interaction with the hardware. Our events tracer is capable of recording the occurrence of an architectural event by intercepting it in the hypervisor. In our work, the events tracer targets a set of common events of modern virtualized platforms to meet the requirement *R3*. All the events we are interested in will be stored in a log file.

3.2 Behavior Profiling

To profile the behavior of programs in the guest VM, we need to filter and aggregate the traced event sequences to generate features. The features used in our work are constructed in the following steps. First, we divide an event sequence into multiple subsequences based on the domain ID, since the traced log file contains events from all the VMs running on the hypervisor. We leverage the VM scheduling events to separate and aggregate all the events into VMs. Next, we apply statistical methods on the event sequences to generate feature vectors. We use time-based windows to generate the features of all the selected events. These features are constructed by counting the occurrences of selected events within the window. Finally, the features of different events are aggregated into a vector for model building.

To identify malicious activities more accurately, we select the most informative features before the models built. First, we check all available features manually and exclude those appears all-zeros. Then we investigate the rest features and select the most informative ones according to the mutual information (MI) scores. Finally, we select 12 informative features to build behavior profiles of VMs. Their MI scores are shown in Fig. 1. The selected features contribute more information to generating a unique profile than others. Table 1 lists the top-4 features (due to space limitations) in the rank of MI scores and explains the details of them. Once the features are selected, the dataset used to build models only need to contain feature vectors that correspond to the chosen features.

3.3 Implementation

We implement the approach in the HyperMon monitoring system based on Xen hypervisor. Our implementation assumes that the underlying hardware and hypervisor are trusted. We illustrate the architecture of HyperMon in Fig. 2. The overall system consists of two subsystems: (1) events tracing and (2) behaviors analysis. The first subsystem executes each benign program or rootkit in a VM and collects event traces during the execution. This subsystem integrates with the target hypervisors, i.e., Xen in our implementation. Once the VM starts running, the monitor module starts the events tracing tools in the events tracer module at the hypervisor. We implement the events tracer module by modifying *xentrace* tool to trace the selected events during execution of monitored VM. At the end of each tracing procedure, the captured data are copied to the dataset for behaviors analysis.

The second subsystem constructs statistical models for program behaviors to identify malicious activities in the VM. Our behaviors analysis subsystem is implemented in Python 3.7. The feature constructor module creates features with the method described in Sect. 3.2. We make use of the `scikit-learn` package to implement the model builder module. We implement four classifiers for comparison, including k-Nearest Neighbors (k-NN), Decision Tree (DT), Random Forest (RF), and AdaBoost. In the training phase, this subsystem trains each classifier with the training dataset. With the trained classifiers, the detector module in the first subsystem can detect malicious activities against

Table 1 Top-4 features selected in HyperMon based on MI.

Features ID	Description
11	# of page fault events captured by Xen
5	# of operations for updating page tables
8	# of VMExit events caused by accessing CR registers
4	# of VMExit events caused by violating access permissions of EPT

Fig. 1 The mutual information scores of the selected features.

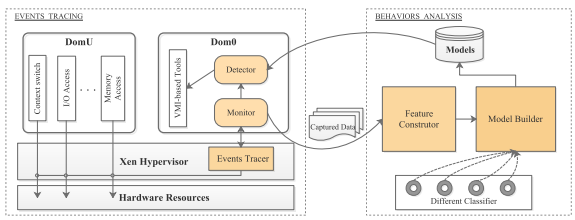

Fig. 2 The architecture overview of HyperMon.

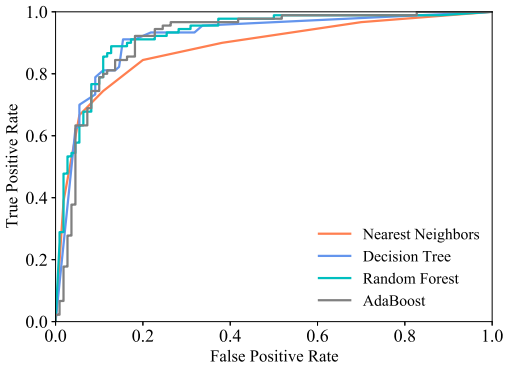

Fig. 3 Receiver Operating Characteristic (ROC) curves of trained models. The AUC of k-NN, DT, RF and AdaBoost is 88.40%, 91.57%, 92.49%, 91.53% respectively.

VMI and inform the VMI-based tools to correct their results in the detection phase.

4. Evaluation

To gauge the ability of the proposed method to identify malicious activities against VMI techniques, we deploy the prototype system of HyperMon on our testbed equipped with Intel i3-2130 3.4GHz processor and 16GB of RAM. We set a VM which has one vCPU and 1GB of RAM to run benign programs and rootkits. We installed Windows7 OS and a set of programs essential for rootkits execution. To profile behaviors of guest VM, we choose a set of system programs and benchmark programs as benign programs. We use different variants of well-known Windows7 rootkits which cover the preceding attacks against VMI-based tools. All the samples were collected from VirusTotal[†]. To vary the collected traces to get close to the real world, we run each benignware program and each malware program multiple times under different background conditions.

In our experiments, we divide our dataset into a training set and a testing set with a split of 80 : 20 ratio randomly. To compare the feature values in the machine learning context fairly, we normalize them to have a mean of zero and a standard deviation of one. HyperMon creates statistical models based on the training set to detect malicious activities in the VM. We measure the detection performance of these models by accessing the trade-offs between the true positive rates and false positive rates. Figure 3 shows the ROC curves for different models.

The results show that the true positive rates of RF are higher with most of the thresholds, compared to other models. RF model target classifying outliers in the dataset by using a collection of DTs, which is suitable for identifying malicious activities in the VM. Overall, HyperMon has AUCs above 88% with different models. The best model in our experiments, i.e., RF, has an AUC of 92.49%.

For the models involve randomness in their training, the evaluation results fluctuate with different training and testing. To avoid the overfitting of models and make the evaluation independent of the splitting of the dataset, we use cross-validation to evaluate our models in HyperMon. We compare the evaluation results in terms of precision, recall, F1-score, and AUC in both models.

[†]https://www.virustotal.com/

Table 2 Results of 10-fold cross-validation.

Models	Precision[%]	Recall[%]	F1_Score[%]	AUC[%]
Nearest Neighbors	85.69	89.41	87.39	87.25
Decision Tree	87.90	86.33	87.05	87.14
Random Forest	90.82	90.20	90.42	90.51
AdaBoost	88.30	86.76	87.31	87.52

Table 2 shows the average metrics for the performed experiments over 10-fold cross-validation. According to our results, the model of RF outperforms other models. The mean AUC values of k-NN, DT, RF, and AdaBoost is 87.25%, 87.14%, 90.51%, and 87.52%, respectively. The result illustrates the ability of hardware architectural events for classifying malicious behavior from normal behavior. Hence, in terms of identifying malicious activities against VMI-based tools, HyperMon is effectively making it a perfect supplement to existing VMI-based tools.

5. Conclusions

In this paper, our work suggests the fragility of existing VMI techniques in bridging the semantic gap. We demonstrate and analyze how potential attacks could invalidate VMI-based tools. Aimed at these problems, we introduce HyperMon, a system that identifies malicious activities in a VM based on hardware architectural features. HyperMon traces low-level events at the hypervisor to construct features. By utilizing supervised machine learning methods with those features, HyperMon builds a statistical model for program behaviors in a VM. The evaluation demonstrates the feasibility of HyperMon in identifying malicious behavior against VMI-based tools.

References

[1] T. Garfinkel and M. Rosenblum, "A virtual machine introspection based architecture for intrusion detection," Proc. Conference on Network and Distributed System Security Symposium (NDSS), pp.191–206, Internet Society, 2003.

[2] S. Laurén and V. Leppänen, "Virtual machine introspection based cloud monitoring platform," Proc. 19th International Conference on Computer Systems and Technologies, pp.104–109, ACM, 2018.

[3] A. Saberi, Y. Fu, and Z. Lin, "Hybrid-bridge: Efficiently bridging the semantic gap in virtual memory introspection via decoupled execution and training memoization," NDSS 2014, 2014.

[4] B. Jain, M.B. Baig, D. Zhang, D.E. Porter, and R. Sion, "Sok: Introspections on trust and the semantic gap," Proc. 2014 IEEE Symposium on Security and Privacy (SP), pp.605–620, IEEE, 2014.

[5] J. Demme, M. Maycock, J. Schmitz, A. Tang, A. Waksman, S. Sethumadhavan, and S. Stolfo, "On the feasibility of online malware detection with performance counters," Proc. 40th Annual International Symposium on Computer Architecture, ISCA '13, New York, NY, USA, pp.559–570, ACM, 2013.