A Study on Malware Analysis Leveraging Sandbox Evasive Behaviors

（マルウェアの回避挙動を利用した動的解析に関する研究）

by

Takahiro KASAMA

March, 2014

Thesis submitted to
the Graduate School of Environment and Information Sciences,
Yokohama National University
for the Degree of Doctor of Philosophy in Engineering.

Principal Advisor: Professor Tsutomu MATSUMOTO

Acknowledgements

First, I would like to express my deepest appreciation to my supervisor, Professor Tsutomu Matsumoto, for his tremendous support, comments, and encouragement during the course of my study. His insights into research and his invaluable suggestions greatly inspired me in my research pursuits. I am extremely grateful to Associate Professor Katsunari Yoshioka who taught me how to conduct research and write technical papers. Without his help I would not have been able to complete my dissertation. I also gratefully acknowledge Associate Professor Junji Shikata whose insightful comments and advice significantly contributed to improving my dissertation. I would like to thank Professors Tomoharu Nagao and Tatsunori Mori for serving on my dissertation committee. Their invaluable comments and feedback were extremely helpful.

Special thanks go to Mr. Koji Nakao, Dr. Daisuke Inoue, Dr. Masashi Eto, and all my colleagues at the National Institute of Information and Communications Technology for fruitful discussions, insightful comments, and a beneficial working environment. I am also indebted to the past and present members of the Matsumoto Laboratory, especially Mr. Tatsunori Orii, Mr. Daisuke Mizushima, Mr. Taiki Kobayashi, and Mr. Kousuke Murakami. The days spent together with them are irreplaceable.

Finally, I would like to thank my family for their support and sacrifices.

Abstract

Internet security threats utilizing highly functional malicious programs called *malware* are recently on the rise, and extensive research efforts have been made to counter them. With this explosive increase of malware, it is becoming nearly impossible to manually analyze all its forms by reverse engineering. An effective countermeasure for this problem, malware sandbox analysis, in which a malware sample is executed in a testing environment (a sandbox) to observe its behaviors, has been widely studied. Malware authors have responded by making their work more sophisticated to evade this analysis. One example is a type of malware called a bot, which changes its behaviors in accordance with the behaviors of remote servers with which it interacts, such as Command and Control (C&C) servers and malware download servers. Since a bot does not work unless it meets the conditions for activation, it is difficult to analyze it sufficiently with traditional sandbox analysis. Another example is a type of malware that stops or changes its behaviors when it detects a sandbox environment by checking Internet connectivity, the existence of a virtual machine, etc. Sandbox analysis thus faces a serious problem in dealing with this evasive malware.

This dissertation first describes techniques performed by malware and malware authors for evading analysis and detection, and categorized evasion techniques against sandbox analysis, into two approaches: making comprehension of malware behaviors more difficult and detecting sandboxes. Then this study indicates a direction on how to develop a countermeasure technique against evasive malware without being evaded by an attacker – leveraging differences between malware and benign software that come from malware's mechanism for evading the analysis/detection mechanism; that is, when proposing a new analysis method, the method of detecting malware that evades the analysis method should be considered. Consequently, the attackers can be given fewer choices.

Chapter 4 proposes a novel sandbox analysis method that realizes better observability and efficiency against malware using techniques to make comprehension of malware behaviors more difficult. The method focuses on a function of malware that changes its behaviors in accordance with the behaviors of remote servers with which it interacts, such as C&C and malware download servers, and analyzes the server behaviors and corresponding malware behaviors. Experiments with

samples captured in the wild confirm that the method can observe more variety in their behaviors.

Chapter 5 clarifies targeted sandbox detection vulnerability in public malware sandbox analysis systems (public MSASs) for pursuing better observability. First, properties of sandbox information for decoy injection attack, in which an attacker detects the sandbox based on its sandbox information disclosed by submitting a decoy sample, are defined: stability, uniqueness, and stealthiness of collection. Then, 16 different kinds of characteristic information of the sandbox for its detection are analyzed in terms of those properties. Experiments with real public MSASs in operation confirm the broad applicability of the decoy injection attack as well as the need for comprehensive countermeasures.

Chapter 6 proposes a novel behavior-based malware-detection method using sandbox-evasive behaviors. Malware authors have been embedding functions that act as countermeasures against malware analysis and detection that often change runtime behaviors in each execution. The proposed method focuses on such characteristics. It conducts dynamic analysis on an executable file multiple times in the same sandbox environment to obtain multiple logs of API call and traffic, and then compares them to find the difference between the multiple executions. Experiments with malware samples captured in the wild and benign software samples confirm effectiveness of the method.

Table of Contents

List of Figures

List of Tables

Chapter 1

Introduction

1.1 Motivation and Contributions

As the Internet has become an increasingly essential medium in our life, security threats (e.g., divulging of private information, phishing, and denial-of-service [DoS] attacks) have also increased. In these threats, *malware*, a generic term for computer viruses, worms, Trojan horses, spyware, adware, and bots, plays a significant role, and extensive research efforts have been made to tackle it. The recent explosive increase of malware has made it nearly impossible to analyze all its forms manually using reverse engineering techniques. *Malware sandbox analysis* [1] [2] [3] [4] [5] [6] [7] [8] [9] [10] [11] [12] [13] [14] [15] [16] [17] [18] [19] [20] [21] [22] [23] [24] [25] [26] [27] has been studied widely as an effective countermeasure. Its basic idea is to actually execute a captured malware sample in a testing environment (*sandbox*) in order to observe and analyze its behaviors. A great advantage of sandbox analysis is that it can be implemented in a highly automated fashion.

Malware authors, however, have been making their malware more sophisticated in order to evade this analysis. For example, one type of malware, called a *time bomb*, does not activate until a certain date, and a bot changes its own behaviors in accordance with the behaviors of the remote servers with which it interacts; such as *Command and Control* (*C&C*) servers and malware download servers. Since these forms of malware do not work unless they meet the conditions for activation, it is difficult to sufficiently analyze them with traditional sandbox analysis. Another example is a type of malware that stops or changes its behaviors when it detects a sandbox environment by checking Internet connectivity, existence of a virtual machine, etc. Sandbox analysis thus faces a weighty problem in how to deal with such evasive malware.

This dissertation contains three main contributions toward solving the problem. First,

techniques performed by malware and malware authors for evading analysis and detection are described, and evasion techniques against sandbox analysis are categorized into two approaches: *making comprehension of malware behaviors more difficult* and *detecting sandboxes*.

An example of techniques to make comprehension of malware behaviors more difficult is making conditions for activation. A malware author often creates malware that conducts malicious activity only when the activation condition is met. A famous example is a *bot*, which changes its behaviors in accordance with the behaviors of remote servers with which it interacts, such as by conducting malicious activities in accordance with a C&C message received from a C&C server, downloading an additional binary from a malware download server, and running it. So in this case, when an activation condition is not met, malicious activity cannot be observed and sufficiently analyzed in sandbox analysis. Chapter 4 proposes a novel sandbox-analysis method that realizes better observability and efficiency against malware that changes its behaviors in accordance with the behaviors of remote servers with which it interacts, such as C&C servers and malware download servers. The method can efficiently analyze the server behaviors and corresponding malware behaviors.

There are some detection techniques based on specific sandbox characteristics [8] [28] [29] [30] [31] [32]. For example, the malware author tries to detect by using information unique to each sandbox, such as specific Dynamic Link Libraries (DLLs), system resources like the OS product key, and the global IP address. A previous study pointed out the vulnerability of public MSASs against *decoy injection attacks*, in which an attacker detects the sandbox based on its IP address disclosed by submitting a *decoy sample*. However, the possibility of detection using sandbox information other than the IP address was not discussed in detail. Chapter 5 clarifies targeted sandbox-detection vulnerability in *public Malware Sandbox Analysis Systems* (*public MSASs*) for pursuing better observability. First, properties of sandbox information for decoy injection attack are defined: *stability*, *uniqueness*, and *stealthiness of collection*. Then, 16 different kinds of characteristic information of the sandbox for its detection are analyzed in terms of those properties. Experiments with real public MSASs in operation confirm the broad applicability of the decoy injection attack as well as the need for comprehensive countermeasures

A direction is also indicated on how to develop a countermeasure technique against evasive malware without the attacker evading it – leveraging differences between malware and benign software that come from malware's mechanism for evading analysis/detection; that is, when proposing a new analysis method, it should be considered in advance how to detect malware that evades the method. This results in fewer choices for the attackers. As an example of the concept, Chapter 6 proposes a novel behavior-based malware-detection method using sandbox-evasive behaviors. Malware authors have recently been embedding functions that act as countermeasures against malware analysis and detection, so modern malware often changes its runtime behaviors in

each execution for this purpose. The proposed method herein focuses attention on such characteristics. It conducts dynamic analysis on an executable file multiple times in the same sandbox environment to obtain multiple logs of API call and traffic, and then compares them to find the *difference between the multiple executions*. If attackers try to evade the proposed detection method, they have to use deterministic malware, and then (especially dynamic) analysis is made easier. Experiments with malware samples captured in the wild and benign software samples confirm effectiveness of the method.

1.2 Organization

The rest of this dissertation is organized as follows. Chapter 2 presents the background and related works. Chapter 3 describes evasion techniques that malware authors use and indicates a direction for how to develop countermeasure techniques for evasive malware. Chapter 4 details the design and implementation of a novel sandbox analysis for improving observability against malware that changes behaviors in accordance with the responses from remote servers. Chapter 5 describes decoy injection attacks and clarifies vulnerability in public malware sandbox analysis systems against the attack. Chapter 6 discusses a novel behavior-based malware-detection method by using sandbox-evasive behavior. The dissertation concludes with Chapter 7.

Chapter 2

Background and Related Works

2.1 Malware

The term "*malware*" is derived from "malicious software," and is a generic expression for computer viruses, worms, Trojan houses, spyware, adware, bots, etc. Malware has a long history. In 1982, *Elk Cloner*, one of the first known computer viruses, appeared and *Brain*, the first computer virus targeting IBM PCs, appeared in 1986. In the late 1980s to early 1990s, the attackers' objective was personal pleasure or showing off, and this malware caused phenomena easily recognized by users, such as a screen display notifying of the infection. Since the late 1990s, however, the attackers' objective has clearly changed to monetary gain, and malware behaviors have grown increasingly stealthy and sophisticated so that the user cannot notice the infection. Malware has triggered most of the recent large-scale security incidents, such as divulging of private information, phishing, and denial of service (DoS) attacks.

An attacker uses a wide variety of methods to attempt to run malware on the target system. These methods can be categorized into two types: *exploiting human vulnerability* and *exploiting program vulnerability*. Since users with low computer security literacy tend to more carelessly download and install software publically available on the Internet, in the former methods an attacker has a user manually execute and install malware on a targeted computer under the pretense of it being useful software.

In contrast, more recent methods have the attacker attempting to exploit program vulnerability for automatically infecting a targeted machine with malware. This is because when a vulnerability is found in a program that runs on many computers, the attacker can exploit it to easily propagate malware to those computers. The program vulnerability often leads to a pandemic. An attacker tries

to exploit a wide variety of programs, such as operating systems, server applications, and client applications (e.g., web browsers). Attackers have long targeted operating systems and server applications. A well-known example, *Conficker* [33], found in 2008, exploits vulnerability in the Windows OS server service, and caused a pandemic. To exploit operating systems and server applications, malware needs to actively scan and identify a vulnerable computer. Since it basically does not know where the vulnerable computer exists, the scanning activity is often conducted widely on the Internet, and is often observed by researchers using network-monitoring techniques (e.g., *darknet monitoring*). On the other hand, to exploit client applications such as web browsers and browser plug-ins, first an attacker gives malicious web servers exploit code for propagating malware, and then tries to induce a user to the malicious servers via spam e-mail with malicious URLs or other techniques. In this type of malware infection, since an attacker only requires targeted and limited end-to-end communication without large-scale scanning, it is more difficult to observe and comprehend the actual conditions of malware propagation.

2.2 Collecting Malware Samples

In order to analyze malware, malware samples first need to be collected. Antivirus or security appliance vendors mainly collect malware samples from users' submissions and reports from antivirus software or security appliances.

Another way of collecting malware samples is to use a *honeypot*, which is a trap system used for detecting exploit codes and collecting malware samples installed using them. A honeypot looks like a simple vulnerable server/client to an attacker, but it is configured for blocking external attacks even if it is compromised. Honeypots can be classified based on their level of interaction with an attacker as *low-interaction* and *high-interaction*. A low-interaction honeypot emulates a vulnerable host and simplifies the detailed processing. Its advantage is that it requires lower construction and maintenance costs than a high-interaction honeypot, though it collects less information. *Nepenthes* [34] and *Honeyd* [35] are examples. A high-interaction honeypot uses an actual vulnerable host, and can collect more information. However, its performance is on the same level or less than that of an actual system, and it is at a high risk of being taken over by an attacker. *Honeynet* [36] is an example.

Conventional honeypots are designed for receiving remote exploit attacks from worm-type malware, and only wait for attacks via the Internet. These are called server honeypots. Recently, however, client-side attacks such as drive-by-download (DBD) attacks have been on the rise. Unlike remote exploit attacks, DBD attacks are triggered by users accessing a malicious website. A server honeypot therefore cannot observe client-side attacks. To detect and collect these attacks, a

client honeypot equipped with a client application that connects to remote servers and collects information on client-side attacks has been proposed. *HoneyMonkey* [37] and *MARIONETTE* [38] are examples.

2.3 Malware Analysis

Malware analysis is a vital approach to handling security incidents. Its goal is to provide sufficient information about functionalities and characteristics of malware in order to develop effective countermeasures. It can be categorized into two approaches: *static analysis* and *sandbox analysis* (*dynamic analysis*).

2.3.1 Static Analysis

Static analysis is called *white-box analysis* and is a traditional method for analyzing malware binaries without execution. It typically disassembles malware binaries and analyzes the functionality and characteristics of malware in detail at the assembly level.

The advantage of static analysis is that, in principle, it can reveal all the behavior information of malware included in malware binaries by analyzing all the disassembly code. However, malware authors often use packing tools such as *UPS*, *AS Pac*k, and *FSG* that obfuscate malware and hinder the analyst's understanding of the malware's intent. Thus, in order to conduct static analysis, an analyst must unpack the packed malware code in advance using techniques like memory dump.

The disadvantage of static analysis is that it is time-consuming and requires an analyst to have high-level network and programming skills. With the recent explosive increase of malware, it is becoming nearly impossible to analyze all its variations using static analysis by reverse engineering.

2.3.2 Sandbox Analysis

Sandbox analysis is called *black-box analysis* and is a method to analyze malware behaviors by executing a malware sample in a controlled environment (*sandbox*) and monitoring these behaviors, such as file activity, registry activity, and network activity. One advantage is that it is not disturbed by packing and code-obfuscation techniques, which malware developers often use to make static reverse engineering more time-consuming. Another advantage is that the sandbox analyzer can be implemented in a highly automated fashion. A disadvantage is that it can only observe and analyze

behaviors that are actually executed during the time of execution.

The previous studies of malware sandbox analysis can be categorized into two approaches in terms of their Internet connectivity: a *totally isolated sandbox* and a *sandbox with an Internet connection*.

Examples of the former approach include *Norman SandBox* [18], *NICTER Micro analysis System* [12] [13], and others [16] [17]. With this approach, malware can be safely analyzed with no impact on the Internet. Recently, however, since almost all malware communicates with remote servers such as C&C servers and malware download servers and its behaviors can be diverse depending on the behaviors of the remote servers, malware behaviors cannot be sufficiently observed with a totally isolated sandbox. These sandbox analysis systems have therefore emulated networks that provide malware samples with Internet services into the sandbox. Yet there is a limitation to this approach in that it is difficult to emulate remote hosts in the real Internet since malware make various different kinds of communications. Especially, when it interacts with a server such as a C&C or file server, it can use arbitrary (even customized) protocols for data transmission and authentication, which makes emulation increasingly challenging.

The other approach is to carefully connect the sandbox with the real Internet. Examples of this include *CWSandbox* [23] [27], *Anubis* [2] [8] [9], and others [4] [5] [24] [26]. In these systems, since a malware sample can communicate with the actual remote server on the Internet, the behaviors corresponding to the remote server's responses can be observed. However, there is also a risk that their attack may spill out of the sandbox and induce secondary infections, and the systems must carefully observe the outbound traffic from malware and filter out high-risk communication.

There are three important properties of malware sandbox analysis [24].

- *Observability*

Observability is a property in terms of observing the malware behaviors in consideration. Those writing removal tools or AV signatures may focus on internal behaviors such as changes of registry keys and creation and deletion of files. A network administrator may be interested in their network behaviors for writing an intrusion detection system (IDS) signature. In any case, the sandbox analysis should be able to provide the analyst with sufficient information.

- *Containment*

Containment represents two sub-properties: one for preventing the executed sample from attacking or infecting a remote host outside the sandbox (*containment of outgoing attacks*) and for suppressing a leakage of important information of the analysis system itself since that can be used against the system (i.e., sandbox detection). This is referred to as *containment of system*

information.

- *Efficiency*

  Efficiency is a property of constantly providing analysis results with sufficient information in a reasonable amount of time.

These three properties have trade-offs among each other and the means of simultaneously keeping all three properties at a high level is the design goal of malware sandbox analysis.

2.3.3 Malware Clustering

There is a great deal of malware that has similar functions in the wild. These reasons are as follows.

- Reuse of malware source code
- Change of compilers and compiler options
- Use of polymorphic malware
- Use of code-obfuscation techniques

In particular, as mentioned above, malware authors often use packers that obfuscate the malware and hide its characteristic code, and can easily generate a great deal of malware variants with different signatures (e.g., MD5 hash values). Malware-clustering methods that group malware variants according to their similarities have been widely studied as a means of efficiently analyzing a great deal of malware variants.

Malware-clustering methods can be classified into two approaches: *static-features-based* and *behavioral-features-based*. In the former approach [39] [40], in many cases unpacking is required for obtaining features from packed malware code. In the latter approach [5] [7], the major benefit of using behavioral features is that they are less susceptible to packing and other code obfuscation techniques. Yet there is a disadvantage of the behavioral-features-based approach in that only behavioral features of malware observed during the time of execution can be obtained.

2.4 Countermeasures Based on Malware Analysis

2.4.1 Generating Malware Signatures

An antivirus vendor collects and analyzes malware samples in order to extract characteristic strings and create signatures for detecting malware. There are some open source antivirus software packages such as *ClamAV* [41], and open source network-based IDSs such as *Snort* [42]. Using such software allows customizing of signatures, adding new signatures created by volunteers, and personally creating original signatures. As mentioned above, antivirus vendors mainly collect malware samples from user submissions, antivirus software reports, and honeypots. They can also collect samples from malware sample-sharing sites [43] [44].

Conventional signatures are created based on the characteristic binary code of malware and exploit codes. However, recently, since attackers often use packing techniques and polymorphic shellcodes for evading signature-based detection, security vendors have produced antivirus software and security appliances that have behavior-based detection engines and use the characteristic behaviors of malware and shellcodes as a signature.

2.4.2 Detecting C&C Traffic

Since the mid-2000s, a botnet, which consists of a large number of computers infected with bots, has posed a great threat to the Internet. Reports from security vendors exist on botnets consisting of more than one million computers.

The controller of a botnet, called a herder, can direct the malicious activities (e.g., sending spam email, performing distributed denial-of-service [DDoS] attacks) of those compromised computers through command and control (C&C) channels formed by standards-based network protocols such as Internet Relay Chat (IRC) and Hypertext Transfer Protocol (HTTP), or a customized protocol. Since a bot needs to frequently access and communicate with the remote host for receiving C&C messages, computers infected with one can be detected by detecting the C&C traffic.

Conventional botnets (e.g., *Agobot*, *SDbot*, *IRCbot*) often used the IRC protocol as a C&C channel, so there are many studies on detecting IRC-based C&C traffic [45]. And because the use of benign IRC communication has decreased, IRC-based C&C communication can now be detected

more easily. Accordingly, attackers came to adopt not the IRC protocol but more popular protocols such as HTTP, P2P, or original protocols as a C&C in order to evade detection.

In particular, when a botnet uses an HTTP protocol as a C&C, since malware accesses not only HTTP-based C&C servers but also benign web servers for checking Internet connectivity, all web servers accessed by malware in the sandbox analysis cannot be regarded as C&C servers, and a discernment must be made between HTTP-based C&C traffic and benign HTTP traffic.

2.4.3 Blacklisting

IP addresses, domain names, and uniformed resource locators (URLs) of malicious hosts (e.g., C&C servers, malware download sites) can be obtained by analyzing malware samples. This information is extremely useful as a blacklist for filtering access to malicious hosts and detecting infected hosts. There is also DNS-level filtering such as DNSBL and DNS sinkholes.

In addition, recently, the most common way of infecting vulnerable computers with malware is to present them with a web page that performs a DBD attack. Thus, there are many efforts to interrupt access to malicious websites by using a blacklist for protecting users from DBD attacks. Google, for instance, provides *a safe browsing service* [46] that enables applications to check URLs against Google's continually updated lists of malicious websites. Some web browsers, such as Mozilla Firefox and Google Chrome, use so-called safe browsing and display a warning message if a user tries to access a URL contained in the blacklist. A similar filtering system called *SmartScreen Filter* [47] is implemented in Microsoft Internet Explorer.

Although filtering systems based on a blacklist help in making users safer, they suffer from a number of limitations. For example, attackers frequently change domain names and URLs of malicious sites, which makes them difficult to keep up to date with. Attackers also use cloaking techniques for evading inspection by client honeypots.

2.4.4 Botnet Takedown

The most effective countermeasure against a botnet is to shut down the C&C communication. If this is possible, the botnet can be destroyed and potential victims can be prevented from being affected by cyber-attacks from it. Legal and technical action needed against botnets requires collaboration between technology companies and law enforcement. Some successful cases of botnet takedowns are *Waledac* [48], *Rustock* [49], and *ZeroAccess* [50].

Chapter 3

Analysis and Detection vs. Evasion

As malware analysis and detection techniques become known, malware authors have begun to utilize anti-analysis techniques for evading analysis and detection. This chapter describes evasion techniques that malware authors use and indicates a direction on how to develop a countermeasure technique against evasive malware.

3.1 Evading Static Analysis

Malware authors try to make static analysis more time-consuming. A typical example is code obfuscation by using a packer. Recently, there are many sophisticated packers that can conduct multilayer packing and debugger detection. And as mentioned in Section 2.3.3, a malware author can create many malware variants more easily by reusing malware source code, changing compilers and compiler options, and developing malware creation tools. Thus, with the recent explosive increase of malware, it is becoming nearly impossible to analyze it all by static analysis with reverse engineering. To address this problem, many studies has examined automatic unpacking [51] [52] and malware clustering.

Another example of evading static analysis is a type of malware called a *downloader* that connects to a remote site to download an additional program as necessary and runs it on an infected computer. In such a case, it is difficult to acquire information about an additional program from a downloader by conducting static analysis.

3.2 Evading Sandbox Analysis

Techniques to evade sandbox analysis are classified into two approaches: *making comprehension of malware behaviors more difficult* and *detecting sandboxes*.

An example of techniques to make comprehension of malware behaviors more difficult is making conditions for activation. A malware author often creates malware that conducts malicious activity only when the activation condition is met. So in this case, when an activation condition is not met, malicious activity cannot be observed and sufficiently analyzed in sandbox analysis. There are many types of conditions. A typical example is a time condition, and a Trojan house that activates on a certain date is often called *time bomb*. There is also malware that activates in a certain period of execution. Another example is a *bot*, which changes its behaviors in accordance with the behaviors of remote servers with which it interacts, such as by conducting malicious activities (e.g., sending spam email, performing distributed denial of service attacks) in accordance with a C&C message received from a C&C server, downloading an additional binary from a malware download server, and running it. Many analysis techniques have been studied for dealing with malware with activation conditions, such as disclosing activation conditions [53] and exploring multiple execution paths of malware [17]. However, these techniques cannot reveal additional functionalities of malware that are added by the interaction with the remote servers.

Another example is *randomizing behaviors*. Some malware changes its runtime behaviors in each execution to evade analysis and detection. The most well-known example is Conficker, which has been a form of pandemic malware since 2009. It generates a to-be-accessed domain name by using a pseudorandom number generator, making it difficult to list all accessed domain names of malicious servers. Malware authors sometimes also create *kernel mode malware*, commonly known as a rootkit, which runs in the OS kernel with absolute rights to system resources. Its behaviors cannot be observed and analyzed using standard analysis techniques for user mode malware.

Techniques to detect a sandbox can be categorized into two approaches: *based on characteristics that almost all sandboxes have in common* and *based on a specific sandbox's characteristics*.

An example of the former is *debugger detection*, a traditional technique to evade analysis, and many types of malware have a debugger-detection function (e.g., *Rbot* [54], *Sdbot* [55], *IRCBot* [56]). There are many techniques for detecting the presence of a debugger, such as using a Win32 API called *IsDebuggerPresent*, which will return a Boolean true if the program is being debugged, searching system resources of well-known debuggers, and monitoring the system clock to confirm whether too much time has elapsed between instructions. Malware also often checks for the presence of monitoring mechanisms or tools like a process monitor or API hook in order to detect a

sandbox. Therefore, there are some studies on evading debugger detection [57]. In addition, because a malware sample is actually executed in a sandbox, the sandbox environment must be recovered after analysis. Thus, virtual machine monitors (VMMs) are often used for developing a sandbox analysis system in order to make recovery easier. However, there are not so many cases of home-use virtual machines. Consequently, some types of malware detect the use of a virtual machine environment, and change or stop their behaviors if they are running on a virtual machine. There are many ways to detect virtual machine environments [58] [59] [60] [61], such as using specific processes/files/directories/registry keys, using specific information about virtualized hardware (e.g., MAC addresses on NICs, USB controller type, SCSI device type), and using specific processor instructions and capabilities. An example of these types of malware is *Agobot* [62], which has functions to detect the presence of well-known virtual machine monitors such as *VMware* [63] and *Virtual PC* [64]. When using a sandbox with Internet connectivity, since there is also a risk of that attack from malware spilling out of the sandbox and inducing secondary infections, the outbound traffic from malware must be carefully observed and high-risk communication filtered out. There is a sandbox-detection method that checks for the presence of filtering mechanism for outgoing traffic [65].

There are some detection techniques based on specific sandbox characteristics [8] [28] [29] [30] [31] [32]. For example, the malware author tries to detect by using information unique to each sandbox, such as specific *Dynamic Link Libraries* (*DLLs*), system resources like the OS product key, and the global IP address. Previous works [31] [32] have pointed out a vulnerability of a certain type of malware sandbox systems against detection based on the IP address. It was reported that in the real world this type of attack had been conducted against several sandbox systems [66].

3.3 Evading Detection Techniques

Antivirus software and personal firewalls are used in Windows systems as a basic protection method against malware. Therefore, there is malware that tries to evade detection by searching the processes of security software and kills them after intruding on a computer.

Since antivirus software often uses a signature-based detection engine, malware authors use techniques for evading signature-based detection. For example, when using a binary in a file as a signature, a malware author tries to conceal the characteristic binaries by using runtime packers, and when using a file/directory name or registry entry name as a signature, malware changes its runtime behaviors in each execution to evade detection. Malware also uses a polymorphic shellcode for evading signature-based detection in IDSs. Polymorphic shellcode engines create different forms of the same initial shellcode by encrypting the payload with a different random key each time,

and by prepending to it a decryption routine that makes it self-decrypting. To evade detection methods based on lists of malicious IP addresses/domains/URLs, malware also uses evasion techniques like a *domain generation algorithm* (*DGA*) and *fast flux* [67].

For a DBD attack, the lifetime of a malicious URL is short because attackers frequently changes URLs of malicious websites in order to evade blacklist-based detection. Attackers also use cloaking techniques such as delivering contents based on the IP addresses or the User-Agent HTTP header of the user requesting the page for evading inspection by client honeypots. That is, if an accessed user is identified as a crawler/client honeypot of security vendors/researchers, a server delivers benign dummy contents for deceiving the crawler/client honeypot.

3.4 Direction: Leveraging Evasive Behaviors for Countermeasures

Since malware emerged, security researchers have been engaged in a type of arms race against it. There has been repeated development of analysis/detection methods against sophisticated malware by security researchers, and attackers have responded by developing techniques to evade the methods. There is also the dilemma of academic researchers needing to publish their research, which can lead to attackers subsequently evading the published analysis/detection methods. Therefore it is necessary to consider how to develop/publish countermeasure techniques against evasive malware without attackers evading them.

We pay attention to the many cases in which malware has characteristic behaviors that differ from benign software's behaviors and come from the malware's mechanisms for evading analysis/detection. For example, there is malware that detects a particular malware sandbox and terminates (*environment-sensitive malware* [6]). However, no benign software is known to do so, and environment-sensitive programs can be regarded as malware. When using the detection method, if attackers try to evade the detection they must stop using environment-sensitivity, upon which malware analysis is made easier. Another example is malware that detects and kills the antivirus process when executed. However, no benign software is known to do so, and such types of programs can be regarded as malware [68]. When using the detection method, if attackers try to evade detection, they must stop killing the antivirus process, upon which protection is made easier.

Consequently, we indicate a direction on how to develop/publish countermeasure techniques against evasive malware without the attacker evading them. Differences between malware and benign software that come from malware's mechanism of evading analysis/detection should be leveraged; that is, when proposing a new analysis method, it should be considered in advance how

to detect malware that evades the analysis method. The attackers are then given fewer choices.

Chapter 4

Sandbox Analysis Utilizing Dummy Client

4.1 Introduction

As mentioned in the above section, sandbox analysis is an effective solution for analyzing a large amount of malware. However, there are several important issues to be addressed in malware sandbox analysis. One is that it can observe only a single execution path of the malware sample by each execution and important behavior that can be crucial for developing an efficient countermeasure may not be observed. In particular, because recent malware communicates with remote hosts on the Internet for receiving C&C commands, updating themselves, etc., its behaviors can be diverse depending on remote hosts' behaviors. In fact, a previous study [26] reports that the malware behaviors observed by malware sandbox analysis can differ greatly when the analyses are performed in two different time periods because some of the remote servers that the analyzed samples communicated with changed their behaviors over time. Therefore, in malware sandbox analysis, it is important not only to focus on behaviors of the malware sample itself but also those of the remote servers that attackers control. A simple solution to achieve this is observing the live sample with an Internet-connected sandbox for a long period of time. However, since it is not known when these servers will send meaningful responses, the sample being executed needs to be kept in the sandbox, which is a costly operation. Additionally, leaving the live malware in the Internet-connected sandbox increases the risk of its attacks spilling out of the sandbox and inducing secondary infections.

In this chapter, we propose a novel sandbox analysis method utilizing a *dummy client*, an

automatically generated lightweight script to interact with the remote servers instead of the actual sample. In the proposed method, first a malware sample is executed in the sandbox connected to the real Internet and an Internet emulator. Then the traffic observed in the sandbox is inspected and high-risk communications are filtered out. The dummy client then uses the rest of the traffic to interact with the remote servers instead of the sample itself, and effectively collects the responses from the servers. The collected server responses are then fed back to the Internet emulator in the sandbox and will be utilized for improving observability of malware sandbox analysis. For example, the next time a malware sample is analyzed and the remote servers cannot be connected, the dummy server can emulate the remote servers by sending the collected responses. Another example is when a new response from the remote servers is observed, the malware sample that accesses the corresponding server can be re-analyzed and the responses fed back to the sample in order to observe its corresponding reactions.

The advantage of the proposed method is that by utilizing lightweight dummy clients instead of observing the interactions by live malware itself, observability of malware behaviors can be increased by continuously monitoring many remote servers in parallel while not too greatly decreasing the efficiency. Apart from that, since the malware traffic can be closely inspected and potentially dangerous traffic filtered out before replaying it with the dummy client, containment of the outgoing attacks is also expected to improve.

We evaluate the proposed method using samples captured in the wild. By using a low-risk containment policy of emulating only harmless HTTP and IRC connections, we confirm that, in comparison with the simple Internet-connected sandbox, the method can improve observability of malware sandbox analysis, and it revealed more malware behaviors.

The rest of this chapter is organized as follows. Section 4.2 gives related works. Section 4.3 explains the proposed method and Section 4.4 its implementation. Section 4.5 covers the experiments for evaluation of the proposed method. Section 4.6 discusses the challenges of the proposed method and Section 4.7 summarizes the chapter.

4.2 Related Works

As mentioned Chapter 2, there are many studies on sandbox analysis, but they mainly focus on how to observe behaviors of a malware sample itself and therefore do not deeply discuss the issue of how the sample is influenced by the variety of responses from the remote servers such as C&C and malware download servers controlled by attackers. When viewed from this perspective, analysis with a totally isolated sandbox is not desirable because it cannot observe the behavior of the remote servers on the Internet. In contrast, analysis with an Internet-connected sandbox can observe how

malware samples interact and are influenced by the remote servers. However, since it is not known when these servers send meaningful responses to the sample executed in the sandbox, it is necessary to continue executing it to observe the responses. Other related technology with a similar goal is to explore multiple execution paths. In a previous study [17], multiple execution paths of the malware sample can be observed by controlling the conditional branches. However, even exploring multiple execution paths cannot reveal additional functionalities of malware that the interaction with the remote servers adds. We propose a sandbox analysis utilizing a dummy client for solving this problem.

Some related works also use a dummy client. *Caballero et al.* [69] performed a measurement study of the *pay-per-install* (*PPI*) market by infiltrating four PPI services to gather and classify the resulting malware executable files the services distribute. They built and used a dummy client called a *milker* to "milk" programs that the PPI services distributed. Their approach leverages techniques for automatic binary reuse [70] [71] that from an executable extract a specific function defined by an analyst. *Cho et al*. [72] proposed a technique to extrapolate complete protocol state machines and applied it to analysis of botnet C&C protocols. They built a bot emulator that interacts with the C&C servers, reverse-engineered the message formats and their semantic content using automatic protocol reverse engineering [70], and extracted encryption/decryption modules from the bot binary [73]. In comparison with above related works, the advantage of the proposed method is that the method is simple and easy to automate.

4.3 Proposed Method

This section explains the proposal of a *sandbox analysis method utilizing a dummy client*. In the method, first, a malware sample is executed in the sandbox connected to the real Internet and Internet emulator. The emulator consists of numerous dummy servers and hosts with emulated/real vulnerable services. These are called a *Honeypot in the Sandbox* (*HitS*). Then the traffic observed in the sandbox is inspected and high-risk communications such as port scanning, remote exploitation, and DoS are filtered out. The dummy client then uses the rest of the traffic data to interact with the remote servers. Then the dummy client continually interacts with the remote servers instead of the sample itself and effectively collects the responses from the servers. The accumulated server responses are fed back to the Internet emulator in the sandbox and will be utilized for improving observability of malware sandbox analysis. For example, the next time a malware sample is analyzed and the remote servers cannot be connected, the dummy server can emulate the remote servers by sending the collected responses. As another example, when a new response from the remote servers is observed, the malware sample that accesses the corresponding server can be

re-analyzed and the responses fed back to the sample in order to observe its corresponding reactions. The advantage of the proposed method is that observability of malware behaviors can be increased by continuously monitoring many remote servers in parallel while not too greatly decreasing the efficiency by utilizing lightweight dummy clients instead of observing the interactions by live malware itself. Apart from that, since the malware traffic can be closely inspected and potentially dangerous traffic filtered out before replaying it with the dummy client, containment of outgoing attacks is also expected to improve.

First, an overview of the proposed sandbox system is given in Figure 4.1. In the figure, solid arrows indicate the communications by the analyzed malware sample and the dummy client, and dotted arrows indicate communications by the sandbox system for its operation. The proposed sandbox consists of two units: a *sandbox unit* and *dummy client unit*. The sandbox unit is where the malware sample is actually executed and analyzed. It consists of five components: *victim host*, *Internet emulator*, *access controller*, *analysis manager*, and *data spool*. The dummy client unit is where the dummy clients are generated and executed. This unit contains two components: a *dummy client generator* and *dummy client*. Each component is described in Section 4.3.1. The implementation of each component is also described in Section 4.4.1.

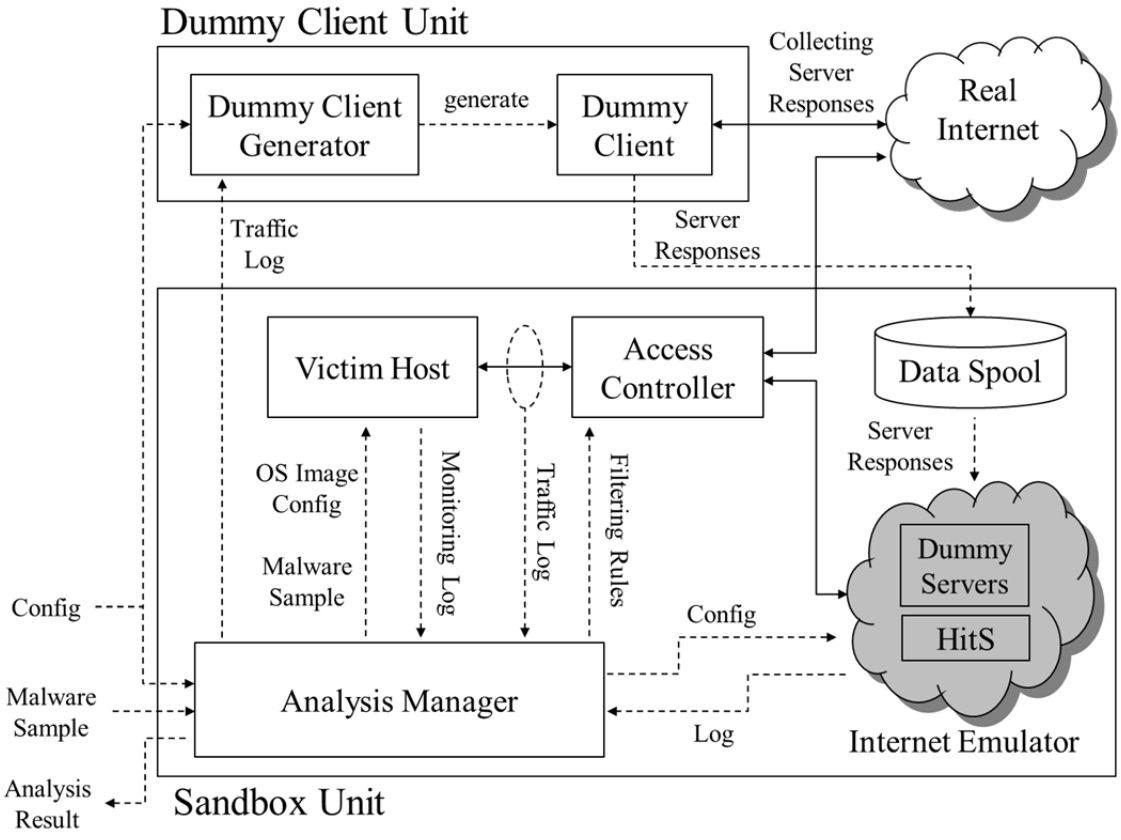

Figure 4.1 Overview of Proposed System

4.3.1 Components

The proposed system consists of seven components, as follows.

- *Victim Host*

  The victim host is a host on which a malware sample is first executed to be observed. It is important that the security of the victim host is properly configured so that the executed malware performs further actions for us to observe.

- *Access Controller*

  The access controller controls the traffic from the victim host. It receives all packets from the victim host and redirects it to either the Internet emulator or the real Internet in accordance with the filtering rules. The analysis manager generates the filtering rules.

- *Internet Emulator*

  The Internet emulator provides various network services to the victim host. It consists of dummy servers such as HTTP, SMTP, FTP, NTP, IRC, DNS, and feedback servers. A *feedback server* emulates behaviors of remote servers with which malware interacts by sending the collected server responses. Apart from those servers, it also deploys hosts with unpatched vulnerable services, HitS. Suspicious traffic from the malware sample can be tested with HitS to see if it actually compromises it.

- *Analysis Manager*

  The analysis manager is the core component that manages the full analysis procedures. Based on a simple *config file*, it loads and refreshes an OS image of the victim host, boots up and shuts down the victim host, executes malware sample in the victim host, receives and inspects all traffic logs and internal logs, sends all traffic logs to the dummy client generator, and finally outputs the analysis results to the analyst.

- *Dummy Client Generator*

  The dummy client generator receives traffic logs from the analysis manager, inspects them, and generates a dummy client.

- *Dummy Client*

  The dummy client generated by the dummy client generator is executed in the environments with the real Internet connection and emulates malware communications. When the dummy client

receives server responses, those responses are stored in the data spool.

- *Data Spool*

  The data spool is the component that stores server responses collected by the dummy clients. The Internet emulator loads the stored server responses and sends them back to the malware executed in the victim host from dummy servers.

4.3.2 Analysis Procedure

The following is, in brief, the procedure of the proposed sandbox analysis.

(1) The analyst inputs a malware sample and the configurations (OS image of the victim host, filtering rules for access controller, etc.) to the system.

(2) The analysis manager reflects the configurations to the access controller and Internet emulator and boots up the victim host.

(3) The victim host executes the malware sample. All traffic from the victim host is first sent to the access controller, and the access controller redirects the traffic to either the real Internet or Internet emulator in accordance with the filtering rules.

(4) After a certain instructed time has elapsed, the victim host sends all traffic logs and internal monitoring logs to the analysis manager. The Internet emulator also sends its logs to the analysis manager.

(5) The analysis manager receives all logs from the victim host and the Internet emulator, shuts down and refreshes the victim host, and sends traffic logs to the dummy client generator.

(6) The dummy client generator inspects the received traffic logs to eliminate high-risk communications, such as port scanning and remote exploitation, and generates the dummy client that emulates the remaining communications. Section 4.4.2 describes how to generate the dummy client.

(7) The dummy client is executed in the environment with the real Internet connection. It repeats malware communication with the remote servers on the Internet, continues collecting server responses, and stores them in the data spool.

4.4 Implementation

Figure 4.2 gives an overview of the implementation of the proposed sandbox analysis system. Solid arrows indicate the communication by the analyzed malware and dotted arrows indicate communication by the sandbox system for its operation.

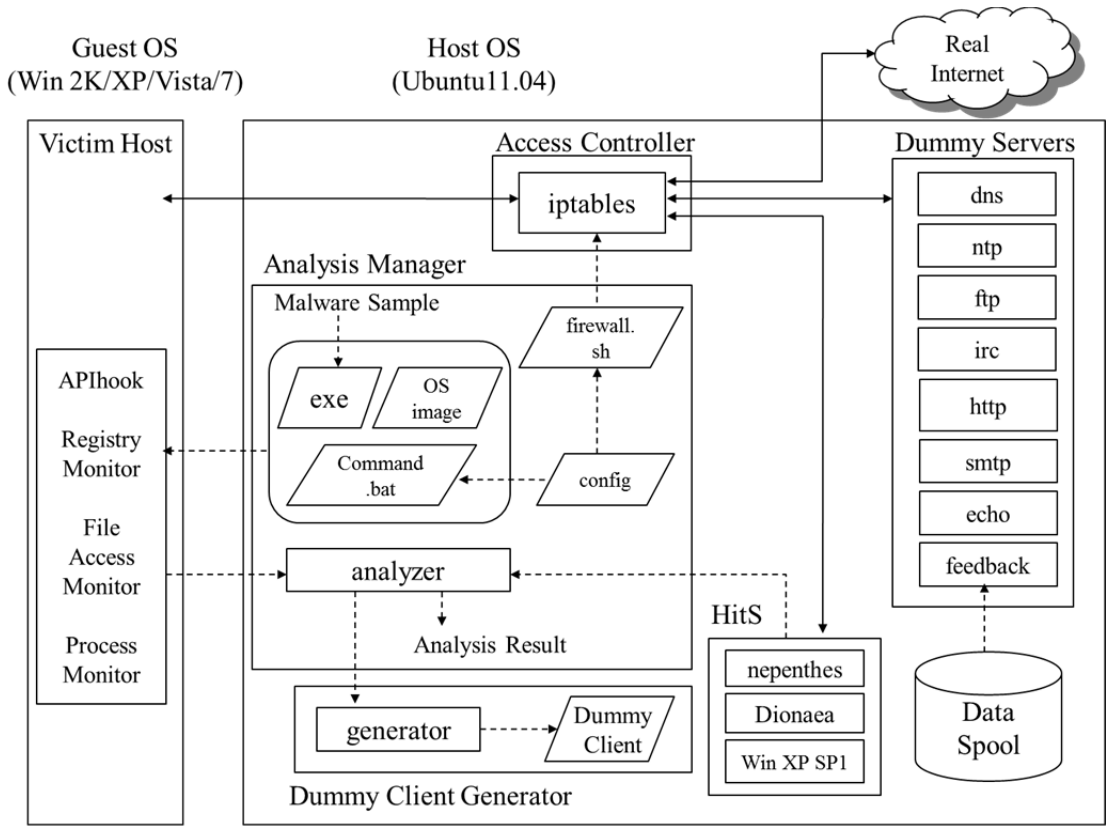

Figure 4.2 Implementation of Proposed System

4.4.1 Components

The entire system was implemented in a single real machine, except for the environment for the dummy client to be executed. A virtual machine by VMware Player 3.1.4 running Windows XP Professional SP1 was used as the victim host. The host OS is Ubuntu 11.04, on which the analysis manager, access controller, Internet emulator, and dummy client generator are implemented. The network between the victim host and host OS is realized by a virtual private network provided by VMware Player. Each component is implemented as follows.

- *Victim Host*

The victim host is implemented as a virtual machine running Windows XP SP1. To avoid VMware detection, the default port number of the VMware server's console and MAC address of the virtual NIC were changed. Basic monitoring tools such as Regmon and Filemon [74] are currently installed in the victim host. Other monitoring tools such as InCtrl [75] or techniques like API hooking [76] can also be deployed. The victim host is also configured to automatically download and execute a Windows batch file command.bat from the analysis manager upon each boot-up using SSH. The batch file contains the further instructions to be followed by the victim host.

(1) Download the malware sample
(2) Start designated monitoring tools
(3) Execute the sample
(4) Send the monitoring results to the manager after a designated time period

Since the analysis manager can modify the command.bat file, the manager can easily control the procedure of the victim host remotely.

- *Internet Emulator*

The Internet emulator consists of two subcomponents: dummy servers and HitS. Both of these run on the system's host OS. The dummy DNS, IRC, HTTP, HTTPS, NTP, SMTP, FTP, ECHO, and feedback servers are implemented as lightweight simplified server scripts using Perl to emulate the network services on the real Internet. When A-record query is received from a malware, the DNS server checks whether the responses from the server of the queried domain name have been stored in the data spool. If the responses exist, the dummy DNS server returns the IP address from the specific range. Otherwise, the dummy DNS server works as the DNS proxy. In addition, IP addresses that the dummy DNS server returns are unique during the same analysis pass. The feedback server is the server to send the collected server responses to the malware. When the malware attempts to access the remote servers, the feedback server obtains the server's domain name by issuing a DNS query for the PTR record, and checks whether the server can be accessed. If the server can be accessed, the feedback server works as the proxy server. Otherwise, it searches the server responses corresponding to the domain name, received data, and collected time in the data spool. The feedback server has two response modes: latest response mode and unique response mode. In the latest response mode, it searches the latest collected server response and sends it to the malware. It can also send the response collected at a specific time. The latest malware behaviors can be analyzed by using the latest response mode. How the malware works at the specific time can

also be examined. In the unique response mode, for the HTTP protocol the feedback server searches and lists the unique responses and sends them to the malware by rotation, and for the IRC protocol the feedback server searches the latest collected response and unique PRIVMSGs, and sends them to the malware. The malware's various behaviors corresponding to the responses can be efficiently analyzed using the unique response mode. And for the IRC protocol the feedback server replaces the nickname appropriately.

HitS is dedicated to inspecting the connections initiated by the sample to see if they contain any harmful attacks. In order to do this, HitS is designed to run emulated/real vulnerable network services for the sample to exploit, like a honeypot. The current implementation utilizes *low-interaction honeypot* programs *Nepenthes v0.2.2* [34] and *Dionaea* [77] since it can emulate multiple vulnerable services. A virtual/real machine running a full vulnerable OS to detect zero-day exploits can also be deployed, though such an implementation is a subject of future work.

- *Access Controller*

The access controller is implemented by iptables [78], a packet-filtering application. Before analysis, the analysis manager generates and executes a shell script, called firewall.sh, to apply newly generated filtering rules. All traffic from the victim host is redirected to either the real Internet or Internet emulator in accordance with the filtering rules. For the connections to the Internet emulator, their destination IP addresses are changed to those of the host OS by the REDIRECT target of a PREROUTING chain in iptables so that they are sent to the servers running on the host.

- *Analysis Manager*

The analysis manager loads the configuration, and based on that, changes the configuration of the Internet emulator and generates a Windows batch file *command.bat* and a shell script *firewall.sh*.

- *Dummy Client Generator*

The dummy client generator inspects the traffic logs and removes high-risk communications, and generates the dummy client that replays the communications. There are several choices of how to replay. For example, a dummy client can be generated for each malware sample to emulate its communications to remote servers. In this way, the server responses corresponding to each type of malware can be collected. However, when the different types of malware access the same server with the same query, redundant data are saved in the data spool, and the quantity of data increases. Therefore, for efficiency, a method was implemented to generate a single dummy client that emulates all unique communications observed by the execution of all malware samples.

Section 4.4.2 describes the process of the dummy client generator.

4.4.2 Generating Dummy Client

Generating a dummy client that emulates communications observed by malware sandbox analysis is performed as follows.

1) The dummy client generator divides the traffic logs into all TCP and UDP sessions. A session is a series of packets exchanged between a port of the victim host and a port of a remote host in the analysis.

2) Each reconstructed session is closely inspected in the following rules to eliminate high-risk communications. These are called *attack-filtering rules*.
   i. Do not allow any session whose source IP address is spoofed.
   ii. Do not allow any session that the inspection determines is a port scan. Likewise, do not allow any session that the inspection determines is part of a DoS attack. In order to detect a DoS attack, it counts the number of sessions the victim host initiates for the same destination IP address and port. If over a threshold number Th_{DoS} of sessions are initiated, it is considered a DoS attack. In order to detect a port scan, it counts the number of unique IP addresses the victim host accesses on each destination port without DNS name resolution. A port is considered scanned if the victim host accesses over a threshold number Th_{ps} of distinct IP addresses on that port.
   iii. Do not allow any session that caused a successful exploitation of the vulnerabilities in HitS.
   iv. Do not allow any session whose application protocol is not authorized. Recognition of the application protocol is based on the message flow analysis. Namely, a check is performed as to whether certain messages that characterize the application protocol, such as methods in HTTP and commands in IRC, are transmitted in the legitimate order of the protocol.

3) For each session that passed the above-mentioned rules, the dummy client generator extracts the payload. The extracted payload is appended with the information on the destination (e.g., domain name or destination IP address, destination port, protocol) and stored in the host OS.

4) Each session is inspected in the following rules to increase efficiency. These are called

duplication-reduction rules.

i. Leave only one session if there are two or more sessions that send the same payload to the same server.

ii. Delete a session whose payload is included in another session.

iii. Leave only one session if there are two or more sessions that join the same IRC server with different IRC nicknames. The reason for utilizing this filtering rule is that malware often communicates with an IRC server by a random nickname, and if each session is emulated, the number of emulated sessions will greatly increase.

iv. Leave only one session if there are two or more sessions that send the same HTTP GET queries with different arguments to the same server. This filtering rule is utilized because malware sends the information about the infected host, the own ID, etc. by using arguments of a HTTP GET query, and if each session is emulated, the number of emulated sessions will greatly increase.

5) The dummy client generator generates the dummy client that replays the remaining communications. It emulates the communication with remote servers via the following procedures.

i. The dummy client loads the destination information (i.e., domain name or destination IP address, destination port, protocol) of each session.

ii. If the destination information includes a domain name, the dummy client resolves it and obtains the corresponding destination IP address.

iii. If the protocol is TCP, the dummy client tries to connect to the destination IP address on the destination port via a three-way handshake. When the connection is established, the dummy client sends the first packet to the destination. If the protocol is UDP, it sends the first packet to the destination IP address on the destination port.

iv. The dummy client waits for a response from the destination. When it receives the response, it stores it in the data spool.

v. If the packet to be sent to the destination is remaining and no response for the previous packet has arrived within a certain period of time the dummy client sends the next packet to the destination.

vi. After sending all packets, the dummy client waits for the response from the server, unless the server has closed the session, and stores the received response in the data spool. For the IRC protocol, the dummy client also automatically replies with a PONG command when receiving a PING command from the IRC server for that not to be closed session by the server.

4.5 Evaluation

Experiments were conducted to evaluate the proposed method using 7,184 malware samples captured in the wild by low-interaction honeypots (Nepenthes and Dionaea), high-interaction honeypots, client honeypots, and others. A total of 557 names of samples were obtained from Symantec, and these were randomly divided into two sets. Then, one was used one for creating the dummy client and the other for comparison. The former set is called the *training set* and the latter the *test set*. Table 4.1 is a list of the top 20 virus names of samples in each set.

Table 4.1 Top 20 Names of Samples in Each Set Obtained from Symantec

(a) training set

Malware Name	# of samples
W32.Virut.W	571
W32.Spybot.Worm	347
Unknown	320
W32.Virut.B	281
Trojan Horse	189
W32.Sality.AE	147
W32.Pinfi	139
W32.IRCBot	127
W32.Korgo.S	125
W32.Rahack.W	125
W32.Virut.U	93
W32.Gobot.A	89
W32.Korgo.V	85
Hacktool	80
Backdoor.Trojan	54
W32.Rahack.H	51
Trojan.Gen	42
Backdoor.Graybird	38
Suspicious.IRCBot	32
W32.IRCBot.Gen	31
Others	626

(b) test set

Malware Name	# of samples
W32.Virut.W	597
W32.Spybot.Worm	321
Unknown	292
W32.Virut.B	271
W32.IRCBot	170
Trojan Horse	153
W32.Sality.AE	143
W32.Pinfi	130
W32.Rahack.W	127
W32.Korgo.S	120
W32.Virut.U	104
W32.Korgo.V	90
Hacktool	76
W32.Gobot.A	72
W32.Rahack.H	63
Backdoor.Trojan	62
Trojan.Gen	48
W32.Korgo.W	36
Suspicious.IRCBot	35
W32.IRCBot.Gen	32
Others	650

The objective of the experiment is twofold. First, whether the behaviors of remote servers can be observed by using the dummy client is confirmed. Then, the possibility of improving the observability of the proposed sandbox as opposed to a simple Internet-connected sandbox is confirmed.

4.5.1 Procedure

In order to evaluate the proposed method, two sandboxes were prepared for comparison: the *proposed sandbox with accumulated server responses collected by dummy clients* and an *Internet-connected sandbox*. The former is called the proposed sandbox and the latter the Internet-connected sandbox. Table 4.2 shows the configuration of the proposed sandbox. Each sandbox is implemented on a single real machine with the same specifications: Intel XEON 2.66 GHz × 4 with main memory of 4 GB RAM, and the Internet-connected sandbox was implemented by removing the dummy client unit and feedback server in the proposed sandbox.

The analysis by the proposed sandbox was performed as follows.

(1) *First Analysis* – First, the training set in the Internet-connected sandbox was analyzed on August 5-9, 2011.

(2) *Collection of Server Responses* – Then, a dummy client was generated with the observed traffic logs, and executed to collect server responses. The dummy client accessed each of the remote servers every hour from September 6 to October 25, 2011.

(3) *Second Analysis* – Finally, the test set was analyzed in the proposed sandbox and in the Internet-connected sandbox on November 10-15, 2011.

Table 4.2 Configuration of Proposed Sandbox

EXECUTION_TIME	60 [sec]
VICTIM_HOST_OS	Windows XP Professional SP1
FILEMON	OFF
REGMON	OFF
HitS	Nepenthes v0.2.2
Dummy Servers	HTTP, NTP, IRC, FTP, SMTP, DNS, ECHO, Feedback
Threshholds	$Th_{ps} = 50$, $Th_{DoS} = 50$
Authorized Application	HTTP, IRC
Execution interval of the dummy client	1 [hour]

4.5.2 Results

(1) First Analysis

Of the 3,592 samples, 2,322 attempted to connect to remote hosts with 4,848,791 sessions

consisting of 4,848,328 TCP sessions and 463 UDP sessions (except for DNS queries). Among them, 4,846,095 sessions (99.94%) were filtered out using attack-filtering rules. Most of the filtered sessions were considered as port scans. The remaining 2,696 sessions were again reduced to 242 by duplication-reduction rules. Most of the session reduction was due to duplicated IRC sessions. Likewise, the number of destinations to which the samples attempted to connect was 4,617,688. Here, a destination means either a destination IP address or a domain name of the destination host. The number of destinations was reduced to 148 using attack-filtering rules. Ultimately, 242 different types of sessions, consisting of 224 HTTP sessions and 18 IRC sessions to 148 distinct destinations, were selected for replay by the dummy client to obtain server responses.

(2) Collection of Server Responses

The dummy client was run from September 6 to October 25, 2011 to collect responses from the servers. The client accessed the 148 destinations by replaying the 242 different types of sessions every hour. Results are as follows.

A) For 70 types of replayed sessions, the client always received the same response with the same content every hour.
B) For 23 types of replayed sessions, the client received no response during the experiment.
C) For 37 types of replayed sessions, the client received either a response with the same content or no response.
D) For 112 types of replayed sessions, responses from the server changed during the experiment.

In case C, connectibility of the servers changes depending on when the client accesses the servers. In case D, responses from the servers change depending on when the client accesses the servers. Thus, analysis time may influence the result of normal sandbox analysis of malware that attempts to access the remote servers with these sessions. Here, the change of the response means that the hash value of the received HTTP content is changed or the IRC PRIVMSG is changed.

Windows Executable Files

Figure 4.3 shows the number of new Windows executable files the dummy client collects over time. Only new executables are counted based on their hash values. From the figure, it can be confirmed that the dummy client can continuously receive a number of new Windows executable files. Since downloaded executables can be used for further malicious activity, it is important to observe such behavior.

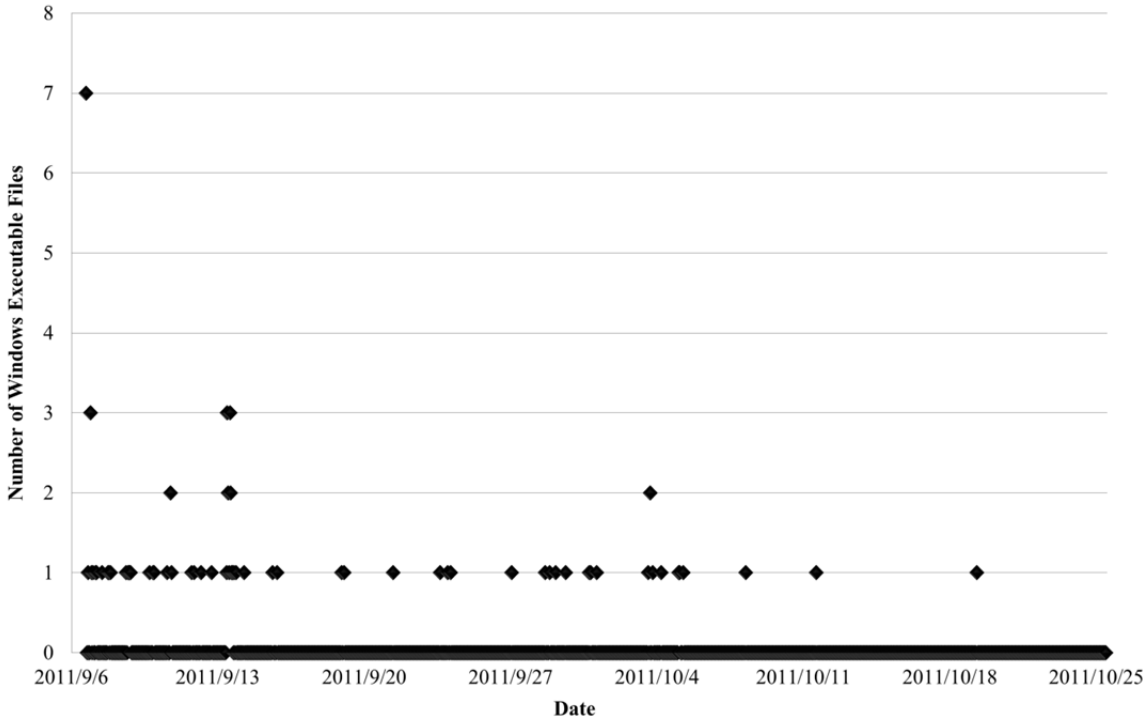

Figure 4.3 Number of Newly Observed Windows Executable Files

IRC PRIVMSG

Figure 4.4 shows the number of new IRC PRIVMSGs collected by the dummy client over time. From the figures, it is confirmed that the dummy client can continuously receive a number of new PRIVMSGs from the IRC-based C&C servers. Since PRIVMSG is often used as a C&C command between a bot and its herder, it is important to observe such commands. For example, there are messages such as *!get http://netnetnet1.com/sd7.txt*, which seems to be a command for downloading a file. Actually, in the sandbox analysis, after having received such a message, the malware accessed the URL and downloaded an executable file.

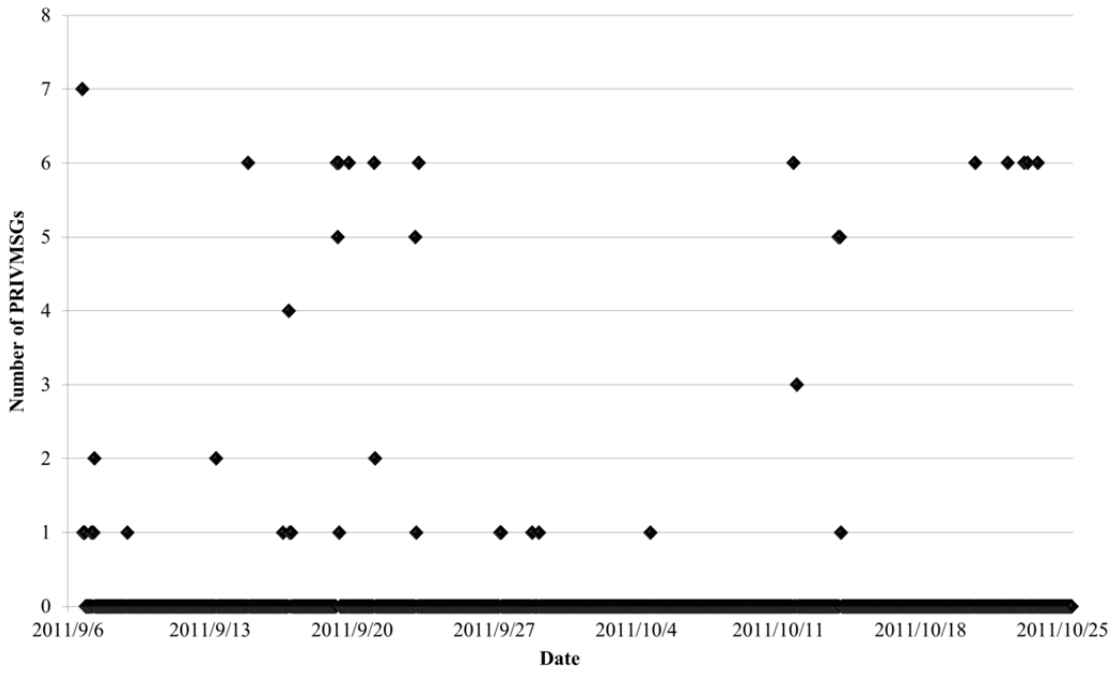

Figure 4.4 Number of Newly Observed PRIVMSGs

(3) Second Analysis

Table 4.3 shows the comparison between the analysis results of the proposed sandbox and Internet-connected sandbox. In Table 4.3, almost all items increased with the proposed sandbox. In particular, numbers of samples that used HTTP POST, samples that received an executable file by HTTP GET, samples that used SMTP, samples that used DNS MX-record queries and PTR-record queries, unique queried domain names, and samples that performed a port scan greatly increased. These results indicate an improved observability of malware behaviors by the proposed sandbox. In the proposed sandbox, 1,347 malware samples attempted to connect to the 19 remote servers whose responses were collected by the dummy client. Of these 19, 8 servers were not able to be accessed

in the second analysis; thus, the feedback server replied using the collected responses. This is one of the main reasons for this improvement.

The above results demonstrate that the proposed sandbox has a certain level of feasibility for observing malware behaviors corresponding to the behaviors of remote servers.

Table 4.3 Comparison of Proposed Sandbox and Internet-connected Sandbox

	Internet-Connected Sandbox	Proposed Sandbox	Unit
Samples that used TCP	1982	2073	(sample)
Samples that used HTTP	1310	1354	(sample)
Samples that used HTTP GET	1308	1344	(sample)
Samples that used HTTP POST	84	203	(sample)
Samples that received EXE by GET	310	438	(sample)
Samples that used SMTP	78	162	(sample)
Samples that used IRC	448	480	(sample)
Samples that used IRC PASS	29	30	(sample)
Samples that used IRC NICK	445	472	(sample)
Samples that used IRC USER	445	472	(sample)
Samples that used IRC JOIN	439	440	(sample)
Samples that used IRC MODE	43	46	(sample)
Samples that received IRC PRIVMSG	404	409	(sample)
# of unique received PRIVMSGs	8	37	(PRIVMSG)
Samples that used UDP	2089	2067	(sample)
Samples that used DNS	2088	2063	(sample)
Samples that used DNS A-record queries	2088	2063	(sample)
Samples that used DNS MX-record queries	18	142	(sample)
Samples that used DNS PTR-record queries	30	128	(sample)
# of unique queried domain names	687	1683	(domain)
Queried domain names per sample	2.2	4.3	(domain)
Samples that performed port scan	1619	1720	(sample)
Samples that performed DoS attack	40	86	(sample)
Samples that sent shell code to 135/TCP	26	23	(sample)
Samples that sent shell code to 139/TCP	175	273	(sample)
Samples that sent shell code to 445/TCP	1503	1591	(sample)
Samples that sent shell code to 3127/TCP	79	79	(sample)
Size of traffiic log per sample	11.1	75.7	(MB)

4.6 Discussion

Although the proposed sandbox showed the possibility of working for observing malware behaviors in accordance with server behaviors, there are several limitations that need to be discussed.

4.6.1 Detection of Dummy Client by Remote Servers

In this implementation, the dummy client simply replays the same communication observed in the sandbox. An attacker can easily overcome such a simple replay. The attacker can include sensitive information (source IP address, timestamp, other internal system parameter, or their hash values) in URL parameters, and reject queries with bad parameters. Moreover, if the remote servers change responses in accordance with URL parameters, not all responses can be obtained because applying duplication-reduction rules eliminates some sessions. The attacker can also adopt SSL or any other kind of protocol that is highly interactive. The dummy client needs to be made more interactive to deal with these interactive protocols. The internal behavior of the malware samples needs to be examined through techniques like API hooking in order to obtain necessary information for more complete emulation of server-client interactions.

In addition, in the experiment, the dummy client used the same IP address for accessing the remote servers. Therefore, the attacker can detect the dummy client by looking at the frequently used IP addresses of the clients connecting their servers. Therefore, to avoid this network-based detection of the dummy client, the client's IP address should be frequently changed. Another solution is to use anonymity networks like *The Onion Router* (*TOR*) [79]. TOR relays a multilayered encrypted message among its onion routers for sender/receiver anonymity. By using TOR, a dummy client can connect to the remote servers without revealing its IP address. However, careful consideration from the attackers' viewpoint needs to be taken. Although TOR reasonably provides sender anonymity, there are several techniques for the receiver to determine if the sender is using TOR [80] [81]. If a normal malware victim hardly ever uses TOR, the very usage of it can raise an attacker's suspicion. A similar discussion applies for an anonymous proxy [82].

4.6.2 Determining Changes of Server Behavior

In the proposed method, it is important to determine whether remote servers have changed their behavior in order to emulate remote servers effectively. This is not a trivial issue. There are indeed some parts of a server's responses that change every time a client accesses it, such as a time stamp. These changes need to be automatically distinguished from those caused by the attacker's behavior in order to efficiently perform the feedback analysis. Moreover, the attacker can disturb this decision by intentionally randomizing responses of its server or introducing a challenge-response protocol to avoid a replayed query. However, it should be noted that for stable connection attackers

sometimes use a paid server-hosting service that supports only a standard protocol, and in such cases such customized protocols cannot be utilized. In fact, it was confirmed that some of the remote servers observed were hosted by a paid service and they used only a regular HTTP protocol; thus, this method worked effectively.

4.6.3 Filtering High-risk Communications

In our proposed method, the dummy client keeps replaying the observed malware traffic. It means that the client might mistakenly keep replaying a high-risk communications, such as remote exploits and DoS, to innocent hosts. In addition, although we don't filter HTTP traffic in the experiments, there are various exploits by using HTTP query such as *SQL injection*, *Remote File Inclusion*, and *Comment Spamming*. Therefore, the filtering of high-risk communication is a critical issue in our proposal. Besides the presented filtering rules, we can also estimate the likeliness of the remote server to be an attacker's server in various ways. One possible solution is to use a blacklist of malicious servers.

Chapter 5

Decoy Injection Attack on Public Malware Sandbox Analysis Systems

5.1 Introduction

With the growing popularity of malware sandbox analysis, there are a number of systems [1] [2] [3] [4] [10] [11] [18] [19] [20] [21] [22] [27] [83] [84] [85] [86] [87] [88] [89] [90] [91] that use a public interface to accept online submissions of samples from arbitrary users, automatically analyze them using a sandbox, and send analysis reports back to the user. Most of these systems focus on Windows executable files, but some focus on JavaScript [88], Flash [88], DLL [4], and PDF [4] [88]. Similar systems also exist for the analysis of suspicious websites [1] [2] [83] [84] [85] [86] [87] [88] [89] [91]. This dissertation refers to a malware analysis system with a public interface as a *public malware sandbox analysis system* (*public MSAS*). One public MSAS reportedly received over 900,000 submissions of unique samples (based on MD5 hashes) in less than two years [8], demonstrating the popularity of such systems. Previous works [31] [32] have pointed out a vulnerability of public MSASs against *decoy injection attacks*, in which an attacker detects the sandbox based on its IP address that can be obtained by submitting a *decoy sample* designed for this purpose. Yet they did not further investigate the possibility of detection using sandbox information other than its IP address.

In this chapter, in order to better understand the vulnerability and develop an effective countermeasure, first, we define properties of sandbox information for decoy injection attack:

stability, *uniqueness*, and *stealthiness of collection*. Then, we evaluate 16 different kinds of characteristics in the sandbox in terms of those properties. As a result of experiments with real public MSASs in operation, we found that characteristic information such as a *Windows OS product key*, *MAC address*, and *Windows OS install date* can be utilized for sandbox detection, except for particular systems that appeared to have deployed a countermeasure. Moreover, besides network-based disclosure, we show that such characteristic information of the sandbox can be disclosed via an analysis report provided to the user, which means that the decoy injection attack can be performed against the sandbox isolated from the real Internet. Thus, this study confirms the broad applicability of the decoy injection attack as well as the need for comprehensive countermeasures.

The rest of this chapter is as follows. Section 4.2 describes the model of public MSASs, and Section 4.3 explains some examples of existing public MSASs. Section 4.4 addresses decoy injection attacks against public MSASs, and Section 4.5 explains the properties of 4.5 properties of sandbox information for decoy injection attacks. Section 4.6 describes a case study with 15 existing public MSASs for evaluating the impact of a decoy injection attack. Section 4.7 provides discussion, and Section 4.8 summarizes this chapter.

5.2 Models

This section explains the model of a public MSAS, which can be classified into two groups; one for analyzing a submitted file and the other for analyzing a website of a submitted URL. In this dissertation, the former is called a *public MSAS for sample files* (*public MSAS-F*) and the latter a *public MSAS for websites* (*public MSAS-W*).

5.2.1 Public MSAS-F

Figure 5.1 and Figure 5.2 respectively show the model of a public MSAS-F with an isolated sandbox and that with an Internet-connected sandbox. In both systems, the *submitter* is the user of the analysis system who submits a sample file to the system. The *reception* is the publicly known interface of the system, which is typically realized by a website to accept sample submissions. The *sandbox* represents the testing environment in which the submitted sample is executed and analyzed.

An isolated sandbox does not connect to the real Internet, but rather the emulated Internet, which consists of various dummy servers, as depicted in Figure 5.1. The Internet-connected

sandbox connects to the Internet, as in Figure 5.2. The outbound traffic from an Internet-connected sandbox is carefully checked in order to mitigate the risks of infection outside the sandbox [9] [24]. The sandbox implements various means such as API hooking to monitor internal behavior of the executed sample. Finally, the *analysis report* that describes the detailed malware behavior such as API calls, file access, registry access, process creation, network activities, is provided to the submitter via reception or other electronic means, such as e-mail.

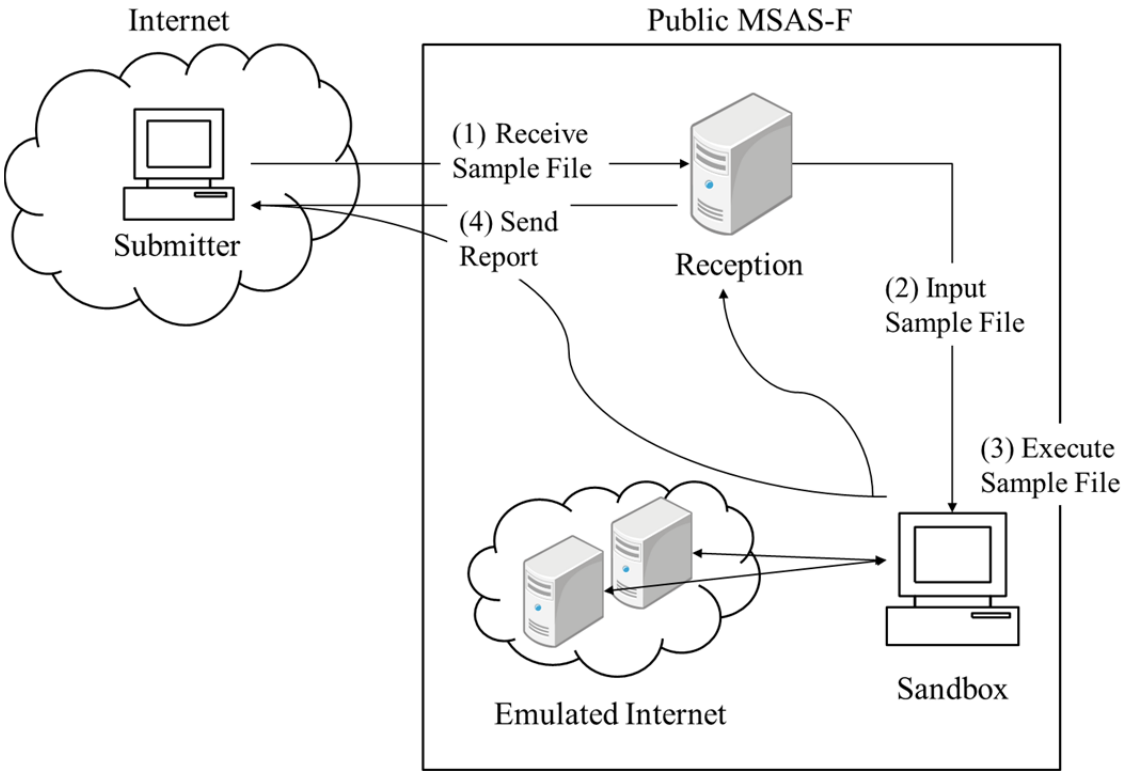

Figure 5.1 Model of Public MSAS-F for Sample Files (Public MSAS-F) with an Isolated Sandbox

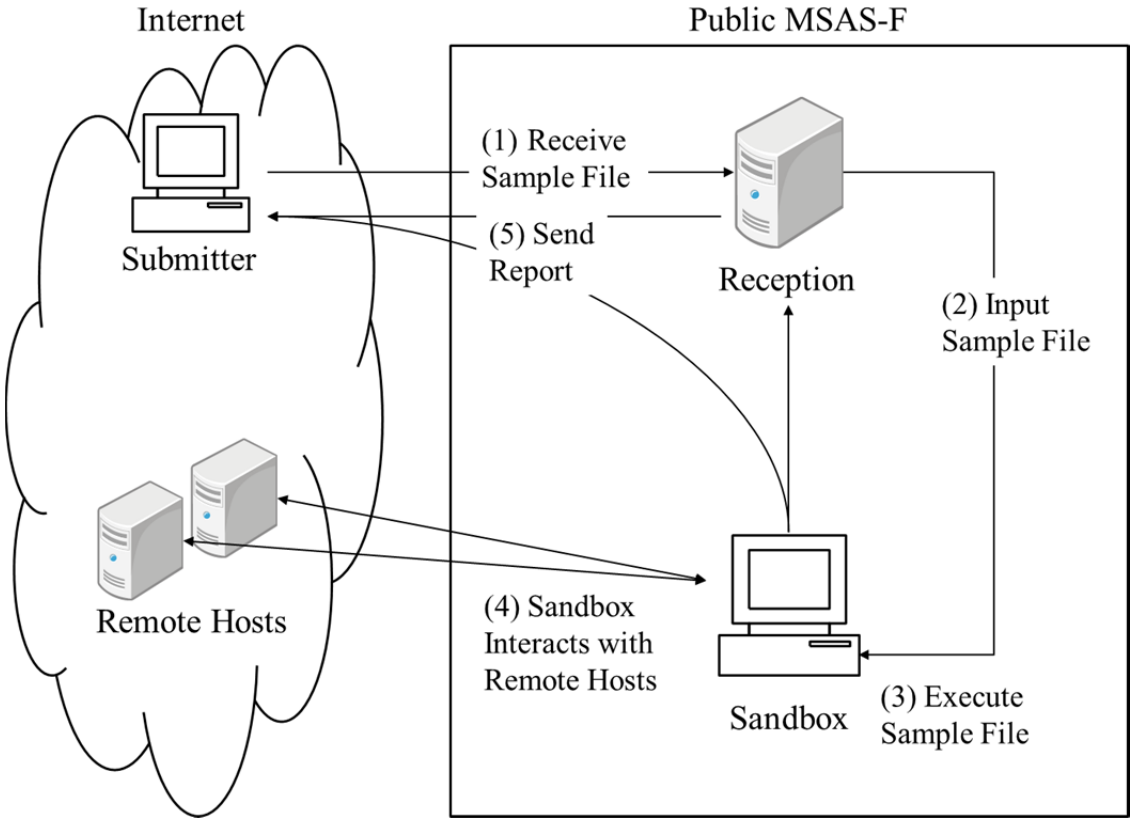

Figure 5.2 Model of Public MSAS for Sample Files (Public MSAS-F) with an Internet-connected Sandbox

5.2.2 Public MSAS-W

Figure 5.3 shows the model of a public MSAS-W. The analysis of a website works similarly as with a public MSAS-F. The submitter first submits a URL of a website for which he or she wants to check the level of safety with regard to the system. Then, the sandbox actually accesses the website to obtain the web contents for analysis. To analyze a website that refers to other websites the sandbox also connects to them to obtain the referred content; so the sandbox for a public MSAS-W is Internet-connected by nature.

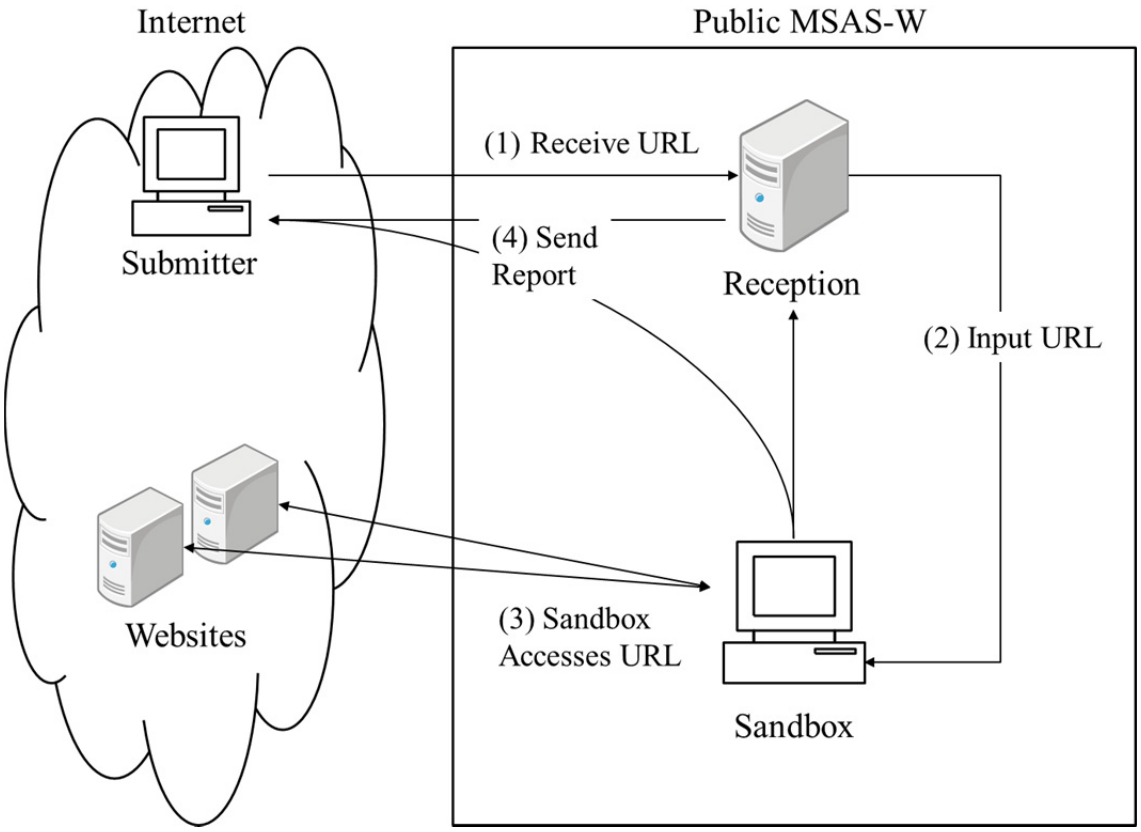

Figure 5.3 Model of Public MSAS for Websites (Public MSAS-W)

5.3 Samples of Public MSAS

This section gives some examples of existing public MSASs.

- *Norman SandBox*

Norman SandBox [18] is a malware sandbox analysis system with an isolated sandbox, developed by *Norman Safeground*. The company provides Norman SandBox as a public MSAS-F, and users are able to submit executable files to the system via the Norman Safeground website. The system analyzes submitted files and provides an analysis report to the submitter via e-mail. Analysis reports of all submitted files are also published on the website for public perusal.

Figure 5.4 and Figure 5.5 respectively show the web interface of the system for submitting an executable file and an example of a published analysis report.

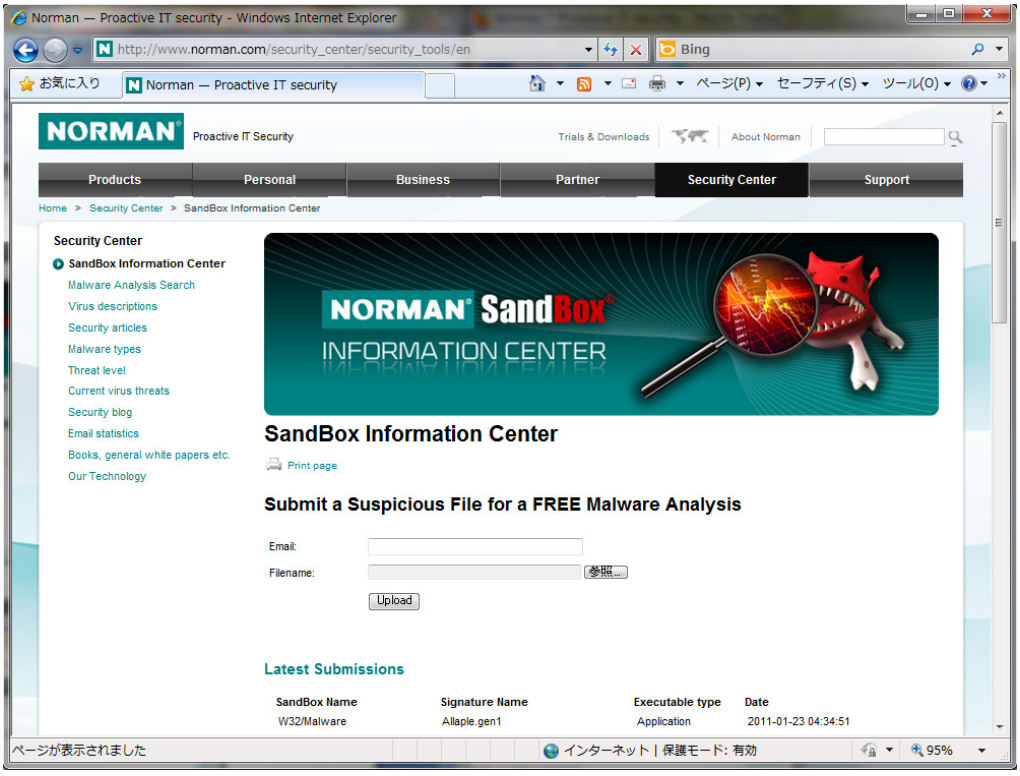

Figure 5.4 Web Interface of Norman SandBox for Submitting a File

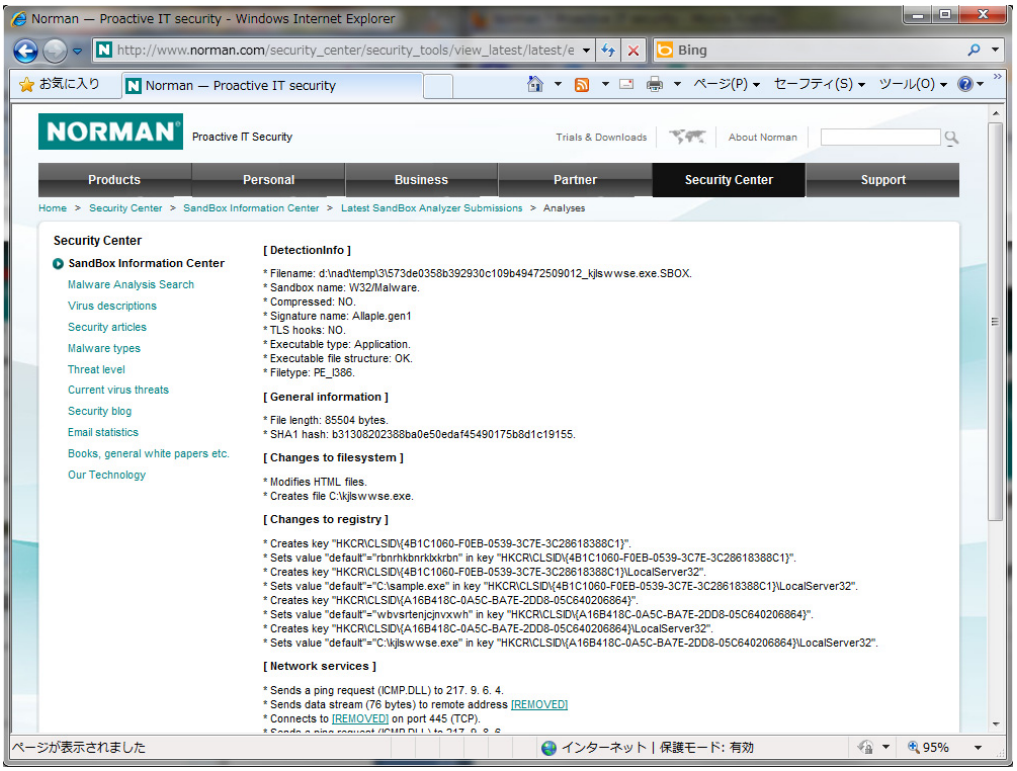

Figure 5.5 Example of Analysis Report of Norman SandBox

- *Anubis*

Anubis [2] is a malware sandbox analysis system with an Internet-connected sandbox, developed by *International Secure Systems Lab*. Anubis is also provided as a public MSAS. It is able to analyze not only Windows executable files but also Android APK files and websites. The Analysis report can be received via e-mail or website in a preferred format (e.g., HTML, XML, PDF, text file). Anubis also provides the traffic log (PCAP format) of the analysis. It is one of the most well-known public MSASs and, as previously reported [8], it had over 900,000 unique samples (based on MD5 hashes) submitted in less than two years.

Figure 5.6 and Figure 5.7 respectively show the web interface of the system for submitting an executable file and an example of a published analysis report.

Figure 5.6 Web Interface of Anubis for Submitting a File or URL

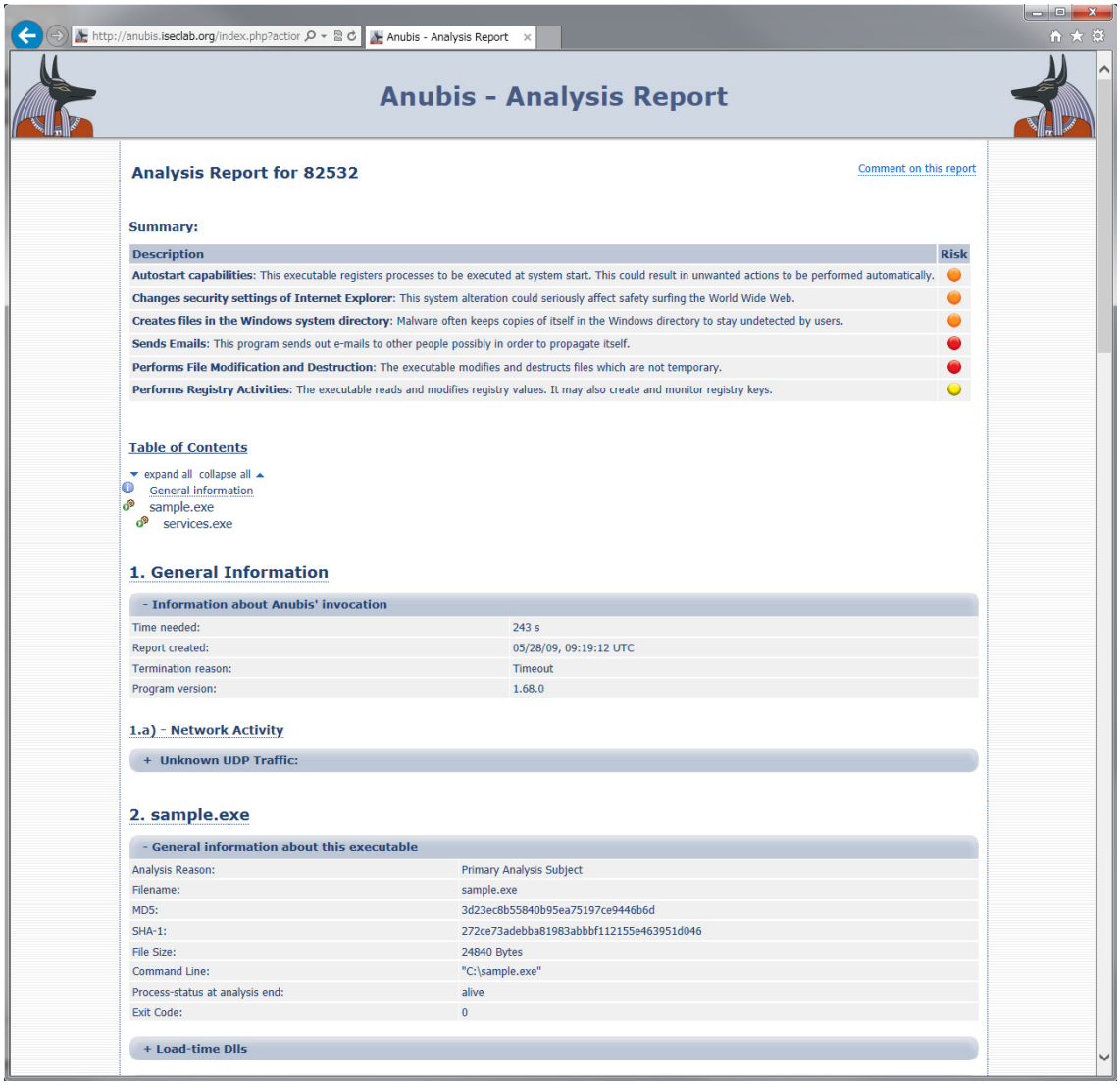

Figure 5.7 Example of Anubis Analysis Report

- *gred*

gred [89] is a malware sandbox analysis system for websites, developed by *SecureBrain Corporation* and provided as a public MSAS-W. When submitting a URL, submitters can request that it checks all links found within the web page of the submitted URL. gred then judges whether or not the URL is malicious. If it is, the analysis report shows what kind of threats it includes by using icons indicating threat categories (e.g., phishing site, one-click fraud, bogusware). In addition, if HTML files of the malicious URL include suspicious files, a submitter can receive an additional analysis report about those files via e-mail.

Figure 5.8 and Figure 5.9 respectively show the web interface of the system for submitting an URL and an example of a published analysis report.

Figure 5.8 Web Interface of gred for Submitting a URL

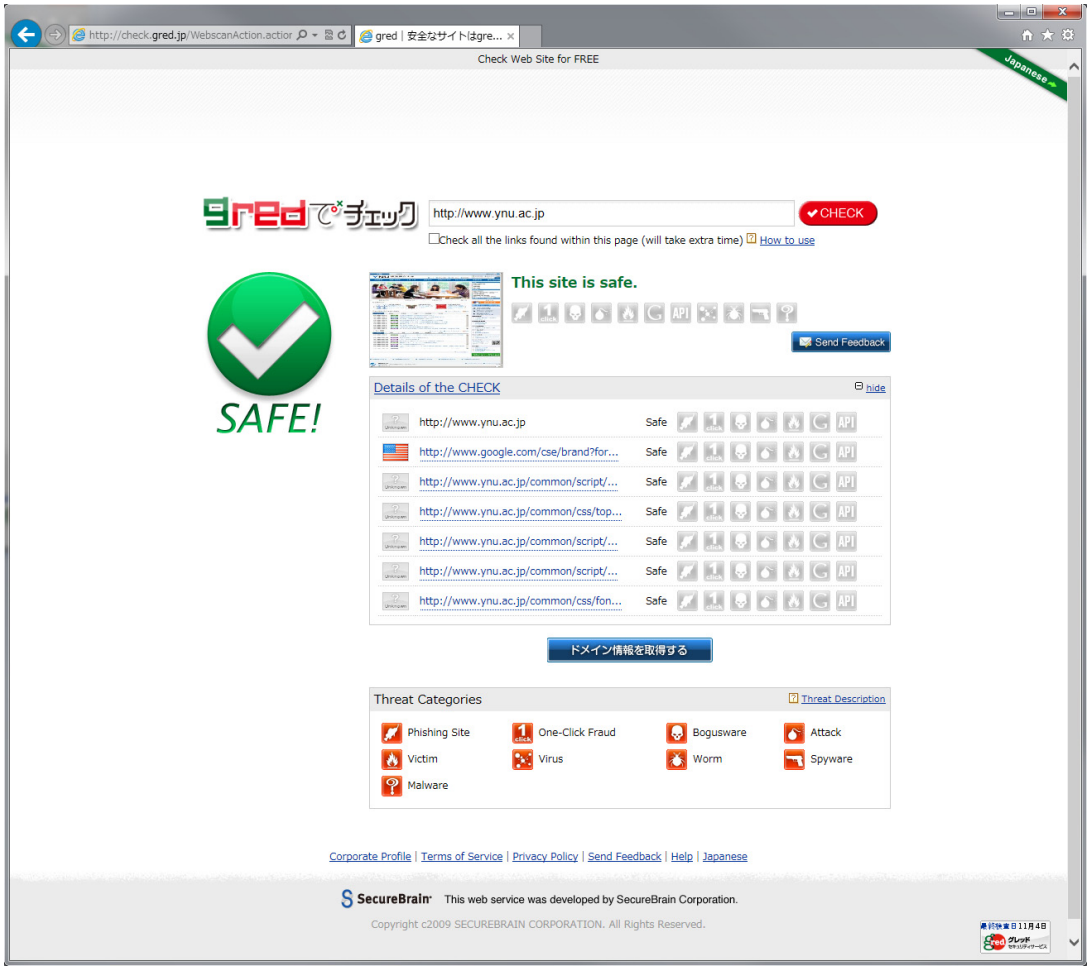

Figure 5.9 Example of Analysis Report of gred

5.4 Decoy Injection Attack

Since public MSASs are available to arbitrary users, even an attacker can use them as a submitter to learn the systems. The basic idea of a decoy injection attack is to submit a decoy sample that collects and discloses information on the targeted public MSAS. Typically, unique characteristics of the sandbox, such as its operating system's product ID or the existence of certain files, registry keys, and processes, can be utilized for sandbox detection.

This section explains the decoy injection attack that can be conducted against public MSASs. It consists of two phases. In the first phase, the *disclosing sandbox information phase*, the attacker submits a decoy to the targeted public MSAS in order to disclose its sandbox information. Then, in the second phase, the *sandbox-detection phase*, the attacker uses the disclosed sandbox information to detect the sandbox.

5.4.1 Disclosing Sandbox Information

In the disclosing sandbox information phase, the attacker's objective is to disclose sandbox information that can be utilized for detecting the targeted public MSAS by injecting decoy samples or decoy URLs into the system.

Figure 5.10 shows an overview of disclosing sandbox information against a public MSAS-F. To disclose the sandbox information from a public MSAS-F, the attacker simply submits a *decoy sample* to the targeted system. The sample is eventually executed in the system and discloses the system information it has collected. It should be noted that there are two possible channels over which the system information can be disclosed. The first is *communication between an Internet-connected sandbox and remote hosts*. The decoy sample can simply send the collected information over the network to a designated remote host under the attacker's control, called a *colluding server*, outside the sandbox. Obviously, this channel does not exist in a public MSAS using an isolated sandbox. The second channel is the *analysis report*. The attacker submits a decoy sample that *embeds* the obtained system information in the analysis report. For example, a decoy sample can encode a Windows OS product key and use it as the name of a file it creates. Since the name of the file the sample creates is likely to be mentioned in the analysis report, the attacker can obtain the product key via the report. This channel exists even in a public MSAS using an isolated sandbox.

Figure 5.11 shows an overview of disclosing sandbox information against a public MSAS-W. First, the attacker prepares a web server, called a *colluding web server*, and submits a decoy URL of a colluding web server.

When the sandbox accesses the colluding web server, there are three way of replying, as follows.

- Replying with benign contents or nothing
- Replying with benign contents with a client-side script (e.g., JavaScript) that collects sandbox information
- Replying with malicious contents with attack script that exploits a vulnerability of the web browser used by the sandbox in order to acquire a decoy sample downloaded and executed in the sandbox

When replying with benign contents or nothing, the attacker can only learn the IP address of the sandbox and HTTP request headers. When replying with a client-side script that collects sandbox information, the attacker can learn some additional information such as *browser plug-in*, *time zone*, and *monitor resolution*. Since there are some studies about browser fingerprinting based on information collected by JavaScript [92], the attacker has a chance of detecting the sandbox via identifying the web browser of the sandbox. When replying with malicious contents, since the web contents contain the exploit, a decoy sample is downloaded and executed in the sandbox. The executed decoy sample then works the same as for a public MSAS-F.

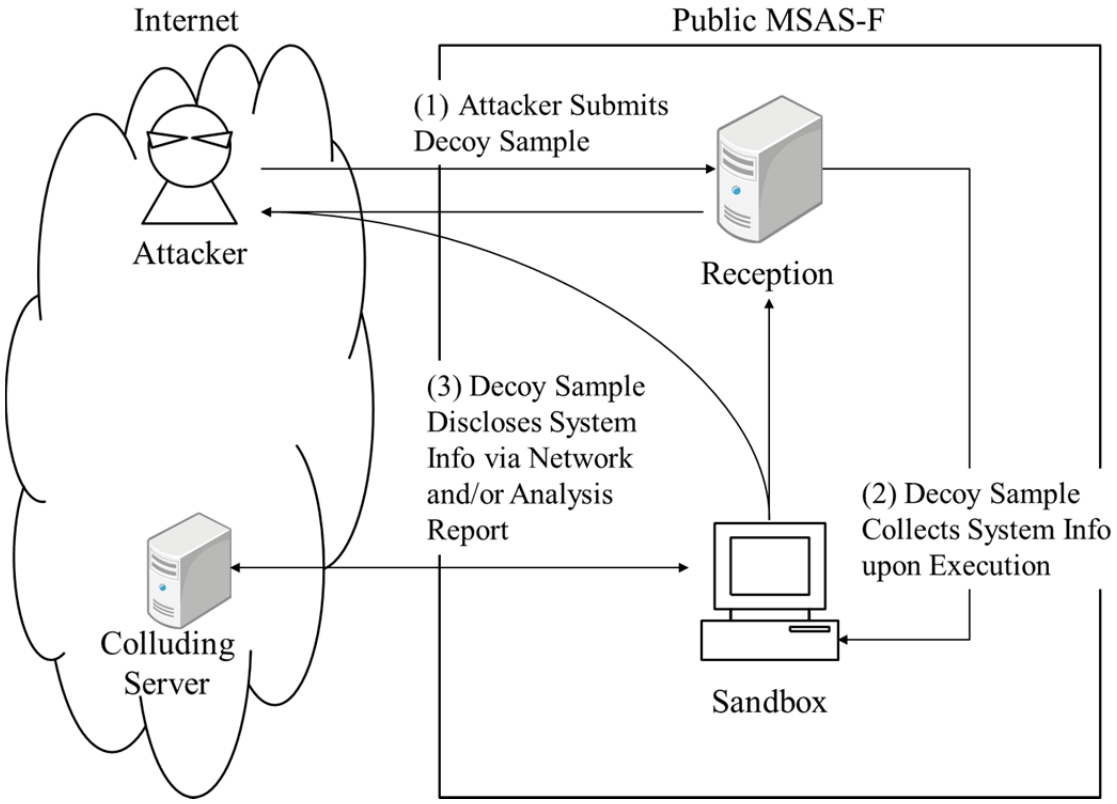

Figure 5.10 Disclosing Sandbox Information Against Public MSAS-F

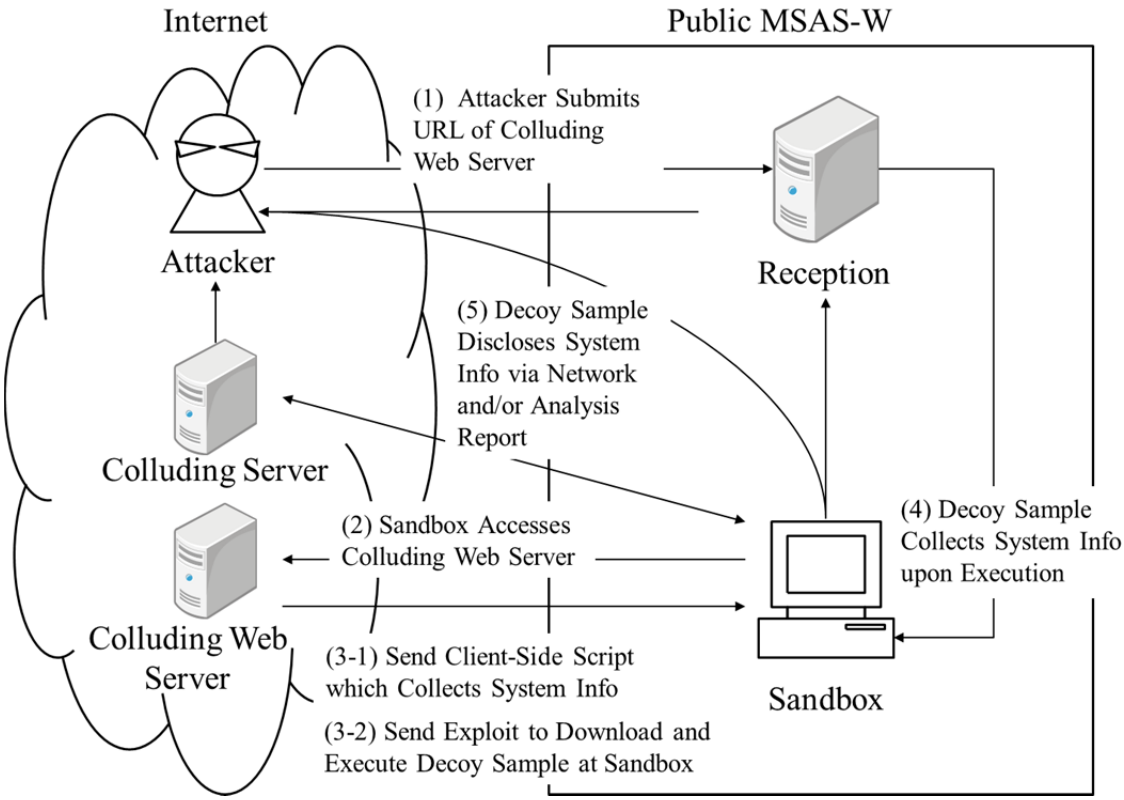

Figure 5.11 Disclosing Sandbox Information Against Public MSAS-W

5.4.2 Sandbox Detection

In the sandbox-detection phase, the attacker's objective is to detect and evade sandbox analysis by leveraging sandbox information disclosed in the disclosing sandbox information phase. There are two types of sandbox detection: *host-based* and *network-based*.

In host-based sandbox detection, the attacker has embedded disclosed sandbox information of the targeted public MSAS in malware. The attacker also implements a function that checks sandbox information of the run-time environment and compares it with embedded information in malware. Thus, if embedded information is found in the run-time environment, the malware judges the environment as a sandbox and stops or changes behaviors in order to disrupt analysis. Figure 5.12 shows an overview of host-based sandbox detection against a public MSAS-F.

In network-based sandbox detection, first the attacker has embedded disclosed information of the targeted public MSAS in a C&C or malware download server, malicious web server, etc. Figure 5.13 shows an overview of network-based sandbox detection against a public MSAS-F. The attacker implements a function that collects sandbox information on the run-time environment and sends the information to a remote server such as a C&C or malware download server. Then, the remote server compares the information received from the malware with that disclosed at the

disclosing sandbox information phase. If it matches, the remote server changes responses to the malware in order to disrupt analysis of the sandbox (e.g., the remote server never sends a C&C message to the malware in the sandbox). Figure 5.14 shows an overview of a network-based sandbox detection using an IP address against a public MSAS-W.

The advantage of host-based sandbox detection is that it has the chance to detect a public MSAS-F with an isolated sandbox. The advantage of network-based sandbox detection is that it is easy to update the disclosed sandbox information for detection.

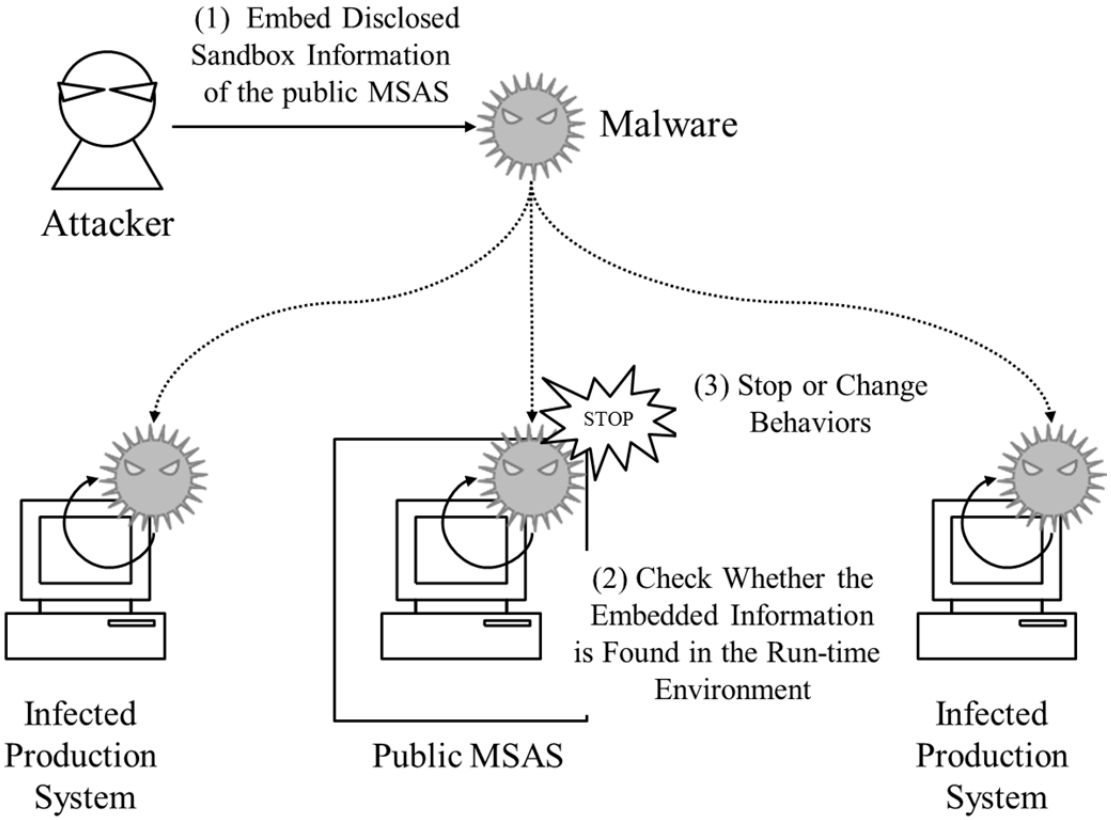

Figure 5.12 Host-based Sandbox Detection Against Public MSAS-F

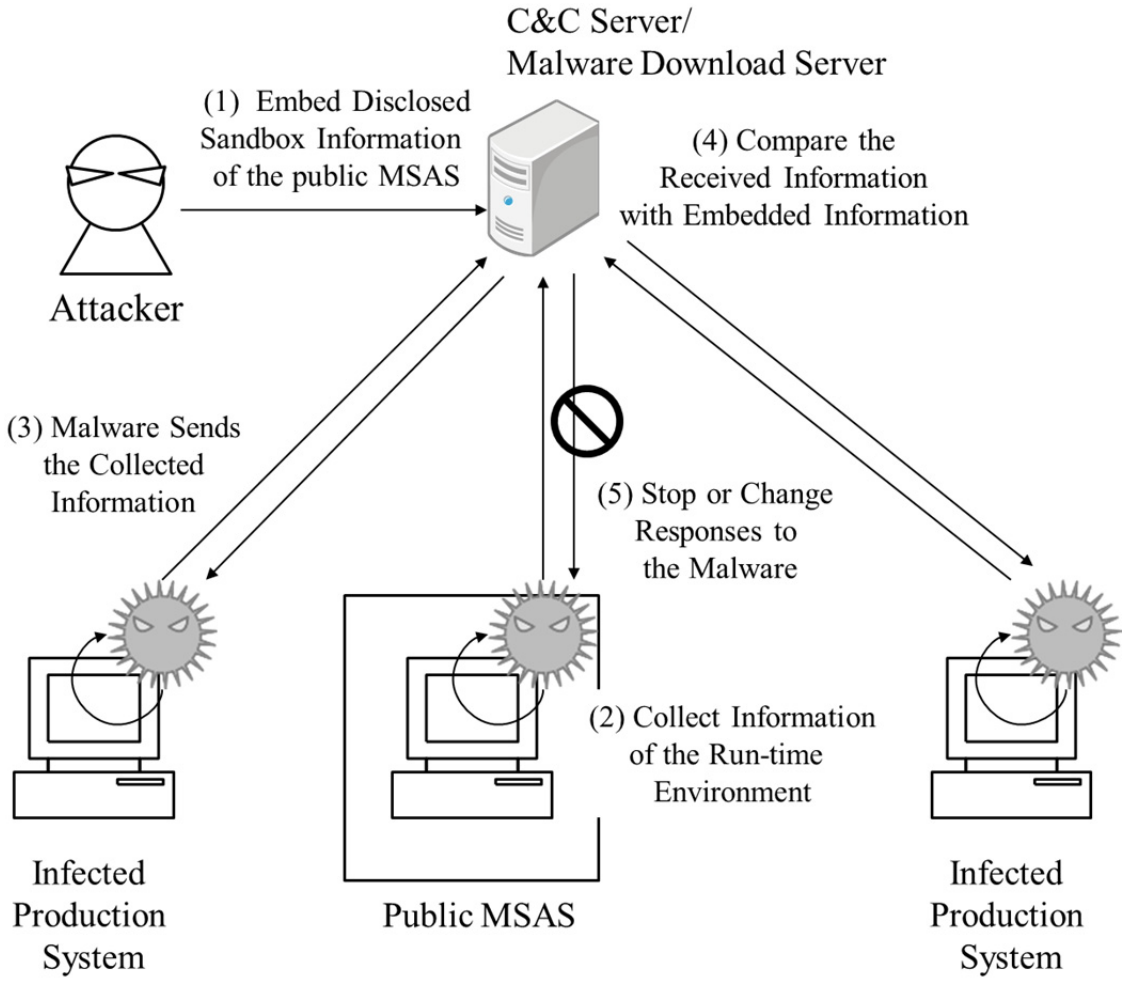

Figure 5.13 Network-based Sandbox Detection Against Public MSAS-F

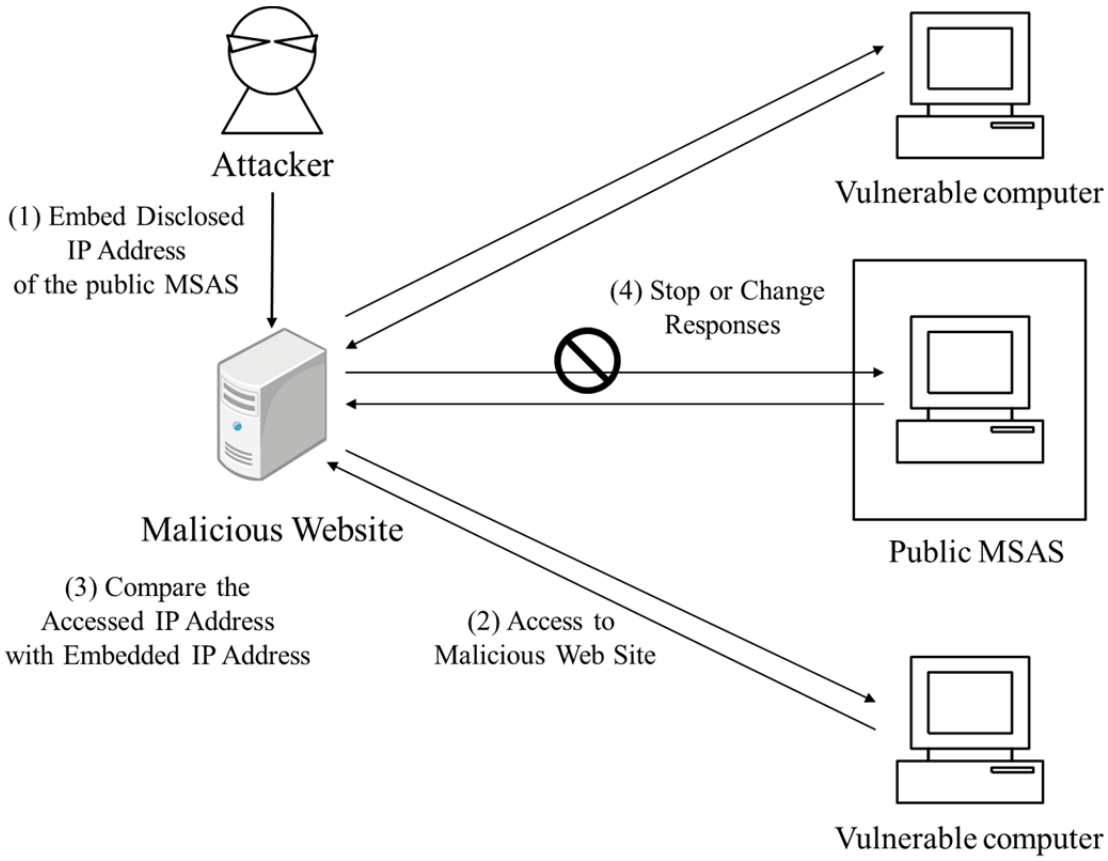

Figure 5.14 Network-based Sandbox Detection Using IP address Against Public MSAS-W

5.5 Properties of Sandbox Information for Decoy Injection Attack

This section considers three important properties of sandbox information for a decoy injection attack: *stability*, *uniqueness*, and *stealthiness of collection*.

Stability is the property of being able to collect the same value of the sandbox information every time. Since the attacker must collect the same information in the two phases of the decoy injection attack for efficient detection, the sandbox information utilized in the decoy injection attack requires a high level of stability.

Uniqueness is the property of the values of the information collected from the systems being different from each other. If the attacker uses sandbox information with a low level of uniqueness

for the decoy injection attack, it leads to many false positives, so a high level of uniqueness is required.

Stealthiness of collection is the property that can conceal collecting the sandbox information from the public MSAS side. If the attacker accesses sandbox information that has a high level of uniqueness and a low level of stealthiness of collection, the possibility of the sandbox detection by public MSAS side may be detected, so a high level of stealthiness of collection is required.

5.6 Evaluation

This section describes the experiments for evaluating the impact of a decoy injection attack on 15 existing public MSASs, consisting of eight public MSAS-Fs and seven public MSAS-Ws. The *i-th* system is referred to as System *i* for *i = 1, 2, .., 15*. Note that Systems 1-8 are public MSAS-Fs and 8-15 are public MSAS-Ws.

5.6.1 Disclosing Sandbox Information

In this section, decoy injection to public MSAS was conducted for disclosing sandbox information. Then, the sandbox information used in the sandbox-detection phase was selected based on the results of the disclosing sandbox information phase.

5.6.2 Generating Decoy Samples and Decoy URLs

For evaluating the eight public MSAS-Fs, a decoy sample was prepared that behaved as follows.

1) It attempts to collect 15 types of sandbox information, as shown in Table 5.1, by using the Windows API. The colluding server collects the IP address.

2) If the sandbox information is collected, the sample base64 encodes the value of the sandbox information, and the URL encodes the base64-encoded data. In the following step, the sandbox information indicates the data after the encoding.

3) The sample creates a registry subkey in *HKEY_CURRENT_USER*.
4) The sample creates registry entries under the subkey by using the sandbox information code as the entry name and the value of sandbox information as the entry value.

5) It attempts to connect to the colluding server. If the domain name of the server is not resolved or there is no reply from the colluding server, the sample halts.

6) When connected to the colluding server, it issues an HTTP GET request. The name of the requested file represents the unique identifier of the sample. The parameters of the request represent the sandbox information.

There are two main features of the decoy sample.

- Creating the registry keys to disclose the sandbox information via an analysis report
- Connecting to the colluding server with a unique identifier to disclose the sandbox information via network access

Table 5.1 Sandbox Information Collected by Decoy Sample

	Sandbox Information
arp	MAC Addresses of Surrounding PC's
bbn	Motherboard Product Name
bbns	Motherboard Serial Number
bn	BIOS Product Name
bs	BIOS Serial Number
cn	Computer Name
cpu	CPU Product Name
cpuid	CPU Serial Number
dn	Disk Drive Name
ds	Disk Drive Serial Number
insd	Windowa OS Install Date
mac	MAC Address
pk	Windows OS Product Key
pn	Windows OS Product Name
un	User Name

For evaluating the seven public MSAS-Ws, a decoy URL was also prepared by combining the domain name of the colluding server and the unique identifier of the sample (e.g., http://disclosing-server.com/ID.cgi).

5.6.3 Preparing Colluding Servers

A colluding server and CGI files with names corresponding to each decoy ID were prepared. The server works as follows.

1) It waits for an incoming TCP connection request from a client.

2) If an HTTP GET request arrives from a client, the server checks whether the name of the requested file is in the list of valid identifiers.

3) The server collects the parameters of the request, acquires the sandbox information by URL decoding and base64 decoding the parameters, and stores that in log files. The server also collects and stores the HTTP request header and IP address of the client. Then the server sends the benign HTTP response to the client.

5.6.4 Procedure

An experiment was performed concerning the disclosing sandbox information phase during a one-week period in November 2010. Five decoy samples/URLs per day per system were submitted to the 15 target systems. All submitted samples/URLs contained a distinct identifier.

5.6.5 Results

Table 5.2 shows a summary of the experiments. It includes the number of received analysis reports, number of accesses to colluding servers, and results of disclosure about the information. Note that the sandbox information, except for the IP address in Systems 3-7, was able to be disclosed via both network access and analysis report, and the table includes both results. The results except for the IP address were represented in the form of *number of unique values* / *number of successful disclosures*. When multiple values of the sandbox information were collected, the lists were compared, and if they matched exactly they were regarded as the same.

Note that in the experiments exploit code was not sent to the public MSAS-W in order to acquire a decoy sample downloaded, and executed in the sandbox but only the IP address of the sandbox was collected. Therefore, in Systems 9-15, there are no results of sandbox information except for the IP address.

- *Disclosing Sandbox Information via Network Access*

HTTP GET requests were observed at the colluding server immediately after each decoy submission to 12 target systems: 3-7, and 9-15. Thus, the sandbox information was able to be disclosed via network access. They are considered to utilize an Internet-connected sandbox. Among them, seven systems – Systems 5 and 10-15 – used a single IP address or a few IP addresses for the entire experimental period of one week. There was no connection that made attempts to the colluding servers using identifiers for the three target systems: Systems 1, 2, and 8. These systems are considered to utilize an isolated sandbox. They are tolerant of disclosing sandbox information via network access. However, disclosing sandbox information via analysis report is still effective against these systems.

In System 6, the number of accesses to colluding servers is less than the number of submitted decoy samples. This is because that DNS query of colluding server's domain failed due to a system error or some other factor. In contrast, in Systems 10, 11, and 15, the number of accesses to colluding servers is greater than the number of submitted decoy URLs. This is assumed to be because the systems conduct multiple accesses to the server with different environments, such as different versions of a web browser, for improving detection capability.

- *Disclosing Sandbox Information via Analysis Report*

The sandbox information was able to be disclosed via analysis report in the public MSAS-F, with the exception of Systems 1 and 2. In these systems, there is less behavior information of the sample included in the analysis report than for other systems, and it is assumed that a decoy sample might fail to collect sandbox information. Since the sample can collect sandbox information in a normal environment, the attacker has a chance to detect the sandbox by a fact not to be able to collect sandbox information.

Table 5.2 Summary of Experiments About Disclosing Sandbox Information Phase

No.	Type of public MSAS	# of Submitted Decoy Samples or URLs	# of Received Analysis Reports	# of Accesses to Colluding Servers	Results (# of Unique Values / # of Successful Disclosure)															
					IP Address	arp	bbn	bbs	bn	bs	cn	cpu	cpuid	dn	ds	insd	mac	pk	pn	un
1	public MSAS-F	35	35	0																
2	public MSAS-F	35	35	0																
3	public MSAS-F	35	35	35	8 (7 in a /28 NW)	1/35	0/0	0/0	1/35	0/0	10/35	1/35	0/0	1/35	0/0	1/35	35/35	1/35	1/35	0/0
4	public MSAS-F	35	35	35	7 (7in a /18 NW)	1/35	1/35	6/35	1/35	0/0	1/35	1/35	0/0	1/35	0/0	1/35	6/35	1/35	1/35	1/35
5	public MSAS-F	35	34	35	1	35/35	0/0	0/0	1/35	0/0	1/35	1/35	0/0	1/35	0/0	1/35	4/35	1/35	1/35	1/35
6	public MSAS-F	35	34	31	25	2/35	0/0	0/0	1/35	0/0	1/35	1/35	0/0	1/35	0/0	1/35	1/35	4/35	1/35	0/0
7	public MSAS-F	35	35	35	10 (10 in a /27 NW)	35/35	1/35	1/35	1/35	1/35	1/35	1/35	0/0	0/0	1/35	0/0	10/35	1/35	1/35	1/35
8	public MSAS-F	35	35	0		2/32	1/32	1/32	1/32	2/32	2/32	1/32	0/0	1/32	0/0	1/32	2/32	2/32	1/32	1/32
9	public MSAS-W	35	35	35	8 (7 in a /28 NW)															
10	public MSAS-W	35	35	70	1															
11	public MSAS-W	35	35	140	3															
12	public MSAS-W	35	35	35	1															
13	public MSAS-W	35	35	35	2															
14	public MSAS-W	35	35	35	4 (4 in a /30 NW)															
15	public MSAS-W	35	35	107	4															

5.6.6 Selecting Sandbox Information for Sandbox Detection

This section assessed the sandbox information shown in Table 5.2 from the perspective of the properties described in Section 4.5, and selected suitable sandbox information for a decoy injection attack.

- *Stability*

First, in all public MSASs with an Internet-connected sandbox, access to the colluding server was able to by observed and the IP address was able to be collected every time, with the exception of the error in the DNS response in System 6. In addition, in Systems 3, 5, 7, and 9-15, it was observed that they used a single IP address or a few IP addresses within a small subnet. Because of the above results, it is considered that the IP address has a high level of stability for a decoy injection attack. More detailed information about IP addresses used in Systems 4 and 6 are given in the results of the sandbox-detection phase.

On the other hand, for sandbox information except for the IP address, seven types of sandbox information – *BIOS product name* (*bn*), *computer name* (*cn*), *CPU product name* (*cpu*), *disk drive name* (*dn*), *Windows OS install date* (*insd*), *Windows OS product key* (*pk*), and *Windows OS product name* (*pn*) – ware able to be collected every time, and the numbers of unique values of the sandbox information are very low in all public MSAS-Fs. Thus, the seven types of sandbox information apparently have a high level of stability for a decoy injection attack.

- *Uniqueness*

.basically unique to each environment, and have a high level of uniqueness. Actually, in this experiment, these items were unique to each public MSAS. However, it must be confirmed in the future whether the information is actually unique to each environment included in the normal user's environment.

On the other hand, it is considered here that the information such as the BIOS product name, CPU product name, and disk drive name becomes the same value when the systems have the same model of such parts. Actually, in this experiment, it was observed that these types of the information are sometimes the same between different systems, so it is difficult to use the information for identifying public MSASs. In addition, information such as the user name and computer name was unique to each public MSAS. However, sandbox information such as the user name and computer name apparently does not have a sufficient level of uniqueness since a user can easily change the information.

- *Stealthiness of Collection*

In this experiment, since information was collected using the Windows API except for the IP address, the means of collection causes no difference in the level of stealthiness of collection. Since the behavior that collects the information of the Windows OS product key from the registry key is not usually observed for benign software, the behavior apparently does not have a high level of stealthiness of collection. In contrast, the IP address can easily be collected via access to the colluding server. Since the behavior that accesses the Internet is often observed in benign software, it is considered that such behavior has a high level of stealthiness of collection.

In consideration of the above results, four types of information were selected – *IP address*, *Windows OS product key*, *MAC address*, and *Windows OS install date* – as the information used in sandbox detection. Note that in this experiment about public MSAS-W, since only the IP address was collected, an attempt was made to detect a public MSAS-W by only using the IP address.

5.7 Sandbox Detection

This section addresses an experiment conducted to confirm whether the public MSAS can be detected by using the information selected in Section 5.6.6

5.7.1 Generating Test Samples for Public MSAS-F

For evaluating the eight public MSAS-Fs, a *test sample* was first prepared. It was embedded in the three types of sandbox information (i.e., Windows OS product key, MAC address, Windows OS install date) collected in the disclosing sandbox information phase. The test sample behaved as follows.

1) It attempts to collect the Windows OS product key, MAC address, and Windows OS install date by using the Windows API.

2) If the sandbox information is collected, the sample base64 encodes the value of the sandbox information, and the URL encodes the base64-encoded data. In the following step, the sandbox information indicates the data after this encoding.

3) The sample creates a registry subkey in *HKEY_CURRENT_USER*.

4) It compares the sandbox information collected in the run-time environment with the list of the embedded sandbox information collected in the disclosing sandbox information phase.

5) If the collected information is not included in the list, the sample creates registry entries under the subkey by using the sandbox information code as the entry name and the value of sandbox information as the entry value.

6) It attempts to connect to the colluding server. If the server's domain name is not resolved or there is no reply from the colluding server, the sample halts.

7) When connected to the colluding server, it issues an HTTP GET request. The name of the requested file represents the unique identifier of the sample and the parameters of the request represent the sandbox information.

8) If it receives the requested file from the server, the sample checks whether the file contains a keyword ASCII string.

9) If a keyword string is found in the file, the sample proceeds with its hidden behavior; it creates a file on the infected system and tries to connect to another server. If a keyword string is not found, the sample halts.

There are two main features of the test sample.

- Attempting host-based detection by comparing the collected sandbox information in the run-time environment with the list of the embedded sandbox information collected in the disclosing sandbox information phase
- Attempting network-based detection by sending the collected sandbox information to the colluding server and checking the response from the colluding server

For evaluating the seven public MSAS-Ws, a *test URL* was prepared by combining the domain name of the colluding server and the unique identifier of the sample (e.g., *http://detecting-server.com/ID.cgi*).

5.7.2 Preparing Colluding Servers

A colluding server and CGI files with names corresponding to each test sample ID were prepared. The server works as follows.

1) It waits for an incoming TCP connection request from a client.

2) If an HTTP GET request arrives from a client, the server checks whether the name of the requested file is in the list of valid identifiers.

3) The server collects the parameters of the request, acquires the sandbox information by URL decoding and base64 decoding the parameters.

4) It compares the received sandbox information with the list of the sandbox information that was collected in the disclosing sandbox information phase.

5) If the received sandbox information is not included in the list, the server sends the response that includes a *keyword string* or $HTML_{malicious}$. Note that this experiment used benign content as $HTML_{malicious}$.

5.7.3 Procedure

An experiment on the sandbox-detection phase was performed during a one-week period in November 2010. Five test samples/URLs per day per system were submitted to the 13 target systems, with the exception of Systems 1 and 2. All submitted samples/URLs contained a distinct identifier.

5.7.4 Results

Table 5.3 shows a summary of experiments about the sandbox-detection phase. It includes the number of received analysis reports, number of accesses to the colluding servers, and results of the host-based sandbox detection and network-based sandbox detection. The results were represented in the form of *number of successful detections* / *number of successful collections of sandbox information*. When multiple values of the sandbox information were collected, the lists were compared, and if they matched exactly, they were regarded as the same.

- *Host-based Sandbox Detection*

All public MSASs were able to be detected by host-based sandbox detection, but detection with the Windows OS product key failed four times in System 6, and detection with the MAC address never succeeded in System 3. In addition, in System 7, the Windows OS install date between all of the experiment periods was not able to be collected.

In System 6, since the sample collected the information not observed in the disclosing sandbox information phase, the system was not able to be detected in four of 10 times. But the number of unique values of the new information is only one, and it is considered here that if the period of the disclosing sandbox information phase is lengthened, all of the values of Windows OS install date can be disclosed. In System 3, the MAC address was able to be collected every time in the disclosing sandbox information phase and sandbox-detection phase, but the MAC address values differed. This characteristic clearly differs from other systems. For this reason, it is considered here that the MAC address is changed at every analysis or using the API hooking in System 3 for evading sandbox detection changes the return value of the Windows API.

- *Network-based Sandbox Detection*

In Systems 3-7, the results of network-based sandbox detection with the information except for the IP address are the same as those of host-based detection. Since System 8 used an isolated sandbox, network-based sandbox detection against that system never succeeded. In Systems 5, 7, and 10-15, the systems were always able to be detected based on their IP address. However, detection sometimes failed with the IP address in Systems 3, 6, and 9. In addition, System 4 was never able to be detected.

In Systems 3 and 9, since the newly observed IP addresses are within /28 subnet, it is considered that the system could easily be detected using /28 subnet. Next, in System 4, since the IP address was changed every other day within /28 subnet, it was never detected using the list of IP addresses observed in the disclosing sandbox information phase. The IP addresses were investigated and it was found that they are part of the IP address space assigned to a certain Internet service provider (ISP). It was therefore considered that System 4 uses a dynamic IP address assignment by the ISP for Internet connection, and changed the IP address every other day by stopping and reinitiating a Point-to-Point Protocol over Ethernet (PPPoE) session for evading sandbox detection with the IP address. Finally, in System 6, many IP addresses that varied in both country and area were observed. The IP addresses were investigated and it was found that they were included in the IP address list of Onion routers [93]. It was therefore considered that System 6 uses TOR [79] for Internet connection in order to evade sandbox detection based on the IP address.

Table 5.3 Summary of Experiments About Sandbox-detection Phase

No.	Type of public MSAS	Speculated Sandbox Type	# of Submitted Test Samples or URLs	# of Received Analysis Reports	# of Accesses to Colluding Servers	Results of Host-based Detection			Results of Network-based Detection			
						pk	mac	insd	pk	mac	insd	IP Addresss
1	public MSAS-F	Isolated										
2	public MSAS-F	Isolated										
3	public MSAS-F	Internet-connected	10	10	10	10/10	0/10	10/10	10/10	0/10	10/10	7/10
4	public MSAS-F	Internet-connected	10	10	10	10/10	10/10	10/10	10/10	10/10	10/10	0/10
5	public MSAS-F	Internet-connected	10	10	10	10/10	10/10	10/10	10/10	10/10	10/10	10/10
6	public MSAS-F	Internet-connected	10	10	10	6/10	10/10	10/10	6/10	10/10	10/10	5/10
7	public MSAS-F	Internet-connected	10	10	10	10/10	10/10		10/10	10/10		10/10
8	public MSAS-F	Isolated	10	10	0	10/10	10/10	10/10				
9	public MSAS-W	Internet-connected	10	10	10							4/10
10	public MSAS-W	Internet-connected	10	10	20							20/20
11	public MSAS-W	Internet-connected	10	10	40							40/40
12	public MSAS-W	Internet-connected	10	10	10							10/10
13	public MSAS-W	Internet-connected	10	10	10							10/10
14	public MSAS-W	Internet-connected	10	10	10							10/10
15	public MSAS-W	Internet-connected	10	10	20							20/20

5.7.5 Summary

This experiment confirmed that disclosing sandbox information via an analysis report is effective against the existing public MSASs, and against not only public MSASs with an Internet-connected sandbox but those with an isolated sandbox. It was also clarified that analysis of public MSASs can be evaded by using the sandbox information that has properties described in Section 4.5, such as the IP address, Windows OS product key, MAC address, and Windows OS install date.

It was also confirmed that some systems made countermeasures against sandbox detection based on the IP address and MAC address. However, in these systems, we were able to detect by using other sandbox information, and it is insufficient as a countermeasure against a decoy injection attack.

5.8 Discussion

5.8.1 Countermeasures Against Decoy Injection Attacks

In order to prevent sandbox detection based on an IP address, the IP address of the sandbox should be changed frequently. However, since the operation costs of decoy injection attacks are so low, it can be assumed that attackers conduct them frequently enough to disclose and blacklist all used IP addresses. For preventing such straightforward blacklisting by the attacker, a time-sharing of the IP addresses can be deployed with real production systems. If an attacker mistakenly blacklists an IP address of a production system infected by malware, it will lose an opportunity to control that system. In fact, from the evaluation results it can be assumed that System 4 is leveraging a dynamic IP address assignment by a commercial ISP; thus, in this experiment, it was never possible to detect System 4 by using the sandbox's IP address. However, there are drawbacks in using the ISP-provided addresses. First, ISPs may also filter specific traffic and since the means of filtering cannot be controlled the analysis results may not be as reliable. Another solution is use of anonymity networks like TOR. In fact, in this experiment it was observed that System 6 used TOR, and the sandbox sometimes was not detected. However, while this seems to be a perfect solution for a decoy injection attack with an IP address, as mentioned in Section 4.6.1, careful consideration needs to be taken from the attacker's viewpoint.

In order to prevent sandbox detection based on sandbox information such as MAC address, Windows OS product key, and Windows OS install date, the sandbox information should also be changed frequently. One solution is use of API hooking for changing the return value of the Windows API which collects the sandbox information.

5.8.2 Possibility of Disclosing Sandbox Information via Analysis Report

A previous study only revealed the possibility of disclosing sandbox information via network access. This study evaluated the possibility of disclosing sandbox information via not only network access but also analysis report. The simple countermeasure against disclosing sandbox information via analysis report is to reduce the information about malware behaviors included in the analysis report. However, care needs to be taken because the reduction leads to a decrease in users' observability.

5.8.3 Comparison with Other Sandbox Detection Methods

In comparison with decoy injection attacks and other sandbox-detection methods such as virtual machine detection and debugger detection [58] [59] [60] [61], the advantage of other sandbox-detection methods is that they can detect all sandbox systems that have target characteristics such as the existence of a virtual machine or debugger. However, not only sandbox systems but also normal users' environments use virtual machines, and the case leads to a false positive. On the other hand, the decoy injection attack's advantage is that because it uses the sandbox information of a specific public MSAS, if using information that has a high level of uniqueness, the false positive rate becomes lower than with other methods. Recently, because using public MSASs has grown more popular, the ability to evade their analysis is a big advantage for attackers. The decoy injection attack's disadvantage is that it never detects sandbox systems that an attacker does not know. But decoy injection attacks can still be used not only on public MSASs but also other services. For example, online scanning service [43] and malware sample-sharing services [44] can also be attack targets.

Chapter 6

Malware Detection Based on Behavioral Differences Between Multiple Executions

6.1 Introduction

Recently, malware authors have been embedding malware with functions for countermeasures against malware analysis and detection. For example, when certain malware is executed, it detects debuggers or virtualization systems that are typically used in malware analysis, and changes or stops their behaviors in order to disrupt analyses. Another example is when malware attacks another computer it obfuscates exploit codes for avoiding detection by an intrusion detection system (IDS). Among them is a type of malware that changes its runtime behaviors in each execution to evade malware analysis and detection. For example, when malware copies itself on a file system, it can randomly determine its file name in order to avoid detection. Another example is that when malware tries to connect its C&C server, it randomly creates or chooses a domain name from a hard-coded domain name list to avoid being blocked via a static blacklist of malicious domain names. The most well-known example of this type is *Conficker*, a pandemic malware since 2009, which generates a to-be-accessed domain name by using a pseudorandom number generator [94]. It was assumed that such evasive behaviors are unnecessary for benign software. Therefore, these behaviors can be clues for distinguishing malware from benign software.

In this chapter, we propose a novel behavior-based malware-detection method that focuses on such characteristics. The proposed method conducts dynamic analysis on an executable file

multiple times in the same sandbox environment so as to obtain multiple logs of API call and traffic, and then compares them to find the difference between the multiple executions. If attackers try to evade the proposed detection method, they have to use deterministic malware, and then (especially dynamic) analysis is made easier. In experiments with 5,697 malware samples captured in the wild and 819 benign software samples collected from a Windows 7 host, Windows XP host, and free-software download site, about 70% of malware samples can be detected and the false positive rate is about 1%. In addition, about 50% of malware samples that the antivirus software engines did not detect can be detected. The possibility of the proposed method being able to improve the accuracy of malware detection utilized in combination with other existing methods is therefore confirmed.

The rest of this chapter is organized as follows. Section 6.2 gives related works. Section 6.3 first covers the proposed method and then the situation in which the method can be deployed. Section 6.4 explains experiments for the evaluation. Section 6.5 discusses the challenges of the proposed method, and Section 6.6 summarizes the chapter.

6.2 Related Works

Balzarotti et al. [6] proposed a method for detecting malware when it behaves differently in an emulated analysis environment and on an un-instrumented reference host. A malware sample is executed in a reference host in which all system calls and their return values are logged. Second, the malware sample is executed again in a different analysis system in which every system call returns the previously logged values to precisely replay the execution of the malware sample. The analysis system then compares the runtime behaviors of the malware sample in the analysis system and the reference host for detecting their difference.

Kolbitsch et al. [95] proposed a behavior-based malware-detection method at the end host. They first analyze a malware program in a controlled environment to build a model that characterizes its behavior. Such models describe the information flows between the system calls essential to the malware mission. Then, they extract the program slices responsible for such information flows. For detection, these slices are executed to match the models against the runtime behavior of an unknown program.

Yoshioka et al. [96] proposed a network-based detection method of malware-infected hosts. They focus on malware that continues listening on a port to communicate with other hosts. Their method generates signatures by sending test inputs to the port to which malware executed in a sandbox is listening and observing its response.

Sakai et al. [97] focused on detecting malware with repetitive behavior in its infection and

propagation. They detect the repetitive infection behavior (e.g., copying itself and registering it for auto run) of malware when its execution environment is recovered to the pre-infection state.

Matsuki et al. [68] proposed an anti-malware technique in which they focus on malware that kills processes of antivirus software and firewall for avoiding detection. In the proposed method, they execute decoy processes and when detecting a process that tries to kill the decoy processes, they consider it a malware process and kill it.

6.3 Proposed Method

Figure 6.1 Flowchart of Proposed Method shows the brief procedure of the proposed method.

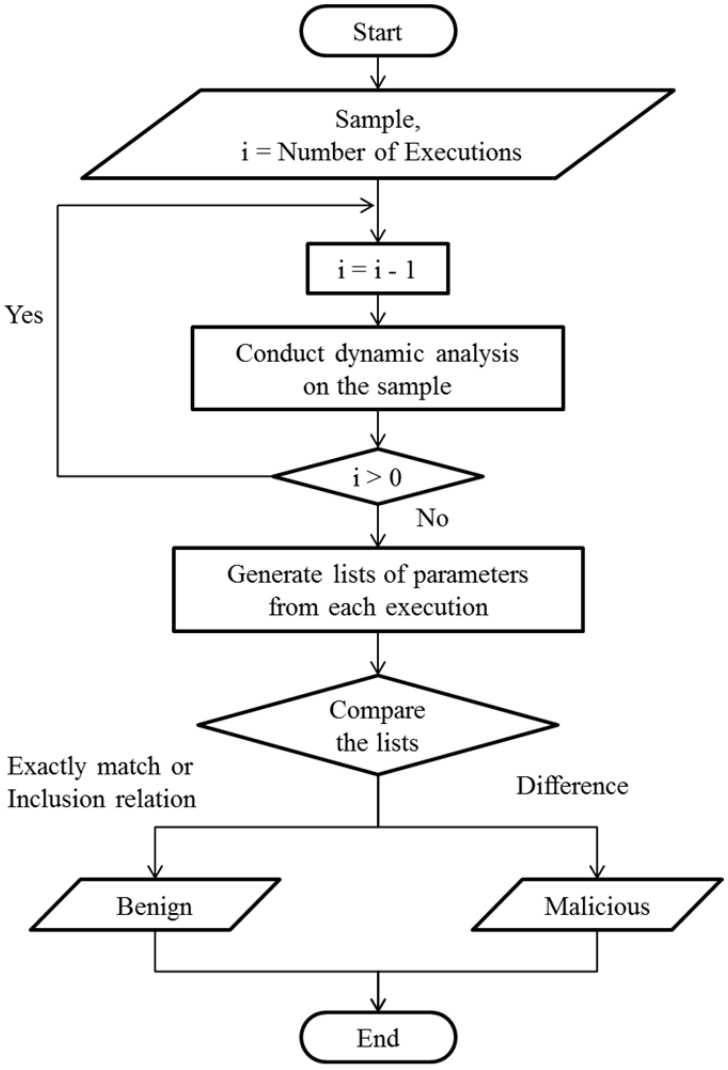

Figure 6.1 Flowchart of Proposed Method

First, a sample (i.e., executable file) and the number of executions are input. There is a trade-off between *accuracy* and *efficiency*. Although accuracy will improve by increasing the number of executions, efficiency will also degrade because of the longer inspection time.

Second, dynamic analysis is conducted on the samples multiple times in the same sandbox environment to obtain the lists of API calls.

Third, lists of parameters used for a predefined set of API calls in Table 6.1 are generated. The table shows the set of API calls and their parameters used in the proposed method.

Table 6.1 APIs and Their Parameters

API	Parameter
RegSetValueEx	Subkey Name
RegSetValue	Subkey Name
CreateFile	File Name
LZOpenFile	File Name
_lcreat	File Name
CreateDirectoryEX	Directory Path
CreateDirectory	Directory Path
CopyFileEx	New File Name
CopyFile	New File Name
LZCopy	New File Name
MoveFileEx	New File/Directory Name
MoveFile	New File/Directory Name

Based on experiences with malware analysis, the following behaviors are regarded as *possibly randomized behaviors*.

- File-related behaviors (i.e., file name at the time of creating/moving/copying a file)
- Registry-related behaviors (i.e., entry name at the time of registration of the Run key)
- Network-related behaviors (i.e., domain name/IP address of remote host at the time of network access)

Then, APIs are selected and their parameters, as shown in Table 6.1, related to the above behaviors. The following Run keys related to automatic program execution on Windows XP are also selected.

- *HKEY_LOCAL_MACHINE¥Software¥Microsoft¥Windows¥CurrentVersion¥Run*
- *HKEY_CURRENT_USER¥Software¥Microsoft¥Windows¥CurrentVersion¥Run*
- *HKEY_LOCAL_MACHINE¥Software¥Microsoft¥Windows¥CurrentVersion¥RunOnce*

- *HKEY_CURRENT_USER¥Software¥Microsoft¥Windows¥CurrentVersion¥RunOnce*

A related work [6] reports that even if an executable file that has not randomized behaviors in the same sandbox environment is executed twice, these orders of the API calls and their duplication do not necessarily become the same because of the return value of the API call, etc. To solve the problem, this work records the API calls and its parameters of the program and then replays it the next time the program is executed. However, because the replay of API calls is a complex operation, there is a concern about versatility. Therefore, the proposed method simply ignores the order of the API calls and network accesses and their duplication. In fact, in the experiments, it is confirmed that the false positive rate increased when the order of the API calls and network accesses are not ignored. The name of the file and directory that is created in the temporary folder (*C:¥Document and Settings¥{user name}¥Local Settings¥Temp¥*) is also ignored.

Finally, the lists obtained from each execution are compared, and if they are an exact match or there is a relation of inclusion, the sample is determined as *benign software*. Otherwise, it is determined as *malicious software*.

6.4 Evaluation

Experiments were conducted to evaluate the proposed method using 5,697 malware samples and 819 benign software samples. First, Section 6.4.1 explains the samples. Section 6.4.2 then covers the malware dynamic analysis system used in the experiments. Section 6.4.3 explains the results.

6.4.1 Malware Samples and Benign Software Samples

The experiments used 5,697 malware samples captured in the wild by using honeypots, client honeypots, etc. Table 6.2 is a list of the top 20 malware names of the samples. A total of 448 names of the samples were obtained from *Symantec*, and 881 from *McAfee*. Table 6.2 summarizes the samples into the same families.

Also used were 819 benign software samples collected from a Windows 7 host, Windows XP host, and free-software download site [99]. In the free-software download site, some software is provided as an installer, and in such cases the installer was used for evaluation. Note that each

sample is imported to the analysis system in the form of a single file and no other related files were imported to the analysis system even if they existed.

Table 6.2 Top 20 Names of Malware Samples

(a) Symantec

Malware Name	# of samples
W32.Virut family	2036
W32.Spybot.Worm	656
W32.Korgo family	508
W32.Rahack family	301
W32.Pinfi	269
Backdoor family	254
W32.Sality family	252
W32.IRCBot	242
W32.Gobot.A	160
Hacktool family	148
Suspicious.IRCBot	67
Infostealer family	50
W32.Bobax!dr	50
Trojan Horse	40
W32.SillyFDC family	34
W32.Pilleuz family	32
Packed.Generic family	29
W32.Sasser family	29
Trojan.Gen family	24
W32.Linkbot family	21
Others	495

(b) McAfee

Malware Name	# of samples
w32/virut family	2293
w32/sdbot.worm family	447
artemis! family	391
w32/rahack	307
w32/pate family	297
w32/korgo.worm family	291
backdoor family	232
w32/sality family	225
exploit-mydoom	152
generic.dx family	137
generic pws family	99
w32/bobax family	98
generic dropper family	65
downloader family	50
generic backdoor family	49
generic malware family	28
w32/gael.worm.a	24
w32/rimecud family	23
w32/ircbot family	21
fakealert family	17
Others	451

6.4.2 Malware Sandbox Analysis System

The experiments use the malware dynamic analysis system called the *Micro analysis System* [12] developed at Japan's *National Institute of Information and Communications Technology (NICT)*. The behaviors regarded as possibly randomized are usually observed at an early stage of infection. A previous study [100] showed that execution for 60 seconds is sufficient for observing such behaviors; so this experiment executed each sample for 60 seconds in the execution environment (*victim host*) with Windows XP SP2 and collected the API call sequence and the traffic log.

The feature of the Micro analysis System is that it executes a sample on a real machine, and it is not disturbed by anti-analysis malware with emulator detection and virtual machine detection capabilities. The system is also isolated from the real Internet and only connected to the Internet emulator, which provides various network services to the victim host. Using the Internet emulator

for the proposed method is suitable since it allows a more stable and controlled network environment to be provided for precise behavior comparison.

6.4.3 Results

6.4.3.1 True Positive and False Positive

Table 6.3 shows the results of the experiment with the malware samples. Note that in Table 6.3 and Table 6.4, the *All* row indicates the result of the proposed method using all API parameters in Table 6.1 and all hostnames (or IP addresses). The *Registry* row indicates the result of the proposed method using the API parameters related to registry activity (i.e., *RegSetValueEx* and *RegSetValue*), the *File* row shows the result with the API parameters related to file activity (i.e., *CreateFile*, *LZOpenFile*, *lcreat*, *CreateDirectoryEx*, *CreateDirectory*, *CopyFileEx*, *CopyFile*, *LZCopy*, *MoveFileEx*, and *MoveFile*), and the *Network* row shows the result with hostnames (or IP addresses).

Table 6.3 Experiment with Malware Samples

		True Positive	False Negative	TP Rate
All		3864	1833	67.83
	Registry	478	5219	8.39
	File	3799	1898	66.68
	Network	2018	3679	35.42

Table 6.4 Experiment with Benign Software Samples

		False Positive	True Negative	FP Rate
All		13	806	1.59
	Registry	0	819	0.00
	File	12	807	1.47
	Network	1	818	0.12

As shown in the *All* row, the proposed method detected 3,962 of the 5,697 malware samples (true positive rate = 69.55%). In other words, in the experiment about two-thirds of all malware changed its runtime behaviors in each execution. This malware has 220 malware names from Symantec and 359 from McAfee. Since the detected samples cover various types of malware such as *worms*, *spyware*, *adware*, Trojan horses, and *bots*, it is shown that randomized behavior focused

on here is a common characteristic that a great deal of malware shares.

More than 66% of the malware samples were detected as having randomized file activity. Table 6.5 shows the file extensions of the files the samples created. In the malware samples that exhibited randomized behavior in file activity, 96% (3,642) were detected through the name of the created executable files. Since they actually copy themselves on a file system with a name that is generated in each execution, the proposed method can detect them with a high level of accuracy. Some samples also used a benign-looking file name such as firewall.exe, iexplore.exe, or logon.exe. Since these samples seem to contain a list of file names from which one was randomly selected, the created file name was different in each execution, and the proposed method can highly accurate at detecting them. Among the samples that were detected with randomized file activity, 90% (3,419) also created a copy of themselves in the directory (*c:¥windows¥system32¥*), and some created a copy of themselves in other directories (*c:¥windows¥*, *c:¥documents*, *settings¥user¥local settings¥application data¥*, etc.). Some created a batch file to execute the copied file and delete the original file, and the method can also detect the batch files whose names were randomly determined.

Table 6.5 File Extensions of Detected Files Named Randomly by Malware Samples

File Extension	# of samples
exe	3643
nothing	46
bat	38
ini	35
com	8
dll	6
tmp	4
sys	4
dat	3
zip	1
vir	1
txt	1
rdf	1
log	1
jtp	1
job	1
fon	1
exs	1
exm	1
exk	1
bmp	1

About 56% of malware samples were detected through random network behavior. Among them, 67% (2,140 samples) were detected by their random access toward a private network without DNS name resolution. Almost all of them accessed 445/TCP and these seem to have conducted network scanning for propagating infection within a private network. Since the target IP addresses are randomly selected upon each execution, the proposed method can detect with a high level of accuracy. Eighteen percent (579 samples) were detected through their random access to global IP addresses without DNS name resolution. Most accessed 445/TCP and these seem to have conducted network scanning for propagating infection. Some of them also accessed many hosts by using a high port number and these seem to have conduct P2P communication for receiving a C&C message. Fifteen percent (472 samples) were detected through their random DNS access. Almost all these communications are a name resolution for connecting a C&C server and some communication seems to be for sending spam or confirming the connectivity.

On the other hand, only about 8% were detected through randomized registry activity.

Table 6.4 shows the results of the experiment with benign software samples. As shown in the *All* row, 13 of 819 benign software samples were falsely detected as malware (false positive rate = 1.59%). Most of these were detected by using the API parameters related to file activity. These software samples created, for instance, an RSA key container and a customized temporary file outside the default temporary folder, which caused the false detection.

6.4.3.2 Execution Time

Figure 6.2 shows the cumulative percentage of detected malware with respect to its execution time. The x-axis shows the execution time when the detected behavior was observed. The y-axis shows the cumulative percentage. Almost all detected malware showed the randomized behavior within 10 seconds. This result shows that it is enough to execute each sample for 60 seconds for effective detection.

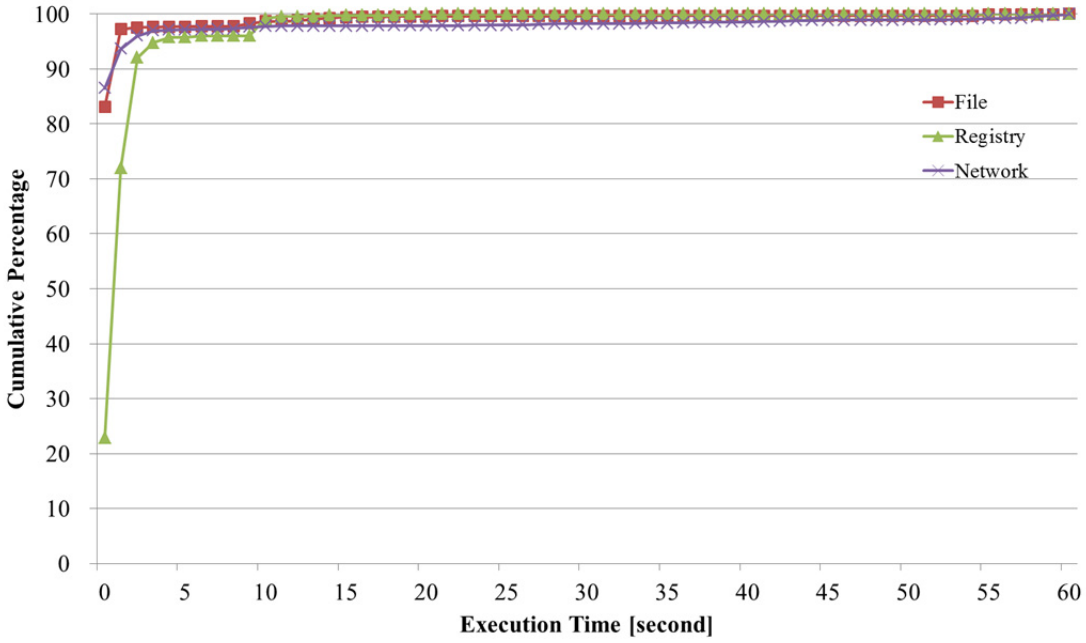

Figure 6.2 Cumulative Percentage of Detected Malware with Respect to Their Execution Time

6.4.3.3 Increase in the Number of Executions

Table 6.6 and Table 6.7 respectively show the result of transition of true positive rate and false positive rate caused by increasing the number of executions. In *three-time executions* and *four-time executions*, lists are compared in all possible pairs and a sample is determined as malware if the lists are different in even one pair. As shown in Table 6.6, the true positive rates (*TP_All*, *TP_Registry*, *TP_File*, and *TP_Network*) improved slightly as the number of executions increased. On the other hand, the false positive rates *(FP_All* and *FP_File*) deteriorated as the number of executions increased from two to three times.

Consequently, execution of a sample twice seems suitable for the proposed method because the true positive rate increased little even if the number of executions increases to more than two.

Table 6.6 Transition of TP Rate

	Twice	Three Times	Four Times
TP_All	69.55	70.65	71.27
TP_Registry	8.39	9.67	10.22
TP_File	66.68	67.42	67.77
TP_Network	56.01	58.03	58.57

Table 6.7 Transition of FP Rate

		Twice	Three Times	Four Times
FP_All		1.59	1.95	1.95
	FP_Registry	0.00	0.00	0.00
	FP_File	1.47	1.83	1.83
	FP_Network	0.12	0.12	0.12

6.4.3.4 Combination with Antivirus Software

In the above experiments, the proposed method can detect about 70% of malware samples. This true positive rate is not sufficient for it to be used alone. Therefore, it seems that using the method in combination with existing methods is a good way of improving detection capability. Therefore, this section discusses combination with antivirus software in a situation using in-cloud detection such as CloudAV.

File scan reports of about 5,697 malware samples were searched by sending queries to *virustotal*, and Table 6.8 shows the result of the search. Note that since searching was conducted not by sending sample files but sending hash values of sample files, the scan reports on malware samples that have never previously been submitted to the virustotal were not able to be obtained and the scan reports are not up to date. In Table 6.8, each row indicates a scan result when using an antivirus software engine. In addition, the *Detected* column indicates the number of malware samples that were detected by the antivirus software, the *Not Detected* column indicates those not detected, and the *No Result* column indicates the number of malware samples that we were not able to obtain the scan report about it.

Table 6.8 Scan Results of virustotal

No.	Detected	Not Detected	No Result
1	4644	205	848
2	4563	284	850
3	4527	327	843
4	4453	398	846
5	4449	389	859
6	4446	377	874
7	4407	405	885
8	4394	451	852
9	4384	470	843
10	4380	358	959
11	4303	527	867
12	4282	549	866
13	4234	421	1042
14	4221	631	845
15	4119	103	1475
16	3974	881	842
17	3954	231	1512
18	3866	902	929
19	2905	510	2282
20	2806	705	2186

First, a situation was assumed that uses the proposed method with one antivirus software engine and confirmation was made on whether the proposed method detects malware samples that were not detected by the antivirus software engines. Figure 6.3 shows that the rates of detection by the proposed method in the samples of Not Detected. The x-axis shows each antivirus software engine, and the y-axis shows the rate of detection. As shown in Figure 6.3, the proposed method was able to detect 22% to 74% (average ≈ 50%) malware samples that were not able to be detected by the antivirus software engines.

The No Result samples are probably recent malware since they have never been submitted to virustotal in the past, and they should be used for evaluation of the proposed method. However, because of the policy of malware sample management, only the hash values of the samples, not the actual samples themselves, can be sent to virustotal in order to obtain the updated scan reports. Consequently, we were not able to confirm whether the samples are detected by those antivirus software engines or not. Therefore, we conducted offline scan of the No Result samples using the available antivirus software. There are 221 samples that the updated antivirus software was not able to be detected, and 118 of them were able to be detected by the proposed method. The result shows that the proposed method can improve the accuracy of malware detection when utilized in

combination with antivirus software.

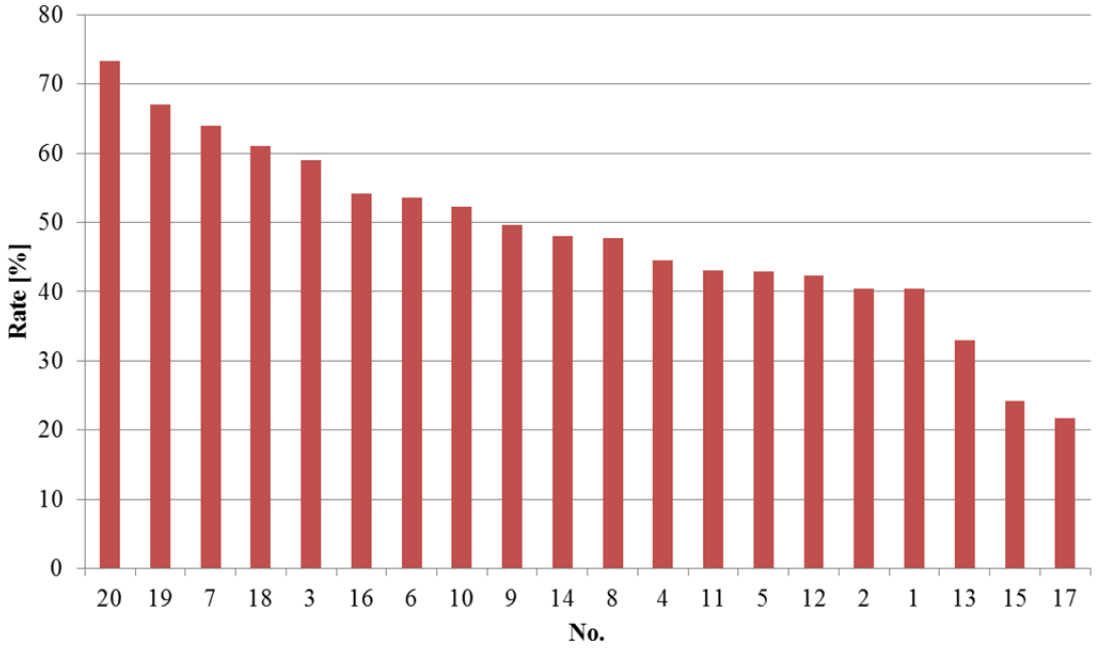

Figure 6.3 Detection Rates by Proposed Method Not Detected Samples

6.5 Discussion

6.5.1 Comparison with Other Detection Methods

Table 6.9 shows a comparison with related works. One related work [6] was omitted from the table since it is not a method for distinguishing malware from benign software but for distinguishing malware that changes its behaviors in accordance with the execution environment from other malware.

The advantage of the proposed method is that it does not need any training phases for creating signatures. Two related works [95] [96] need training phases and have a risk of their detection rate being strongly influenced by the quality of training datasets. These learning-based methods also usually need relearning and updating of signatures. Though it is difficult to make a fair comparison between the detection rates because of the differences of malware sample sets, the method was evaluated by using many more malware samples than other related works and successfully detected about 70% of them. The disadvantage of the proposed method is that because it conducts dynamic

analysis on an executable file multiple times, it is more costly to operate at the time of detection than the related works. However, conducting dynamic analysis in parallel and using the whitelist of benign software can reduce the operation cost.

Table 6.9 Comparison with Other Detection Methods

	Proposed method	Kolbitch [95]	Yoshioka [96]	Sakai [97]	Matsuki [68]
Target Characteristics	behavioral differences between multiple executions	system call graph	responses from malware-infected hosts	repetitiveness of intrusion	killing processes of security programs
Application	host-based detection	host-based detection	network-based detection	host-based detection	host-based detection
Detection Rate	3963 / 5697 samples	168 / 263 samples	121 / 434 samples	107 / 149 samples	19 / 3385 species
False Positive Rate	13 / 806 samples	0 / 7 samples	0 / 33 services	0 / 15 samples	-
Training Phase	not necessary	necessary	necessary	not necessary	not necessary
Operation Cost	high	low	low	medium	medium

6.5.2 Application

The proposed detection method is slow since it requires multiple executions of an executable file, so it is not suitable for real-time detection at an end host, such as antivirus software. Therefore the proposed method seems suitable for in-cloud detection.

For example, use of public malware dynamic analysis systems that receive online submissions of possibly malicious files or URLs from an arbitrary user, analyze their behavior by executing or visiting them via a sandbox, and send analysis reports back to the user, have grown increasingly popular [2] [18] [27] [32]. In these systems, the proposed method can be utilized for determination of whether or not a sample is malicious.

Another example is *Oberheide et al*. [98], who proposed an in-cloud antivirus system called *CloudAV*, which contains a lightweight, cross-platform host agent and a network service with 10 antivirus engines and two behavioral detection engines. Instead of running complex analysis software on all end software, they suggest that each end host run a lightweight process to detect new files, send them to a network service for analysis, and then permit access or quarantine them based on a report returned by the network service. In such a system, the proposed method can also be utilized as a behavioral detection engine. A similar system was proposed in [14]. That system's goal is to automatically generate the removal tool that removes/repairs files and registries that malware created/modified. In generating a removal tool it is extremely helpful to figure out malware evasive behaviors using the proposed method.

6.5.3 Temporary File

The experiments ignore the name of the file and directory created in the temporary folder (*C:¥Document and Settings¥{user name}¥Local Settings¥Temp¥*) because a great deal of benign software creates a temporary file in a temporary folder.

Using the name of the file and directory created in the temporary folder also increases the false positive result rate to about 17%. Malware behavior that creates a file in a temporary folder cannot be detected if the temporary file is ignored, but it seems rare for malware to create an important file such as a copy of itself in a temporary folder since a file in a temporary folder is usually deleted at the time of exiting the program and the Windows OS disk cleanup tool also deletes such files.

Chapter 7

Conclusion

7.1 Conclusion

Recently, malware has played a critical role as the infrastructure of cyber-attacks. With its recent explosive increase, it is becoming nearly impossible to manually analyze all malware using reverse engineering techniques. Malware sandbox analysis has been studied widely for tackling this problem. However, malware authors have been embedding new functions for evading such analysis. For example, malware such as bots do not work unless they meet conditions for activation, and it is difficult to analyze them sufficiently using traditional sandbox analysis. Another example is malware that changes its runtime behaviors in each execution to evade malware analysis and detection. In this thesis, we researched countermeasures to such malware with sandbox-evasive behaviors.

In Chapter 3, we described techniques performed by malware and malware authors for evading analysis and detection, and categorized evasion techniques against sandbox analysis into two approaches: a technique to make comprehension of malware behaviors more difficult and a technique to detect a sandbox.

Against the former techniques, in Chapter 4, we proposed a novel sandbox analysis method that realizes better observability and efficiency against malware that possesses techniques to make it harder to recognize. The method focuses on a function of malware that changes its behaviors in accordance with the behaviors of remote servers with which it interacts, such as C&C and malware download servers, and analyzes the server behaviors and corresponding malware behaviors. In the proposed method, first, a malware sample is executed in the sandbox that is only connected to the emulated Internet. Then the traffic observed in the sandbox is inspected and high-risk

communications are filtered out. The dummy client, an automatically generated script to interact with remote servers instead of the actual sample, then uses the rest of the traffic data. The dummy client is then continually executed in environments with a real Internet connection and effectively collects the responses from remote servers. The collected server responses are then fed back to the Internet emulator in the sandbox so that the next time the sample is analyzed a dummy server in the Internet emulator can respond in accordance with the responses. The proposed method was evaluated with samples captured in the wild and was able to observe a greater variety in their behaviors, such as receiving C&C commands, downloading new executables, and performing port scans.

In Chapter 5, we clarified targeted sandbox-detection vulnerability in public Malware Sandbox Analysis Systems (public MSASs) for pursuing better observability. A previous study pointed out the vulnerability of public MSASs against decoy injection attacks, in which an attacker detects the sandbox based on its IP address disclosed by submitting a decoy sample. However, the possibility of detection using sandbox information other than the IP address was not discussed in detail. Therefore, we defined properties of sandbox information for decoy injection attack: stability, uniqueness, and stealthiness of collection. Then, we evaluated 16 different kinds of characteristic information of the sandbox for its detection. Experiments with real public MSASs in operation resultantly found that characteristic information such as the Windows OS product key, MAC address, and Windows OS install date can be utilized for sandbox detection. Moreover, besides network-based disclosure, we showed that such characteristic information of the sandbox can be disclosed via an analysis report provided to the user, which means that the decoy injection attack can be performed against a sandbox isolated from the real Internet. Thus, this study confirmed broad applicability of the decoy injection attack as well as the need for comprehensive countermeasures. We also discussed countermeasures against decoy injection attacks.

We also indicated a direction on how to develop a countermeasure technique against evasive malware without an attacker evading it. Differences between malware and benign software that come from the malware's mechanism of evading analysis/detection should be leveraged; that is, when proposing a new analysis method, the means of detecting malware that evades the analysis method should be considered in advance. Consequently, the attackers are given fewer choices. As an example of the concept, in Chapter 6, we proposed a novel behavior-based malware-detection method using sandbox-evasive behaviors. Recently, malware authors have been embedding functions for countermeasure against malware analysis and detection. Accordingly, modern malware often changes its runtime behaviors in each execution to avert malware analysis and detection. The proposed method focuses attention on such characteristics. The method conducts dynamic analysis on an executable file multiple times in the same sandbox environment so as to obtain multiple logs of API call and traffic, and then compares them to find the difference between

the multiple executions. If attackers try to evade the proposed detection method, they have to use deterministic malware, and then (especially dynamic) analysis is made easier. Experiments with malware samples captured in the wild and benign software samples showed effectiveness of the method.

7.2 Future Work

There are many studies on the topics of malware analysis and detection. Unfortunately, malware authors have now developed evasion techniques against most existing analysis/detection methods. Future work, as indicated in this dissertation, should continue developing countermeasures that leverage differences between malware and benign software that come from malware's methods of evading analysis/detection in order to give attackers fewer choices and make creating malware more costly. Furthermore, since cyber-attacks have become more complicated and sophisticated with wide variety of means such as DBD attack and social engineering, information gained from sandbox analysis is not sufficient for grasping the actual state of cyber-attacks. To comprehend overall cyber-attacks, we should conduct multimodal analysis based on observations from diverse sensors such as sandbox analysis, various honeypots, and network monitoring.

Bibliography

[1] "Aguse," http://www.aguse.jp.

[2] "Anubis: Analyzing Unknown Binaries," http://anubis.iseclab.org/.

[3] "Autovin," http://autovin.pandasecurity.my/.

[4] "Automated Malware Analysis - Joe Security LLC Home," http://www.joesecurity.org/.

[5] M. Bailey, J. Oberheide, J. Andersen, Z. M. Mao, F. Jahanian , J. Nazario, “Automated Classification and Analysis of Internet Malware,” *In Proceedings of the 10th international conference on Recent advances in intrusion detection* (*RAID 2007*), pp. 178-197, September 2007.

[6] D. Balzarotti, M. Cova, C. Karlberger, C. Kruegel, E. Kirda , G. Vigna, “Efficient Detection of Split Personalities in Malware,” *In Proceedings of the 17th Annual Network and Distributed System Security Symposium* (*NDSS 2010*), February 2010.

[7] U. Bayer, P. M. Comparetti, C. Hlauschek, C. Kruegel , E. Kirda, “Scalable, Behavior-Based Malware Clustering,” *In Proceedings of the 16th Annual Network and Distributed System Security Symposium* (*NDSS 2009*), February 2009.

[8] U. Bayer, I. Habibi, D. Balzarotti, E. Kirda , C. Kruegel, “A View on Current Malware Behaviors,” *In Proceedings of the 2nd Usenix Workshop on Large-Scale Exploits and Emergent Threats* (*LEET 2009*), April 2009.

[9] U. Bayer, C. Kruegel , E. Kirda, “TTAnalyze: A Tool for Analyzing Malware,” *In Proceedings of the 15th Annual Conference of the European Institute for Computer Antivirus Research* (*EICAR 2006*), May 2006.

[10] "BitBlaze: Binary Analysis for Computer Security," http://bitblaze.cs.berkeley.edu/.

[11] "Comodo Instant Malware Analysis," http://camas.comodo.com/cgi-bin/submit.

[12] D. Inoue, K. Yoshioka, M. Eto, Y. Hoshizawa , K. Nakao, “Automated Malware Analysis System and its Sandbox for Revealing Malware's Internal and External Activities,” *IEICE Transactions on Information and Systems*, Vol. E92-D, No. 5, pp. 945-954, May 2009.

[13] D. Inoue, K. Yoshioka, M. Eto, Y. Hoshizawa , K. Nakao, “Malware Behavior Analysis in Isolated Miniature Network for Revealing Malware's Network Activity,” *In Proceedings of the 2008 IEEE International Conference on Communications* (*ICC 2008*), pp. 1715-1721, May 2008.

[14] N. Kawaguchi, T. Yoda, H. Yamaguchi, T. Kasagi , Y. Hoshizawa, “Automatically Detecting and Removing Malware using Dynamic Analysis Systems,” *In Proccesings of the 14th International Symposium on Recent Advances in Intrusion Detection* (*RAID2010*), (Poster Session), September 2011.

[15] E. Kirda, C. Kruegel, G. Banks, G. Vigna , R. Kemmerer, “Behavior-based Spyware Detection,” *In Proceedings of the 15th Conference on USENIX Security Symposium*, July 2006.

[16] S. Miwa, T. Miyachi, M. Eto, M. Yoshizumi , Y. Shinoda, “Design and Implementation of an Isolated Sandbox with Mimetic Internet Used to Analyze Malwares,” *In Proceedongs of the DETER Community Workshop on Cyber Security Experimentation and Test 2007*, August 2007.

[17] A. Moser, C. Kruegel , E. Kirda, “Exploring Multiple Execution Paths for Malware Analysis,” *In Proceedings of the 2007 IEEE Symposium on Security and Privacy*, pp. 231-245, May 2007.

[18] "Norman Sandbox," http://www.norman.com/security_center/security_tools/.

[19] "Sandboxie - Sandbox software for application isolation and secure Web browsing," http://www.sandboxie.com/.

[20] "ThreatExpert - Automated Threat Analysis," http://www.threatexpert.com/.

[21] "ThreatTrack Security - Malware Research Labs," http://www.sunbeltsecurity.com/.

[22] "ViCheck.ca - Find embedded malware in documents, PDFs or emails," https://vicheck.ca/.

[23] C. Willems, T. Holz , F. Freiling, “Toward Automated Dynamic Malware Analysis Using CWSandbox,” *IEEE Security & Privacy*, Vol. 5, Issue 2, pp. 32-39, March 2007.

[24] K. Yoshioka , T. Matsumoto, “Multi-Pass Malware Sandbox Analysis with Controlled Internet Connection,” *IEICE Transactions on Fundamentals of Electronics, Communications and Computer Sciences*, Vol. E93-A, No.1, pp. 210-218, May 2010.

[25] K. Yoshioka, D. Inoue, M. Eto, Y. Hoshizawa, H. Nogawa , K. Nakao, “Malware Sandbox Analysis for Secure Observation of Vulnerability Exploitation,” *IEICE Transactions on Information and Systems*, Vol. E92-D, No. 5 , pp. 955-966, May 2009.

[26] K. Yoshioka, T. Kasama , T. Matsumoto, “Sandbox Analysis with Controlled Internet Connection for Observing Temporal Changes of Malware Behavior,” *In Proceedings of the 4th Joint Workshop on Information Security* (*JWIS 2009*), August 2009.

[27] ":: mwanalysis :: CWSandbox ::," http://mwanalysis.org/.

[28] "Evil Fingers: Sandbox Awareness," http://evilfingers.blogspot.jp/2009/01/sandbox-awareness.html.

[29] "OffensiveCOding updated – Emulation/AV Awareness," http://evilcodecave.wordpress.com/tag/cwsandbox/.

[30] "Detect 5 different sandboxes," http://www.opensc.ws/snippets/3558-detect-5-different-sandboxes.html.

[31] K. Yoshioka, Y. Hosobichi, T. Orii , T. Matsumoto, “Vulnerability of Malware Sandbox Analysis as an Online Service,” *In Proceedings of the 2009 Computer Security Symposium 2009* (*CSS2009*), (in Japanese), October 2009.

[32] K. Yoshioka, Y. Hosobuchi, T. Orii , T. Matsumoto, “Your Sandbox is Blinded: Impact of Decoy Injection to Public Malware Analysis Systems,” *IPSJ Journal*, Vol. 52, No. 3, pp. 1144-1159, March 2011.

[33] "Win32/Conficker," http://www.microsoft.com/security/portal/threat/encyclopedia/entry.aspx?Name=Win32%2fConficker.

[34] P. Baecher, M. Koetter, T. Holz, M. Dornseif , F. C. Freiling, “The Nepenthes Platform: An Efficient Approach to Collect Malware,” *In Proceedings of the 9th international conference on Recent advances in intrusion detection* (*RAID 2006*), pp. 165-184, September 2006.

[35] N. Provos, “A Virtual Honeypor Framework,” *In Porceedings of the 13th USENIX Security Symposium*, pp. 1-14, August 2004.

[36] "The Honeynet Project," http://www.honeynet.org/.

[37] Y. M. Wang, D. Beck, X. Jiang, R. Roussev, C. Verbowski, S. Chen , S. King, “Automated web patrol with strider honeymonkeys: Finding web sites that exploit browser vulnerabilities,” *In Proceedings of the 13th Annual Network and Distributed System Security Symposium* (*NDSS 2006*)*,* February 2006.

[38] M. Akiyama, M. Iwamura, Y. Kawakoya, K. Aoki and M. Itoh, "Design and Implementation of High Interaction Client Honeypot for Drive-by-download Attacks," *IEICE Transactions on Communication*, Vol. E93-B, No. 5, pp. 1131-1139, May 2010.

[39] H. Flake, “Structural Comparison of Executable Objects,” *In Proceedings of the IEEE Conference on Detection of Intrusions and Malware & Vulnerability Assessment* (*DIMVA*), pp. 161-173, July 2004.

[40] M. Iwamura, M. Itoh , Y. Muraoka, “Automatic Malware Classification System Based on Similarity of Machine Code Instructions,” *IPSJ Transactions of Information Processing Society*, Vol. 51, No. 9, pp. 1622-1632, September 2010.

[41] "Clam AntiVirus," http://www.clamav.net/lang/en/.

[42] "Snort," http://www.snort.org/.

[43] "VirusTotal - Free Online Virus, Malware and URL Scanner," https://www.virustotal.com/.

[44] "Open Malware | Community Malicious code research and analysis," http://www.offensivecomputing.net/.

[45] J. Gobel , T. Holz, “Rishi: Identify Bot Contaminated Hosts by IRC Nickname Evaluation,” *In Proceedings of the first conference on First Workshop on Hot Topics in Understanding Botnets* (*HotBots*), April 2997.

[46] "Safe Browsing API," https://developers.google.com/safe-browsing/?hl=ja.

[47] "SmartScreen Filter," http://windows.microsoft.com/ja-jp/internet-explorer/products/ie-9/features/smartscreen-filter

[48] Microsoft, "What we know (and learned) from the Waledac takedown," http://blogs.technet.com/b/mmpc/archive/2010/03/15/what-we-know-and-learned-from-the-waledac-takedown.aspx.

[49] "Taking Down Botnets: Microsoft and the Rustock Botnet," http://blogs.technet.com/b/microsoft_on_the_issues/archive/2011/03/18/taking-down-botnets-microsoft-and-the-rustock-botnet.aspx.

[50] Microsoft, "Microsoft, the FBI, Europol and industry partners disrupt the notorious ZeroAccess botnet," http://www.microsoft.com/en-us/news/press/2013/dec13/12-05zeroaccessbotnetpr.aspx.

[51] M. G. Kang, P. Poosankam and H. Yin, "Renovo: a hidden code extractor for packed executables," *In Proceedings of the 2007 ACM workshop on Recurring malcode*, pp. 46-53, November 2007.

[52] P. Royal, M. Halpin, D. Dagon, R. Edmonds , W. Lee, “PolyUnpack: Automating the Hidden-Code Extraction of Unpack-Executing Malware,” *In Proceedings of the Computer Security Applications Conference* (*ACSAC*), pp. 289-300, December 2006.

[53] Y. Hoshizawa, K. Okada , T. Tachikawa, “Extracting BOT commands automarically by dynamic analysis,” *In Proceedings of the anti Malware engineering WorkShop 2008* (*MWS 2008*), October 2008.

[54] "Win32/Rbot," http://www.microsoft.com/security/portal/threat/encyclopedia/entry.aspx?name=win32%2frbot.

[55] "Win32/Sdbot," http://www.microsoft.com/security/portal/threat/encyclopedia/entry.aspx?name=Win32%2fSdbot.

[56] "Win32.IRCBot," http://www.symantec.com/ja/jp/security_response/writeup.jsp?docid=2002-070818-0630-99.

[57] Y. Kawakoya, M. Iwamura, E. Shioji , T. Hariu, “API Chaser: Anti-analysis Resistant Malware Analyzer,” *In Proceedings of the 16th International Symposium on Research in*

Attacks, Intrusions and Defenses (*RAID 2013*), October 2013.

[58] X. Chen, J. Andersen, Z. M. Mao, M. Bailey , J. Nazario, “Towards an Understanding of Anti-virtualization and Antidebugging Behavior in Modern Malware,” *In Proceedings on the 38th Annual IEEE/IFIP International Conference on Dependable Systems and Networks* (*DSN 2008*), pp. 177-186, June 2008.

[59] T. Holz , F. Raynal, “Detecting Honeypots and other Suspicious Environments,” *In Proceedings of the 2005 IEEE Workshop on Information Assurance and Secrity*, pp. 29-36, June 2005.

[60] T. Raffetseder, C. Kruegel , E. Kirda, “Detecting System Emulators,” *In Proceedings of the 10th international conference on Information Security* (*ISC 2007*), pp. 1-18, 2007.

[61] J. Rutkowska, "Red Pill... Or How To Detect VMM Using (Almost) One CPU Instruction," http://www.securiteam.com/securityreviews/6Z00H20BQS.html.

[62] "Win32.agobot," http:/www3.ca.com/securityadvisor/virusinfo/virus.aspx?id=37776.

[63] VMware, "VMware Virtualization for Desktop & Server, Application, Public & Hybrid Clouds," http://www.vmware.com/.

[64] Microsoft, "Download Windows Virtual PC from Official Microsoft Download Center," http://www.microsoft.com/en-us/download/details.aspx?id=3702.

[65] P. Wang, L. Wu, R. Cunningham , C. C. Zou, “Honeypot Detection in Advanced Botnet Attacks,” *International Journal of Information and Computer Security 2010*, Vol. 4, No. 1, pp. 30-51, February 2010.

[66] V. Kamluk, "A black hat loses control," http://www.securelist.com/en/weblog?weblogid=208187881.

[67] "HOW FAST-FLUX SERVICE NETWORKS WORK," https://www.honeynet.org/node/132.

[68] T. Matsuki, Y. Arai, M. Terada , N. Doi, “Proposal of Anti-Malware Technique Turning Security Disabling Attack to Advantage,” *IPSJ Journal*, Vol. 50, No. 9, pp. 2127-2136, (in Japanese), September 2009.

[69] J. Caballero, C. Grier, C. Kreibich , V. Paxson, “Measuring Pay-per-Install: The Commoditization of Malware Distribution,” *In Proceedings of the 20th USENIX Security Symposium*, August 2011.

[70] J. Caballero, P. Poosankam, C. Kreibich , D. Song, “Dispatcher: Enabling active botnet infiltration using automatic protocol reverse-engineering,” *In Proceedings of the 16th ACM conference on Computer and communications security* (*CCS 2009*), pp. 621-634, November 2009.

[71] C. Kolbitsch, T. Holz, C. Kruegel , E. Kirda, “Inspector GAdget: Automated Extraction of

Proprietary Gadgets from Malware Binaries," *In Proceedings of the 2010 IEEE Symposium on Security and Privacy*, pp. 29-44, May 2010.

[72] C. Y. Cho, D. Babic, E. C. R. Shin , D. Song, "Inference and Analysis of Formal Models of Botnet Command and Control Protocols," *In Proceedings of the 17th ACM Conference on Computer and Communication Security* (*CCS 2010*), pp. 426-439, October 2010.

[73] J. Caballero, N. M. Johnson, S. McCamant , D. Song, "Binary code extraction and interface identification for security applications," *In Proceedings of the 17th Annual Network and Distributed System Security Symposium* (*NDSS 2010*), February 2010.

[74] "Windows Sysinternals: Documentation, downloads and additional resources," http://technet.microsoft.com/en-us/sysinternals/.

[75] "InCTRL Reporting & Analysis | Measuretronix Ltd.," http://www.measuretronix.com/en/products/inctrl-reporting-analysis.

[76] H. Father, "Hooking Windows API – Technics of Hooking API Functions on Windows," *CodeBreakers Journal*, Vol. 1, No. 2, 2004.

[77] "dionaea - chatches bugs," http://dionaea.carnivore.it/.

[78] "netfilter/iptables project homepage," http://www.netfilter.org/projects/iptables/.

[79] R. Dingledine, N. Mathewson , P. Syverson, "Tor: The Second-Generation Onion Router," *In Proccedings of the 13th USENIX Security Symposium*, pp. 303-320, August 2004.

[80] S. Chakravarty, A. Stavrou , A. D. Keromytis, "Identifying Proxy Nodes in a Tor Anonymization Circuit," *In Proceedings of the 4th IEEE International Conference on Signal Image Technology and Internet Based Systems* (*SITIS 2008*), pp. 633-639, November 2008.

[81] "Tor or not Tor: How to tell if someone is coming from a Tor exit node, in PHP," http://www.irongeek.com/i.php?page=security/detect-tor-exit-node-in-php.

[82] J. Brozycki, "Detecting Anonymous Proxy Usage (2008)," http://www.sans.edu/student-files/presentations/detecting_anonymous_proxies_handouts.pdf.

[83] "Dr.Web Online check," http://online.us.drweb.com/?url=1.

[84] "Online Virus Scanner - Scan Links for Malware, Trojans and Viruses," http://onlinelinkscan.com.

[85] "Website Safety Ratings and Reputation - AVG Threat Labs," http://www.avgthreatlabs.com/website-safety-reports/.

[86] "Webutation - Website Reputation Community against fraud and badware," http://www.webutation.net/.

[87] "Website Security Check - Unmask Parasites," http://www.unmaskparasites.com/.

[88] "Wepawet," http://wepawet.iseclab.org/.

[89] "gred," http://check.gred.jp/.

[90] "jsunpack - a generic JavaScript unpacker," http://jsunpack.jeek.org/dec/go.

[91] "vURL Online - Quickly and safely dissect malicious or suspect websites," http://vurldissect.co.uk/.

[92] P. Eckersley, "How Unique is Your Web Browser?," https://panopticlick.eff.org/browser-uniqueness.pdf.

[93] "Proxy.org - Tor servers - Tor IP List," http://proxy.org/tor.shtml.

[94] P. Porras, H. Saidi , V. Yegneswaran, “A Foray into Conficker's Logic and Rendezvous Points,” *In Proceedings of the USENIX Workshop on Large-Scale Exploits and Emergent Threats* (*LEET 2009*), pp. 7-7, April 2009.

[95] C. Kolbitsch, P. M. Comparetti, C. Kruegel, E. Kirda, X. Zhou , X. Wang, “Effective and Efficient Malware Detection at the End Host,” *In Proceedings of the 18th USENIX Security Symposium*, pp. 351-366, August 2009.

[96] K. Yoshioka, K. Murakami , T. Matsumoto, “Network Scan Method and Its Automated Signature Generation for Detecting Malware Infected Hosts,” *IPSJ Journal*, Vol. 51, No. 9, pp. 1633-1644, (in Japanese), September 2010.

[97] T. Sakai, K. Takemori, R. Ando , M. Nishigaki, “A Bot Detection Based on the Repetitiveness of Intrusion,” *IPSJ Journal*, Vol. 51, No. 9, pp. 1591-1599, (in Japanese), October 2010.

[98] J. Oberheide, E. Cooke , F. Jahanian, “CloudAV: N-Version Antivirus in the Network Cloud,” *In Proceedings of the 17th USENIX Security Symposium*, pp. 91-106, July 2008.

[99] "WINDOWS FOREST," http://www.forest.impress.co.jp/.

[100] T. Kasama, K. Yoshioka, D. Inoue, M. Eto, K. Nakao , T. Matsumoto, “Consideration on Malware Execution Time for Malware Sandbox Analysis,” *In Proceedings of the 2009 Symposium on Cryptography and Information Security* (*SCIS2009*), January 2009.

List of Papers

Reviewed Papers in Journals

(1) 笠間 貴弘, 織井 達憲, 吉岡 克成, 松本 勉, “公開型マルウェア動的解析システムに対するデコイ挿入攻撃の脅威,” 情報処理学会論文誌, Vol. 52, No. 9, pp. 2761-2774, 2011 年 9 月.

(2) T. Kasama, K. Yoshioka, T. Matsumoto, M. Yamagata, M. Eto, D. Inoue, K. Nakao, "Malware Sandbox Analysis with Efficient Observation of Herder's Behavior," IPSJ Journal of Information Processing, Vol. 20, No. 4, pp. 835-845, October 2012.

(3) T. Kasama, K. Yoshioka, D. Inoue, T. Matsumoto, "Catching the Behavioral Differences between Multiple Executions for Malware Detection," IEICE Transactions, Vol. E96-A, No. 1, pp. 225-232, January 2013.

Reviewed Papers in International Conference Proceedings

(4) T. Kasama, K. Yoshioka, T. Matsumoto, M. Yamagata, M. Eto, D. Inoue, and K. Nakao, "Malware Sandbox Analysis with Automatic Collection of Server Responses using Dummy Client," In Proceedings of the 5th Joint Workshop on Information Security (JWIS2010), August 2010.

(5) T. Kasama, K. Yoshioka, D. Inoue, and T. Matsumoto, "Malware Detection Method by Catching Their Random Behavior in Multiple Executions," In Proceedings of the 3rd Workshop on Network Technologies for Security, Administration and Protection (NETSAP 2012), July 2012.

(6) K. Yoshioka, T. Kasama, and T. Matsumoto, "Sandbox Analysis with Controlled Internet Connection for Observing Temporal Changes of Malware Behavior," In Proceedings of the 4th Joint Workshop on Information Security (JWIS2009), August 2009.

Technical Reports

(7) 笠間 貴弘, 吉岡 克成, 井上 大介, 衛藤 将史, 中尾 康二, 松本 勉, "マルウェア動的解析におけるマルウェア実行時間に関する検討," 電子情報通信学会 暗号と情報セキュリティシンポジウム 2009 (SCIS2009), 2009 年 1 月.

(8) 笠間 貴弘, 吉岡 克成, 松本 勉, 山形 昌也, 衛藤 将史, 中尾 康二, "疑似クライアントを用いたサーバ応答蓄積型マルウェア動的解析システム," 情報処理学会 マルウェア対策研究人材育成ワークショップ 2009 (MWS2009), 2009 年 10 月.

(9) 細渕 嘉彦, 笠間 貴弘, 吉岡 克成, 松本 勉, "インターネット接続型動的解析における IP アドレス使用法に関する考察," 情報処理学会 第 48 回コンピュータセキュリティ合同研究発表会 (CSEC48), 2010 年 3 月.

(10) 笠間 貴弘, 織井 達憲, 吉岡 克成, 松本 勉, "マルウェア動的解析オンラインサービスの脆弱性（その２）," 情報処理学会 コンピュータセキュリティシンポジウム 2010 (CSS2010), 2010 年 10 月.

(11) 村上 洸介, 織井 達憲, 笠間 貴弘, 吉岡 克成, 松本 勉, "マルウェア動的解析オンラインサービスの脆弱性解消方法の検討," 情報処理学会 第 52 回コンピュータセキュリティ合同研究発表会 (CSEC52), 2010 年 3 月.

(12) 笠間 貴弘, 吉岡 克成, 井上 大介, 松本 勉, "実行毎の挙動の差異に基づくマルウェア検知手法の提案," 情報処理学会 コンピュータセキュリティシンポジウム 2011 (CSS2011), 2011 年 10 月.

(13) 笠間 貴弘, 井上 大介, 衛藤 将史, 中里 純二, 中尾 康二, "ドライブ・バイ・ダウンロード攻撃対策フレームワークの提案," 情報処理学会コンピュータセキュリティシンポジウム 2011 (CSS2011), 2011 年 10 月.

(14) 笠間 貴弘, 中里 純二, 鈴木 未央, 衛藤 将史, 井上 大介, 中尾 康二, 秋山 満昭, 青木 一史, 岩村 誠, 八木 毅, 斉藤 典明, 針生 剛男, "多様なセンサの観測情報を用いたマルチモーダル分析," 電子情報通信学会 暗号と情報セキュリティシンポジウム 2012 (SCIS2012), 2012 年 1 月.

(15) 笠間 貴弘, 衛藤 将史, 井上 大介, "マルチモーダル分析による不正通信の検出," 電子情報通信学会 信学技報, Vol. 112, No. 315, ICSS2012-49, pp. 25-30, 2012 年 11 月.

(16) 笠間 貴弘, 嶌田 一郎, 衛藤 将史, 井上 大介, "ブラウザ組込型センサとゲートウェイセンサの観測情報を用いたマルウェア感染ホストの検出," 電子情報通信学会 暗号と情報セキュリティシンポジウム 2013 (SCIS2013), 2013 年 1 月.

(17) 笠間 貴弘, 中里 純二, 衛藤 将史, 井上 大介, 中尾 康二, 秋山 満昭, 岩村 誠, 八木 毅, 斉藤 典昭, 針生 剛男, "マルウェアの攻撃プロセスに着目したマルチモーダル分析," 電子情報通信学会 暗号と情報セキュリティシンポジウム 2013 (SCIS2013), 2013 年 1 月.

(18) 竹久 達也, 井上 大介, 衛藤 将史, 吉岡 克成, 笠間 貴弘, 中里 純二, 中尾 康二, "サイバーセキュリティ情報遠隔分析基盤 NONSTOP," 電子情報通信学会 信学技報, Vol. 113, No. 95, ICSS2013-15, pp. 85-90, 2013 年 6 月.

(19) 神薗 雅紀, 畑田 充弘, 寺田 真敏, 秋山 満昭, 笠間 貴弘, 村上 純一, "マルウェア対策のための研究用データセット ～ MWS 2013 Datasets ～," 情報処理学会 マルウェア対策研究人材育成ワークショップ 2013 (MWS2013), 2013 年 10 月.

(20) 笠間 貴弘, 神薗 雅紀, 井上 大介, "Exploit Kit の特徴を用いた悪性 Web サイト検知手法の提案," 情報処理学会 マルウェア対策研究人材育成ワークショップ 2013 (MWS2013), 2013 年 10 月.