International Journal of Advances in Engineering and Management (IJAEM)
Volume 5, Issue 1 Jan. 2023, pp: 133-139 www.ijaem.net ISSN: 2395-5252

# Virtual Machine Introspection Modeling For Virtualization-Based Privacy Preservation against Malware in Unstructured Network

Oviebor, E. O., Ogheneovo, E. E. & Egbono, F.
*Department of Computer Science, University of Port Harcourt, Port Harcourt, Nigeria.*

Date of Submission: 01-01-2023 Date of Acceptance: 08-01-2023

**ABSTRACT**
Unstructured network computing gives end users scalable virtualized on-demand services with more flexibility and with a less infrastructure investment. Under the supervision of various network service providers, these services are offered via the Internet utilizing well-known networking protocols, standards and formats. Existing flaws and weaknesses in unstructured network architecture and antiquated protocols frequently act as entry points for intrusion. This study used the Virtual Machine Introspection approach (VMI) to identify and stop various intrusions of various malware with distinguish term-sizes that could compromise the host machine's resources' availability, confidentiality and integrity. It examines and compares the VMI model with the Hypervisor Introspection technique (HI) for Intrusion Detection System (IDS) in unstructured network architecture and performed better than it.

**Keywords:** Virtual Machine, Malware, Hypervisor, Term-size.

## I. INTRODUCTION

Unstructured network is a model that enables on-demand network access in order to share computing resources like network bandwidth, storages, applications, and many more. It encourages rapid scalability with minimal service provider management (Kuyoro et al., 2011). It could be rapidly provisioned and released withminimal management effort or service provider interactions (Mell, and Grance, 2011).The unstructured network in the form of Cloud provides services in various forms: Software as a Service-SaaS (e.g. Google Apps (Google, http://www.google.com/apps/business).While the network offers many benefits, many of the major players may be tempted to stay back until some of the drawbacks are better recognized (Subashini andKavitha,2011).. How to close the semantic gap between the hypervisor and the virtual machine (Yacine , 2015) is one of the main problems that the Hypervisor Introspection-based solutions need to address.

Although an unstructured network like cloud computing is intended to minimize the majority of the client's workload by utilizing virtualization techniques and to give a healthier utilization of resources, it is rife with security risks (Keiko et al., 2013). As a result, we must safeguard data in the midst of unreliable programs (Kevin et al., 2010).A malicious program may attempt to determine whether they are being watched and then modify their behavior accordingly. This frequently occurs when malware is advanced (Martina et at., 2011). In order to identify any dangerous program changes or the execution of any strange or malicious code, Virtual Machine Introspection (VMI) examines the programs that are currently running in a virtual machine (VM) (Hebbal et al., 2015). Virtual machine monitor (VMM) technology, which was absent from conventional IDS methods, is used by VMI model. VMs are created and run by software known as a hypervisor or VMM. Application that want to use system call interface to express its demands or if it wishes to influence other apps or access any external resources, programs are contained by system call interposition-based application sandboxes, which limit the system calls that can be made by them and may change the arguments that can be passed to those calls (Samuel and Sampsa, 2017). Application-level sandboxes frequently use the system call interface's gate-keeping function to keep apps contained (Taesoo and Nickolai, 2013; Maximilian et al., 2019).

There are various ways to implement the VM introspection technique, which include the use of guest-OS hooks, VM state access, kernel debugging, interrupts and hypercall authentication. These methods help to close the semantic gap between the low-level data present in a VM and the high level semantic state of a VM. However, we will implement Application Programming Interface (API) call sequences that can easily represent the activity of malware in its code enabling simple detection of behavioral change within the network traffic, file modification, registry value modification, process creation, and many more.The existing hypervisor introspection system-conducted security solutions only introspect specific guest OS data structures to either protect them against kernel rootkit attacks or detect already injected malicious code inside the kernel. Intrusive attacks within the user-mode processes are very common in unstructured network platforms (Jonathan et al., 2018). Only a few of them make an effort to concentrate on user-mode process protection (Hofmann et al., 2013; Andrei et al., 2015; Chonghua et al., 2015**;**Vogl and Eckert, 2012).

Despite this, there are some intrusions that solely target user processes and cannot be stopped by just protecting the OS kernel. These malicious codes try to steal and manipulate user secret data, such as passwords, personally identifiable information (PII), emails, contacts, etc., attackers use suspicious application to exploits and gain access on user processes and manipulate important user data (e.g., the mail client, the Internet browser) do not require controlling the OS kernel. The virtual machine introspection (VMI) implementationtends to protect both the kernel and user memory sectors, our method seeks to provide defense against these kinds of privacy intrusion.We compared the Hypervisor Introspection (HI) technique that deploy the hardware assistance approach to perform introspection of hypervisor and the host OS kernel states to detect various attacks such as hardware attacks, rootkit attacks, and side channel attacks using various term-size sample of malware and benign file. We found out that, our model, the VMI perform better than the HI architecture.

## II. VIRTUAL MACHINE INTROSPECTION

The virtual CPU, memory, and disk contents are among the internal states and events that are observed and analyzed by the VMI tools. Our trampoline, acting in the capacity of an analyst, is granted access to that level of privilege by the hypervisor, which is in charge of managing interactions between hardware and the operating system (OS).It makes it possible for the hook to observe these actions from outside the virtual machines. Any attempt to modify a virtual machine after the hypervisor has established it will result in an alert being sent to the system administrator or tenant administrator at the beginning of execution.Based on an analysis of their graphs' data flow and information flow, our VMI approach separates malware from legitimate programs. The system call interface has been used for monitoring and profiling application behavior in addition to program containment (Aceto, et al., 2013). The algorithm below separates safe code from questionable user programs. Nodes of the graph are categorized as legitimate or anomalous at the end of the algorithm's test.

**Algorithm 1: For Detection of Malicious Node in a Graph**

```
Step 1: INPUT: Graph objects with code snippet
Step 2: OUTPUT: Decision: Malicious or Legitimate reset Flag;
Step 3: If Process creates or modifies Files OR directories OR registry entries Then
Step 4: Set Flag
Step 5: ElseIf Process generates another thread process OR initiates another process Then
Step 6: Set Flag
Step 7: Else If Process loads suspicious file OR reads data from suspicious file Then
Step 8: Set Flag
Step 9: End If
Step 10: If Flag is Set Then
Step 11: Update node with current Timestamp and label malicious;
Step 12: EndIf
```

## III. MALICIOUS OBJECTS LABELING

Any object's directed graph may have some trustworthy nodes as well as some harmful nodes. Each start point object has a data flow connection or information flow link that connects it to its descendants. Algorithm 1. This algorithm is used to identify harmful object nodes that are actually present in a graph.

**Algorithm 2: For Graph infection Detection**

```
Step 1: For A given graph G (V, E) Do
Step 2: For Every node V Do
Step 3: If Node V is malicious Then
Step 4: Label all its descendant nodes as malicious
Step 5: Label all its predecessor nodes as malicious
Step 6: EndIf
Step 7: EndFor
Step 8: End For
```

Due to the direction of malicious data and information flow, algorithm 2 identifies all nodes descending from node V as malicious. This is an external file or object, yet all antecedents of node X are considered malevolent because node X is accessed by its root. The process that is accessing the foreign object must be the start of malware.

**3.1 Malware Detection**

The files that can potentially be malware samples are chosen by algorithms 1 and 2. For future consideration, the graphs flagged as harmful by Algorithm 2 are taken into account. With the use of such a graph, our VMI technique has even the most basic understanding of the information flow and data flow of that object, as well as its communication with the other system entities. Since the entire graph is taken into account as a single potential malware code unit. Our system determines if a particular code is malware or a legitimate program based on its examination. By examining or introspecting, the sequence of API calls, checking for signature matches, and identifying dangerous malware attributes, code can be proven to be malicious.

**Algorithm 3: For Malware symptoms detection**

```
Step 1: INPUT: Code generated from Entire Graph
Step 2: Reset flag
Step 3: Detect all API calls, jump instructions and remote references
Step 4: If Found System Hook or ASEPs Then
Step 5: Set flag
Step 6: Else If Found Changes in File Properties Then
Step 7: Set flag
Step 8: End If
Step 9: If flag is Set Then
Step 10: Malicious code present
Step 11: End If
```

Algorithm 3 looks for fundamental characteristics that all malware possesses. The first check looks for a code that allows malware to set the Auto Start Extensible Point (ASEP). Every piece of malware configures a technique to enable activation upon system reboot. One of the techniques listed below is used to enable such mechanisms:

(a) Global Windows hooks setting.
(b) Modifying to multi extension executables.
(b) Modify registry entry attributes.

The code and information flow of a certain graph are examined using Algorithm 3 for these signs. If any of the specified collection of codes are found, an alert is generated. The later loop of the Algorithm 3 checks for modifications to the file's characteristics. The following are the implemented checks:

(a) File extension change
(b) Modifying file access control attributes

Algorithms 1, 2 and 3 looked at each potential dangerous program's flow individually. The aforementioned actions fall under the category of harmful activity. These circumstances enable the fundamental tasks that a malware application must perform in order to continue existing on the host computer. For instance, malware must build ASEP in order to activate itself upon every boot. One of the aforementioned methods can be used to do this.

**3.2 Methodology**

The malware analysis components operating on the introspected (host) PC can be detected by sophisticated malware programs without leaving any traces in the system, they attempt to disable or compromise the security tool. In order to prevent being attacked by sophisticated malware codes, we therefore opted to essentially monitor the Guest VMs by installing the security monitor at type two hypervisor. The model is installed at the security VM (Dom0) and is particularly created to offer VM introspection from outside the tenant VM. The technique makes use of the hypervisor's capabilities to introspect libraries, which accesses the VM memory regions through the guest symbol table.The method offers a high-level view of the memory of the Guest VM, which is examined by a security analyzer operating in the security VM (Dom0). The information is communicated to other security modules operating in the security VM via the security module (trampoline code) in the monitor VM. Security analyzer sends a warning to the cloud administrator if any of the VM memory areas are discovered to be suspicious. The model is designed in a way that service providers can track the behavior of the VMs from the VMM. We set up the IDS in the host VM's hypervisor. The method can identify insider VM attacks, VM-VM attacks, and in particular VM-VMM attacks. To further secure our model, we employ a number of tools, which includes, the Secure Server Commands, SECURE Commands, and Command Confirmation.

**Secure Server Commands**

The user's direct connection to the kernel is through the Secure Server. By hitting the Secure Attention Key, a user calls up a trusted route to the Secure Server. This key is always active and cannot be read by unauthorized code. The Secure Attention Key that we have selected is the BREAK key. Using commands like CONNECT, DISCONNECT, RESUME, and SHOW SESSIONS, the Secure Server manages terminal connections to virtual machines in the same manner that a terminal server manages terminal connections to physical machines.Users can simply move between sessions they have created with multiple virtual machines at various access classes. The Secure Server command interface is entirely composed of trusted code and provides only the barest of command-line editing features.

**SECURE Commands**

The SECURE commands are the management tools for the system. The VMS operating system comes pre-installed with the full collection of SECUREcommands and tools. Features like command-line recall and command procedures are available to the user. The system has two different types of secure commands: VM secure commands and User secure commands. The operating-system command level of the VM is used to issue both varieties of SECURE commands. The issuing VM is used to execute the VM SECURE instructions. The Secure Server receives the User SECURE commands and executes them. The type of subject a user or a virtual machine holding the access class and privileges required to issue the command distinguishes the commands.

**Command Confirmation**

While both the User and VM SECURE commands are administrative commands, only the User SECURE commands must be trusted. For such security-relevant commands, our system requires and assurances that:
(i) The command was issued by a user and not by a virus (Trojan horse) in a VM.
(ii)The command received by the Secure Server is exactly the same command typed by the user and not a command that was covertly modified by a Trojan horse.
(iii) The user who issued the command can be identified in the audit log.
Our design for the User SECURE commands provides both trust and individuality accountability even for commands issued from an untrusted environment.

**Hypervisor Introspection (HI)**

Hypervisor Introspection (HI) examines hypervisor-related data structures, memory regions, hypercalls, control flow data, non-control flow data, etc. It also works to stop and detect attacks on the hypervisor, which essentially take a low level view of the state of the virtual machine and cause a semantic gap for this technique. However, the VMI we provide aims to address this semantic difficulty because the method can access the high level view of the VM state.

## IV. RESULTS AND DISCUSSION

**Datasets**

In this experiment, we used system call datasets from eicar.com database. The datasets consist of program execution traces observed both in a synthetic environment and on real world machines with actual users virtual machine and under normal operating conditions. Different datasets were made used of in this experiment. The first ones are the collection of execution traces of malware samples randomly extracted from eicar.com. They are called malware and it includes a mixture of some categories such as worms, dropper, Trojan horses, etc. The second dataset is labeled as benign and it contains execution traces collected from ten different databases in eicar.com and it contains traces of certain benign applications executed under a real virtual machine. These datasets are originally in the form of 1-gram format. Each trace in the dataset is an execution trace of a process, which consist of set of system calls (Windows APIs) number. The details of classes and quantity of each malware dataset are shown in Table 1.

Table 1: Experimental Dataset

| S/N | Malware Class | Quantity | Percentage (%) | Maximum Size (KB) |
|---|---|---|---|---|
| 1 | Benign | 17,214 | 21.91 | 122,154 |
| 2 | Trojan | 16,146 | 20.55 | 88,547 |
| 3 | Virus | 19,821 | 25.23 | 99,784 |
| 4 | Worm | 13,541 | 17.23 | 76,000 |

| | | | | |
|---|---|---|---|---|
| **5** | Rootkit | 357 | 0.45 | 789 |
| **6** | Backdoor | 8,541 | 10.87 | 68,054 |
| **7** | Flooder | 1,424 | 1.81 | 29,785 |
| **8** | Spyware | 1,524 | 1.94 | 39,431 |
| | **Total** | **78,568** | **99.99** | **524,544** |

**Table 2:** Confusion Matrix

| | | **Actual Class** | | |
|---|---|---|---|---|
| | | **Malware** | **Benign** | **Total** |
| **Detected/Predicted Class** | **Malware** | **TP** | **FP** | **TP + FP** |
| | **Benign** | **FN** | **TN** | **FN + TN** |
| | **Total** | **TP + FN** | **FP + TN** | |

Table 3 contents the Confusion Matrix value for Term-Size 1 dataset sample for our experiment. These values are generated after the VMI and other HI system were subjected to dataset sample of 78,568 both malware binary and benign binary as shown in Table 1. The number of malware binary is 61,354 and 17,214 benign binary which gave us the total number of dataset mentioned earlier for this experiment. In table 3, we will observe that when the designed VMI and the HI system were subjected to the total number of dataset for possible content poisoning and other malware attack against a vulnerable client machine within an virtualized unstructured networkenvironment, the VMI was able to detect and prevent a total of 58,200 TP virus as against; 48,113 TP for HI, Other confusion matrix values for the VMI and the HI are shown in Table 3.

**Table 3:** Confusion Matrix values for Term-Size 1

| Existing System/Proposed System For Term-Size 1 | HI | VMI |
|---|---|---|
| Total number of sample tested | 78,568 | 78,568 |
| TP | 48,113 | 58,200 |
| FP | 3,923 | 946 |
| TN | 8 | 4,513 |
| FN | 0 | 0 |

In Table 4 with term-size 2, we could observe that the VMI shows a better performance of lesser FP detection and prevention of 2,425 malware files to that of the HI with the total number of detected and prevented malware binary as 4,700. For the benign codes the VMI detected and prevented a total of 2,425TN files when compared with the HI with total number of benign code detected and prevented to be 67 TN. When both systems were subjected with a more advanced malware and benign code with term size 2 as shown in Table 4, it indicate that the VMIhave a better perform as the term-size of the bit sequence of the malware and benign increases. Other appreciable feat of the VMI can be seen in Table 4.

**Table 4**: Confusion Matrix values for Term-Size 2

| Existing System/Proposed System For Term-Size 1 | HI | VMI |
|---|---|---|
| Total number of sample tested | 78,568 | 78,568 |
| TP | 48,113 | 58,200 |
| FP | 4,700 | 2,425 |
| TN | 67 | 2,952 |
| FN | 2,800 | 234 |

From the results achieved in the VMI in comparison to the HI system, we couldobserved the following in terms of performance metrics in table 5.

**Table 5:**

| **Algorithm** | **FP Rate** | **Accuracy** | **Term size** |
|---|---|---|---|
| HI | 0.998 | 0.785 | 1 |
|  | 0.986 | 0.839 | 2 |
| VMI | 0.452 | 0.958 | 1 |
|  | 0.452 | 0.958 | 2 |

**FP Rate:** The specificity of theVMIsystem to that of theHImodel as regards the term-sizes that ranges from 1, to 2.The FP Rate values for HI is higher than that of the VMI model valuesas recorded in table 5.
**Accuracy:** For accuracy, the VMI model has an outstanding accuracyfor detection and preventionof poisonous code when compared to the HIsystem with both term sizes of 1 and 2 as shown in table 5.

To evaluate the experiment and to examine the effectiveness of the proposed system to other existing system, we used the common evaluation metric that is widely used in information retrieval area and they are as follows:
**True Positive (TP):** Number of malware detected as malware
**False Positive (FP):** Number of malware detected as benign
**True Negative (TN):** Number of benign detected as benign
**False Negative (FN):** Number of benign detected as malware

**(a) FP Rate**: It is a measure of how many benign samples are labeled as malware by classifier.

$$FPRate=\frac{\mathrm{FP}}{\mathrm{FP+TN}} \quad (1)$$

(b) **Accuracy**: Accuracy is the proportion of true results (number of malware and benign detected correctly) in the total number of samples.

$$Accuracy=\frac{\mathrm{TP+TN}}{\mathrm{TP+FP+TN+FN}} * 100\% \quad (2)$$

## V. CONCLUSIONS

In the course of this study, we found out that the HI find it difficult to track down attackers with high level view during execution, it only depends on the low level view of the suspicious code to resolve poisonous program in a network. But, in VMI, we designed an appropriate modelthat is proactiveto limit malicious activities in the network, while malicious nodes are identified proactively. Hypervisor Introspection (HI) based security approach mainly depends on the hardware assistance to perform introspection of hypervisor/host OS kernel states and detect various attacks such as hardware attacks, rootkit attacks, and side channel attacks. The designed VMI can detect and prevent malicious programs both in the user space as well as in the kernel mode in real time.

## REFERENCES

[1]. Aceto, G., Botta, A., de Donato, W. and Pescapè, A. (2013). Cloud monitoring: a survey. Comput. Netw. 57(9), 2093–2115.

[2]. Andrei, L., Adrian C., Sandor, L. andDan H. L. (2015). Hypervisor-based protection of user-mode processes in Windows.Journal of Computer Virology and Hacking Techniques.

[3]. Chonghua, W., Xiaochun, Y., Zhiyu, H., Lei, C., Yandong, H. and Qingxin Z. (2015). Exploring Efficient and Robust Virtual Machine Introspection Techniques. International Conference on Algorithms and Architectures for Parallel Processing. 429–448.

[4]. Hebbal, Y., S.LaniepceandJ.-M., Menaud (2015). Virtual machine introspection: Techniques and applications. In: 10th International Conference on Availability, Reliability and Security (ARES). IEEE, 676-685.

[5]. Hofmann, O.S., Kim, S., Dunn, A.M., Lee, M. Z.,Witchel, E. (2013). Ink-Tag: secure applications on an untrusted operating system. SIGPLANNot.48(4), 265–278

[6]. Maximilian, B., Jakob, R., Christian, B., Tobias, M. and Hannes, F.(2019). State of the Sandbox: Investigating macOS Application Security.Application Security. In 18th Workshop on Privacy in the Electronic Society (WPES '19), London, UK. ACM, New York, NY, USA, 12.

[7]. Jonathan,G.,Irfan, A., Vassil, R.andManish, B. (2018). Automatic Mitigation of Kernel Rootkits in Cloud Environments. In book: Information Security Applications. 137-149.

[8]. Google apps. [Online]. Available: http://www.google.com/apps/business.

[9]. Kuyoro, S. O., F.,Ibikunle and O., Awodele (2011).Cloud Computing Security Issues and Challenges International Journal of Computer Networks (IJCN). 3(5), 247-255.

[10]. Martina, L., Clemens, K. and Paolo, M. C. (2011). Detecting environment-sensitive malware. In International Workshop on Recent Advances in Intrusion Detection. Springer, 338–357.

[11]. Mell, P. andGrance, T. (2011). The NIST Definition of Cloud Computing (Draft). NIST [Online]. Available:http://csrc.nist.gov/publications/drafts/800-145/Draft-SP-800-145_cloud-definition.

[12]. Yacine, H., Sylvie, L. andJean-Marc, M. (2015). Virtual Machine Introspection: Techniques and Applications. 10th International Conference on Availability, Reliability and Security (ARES).

[13]. Samuel, L. and Sampsa, R. (2017). A Survey on Application SandboxingTechniques (The ACM International Conference Proceedings Series). 8.

[14]. Keiko, H., David G. R., Eduardo Fernández-M. and Eduardo B. F.(2013). An analysis of security issues for cloud computing. Journal of Internet Services and Application. 5.

[15]. Taesoo Kim and NickolaiZeldovich. (2013). Practical and Effective Sandboxing forNon root Users.. In USENIX Annual Technical Conference. 139–144.

[16]. Subashini, S. andKavitha, V. (2011). A survey on security issues in service delivery models of cloud computing. Journal of Network and Computer Applications 34, 1–11

[17]. Vogl, S., and Eckert, C. (2012). Using hardware performance events for instruction-level monitoring on the x86 architecture. In: Proceedings of the 2012 European Workshop on System Security(EuroSec'12).

[18]. Kevin, H., K. Murat, K., Latifur and T. Bhavani (2010). Security Issues for cloud computing, International Journal of Information Security and Privacy, 4(2). 39-51.