Design and Implementation of an Isolated Sandbox with Mimetic Internet used to Analyze Malwares

Shinsuke MIWA[†,§] Toshiyuki MIYACHI[‡,§] Masashi ETO[†]
Masashi YOSHIZUMI[¶,‡] Yoichi SHINODA[†,‡,§]
[†]*Information Security Research Center, National Institute of Information and Communications Technology, Japan*
[‡]*Hokuriku Research Center, National Institute of Information and Communications Technology, Japan*
[§]*Internet Research Center, Japan Advanced Institute of Science and Technology*
[¶]*School of Information Science, Japan Advanced Institute of Science and Technology*

Abstract

Recent viruses, worms, and bots, called malwares, often have anti-analysis functions such as mechanisms that confirm connectivity to certain Internet hosts and detect virtualized environments. We discuss how malwares can be kept alive in an analyzing environment by disabling their anti-analyzing mechanisms. To avoid any impacts to/from the Internet, we conclude that analyzing environments should be disconnected from the Internet but must be able to make malwares believe that they are connected to the real Internet. We also conclude that, for executing environments to analyze anti-virtualization malwares, they should not be virtualized but must be as easily re-constructable as a virtualized environment. To reconcile these cross-purposes, we designed an isolated sandbox that consists of a mimetic Internet and renewable actual nodes. We implemented a prototype system and conducted an experiment to test the efficiency of our sandbox.

1 Introduction

Malwares[1], such as viruses, worms, and bots, are daily becoming more sophisticated. To confront the malwares threat, it is necessary to observe their behavior, analyze their mechanisms, and identify issues. Isolated sandboxes are good analyzing environments for carrying out such observations because they have tolerance to attacks and infections from the outside.

Isolated sandboxes are now easy to build because of improvements in technologies such as OS and hardware virtualizations. Since malwares often damage analyzing environments, the environments must frequently be rebuilt. Because of this, virtualization technologies have been widely used because analyzing environments introduced to a virtualization technology are easy to rebuild. We had developed a isolated sandbox based on a VMM (Virtual Machine Monitor) that called ***VM Nebula***.[2]

Unfortunately, some recent malwares have anti-analysis mechanisms for examining the environment in which they are being executed. Therefore, such malwares cannot be efficiently analyzed in isolated sandboxes and virtualized environments because they are capable of detecting analyzing environments. Our developed *VM Nebula* could not clearly avoid these issues.

We discuss how malwares that have anti-analysis mechanisms can be safely live observed, analyzed, and identified in analyzing environments. On the basis of the discussion, we propose two approaches. The first is the mimetic Internet, which cannot be recognized as an isolated environment by these malwares. The second is an executing environment that is based on renewable actual (not virtualized) nodes for easy reconstruction. We also designed and implemented a prototype system that incorporates our approaches and used it to conduct an experiment to test its efficiency.

In this paper, we will use the following terminology: an experimental environment for analyzing malwares is called an ***analyzing environment***, a malware on an executable format is called an ***executable instance***, and an environment for executing a malware is called an ***executing environment***. A mechanism designed to avoid any impacts to/from the outside, i.e., the Internet, from/to the analysis in an isolated analyzing environment is called an ***isolated sandbox***. The term ***actual node*** is used to mean an actual computer that is not virtualized. Finally, a virtualized computing environment with OS virtualization technologies or hardware virtualization technologies is called a ***virtualized environment***.

2 Anti-analysis on Malwares

As mentioned above, mechanisms to disrupt analysis have been introduced on the latest malwares. In this section, we describe these anti-analysis mechanisms.

Anti-analysis mechanisms can be classified into two types: obfuscate executable instances, and restrict exe-

cuting environments. We describe these briefly below.

2.1 Obfuscating executable instances

The purpose of obfuscating executable instances is to disrupt static code analysis with offline reversing tools such as disassemblers and debuggers. The two major techniques for disrupting static analysis[3] are:

- Program code obfuscation
- Binary code transformation

These techniques have often been combined to enhance anti-analysis capability. In this paper, we will not elaborate on them because the static analysis is not our target.

2.2 Restricting executing environments

The purpose of restricting executing environments is to disturb live code analysis by analyzing environments such as isolated sandboxes and virtualized environments. The two major steps are:

Detecting executing environment When a malware detects an executing environment as an analyzing environment, it decides whether to execute itself or not. The malware checks whether it is stepping on debuggers or executing itself in isolated sandboxes or in virtualized environments. Then, it carries out a controlling execution of itself and a hiding instance of itself based on the result of its decision to prevent live code analysis.

Controlling execution and hiding instance A malware controls its execution and hides its instance by disturbing the live code analysis when it detects a poorly executed environment, such as an analyzing environment that may be able to analyze it, or an environment too insignificant for its execution. Common techniques for malwares to control their execution are changing their behavior to non-native behavior or halting their execution. And also, some malwares hide their executable instances to disrupt collection of the unpacked instance, which could be used to analyze them.

These steps are usually used sequentially, with the first being used to determine whether or not the second must be implemented.

3 Related Work

Honeypot technologies such as the Honeynet project[4] are major technologies for mimicking sites and hosts on the Internet. Honeypots mimic an analyzing environment as generic hosts and local sites to intruders. The Honeynet project has been developing mimicking technologies with virtualization technologies and mimicking techniques for large-scale sites, which have been using them to analyze actual incidents. Our approach is related to Honeypot technologies, but has some differences. Our current target is automatic intrusions by malwares, and our interest is mimicking parts of the Internet.

Virtualization technologies[5] build a virtualized computer in an actual computer. Therefore, they should provide the following functions: 1) a single computer should perform as well as multiple computers. 2) a virtualized environment, which is generally called a ***guest environment***, should be easily managed by a virtualization technology's executing environment, which is generally called a ***host environment***. 3) the host environment should be easily concealed from the guest environment.

Because of these functions, virtualization technologies have been widely used on high-availability server technologies and testbeds for testing and experimenting with systems.

In the recent years, support mechanisms for virtualization technologies have been introduced on some processors for personal computers. In response to this, malware techniques that conceal themselves using processor virtualization technologies[6] have been developed. These techniques cannot always be detected by a host OS because host OSs sometimes execute in ultra thin host environments[7] prepared by the malware. To counter these techniques, techniques for detecting a virtualized environment using specific processor instructions have been proposed[8]. Different results are returned by these counter techniques when they are running in a virtualized environment. Techniques for avoiding[9] detection have also been discussed. In other words, a cat-and-mouse game is developing into a vicious spiral.

Strider HoneyMonkeys[10] is capable of exploit detection with virtualization technologies. To detect exploitation, it drives the Internet Explorer browser and some programs in a way that mimics a human user's operation, and analyzes behaviors of the drived programs. Many web sites that exploit browser vulnerabilities could be found with it. Although the HoneyMonkeys seems to be efficient to detect exploitation, yet it seems to have issues of anti-virtualization malwares as mentioned the above.

Potemkin Virtual Honyefarm[11] is a honeyfarm with XEN[12] virtual machine monitor which modified to support their virtual honeyfarm architecture. Potemkin goal is high-fidelity honeypots over a large number of IP address, and prototype Potemkin implementation achived high-fidelity host emulation, based on the par-

avirtualization of modified XEN, for hundreds of thousands of IP address while using only tens of physical servers. Although the Potemkin seems to provide real Internet connectivity and high-fidelity emulated hosts to execute specimens, yet it seems to have issues of anti-virtualization malwares as same as the HoneyMonkeys.

4 Our Method

To account for the anti-analysis mechanisms in the malwares mentioned above, we focused on techniques of detecting executing environments. In this section, we will discuss how we can fool the detection, and describe the design of the prototype system.

This paper focuses on developing analyzing environments. Methodologies of static and live code analysis, such as using debuggers, are outside the focus of this paper.

4.1 Requirements

There are two kind of requirements to construct a sandbox for analyzing live codes of malwares, which have the detecting executing environment mechanisms. First is for analyzing malwares, and second is for fooling the detecting mechanisms. Both requirements must be satisfied in the sandbox.

4.1.1 For Analyzing Malwares

Live malware codes might attempt to infect and attack other hosts and sites based on the malware's purposes. Therefore, when there are live codes in a sandbox, the sandbox helps the infections and attacks.If an analyzing environment is directly connected to the Internet, it will suffer from impacts from the Internet because the Internet enables attacks. Therefore, environments for analyzing live malware codes must be isolated from the Internet to avoid impact to/from the Internet.

Since malwares often damage analyzing environments, to observe malware behavior, the analyzing environment must be rebuilt to recover from damages caused by the malware. Therefore, analyzing environments must be easy to rebuild.

In these view, virtualized analyzing environments have the advantages of:

- making it easy to construct an isolated sandbox because a virtualized environment usually has a closed executing environment,
- making it easy to rebuild an environment using snapshoting itself, and rolling back previous states which are provided in many implementations of the virtualization technology.

Even if an analyzing environment would be constructed based on actual nodes instead of virtualized environments, these advantages must be kept or alternative methods for isolation and easy to rebuild must be provided.

4.1.2 For Fooling the Detecting Mechanisms

They must also be constructed so that they are able to fool the detecting mechanisms of malwares:

(1) Check whether or not specific targeted hosts and services are reachable

(2) Detect virtualized environments

There is no simple approach to the reachability check (1) technique because providing reachability from the isolated sandbox causes a troublesome conflict between isolating requirement, which was mentioned the above subsection. Two common approaches are capable of reconciling the conflict. The first, called the online approach, provides reachability to specific hosts and services on demand. In this approach, curious and intelligent traffic filters must be provided because attacking and infecting traffic from malware must not be able to pass through to the Internet. The second, called the offline approach, provides counterfeit reachability. In this approach, counterfeit hosts and services must be prepared based on the malware's targets.

In approaches against the virtualized environment detection (2) technique, the advantages of virtualized environments, which were mentioned the above subsection, must be kept, or alternative methods must be provided. There are also two fundamental approaches. The first is making virtualized environments seem not virtualized. The second is providing the same functions as virtualized environments provide in non-virtualized environments.

Based on these issues, we discuss below how to make an isolated sandbox appear to be connected to the real Internet and how to retain the advantages of virtualized environments in an executing environment based on actual nodes.

4.2 Our Approaches and Functions

In response to the reachability check technique mentioned in the above, we use the offline approach using a function that enables an analyzing environment to pass a malware's reachability check even though it is executed in an isolated sandbox is called the ***mimetic Internet***. This kind of function must mimetically provide targeted hosts, services, and network environments. We chose this approach because, with the online approach, it is difficult to eliminate the risk of violating isolation

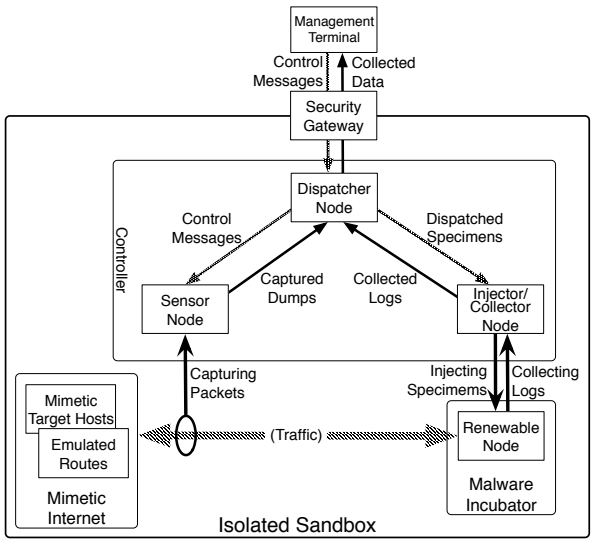

Figure 1: Functional Image of Proposed Environment

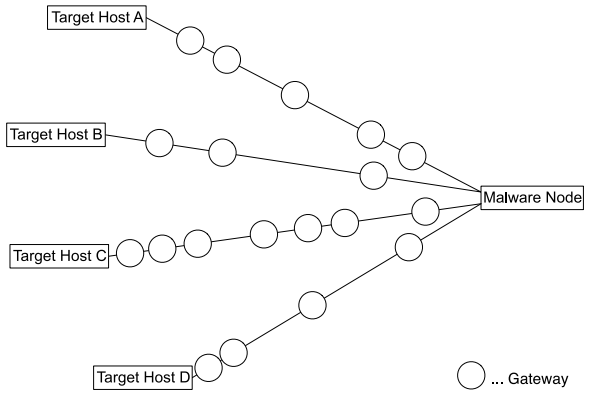

Figure 2: End-Host View of Internet

because traffic filters not only pass through traffic for the reachability check but may also pass attack and infection traffic.

For the virtualized environment detection technique, we use an executing environment based on actual nodes instead of in virtualized environments, which is called the ***malware incubator***, that is also capable of providing the same functionality as a virtualized environment. This is because techniques for concealing virtualizations probably interact with techniques for detecting concealed virtualizations, causing a vicious circle (*cf.* section 3). We attempt to avoid the vicious circle.

Figure 1 shows a functional image of our proposed environment. We discuss these functions in detail below.

4.3 Isolated Sandbox and Controller nodes

An isolated sandbox based on actual nodes can easily be built to function as a strictly disconnected analyzing environment. However, in this disconnected environment, management of the environment through actions such as introducing and executing specimens, and observations such as capturing packet dumps and collecting related logs must be carried out without network connections to the environment.

Our proposed system provides the isolated sandbox with separated VLANs, a security gateway and controller nodes. The analyzing environment is separated on specified VLANs, which are disconnected from other environments, and the security gateway, which is built up between the isolated sandbox and a management terminal, provides communication channels for controlling and observing the isolated sandbox.

According to control messages from the management terminal, controller nodes control the mimetic Internet and the malware incubator, and collect related data. A sensor node provides captured packet dumps with a packet capture software and collects related IDS logs. An injector node injects specimens to the malware incubator and issues commands to executing specimens. A collector node collects related logs from the malware incubator.A dispatcher node dispatchs a specimen to the injector node and collects related data from the sensor node and the collector node.

4.4 Mimetic Internet

To construct a mimetic Internet, what is important is whether or not malwares mistake the mimetic Internet for the real Internet. This may amount to what could be observed in the real Internet from an end-host and how our mimetic Internet mimics the real Internet to the end-host.

We conclude that the end-host view of the Internet could be modeled as a collection of behavior of target hosts involving some services, a count of gateways, and link qualities such as bandwidths and RTTs, for each target host and gateway. Figure 2 shows this model. In the sections below, we use the term target host to mean the targeted host of malwares checking reachability.

According to the model of the end-host view of the Internet, if we could emulate the behavior, number of gateways, and link qualities of each target host and gateway, malwares would misidentify as themselves on the real Internet. Therefore, our proposed mimetic Internet will be composed of:

- Mimetic target hosts
- Emulated routes

We must regard that the mimetic Internet can be constructed on virtualized environments because detect virtualized environment techniques on malwares could only perform to its executing host.

4.4.1 Mimicking Target Hosts

If a malware has information about all the hosts on the Internet and can check them all, the same number of hosts must be mimicked on the mimetic Internet. The malware, however, cannot inspect all the hosts on the Internet because there is far too much information involved. Therefore, the malware should be able to choose target hosts from the following classes and combinations of them:

(1) hosts that are easy to inspect,

(2) principal or fundamental hosts that are always present,

(3) and hosts specific to the malware.

Accordingly, the proposed system uses mimetic target hosts from each class. For the easy-to-inspect (1) class, neighboring hosts on the same network segment and server hosts for major local services such as SMTP, POP, Web, and DNS, which are provided on generic sites, were prepared. For the principal or fundamental (2) class, global DNS services such as root DNSs, `time.windows.com`, which is a default NTP server for Microsoft Windows XP hosts, famous search engines *Yahoo!*[1] and *Google*[2], and Microsoft Web site[3] and *Windows Update*[4] site were prepared. A dynamic configuration of specific target hosts based on the results of a pre-experiment that executes the malware and collects communication history were discussed as examples of the class (3) specific to the malware.

4.4.2 Emulating Routes

On the real Internet, an observed route would perhaps be changed for each observation because the route would be chosen by routing mechanisms from many possible routes. However, the observed route usually has only a single gateway count because there usually is no other route to the target host, and the route should be stably chosen.

The proposed method emulates a route to a target host using the route emulator and the link quality emulator. The route emulator emulates as each target host transits a fixed count of gateways, and the link quality emulator emulates bandwidths and delays to each gateway using a network emulation software.

4.5 Actual Nodes Instead of Virtualized Environment

An actual node usually takes more time and work to rebuild. To solve these problems, our malware incubator was designed as discussed below.

4.5.1 Malware Incubator with Renewable Node

In live code analysis of malwares, an executing environment must frequently be rebuilt to recover from damage caused by malwares. This is time consuming and requires much works.

The proposed method will use a renewable node technique to reduce the amount of work required. In this technique, an installed executing environment would be previously recorded into a disk-image. Then an actual node can be rebuilt from the disk-image of the same executing environment when the executing environment sustains damage.

And note that disk-images must be separated from a executing environment to avoid damages effect disk-images. Furthermore, to provide clean environment after renewing, the executing environment must be rebooted forcedly because some specimens would often disrupt rebooting its environment.

4.5.2 Parallel Sandboxing

The renewable node technique should reduce the amount of work required. However, the time required cannot easily be reduced.

Therefore, in our method, the environment would be built up on large-scale practical estimation testbeds, such as StarBED[13], to solve these problems. The testbed is composed of an enormous number of the same kind of actual nodes. The proposed method will use a parallel sandboxing technique. In this technique, the time required to rebuild, which naturally involves stopping the analysis, could be reduced because many isolated sandboxes would be constructed on the large scale testbed, and then many specimens can be executed on each sandbox in parallel.

5 Implementation

A prototype system of our analyzing environment was implemented on StarBED. In this section, we describe the implementation of our prototype system. Figure 3 shows an overview of the implementation.

5.1 Isolated Sandbox and Controller nodes

An isolated sandbox must be disconnected from the Internet. In the implementation, it was isolated using sep-

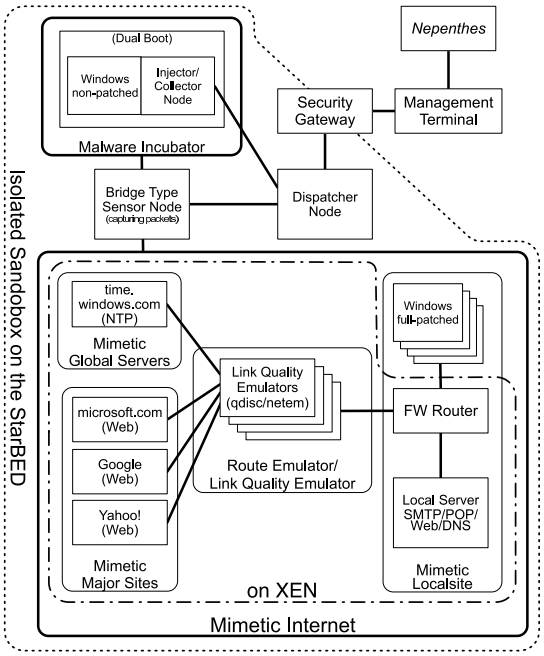

Figure 3: Overview of Implementation

arated VLANs. We verified its separateness with a penetration test using `nmap`[14]. A security gateway is implemented as a simple firewall router. The isolated sandbox can be managed with controller nodes via the security gateway.

Controller nodes are:

- A sensor node using `tcpdump` and `snort`[15], which was in bridge mode, for capturing packets
- An injector/collector node is constructed on the malware incubator
- A dispatcher node using `ssh` and `scp` for controlling the sensor and the injector/collector

The injector/collector node is implemented on same node of a malware incubator as a second boot OS (Linux). The dispatcher selects the malware incubator or the injector/collector which will be booting. When boot up as the injector/collector, the injector/collector collects related logs from a hard disk area of the malware incubator, renews the malware incubator using a clear disk-image, and injects a specimen to the hard disk area.

5.2 Mimetic Internet

The implementation of mimetic Internet included:

- Mimetic local site with four actual nodes that were installed in fully patched Windows for local hosts, SMTP/POP/Web/DNS local servers, and a firewall router
- Four simple link quality emulators with `qdisc` (netem)[16] to emulate routes to mimetic global servers and sites
- Mimetic *time.windows.com*, which is a default NTP server for Windows XP, as an NTP server on the mimetic global server
- Mimetic *Microsoft.com*, *Google*, and *Yahoo!* as web servers on the mimetic major sites

And note that there are no support mechanism for specific mimetic hosts for each malware because no mechanism for dynamic configuration of specific targets was implemented.

Many mimetic Internets can be easily built up with a shell script and image files of mimetic environments for XEN because all of the above except fully patched Windows for local hosts (shown in figure 3 as "on XEN") are constructed on a XEN host as virtualized hosts in the implementation.

5.3 Malware Incubator

A malware incubator was installed in non-patched Microsoft Windows XP professional for executing specimens with system monitoring tools for Microsoft Windows[17](FileMon, RegMon, TCPview, and Process Explorer). According to the dispatcher's selection, when boot up as the malware incubator, the malware incubator starts up some logging tools and executes a specimen which was set up by the injector.

Firstly, the dispatcher boots up the injector/collector. After collecting related logs, injecting the specimen and cleaning the malware incubator, the dispatcher reboots as the malware incubator. And after 5 minutes wait for executing the specimen, the dispatcher forcedly reboots the malware incubator as the injector/collector using IPMI[18]. Then, next specimen will set up, and repeats the above steps.

At present, our malware incubator was implemented with an injector/collector on a single node. The reason for dual role on a single node is for collecting logs from the malware incubator and cleaning them, because it may not be able to controll by the dispatcher when it was infected which means it was controlled under a malware. To solve this problem, the dispatcher forcedly reboots the malware incubator using IPMI.

5.4 Parallel Sandboxing

Many isolated sandboxes can easily be prepared for parallel sandboxing because the mimetic Internets were con-

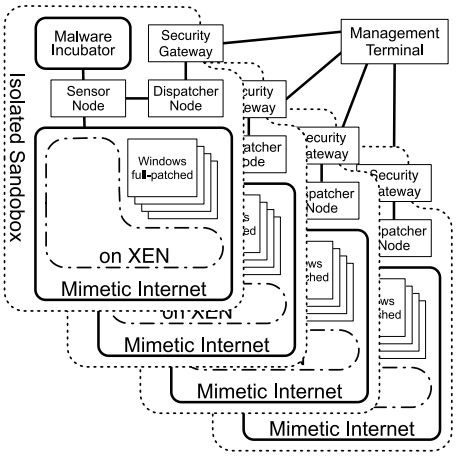

Figure 4: Parallel Sandboxing

Table 1: Classification of Specimens

Classifications	Counts
Collected specimens (unified by MD5)	581
Windows (or MS-DOS) executables	581
Known specimens (scanned by *ClamAV*)	508
(Known variants)	168
Unknown specimens (filtered out by *ClamAV*)	73

Table 2: Observed Behavior

Behavior	#	(Unknown)
HTTP access to *Google*	14	5
HTTP access to *Microsoft.com*	7	3
HTTP access to *Yahoo!*	0	0
NTP access to *time.windows.com*	0	0
DNS retrieve specific hosts	351	37

# of specimens

structed on a XEN host as virtualized hosts and large number of the same kind of actual nodes were prepared on StarBED.

The management terminal manages multiple isolated sandboxes, such as dispatching specimens to each sandbox. Figure 4 shows a concept of the parallel sandboxing.

6 Evaluation

We conducted an experiment to test the efficiency of our method. In this section, we describe the experiment and its results.

6.1 Experiment

The purposes of the experiment were to determine what kinds of malwares the mimetic Internet could fool. We experimented each specimen without the mimetic Internet and with the mimetic Internet, because differences of observed behaviors show that our mimetic Internet could effect to specimen's behaviors.

To determine the things, we executed some specimens which had been collected in *Nepenthes*[19] and were filtered out by *ClamAV*[20] environment on our sandbox implementation mentioned in the above section. Then, we observed the behavior of their live codes in the environment with `tcpdump`, `snort`, and system monitoring tools and the screen capture tool for Microsoft Windows.

Table 1 shows classification of collected specimens. We had been collecting 581 specimens, which included 73 unknown specimens, on *Nepenthes* from November 10, 2006 to March 2, 2007. We analyzed all of 581 specimens, all of which are Windows or MS-DOS executables.

We should note that this experiment did not focus to the renewable actual nodes for reducing required re-experiment time interval.

6.2 Results

Table 2 shows observed behaviors of specimens. We observed that 21 specimens, which included 8 unknown specimens filtered out by *ClamAV*, fooled by our mimetic global sites and global servers, accessed our mimetic Internet. In the observations, the expected behaviors, accessing *Google* or *Microsoft.com* via HTTP, were observed in some specimens. In another behavior, which was expected and observed but not addressed, 349 specimens, which included 36 unknown specimens, attempted to retrieve DNS each specific target. This shows that our mimetic Internet can fool malwares and that dynamically introducing targets is an important issue. And some expected behaviors, accessing *Yahoo!* via HTTP or *time.windows.com* via NTP, were not observed from all specimens. Furthermore, we should notice that an another expected behavior, attempts to probe network routes, did not analyzed whether did or not on this experiment.

Table 3 shows a number of specimens, which were observed different behavior between without the mimetic Internet and with the mimetic Internet, about a number of packets by each protocol. The column "Increase#" shows a number of specimens, which a number of packets increased in observing with the mimetic Internet, "Decrease#" shows decreased with the mimetic Internet, and "Same#" mean same number of packets. This result shows that almost malwares, except only one specimen[5], would change behavior according to result of checking

Table 3: Without vs. with Mimetic Internet (differences in # of packets)

Protocol	Increase#	Same#	Decrease#
ARP	508	56	17
ICMP	103	441	37
UDP	578	1	2
TCP	215	366	0
(Total)	569	1	11

# of specimens

Table 4: Without vs. with Mimetic Internet (# of retrieved domains)

Specimens	w/o	w/
unknown	6 (12)	37 (102)
Microsoft.com	0 (0)	7 (26)
www.google.com	0 (0)	14 (28)
Total	107 (123)	351 (658)

# of specimens (# of retrieved DNS domains)

connectivity to the Internet, and our mimetic Internet can fool them.

Finally, Table 4 shows a number of specimens, which were observed retrieve DNS without/with the mimetic Internet. The line "unknown" shows unknown specimens which filtered out by *ClamAV*, "Microsoft.com" shows specimens which attempt accessing *Microsoft.com* web site, and "www.google.com" shows specimens which attempt accessing *Google* web site. The column "w/o" shows a number of specimens which retrieve DNS without the mimetic Internet, and "w/" shows them with the mimetic Internet. Numbers into a couple of bracket shows a number of retrieved DNS domains. This result shows would change behavior about retrieving DNS according to result of checking connectivity to the Internet, and our mimetic Internet can fool them.

Thus, we can conclude that the mimetic Internet is capable of changing behaviors of some malwares, which have mechanisms of checking connectivity to the begin. However, to insure that these changes of behaviors equal being fooled, we have to analyze in detail behaviors of each malware. We also aware that this experiment and results could be confirmed our concept is whether effective or not, but also did not be confirmed it could be satisfied or not.

7 Conclusion and Future Work

We focused on anti-analysis mechanisms of malwares and discussed how we can fool these mechanisms in an isolated sandbox. We used the mimetic Internet and the malware incubator with a renewable actual node. We also conducted an experiment on StarBED to test our prototype system. The results of the experiment show that the mimetic Internet is effective, although some improvements must be made.

In the future we plan to:

- Demonstrate that malwares that contain countermeasures against virtualized environments can be executed in our sandbox
- Design the dynamic introduction of specific target hosts for each malware to the mimetic Internet
- Design the mimetic dynamic contents generation on mimetic servers
- Enable specimens to access cryptographic services such as SSL
- Enable specimens to download files and join command through real networks from the isolated sandbox
- Propose applications of the mimetic Internet such as experimenting with IP trace-back on routing parts of the mimetic Internet and generating synthetic traffic patterns from the mimetic Internet.

We hope that by using our methods, malwares, which have anti-analysis mechanisms, such as the ability to detect isolated sandboxes and virtualized environments, will be analyzed correctly.

Acknowledgements

The authors give special thanks to Takashi UENO, a master's degree candidate at the Japan Advanced Institute of Science and Technology (JAIST) for his assistance in building, operating, and collecting specimens on a nepenthes environment.

References

[1] E. Skoudis with L. Zeltser, "MALWARE – Fighting Malicious Code –", Prentice Hall PTR, ISBN 0-13-101405-6, Pearson Education Inc., 2004.

[2] S. MIWA and H. OHNO, "A Development of Experimental Environments "SIOS" and "VM Nebula" for Reproducing Internet Security Incidents", Journal of the National Institute of Information and Communications Technology, Vol.52 Numbers 1/2 (pp.23-34) 2005, ISSN 1349-3205, Oct. 2005.

[3] E. Eilam, "Reversing: Secrets of Reverse Engineering", ISBN 0-7645-7481-7, Wiley Publishing, Inc., 2005.

[4] The Honeynet Project, "Know Your Enemy — Revealing the Security Tools, Tactics and Motives of the Blackhat Community", Addison Wesley, ISBN 0-201-74613-1, Pearson Education Corp., 2002.

[5] J. E. Smith and R. Nair, "Virtual Machines — Versatile Platforms for Systems and Processes", Morgan Kaufmann, ISBN 1-55860-910-5, Elsevier Inc., 2005.

[6] S. T. King, P. M. Chen, Y. Wang, C. Verbowski, H. J. Wang, and J. R. Lorch, "SubVirt: Implementing malware with virtual machines", *In Proceeding of IEEE Symposium on Security and Privacy*, May 2006.

[7] "Introducing Blue Pill", ⟨URL: `http://theinvisiblethings.blogspot.com/2006/06/introducing-blue-pill.html`⟩.

[8] "Red Pill... or how to detect VMM using (almost) one CPU instruction", ⟨URL: `http://invisiblethings.org/papers/redpill.html`⟩.

[9] "Blue Pill Detection!", ⟨URL: `http://theinvisiblethings.blogspot.com/2006/08/blue-pill-detection.html`⟩.

[10] Y. Wang, D. Beck, X. Jiang, R. Roussev, C. Verbowski, S. Chen, and S. King, "Automated Web Patrol with Strider HoneyMonkeys: Finding Web Sites That Exploit Browser Vulnerabilities", *In Proceedings of the Network and Distributed System Security Symposium*, NDSS 2006, Internet Society 2006, Feb. 2006.

[11] M. Vrable, J. Ma, J. Chen, D. Moore, E. Vandekieft, A. C. Snoeren, G. M. Voelker, and S. Savage, "Scalability, Fidelity, and Containment in the Potemkin Virtual Honeyfarm", *In Proceedings of 20th ACM Symposium on Operating Systems Principles*, SOSP 2005, Oct. 2005.

[12] S. Crosby, D. E. Williams and J. Garcia, "Virtualization With Xen: Including XenEnterprise, XenServer, and XenExpress", Syngress Media Inc.,ISBN 1-597-49167-5, 2007.

[13] T. Miyachi, K. Chinen, and Y. Shinoda, "StarBED and SpringOS: Large-scale General Purpose Network Testbed and Supporting Software", Valuetools2006, ACM Press, Oct. 2006.

[14] INSECURE.ORG, "Nmap - Free Security Scanner For Network Exploration & Security Audits", ⟨URL: `http://insecure.org/nmap/`⟩.

[15] B. Caswall, J. Beale, J. C. Foster and J. Faircloth, "Snort 2.0 Intrusion Detection", ISBN 1-9318-3674-4, Syngress Publishing, 2003.

[16] S. Hemminger, "Network Emulation with NetEm", *In Proceedings of Linux Conf Au 2005*, Apr. 2005.

[17] Microsoft TechNet, "Sysinternals Utilities: System Information", ⟨URL: `http://www.microsoft.com/technet/sysinternals/systeminformationutilities.mspx`⟩.

[18] Intel Corporation, "IPMI v2.0 specifications Document Revision 1.0", 2004.

[19] P. Baecher, M. Koetter, T. Holz, M. Dornseif, and F. Freiling, "The Nepenthes Platform: An Efficient Approach to Collect Malware", 9th International Symposium On Recent Advances In Intrusion Detection, RAID06, LNCS 4219, Springer, Sept. 2006.

[20] T. Kojm, "Clam AntiVirus", ⟨URL: `http://www.clamav.net/`⟩.

Notes

[1] `http://www.yahoo.com/`
[2] `http://www.google.com/`
[3] `http://www.microsoft.com/`
[4] `http://windowsupdate.microsoft.com/`
[5] It seems that the specimen is an incomplete part of executable instance. The specimen was not taken any communications.