Enforcing Mandatory Access Control in Commodity OS to Disable Malware

Zhiyong Shan, Xin Wang, Tzi-cker Chiueh

Abstract—Enforcing a practical Mandatory Access Control (MAC) in a commercial operating system to tackle malware problem is a grand challenge but also a promising approach. The firmest barriers to apply MAC to defeat malware programs are the incompatible and unusable problems in existing MAC systems. To address these issues, we manually analyze 2,600 malware samples one by one and two types of MAC enforced operating systems, and then design a novel MAC enforcement approach, named Tracer, which incorporates intrusion detection and tracing in a commercial operating system. The approach conceptually consists of three actions: detecting, tracing and restricting suspected intruders. One novelty is that it leverages light-weight intrusion detection and tracing techniques to automate security label configuration that is widely acknowledged as a tough issue when applying a MAC system in practice. The other is that, rather than restricting information flow as a traditional MAC does, it traces intruders and restricts only their critical malware behaviors, where intruders represent processes and executables that are potential agents of a remote attacker. Our prototyping and experiments on Windows show that Tracer can effectively defeat all malware samples tested via blocking malware behaviors while not causing a significant compatibility problem.

Index Terms—Access controls, operating system, invasive software, OS-level information flow.

1 INTRODUCTION

MALICIOUS software (i.e., Malware) has resulted in one of the most severe computer security problems today. A network of hosts which are compromised by malware and controlled by attackers can cause a lot of damages to information systems. As a useful malware defense technology, Mandatory Access Control (MAC) works without relying on malware signatures and blocks malware behaviors before they cause security damage. Even if an intruder manages to breach other layers of defense, MAC is able to act as the last shelter to prevent the entire host from being compromised. However, as widely accepted [2][3][5], existing MAC mechanisms built in commercial operating systems (OS) often suffer from two problems which make general users reluctant to assume them. One problem is that a built-in MAC is incompatible with a lot of application software and thus interferes with their running [2][3][5], and the other problem is low usability, which makes it difficult to configure MAC properly [2]. Thus, enforcing a practical MAC on commercial OS to defend against malware is a promising but challenging task.

In order to devise a new MAC enforcement method to defeat malware, we have performed two preliminary studies. First, we analyzed the technical details of 2,600 samples so as to get a deep and overall view on malware programs. We extracted 30 critical malware behaviors and found three common malware characteristics that can guide anti-malware system design. Second, we investigated the root cause of incompatibility and low usability of existing MAC models through experiments on two types of MAC enforced operating systems. Our observations are as follows. The incompatibility problem is introduced because the security labels of existing MACs are unable to distinguish between malicious and benign entities, which causes a huge number of false positives (i.e. treating benign operations as malicious) thus preventing many benign software from performing legal operations; the low-usability problem is introduced, because existing MACs are unable to automatically label the huge number of entities in OS and thus require tough configuration work at end users.

With these investigation results, we propose a novel MAC enforcement approach, Tracer, which consists of three actions: detection, tracing and restriction. Each process or executable has two states, suspicious or benign. An executable in this paper represents an executable file with a specific extension, such as .EXE, .COM, .DLL, .SYS,.VBS, .JS, .BAT, or a special type of data file that can contain executable codes, say a semi-executable, such as .ZIP, .RAR, .DOC, .PPT, .XLS, and .DOT. The actions of detection and tracing change the state of a process or executable to suspicious if it is suspected to be malicious, and the entity marked as suspicious is called a suspicious intruder. The action of restriction forbids a suspicious intruder to perform malware behaviors in order to maintain confidentiality, integrity and availability of the system, as well as to stop malware propagation. To be precise, once detecting a suspicious process or executable, Tracer labels it to be suspicious and traces its descendent and interacted processes, as well as the executables it generates. Tracer does not restrict any operations of benign processes. Meanwhile, it permits suspicious processes to run as long as possible but only forbids their malware behaviors.

The novelty of Tracer is that, it incorporates light-weight intrusion detection and tracing techniques for configuring security labels, i.e., labeling suspicious OS entities, which is often done manually. Moreover, rather than restricting information flow as a traditional MAC does, it traces suspected intruders and restricts the malware behaviors of suspected intruders, i.e., processes and executables that are potential agents of remote attackers. These novelties lead to two advantages. First, Tracer is able to better identify

- *Zhiyong Shan is with the Key Laboratory of DEKE and Computer Science Department, Renmin University of China. E-mail: shanzhiyong@ruc.edu.cn.*
- *Xin Wang is with the Electrical and Computer Engineering Department, Stony Brook University, USA. E-mail: xwang@ece.sunysb.edu.*
- *Tzi-cker Chiueh is with the Computer Science Department, Stony Brook University, USA. E-mail: chiueh@ cs.sunysb.edu.*

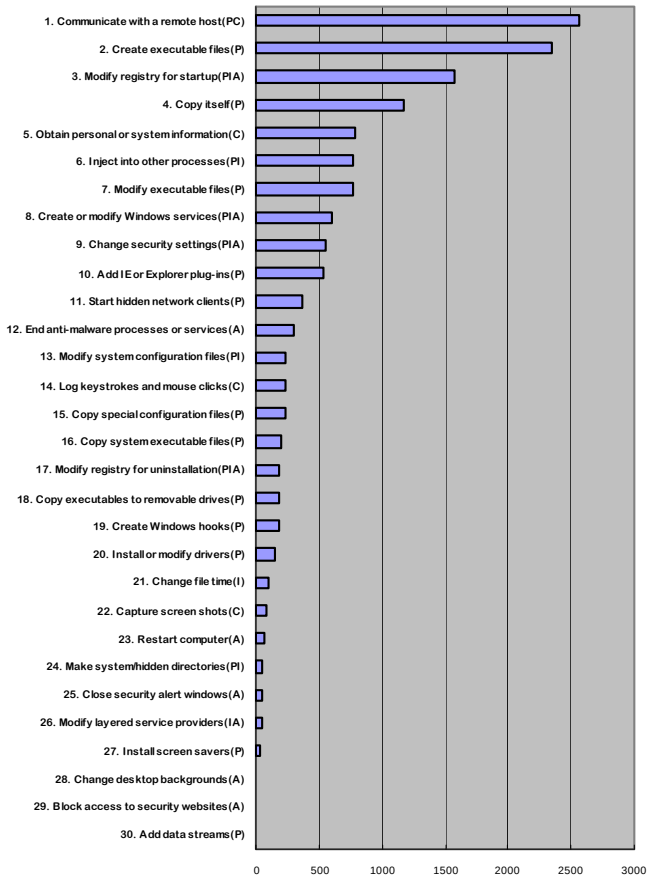

Fig. 1. The top 30 Critical malware behaviors

potentially malicious OS entities and regulate their behaviors, which in turn significantly reduces the false positive (FP) rate which is the root cause of incompatibility in existing MAC-enforced systems. Second, Tracer is able to label OS entities automatically to tackle the low usability problem which is the other major issue of existing MAC systems [2].

We have implemented Tracer on Windows and have been using evolving prototypes of the Tracer system in our lab for a few months. Our experiments on the function of Tracer with a set of real-world malware samples demonstrate that it can effectively block malware behaviors while offering good compatibility to applications and good usability to normal users.

Moreover, we have added another experiment to compare Tracer with existing practical online malware defense technology. The result shows that Tracer causes much fewer FPs than commercial anti-malware tools and MIC (Mandatory Integrity Control) which is a MAC mechanism on Windows Vista [4][16].

The contributions of this paper are as follows:

1. We introduce Tracer, a novel MAC enforcement approach which integrates intrusion detection and tracing techniques to disable malware on a commercial OS in a compatible and usable manner.
2. We have implemented Tracer on Windows OS to disable malware timely without need of malware signatures. Developing a prototype on Windows is important, because most of the over 236,000 known malware items are designed for the attacks in the Windows environment, only about 700 malware items target for the attack of various Unix/Linux distributions [12].
3. Based on the analysis of 2,600 malware samples, we extract 30 critical malware behaviors and summarize three useful malware characteristics, which will benefit future anti-malware researches.
4. We investigate the root reasons of incompatibility and low usability problems of existing MACs. Although not all the observations are brand new, we believe that understanding these reasons more comprehensively and illustrating them through the design of an actual system are useful for other MAC researchers.

The rest of the paper is organized as follows. Section 2 introduces in details our investigation on various behaviors of malware programs, and our analysis on existing problems in MAC. Section 3 describes Tracer approach. Section 4 provides our prototype and tests of Tracer on Windows. Section 5 discusses the approach from the perspectives of security, compatibility and usability. Lastly, we present the related research in Section 6 and conclude the work in Section 7.

2 PRELIMINARY STUDIES

2.1 Malware Investigation

Malware contribute to most Internet security problems. Anti-malware companies typically receive thousands of new malware samples every day. An analyst generally attempts to understand the actions that each sample can perform, determines the type and severity of the threat that the sample constitutes, and then forms detection signatures and creates removal procedures. Symantec Threat Explorer [6] is such a publicly available database which stores the analysis results of thousands of malware samples from various sources and is thus valuable to malware researchers. To have a thorough understanding of the philosophies behind malware design, we have spent considerable amount of time analyzing the behaviors of malware programs. Specifically, since 2008, we have read, recorded and analyzed the technical details of 2,600 malware samples of a wide range of formats and varieties, such as viruses, worms, backdoors, rootkits, and Trojan horses. As taking many samples from the same malware family might make the analysis results biased, we have intentionally not chosen multiple samples of a polymorphic malware or similar malware.

Figure 1 depicts the top 30 critical malware behaviors extracted from the samples and ranked in the descending order of their appearance times. For the behavior repeatedly appearing in a single malware, we only count it once. As the analysis is made on a great number of malware samples, we expect the behaviors captured to reflect the popular attacking techniques taken by the community of malware writers. Our performance studies in Section 4.2 have demonstrated that these behaviors are helpful to defend against unknown malwares. Details of the behaviors are provided in the supplemental materials.

Moreover, from the details of 2,600 malware samples, we discovered three common characteristics of malware

that can guide our subsequent anti-malware design:

(1) **Entrance-Characteristics**. All malware samples break into hosts through two entrances, network and removable drive. Most breaking-ins are via network through frequently used protocols such as HTTP and POP3.

(2) **Damage-Characteristics**. Malware behaviors can impose multiple forms of damages, i.e., resulting in problems in confidentiality, integrity and availability. Besides, we consider malware propagation as another type of damage since it can indirectly cause the former three forms of damages and eventually lead the entire host to be taken over. For example, the behavior "Copy itself" does not directly hurt security but is an essential step towards propagating itself and then executing malicious behaviors on a host. Therefore, we evaluate the damages of each behavior and record them in Figure 1, using C, I, A, and P to represent the damages related to confidentiality, integrity, availability and propagation respectively.

(3) **Attack-Characteristics**. Malware samples from the network have two attack patterns. One is that, most malware samples exploit bugs in network-facing daemon programs or client programs to compromise them, then immediately spawn a shell or back-door process. Next, an attacker typically tries to download and install attacking tools and rootkits, as well as performs some other adversary behaviors. The other attack pattern is that, malware samples increasingly use social engineering methods to lure users into downloading and launching them. After started, a malware sample usually copies itself and makes itself a resident in a host.

2.2 Problems in MAC

Incompatibility is a well-known problem when enforcing a MAC model in a commercial operating system [2][3][5]. To investigate its root reason, in a secure network environment, we set up two machines to run MAC enforced operating systems including SELinux [14] with MLS policy enabled and RSBAC [15] with MAC module enabled. After a few days, we observed that these MAC systems produced a huge number of log records about denied accesses, which indicated that some applications failed and some acted abnormally. As the operation environment is secure without intrusion and malware, these denied accesses are thus "false positive". In other words, MAC systems consider benign accesses malicious and refuse them. Many FPs together could make the whole system finally unusable. Although part of the FPs can be removed by experts through fine-granular policy configuration, many of them are not removable, and thus the impacted applications need to be modified before running on the MAC enabled systems.

These unremovable FPs are resulted because most MAC models aim to forbid illegal information flow rather than forbid intrusive behaviors directly. An example of such FPs is the self-revocation problem [3] in Low-Water-Mark model, which forbids a process to write a file created by itself if it has read a file with a lower integrity level before the writing. From the perspective of stopping illegal information flow, forbidding the write operation is reasonable. However, from the perspective of stopping intrusion, the write operation should not be denied if the process is actually not serving for an attacker. Another example of such FPs on a BLP-enforced Unix/Linux stems from the access control of the directory "/tmp" shared by the entire system [17]. To prevent illegal information flow, a process with a lower sensitive level can not read from /tmp or a process with a higher sensitive level can not write to /tmp. However, from the view of intrusion prevention, these processes do not necessarily represent intruders so that their "read" or "write" accesses to the /tmp should not be simply denied. Although it is possible to resolve this problem by adding "hiding sub directories" under /tmp, it is still difficult to eliminate the FPs resulting from many other shared entities on an OS, such as shared files, devices, pipes and memories.

Meanwhile, the security labels of MAC models also do not suit for fighting against malware, as they are designed to represent information integrity level or confidentiality level but not to distinguish between malicious and benign entities. In fact, a lower integrity level alone can not indicate that a process is malicious, as "malicious" also has other meanings, e.g., lower confidentiality and the risk of damaging system availability. Similarly, a lower confidentiality level alone cannot indicate that a process is malicious. Moreover, MAC labels are defined before an intrusion happens and can not be changed dynamically to reflect intrusion propagation in an OS. Although some of the MAC models are able to adjust label states, e.g. LOMAC [3] and DTE [9], they are still not flexible enough to track the intrusion propagation at the whole system level. Consequently, MAC labels can not differentiate between malicious and benign entities. Relying on these labels, a MAC system often fails to make correct decisions on intrusion blocking which eventually results in many FPs.

Low-usability is another problem in a MAC-enabled system, as it often requires complicated configurations and unconventional ways of usage. In a modern OS, there are a wide range of entities including processes, files, directories, devices, pipes, signals, shared memories and sockets, etc. If just considering the files, there are more than 100,000 files on a typical Windows XP or Linux desktop. Moreover, MAC systems have complex policy interfaces which are difficult to configure. For instance, SELinux has 29 different classes of objects, hundreds of possible operations, and thousands of policy rules for a typical system. Hence, it is cumbersome for a common user to correctly configure labels for all entities without leaving security vulnerabilities. In addition, after enforcing a MAC, users must break their usage convention and learn how to use the MAC. Consequently, the ideal way for MAC to provide good usability is to automatically initialize and change entity labels without changing users' usage convention or requiring extra knowledge.

3 TRACER APPROACH

In this section, we present our Tracer approach that aims to disable malware in a commodity OS by disallowing malware behaviors. The adversaries of Tracer are malware programs that break into a host through the network or removable drives. As Windows is the most popularly used

OS and attractive to hackers, the description of Tracer is based on our design for Windows. We believe the approach can also be applied to other operating systems (e.g. Linux) with some changes. Investigating the suitability of Tracer for non-Windows operating systems is beyond the scope of this paper.

3.1 Overview

The design of an access control mechanism needs to answer two questions. The first is how to define the security label. Based on the analysis in Section 2.2, we introduce a new form of security label called *suspicious label* for our Tracer approach. It has two values: suspicious and benign. A suspicious label indicates that the associated process is potentially serving for an intrusion purpose and thus possible to initiate some malicious behaviors. Meanwhile, Tracer only assigns a suspicious label to a process or an executable, because a process is possibly the agent of an intruder and an executable determines the execution flow of a process which represents an intruder. All other entities in OS, e.g. non-executables, inter-process communication objects, registry entries, etc, do not need suspicious labels. When a process requests to access these entities, Tracer mainly utilizes their DAC information to make access control decisions, thus a huge amount of configuration work can be reduced while keeping traditional usage conventions unchanged. The second design question is how to configure security labels. As discussed in Section 2.2, in order to achieve good usability, a MAC approach must have the capability of automatically deploying security labels. Accordingly, we introduce two types of actions named "detection" and "tracing" to automate the security label deployment progress. The two actions employ intrusion detection and tracing techniques respectively to recognize and mark suspicious processes and executables.

Figure 2 gives an overview of Tracer which consists of three types of actions, detection, tracing and restriction. Each process or executable has two states, suspicious and benign. The actions of detection and tracing change the state of a process or executable to suspicious if it is identified as a potential intruder. The restriction action forbids a suspected intruder to perform malware behaviors in order to protect CIAP. That is to protect confidentiality, integrity and availability, as well as to stop malware propagation. The three actions work as follows. Once detecting a suspected process or executable, Tracer labels it as suspicious and traces its descendent and interacted processes, as well as its generated executables. Tracer does not restrict benign processes at all, and permits suspicious processes to run as long as possible but stops their malware behaviors that would cause security damages. In addition, Tracer also provides a special system call to allow a user to change the state of a suspicious process or executable back to benign if the user trusts it.

A **malware behavior** is defined as a triple <*operation, object, parameter*>, where the operation is a generalization of one or several system calls that have similar functions. The object and parameter represent the target and parameter of the operation respectively. Specific malware behaviors monitored in the current version of Tracer are listed in Table 1, which contains the 30 critical malware behaviors

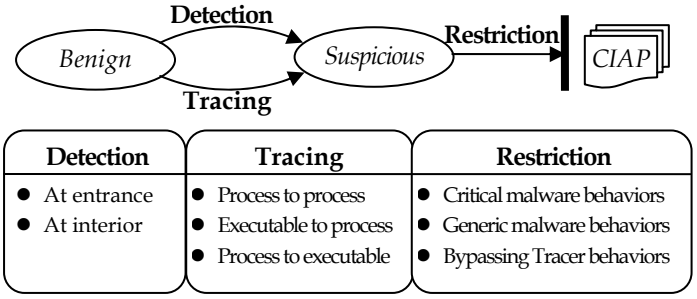

Fig. 2. Tracer approach

shown in Figure 1. Moreover, Tracer allows dynamic addition of new behaviors.

The access control decision of Tracer is made in accordance with normal MACs. Tracer uses the subject label and behavior to make a decision while normal MACs use the subject label, operation, object label and parameter. As a behavior consists of operation, object and parameter, Tracer actually uses the same four factors of normal MAC decision. Moreover, Tracer's decision procedure generates three possible access control results: "allow", "deny" and "change label", which resemble those of normal MACs. The detailed decision logic of Tracer is shown in Table 1. The detection and tracing actions lead to the decision result "change label", while restriction action leads to "deny". All access requests not denied are allowed.

As an online approach, Tracer can produce the FP rate lower than that of behavior-blocking mechanisms in commercial anti-virus software. This is achieved by two means. First, as a MAC system, Tracer blocks a behavior based simultaneously on the behavior and security label (i.e., the suspicious label of the current process), rather than merely the behavior as done by a behavior-blocking system. Second, Tracer does not simply refuse all critical malware behaviors in Figure 1. The behaviors that are indispensable to benign programs while do not directly hurt security are not blocked but traced, which are shown in Table 1.

In the rest of this section, we describe Tracer approach in details, including detecting, tracing and restricting intruders.

3.2 Detecting Intruders

The detecting action is responsible for identifying all potential intruders. We do not intend to design a complex intrusion detection algorithm to achieve a low FP rate at the cost of heavy overhead. Instead, we design a light-weight intrusion detection algorithm that can identify all potential intruders but may have a relatively higher FP rate at the initial step. However, even if the detecting action wrongly denotes a benign process as suspicious, the subsequent actions of Tracer, i.e., tracing and restricting actions, will still allow it to run rather than stop it immediately, but only prevent it from executing featured malware behaviors. In other words, Tracer is built to have a good tolerance to the FPs caused by the detecting action.

As depicted in Figure 2, the detection works at two levels: entrance and interior. The detection at entrance attempts to check all possible venues through which a malware program may break into the system. Network communications is the main type of entrances and most malware programs exploit several common protocols to

compromise hosts according to the Entrance-Characteristics presented in Section 2.1. We define these protocols as **dangerous protocols**. Dangerous protocols are permitted by firewalls, thus malware programs often use these protocols to penetrate firewalls by disguising themselves as popular software that generate benign network traffic. Dangerous protocols mainly include HTTP, POP3, IRC, SMTP, FTP and ICMP. The types of dangerous protocols should be adjusted based on the actual firewall configuration.

Hence, we denote a process as suspicious if it receives network traffic through dangerous protocols. A frequently used application (e.g., web browser) thus might be denoted as suspicious but its normal running will not be affected, because Tracer doesn't restrict the suspected processes instantly and permits them to perform as long as possible except stopping their critical malware behaviors.

The non-dangerous protocols are difficult to be exploited by malware programs, because they are not permitted by firewalls since benign software rarely uses them. Here we assume a "deny" default action for firewalls [31], thus any traffic not specifically allowed by firewall rules are denied. Nevertheless, in order to completely monitor all the network traffic, we denote a process as suspicious if it receives network traffic through a non-dangerous protocol and then exhibits any of the malware behaviors. Instead of only checking non-dangerous network protocols, further checking malware behaviors can reduce the extra high FP rate. The Attack-Characteristics summarized in Section 2.1 supports this point. That is, a process exploited by a malware program from the network necessarily executes at least one critical malware behavior, e.g., launching a shell process or downloading an executable, to propagate the malware program within the system. Although a carefully crafted malware program that subverts a process through a non-dangerous protocol can perform some behaviors before performing a malware behavior, it is difficult for the process to make significant damages on the system. The reason is that the malware behaviors monitored by Tracer include all of the behaviors that can cause significant damages, let alone that malware programs are difficult to attack a host through non-dangerous protocols which are usually blocked by firewalls.

The other type of entrances through which malware programs get into the system is removable drives according to the Entrance-Characteristics, hence we denote a process as suspicious when it opens or loads an executable from a removable drive.

With these detection approaches enforced, however, two types of system maintenance tasks, i.e., updating software through the network and installing software from a removable drive, can not be performed because the processes that perform these tasks are treated as suspicious. As presented in the literature work [2] [3], a MAC policy should have ways to specify exceptions since no simple policy model can capture all accesses that need to be allowed and at the same time forbid all illegal accesses. Hence, we provide two means to facilitate these system maintenance tasks. One is trusted communications through which processes can update software remotely without being marked as suspicious. A communication is considered to be trustful if the three factors associated with it, i.e., "image file of the local process", "communication protocol" and "remote host" are all trusted. Meanwhile, a trusted communication is time limited, i.e., effective only within a predefined time period. Although a trusted protocol, e.g. SSL, is not absolutely secure, a further check of the process' image file, the remote host and the time stamp simultaneously will greatly reduce the attack surface. The other means is a new system call to facilitate a user to manually remove suspicious labels on specific processes or files if the user trusts them. For example, when installing benign software from a CD disk, a user can remove the suspicious labels from the processes which read the executables on the CD disk then Tracer will not affect the installation progress any more. Note that, only a process without a suspicious label has the privilege to use the system call so as to prevent a malware program from bypassing Tracer.

However, it is difficult for a normal user to identify a trusted communication or detect a particular process that is reading executables from a CD disk. To address this issue, we introduce a special bit in kernel, namely attention bit. A user can turn on the bit and start such communication or process within a short time period. Tracer then intercepts it and pops up a window, which displays the information of the communication or process, to require the user's consent. Once conformed by the user, the communication is set to be trusted or the suspicious label is removed from the process. And then, the attention bit turns off automatically. This mechanism avoids the manual work of recognizing the trusted communication and the process without annoying the user by frequently popping up windows.

Although bypassing the detection at entrances is difficult, in case that a sophisticated malware program unexpectedly breaks into the system, we prepare a type of detection at the interior of the system to ambush it. This type of detection monitors the exclusive malware behaviors that a benign program will not exhibit. The current version of Tracer conservatively uses five such behaviors to detect malware programs inside a system, including "Copying itself", "Injecting into other processes", "Modifying executable files", "Starting hidden network clients" and "Ending anti-malware processes or services". More behaviors can be monitored for malware detection in the interior at the cost of additional FPs. Actually, these behaviors together provide a strong detection capability as they are indispensable to most malware programs, e.g., "Copy itself". In addition, this type of detection will not bring extra performance overhead since the restricting action of Tracer also needs to monitor such behaviors, which will be presented in Section 3.3.

In short, the detection action identifies a process as suspicious if it meets one of the detection rules:

- *Receiving network traffic through dangerous protocols;*
- *Receiving network traffic through non-dangerous protocols then exhibiting any of the malware behaviors;*
- *Reading or loading an executable from a removable drive;*
- *Exhibiting any of the five exclusive malware behaviors.*

Columns "Detect" in Table 1 show the details of the detection action.

TABLE 1. DECISION LOGIC OF TRACER.

The Benign Process and Suspicious Process columns represent that the processes requesting the behaviors below are benign or suspicious respectively. Cp and Ce indicate changing the label of related process or executable to suspicious respectively. D indicates denying the behavior

Malware Behaviors		Benign Process			Suspicious Process		
		Detect	Trace	Restrict	Detect	Trace	Restrict
1. Communicate with a remote host	Normal	C_P					
	Trusted						
2. Create executable files						C_e	
3. Modify registry for startup							D
4. Copy itself		C_P					D
5. Obtain personal or system information							D
6. Inject into other processes		C_P					D
7. Modify executable files		C_P					D
8. Create or modify Windows services							D
9. Change security settings							D
10. Add IE or Explorer plug-ins							D
11. Start hidden network clients		C_P					D
12. End anti-malware processes or services		C_P					D
13. Modify system configuration files							D
14. Log keystrokes and mouse clicks							D
15. Copy special configuration files							D
16. Copy system executable files						C_e	
17. Modify registry for uninstallation							D
18. Copy executables to removable drives							D
19. Create Windows hooks							D
20. Install or modify drivers							D
21. Change file time							D
22. Capture screen shots							D
23. Restart computer							D
24. Make system/hidden directories							D
25. Close security alert windows							D
26. Modify layered service providers							D
27. Install screen savers							D
28. Change desktop backgrounds							D
30. Add data streams							D
31. Damage system integrity							D
32. Steal confidential information							D
33. Read executables on removable drives		C_P					
34. Change file attributes							D
35. Change registry entry attributes							D
36. Create processes						C_P	
37. Load suspicious executables						C_P	
38. Read suspicious executables						C_P	
39. Communicate via dangerous IPCs						C_P	
40. Execute non-executable files							D
41. Execute Tracer special system calls							D

3.3 Tracing Intruders

To track intruders within an operating system, one can use OS-level information flow as done in [18] [26]. However, a major challenge for leveraging OS level information flow to trace suspicious entities is that, file and process tagging usually leads the entire system to be floated with "suspicious" labels and thus incurs too many FPs. To address this issue, we propose the following two methods to limit the number of tagged files and processes in a single OS while preventing malware programs from evading the tracing as much as possible.

For tagging files, unlike the approaches in [18] [26] and the schemes of many malware detection and MAC systems [1][2][5][21] that trace information flow on OS level, Tracer only focuses on the tagging of executables while ignoring non-executables and directories. This is because an executable represents the possible execution flow of the process loading it, thus it should be deemed as an inactive intruder while a process is considered as an active intruder. On the other hand, since there are a huge number of non-executable files and directories within a single OS, not tracing them can prevent the entire file system from being floated with the suspicious labels that mostly are due to FP.

For tagging processes, we observed that the excessive number of tags mainly come from tracing IPC (Inter-Process Communication), i.e. marking a process as suspicious if it receives IPC data from a suspicious process, just as the approaches assumed in [2] [18]. To address this issue, Tracer only tags a process receiving data from dangerous IPCs that can be exploited by a malware program to take control of the process to perform arbitrary malicious behaviors. Note that, dangerous IPCs do not include the other types of vulnerable IPCs that can be used to launch denial-of-service attack, or disclose sensitive information, or escalate the privileges of the processes which send IPC data. Moreover, a dangerous IPC only involves the local IPCs instead of the IPCs over the network, since the detection at entrance can mark a process that receives IPC data from the network as suspicious. In order to identify the dangerous IPCs, we investigated Microsoft Security Bulletins [19], a database storing information about security vulnerabilities on Windows family OS and other Microsoft software. As malware programs usually exploit these vulnerabilities to compromise Windows hosts, Microsoft Security Bulletins become primary sources for analyzing attack vectors of Windows OS as done in [11]. Concretely, we analyzed all vulnerabilities recorded in security bulletins related to named-pipes, local procedure calls, shared memories, mailslots and Windows messages from 1998 to 2009, as these IPCs send free-formed data that can be crafted to exploit bugs in the receiving process. However, among all of the security bulletins, we only found one dangerous IPC, i.e. MS03-025 [19]. The result reveals that in reality it is quite difficult to propagate malware through local IPCs within a Windows OS since people could only find one dangerous IPC over the period of eleven years. Consequently, Tracer employs a Dangerous-IPC-List to record and trace each type of dangerous IPC since there should be a very limited number of dangerous IPCs in a Windows OS.

Therefore, we have the following tracing rules to mark entities as suspicious:

- *A process spawned by a suspicious process;*
- *An executable or semi-executable created or modified by a suspicious process;*
- *A process loading an executable with a suspicious label;*
- *A process receiving data from a suspicious process through a dangerous IPC;*
- *A process reading a semi-executable or script file with a suspicious label.*

Columns "Trace" in Table 1 show the details of the tracing action.

A script file is written in interpreting language, e.g. JavaScript or VBScript, and thus needs execution engine, e.g. wscript.exe or cscript.exe, to load and run it. Accordingly, to defend against a script virus, Tracer should restrict the engine processes that are reading and interpreting a suspicious script file. On the other hand, a semi-executable

represents certain types of data files that might contain executable codes, which mainly involves various types of compressed files and Microsoft Office documents. Compressed files such as zip files might include executable files, and Office documents such as Word files might enclose macro virus. Although the macro virus protection in Office software can reduce the chances of macro virus infection, relying on it is very dangerous because crafted macro codes are able to subvert it and cause destructive damages, for example, viruses Melissa and W97M.Dranus.

3.4 Restricting Intruders

In order to disable malware programs on a host, the restricting action monitors and blocks intruders' requests for executing critical malware behaviors listed in Figure 1. To follow the principle of complete mediation [13] for building a security protection system, Tracer further restricts two extensive behaviors, called **generic malware behaviors**, to protect security more widely. The first one is "Steal confidential information", which represents all illegal reading of confidential information from files and registry entries. The other is "Damage system integrity", which represents all illegal modifications of the files and registry entries that require preserving integrity. In addition, other behaviors that can be used to bypass Tracer mechanism also need to be monitored and restricted, including "Change file attributes", "Change registry entry attributes", "Execute non-executable files" and "Execute Tracer special system calls". The behavior "Change file attributes" represents changing file extension names to executable or changing file DAC information. All behaviors restricted are listed on the column "restrict" in Table 1. In summary, the restricting action consists of three rules:

- *Restricting critical malware behaviors*
- *Restricting generic malware behaviors*
- *Restricting behaviors bypassing Tracer*

By mediating all these behaviors, Tracer is able to preserve system security and prevent a malware program from propagating itself in the system. To be specific, confidentiality is mainly achieved by blocking the generic behavior "Steal confidential information"; integrity is mainly protected by blocking the generic behavior "Damage system integrity"; availability is defended by blocking the behaviors listed in Figure 1 with the capital letter A attached; propagation is prevented by blocking the behaviors in Figure 1 with the capital letter P attached.

Meanwhile, blocking these behaviors can help to defend against unknown malware programs for two reasons. First, these behaviors are extracted from thousands of malware samples and thus represent popular hacking techniques that are often used in unknown malware programs by malware authors. For example, the behavior "Add IE or Explorer plug-in" is also a popular technique that is used by enormous amount of malware programs both known and unknown to hide and automatically launch themselves, as well as monitor user data. Second, these behaviors are high-level behaviors so that they widely cover various low-level behaviors of various types of malware programs known or unknown. For example, "Communicate with a remote host" involves downloading hacker tools, sending emails to spread malware programs, connecting with a remote host to accept hacker commands, etc. Particularly, the two generic malware behaviors presented previously actually cover all illegal accesses of files/directories and registry entries in the system.

To efficiently restrict these malware behaviors, two issues need to be addressed. The first is how to determine the generic malware behaviors. We identify behaviors "Steal confidential information" and "Damage system integrity" by monitoring illegal reading on read-protected objects and illegal writing on write-protected objects, respectively. However, it is difficult to identify the objects that need protection among a large number of candidates in a Windows OS in order to recognize the generic malware behaviors. A traditional MAC requires users to give every object a security label to identify whether the object needs protection, which in turn becomes a heavy burden on general users.

In Tracer, we use the DAC information of an object to determine whether it is protected. To be specific, a file, directory or registry key is treated as read-protected when the user group "users" does not have a read permission on it. A file, directory or key not readable by "users" means that it should not be readable by the world, and thus should be read-protected. Similarly, a file, directory or key not writable by "users" is treated as write-protected. For other types of objects, e.g., IPC objects and system devices, we use "everyone" group to recognize protected objects.

However, considering the complexity and diversity of practical application scenarios, it is inevitable to require some exceptions for this method. Based on our extensive experiments, we design four novel rules to handle exceptions: (1) allowing exceptional read and write if the path of the targeted registry key contains the program name or alias of the process requesting the access, as such key stores the process' exclusive data; (2) allowing exceptional read and write if the path of the targeted file or directory simultaneously contains the program and user names of the process requesting the access, as such file or directory stores the process' exclusive data; (3) allowing exceptional write if the targeted file, directory or key is commonly writable by various programs, as such object stores the output data of multiple processes across the system; (4) allowing exceptional read and write if the targeted IPC or device object is providing system wide service.

DAC information and file extensions are not allowed to be changed by attackers. As a result, attackers can not alter a file or registry entry from a protected state to an unprotected state to escape the access control mechanism. With above methods, the configuration work required to identify files and registry entries to be protected is significantly reduced without changing the user's usage convention.

The second issue is how to identify the file-copying like behaviors in Figure 1 that require correlating two system calls for reading and writing files respectively. These behaviors are frequently used by malware programs but at the same time difficult to be detected without a hardware-level taint tracking which is not applicable to an online system [1]. From the work of literature, we did not find a proper online approach to detect all file-copying behaviors. Thus, we devise a pair of novel algorithms to

correlate the read and write operations to identify a file-copying behavior, which are shown in Algorithms 1 and 2. The Algorithm 1 intercepts a read operation and adds the file name and read buffer into the read-list. The Algorithm 2 intercepts a write operation and determines that it serves for a file-copying behavior if the content of the write buffer is equal or similar to a read buffer recoded in the read-list. A read-list of a process consists of a list of nodes each of which stores a read file and its buffers. The maximum number of nodes and memory size for the list are limited to prevent DOS attacks. The algorithms only monitor the files that are copying-behavior-involved including executables, special configuration files (e.g., desktop.ini and autorun.inf) and processes' image files.

The algorithms may impose a relatively high overhead only on the malware processes that frequently exhibit file-copying behaviors but not on benign processes and the suspected processes that are actually benign. For a benign process, according to the detection action at interior presented at the end of Section 3.2, the algorithms only need to monitor the file-copying behavior "Copy itself" by watching the read operation on the process' image file. However, in reality a benign process rarely tries to copy itself, thus the read-list is often empty and the algorithms do not need to do anything. For a suspected process that is actually benign, it rarely reads an executable at runtime. It only needs to read an executable in two situations while such read operations do not need to be recorded into the read-list. One is to load an image file for a process. As a loaded image in memory is often marked as "read forbidden", a file-copying behavior will not happen. The other situation is to install software. However, processes installing software from the local host or from a remote host through a trusted communication will be deemed as benign according to the detecting action presented in Section 3.2. Therefore, the read-list of a benign process is often empty or short. A short read-list in turn leads to a low performance overhead, because the Algorithm 2 will spend little time searching the matched read buffer from the read-list. Our performance evaluation in Section 4.2 will further demonstrate that actually benign programs have lower performance overhead than that of malware programs.

In addition, this pair of algorithms that correlate read and write operations by comparing buffer contents are more difficult to be circumvented than other candidate algorithms, e.g., comparing buffer addresses. In the worst case that a malware program successfully circumvents the algorithms, Tracer still can tail it by monitoring related behaviors, e.g., "Create executables", since file-copying behaviors need to create executables.

Algorithm 1. Recording file-reading operations

```
INPUT: file A to be read; read buffer R
IF((file A is not copying-behavior-involved)OR((the current process is benign)AND(file A is not an image file of the current process)))
    RETURN permit the operation;
END
IF((file A is a copying-source)OR(the memory or nodes of the read-list reach the maximum))
    RETURN block the operation;
END
FOR each node in the read-list
    IF(file A is in the node)
        Attach buffer R into the node;
        BREAK;
    END
END
IF(file A is not in the read-list)
    Create a new read operation node;
    Fill file A into the node;
    Copy buffer R into the node;
    Add the new node into the read-list;
END
RETURN permit the operation;
```

Algorithm 2. Detecting file-copying behaviors

```
INPUT: file B to be written; write buffer W
IF(the read-list is null)
    RETURN permit the operation;
END
IF(file B is a copying-target)
    RETURN block the operation;
END
FOR each node in the read-list
    FOR each read buffer in the node
        Compare the read buffer and buffer W;
        IF(the buffers are equal or similar)
            IF(the file of the node is the image of the process)
                Identify the behavior "Copy itself";
            ELSE IF(file B is an executable in removable drive)
                Identify the behavior "Copy executables to removable drives";
            ELSE IF(the file of the node is a system executable)
                Identify the behavior "Copy system executable files";
            ELSE IF(the file of the node is a special configuration file)
                Identify the behavior "Copy special configuration files";
            END
            BREAK;
        END
    END
    IF(identified a file-copying behavior)
        Mark file B as a copying-target;
        Mark the file of the node as a copying-source;
        Destroy the node;
        RETURN block the operation;
    END
END
RETURN permit the operation;
```

3.5 Dynamic Addition of New Behaviors

If the detection is purely based on known malware characteristics and behaviors, a detector may not be able to function effectively in the long run as new malware characteristics and behaviors may emerge over the time. To address this limitation [29], a novel extensible mechanism is implemented in Tracer so it can dynamically add in new behaviors to monitor.

A behavior consists of operation, object and parameter. An operation is an abstract of one or several system calls with similar functions. For example, the operation create_file corresponds to two system calls: NtOpenFile and NtCreateFile. In contrast, a single system call may contain more than one operation. For example, NtOpenFile contains four operations: read_file, write_file, create_file, and delete_file. The object and parameter of a behavior are extracted from a related system call.

Figure 3 illustrates how to dynamically add malware behaviors. In each concerned system call, we set up one or more checkpoints, each of which is responsible for checking the behaviors belonging to the same operation with the support of a modifiable behavior list in memory. The new malware behaviors are read from a configuration file and distributed to proper behavior lists corresponding to

different operations in memory. At each checkpoint, Tracer searches for the object and parameter currently requested in the corresponding list to determine whether the current access forms a malware behavior.

4 IMPLEMENTATION

To evaluate the effectiveness of Tracer approach, we have developed a prototype implementation for Windows XP, and carried out a series of experiments. Although XP is not as new as Vista, it is enough for verifying the Tracer approach since both versions of OS have very similar system calls and Win32 API functions based on which Tracer works. Moreover, if developing the prototype on Vista, the MIC might interfere with Tracer as both schemes attempt to complete MAC tasks.

4.1 Implementation

Tracer implementation consists of two parts: Interception and Decision. Most of the implementations are located in the kernel so that they are difficult to be bypassed. The Interception part monitors Native Windows API functions (i.e. system call) at the kernel level and Win32 API functions (i.e. system library functions) at the application level, then issues behavior requests to the Decision part, and allows or disallows a behavior according to the result returned from the Decision part. The intercepted behaviors are listed in Table 1.

Except the "file copying" like behaviors presented in Section 3.4, all other behaviors can be intercepted by monitoring only one essential system call function or a Win32 API function, for example, monitoring NtDeviceIoControlFile() for "Communicate with a remote host", monitoring NtCreateFile() for "Create executable", monitoring NtOpenFile() for "Steal confidential information". Some behaviors consist of more than one system call or Win32 function, for instance, the behavior "Inject into other processes" consists of OpenProcess(), VirtualAllocEx(), WriteProcessMemory(), CreateRemote-Thread(), etc. Considering the performance impact, we only intercept the first essential function, i.e. OpenProcess(), and block it if a suspicious process tries to perform an execution, such that the subsequent calls, i.e.WriteProcess-Memory() and CreateRemoteThread(), which would cause damages are not executed any more. Moreover, to prevent intended bypassing, Tracer always intercepts a function at the kernel level rather than the application level if possible. Thus, for the behavior "Inject into other processes", Tracer actually intercepts NtOpenProcess() at the kernel level rather than OpenProcess() at the application level.

The Decision part residing in the kernel handles behavior requests from the Interception part. When making a decision, it first reads the Tracer attributes of processes and files, e.g., suspicious flags and DAC information, and then decides whether to permit the behaviors and whether to modify the Tracer attributes according to the Tracer actions presented in Section 3. Table 1 shows the decision logic implemented in the Decision part.

To be permanent, the suspicious flag of an executable is stored in a specially created file stream of the executable file. The suspicious flag of a process, however, is stored in a data structure associated with the process in the memory. The data structure also records whether the process has received a network package through non-dangerous protocols. The whole implementation is encapsulated in a kernel driver and a DLL. The kernel driver is responsible for intercepting system calls via modifying the system call entry point in the System Service Dispatch Table (SSDT), and implementing the Decision part within the kernel. The DLL is responsible for intercepting Win32 API functions via modifying the library function entry point in the Import Address Table (IAT) of application processes. Note that, our Tracer implementation does not need to impose any modifications on the Windows or application codes, thus it is highly compatible with existing software.

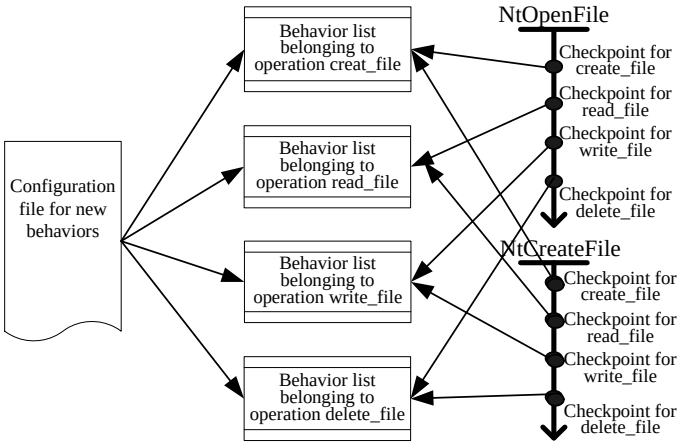

Fig. 3. The mechanism to dynamically add new malware behaviors to be monitored

4.2 Evaluation

We evaluate Tracer performance from three important perspectives: its effectiveness in ensuing security, its compatibility with application software, and the overhead added after enabling Tracer on OS.

Security. To verify the capability of Tracer on restricting malware behaviors, we collected 93 real-world malware samples, most of which are obtained from a publicly available website [20]. 32 of the samples are unknown to Tracer, because they can not be found with the same or different names in Symantec Threat Explorer from which the critical malware behaviors are extracted. We also prepared 54 benign samples mostly from two reputable 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 which consists of two servers and two hosts as a testing environment. One server machine, on which the samples to be tested are intentionally placed, runs an IIS web server, an FTP server and an EZ-IRC server. The other server machine, on which only benign samples are placed, runs an IIS web server to act as a trusted site for testing trusted communications. Note that, in reality the trusted sites can be easily recognized by general users because a host only

TABLE 2. SECURITY TEST RESULTS.
FP Rate is 5.6% and FN Rate is 0%.

Samples		Programs			Behaviors		
		Total	FNs	FPs	Total	FNs	FPs
Known malware	Worm	20	0	-	274	0	-
	Trojan	19	0	-	155	0	-
	Backdoor	17	0	-	152	0	-
	Script Virus	2	0	-	65	0	-
	Macro Virus	3	0	-	49	0	
Unknown malware		32	0	-	491	0	-
Benign program	Security utilities	11	-	1	103	-	8
	System utilities	10	-	2	83	-	15
	Games	7	-	0	82	-	0
	Multi-media	10	-	0	36	-	0
	Web Pages	16	-	0	99	-	0
Sum		147	0	3	1589	0	23

has to connect to several well-known websites to upgrade its important software. The host machines installed with Windows XP run the client programs that are often the attacking vectors for malware samples, including mIRC, MSN Messenger, MS Outlook, eMule, KaZaA, IE and FTP client, etc. We set protocols HTTP, POP3, IRC, SMTP, FTP, FastTrack, eDonkey and ICMP as dangerous.

To emulate the real-world usage scenarios, we login the hosts and perform various types of tasks, such as browsing the malicious website and FTP server in the local network and downloading samples, sending and receiving malicious instant messages and emails, accessing P2P shared folders or removable drives that contain samples. Thus, the samples are introduced into a host through various channels. With this testing environment, the capability of Tracer to detect, trace and restrict malware behaviors can be thoroughly evaluated.

For every sample, we perform a two-step experiment. First we run a sample on a host without turning on Tracer and record what happens using the monitoring tools above. Then, we enable Tracer protection, run the same sample, and record what happens again. We can determine whether a sample is indeed disabled from two perspectives. First, we deduce whether malware behaviors are successfully executed by comparing the two versions of logs produced by ApiMonitor, Regmon and Filemon without or with protection. Second, we manually check whether the files, registry entries and processes that are created by the sample and recorded in the former logs are exactly present or not in the logs after turning on the Tracer. Moreover, we restart the computer to see if the sample can be enabled automatically.

The testing results are reported in Table 2. For each type of samples, after turning on Tracer, we record the number of false negatives, i.e., FNs, and the number of FPs. We can see that Tracer was able to correctly disable all malware samples including known and unknown ones, as well as block or cancel all their malware behaviors. However, it falsely stopped 3 benign samples by blocking their behaviors. The FPs were a personal firewall program, a file system tool and a process tool, downloaded from the IRC and web server with which we did not set up a trusted communication. By analyzing the logs, we observed that some behaviors of these benign programs closely resemble those of malware, for example, "Create or modify Windows services", "Modify system configuration files", "Install or modify drivers", "Modify registry for startup", etc. As Tracer relies on the source and behaviors of a program to identify a malware program, the benign programs that come from a remote host through an untrusted communication are tracked and restricted as suspicious ones. However, one still can make the programs work by manually removing the suspicious flags from the program files before running them.

Table 3 further describes the detailed test results of 20 selected malware samples. We can see that all the malware samples are successfully disabled via the restriction of their malware behaviors. For example, the worm "Worm.Win32.Leave.i" downloaded from the local website has the following main steps for function: it firstly copies itself, i.e., regsv.exe, to C:\Windows, then runs regsv.exe as a new process, the new process then adds a value under registry key HKEY_LOCAL_MACHINE\Software\Microsoft\Windows\CurrentVersion\Run to point to C:\WINDOWS\regsv.exe so that it can be launched when the system restarts, finally listens at port 113 to accept commands from a remote attacker. On a host without Tracer enabled, all above steps are successfully executed. However, after activating the Tracer protection, the malware behavior "Copy itself" is blocked, i.e., the malware can not create a new copy of itself in the system folder. Consequently, the rest of the behaviors do not appear any more because these behaviors depend on the new process launched from the malware's copy, i.e. C:\WINDOWS\regsv.exe. In other words, the worm is disabled.

To compare with other anti-malware techniques on Windows, we performed an experiment to test three popular commercial tools: Kaspersky [27], VIPRE [28] and MIC. The former two running on XP are well known anti-malware tools and have modules blocking suspicious behaviors to defend against unknown malware. The anti-malware tools relying only on signatures can not detect unknown malware [1] and thus are inappropriate to compare with Tracer especially on FP rate. MIC is a partial enforcement of BIBA model in Vista kernel [16], which is the only MAC mechanism in Windows OS family. For every anti-malware technique, we tested all of the samples in Table 2. We count a program as a FP if the anti-malware technique abnormally refuses or alarms at least one of its access requests, since this will affect the running of the testing program or annoy the user. We do not count a program as a FP if it fails on Vista but the failure is not caused by MIC. Figure 4 shows the FP rates (FPR) obtained. MIC and the anti-malware tools have FP rates above 34%, whereas, Tracer has FP rate of merely 5.6%. The high FP rate of MIC comes from the no-write-up rule of BIBA model. The modules that block suspicious behaviors contribute to most of FPs of the anti-malware tools. The fundamental reason is that the anti-malware tools identify a suspicious behavior only based on the behavior itself while Tracer further considers the suspicious label of the process requesting the behavior. On the other hand, the FN rates of Kaspersky, VIPRE and Tracer are almost all zero. However, MIC is observed to have a high FN rate of 42%.

TABLE 3. DETAILED TESTING RESULTS

Malware	Channels	Restricted Behaviors
Worm.Win32.Leave.i	Web	Copy itself
Net-Worm.Win32.Welchia.a	Web	Copy itself, Create or modify Windows services, Restart computer
Trojan-Spy.Win32.Dks.11.a	Web	Copy itself, Modify registry for startup, Create Windows hooks
IRC-Worm.Win32.Fagot.a	Web	Copy itself, Modify registry for startup, Change security settings, End anti-malware processes or services
Trojan-PSW.Win32.QQlog.b	FTP	Copy itself, Modify registry for startup
Trojan.Swizzor.1	FTP	Copy itself, Start hidden network clients, Inject into other processes, Modify registry for startup
Backdoor.Win32.Gobot.r	IM	Copy itself, Modify registry for startup, End anti-malware processes or services, Modify executables, Restart computer, Steal confidential information
Backdoor.Win32.Rbot.15	IM	Copy itself, End anti-malware processes or services, Create Windows hooks , Capture screen shots, Modify registry for startup, Change security settings, Steal confidential information, Create or modify Windows services, Install or modify drivers
Backdoor.Win32.SdBot.04.d	IM	Copy itself, Modify registry for startup, Steal confidential information
Virus.VBS.GaScript.b	Email	Copy itself, Modify registry for startup
Email-Worm.Win32.Kitro.d	Email	Copy itself, Modify registry for startup, Steal confidential information
Email-Worm.Win32.Centar.j	Email	Copy itself, Modify registry for startup, Steal confidential information
Trojan.Win32.KeyPanic.b	Email	Modify registry for startup, Create Windows hooks
P2P-Worm.Win32.Sytro.a	P2P	Copy itself, Modify registry for startup, Damage system integrity
P2P-Worm.Win32.Skater.a	P2P	Copy itself, Modify registry for startup
P2P-Worm.Win32.Surnova.k	P2P	Copy itself, Modify registry for startup, Damage system integrity
Backdoor.Win32.Agobot.an	RPC	Copy itself, Modify registry for startup, Create or modify Windows services, Steal confidential information, End anti-malware processes or services, Modify system configuration files
Rootkit.Win32.Agent.h	removable drive	Install or modify drivers
Backdoor.Win32.MoSucker.06	removable drive	Copy itself, Modify registry for startup, Restart computer, Steal confidential information
Backdoor.Win32.SilentSpy.209	removable drive	Copy itself, Modify registry for startup, Create Windows hooks

One possible reason is that MIC does not implement the no-read-down rule of BIBA model [16] in order to avoid a significant impact on the usability and compatibility of Windows which is a commodity OS. As a result, some sophisticated malware programs can manage to bypass it. Nevertheless, with MIC, Vista can still achieve a significant security improvement compared with XP that can not defeat any malware samples by itself.

Compatibility. The requirement for compatibility is that existing Commercial Off-The-Shelf (COTS) software can run on the MAC prototype without causing significant incompatibility problems. On the two hosts with Windows XP installed, we run many commonly used network-dependent applications and local applications e.g. Internet Explorer, MS Outlook Express, MS word, MS excel, MS Power Point, MS Messenger, mIRC, Visual C++, Firefox, Adobe Reader, QQ, Foxmail, Windows Media Player, Putty SSH client, WinRAR, VMWare Workstation, AVG Antivirus, Windows tasklist, Skype, Windows FTP client, Beyond Compare, Source Insight, Calculator, Utility Manager, Notepad, Minesweeper, Hearts, WebBench Client, WebBench Controller, Winamp. We send emails, browse websites through Internet, edit word documents, develop VC++ programs, share files remotely, update Windows and move files through USB disks, etc. The system works well for the past a few months, without need of modifications of existing software or running into failures.

Performance overhead. The performance overhead of Tracer comes from the overhead of executing additional instructions associated with every intercepted system call and API function. In the following experiment, we evaluate the additional overhead imposed by Tracer enforcement. The test-bed is a Pentium-4 2.8GHz machine with 1GB memory running Windows XP SP2. We first disable Tracer, run a group of benign and malware programs, and count the average CPU cycles spent in each system call and API function through rtdsc instruction. Second, we enable Tracer, run the malware programs, the benign programs with suspicious flags and without suspicious flags to perform the test again. In all tests, the average CPU cycles of every system call or API function is calculated from 100 invokes. Results are shown in Table 4. With Tracer enabled, the malware programs have 1.7%~38.1% more performance penalty than native, while the benign programs have only 0~13.5%. The highest performance penalty comes from the interception of NtWriteFile() as a result of capturing file-copying behaviors. The overhead incurred on benign programs is lower than 2%. Therefore, the general performance impact from the system call and Win32 API function interception is acceptable.

5 DISCUSSIONS

5.1 Security

Security Protection. According to the ring policy of BIBA model [8], Tracer can correctly protect integrity if satisfying the following three conditions: (1) any subject (S) can read any object (O), regardless of their integrity levels (i); (2) $s \in S$ can write $o \in O$, only if $i(o) \leqslant i(s)$; (3) $s1 \in S$ can invoke $s2 \in S$, only if $i(s2) \leqslant i(s1)$. The first condition means that the ring policy has no requirements on read operations. The second condition forbids the write operations where $i(o) > i(s)$. Tracer meets this condition by restricting the malware behaviors of low integrity level subjects (i.e., suspicious processes) that try to write high integrity level objects (e.g., executables, system configuration and

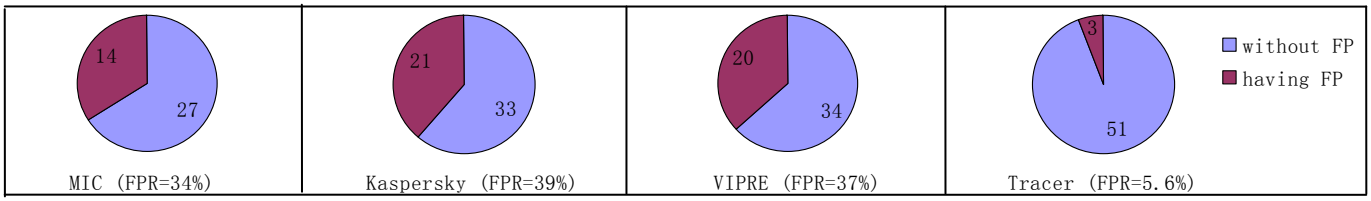

Fig. 4. Comparing false positives with commercial anti-malware techniques on Windows

TABLE 4. OVERHEAD OF TRACER (CPU CYCLES)

The columns Tracer-m, Tracer-bf and Tracer-b show the CPU cycles taken by the malware programs, the benign programs with and without suspicious flags running on Tracer, respectively.

Functions	Native	Tracer-m	Tracer-bf	Tracer-b	Functions	Native	Tracer-m	Tracer-bf	Tracer-b
NtCreateFile	334492	348523(4.2%)	348197(4.1%)	338506(1.2%)	CreateService	6568120	6679969(1.7%)	6679625(1.7%)	6568323(<0.1%)
NtOpenFile	167620	175311(4.6%)	173235(3.3%)	169713(1.2%)	OpenService	5490443	5609529(2.2%)	5609352(2.2%)	5490560(<0.1%)
NtWriteFile	245179	338524(38.1%)	278286(13.5%)	249897(1.9%)	NtSetValueKey	210491	225185(7%)	225093(6.9%)	210493(<0.1%)
NtCreateNamedPipeFile	204711	214798(4.9%)	214751(4.9%)	204789(<0.1%)	NtCreateKey	281722	296451(5.2%)	296008(5.1%)	281784(<0.1%)
NtCreatePort	37241	40281(8.2%)	40180(7.9%)	37275(<0.1%)	NtCreateProcess	206458	215487(4.4%)	215426(4.3%)	208849(1.2%)

write-protected objects). According to Section 3.4, the "damage system integrity" behavior together with other write style (e.g., modify, change, install and create) behaviors listed in Table 1, cover all categories of objects that need integrity protection, including files, directories, registry keys, IPC objects, processes and system devices. Thus, by preventing such behaviors, Tracer is able to protect the high integrity objects of the whole system. In other words, the second condition is satisfied. The third condition requires preventing a low integrity level subject from launching a high integrity level subject. In Tracer, processes generated by a suspicious process are always suspicious according to the tracing rules. In other words, the third condition is met. Therefore, Tracer satisfies the ring policy of BIBA model.

Tracer also can provide certain confidentiality protection by restricting the read style malware behaviors of low confidentiality level subjects (i.e., suspicious processes) that try to observe high confidentiality level objects. According to Section 3.4, the "Steal confidential information" behavior together with other read style behaviors listed in Table 1 actually cover all categories of sensitive objects across the system. Hence, by stopping such behaviors, Tracer can prevent direct leakage of secrecy. However, it can not prevent indirect leakage. For example, a benign process might unconsciously read sensitive information and then output to a file without read protection. We will try to resolve this issue in the future.

Moreover, by restricting the behaviors that could affect the system availability, e.g. "End anti-malware processes or services" and "Restart computer", Tracer is able to protect availability to a certain degree.

Implementation Security. Tracer modules in Windows can act as a reference monitor to completely monitor all dangerous operations, and is tamperproof, always-invoked, carefully analyzed and tested as well. First, Tracer prototype is difficult to be bypassed or subverted. This is because most of the implementations of Tracer are located in the kernel, and the behaviors that can be employed to bypass or subvert Tracer, e.g. "Change file attributes", "Execute Tracer special system calls", "Execute non-executable files", "Install or modify drivers" and "Create Windows hooks", are prohibited from being executed by suspicious processes. Second, as presented in Section 3.4, Tracer prototype obeys the principle of complete mediation [13]. That is, the monitored behaviors including critical malware behaviors, generic malware behaviors and the bypassing Tracer behaviors actually cover all security sensitive operations on Windows. The complete coverage is formed mainly because that the two generic behaviors, "Damage system integrity" and "Steal confidential information", represent all illegal operations on the protected objects in an OS. Third, Tracer modules are carefully analyzed and tested as they are independent from Windows OS and concise in internal logic.

Potential Evasions. There are three potential evasions in Tracer. The first evasion exploits the trusted communications to control a privileged process or download an executable without being attached with a suspicious flag. However, as presented in Section 3.2, the attack surface on trusted communications is tremendously narrowed by checking the image file, protocol, remote host and time simultaneously. Even if a malware program manages to get into a host, Tracer is still able to detect and restrict the malware behaviors. According to our investigations presented in Section 2.1, all malware samples need network communications for their functions, e.g. downloading tools from malicious website. Thus Tracer can detect a malware program by the detection action at network entrance. Moreover, the detection action at interior also has many chances of detecting the malware program by monitoring exclusive malware behaviors in the system. Although these approaches may not ensure absolute security, there is a tradeoff between ensuring a higher security and ensuring a good usability when there is a need to facilitate system maintenance from the remote site.

The second potential evasion breaks intrusion tracing via compromising a local IPC that is not considered as dangerous. According to our investigation on Microsoft Security Bulletin mentioned in Section 3.3, it is considerably difficult to evade tracing through a local IPC. Although it is rare, the evasion will occur when a malware author exploits an IPC that is detected to be dangerous but not released to the public, and thus cannot be traced by Tracer as it is not

included in the Dangerous-IPC-List. Even under this situation, Tracer is still effective to recognize and confine the malware to a certain extent because the network traffic and exclusive malware behaviors will activate the detections at entrances and interior respectively. There is a tradeoff between a better security and a good compatibility as tracing extra IPCs would lead to the result that a huge number of benign processes are marked as suspicious and thus unexpectedly restricted.

The last potential evasion is to exploit kernel vulnerabilities. Tracer can defend against most kernel rootkits by restricting the behaviors of suspicious processes which can be used to compromise kernel integrity, including "install or modify drivers", "create Windows hook", "modify executable files" and "inject into other processes". However, if a malware exploits kernel vulnerability of an OS which is not updated timely to fill the security holes, the Tracer might be bypassed. This is a common limitation of all access control mechanisms [30] since they live inside the OS kernel. In this case, we can use Tracer together with a kernel integrity preserving mechanism [24][30] that lives outside of the OS kernel. These two can complement each other, as a single mechanism is difficult to detect both general malwares and the special kernel rootkits that compromise kernel.

5.2 Compatibility

The good compatibility of Tracer is resulted from the reduction of false positives. First, it directly stops malicious behaviors rather than stop illegal information flow that may incur FPs as presented in Section 2.2. Second, when determining a malicious behavior, Tracer considers not only the behavior itself but also the security label of the process. As the security label is the result of a historic behavior, Tracer actually determines a malicious behavior based on two behaviors. Suppose using the current behavior or the historic behavior alone to make decision would produce FP rate $0 < p < 1$ or $0 < q < 1$, respectively. As in most cases the two behaviors are independent, using the two behaviors simultaneously would produce FP rate $p*q$ where $p*q < \min(p, q)$. Hence, Tracer can achieve a lower FP rate as compared to a scheme that simply monitors one of the behaviors. Third, Tracer only blocks the behaviors with a small FP rate. For the rest of the behaviors, Tracer traces them rather than blocks them immediately. This would reduce the overall FP rate of Tracer. Third, for the situations similar to malware activities, we devise exception mechanisms to avoid producing a huge number of false positives, which include trusted communications for remote system maintenance and exception rules for identifying generic malware behaviors. This helps to further reduce the overall FP rate of Tracer.

5.3 Usability

Usability concerns with the amount of configuration work and the impact on user's usage convention. Most of the configuration work of a MAC system is to arrange security labels to a large number of entities in an OS. Tracer is able to automatically fulfill this task by detecting and tracing potential intruders. If considering the leverage of DAC information to denote an entity to be protected, the configuration work is then further reduced to nearly zero. Thus, the rest of configuration work only involves dangerous protocols and trusted communications that can be specified by a default setting. On the other hand, by utilizing existing operating system information, i.e. DAC permissions and file extension, to identify files and registry entries that require protection, Tracer follows conventional usage styles very well.

6 RELATED WORK

DTE proposed by Lee Badger et al. [9] is a classical MAC model to confine process execution, which groups processes and files into domains and types respectively, and controls accesses between domains and types. Tracer can be regarded as a simplified DTE that has two domains (i.e., benign and suspicious) and four types (i.e., benign, read-protected, write-protected and suspicious). Moreover, Tracer can automatically configure the DTE attributes (i.e., domain and type) of processes and files under the support of intrusion detection and tracing so as to improve usability.

LOMAC [3], UMIP [2], PPI [7] and MIC [4] aim to add usable and compatible mandatory integrity protections into mainstream operating systems. LOMAC deals with the pathological cases in the Low-Water Mark model's behaviors to decrease its partial compatibility cost. UMIP is designed to preserve system integrity in the face of network-based attacks in a highly usable manner. PPI automates the generation of information flow policies by analyzing software package information and logs. MIC implements the no-write-up rule of classical BIBA model in Windows Vista kernel, but it does not implement the no-read-down rule in order not to compromise compatibility significantly. PRECIP [5] addresses several practical issues that are critical to contain spyware that intends to leak sensitive information. Tracer, however, differs from these MAC models in that, it traces suspected intruders and restricts their behaviors rather than restricts information flow. With this novel concept, it is able to considerably reduce FPs and automatically deploy security labels, which result in good compatibility and usability. Meanwhile, the philosophy of Tracer is roughly similar to the risk-adaptive access control [10] that targets to make access control more dynamic so as to achieve a better tradeoff between risk and benefit. Tracer dynamically changes security labels of certain processes to reduce the risk of executing malware behaviors, while not restricting other behaviors and processes at all to preserve the benefits of compatibility and usability.

Most existing anti-malware technologies are based on detection [22][23]. Tracer tries to combine detection and access control so that it not only can detect but also can block malware behaviors before their harming security. Another anti-malware technology that resembles Tracer is behavior blocking [25], which can confine the behaviors of certain adverse programs that are profiled in advance. However, Tracer does not need to profile program behaviors beforehand, and can confine the adverse programs that execute malware behaviors.

Many commercial anti-malware tools [27][28] also have a behavior-based module to defend against unknown

malware programs. The major difference between Tracer and the commercial tools is that Tracer determines a malicious behavior based not only on the behavior itself but also the security label of the process requesting the behavior, rather than merely the behavior as anti-malware tools do. As a consequence, Tracer produces much less false positives than that of the commercial-tools as shown in Section 4.2.

7 CONCLUSIONS

In this paper, we propose a novel MAC enforcement approach that integrates intrusion detection and tracing to defend against malware in a commercial OS. We have extracted 30 critical malware behaviors and three common malware characteristics from the study of 2,600 real-world malware samples and analyzed the root reasons for the incompatibility and low usability problems in MAC, which will benefit other researchers in this area. Based on these studies, we propose a novel MAC enforcement approach, called Tracer, to disable malware timely without need of malware signatures or other knowledge in advance. It detects and traces suspected intruders so as to restrict malware behaviors. The novelty of Tracer design is two-fold. One is to use intrusion detection and tracing to automatically configure security labels. The other is to trace and restrict suspected intruders instead of information flows as done by traditional MAC schemes. Tracer doesn't restrict the suspected intruders right away but allows them to run as long as possible except blocking their critical malware behaviors. This design produces a MAC system with good compatibility and usability. We have implemented Tracer in Windows OS and the evaluation results show that it can successfully defend against a set of real-world malware programs, including unknown malware programs, with much lower FP rate than that of commercial anti-malware techniques.

REFERENCES

[1] H. Yin, D. Song, M. Egele, C. Kruegel, and E. Kirda. Panorama: capturing system-wide information flow for malware detection and analysis. In Proceedings of the 14th ACM conference on Computer and communications security (CCS '07). Pages 116-127.

[2] N. Li, Z. Mao, and H. Chen. Usable Mandatory Integrity Protection for Operating Systems. In Proceedings of the IEEE Symposium on Security and Privacy (SP '07), pages 164-178.

[3] T. Fraser. LOMAC: Low Water-Mark Integrity Protection for COTS Environments. In Proceedings of the IEEE Symposium on Security and Privacy (SP '00). Pages 230-245.

[4] Microsoft, Mandatory Integrity Control, http://en.wikipedia.org/wiki/Mandatory_Integrity_Control

[5] X. Wang, Z. Li, J. Y. Choi, N. Li. PRECIP: Towards Practical and Retrofittable Confidential Information Protection. In Proceedings of 15th Network and Distributed System Security Symposium, 2008.

[6] Symantec, Inc, http://www.symantec.com/business/security_response/threatexplorer/threats.jsp.

[7] W. Sun, R. Sekar, G. Poothia and T. Karandikar. Practical Proactive Integrity Preservation: A Basis for Malware Defense. In Proceedings of the IEEE Symposium on Security and Privacy (SP'08).

[8] K. J. Biba. Integrity considerations for secure computer systems. Technical Report MTR-3153, MITRE, April 1977.

[9] L. Badger, D. F. Sterne, D. L. Sherman, K. M. Walker, and S. A. Haghighat. 1995. Practical Domain and Type Enforcement for UNIX. In Proceedings of the IEEE S&P 1995. Pages 66-77.

[10] P.-C. Cheng, P. Rohatgi, C. Keser, P. A. Karger, G. M. Wagner, and A. S. Reninger. Fuzzy Multi-Level Security: An Experiment on Quantified Risk-Adaptive Access Control. In Proceedings of the IEEE Symposium on Security and Privacy. Pages 222-230.

[11] M. Howard, Fending Off Future Attacks by Reducing Attack Surface http://msdn.microsoft.com/en-us/library/ms972812.aspx, 2003.

[12] M. Oers, OSX Malware not taking off yet, http://www.avertlabs.com/research/blog/index.php/2007/03/20/osxmalware-not-taking-off-yet/, 2007.

[13] J. Saltzer and M. Schroeder. The protection of information in computer systems. Communications of the ACM, 17(7), 1974.

[14] P. Loscocco and S. Smalley. Integrating Flexible Support for Security Policies into the Linux Operating System. In Proceedings of the FREENIX Track: 2001 USENIX Annual Technical Conference, Clem Cole (Ed.). Pages 29-42.

[15] M. Jawurek, R. Aachen. RSBAC - a framework for enhanced Linux system security, http://www.rsbac.org/documentation, 2006.

[16] Microsoft, Windows Vista Integrity Mechanism, http://msdn.microsoft.com/en-us/library/bb625964(v=MSDN.10).aspx

[17] V. D. Gligor, C. S. Chandersekaran, R. S. Chapman, L. J. Dotterer, M. S. Hecht, W.-D. Jiang, A. Johri, G. L. Luckenbaugh, and N. Vasudevan. Design and Implementation of Secure Xenix. IEEE Trans. Softw. Eng. 13, 2 (1987), 208-221.

[18] S. T. King and P. M. Chen. Backtracking intrusions. In Proceedings of the 19th ACM symposium on Operating systems principles (SOSP '03). Pages 223-236.

[19] Microsoft Security Bulletins, http://www.microsoft.com/technet/security/current.aspx.

[20] Offensive Computing, http://www.offensivecomputing.net/.

[21] F. Hsu, H. Chen, T. Ristenpart, J. Li, and Z. Su. 2006. Back to the Future: A Framework for Automatic Malware Removal and System Repair. In Proceedings of the 22nd Annual Computer Security Applications Conference. Pages 257-268.

[22] E. Kirda, C. Kruegel, G. Banks, G. Vigna, and R. A. Kemmerer. Behavior-based spyware detection. In Proceedings of the 15th conference on USENIX Security Symposium (USENIX-SS'06).

[23] L. Martignoni, E. Stinson, M. Fredrikson, S. Jha, and J. C. Mitchell. A Layered Architecture for Detecting Malicious Behaviors. In Proceedings of the 11th international symposium on Recent Advances in Intrusion Detection, Pages 78-97.

[24] Z. Wang, X. Jiang, W. Cui, and P. Ning. Countering Kernel Rootkits with Lightweight Hook Protection. In Proceedings of the 16th ACM Conference on Computer and Communication Security, 2009.

[25] C. Nachenberg. Behavior Blocking: The Next Step in Anti-Virus Protection. http://www.securityfocus.com/infocus/1557, March 2002.

[26] A. Goel, K. Po, K. Farhadi, Z. Li, and E. Lara. The taser intrusion recovery system. In Proceedings of the twentieth ACM symposium on Operating systems principles (SOSP '05). Pages 163-176.

[27] Kaspersky Lab. http://www.kaspersky.com/.

[28] Vipre, Inc, http://www.vipre.com/vipre/.

[29] 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, pages 135~144, March 2011.

[30] A. Seshadri, M. Luk, N. Qu, and A. Perrig. SecVisor: a Tiny Hypervisor to Provide Lifetime Kernel Code Integrity for Commodity OSes. In Proceedings of the 21st ACM Symposium on Operating Systems Principles, 2007.

[31] E. Al-Shaer and H. Hamed. Discovery of policy anomalies in distributed firewalls. In *Proceedings of the 23rd IEEE Infocomm*, Hong Kong, Mar 2004.

Zhiyong Shan is an associate professor in the department of Computer Science of the Renmin University of China. He was a postdoctoral research associate in the department of computer science of the Stony Brook University. He received the PhD degree in computer science from Chinese Academy of Science. Dr. Shan won president award of Chinese Academy of Science in 2004 and Beijing Science & Technology Achievement Award in 2005. His research interests include operating system and computer security.

Xin Wang is an associate professor in the department of Electrical and Computer Engineering and an affiliated professor in the department of Computer Science of the Stony Brook University. She received the PhD degree in electrical and computer engineering from Columbia University. Her interests include wireless networks, mobile and distributed computing, computer security. She won NSF career award in 2001.

Tzi-cker Chiueh is a professor in the department of Computer Science of the Stony Brook University. He received the PhD degree in computer science from UC Berkeley. His research interests include computer security and storage system. He received an NSF CAREER award in 1995, a Best Paper Award from 2005 Annual Computer Security Applications Conference.