www.ijaret.org Vol. 2, Issue I, Jan. 2014
ISSN 2320-6802

INTERNATIONAL JOURNAL FOR ADVANCE RESEARCH IN ENGINEERING AND TECHNOLOGY
WINGS TO YOUR THOUGHTS…..

INTRUSION FAULT TOLERANT AND SECURE COMMITMENT OF OS LEVEL VIRTUAL MACHINE

P. Srinivasan[1], S. Balakrishnan[2]

[1]M.E CSE, Rajiv Gandhi College of Engineering,
Sriperumbudur,
p.srinivasan@outlook.com

[2]Associate Professor, Rajiv Gandhi College of Engineering,
Sriperumbudur,
balkiparu@gmail.com

Abstract: *A virtual machine (VM) can be simply created upon use and disposed upon the completion of the tasks or the detection of error. The disadvantage of this approach is that if there is no malicious activity, the user has to redo all of the work in her actual workspace since there is no easy way to commit (i.e., merge) only the benign updates within the VM back to the host environment. A practical application is to allow users to install and try new applications without worrying about malware. In other words, if something abnormal happens, one can simply throw away the infected VM but our proposed system is fault tolerant it roll back to previous position. Changes within an OS-level VM include files, directories, and registry entries that are created, modified, and deleted by the processes running in the VM. Proposed system consists of four steps: grouping state changes into clusters, distinguishing between benign and malicious clusters, committing benign clusters, and recovering from malicious state. To reduce the false-positive rate when identifying malicious clusters, it simultaneously considers two malware behaviors that are of different types and the origin of the processes that exhibit these behaviors, rather than considers a single behavior alone as done by existing malware detection methods.*

Keywords: *VM, Malware, Malicious, Cluster, Host environment.*

1. INTRODUCTION

A Virtual Machine (VM) is a software implementation of a machine (i.e. a computer) that executes programs like a physical machine. A VM can only be committed when it has completed all the tasks and is at the stage of being shut down, because many objects and processes within a running VM cannot be merged into the host environment. Committing a VM overwrites files and registries on the host with the VM's private versions. As malware contributes to most security problems, to protect the integrity of the host environment, files and registries that have been attacked by malware programs should be discarded when committing the VM. First, the overhead of a commitment mechanism imposed on the host OS should be as low as possible because the virtual machine mechanism has already incurred no trivial overhead which leads to the performance degradation. Second, a commitment mechanism should be able to clean up all malicious changes rather than part of them. However, existing technologies such as logging and analysis, host-based intrusion detection and antimalware cannot simultaneously address both issues.

These techniques either cannot identify all malicious changes made by a malware program or incur a big overhead on a system although they may be effective in detecting intrusions. The main features of the system is that VM mainly leverages lightweight techniques such as tracing OS-level information flows and monitoring malware behaviors to ensure secure commitment, rather than uses logging technique which often incurs significant storage and time overhead, and even requires a backend host.

Secure commitment means merging only benign changes into the host environment but filtering out malicious changes when committing a VM. In our system, a scheme toward securely committing OS-level virtual machines, which is required by intrusion and fault tolerant applications and system administrations to save benign changes within a VM to the host environment. A VM can only be committed when it has completed all the tasks and is at the stage of being shut down, because many objects and processes within a running VM cannot be merged into the host environment. To identify malicious objects in a cluster fashion, first we have to address the challenge of correlating suspicious objects into clusters. The way we handle the fault tolerant is complicated since the system has to quarantine the malicious part of the objects in the cluster and should safely commit the benign objects to the host. . To classify the malicious and benign objects we use dependency graph and by tracing the OS information flow we identify the root of the graph. The scheme not only eliminate malicious clusters it should be fault tolerant to retrieve important information that are mixed with malicious objects otherwise all the necessary information are filtered out in the commitment. Moreover the scheme tend to have less false positive rate than other commercial antimalware tools available in the market.

2. LITERATURE REVIEW

An effective approach for realizing safe execution environments, Weiqing Sun, Zhenkai Liang, R.Sekar, V.N.Venkatakrishnan. It introduced an approach for realizing a [2] safe execution environment (SEE) that enables users to "try out" new software (or configuration changes to existing software) without the fear of damaging the system in any manner. A key property of our SEE is that it faithfully reproduces the behavior of applications, as if they were running natively on the underlying host operating system. This is accomplished via one-way isolation: processes running within the SEE are given read-access to the environment provided by the host OS. The taser intrusion recovery system, Ashvin Goel, Kenneth Po, Kamran Farhadi, Zheng Li, Eyal de Lara. Recovery from intrusions is typically a very time consuming operation in current systems. At a time when the cost of human resources dominates the cost of computing resources, we argue that next generation systems should be built with automated intrusion recovery [8] as a primary goal. In this paper, they describe the design of Taser, a system that helps in selectively recovering legitimate file system data after an attack or local damage occurs. Taser reverts tainted. This process is difficult for two reasons. First, the set of tainted operations is not known precisely. Second, the recovery process can cause conflicts when legitimate operations depend on tainted operations. To handle conflicts, Taser uses automated resolution policies that isolate the tainted operations. Container based OS virtualization:a scalable, high [1] Performance alternative to hypervisors, Stephen Soltesz, Herbert Pötzl, Marc E. FiuczynskI, AndyBavier and Larry Peterson. Hypervisors are quickly becoming commodity. They are appropriate for many usage scenarios, but there are scenarios that require system virtualization with high degrees of both isolation and efficiency. We present an alternative to hypervisors that is better suited to such scenarios. The approach is a synthesis of prior work on resource containers and security containers applied to general-purpose, time-shared operating systems. Design, implementation, and evaluation of a repairable file system, Ningning Zhu and Tzi-Cker Chiueh. The data contents of an information system may be corrupted due to security breaches or human errors. This project focuses on intrusion tolerance techniques that speedup the process of repairing a damaged file system. The pro-posed system, called Repairable File Service(or RFS), is specifically designed to facilitate the repair of compromised network file servers. An architectural innovation of RFS is that it is decoupled from and requires no modifications on the [7] shared file server that is being protected. RFS supports fine-grained logging to allow roll-back of any file update Operation, and keeps track of inter-process dependencies to quickly determine the extent of system damage after an attack/error. Defending browsers against drive-by downloads: mitigating heap-spraying code injection attacks, Manuel Egele, Peter Wurzinger, Christopher Kruege land Engin Kirda. Drive-by download attacks typically exploit memory corruption vulnerabilities in web browsers and browser plugins to execute shell [5] code, and inconsequence, gain control of a victim's computer. Compromised machines are then used to carry out various malicious activities, such as joining botnet, sending spam emails, or participating in distributed denial of service attacks. To counter drive-by downloads, they proposed a technique [3] that relies on x86 instruction emulation to identify JavaScript string buffers that contain shell code. Our detection is integrated into the browser, and performed before control is transferred to the shell code, thus, effectively thwarting the attack.

3. SECOM APPROACH

3.1 Overview

Committing a VM overwrites files and registries on the host with the VM's private versions. As malware contributes to most security problems, to protect the integrity of the host environment, files and registries that have been attacked by malware programs should be discarded when committing
the VM. The design of Secom is based on results obtained from our preliminary study of malware behaviors. To find an approach to identify malware objects from the contents of a VM, we have analyzed the technical details of a large number of malware samples from Symantec Threat Explorer [9] that stores analysis results of thousands of malware samples by analysts. With the study results, we design and develop a novel approach, Secom, to commit VM. It mainly leverages
lightweight techniques such as tracing OS-level information flows and monitoring malware behaviors to ensure secure commitment, rather than uses logging technique which often incurs significant storage and time overhead, and even requires a backend host. Secom consists of three steps, i.e., "correlate," "recognize," and "commit," which can [4] be conceptually depicted in Fig. 1. The first step correlates suspicious OS objects within a VM that are potentially malicious into different clusters. The second step recognizes real malicious clusters and marks them. The third step commits all OS objects in a VM to the host except the ones in malicious clusters or changes made on write-protected files.
A VM can only be committed when it has completed all the tasks and is at the stage of being shut down, because many objects and processes within a running VM cannot be merged into the host environment. For example, some objects (e.g., files) are often locked when accessed by some processes. In addition, the running of most processes often depends on some kernel objects, interposes communication (IPCs) objects or process properties that are tied with a specific VM. Moreover, committing a running VM may result in a partial merge of results from a task still being performed. Therefore, the "correlate" and "recognize" steps are executed when a VM is running, while the "commit" step is only executed after a VM is stopped. In the rest of this section, we describe the three steps involved in securely committing a VM.

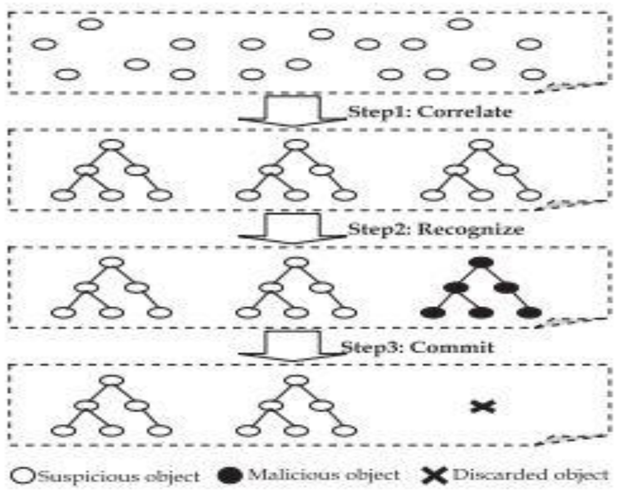

Figure 1: Secom approach

3.2 Correlating Suspicious Objects

One novel feature of our VM commitment approach is to identify malicious OS objects in a cluster fashion for a more efficient commitment, rather than one by one as done in traditional malware detection and analysis methods. Moreover, it is also able to remove malicious
objects more completely, because malware programs generate or modify a nontrivial number of files or registry entries on a single OS and only removing part of the objects being affected could not get rid of the impacts of a malware program thoroughly. [6] To identify malicious objects in a cluster fashion, first of all, we have to address the challenge of correlating suspicious objects into clusters. Since objects of a malware program often have various types and are scattered all over the system, it is difficult to associate them together. We observe that objects of a malware program can be correlated together by tracing information flows, and at the same time the malicious objects can be clearly separated from the other objects through a proper way of attaching cluster labels to them. Accordingly, we devise a novel approach to correlate suspicious objects into clusters, which includes tracing and labeling suspicious objects.

3.2.1 Tracing Suspicious Objects

As all malware programs come from either the network or removable drives, we treat the following objects as suspicious and start points to trace more suspicious objects, and we call them start-point objects: . Processes conducting remote communications; and. Executables (i.e., executable file) located at removable drives. An executable in this paper represents an executable file with a specific extension, such as .EXE, .COM, .DLL, .SYS, .VBS, .JS, BAT, and so on, or a special type of data file that can contain macro codes, say a semi executable, such as .DOC, .PPT, .XLS, .DOT, and so on. Secom does not allow a suspicious process to change the extension of a file to prevent its potential evasion of tracing. With these two rules, all malware programs that attempt to enter the system can be tracked as there are only two ways for them to break into system, either through network communications or through a removable drive. To track OS-level information flow, Back Tracker [2] is a successful approach. However, the major challenge is how to make sure that the entire system does not get marked as suspicious and at the same time malware programs cannot evade being traced. This actually requires a tradeoff between reducing the number of marked objects and reducing the risk of malware evasion. Our principle to achieve the tradeoff is to trace preferentially the information flows with a high risk of propagating malware programs while not tracing the information flows with a low risk. Based on this principle, we mark the following objects as suspicious:

1. Files, directories, and registry entries created or modified by a suspicious process.
2. Processes spawned by a suspicious process; and
3. Processes loading a suspicious executable file or reading a suspicious semi executable or script file.

The first rule records all permanent changes in a VM made by suspicious processes so that maliciously changed application data, executable files, system configurations, directories, registry entries, and so on can be filtered out thoroughly when committing a VM. To track the information flows with a high risk of propagating malware programs, the last two rules focus on tracing executables and processes. As an executable represents an inactive malware while a process represents an active malware, the information flows presented in these three rules have a high possibility of propagating malware programs. In the third rule, semi executable and script file possibly contain malware programs (e.g., macro virus in MS Word), and thus the processes reading them need to be marked. Although the macro virus protection in Office software can reduce the chances of macro virus infection, relying on it is very dangerous as crafted macro codes are able to subvert it and cause destructive damages. This has been observed in virus Melissa and W97M.Dranus.

4. ALGORITHMS

4.1 Loading the hash tables

The detection of malware is only based on their
predefined behaviors, the identification of malicious clusters cannot be dynamic over time. That is, a detection tool cannot recognize the new malware characteristics and behaviors that newly emerge. To address this issue, Secom is novelly implemented as an extensible mechanism for dynamically adding in new behaviors to be monitored.
A behavior can be defined as follows:

behavior:: = (Operation, Object, parameter, type).

Figure 2 presents how to load the four tables from the configuration file. For a given behavior, we fill it in a table according to the priority sequence:

HashTable1 → HashTable2→ HashTable3→ OverTable.

In each concerned system call, we use the operation, object, and parameters to determine whether the current access forms a malware behavior and the type of the behavior by searching through a carefully designed behavior table in the memory. The behavior table is read from a modifiable configuration file when the FVM boots up.

```
INPUT: Configuration file
 1: Load the configuration file into ConfigTable;
 2: FOR each behavior X in the ConfigTable
 3:   SN = HashFunc(X.object,ConfigTable);
 4:   IF(HashTable1(SN).B = true)
 5:     IF(HashTable1(SN).C = false)
 6:         Record the behavior X and the behavior corresponding
            to HashTable1(SN) into ConflictTable1;
 7:         HashTable1(SN).C = true;
 8:     ELSE
 9:         Record the behavior X into the ConflictTable1;
10:     END
11: ELSE
12:     HashTable1(SN).T = X.type;
13:     HashTable1(SN).B = true;
14:   END
15: END
16: FOR each behavior Y in the ConflictTable1
17:   SN = HashFunc(Y.parameter,ConflictTable1);
18:   IF(HashTable2(SN).B = true)
19:     IF(HashTable2(SN).C = false)
20:         Record the behavior Y and the behavior
            corresponding to HashTable2(SN) into
            ConflictTable2;
21:         HashTable2(SN).C = true;
22:       ELSE
23:         Record the behavior Y into the ConflictTable2;
24:       END
25:   ELSE
26:     HashTable2(SN).T = Y.type;
27:     HashTable2(SN).B = true;
28:   END
29: END
30: FOR each behavior Z in the ConflictTable2
31:   SN = HashFunc(Z.operation,ConflictTable2);
32:   IF(HashTable3(SN).B = true)
33:     IF(HashTable3(SN).C = false)
34:         Record the behavior Z and the behavior
            corresponding to HashTable3(SN) into OverTable;
35:         HashTable3(SN).C = true;
36:       ELSE
37:         Record the behavior Z into the OverTable;
38:       END
```

Figure 2: Loading the hash tables

4.2 Finding the type of a behavior

Each element of the former three tables has merely four bits. The first bit named "B" records whether there is a corresponding behavior, the second bit named "C" indicates whether there is a conflict, and the last two bits named "T" record the type of the corresponding behavior. The Over- Table stores the behaviors that conflict with each other in the former three hash tables. Each element of the OverTable records the complete information of a behavior including its operation, object, parameters, and type. The hash function is used to find the type of a given behavior from a given hash table. The type of a given behavior from the hash tables. We also follow the priority sequence to find the type of the behavior. The algorithms are efficient for an online application, as we do not need to perform string searching and only require four bits for a behavior.

```
INPUT: behavior X
 1: SN = HashFunc(X.object,HashTable1);
 2: IF(HashTable1(SN).B = false)
 3:     RETURN no behavior;
 4: END
 5: IF(HashTable1(SN).C = false)
 6:     RETURN HashTable1(SN).T;
 7: END
 8: SN = HashFunc(X.parameter,HashTable2);
 9: IF(HashTable2(SN).C = false)
10:     RETURN HashTable2(SN).T;
11: END
12: SN = HashFunc(X.operation,HashTable3);
13: IF(HashTable3(SN).C = false)
14:     RETURN HashTable3(SN).T;
15: END
16: FOR each behavior Y in the OverTable
17:       IF((Y.object = X.object) AND
        (Y.parameter = X.parameter) AND
        (Y.operation = X.operation))
18:           RETURN Y.type;
19:     END
20: END
```

Figure 3: Finding the type of a behavior

5. IMPLEMENTATION

5.1 Overview

Implementation includes all those activities that take place to convert from the old system to the new. The new system may be totally new, replacing an existing module or automated system, or it may be a major modification to an existing system. In either case proper implementation is essential to provide a reliable system to meet organization requirements. All planning has now, be completed and the transformation to a fully operational system can commence. The first job will be writing, debugging documenting of all computer programs and their integration into a total system. The master and transaction files are decided, and this general processing of the system are established. Programming is complete when the programs confirmed to the detailed specification. When the system is ready for implementation, emphasis switches to communicate with the finance department staff. Open discussion with the staff is important from the beginning of the [10] project. Staff can be expected to be concerned about the effect of the automation on their jobs and the fear of redundancy or loss of status must be allayed immediately. During the implementation phase it is important that all staff concerned be appraised of the objectives of overall operation of the system. They will need shinning on how computerization will change their duties and need to understand how their role relates to the system as a whole. An organization-training program is advisable, this can include demonstrations, newsletters, seminars etc.

The department should allocate a member of staff, who understands the system and the equipment, and should be made responsible for the smooth operation of the system. An administrator should coordinate the users to the system. Users should be informed about new aspects of the system that will, affect them. The features of the system explained with the adequate documentation. New services such as security, on-line application form and back-ups must be advertised on the staff when the time is ripe.

5.2 Architecture

To demonstrate the applicability of the OS-level VM commitment approach, we have successfully developed a prototype under the framework of the FVM [7] that partitions the name space of a single Windows OS to form a number of virtual machines. The implementation codes of VM commitment stay together with an FVM virtualization layer that consists of a kernel driver and a user level DLL. Besides, the VM commitment mechanism has an extra VM committing module inside the FVM management tool at the user level. Fig 4 shows the general architecture. In the figure, the “Correlating” module is responsible for correlating suspicious objects into clusters. We intercept Windows system calls at the kernel level and Win32 API functions at the user level to attach a proper cluster label to each suspected object according to the tracing and labeling methods. Most of the interceptions are located in the kernel rather than the user level so that it is difficult to be bypassed. For the permanent objects, the labels of files and directories are stored in a specially created stream of each file or directory. The labels of registry keys are recorded in a file under the VM’s directory, which holds all objects changed by the FVM. However, for the volatile objects, e.g., processes, their labels are temporarily stored in memory. In addition, each cluster has a data structure to record whether it is malicious. The “Recognizing” module is responsible for monitoring raw behaviors and determining whether a cluster is malicious according to the decision logic of the detection engine. All raw behaviors are extracted by intercepting a single essential system call/API function and analyzing the parameters. For example, monitoring NtCreateKey() for “Create Windows service.” Some malware behaviors consist of more than one system call or Win32 function, for instance, the behavior “Inject into other processes” consists of Open- Process(), VirtualAllocEx(), WriteProcessMemory(), Create-Remote-Thread(), and so on. We only intercept the first essential function, i.e., OpenProcess(). Moreover, to prevent intended bypassing, we always intercept a function at the kernel level rather than the application level if possible Thus, for the behavior “Inject into other processes,” we actually intercept NtOpenProcess() at the kernel level rather than OpenProcess() at the application level. The “Committing” modules in the kernel and the FVM management tool are responsible for committing virtual machines. The modules first scan the changed files, directories, and registry keys that are stored under the directory of the VM to be committed, and then drop or merge the changed items into the host environment. The commitment modules are only called when a user requests to delete or commit a VM that has already been stopped.

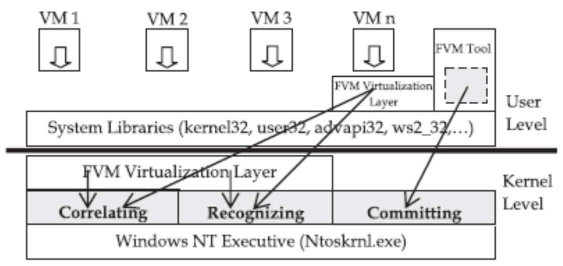

Figure 4: General Architecture

5.3 System Architecture

In the existing system if the malicious cluster has some important information they are eliminated while committing to the host. By this way we can have a secure system but not a robust one. Fig 5 build a fault tolerant system we changed the committing rule of the existing system to quarantine the malicious cluster by roll back to stage before committing. The proposed system leverages on OS information flow to find the root cause of the malicious cluster. The proposed system has less false positive rate when compared to other antimalware tool available in the market.

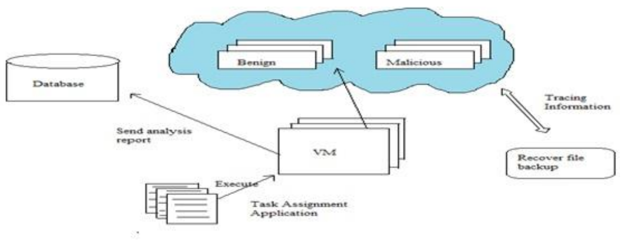

Figure 5: System Architecture

6. CONCLUSION

In this paper, a scheme toward securely committing OS-level virtual machines, which is required by intrusion-tolerant applications and system administrations to save benign changes within a VM to the host environment. So far, none of the publicly available documents on OS-level virtualization technologies ever provides a feasible scheme to securely commit VM. The critical challenge behind securely committing VM is to identify compromised objects thoroughly and lightly. To address the challenge, our system consists of three steps. First, it correlates suspicious OS objects, then, it recognizes a malicious cluster by a behavior-based malware detection engine. Last, it commits VM while discarding malicious clusters. To reduce the false-positive rate and to be fault tolerant, System considers two malware behaviors which are of different types and the originator of the processes which exhibit the behaviors when identifying a malicious cluster. Moreover, compared with commercial antimalware tools, it can erase malware more thoroughly and produce a lower false-positive rate. Hence, it fits the task of VM commitment better.

REFERENCE

[1] "Container-based Operating System Virtualization: A Scalable, High-performance Alternative to Hypervisors" by Stephen Soltesz, Herbert Pötzl, Marc Fiuczynski E, Andy Bavier, Larry Peterson at EuroSys'07, March 21–23, 2007, Lisboa, Portugal

[2] "One-way Isolation: An Effective Approach for Realizing Safe Execution Environments" by Weiqing Sun, Zhenkai Liang, Sekar R, Venkatakrishnan V N

[3] Yu Y, Govindarajan H K, Lam L, and Chiueh T, "Applications of Feather-Weight Virtual Machine," Proc. Int'l Conf. Virtual Execution Environments (VEE), Mar. 2008.

[4] Paleari R, Martignoni L, Passerini E, Davidson D, Fredrikson M, Giffin J, and Jha S, "Automatic Generation of Remediation Procedures for Malware," Proc. USENIX Conf. Security, Aug. 2010.

[5] Egele M, Wurzinger P, Kruegel C, and Kirda E, "Defending Browsers against Drive-By Downloads: Mitigating Heap-Spraying Code Injection Attacks," Proc. Sixth Int'l Conf. Detection of Intrusions and Malware, and Vulnerability Assessment (DIMVA), July 2009.

[6] Zhu J, Jiang Z, Xiao Z, and Li X, "Optimizing the Performance of Virtual Machine Synchronization for Fault Tolerance," IEEE Trans. Computers, vol. 60, no. 12, pp. 1718-1729, Dec. 2011.

[7] Zhu N and Chiueh T, "Design, Implementation, and Evaluation
of Repairable File Service" Proc. Int'l Conf. Dependable Systems and
Networks (DSN),pp. 217-226, 2003

[8] Goel A, Po K, Farhadi K, Li Z, and Lara E, "The Taser Intrusion Recovery System," Proc. 20th ACM Symp. Operating Systems
Principles (SOSP),Oct. 2005.

[9] Sukwong O, Kim H, and Hoe J, "commercial
Antivirus Software Effectiveness: Empirical
Study, "Computer, Vol. 44, no.3, pp.63-70,
Mar. 2011.

[10]Rieck K, Holz T, Willems C, Dussel P, and Laskov P, "Learning and Classification of Malware Behavior," Proc. Fifth Int'l Conf. Detection of Intrusions and Malware, and Vulnerability Assessment (DIMVA), pp. 108-125, June 2008.