Malware Sandbox Analysis for Secure Observation of Vulnerability Exploitation*

Katsunari YOSHIOKA†††, Daisuke INOUE††, Masashi ETO†††, Members, Yuji HOSHIZAWA†††, Nonmember, Hiroki NOGAWA†††, and Koji NAKAO††, Members

SUMMARY Exploiting vulnerabilities of remote systems is one of the fundamental behaviors of malware that determines their potential hazards. Understanding what kind of propagation tactics each malware uses is essential in incident response because such information directly links with countermeasures such as writing a signature for IDS. Although recently malware sandbox analysis has been studied intensively, little work is done on securely observing the vulnerability exploitation by malware. In this paper, we propose a novel sandbox analysis method for securely observing malware’s vulnerability exploitation in a totally isolated environment. In our sandbox, we prepare two victim hosts. We first execute the sample malware on one of these hosts and then let it attack the other host which is running multiple vulnerable services. As a simple realization of the proposed method, we have implemented a sandbox using Nepenthes, a low-interaction honeypot, as the second victim. Because Nepenthes can emulate a variety of vulnerable services, we can efficiently observe the propagation of sample malware. In the experiments, among 382 samples whose scan capabilities are confirmed, 381 samples successfully started exploiting vulnerabilities of the second victim. This indicates the certain level of feasibility of the proposed method.

key words: malware sandbox analysis, exploit code detection, low-interaction honeypot

1. Introduction

As more IT systems become connected to the Internet, concern over ensuring their security is growing. Continuous observations and studies suggest that Internet threats that take advantage of highly functional malicious programs called malware are becoming commonplace [1]–[3], [5], [7]–[16], [22]. A typical and well-known example is a botnet [22], which is a network of computers infected by malware and controlled by certain commanders for attacking and infecting other computers, sending spam/phishing emails, stealing key strokes, disclosing personal information, etc.

In order to protect the IT systems from their attacks or recover the systems from infections, it is necessary to understand the behaviour of malware in detail by in-depth analysis. However, with the underground distribution of development toolkits and utilization of polymorphism and metamorphism, the diversity of malware is growing, so that the adoption of automated analysis needs to be discussed.

One promising approach toward automated analysis is sandbox analysis. In this approach, a malware sample is actually executed in a testing environment, called a sandbox, which is specially prepared to closely observe the detailed behaviour of malware. Compared to the conventional human-operated manual analysis that involves highly skilled reverse engineering techniques [23], the malware sandbox analysis can be implemented in highly automated fashion.

The sandbox analysis can be categorized into two ways. In the first way, the infected host in the sandbox, called the victim host, is connected to the real Internet and in the second way the victim host is completely isolated from the Internet. The former is more desirable in terms of observing diverse activities of malware because many recent malware indeed communicate with other hosts in the Internet. The obvious drawback of this approach is that attacks from the infected host may exit the sandbox, causing unwanted second infections in the real Internet. Moreover, as the number of examined malware increases the risk of such unwanted infection also increases. Previous studies utilize multiple filtering to minimize such risks [3], [5] although it is still difficult to ensure a totally secure operation. The latter approach, using an isolated sandbox, does not suffer from the risk of the unwanted second infection unless the sandbox itself has vulnerability. A drawback of using an isolated sandbox is that it is simply difficult to observe all the malicious activities within an isolated system. For example, if a malware uses a DNS service to find the IP address of the control server, it would not act as it was meant to unless the sandbox provided a DNS service. Moreover, many malware are capable of detecting sandbox environments, for example, by testing connections with real servers in the Internet. Therefore, it is necessary that the isolated sandbox provides various network services to deceive malware. In [7], we proposed a malware analysis system using a completely isolated sandbox. The proposed method utilizes a component called an Internet emulator. This component can provide various network services such as DNS, FTP, TFTP, SMTP, HTTP, HTTPS, and IRC. This method is useful for detecting some of the important behaviours of malware, such as alteration of registry keys and files, creation of mutex, etc. However, it cannot be used to observe
the infection towards other hosts by vulnerability exploitation, simply because there is no other vulnerable host in the sandbox besides the victim host itself. However, the observation of vulnerability exploitation is very important as it characterizes the fundamental hazards of an examined malware. For example we can write a signature for IDS by observing exploit codes utilized by sample malware.

In this paper, we propose a novel sandbox analysis method for observing malware’s exploitation of vulnerabilities in a completely isolated sandbox. In our method, we prepare a sandbox with two victim hosts. We first execute the malware sample on one of these hosts (called the first victim). When the first victim starts propagating, we redirect the attacks on the other victim host (called the second victim) which is running multiple vulnerable services. Then, we closely monitor both victims in order to understand the mechanism of infection. As a simple realization of the proposed method, we have implemented a sandbox using Nepenthes [1], a low-interaction honeypot, as the second victim. Because Nepenthes can emulate a variety of vulnerable services, we expected that the malware activated in the first victim would try to exploit the emulated vulnerabilities of the second victim. By using Ashula [11], [17], an algorithm for detecting exploit codes, we examined all packets exchanged between the first and second victims to see if any exploit codes were being used. Also, we used a simple Portable Executable (PE) format detector to see if the payloads of these packets contain any executable codes. In addition, we parsed the logs from Nepenthes to see if there is any attempted exploitation of vulnerabilities. In the experiment, among 382 samples whose scan capabilities are confirmed, 381 samples successfully started exploiting vulnerabilities of the second victim. This indicates a certain level of feasibility of the proposed method.

The organization of this paper is as follows. In Sect. 2, we briefly summarize the criteria of the malware sandbox analysis. Section 3 explains our proposed method for sandbox analysis and its implementation, and Sect. 4 describes experiments for confirming the feasibility of the proposed method. Section 5 introduces some related works and explains the differences between them and the proposed method. Section 6 provides conclusions and directions for future work.

2. Criteria of Malware Sandbox Analysis

In this section, we show three criteria which we consider essential in designing malware sandbox analysis.

Observability. In general, there are many different reasons why we want to analyze malware executables. For example, it is important to observe the internal behaviour of malware such as file creation, registry alteration, API calls, etc. because such information is necessary to write a signature for anti-virus software as well as removal tools. It is also important to observe the network behaviour of malware as they can be used, for example as IDS signature, to detect and block malicious traffic to mitigate the propagation. In research area, the reason can be more general such as understanding the global trend of malware attacks in the wild [10] or their classification [3]. Either way, the malware sandbox analysis should be able to observe the behaviour of interest for providing sufficient information to achieve its goal.

Security. As the basic concept of the malware sandbox analysis is to activate malware to observe their behaviour, it naturally involves a potential risk of unwanted second infection. Especially, when analysis is performed in an automated fashion and a great number of malware executables are examined, such a risk increases. Therefore, from security point of view, it is safe to totally isolate the sandbox from outside world. However, when observability is concerned, it is not always the best way to isolate the sandbox because many malware would not behave quite the same if they were in an isolated environment. They may require services like DNS or NTP. Bots may try to connect to control servers or file download servers for upgrade. Others may try to connect to arbitrary servers to check Internet reachability. Thus, there is a trade-off between observability and security.

Efficiency. One of the major advantages of the malware sandbox analysis is its efficiency from its high-level automation. Therefore, it is important that we can observe the behaviour of our interests in limited examination time.

3. Proposal: Totally Isolated Malware Sandbox with Multiple Victim Hosts

3.1 Design Principle

The purpose of this study is to observe the vulnerability exploitation by malware because it is one of the fundamental behaviour of malware that decides their potential hazards and is essential information when responding them. As stated in Sect. 2, there is a trade-off among the criteria of the malware sandbox analysis. Therefore, it is important to set a proper goal for each criterion. In our study, we consider the security is the most important criterion of all because the very starting point of this study is to stop the malware infection, not to spread them even though the risk is not high. Therefore, we take the totally isolated sandbox approach for the maximum security level. In terms of observability, we utilize a component called Internet emulator, which is a set of dummy servers to give various responses to the malware for leading them misjudge that they are connected to the real Internet. Finally, for efficiently withdrawing the propagation, we prepared the second victim host with multiple vulnerabilities, such as a honeypot. We first execute the malware sample in the first victim. When the first victim starts propagation, we redirect the attacks on the second victim. Since the second victim runs multiple vulnerable services, we can efficiently observe the propagation by the malware samples.

3.2 Architecture

We show the architecture of the proposed sandbox in Fig. 1.
The Internet emulator is a set of Internet Emulator configuration of the sandbox. It is important to note that the security level of the first victim must be correctly configured so that the executed malware performs further action that can be observed. For the experiment explained in Sect. 4, we used Windows XP SP1 as the operating system of the first victim. In addition, we utilized various technologies such as API hooking [6] to monitor the internal activities of the executed malware. More detailed explanation of these techniques can be found in Sect. 3.3.

First Victim. The first victim is a host in which the examined malware sample is first executed. It is important that the security level of the first victim be correctly configured so that the executed malware performs further action that we can observe. For the experiment explained in Sect. 4, we used Windows XP SP1 as the operating system of the first victim. In addition, we utilized various technologies such as API hooking [6] to monitor the internal activities of the executed malware. More detailed explanation of these techniques can be found in Sect. 3.3.

Second Victim. The second victim is a host to be infected by the malware executed in the first victim. It should run vulnerable services to be exploited; otherwise, no further infection will occur. Normally, we do not know what kind of vulnerability the examined malware will exploit; therefore, the second victim should run or emulate as many active vulnerable services as possible. In the proposed sandbox, we utilized a honeypot system as the second victim since it was originally developed to observe network attacks by intentionally providing vulnerable services. We deployed Nepenthes, a low-interaction honeypot, in the proposed sandbox for improving efficiency, since it can emulate a variety of vulnerable services. We were also able to utilize more than two victim hosts, each of which emulated different vulnerable services; however, this complicated the configuration of the sandbox.

Internet Emulator. The Internet emulator is a set of server programs that talk to the first victim to provide services it would receive in the real Internet, such as DNS, FTP, HTTP, SMTP, NTP, and IRC. Because the sandbox is completely isolated from the real Internet, this functionality is a key to deceive the malware into thinking that they are not in our testing environment.

Distinguisher. The distinguisher is a functionality to determine whether traffic from the first victim should be sent to the second victim or the Internet emulator. It keeps listening to all incoming packets from the first victim and determining where to pass them. The reason why such a component is needed is that malware carry out various communications besides those for propagation. For example, some malware check Internet reachability by accessing certain servers. Others access FTP servers to obtain files they use for their updates. Bots commonly access IRC and HTTP servers to receive command and control (C&C) messages. All these communications are not forms of propagation and therefore should be directed to the Internet emulator, and not the second victim.

Traffic Analyzer. The traffic analyzer captures packets transmitted between the first victim and the other three components. The main role of the traffic analyzer is detecting exploit codes and executable codes. By detecting exploit codes, we know which malware exploits what kind of vulnerable services, and this is the main purpose of this study. Detecting executables also helps understand how malware propagate over network. We use Ashula [11], [17] for detecting exploit codes. It reconstructs TCP streams from the packet logs and then checks each reconstructed stream for certain structures of machine code instructions that are typically observed in buffer overflow exploits. Moreover, from each stream, Ashula extracts a binary string called an exploit code signature. This binary string contains a fundamental sequence of instructions for vulnerability exploitation. The signature can also be used as a signature for IDS. For detecting executables, we simply search the payload of packets for several particular strings† common in Portable Executable (PE) format [21] such as “0x4d5a”, “0x50450000”, etc.

Log Analyzer. The log analyzer analyzes logs from API hooking at the first victim, logs from the second victim, and the Internet emulator. These raw data are interpreted into high-level description of behaviour by pattern matching with pre-defined behaviour database. Meantime, the log from the second victim, namely Nepenthes, is analyzed to see if there is any attempt of exploiting the emulated vulnerabilities of the second victim.

3.3 Implementation of Sandbox

We show an implementation of the proposed sandbox in Fig. 2. We use two real machines: real machine 1 and real machine 2. The real machine 1 is used as the first victim and the real machine 2 is used as the distinguisher, Internet emulator, and second victim. Although our proposed sandbox can be implemented by many approaches, we deployed a simple approach for its evaluation. For the first victim, it is also possible to use a virtual machine for easier recovery although we decided to use a real machine to avoid virtual machine detection by malware.

Real Machine 1 (First Victim). The first victim is implemented as a Linux/Windows dual boot real machine. It is configured to boot with Linux OS and Windows OS alternately by its boot loader. The target malware sample is executed when Windows OS is running. After malware execution of a predefined period of time (e.g., 30 s), the Windows OS shuts down automatically and the Linux OS is

† The detection of executables by these strings is considered as a trigger for further detailed analysis. In experiments, we also checked the detected executables manually for confirmation.
booted. On the Linux OS, the pre-stored image of the original Windows OS (before infection) is written over the infected Windows OS image to refresh it. The first victim can hook and monitor Windows API calls used by the malware by an API hook technique similar to that used conventionally [6]; this enables the extraction of internal activity. The first victim executes the malware in the suspended mode. Then a remote thread is created in the malware process. The remote thread loads a DLL named API_Hook_Lib.dll. Meanwhile, the API addresses in the import address table (IAT) are overwritten so as to redirect the API calls to API_Hook_Lib.dll. In the DLL, the API calls are logged and redirected again to the corresponding original APIs. Even APIs in a DLL linked during run-time can be monitored because the LoadLibrary function is also hooked. After a given period of time, the victim host outputs the Windows API logs to the log analyzer. If the malware adds some run keys in the Windows registry in order for auto run to initiate, the victim host restarts the OS and conducts the analysis again to extract the malware behaviour after the restart. This mechanism is realized as follows: When the API hooking function of the first victim detects the changes of run key in the registry, it generates a file in a predefined location to indicate the change. Then, every time Linux boots up, it mounts the Windows File System and checks if the file exists or not. If it does, Linux shuts down without cleaning up the Windows OS image. Finally, the Windows OS boots up again to start analyzing. Linux counts the number of this loop so that it will not get into an infinite loop even though malware changes run key every time. Such re-execution results in the wide range of the analysis time (i.e., five to ten minutes). Also, if malware uses wait function to pause the activities to avoid being analyzed, the first victim will cancel the wait function to proceed the analysis.

Real Machine 2 (Distinguisher, Internet Emulator, and Second Victim). The distinguisher, Internet emulator, and second victim are all implemented in a single real machine with Linux OS. The Internet emulator is realized as a set of dummy server programs, namely DNS, FTP, TFTP, HTTP, HTTPS, SMTP, NTP, and IRC servers, which are configured to reply back a dummy response for a query from malware sample. They are written in Python. For example, the dummy DNS server generates a random response to each DNS query. Note that the dummy servers record the dummy responses until the end of the analysis of each sample so that it can send the same response for the same request to avoid the detection by malware. The server programs of the Internet emulator keep listening on the default port number of each service.

The second victim is implemented as a low-interaction honeypot program, Nepenthes. See below for details of Nepenthes. Nepenthes listens on various vulnerable ports such as 445/TCP or 135/TCP and replies when these ports are accessed in order to emulate vulnerable services.

The distinguisher is realized by iptables, a packet filtering tool of Linux. All TCP/UDP packets sent from the real machine 1 to the real machine 2 are filtered based on the destination port number. When a TCP SYN packet on a default port number such as 20/TCP for FTP or 25/TCP for SMTP arrives at the real machine 2, it is redirected to the corresponding server program provided by the Internet emulator. Likewise, when a TCP SYN packet arrives on port numbers on which Nepenthes is listening, it is redirected to the second victim. There are some ports, such as 20, 21, 25, 80/TCP, on which both dummy servers and Nepenthes can listen. As the default setting, packets on these ports are sent to the dummy servers. However, we can also change the setting of the distinguisher to send these packets to Nepenthes and perform the analysis again for comparison. Please see Experiment 3 for an example of such a comparison. When a TCP SYN packet arrives on an unknown port, it is redirected to another server program called Dummyd. Dummyd first replies to all TCP SYN packets and then, if it receives the first payload, it parses the payload to find out what kind of protocol is used. The present implementation of Dummyd can only recognize HTTP, HTTPS, and IRC. When the protocol is determined, Dummyd passes the packets to a suitable server program.

Nepenthes. Nepenthes was developed by the University of Mannheim as a low-interaction honeypot that is capable of collecting malware samples automatically. Nepenthes utilizes vulnerability modules to emulate various vulnerable services. When the modules receive packets from malware, Nepenthes parses their payload, detects the exploit codes, and decodes the download attempts including the target URL. When the URL is successfully extracted from the exploit codes, Nepenthes automatically downloads the intended files to collect malware executables. Note that in our sandbox, this downloading capability is limited because the second victim is not connected to the Internet. Our intention for using Nepenthes is to observe exploitation of its emulated vulnerabilities and not to collect malware, although it would be interesting to recursively collect samples by downloading and re-executing them in our sandbox. Nepenthes

†It records the random responses within each operation so that it can reply the same response for the same query.
logs all exploitation attempts from attackers so we checked these logs to see if any exploitation attempt has occurred.

4. Experiments

In this section, we explain the experiments conducted to evaluate the feasibility of our proposed method.

**Malware samples.** In the experiments we used 3,123 samples from two different sources. The first set is 2,815 samples collected by our honeypot (Nepenthes version 0.2.0) from Aug. 2007 to Dec. 2008. The second set is 308 samples obtained from Offensive Computing [24], a website to share samples and information of malware.

**Sandbox.** In the experiments, we used two kinds of sandbox for comparison: primary sandbox and proposed sandbox. The primary sandbox contains only a single victim and the Internet emulator that provides the dummy services, which is the configuration we used in [7]. We used Fedora Core 6 (2.6.22.5-49.fc6) and Windows XP SP1 as the dual-boot operating systems of real machine 1 and CentOS4 (2.6.9-55) for real machine 2. The second victim was implemented as Nepenthes version 0.2.2. The ports listened by the second victim was 42, 110, 135, 139, 143, 220, 443, 445, 465, 993, 995, 1023, 1025, 2105, 3372, 5000, 10000, 17300/TCP.

**Experiment 1.** Experiment 1 is the main experiment we conducted. In this experiment, we compare proposed sandbox and primary sandbox using all 3,123 samples. Each sample is independently analyzed by both of the sandboxes with the execution time of 30 seconds.

**Experiment 2.** Experiment 2 investigates the impact of the execution time. We compare the analysis results of the proposed sandbox by varying the execution time from 10 to 60 seconds. For this experiment, 382 samples whose scanning capabilities are confirmed in Experiment 1 are used.

**Experiment 3.** Experiment 3 investigates the difference caused by the setting of the distinguisher. In the proposed sandbox, some ports, such as, 20, 21, 25, 80/TCP, can be listened by both the Internet emulator and the second victim. As the default setting, these ports are listened by the Internet emulator. In this experiment, we change the setting and let the second victim listen on these ports to compare and validate the adequacy of the setting. For this experiment, 382 samples whose scanning capabilities are confirmed in Experiment 1 are used.

**Experiment 4.** Experiment 4 investigates the effect of randomised behaviours of malware. As some malware may change their behaviours every time they are executed, we performed the analyses by proposed sandbox three times and investigated on the difference in the observed behaviours. For this experiment, 382 samples whose scanning capabilities are confirmed in Experiment 1 are used.

4.1 Experiment 1: The Main Experiment

4.1.1 Statistics

We first show the statistics of results from Experiment 1. Table 1 summarizes the analysis results of samples obtained by Nepenthes. Among the 2,815 samples, 2,026 samples were valid Windows PE [21]. Among the 2,026 samples, 358 samples started scanning in both proposed and primary sandboxes. We considered that a malware was performing port scan if it accessed many different IP addresses without name resolution by DNS. In other words, if the first victim used the port to access more distinct IP addresses than a threshold value, we considered that the port was scanned. In the experiment, the threshold for the number of distinct IP addresses was fixed at 10. According to the exploit code detector, Ashula, among the 358 samples, 224 samples started sending exploit codes in the proposed sandbox while only 1 sample did in the primary sandbox. Also, according to the PE detector, 353 samples sent an executable to the second victim in the proposed sandbox while 134 samples did in the primary sandbox. The reason why these 134 samples sent an executable without the second victim is that these malware exploits backdoors of other malware. Typically they exploited backdoor of MyDoom [18] on 3127/TCP. In such a case, malware sent their copy right after the establishment of TCP connection and therefore interaction with the second victim was not necessary. However, other 219 samples needed to interact with the second victim before sending their copies. The detection rate of exploitation, which is the rate of samples that sent exploit codes or executables to those that started port scan, is 100.0% in the proposed sandbox and 37.7% in the primary sandbox.

Next, we explain the analysis results for samples from Offensivecomputing.net. The summary of the result is in Table 2. Among the 308 samples, 303 samples were valid Windows PE. Among the 303 samples, 24 samples started

| Table 1 | Comparison of analysis results of primary sandbox and proposed sandbox (samples by Nepenthes). |
|---------|-----------------------------------------------------------------------------------------------|
|          | Proposed Sandbox | Primary Sandbox |          |         |
| a. Valid Samples | 2026 | 2026 |          |         |
| b. Samples that Port Scanned | 358 | 358 |          |         |
| c. Samples that Sent Exploit Codes | 224 | 1 |          |         |
| d. Samples that Sent Executables | 353 | 134 |          |         |
| e. Samples that Sent Exploit Codes or Executables | 358 | 135 |          |         |
| f. Detection Rate of Exploitation (100*e/b) [%] | 100.0 | 37.7 |          |         |

| Table 2 | Comparison of analysis results of primary sandbox and proposed sandbox (samples from offensivecomputing.net). |
|---------|-------------------------------------------------------------------------------------------------------------|
|          | Proposed Sandbox | Primary Sandbox |          |         |
| a. Valid Samples | 303 | 303 |          |         |
| b. Samples that Port Scanned | 24 | 24 |          |         |
| c. Samples that Sent Exploit Codes | 23 | 0 |          |         |
| d. Samples that Sent Executables | 23 | 0 |          |         |
| e. Samples that Sent Exploit Codes or Executables | 23 | 0 |          |         |
| f. Detection Rate of Exploitation (100*e/b) [%] | 95.8 | 0.0 |          |         |
port scan in both proposed and primary sandboxes. The proposed sandbox was able to observe both exploit code and executables from 23 samples while the primary sandbox could not observe them at all. In Sect. 4.3, we discuss the one sample which did not show any exploitation behaviours in both of the sandboxes.

Table 3 summarizes the port numbers that are scanned by malware in the proposed sandbox. From Table 3, the most commonly scanned ports are 445/TCP, 3127/TCP, and 139/TCP. Table 4 summarized the port numbers that received exploit codes or executables. As shown in Table 4, the most frequently exploited port among all the samples is 445/TCP by 246 samples. The second is 3127/TCP by 134 samples. As stated earlier, 3127/TCP is the backdoor of MyDoom and executable codes are directly sent to the port without any buffer overflow attack. More detailed considerations on the exploitations of particular samples can be found in Sect. 4.1.2.

### 4.1.2 Concrete Cases

Next, we discuss several concrete analysis cases to explain in details how the proposed sandbox works. We have summarized the analysis results of 35 samples in Table 5. The names of the samples in the table were obtained from Symantec Norton Anti Virus 2007 (2007.07.26.017) and McAfee VirusScan for Client (v5084). The column ‘Port Scan’ indicates the destination port numbers scanned by each malware. The number on the left in parenthesis indicates the number of targeted distinct IP addresses and the number on the right in parenthesis indicates the number of packets used for scanning. For example, the first row in Table 5 indicates that malware sample 1 in the primary sandbox accessed 539 different IP addresses on port 3127/TCP by using 27,442 packets.

The column ‘Exploited Code’ indicates the destination port numbers on which the second victim received an exploit code from the first victim. If the result is with the mark “*”, it means that there was a transmission of executable codes detected between the first and the second victim. These results were obtained by the traffic analyzer. Likewise, the numbers in parenthesis indicate the number of targeted IP addresses and the number of packets. The column ‘Exploitation Detected by Nepenthes’ indicates the temporary name of the exploit code categorized by the traffic analyzer. The number in parenthesis indicates the number of detected exploit codes. For example, according to Table 5, malware sample 3 in the proposed sandbox sent exploit codes with three different signatures to 196 different IP addresses on port 135/UDP by using 600 packets. In addition, the exploit codes of three types are observed exactly 200 times each. Finally, the column ‘Exploitation Detected by Nepenthes’ indicates the type of exploitation detected by Nepenthes. Each symbol represents as follows. S: SMB Shellcode, SM: SMB Name exploit, N: NETDDE exploit, L: LSASS Shellcode, D: DCOM Shellcode, P: PNP Shellcode.

### Port Scans.

From Table 5, we can confirm that all 35 samples started scanning in both sandboxes. The most commonly scanned ports are 445/TCP, 3127/TCP, and 139/TCP, which were scanned by 28, 21, and 20 samples, respectively. 445/TCP and 139/TCP are well-known vulnerable ports. 3127/TCP is the backdoor of MyDoom. These three ports are scanned together as 20 samples scanned all these three ports. Besides these ports, 135/TCP is scanned by 5 different samples, 4 of which are named as Blaster variants [20] by both Norton AntiVirus and McAfee VirusScan.

### Exploitation.

As shown in Table 5, the most often observed exploitation among the 35 samples is on 3127/TCP, attacked by 21 samples. This exploitation is relatively easy to observe. After the execution, the sample in the first victim starts TCP SYN scan on 3127/TCP to randomly chosen global IP addresses. As stated earlier, this port scan is to find a host that is infected by MyDoom variants. When it receives a SYN-ACK from any remote host, it establishes a TCP session and immediately starts transmitting its copy (the executable) to it. Meantime, it sends a “discovery report” to a predefined IRC channel. The report contains the string such as “MyDoom.spreader.found.a.victim:x.x.x.x” where x.x.x.x is the IP address of the discovered host.

Attacked by 7 samples, 445/TCP was the second most popular port to be attacked. However, these 7 samples used only 3 types of exploit codes: type5, type6, type7. The type5 was detected in the attacks by ‘W32.Sasser.C.Worm’ or ‘W32.Virut.B’. The combination of type6 and type7 was often seen in the attacks by ‘W32.Iblo.A’. Finally, type7 was solely detected in the attacks by ‘Backdoor.Berbew.N’.

135/TCP was commonly attacked by Blaster variants. Even though there were many variants according to Norton Antivirus; W32.Blaster.Worm, W32.Blaster.C.Worm, W32.Blaster.E.Worm, and W32.Blaster.F.Worm, their propagation tactics are very similar. They also use the same exploit code type4.

In the following we explain analysis results of some concrete cases to demonstrate how the proposed sandbox
Table 5: Comparison of analysis results by primary sandbox and proposed sandbox. The column ‘Scanned Port’ indicates the destination port numbers that are scanned by each malware. The left number in the parenthesis indicates the number of targeted hosts and the right number in the parenthesis indicates the number of packets. The column ‘Exploited Port’ indicates the number of destination ports that have received an exploit code from the first victim. If a result is with the mark ‘*’, it indicates that an executable code is detected from the traffic. The column ‘MD5 Digest of Exploit Code Signature’ indicates the MD5 digest value of the exploit code signatures obtained from the traffic analyzer. The number in the parenthesis indicates the number of detected exploit codes. Finally, the column ‘Exploit Code Detected by Nepenthes’ indicates the type of exploit codes detected by Nepenthes. Each symbol represents the exploit codes as follows: S: SMB Shellcode, SM: SMB Name exploit, N: NETDDE exploit, L: LSASS Shellcode, D: DCOM Shellcode, P: PNP Shellcode.

| No | MD5 Digest of Sample (Norton/McAfee) | Primary | Exploit Code (Num. IP Addresses | Exploit Code Type | Exploitation Detected by Nepenthes |
|----|--------------------------------------|--------|---------------------------------|------------------|-----------------------------------|
|    | | sandbox type | Port Scan | (Num. Packets) | |                                      |
|    | | | | | |                                      |
|    | | | | | |                                      |
|    | | | | | |                                      |
|    | | | | | |                                      |

| 1 | 02d6d9f7921e6485834171e727017f75 | Primary | 931TCP|3505|2434/123TCP|3505|2043/102600 | SMB | D.L.S. |
|    | | | | | | |                                      |
| 2 | 05912189f59e65f67c7f0a403d8f7f8f | Proposed | 350TCP|3508/1500 | 123TCP|3505|2043/102600 | SMB | D.L.S. |
| 3 | 1e85785160363b706328e82b60726a03 | Primary | 130TCP|3001/1000 | 95TCP|3505|2043/102600 | D.L.P. |
|    | | | | | | |                                      |
| 4 | 234413060049b106b3062582a224a246 | Proposed | 130TCP|3001/1000 | D.L.P. |
| 5 | 28af48319169f331a34a60322b444 | Primary | 464TCP|3211/1000 | 464TCP|3211/1000 | D.L.P. |
| 6 | 5590a20e2b340c25c2b08798b0f88d60 | Proposed | 464TCP|3211/1000 | D.L.P. |
|    | | | | | | |                                      |
| 7 | 376d925db8d6ed0b5c03a2e57c30c91c | Primary | 312TCP|3503/2043/102600 | 123TCP|3505|2043/102600 | N.S.M.P.L.D. |
| 8 | 93bfc2e914567d8f675683bf8d8e467b | Proposed | 312TCP|3503/2043/102600 | 123TCP|3505|2043/102600 | N.S.M.P.L.D. |
| 9 | 586460e7103a9f206175c18c08f462c5 | Primary | 471TCP|3464/102600 | 471TCP|3464/102600 | D.P.S.I. |
| 10 | 80c3d18ab68c3e4b8a977b049369e6d4 | Proposed | 471TCP|3464/102600 | 471TCP|3464/102600 | D.P.S.I. |
| 11 | 5fd65725a57e29a125c024847037304 | Primary | 899TCP|3501/102600 | 899TCP|3501/102600 | N.S.M.D.P.L.P. |
|    | | | | | | |                                      |
| 12 | 02a506c628405043b0a8b447c70a0b47 | Proposed | 350TCP|3508/1500 | 350TCP|3508/1500 | SM.D.P.L.P. |
| 13 | 607b290b854954259a4e749d717981c5 | Proposed | 350TCP|3508/1500 | 350TCP|3508/1500 | SM.D.P.L.P. |
| 14 | 307b25c3a30f7f5129f999915e158f0a0 | Primary | 307TCP|3033/1000 | 307TCP|3033/1000 | D |
|    | | | | | | |                                      |
| 15 | 387254aa9b76f89076f697796f70925cf | Proposed | 387TCP|3871/1000 | 387TCP|3871/1000 | D |
| 16 | 783f32f099e787f395991e4620f0529 | Proposed | 783TCP|7831/1000 | 783TCP|7831/1000 | D |
| 17 | 75be2a5649b75e61f6f30f64e8c4e749 | Proposed | 75beTCP|75be1/1000 | 75beTCP|75be1/1000 | N.S.M.P.L.D.P. |
|    | | | | | | |                                      |
| 18 | 844d6b61ae4f05358006a08c885729c | Proposed | 844dTCP|844d1/1000 | 844dTCP|844d1/1000 | P.S.L.D. |
| 19 | 587b21b644594259a018b74793015203 | Proposed | 587bTCP|587b1/1000 | 587bTCP|587b1/1000 | P.S.L.D. |
| 20 | 9382b6e0fa5f8d7b5b8f8e4b86e488e | Proposed | 938TCP|9381/1000 | 938TCP|9381/1000 | D |

Note: The number in the parenthesis indicates the number of detected exploit codes.
Fig. 3 A TCP session established between the first victim (192.168.20.23) and the second victim (192.168.103.8). After a 3-way handshake in (1)–(3), the two victims exchange 14 packets in (4)–(17) under SMB protocol before the first victim sends an exploit code using three packets in (18), (19), and (21). The exploit code is to make the second victim to download its copy from 192.168.20.23:80/xxxxxxxx. The URL of intended download site can be confirmed from the payload of (18).

works compared with the primary sandbox. The name in the left of each parenthesis was obtained by Symantec Norton Anti Virus 2007 (2007.07.26.017) and the right one is by McAfee VirusScan for Client.

Sample 6 (W32.Ifbo.A|Unknown). After the execution, sample 6 performed a TCP SYN scan on 445/TCP to private IP addresses 192.168.x.x, where x.x was decided randomly to look for a vulnerable host. Immediately, the second victim responded to the scan because Nepenthes was listening on this port. We show in Fig. 3 the actual TCP session established between the first victim (192.168.20.23) and the second victim of the proposed

---

**Raw Text:**

962

IEICE TRANS. INF. & SYST., VOL.E92–D, NO.5 MAY 2009

Fig. 3 A TCP session established between the first victim (192.168.20.23) and the second victim (192.168.103.8). After a 3-way handshake in (1)–(3), the two victims exchange 14 packets in (4)–(17) under SMB protocol before the first victim sends an exploit code using three packets in (18), (19), and (21). The exploit code is to make the second victim to download its copy from 192.168.20.23:80/xxxxxxxx. The URL of intended download site can be confirmed from the payload of (18).

works compared with the primary sandbox. The name in the left of each parenthesis was obtained by Symantec Norton Anti Virus 2007 (2007.07.26.017) and the right one is by McAfee VirusScan for Client.

Sample 6 (W32.Ifbo.A|Unknown). After the execution, sample 6 performed a TCP SYN scan on 445/TCP to private IP addresses 192.168.x.x, where x.x was decided randomly to look for a vulnerable host. Immediately, the second victim responded to the scan because Nepenthes was listening on this port. We show in Fig. 3 the actual TCP session established between the first victim (192.168.20.23) and the second victim of the proposed
sandbox. The first three packets (1)–(3) represent a three-way handshake. After a session was established, the two victims exchanged 14 packets in (4)–(17) under SMB protocol. This interaction was handled by Nepenthes installed in the second victim. The service emulation worked correctly and the first victim finally sent its exploit code by three packets: (18), (19), and (20). This exploit code was intended to force the second victim to download the copy of the malware executable from the HTTP server http://192.168.20.23/xxxxxxx; the malware had already set up this server at the first victim upon execution. It is possible to confirm the intended download site in ASCII code in the packet payload (18); however, this kind of information is often obfuscated. Subsequently, the second victim accessed the HTTP server on the first victim and downloaded a copy of the malware sample. The malware attempted the above propagation to 56 different remote hosts within 30 seconds of the execution time. Contrastively, the primary sandbox was only able to respond to the first SYN packet and therefore it was not able to observe further propagation.

Sample 3 (W32.HLW.Doomjuice.B|W32/Sdbot.worm). After the execution, sample 3 started TCP SYN scan on 135/TCP to private IP addresses 192.168.x.x where x.x is decided randomly. The second victim responded to the SYN packet as it listens on this port. The first and the second victim then exchanged 17 packets transmitting payloads of several hundreds bytes. Then it finished the TCP session. After that, the sample sent three large UDP packets (3736 bytes) on 135/UDP to the same host. The traffic analyzer detected an exploit code from each of these packets. Their signature digests differ from each other so the malware may have sent three different exploit codes to its target. After the UDP packets, the sample sent several packets on 9191/TCP to the same host. We consider that the first TCP session on 135/TCP was to check if its target actually runs a vulnerable service on 135/UDP. As our second victim responded correctly it started sending exploit codes on 135/UDP. Also, the port 9191/TCP should be the backdoor that would have been opened by the exploit codes on 135/UDP although we have not done further detailed analysis for confirmation. The malware attempted the above propagation to 196 different remote hosts within 30 seconds of the execution time. Contrastively, the primary sandbox could only respond to the first SYN packet and therefore the further propagation was not observed in the primary sandbox.

Sample 28 (W32.Blaster.F.Worm | W32/Blaster.worm.f). After the execution, sample 28 started TCP SYN scan on 135/TCP to randomly decided global IP addresses. The second victim responded to the SYN packets as it listens on this port. They established a TCP session and the first victim transmitted a stream of 1777 bytes to the second victim. This stream contained an exploit code according to the traffic analyzer. After the TCP session is closed, the sample accessed on 4444/TCP of the same host. The port 4444/TCP should be the backdoor port. The malware attempted the above propagation to 3 different remote hosts within 30 seconds of the execution time. The primary sandbox could only handle the 3-way handshake and was not able to observe the propagation attempts.

4.2 Experiment 2: Investigation on Execution Time

In Experiment 2, we investigate how the execution time of malware can affect our observation results. Using the execution time of 10, 20, 30, 40, 50, and 60 seconds, we reanalyzed all 382 samples that started scanning in the proposed sandbox. Figure 4 summarizes the average number of scanned ports with respect to the execution time. It can be said that execution of 30 seconds is appropriate to observe the scan behaviours in the proposed sandbox. However, we note that the appropriate execution time may differ depending on the environment of the sandbox and the malware sample sets.

4.3 Experiment 3: Investigation on Distinguisher

In Experiment 3, we investigate the difference caused by the setting of the distinguisher. In the proposed sandbox, some ports, such as, 20, 21, 25, 80/TCP, can be listened by both the Internet emulator and the second victim. As the default setting, these ports are used by the Internet emulator. In this experiment, we change the setting and let the second victim listen on these ports to compare the results. The 382 samples whose scanning capabilities are confirmed in Experiment 1 are reanalyzed by using the above new setting. As a result however, there was no sample that started scanning on 20, 21, 25/TCP. There was one sample that started scanning on 80/TCP. With the new setting, we redirected the scan packets to the second victim but no further exploitation was observed this time. This sample was named “W32/Nimda.E@mm” by Norton Antivirus. The MD5 hash value is “0c58f8279c025a457346834df20db2e2.” It seems that the second victim (Nepenthes) was not able to handle the interaction with this particular sample. However, if the second victim could have handled the interaction, we might have been able to observe its exploitation, which indicates the importance of this multi-pass analysis with different settings.
the second victim so that vulnerability exploitation to other host can not be observed.

Instead of using a virtual or emulated machine, a real machine is used for malware execution in Capture [12] and Joebox [16]. These two tools mainly focus on extracting internal behaviour of malware and do not discuss how to securely observe malware’s network activities.

The method in [8], [9] provides the mimetic Internet environment in a totally isolated sandbox similar to our method. However, these sandboxes do not provide an effective way to observe the vulnerability exploitation while our sandbox effectively redirects attacks to a honeypot with multiple emulated vulnerabilities for efficiently observe the exploitation.

**Our Contribution.** With the totally isolated sandbox, we are able to avoid two kinds of risks. The first is that we can ensure that no attacks will exit the sandbox. Although an access control using IPS or honeywall [24] is effective, it is not reasonable to assume that there is no false negative. Especially when observing the exploitation of vulnerabilities by malware, a failure of blocking outgoing attacks means the infection of a host in the real Internet. Without such a risk, we can fully observe the propagation of malware including how exploit codes involves further propagation steps such as downloading of the executables.

The second is that we can avoid the risk that the addresses of a sandbox (or the proxy it is using) are identified and blacklisted by attackers. If the sandbox frequently accesses C&C servers or download servers that are under control of an attacker, its IP addresses may be spotted out and the analysis may be thwarted.

After all, using totally isolated sandbox or that with the connection to the real Internet is the choice that each malware analyst can make. With the isolated sandbox, he may not be able to observe some of the behaviors, such as interaction with C&C and download servers, but he can be surer that he is not causing any infection to the real Internet and that his analysis activities are totally invisible to the attackers who are in charge of maintaining download servers or C&C servers.

In summary, to the best of our knowledge, our proposal is the first attempt to address and achieve the secure observation of vulnerability exploitation by malware with considerable efficiency. With above consideration, we believe that our proposal can be a useful alternative for malware analyst.

6. Conclusions and Future Works

We proposed a novel sandbox analysis method that utilizes two victim hosts to observe vulnerability exploitation by malware. We have shown an implementation of our method by using a low-interaction honeypot, Nepenthes. Our experiment showed that the implemented sandbox is capable of observing propagation of almost all the pre-screened malware samples.

Although the implemented sandbox showed a certain level of feasibility, it has following limitations, which we
will try to overcome in our future work.

**Diversity of Vulnerabilities:** Nepentes can only handle and emulate vulnerabilities for which there is corresponding vulnerability handler. That means that it can not handle Zero-day exploits. Therefore, we need to implement a second victim with more variety of vulnerabilities. For example, an actual unpatched Windows Machine can be alternatively used as the second victim. Thus the second victim should be carefully implemented to meet users’ requirements. In addition, it is relatively easy for an attacker to detect the existence of Nepentes since it provides an unnaturally large number of vulnerable services. Malware authors could enhance a detection mechanism for such a honeypot and change the behaviors of their malware; however, we are yet to see such complicated detection mechanism implemented on malware. However, it can be a critical issue that we need to work on in the future works.

**Distinguishability:** The distinguisher may not work properly if the same port number is used for propagation and other communication like C&C. For example, if a malware communicates with its herder via HTTP on 80/TCP and in the meanwhile exploits a vulnerability of a web server on 80/TCP, the present distinguisher cannot distinguish between them because the distinction is based on the port number. In order to distinguish between the two, each communication must be closely analyzed in greater detail. We consider the adoption of a multi-path analysis for automated distinction.

**Internet Emulation:** Since our sandbox is completely isolated from the Internet, bots in the sandbox could not receive any C&C commands from their herder unless we emulated them. This means that our sandbox cannot observe malware’s behaviours that are triggered by the C&C commands such as DDoS, file downloads, file uploads, key logger activation, spam mails, etc. We will consider the emulation of herder commands and file downloads in our future works.

**Detection of Exploit Codes and Executables:** Although the traffic analyzer showed certain feasibility in detection exploit codes and executables, there are limitations. The Ashula algorithm for detecting exploit codes may have false negatives. Also, the PE detector for detecting executable codes only works when the executable codes are transmitted over the network without being encoded or masked with a random key string. In order to detect such an executable, we may have to internally monitor the victim machines and extract the code after being decoded.

**Acknowledgments**

The authors would like to thank Mr. Yaichiro Takagi from NICT and Mr. Kouichiro Okada from SecureBrain Corporation for their technical support and dedication in developing the system. Finally, authors would like to thank the anonymous reviewers for their valuable comments.

**References**

[1] P. Baecher, M. Koetter, T. Holz, M. Dorneif, and F.C. Freiling, “The Nepentes platform: An efficient approach to collect malware,” 9th International Symposium on Recent Advances in Intrusion Detection (RAID 2006), pp.165–184, 2006.

[2] M. Bailey, E. Cooke, F. Jahanian, J. Nazario, and D. Watson, “The Internet motion sensor: A distributed blackhole monitoring system,” 12th Annual Network and Distributed System Security Symposium, 2005.

[3] M. Bailey, J. Oberheide, J. Andersen, Z.M. Mao, F. Jahanian, and J. Nazario, “Automated classification and analysis of Internet malware,” Proc. Recent Advances in Intrusion Detection, RAID07, LNCS, vol.4637, pp.178–197, 2007.

[4] F. Bellard, “QEMU, a fast and portable dynamic translator,” 2005 USENIX Annual Technical Conference, FREENIX Track, pp.41–46, 2005.

[5] U. Bayer, C. Kruegel, and E. Kirda, “TTAnalyzer: A tool for analyzing malware,” Annual Conference of the European Institute for Computer Antivirus Research (EICAR), 2006.

[6] H. Father, “Hooking Windows API — Technics of hooking API functions on Windows,” CodeBreakers Journal, vol.1, no.2, 2004.

[7] D. Inoue, K. Yoshioka, M. Eto, Y. Hoshizawa, and K. Nakao, “Malware behavior analysis in isolated miniature network for revealing malware’s network activity,” Proc. IEEE International Conference on Communications, ICC ’08, pp.1715–1721, 2008.

[8] S. Miwa, T. Miyachi, M. Eto, M. Yoshizumi, and Y. Shinoda, “Design and implementation of an isolated sandbox with mimetic Internet used to analyze malwares,” Proc. DETER Community Workshop on Cyber Security Experimentation and Test 2007, 2007.

[9] S. Miwa, T. Miyachi, M. Eto, M. Yoshizumi, and Y. Shinoda, “Design issues of an isolated sandbox used to analyze malwares,” Proc. IWSEC2007, LNCS, vol.4752, pp.13–27, 2007.

[10] K. Nakao, K. Yoshioka, D. Inoue, and M. Eto, “A novel concept of network incident analysis based on multi-layer observations of malware activities,” 2nd Joint Workshop on Information Security (JWIS07), pp.267–279, 2007.

[11] H. Nogawa, “Shellcode detection: Fight against polymorphism,” Joint Information Security Workshop on Internet Monitoring and Analysis (ISWIMA), 2006.

[12] C. Seifert, R. Steenson, I. Welch, P. Komisarczuk, and B.E. Popovský, “Capture — A behavioral analysis tool for applications and documents,” 7th Annual Digital Forensic Research Workshop (DFRWS), 2007.

[13] C. Willems, T. Holz, and F. Freiling, “Toward automated dynamic malware analysis using CW Sandbox,” Security & Privacy Magazine, IEEE, vol.5, no.2, pp.32–39, 2007.

[14] Anubis, http://analysis.seclab.tuwien.ac.at/

[15] NORMAN Sandbox Information Center, http://www.norman.com/microsites/nsic/

[16] Joebox, http://www.joebox.org/

[17] http://www.secure-ware.com/

[18] Win32/MyDoom, http://www.microsoft.com/security/portal/Entry.aspx?name=Win32%2fMyDoom

[19] tcpdump/libpcap, http://www.tcpdump.org/

[20] CERT Advisory CA-2003-20 W32/Blaster worm, http://www.cert.org/advisories/CA-2003-20.html

[21] Microsoft Portable Executable and Common Object File Format Specification, http://www.microsoft.com/whdc/system/platform/firmware/PECOFF.mspx

[22] N. Provos and T. Holz, Virtual Honeypots: From Botnet Tracking to Intrusion Detection, Addison-Wesley Professional, ISBN 0321336321, 2008.

[23] J. Ferguson, D. Kaminsky, J. Larsen, L. Miras, and W. Pearce, Reverse Engineering Code withIDA Pro, Syngress Media, ISBN 159749237X, 2008.
Katsunari Yoshioka received the B.E., M.E. and Ph.D. degrees in Computer Engineering from Yokohama National University in 2000, 2002, and 2005, respectively. From 2005 to 2007, he was a Researcher at the National Institute of Information and Communications Technology, Japan. Currently, he is an Assistant Professor at the Interdisciplinary Research Center, Yokohama National University. His research interest covers wide range of information security, including malware analysis, network monitoring, intrusion detection, and information hiding. He received the Best Paper Award at the 3rd Joint Workshop on Information Security (JWIS 2008).

Daisuke Inoue received his B.E. and M.E. degrees in electrical and computer engineering from Yokohama National University in 1998 and 2000, respectively, and Ph.D. degree in engineering from Yokohama National University in 2003. He joined the Communications Research Laboratory (CRL), Japan, in 2003. The CRL was relaunched as the National Institute of Information and Communications Technology (NICT) in 2004, where he is a senior researcher of the Information Security Research Center. His research interests include security and privacy technologies in wired and wireless networks, incident handling based on network monitoring and malware analysis. He received the Best Paper Award at the 2002 Symposium on Cryptography and Information Security (SCIS 2002).

Masashi Eto received LL.B degree from Keio University in 1999, received the M.E. and Ph.D. degrees from Nara Institute of Science and Technology (NAIST) in 2003, 2005, respectively. From 1999 to 2003, he was a system engineer at Nihon Unisys, Ltd., Japan. He is currently a researcher at National Institute of Information and Communications Technology (NICT), Japan. His research interests include network monitoring, intrusion detection, malware analysis and auto-configuration of the Internetworking. He received the Best Paper Award at the 2007 Symposium on Cryptography and Information Security (SCIS 2007).

Yuji Hoshizawa joined Symantec in 1998, took a position in charge of security research, correspondence to new viruses, and collection and analysis of vulnerability information as the Asia Pacific regional manager of the Symantec Security Response. He has established himself as a top class virus researcher in Japan, and has been contributing to many IT related publications about computer security. Moreover, he gave talks at the various international conference such as Virus Bulletin, EICAR, and AVAR, on the subject of security issues. After leaving Symantec in September 2004, he joined SecureBrain Corporation and took a charge of the present position since October 2004.

Hiroki Nogawa received M.D and Ph.D. degrees in Medicine from Osaka University in 1990 and 1997, respectively. From 2000 to 2004, he was a Lecturer at Cybermedia Center, Osaka University, Japan. Currently, he is a Visiting Professor at Information Center for Medical Sciences in Tokyo Medical and Dental University. Also, he is a Board member of Secureware Inc., which focuses on shellcode detection technology. His recent interest is a broad spectrum of security, from shellcode detection to public policy on security.

Koji Nakao is the Information Security Fellow in KDDI, Japan. Since joining KDDI in 1979, he has been engaged in the research on multimedia communications, communication protocol, secure communicating system and information security technology for the telecommunications network. He is also an active member of Japan ISMS user group, which was established in the 1st Quarter of 2004. He is the board member of Japan Information Security Audit Association (JASA) and that of Telecom ISAC Japan, and concurrently, a Technical Group Chairs (ICSS: information communication system security) of The Institute of Electronics, Information and Communication Engineers. He received the B.E. degree of Mathematics from Waseda University, in Japan, in 1979. He received the IPSJ Research Award in 1992, METI Ministry Award and KPMG Security Award in 2006, and Contribution Award (Japan ITU), NICT Research Award, Best Paper Award (JWIS) and MIC Bureau Award in 2007. He is a member of IPSJ. He has also been a part-time instructor in Waseda University since 2002.