Safe Side Effects Commitment for OS-Level Virtualization

Zhiyong Shan[†]
shanzhiyong@ruc.edu.cn

Xin Wang[‡]
xwang@ece.sunysb.edu

Tzi-cker Chiueh[‡*]
chiueh@cs.sunysb.edu

[‡]Stony Brook University
[†]Key Laboratory of Data Engineering and Knowledge Engineering (Renmin University of China), MOE
[*]Industrial Technology Research Institute

ABSTRACT

A common application of virtual machines (VM) is to use and then throw away, basically treating a VM like a completely isolated and disposable entity. The disadvantage of this approach is that if there is no malicious activity, the user has to re-do 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. In this work, we develop a VM commitment system called Secom to automatically eliminate malicious state changes when merging the contents of an OS-level VM to the host. Secom consists of three steps: grouping state changes into clusters, distinguishing between benign and malicious clusters, and committing benign clusters. Secom has three novel features. First, instead of relying on a huge volume of log data, it leverages OS-level information flow and malware behavior information to recognize malicious changes. As a result, the approach imposes a smaller performance overhead. Second, different from existing intrusion detection and recovery systems that detect compromised OS objects one by one, Secom classifies objects into clusters and then identifies malicious objects on a cluster by cluster basis. Third, 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. We have successfully implemented Secom on the Feather-weight Virtual Machine (FVM) system, a Windows-based OS-level virtualization system. Experiments show that the prototype can effectively eliminate malicious state changes while committing a VM with small performance degradation. Moreover, compared with the commercial anti-malware tools, the Secom prototype has a smaller number of false negatives and thus can more thoroughly clean up malware side effects. In addition, the number of false positives of the Secom prototype is also lower than that achieved by the on-line behavior-based approach of the commercial tools.

Categories and Subject Descriptors

D.4.6 **[Operating Systems]:** Security and Protection

General Terms

Reliability, Security

Keywords

Virtual machine, Malware, Virtual machine commitment

ICAC '11, June 14–18, 2011, Karlsruhe, Germany.

1. Introduction

An OS-level VM has minimal startup/shutdown cost, low resource requirement and high scalability due to its sharing of the execution environment of the host operating system and confining state changes within the VM's environment. It is thus an excellent vehicle for tolerating intrusions and faults, as well as consolidating servers. 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. One disadvantage of this approach is that if all processes run normally within a VM and there is no malicious activity, a user has to re-do all the work in her actual workspace since there is no secure commitment mechanism to save the benign changes within the VM back to the host environment. 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. **Secure commitment** means merging only benign changes into the host environment but filtering out malicious changes when committing a VM. Example applications of such secure commitment include:

1. Committing running results of enterprise applications. In order to consolidate servers, many enterprises often run important applications within VMs [3][4], e.g., web server, file server, database server. Once abandoning the VMs, valuable data and files generated by the applications, e.g., sale data, customer information, product files, technical files, and configuration data, have to be carefully preserved through a secure commitment mechanism.
2. Synchronizing states of fault tolerant applications. To sustain a mission-critical application, fault tolerant systems often run a primary instance and a backup instance of the application in different VMs [16][1], and synchronize states from the primary to the backup every tens to hundreds of milliseconds [31]. However, the backup instance may be immediately compromised once the primary one is compromised. As existing anti-malware technologies can not address this issue properly, it desires a mechanism to quickly and thoroughly filter out malware impacts when performing the state synchronization.
3. Committing working results of a user. To preserve the integrity of a computer system, a user can work within the space of a VM [7][13] to handle emails, download pictures, edit MS Word files, surf Internet, etc. When deleting the VM, all of the working results will get lost. However, the secure commitment mechanism can help the user to save benign results to the host environment.
4. Committing installation results of untrusted software. Freeware, shareware or mobile code can be downloaded or executed within a VM to minimize the security risk [7][13]. Without a secure commitment support, such software and the

running results during the installation will be removed when throwing away the VM.

Although it is of high importance to ensure secure commitment, there is a great challenge to identify all malicious changes which are mixed together with benign ones. Existing OS-level VM technologies, e.g. FreeBSD Jail [2], Linux-VServer [3], Solaris Zones [4], OpenVZ [5] and Virtuozzo [6], do not provide the function of securely committing VM. Though SEE [7] can resolve conflicts on committing file modifications, it can not identify compromised files.

There are two issues to address in order to build a secure commitment mechanism in the framework of an OS-level virtual machine. First, the overhead of a commitment mechanism imposed on the host OS should be as low as possible since 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 anti-malware can not address the two issues simultaneously. These techniques either can not 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.

In this paper, we aim to provide a light-weight commitment approach, named Secom, for an OS-level virtual machine to prevent malicious changes from being merged into the host. To our best knowledge, this is the first effort towards building a secure and practical VM commitment mechanism. As the issue of resolving inconsistency from concurrent modifications of the same resource by multiple processes running within different VMs has been addressed in the literature [7][8], it is not discussed in this paper.

Our proposed approach can automatically filter out malware impacts when committing the content of a VM into the host environment. Thus, a user does not need to manually scan the VM to be committed using anti-virus software each time. Moreover, a user also does not need to install, manage and frequently upgrade the anti-virus software. Therefore, our approach is useful when building self-healing or self-protection systems based on VMs.

The approach consists of three stages. First, it correlates suspicious objects within a VM into a number of clusters by tracking OS-level information flows and attaching a cluster label to each object. Objects in a cluster are only possible to be either all benign or all malicious. Second, it determines malicious clusters using an online malware detection engine to monitor malicious behaviors. Last, it merges benign changes in a VM to the host environment while discarding malicious clusters.

Secom has three novel features. First, unlike the commitment approaches assumed in other fields (e.g. database) which rely on a huge volume of log data, it leverages use of OS-level information flows and malware behaviors to perform secure commitment. As a result, Secom imposes a smaller overhead on host OS, while using a conventional data-logging method would significantly slow down the whole system. Second, different from existing intrusion detection and recovery systems that detect compromised OS objects one by one, it puts correlated objects into clusters thus identifying and discarding compromised objects cluster by cluster. Finally, different from existing behavior-based malware detection methods, it monitors a pair of malware behaviors and labels the sources of the processes that launch the behaviors rather than monitors a single behavior.

The contributions of this paper are five-fold.

1. We propose the first secure commitment approach, Secom, for OS-level VM to identify compromised OS objects and selectively merge only legitimate changes into the host. Its three novel features allow it to complete the task in a light-weight but efficient manner.
2. We propose a clustering approach to segregate benign and malicious changes within a VM. The approach relies on the starting and tracing rules to trace OS-level information flows to collect changes, and the labeling method to group collected changes into clusters.
3. We devise a novel approach to track OS-level information flow. It traces only suspicious processes and executable files that represent malware program themselves rather than tracing all kinds of files, directories and IPC objects. This can avoid the disadvantages of a classical malware tracing approach, which imposes a heavy performance impact on the system or makes the entire system floating with suspicious labels.
4. We propose a new behavior-based malware detection approach. To better identify a malware program with a lower false positive rate, it makes a decision based on two malware behaviors that belong to different types and sources of the processes which exhibit the malware behaviors. Our experiments showed that the number of false positives of our method is much smaller than that of existing online malware detection approaches.
5. We have implemented a prototype of Secom on FVM on Windows. Experiments show that it can effectively filter out a number of real-world malware programs while imposing only a small overhead on FVM. Moreover, it filters malware objects more thoroughly than that of commercial anti-virus software.

Secom approach depends only on tracing OS-level information flows and monitoring malware behaviors without the need of technical details of a specific virtual machine. Therefore, although Secom is designed for OS-level virtual machines, with some changes it should also be applicable to other types of virtual machines (e.g. hardware-level virtual machines) or general operating systems in order to clean up malware impacts. Investigating these applications is beyond the scope of this paper.

In the rest of the paper, we first describe the Secom approach for secure commitment in Section 2, and then the prototype is evaluated from the perspectives of secure commitment and performance overhead in Section 3. Last, we provide related work in Section 4 and conclude the work in Section 5.

2. Secom Approach

2.1 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

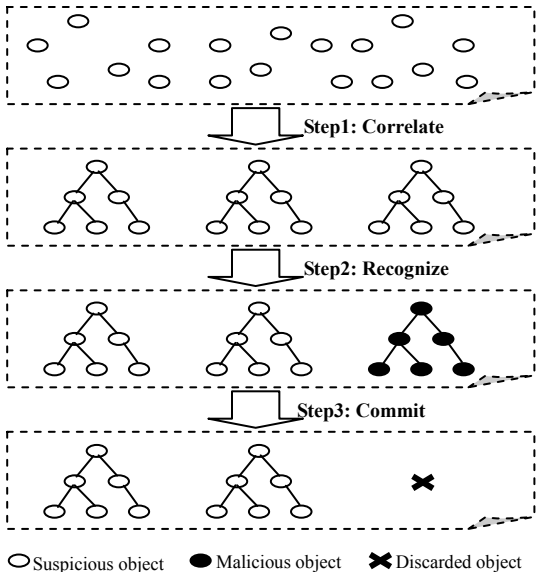


Figure 1. Secom approach

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 light-weight 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 be conceptually depicted in Figure 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 can not 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, inter-process communication 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.

2.2 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 non-trivial 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.

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.

2.2.1 Tracing Suspicious Objects

As all malware programs come from either the network or removable drives, we treat the following objects as start-points to trace suspicious objects, calling them **start-point objects**.

- *Processes conducting remote communications;*
- *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, etc, or a special type of data file that can contain macro codes, say a semi-executable, such as .DOC, .PPT, .XLS, .DOT, etc. Secom does not allow a suspicious process to change the extension of a file in order 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, BackTracker [10] 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 can not evade being traced. This actually requires a trade-off between reducing the number of marked objects and reducing the risk of malware evasion. Our principle to achieve the trade-off 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.

- *Files, directories and registry entries created or modified by a suspicious process;*
- *Processes spawned by a suspicious process;*
- *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

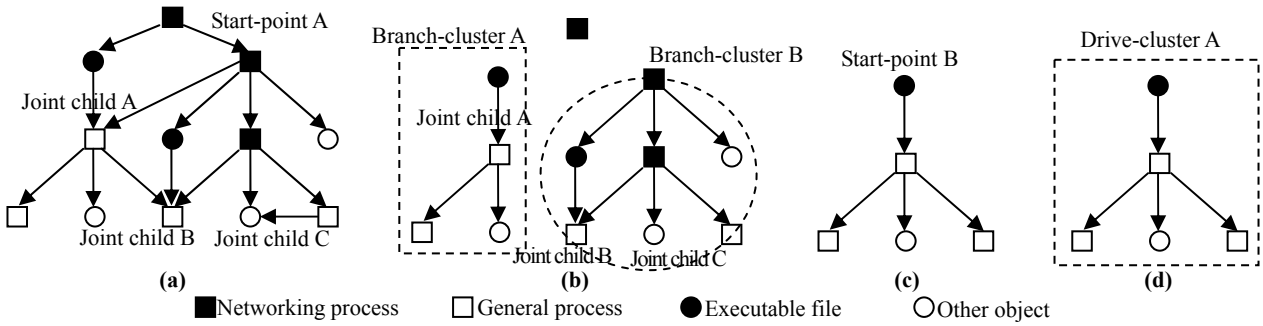


Figure 2. Dependency graph samples and obtained clusters

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.

To avoid tracking the information flows with a low risk of propagating malware programs, the rules do not trace most reading operations on files, directories and registry entries, which are frequently invoked but difficult to propagate malware programs. However, subtle malware programs might evade tracing by changing registry entries or configuration files which subsequently affect the processes reading them, so as to run malicious executables, escalate privileges, impose damages on system, etc. No matter what evasion schemes the malware programs utilize, they need to run their own executables to perform the tasks, which are downloaded from the network, copied from removable drives, or obtained from changing local executables. Since all executable related operations are thoroughly traced by the first and third rules, the malware programs will be captured when trying to load their executables. The two rules are applicable to all existing malware programs because they rely on their own executables to perform malicious tasks on a host, according to our analysis on Symantec Threat Explorer. In case that a malware program relies only on benign programs to perform attacks, the first starting rule will capture it as it would require a remote communication to accept commands to drive the benign program to perform the malicious tasks. In addition, for a few special registry entries and configuration files that can be used by a malware program to fool a benign program to execute arbitrary commands, Secom forbids a suspicious process to modify them. Therefore, although the reading operations on registry entries or configuration files are not traced, malware programs still can not evade being detected by Secom.

To reduce the number of marked processes, the rules do not trace IPCs (Inter-Process Communication). As a result, there is a risk of evasion, i.e., a malware program can bypass tracing via an IPC. However, the risk is very limited because the overwhelming majority of vulnerable IPCs can only be used to launch denial-of-service attack, disclose sensitive information, or escalate the privileges of the processes that send IPC data, rather than take control of the receiver process. Accordingly, they can not be used to propagate malware programs. Moreover, IPCs that can propagate malware programs often rely on network (e.g., Remote Procedure Call) and thus are traced by the first starting rule, according to our investigation on Microsoft Security Bulletins [11], a primary source for analyzing attack vectors of Windows OS [12].

2.2.2 Labeling Suspicious Objects

In this section, we employ the dependency graph to describe how to attach cluster labels to suspicious objects. Actually, for each start-point object, its descendent objects are connected to each other by information flows and form an existent but invisible dependency graph, which had been disclosed by the literature work [10]. The graph is a directed graph and has the start-point object as its root node. Its nodes represent OS objects, e.g. file, process. Its edges represent information flow related operations, e.g. creating a process, modifying a file. Figure 2 (a) and (c) show two dependency graphs which are derived from a networking process and an executable file respectively. Note that, we do not intend to really generate dependency graphs to help label objects since this would not be applicable to an online approach. Instead, the labeling methods are implemented together with the starting and tracing rules as follows: when an object is determined as suspicious by starting or tracing rules, a proper cluster label, i.e. a number and a time stamp, will be attached to it at the same time in order to denote that it is a suspicious object and belongs to the cluster identified by the label. In other words, the labeling methods are enforced along with the starting and tracing rules in real-time, rather than generating a dependency graph and then analyzing it.

When a start-point object is a network facing process, its dependency graph is too coarse grained to be used to recognize malicious objects in a cluster fashion since it might contain both benign and malicious objects. In other words, we can not determine that all objects in a graph are malicious even if most of the objects in the graph are recognized as malicious. Thus, we have to partition the graph into a number of sub graphs, say clusters, so that each cluster contains either only benign or only malicious objects.

According to the recent researches [20][27] and our analysis on thousands of malware descriptions in the Symantec Threat Explorer [9], malware programs break into a host through three basic attack channels. The first is that, malware programs exploit bugs in network-facing daemon programs or client programs and compromise them, then immediately spawn a shell or back-door process [20]. After this, the attacker typically tries to download and install attacking tools and rootkits, as well as performs any other adversary actions. Accordingly, we give a cluster label to a process directly spawned by a network-facing process as well as its descendants, calling them a **branch cluster**, e.g., the Branch-cluster B in Figure 2 (b). A branch cluster corresponds to a sub graph of a dependency graph which roots from a network-facing process. The other attack channel is that, malware programs lure users into downloading and launching them [27]. After started, malware programs copy themselves and make themselves resident in a host. Consequently, we give a cluster label to the downloaded

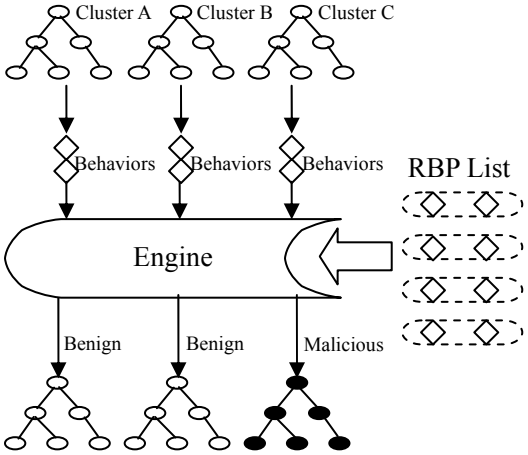


Figure 3. Malware detection engine. It monitors raw behaviors and marks a cluster as malicious if the cluster exhibits any predefined RBP.

executable and its descendants, calling them a **branch cluster** as well, e.g., the Branch-cluster A in Figure 2 (b). The last channel is removable drives. Therefore, we give a cluster label to an executable file located in a removable drive and all its descendent objects, calling the formed cluster a **drive cluster**, e.g., the Drive-cluster A in Figure 2(d).

Another issue for labeling objects is about a joint child who has multiple parent nodes in a dependency graph, e.g. the joint children A to C in Figure 2 (a). That is, when the parent nodes belong to distinct clusters, we have to determine the cluster label of the joint child. Basically, we make decision according to the priority sequence like "process' executable → parent process → other objects". Obviously, the joint child should inherit the cluster label from its parent process or executable file (i.e. a process' image file) if either of them exists instead of other objects. Moreover, as loading an executable is posterior to creating a process and necessarily overwrites the newly created process' code segment, the new process' activity is based on the loaded executable. Hence, the joint child should inherit the label from the loaded executable rather than the parent process if both exist. If more than one parent node has the same priority in the sequence above, the child inherits their labels in the reverse time order. Consequently, the joint children A to C are classed into Branch-cluster A and B respectively, as shown in Figure 2(b).

Splitting a dependency graph into different branch clusters might cause a piece of malware to be split into two separate processes on different branch clusters, which could work together to perform malicious actions and potentially evade Secom's detection. We can prevent this evasion at the time to commit a VM, which will be presented in Section 2.4. Our experiments in Section 4.1 demonstrate the effectiveness of the labeling approach, and Figure 4 further illustrates the labeling results of the objects generated by two in-the-wild malware samples.

2.3 Recognizing Malicious Clusters

To recognize a malicious cluster, we build an on-line engine to monitor whether the processes in the cluster exhibit any malware behaviors. Recent research efforts [22][21] on behavior-based malware detection often employ dynamic data flow tracing techniques to extract featured malware behaviors. The tracing of dynamic data flow involves a big overhead, which significantly slows down the system and is thus not applicable for on-line monitoring [25]. On the other side, to complement with the traditional signature-based detection, commercial anti-virus software often has an online behavior-based malware detection engine. However, the engine identifies a malware program only based on a single suspicious behavior which might also appear in benign software, and thus frequently produces false alarms that distract users [23][22].

Different from existing efforts, our malware detection engine detects malware by combining the techniques of tracing OS-level information flow and online malware detection. More specifically, based on the clusters formulated as a result of tracing OS-level information flow, the engine monitors all the behaviors of processes in a cluster and determines whether a cluster is malicious. A cluster is considered malicious if it exhibits two behaviors that match a predefined raw-behavior-pair, as shown in Figure 3. A **raw-behavior-pair** (**RBP**) consists of two independent raw behaviors. A **raw behavior** is extracted by intercepting a single system/API call and its parameters. It can differentiate malware from benign programs but may result in a few false positives. For example, "modifying registry for automatic startup" is such a malware behavior.

Targeting for on-line detection, however, we don't expect that detecting malware based on a single raw behavior to be as accurate as done through the featured behaviors [22][21], which are extracted via correlating two system/API calls by dynamically tracing data flows at a high overhead. Instead, to reduce the false positive rate, our detection engine uses a RBP to detect malware. If the false positive rate (FPR) of the detection based on a single raw behavior is p, the FPR which uses a RBP consisting of two independent behaviors will be p^2. This will make the false positive rate much lower than that generated with commercial online detection techniques in anti-virus software [23] which only rely on a single raw behavior to identify malware. If the raw behaviors in a RBP are carefully selected, the FPR of the RBP-based detection will be as low as that of the detection based on the featured behaviors. Moreover, since all clusters derive from dangerous sources, i.e. the network and removable drives, our detection approach actually considers not only malware behaviors but also the sources of the process launching the behaviors when determining a malicious cluster. As a consequence, the FPR is further reduced as demonstrated in our performance studies.

Our method is essentially in agreement with the recent research results on behavior-based malware detection [22][21], where two system/API calls are monitored to extract a featured behavior to detect malware accurately. However, different from existing techniques, we trace OS-level information flow instead of data flow, so our method would be much more efficient. As confirmed by studies in [17], a set of independent behaviors can be used to differentiate a specific category of malware programs from other categories and benign programs with very few false positives. In other words, the set of independent behaviors can be used to detect malware effectively. We may use more independent behaviors than two as done in the researches, but this will increase nontrivial false negatives and performance overhead while not reducing the number of false positives significantly. Thus, we choose to monitor two behaviors to detect a malware program.

However, there is a big challenge to realize the system using RBP to detect malware. Commercial online engines [23] can not achieve the goal because they can not correlate two behaviors which may exhibit at different time and associate them with a single malware program. For example, two behaviors may be

launched by a malware program's two distinct processes respectively. Although data flow tracing techniques [22] can find potential dependency between the behaviors, these techniques will levy unacceptable heavy overhead on the system. Instead, with an intelligent tracing of OS-level information flow, our detection engine can correlate the two processes and then naturally associate the two behaviors together.

To increase the opportunity of detecting wide categories of malware programs, the administrator might add a large number of RBPs. However, this makes the configuration very tough. Even worse, the huge number of RBPs will significantly slow down the system as the detection engine needs to frequently search through the long RBP list. Instead, we devise a scheme to address these issues as follows.

We determine a cluster as malicious if it exhibits two different types of raw behaviors. After breaking into a system, a malware program generally fulfills malicious tasks through four basic steps: (Ⅰ) making itself to auto-start after system booting, (Ⅱ) propagating itself across the system, (Ⅲ) hiding itself from users and anti-malware tools, and (Ⅳ) achieving malicious goals. Each step can be fulfilled by a type of behaviors. Accordingly, malware behaviors can be divided into four types. Although not every malware program requires all the four types of behaviors, a malware program does invoke at least a few types of the behaviors. Therefore, a RBP can be constructed with any two raw behaviors that belong to different types respectively. This way, we do not need to maintain a long RBP list to search, but only make detection based on the types of the behaviors.

A crafted malware program might exhibit only one type of raw behaviors instead of two to avoid matching any RBP. Then, it waits until it is committed to the host environment to perform malicious behaviors. However, if the exhibited behaviors in the VM are any type but type Ⅰ, the malware program will be actually disabled after merged into the host environment since it is not able to auto-start anymore. Hence, we only need to consider the behaviors of type Ⅰ that represents the behaviors hooking ASEP (Auto-Start Extensibility Point [24]). An ASEP is used to enable auto-starting of programs without an explicit user invocation, and thus becomes a common target of infection by malware programs [13][26]. Only by setting up an ASEP, a malware program can make itself resident in a host. To fight against such malware programs, at the committing stage, we will discard any changes on ASEP if the corresponding cluster is not derived from a trusted source. Thus, such malware programs will be disabled after the commitment without setting an ASEP in advance.

In case that some malware programs try to thwart the Secom mechanism, we deny some intentional bypassing behaviors including "Create global Windows hooks", "Create multi-extension executables", "Change file extensions", "Change file access control attributes", "Change registry entry attributes" and "Execute Secom special system calls". However, disallowing some behaviors (such as changing a file extension) makes it easier for malware programs to detect the presence of Secom. The malware programs might simply decide to stay dormant in this case, which undermines the malware detection capability. Therefore, we also mark the cluster requesting any bypassing behavior as malicious when disallowing the behavior.

Consequently, the decision logic of the detection engine can be concisely presented by the following five independent rules. (1) If the process launching the current raw behavior does not belong to any cluster, i.e., the process is not derived from dangerous sources including the network and removable drives, the engine will not trigger any action. (2) If the process belongs to a cluster and the cluster does not exhibit any raw behavior so far, the engine will record the type of the current behavior into the cluster. (3) If the types of the current and the recorded behaviors of the cluster are different, the engine will mark the cluster as malicious. (4) If the types of the behaviors are the same, the engine will not take any action. (5) If the current behavior is a bypassing behavior, the engine will mark the cluster as malicious and at the same time refuse it.

There are two potential evasions to our detection approach. First, a malware program might disguise itself as a multi-extension executable (e.g., help.txt.exe) in order to lure the user to launch it after the commitment since it does not have an ASEP. However, this malware program will also be detected as the behavior "Create muti-extension executables" is treated as a bypassing behavior as mentioned previously. The other evasion is that, a malware program does not exhibit any behaviors monitored by Secom. However, such a malware program can not make itself resident in the host, hide itself from anti-malware tools, or even propagate itself in the system. Thus, it will be disabled after being committed into the host environment since it does not set up an ASEP.

A question on the approach of identifying malicious clusters is why only marking rather than blocking after detecting a cluster to be malicious? As some malware programs bind together with useful software or processes [24], forcibly blocking malicious behaviors will make the software or processes unstable or unusable. For example, to hide from anti-virus software, a worm runs as a thread inside a system process. Unexpectedly blocking its behaviors might cause the process and even the whole system to crash as malware programs are often buggy [24]. Moreover, blocking malware behaviors will prevent a VM from performing some important tasks, e.g. analyzing malware programs, assessing system vulnerability [13], which would require that the malicious codes can run smoothly within a VM.

2.4 Committing Benign Clusters

When a VM is stopped and the user requires deleting or committing the VM, Secom invokes the commitment function. Since the VM has been terminated, there is not any pending task or job in the VM. Meanwhile, all of the processes and other volatile objects, e.g., IPC objects, within a VM have already been erased from the OS and thus do not need to be committed. Only the permanent objects, e.g. files, directories, registry entries, need to be considered. The commitment procedure is completed following three steps.

First, we check all benign clusters and mark a benign cluster as malicious if it contains an ASEP but does not derive from a trusted source. Thus, we can prevent the potential evasion to Secom mentioned above, i.e., a malware program merely sets up an ASEP and waits for the commitment to execute the rest of the behaviors in the host environment. We introduce a Remote Administration Point (RAP) to represent a trusted source. A RAP is a special application dedicated to install software or manage the system from the remote. It applies the principle of diversity [28][29] and integrity protection techniques. Specifically, we install two different forms of programs with the same function, e.g., different kinds of web browsers. One is for daily use while the other is for RAP. Thus, we can set tight restrictions on the

Table 1. Committing results of the tested samples

Samples		Total	FNs	FPs
malware programs	Worm	20	0	-
	Trojan	19	0	-
	Backdoor	17	0	-
	Script Virus	2	0	-
	Rootkit	2	0	-
	Sum	60	0	-
Benign programs	Security utilities	10	-	1
	System utilities	9	-	1
	Games	5	-	0
	Multi-media	5	-	0
	Web Pages (ActiveX, Script)	21	-	0
	Sum	50	-	2

RAP program without affecting usability since one can use the other program. The RAP program is configured to have the highest security protection level, and only communicate with a few remote hosts through secure protocols. Moreover, Secom discards any changes made on the configurations and files of the RAP program so as to strictly preserve the integrity of the RAP program.

Second, we check every ASEP in malicious clusters to see whether the corresponding auto-start executables are placed in other benign clusters, and then mark such benign clusters as malicious. Thus, we can completely remove the malware programs that intentionally distribute ASEP hooks and the auto-start executables into different clusters.

Third, we discard all the objects in malicious clusters, and merge the objects in benign clusters into the host. Meanwhile, the objects not included in any cluster also need to be merged as they are not derived from the network or removable drives and thus benign.

Our approach will not lead to a significant internal inconsistency, because the discarded changes do not contain benign working results. As presented in Section 2.2, Secom can make a cluster to include either only malicious changes or only benign ones. Consequently, discarding the malicious clusters will not affect the internal inconsistency of benign working results. Our experiments in Section 4.1 further verified this point.

As mentioned in Section 2.2, a sophisticated malware might thwart Secom by dividing itself into separate clusters to reduce the chance of being captured. According to our VM commitment procedure presented in this section, in the first step, all clusters that contain the ASEPs of the malware program are marked as malicious. In the second step, the clusters containing the auto-start executables are marked as malicious. In the third step, the malicious clusters that contain ASEPs, auto-start executables and other objects of the malware program will be removed. Therefore, after being merged into the host, the malware will not have a chance of starting itself and continuously running.

In addition, as multiple processes within different VMs and host may concurrently access the same system object, an object modified within a VM may have already been independently modified outside the VM after the VM's private space is created. This will cause the inconsistency problem when committing the VM. As Secom focuses on providing secure commitment, it assumes the approaches provided in [7][8] would solve the problem.

3. Evaluations

To demonstrate the applicability of the OS-level VM commitment approach, we have successfully developed a prototype under the framework of Feather-Weight Virtual Machine [1], which partitions the name space of a single Windows OS to form a number of virtual machines. In this section, we present details on the experimental evaluation of our Secom prototype. The evaluation consists of three parts. First, we investigate the effectiveness of our secure virtual machine commitment approach using a group of real-world malware programs. Then, we compare Secom with commercial anti-malware tools. Finally, we perform tests to evaluate the performance overhead of our prototype.

3.1 Secure Commitment

To verify the capability of Secom to filter out malware objects when committing a VM, we collected 60 real-world malware samples from a publicly available website [14]. We intentionally did not select the samples that were used by our preliminary study of malware behaviors. We also prepared 50 benign samples mostly from two trustworthy websites, i.e. technet.microsoft.com and www.download.com. To further facilitate the experiments, we prepare a set of monitoring tools to help check experimental results, which include ApiMonitor to record system call and Win32 API, ProcessExplorer to analyze processes, Regmon to trace registry activities, and Filemon to monitor file operations. Meanwhile, we set up a local network consisting of two servers and two hosts as a testing environment. Server A mainly stores malware samples, and runs IIS web server, ftp server and EZ-IRC server. Server B mainly stores benign samples, and runs IIS web server as a reputable website. The host machines installed with Windows XP run the client programs that are often the attacking vectors for malware programs, including mIRC, MSN Messenger, MS Outlook, eMule, IE, ftp client, etc. On one host, Firefox is installed as a RAP program to download samples from the server B. To emulate the real-world usage scenarios, we login into the hosts and perform tasks including browsing the malicious website and ftp server in the local network and downloading samples, sending and receiving adverse instant messages and emails, accessing P2P shared folders or removable drives that contain samples, etc. Thus, the samples are introduced into a host through various channels. With this testing environment, the capability of Secom to securely commit a VM can be thoroughly evaluated.

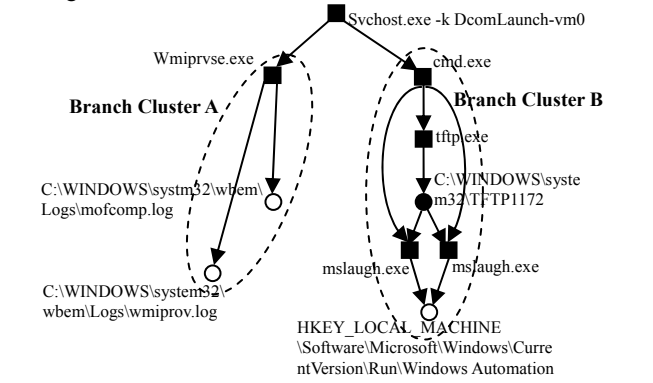
To recognize malicious clusters, we configured following raw behaviors into the detection engine: Type Ⅰ: Modifying registry for auto-startup, Creating or modifying Windows services, Installing or modifying Windows drivers. Type Ⅱ: Self-replication, Injecting into other processes, Creating processes abnormally. Type Ⅲ: Modifying registry to hide its presence, Lowering security settings, Disabling the host firewall, Killing anti-malware processes, Compromising anti-malware files or settings, Closing system restoring mechanism. Type Ⅳ: Stealing confidential information.

For every malware sample, we perform a three-step experiment. First we run the malware sample in a newly created VM without turning on the Secom and record what objects it creates or modifies. Then, we enable the Secom, run the same malware sample and other arbitrary benign applications together in a new VM, and eventually commit the VM. When performing the commitment, we make the VM committing module to print out the names of discarded objects. Lastly, we run the benign applications committed in the host environment in order to check whether the

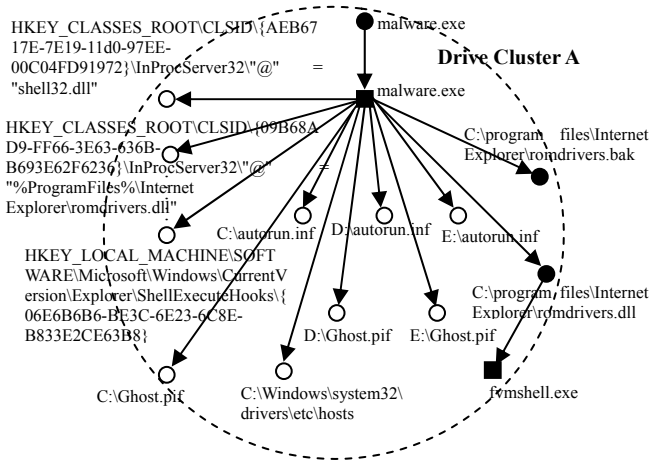
commitment process damages the internal consistency of the benign applications.

The testing results are reported in Table 1. For each type of samples, the table shows the total number of tested sample programs, FNs (false negatives) and FPs (false positives). We can see that Secom was able to correctly discard all malware samples on two servers, but falsely determine two benign samples to be malicious. The false positives are resulted because two benign programs exhibited behaviors of type Ⅰ and Ⅳ, i.e., modifying registry for auto-startup and reading sensitive files. Hence, the FN rate and FP rate of Secom are 0% and 4% respectively. Moreover, all of the benign applications which were mixed together with the malware samples can work smoothly after being committed into the host environment. In other words, the commitment process does not have impact on the internal consistency of the benign changes that coexist with malicious ones in the same VM.

To further illustrate the working procedure of our approach, we present two tested samples in details. "Worm.Win32.Lovesan.a" is a variant of the well-known network worm "Blast". Figure 4(a) illustrates its dependency graph resulted from the analysis of FVM logs printed out by FVM virtualization layer. The graph stems from a compromised svchost.exe process and is divided into three clusters according to the tracing and labeling methods. The Branch-Cluster A contains the



(a) Worm.Win32.Lovesan.a



(b) Virus.Win32.AutoRun.p.bak

■Networking process □General process ●Executable file ○Other

Figure 4. Dependency graph examples of four tested malware samples

Wmiprvse.exe process and its descendents, which is known as Windows Management Instrumentation and thus benign. However, the Branch-Cluster B was recognized as malicious because it showed two malicious behaviors: abnormally creating the process cmd.exe and modifying registry for auto-startup. The cluster includes all of the changes the malware sample made inside a VM. Consequently, Secom successfully discarded all negative changes made by the malware sample after committing the VM by not merging the Branch-Cluster B into the host.

"Virus.Win32.AutoRun.p.bak" spreads through removable storage devices and is depicted in Figure 4 (b). Drive-Cluster A is identified as malicious because it copied itself as romdrivers.bak and added the registry entry romdrivers.dll to restart itself. Secom successfully discarded all of its changes by not committing the Drive-Cluster A.

3.2 Comparison with Commercial Tools

To further evaluate Secom, we performed another experiment using two popular commercial anti-malware tools: Kaspersky and VIPRE. First, we use both commercial tools to scan the commitment results of Secom in the host environment after running a malware sample. For each sample, neither commercial tool could detect the malware in the host environment. Thus, we conclude that Secom has removed the malware to the satisfaction of the commercial tools.

Second, we test all of the samples in Table 1 using the signature-based module and behavior-based module of every anti-malware tool. Figure 5 shows the FP rates obtained from running five categories of benign samples. S and B represent signature-based module and behavior-based module respectively. From the figure, FP rate of Secom is slightly higher than that of signature-based modules while much lower than that of behavior-based modules. The reason is that, behavior-based technique often causes a higher FP rate than signature-based scheme (which can not detect unknown malware programs), but considering double behaviors and the originators of the processes exhibiting the behaviors will dramatically reduce FPs. Figure 6 shows the FN rates of detecting five categories of malicious samples. In the figure, both commercial tools only detect 50~75% of malicious changes regardless of what techniques they use. However, Secom identifies all of the changes imposed by different categories of malicious samples. In other words, Secom successfully clears all malicious changes when committing a VM.

3.3 Overhead

To evaluate an application's overall performance loss under Secom, we further measure the execution time of a set of programs. The test-bed used in this evaluation is a machine that contains a Pentium-4 2.8GHz CPU with 512MB memory and runs applications including WinZip32, xCopy, BCC32 and WebBench. The execution time is the average elapsed time from the start of a program to its termination. The results are listed in Figure 7. We can see that Secom imposes only 0.8%~8.3% more overhead on the system when running these programs compared to that of FVM. Comparing to BackTracer [10] that reports overhead of 9% for only tracing OS-level information flow, these results are encouraging given that Secom is a type of on-line intrusion detection and recovery system based on a single host.

4. Related work

There is no such a project that can securely commit VM on an OS-level virtual machine platform in the literature. Popular OS-level

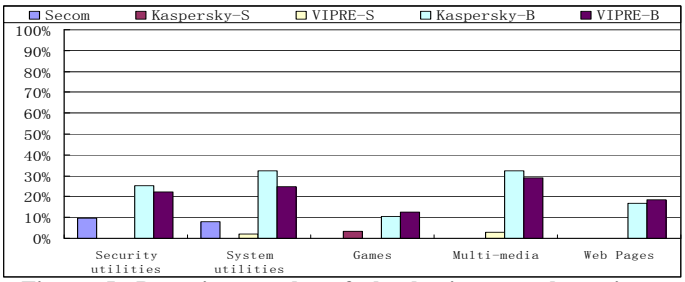


Figure 5. Detecting results of the benign samples using commercial tools and Secom

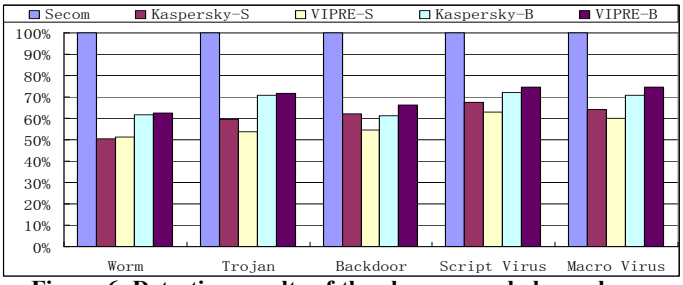


Figure 6. Detecting results of the changes made by malware samples using commercial tools and Secom

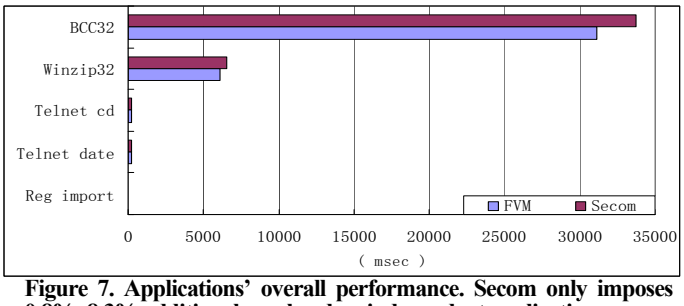


Figure 7. Applications' overall performance. Secom only imposes 0.8%~8.3% additional overhead on independent applications.

VM technologies, e.g. FreeBSD Jail [2], Linux-VServer [3], Solaris Zones [4], OpenVZ [5] and Virtuozzo [6], do not provide the functionality to securely commit VM. Feather-weight Virtual Machine (FVM) [1] only checks and abandons the resource updates under the security-related file directory and registry entry within a VM to eliminate side effects left by malicious programs before VM commitment. However, simply examining special resource updates confined inside a VM is not sufficient to detect all the suspicious behaviors or to recognize the exact scope of the attack due to malware execution. Sun et al. [7] present an approach for realizing a safe execution environment (SEE) that enables users to "try out" new software without the fear of damaging the system in any manner. It implemented a commitment approach to ensure semantic consistency of the committed results. Before commitment, user needs to make his own decision on whether the running results contained in a SEE are safe to commit. Secom, however, aims at automatically identifying unsafe results inside a VM before commitment. Hence, Secom differs from paper [7] and can be a complementary technology to [7] to help recognize unsafe results.

As an alternative technology, system call log analysis is able to detect compromised system resources and prevent them from being committed. Research efforts demonstrate the potential capability of log analysis to securely commit VM since log analysis can identify compromised files or derive malicious process behaviors [10][15][18]. However, the enforcement of log recording and analysis often significantly slows down the system which makes it very inefficient and possibly unusable. Our former work [30] also traces OS-level information flow, but it aims to block critical malware behaviors instead of committing safe changes in a VM back to the host. It also does not further correlate objects into clusters which will facilitate the fast commitment.

Secom is also different from other candidate technologies, such as host-based intrusion detection and anti-malware, though all of them are able to recognize malicious objects. Secom aims to recognize all side effects of a malware program imposed on a system while intrusion detection and anti-malware technologies typically are interested in determining whether a single file or process is adverse. A recent approach proposed in [19] can remove all effects of a malware program, but requires a training stage to generate a remediation program for cleaning the impacts a specific malware. Therefore, Secom meets the requirement of secure commitment better than these candidate technologies.

5. Summary

In this paper, we propose Secom, a scheme towards 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. We thus believe that Secom is the first secure commitment scheme. The critical challenge behind securely committing VM is to identify compromised objects thoroughly and lightly. To address the challenge, we propose Secom that consists of three steps. First, it correlates suspicious OS objects within a VM together by tracking OS-level information flows and grouping them into clusters by intelligently attaching different labels to objects. Then, it recognizes a malicious cluster by a behavior-based malware detection engine. Last, it commits VM while discarding malicious clusters. Secom has three novel features. First, Secom can lightly commit VM using OS-level information flows and malware behaviors. Second, Secom detects and discards malicious changes in a cluster fashion to clean up malware impacts quickly and thoroughly. Finally, to reduce the false positive rate, Secom considers two malware behaviors which are of different types and the originator of the processes which exhibit the behaviors when identifying a malicious cluster. We implemented Secom under the framework of FVM and conducted experiments concerning the performance of commitment and overhead. The experiment results demonstrate that Secom can effectively clean up malware impacts when performing commitment and only causes 0.8% to 8.3% additional performance overhead on system. Moreover, compared with commercial anti-malware tools, it can erase malware more thoroughly and produce a lower false positive rate. Hence, it fits the task of VM commitment better.

6. ACKNOLEDGMENT

We would like to thank Professor Prashant Shenoy and all the anonymous reviewers for their insightful comments and feedback. This work is supported by US National Science Foundation under grants CNS-0751121, CNS-0751121 and CNS-0628093, Natural Science Foundation of China under grants No. 60703103, No. 60833005, No. 60873213 and No. 61070192.

7. References

1. Y. Yu, F. Guo, S. Nanda, L. Lam, T. Chiueh. A Feather-weight Virtual Machine for Windows Applications. In Proceedings of the 2nd ACM/USENIX Conference on Virtual Execution Environments. Pages 24–34, Ottawa, 2006.
2. P.-H. Kamp and R. N. M. Watson. Jails: Confining the omnipotent root. In Proceedings of the 2nd International SANE Conference, 2000.
3. S. Soltesz , H. Pötzl , M. E. Fiuczynski , A. Bavier , L. Peterson. Container-based operating system virtualization: a scalable, high-performance alternative to hypervisors. In proceedings of the 2nd ACM SIGOPS/EuroSys European Conference on Computer Systems 2007, Lisbon.
4. D. Price and A. Tucker. Solaris Zones: Operating system support for consolidating commercial workloads. In Proceedings of the 18th Large Installation System Administration Conference (LISA 2004), pages 241–254.
5. OpenVZ. Unique features of OpenVZ. http://openvz.org/documentation/tech/features.
6. SWsoft. Virtuozzo server virtualization. http://www.swsoft.com/en/products/virtuozzo.
7. W. Sun, Z. Liang, V. Venkatakrishnan, and R. Sekar. One-way isolation: An effective approach for realizing safe execution environments. In Proceedings of 12th Annual Network and Distributed System Security Symposium, 2005, pages 265–278.
8. Y. Yu. OS-level Virtualization and Its Applications. Ph.D. Dissertation, December 2007.
9. Symantec, Inc, http://www.symantec.com/business/ security_response/threatexplorer/threats.jsp.
10. S.T. King and P.M. Chen. Backtracking Intrusions. Proceedings of ACM Symposium on Operating Systems Principles, pages 223–236, October 2003.
11. Microsoft Security Bulletins, http://www.microsoft.com/technet/security/current.aspx.
12. M. Howard. Fending Off Future Attacks by Reducing Attack Surface. http://msdn.microsoft.com/en-us/library/ms972812.aspx, 2003.
13. Y. Yu, H. K. Govindarajan, L. Lam, T. Chiueh. Applications of Feather-Weight Virtual Machine. Proceedings of the 2008 ACM SIGPLAN/SIGOPS International Conference on Virtual Execution Environments, Seattle WA, March 2008.
14. Offensive Computing, http://www.offensivecomputing.net/.
15. F. Hsu, H. Chen, T. Ristenpart, J. Li, and Z. Su. Back to the Future: A Framework for Automatic Malware Removal. In Proc. Annual Computer Security Applications Conference, 2006.
16. B. Cully, G. Lefebvre, D. Meyer, M. Feeley, N. Hutchinson, and A. Warfield, “Remus: High Availability via Asynchronous Virtual Machine Replication”. In Proceedings of the 5th USENIX Symposium on Networked Systems Design and Implementation, 2008.
17. T. Holz, C. Willems, K. Rieck, P. Duessel, and P. Laskov. Learning and Classification of Malware Behavior. In Fifth Conference on Detection of Intrusions and Malware & Vulnerability Assessment, pages 108–125, June 2008.
18. N. Zhu, and T. Chiueh. Design, implementation, and evaluation of repairable file service. In Proceedings of the 2003 International Conference on Dependable Systems and Networks, pages 217–226, June 2003.
19. R. Paleari, L. Martignoni, E. Passerini, D. Davidson, M. Fredrickson, J. Giffin, and S. Jha, "Automatic generation of remediation procedures for malware", USENIX Security Symposium, Washington DC, August 2010.
20. N. Li, Z. Mao, H. Chen, "Usable Mandatory Integrity Protection for Operating Systems," IEEE S&P, pages 164–178, 2007.
21. E. Kirda, C. Kruegel, G. Banks, G. Vigna, and R. Kemmerer. Behavior-based spyware detection. In Usenix Security Symposium, 2006.
22. H. Yin , D. Song , M. Egele , C. Kruegel , E. Kirda. Panorama: capturing system-wide information flow for malware detection and analysis. CCS2007, pages 116–127, New York, NY, USA, 2007.
23. O. Sukwong, H. Kim, J. Hoe, "An Empirical Study of Commercial Antivirus Software Effectiveness", Computer, Jun. 2010. IEEE Computer Society.
24. Y.-M. Wang, R. Roussev, C. Verbowski, A. Johnson, M.-W. Wu, Y. Huang, and S.-Y. Kuo. Gatekeeper: Monitoring auto-start extensibility points (aseps) for spyware management. In Proceedings of 18th Large Installation System Administration Conference, November 2004.
25. C. Kolbitsch, P. M. Comparetti, C. Kruegel, E. Kirda, X. Zhou, and X. Wang. Effective and efficient malware detection at the end host. In USENIX Security Symposium, Montr´eal, Canada, August 2009.
26. C. Verbowski, E. Kiciman, A. Kumar, B. Daniels, S. Lu, J. Lee, Y.-M. Wang, and R. Roussev. Flight data recorder: monitoring persistent-state interactions to improve systems management. In Proceedings of the 7th symposium on Operating systems design and implementation, pages 117–130, 2006.
27. M. Egele , P. Wurzinger , C. Kruegel , E. Kirda, Defending Browsers against Drive-by Downloads: Mitigating Heap-Spraying Code Injection Attacks, Proceedings of the 6th International Conference on Detection of Intrusions and Malware, and Vulnerability Assessment, July 2009, Como, Italy.
28. B. Littlewood and L. Strigini. Redundancy and diversity in security. In Proceedings of the 9th European Symposium on Research in Computer Security (2004), pages 423–438.
29. E. Totel, F. Majorczyk, and L. Me. COTS diversity baded intrusion detection and application to web servers. In Eighth International Symposium on Recent Advances in Intrusion Detection, Seattle, Washington, September 2005.
30. Z. Shan, X. Wang, T. Chiueh. Tracer: Enforcing Mandatory Access Control in Commodity OS with the Support of Light-Weight Intrusion Detection and Tracing. In Proceedings of the 6th ACM Symposium on Information, Computer and Communications Security (ASIACCS), pages 135~144, March 2011.
31. J. Zhu, Z. Jiang, Z. Xiao, and X. Li. Optimizing the Performance of Virtual Machine Synchronization for Fault Tolerance, IEEE Transactions on Computers, 2011.