Measuring Anti-analysis Techniques in Malware

Minho Kim[1], Haehyun Cho[2]*, and Jeong Hyun Yi[2]
[1]School of Software, Soongsil University, Seoul, 06978, Korea
alsgh5016@gmail.com
[2]School of Software, Soongsil University, Seoul, 06978, Korea
haehyun@ssu.ac.kr, jhyi@ssu.ac.kr

Abstract

A lot of malware is reported to the malware analysis system. Several cases of detecting the analysis environment are found. These malware use anti-analysis techniques. Malware that uses the analysis prevention technique detects the analysis environment. If malware detects the analysis environment, they don't act maliciously, resulting in false-positive. Therefore, to reduce false-positive, it is significant to detect malware that uses analysis prevention techniques.

In this paper, before proposing a system that detects and analyzes malware using analysis prevention techniques, we investigated malware using analysis prevention techniques. As a result of analyzing the VirusShare dataset from 2018 to 2020, we found anti-VM at about 2.78% and anti-Sandbox at about 55%. Based on this, we analyzed the trend of analysis prevention techniques used in malware.

Keywords: Anti-analysis, Windows Malware, Anti-VM, Anti-Sandbox

1 Introduction

There is a ton of malware reported to antivirus vendors every day [19, 30]. To safely and automatically analyze the surge of malware, security analysts use various sandboxes. The sandbox is an isolated execution environment implemented to test the execution of programs using virtualization techniques for malware analysis [27, 37, 9, 19, 53, 23, 52, 32]. The sandbox analyzes behaviors of malware by monitoring whether the target sends sensitive information to an external server, modifies important files in the system, and which system calls are called. As sandboxes become more common, malware authors use anti-analysis techniques to prevent malware from being detected in the sandbox [32]. Malware using the anti-analysis technique detects the malware analysis sandbox and performs different actions according to the detection result [32, 6, 30, 27]. For example, if malware detected the sandbox, it can interfere with the analysis by disguising it as a normal program or forcibly terminating it without performing any malicious behavior. As a result, the analysis process can falsely detect the malware as a normal program, and thus, the malware can live for a long time without being detected, causing extensive damage to a lot of users. Therefore, is a matter of great importance to thwart such anti-analysis techniques used in malware to protect potential victim users.

There are various anti-analysis techniques. Among them, there are anti-VM techniques and anti-sandbox techniques as techniques for detecting an execution environment where a program is running. An anti-VM technique is to detect a virtual execution environment implemented by software by checking hypervisor vendors, device drivers, services running in the systems, and registry values [37]. A sandbox detection technique is to detect a malware analysis environment based on virtual machines and cloud

H. Kim, F. Palmieri, T.-Y. Youn (eds.), *The 5th International Symposium on Mobile Internet Security (MobiSec'21)*, Jeju Island, Republic of Korea, October 7-9, 2021, Article No. 25, pp. 1-12
*Corresponding author: School of Software, Soongsil University, Seoul, 06978, Korea, Tel: +82-28287360

environments [7]. Except for similar techniques used to detect anti-VM, sandbox detection techniques can scan hardware information or require user interactions [19, 33].

In this paper, we first investigate anti-analysis techniques. We, then, propose a malware analysis framework that detect malware using anti-VM and anti-Sandbox techniques to bypass them. Based on the results of the investigation, we perform measurements of anti-analysis techniques used in real-world malware. To this end, we used dataset collected from 2018 to 2020 by using VirusShare [48]. Finally, we present our analysis results.
In summary, this paper makes the following contributions.

- We analyzed previous research work to collect and classify anti-analysis detection techniques used in malware.
- We analyzed and evaluated the trends in the anti-analysis technique used by real-world malware.

2 Background

In this section, we introduce known anti-analysis techniques and previously proposed approaches for analyzing malware equipped with anti-analysis techniques.

2.1 Virtual Machines and Sanboxes

A virtual machine (or guest) refers to a computing environment implemented in software that can use the same functions as a commonly used computing environment (or host) [50]. Virtual machines are implemented with various technologies such as bare metal hypervisors, host-based hypervisors, and emulators [20]. Bare metal hypervisors aims to implement an execution environment identical to the host environment [7]. KVM [28], Hyper-V [31], and Xen [38] are virtual machines based on bare metal. BareBox [26], BareCloud [27], and Joe Sandbox [25] are implemented based on bare metal. The host-based hypervisor virtual machine is VMware [49], VirtualBox [35], Parallels [24], etc., and the virtual machine runs on the host environment. Ether [18], Norman Sandbox [45], and CWSandbox [15] are host hypervisor-based sandboxes. Emulator [7] is software for simulating functions or hardware. Emulators for virtual machines are divided into the case of simulating the CPU and memory and the case of simulating the API of the operating system. As virtual machines based on emulators, QEMU [40] and Bochs [12] exist. PANDA [36], PyREBox [39], DECAF [17], and S2E [42] are sandboxes based on emulators.

A sandbox is an environment in which programs can execute using virtualization technology [23, 53]. A sandbox environment isolates guests for safe analysis. Malware in the sandbox cannot attack the host environment. To analyze malware, we use the malware analysis sandbox [52, 53, 19]. Sandbox systems execute malware to monitor whether the malware behaves maliciously, such as in an attempt to access system files and registry keys. Cuckoo Sandbox, Joe Sandbox, Hybrid Analysis, and Triage are representative malware analysis sandboxes. The virtual machines and sandboxes presented in Table 1 and Table 2, and they are widely used in malware analysis [51, 44, 46].

Anti-VM technique checks values related to virtual machines. It detects by scanning device drivers, bios, files, registry, etc. Another method is to compare the execution results of instructions that return different execution results in the host and the guest.

There are various methods for detecting sandboxes [52, 33, 19, 32]. The history detects when there are too few traces of usage compared to the general user environment. Execution is detected when the execution is changed when the host uploads the program to the sandbox. Hardware measures the number of CPU cores and RAM size and detects if it is small compared to the general user environment.

Table 1: Virtual Machines' Hypervisor Type.

VM	Type
KVM	Bare Metal
Hyper-V	Bare Metal
Xen	Bare Metal
VMware Esxi	Bare Metal
VMware	Host Hypervisor
VirtualBox	Host Hypervisor
Parallels	Host Hypervisor
Virtual PC	Host Hypervisor
QEMU	Emulator
Bochs	Emulator

Table 2: Sandboxs' Hypervisor Type.

Sandbox	Type
BareBox	Bare Metal
BareCloud	Bare Metal
Joe Sandbox	Host Hypervisor
Ether	Host Hypervisor
Norman Sandbox	Host Hypervisor
CWSandbox	Host Hypervisor
PANDA	Emulator
PyREBox	Emulator
S2E	Emulator
Cukoo Sandbox	All

Table 3: A Summary of Previous Work against Anti-analysis Techniques.

Article	Idea	Detection Method	Evaluation
[26]	A bare metal system that can analyze malware that detect VMware and QEMU virtual machines	Number of System Call	Comparison of the number of system calls for each system
[13]	Investigate instructions that can detect virtual machines in malware	Instructions	If an instruction exists in the disassembled result, it is detected
[27]	Comparison of execution logs generated by different systems	Execution and network activity log	Compare and evaluate whether the malware detects the sandbox by comparing the behavior of each sandbox.
[29]	Malware analysis that detects virtual machines using heuristic patterns	Heuristic patterns	Analysis of malware that detects virtual machines using instructions and strings
[14]	Malware analysis that detects virtual machines using signatures	API, Instruction, String	Measure the rate of detecting virtual machines by type of malware
[52]	Define the sandbox detection technique after collecting the sandbox fingerprint using the fingerprint collection program	Fingerprints related to Hardware, History, and Execution	Define an effective sandbox detection technique by collecting and classifying fingerprints for many sandbox services
[33]	Define sandbox detection techniques by collecting fingerprints for host and sandbox environments	Fingerprints related to History	Create a classification model using the collected data and evaluate its performance
[19]	A system that randomly generates artifacts based on user behavior	User behavior	Create artifacts based on user behavior and compare them with the host environment
[32]	An analysis system that avoids human-driven detection techniques	Human-Driven Behavior	Comparison of the behavior of sandbox detection malware through sandbox environment change

User-interaction detects whether a user is using an input device by measuring movement and speed. In addition, there are methods to detect well-known analysis modules such as agent.py in Cuckoo sandbox, and methods to examine system-specific values such as ProductId in Windows systems. Therefore, to detect malware that evades virtual machines and sandboxes, we investigate anti-analysis techniques for detecting virtual machines and sandboxes.

2.2 Previous Approaches against Anti-analysis Techniques

Various static and dynamic analysis methods were proposed for analyzing malware that uses anti-analysis techniques [22, 41, 9, 37]. The static analysis approaches investigated instructions with call graphs or control flow graphs, analyzed the API called, and found string information [22, 41, 9]. However, if a packer or an obfuscation technique is applied to malware, in many cases, static analysis is impossible without a proper unpacker or deobfuscation tool [21, 13, 41].

Dynamic analysis methods observe programs' behaviors, monitoring API calls during execution, checking function parameters, tracing command execution, and more [22, 41]. Yokoyama et al. collected fingerprints for all sandbox services and classified them into Hardware, History, and Execution. The features are effective for sandbox detection [52]. Miramirkhani et al. measured 'wear and tears' that occur on real machines. Based on this, they proposed a new sandbox detection technique [33]. These authors

Table 4: Anti-VM Detection Signature

Target System	Category	Type	Fingerprint
VMware	Instruction	Check VMware I/O Port	"in eax, dx"
Generic	Instruction	Check Hypervisor Vendor	"VMwareVMware", "VBoxVBoxVBox", "QEMU Virtual CPU"
VirtualBox	Process	Check list of Process	"vboxservice.exe", "vboxtray.exe"
VirtualBox	Driver	Check Driver File exists	"VBoxMouse.sys", "VBoxGuest.sys", "VBoxSF.sys", "VBoxVideo.sys"
VirtualBox	Registry	Check Registry key exists	"SOFTWARE\Oracle\VirtualBox Guest Additions", "HARDWARE\ACPI\DSDT\VBOX__"
Generic	BIOS	Check BIOS value	"GetSystemFirmwaretable()", "SELECT SMBIOSBIOSVersion FROM Win32_BIOS"
QEMU	Hardware	Check Hardware Info	"SELECT * FROM Win32_VideoController", "SELECT * FROM Win32_PointingDevice"

collected information from sandbox services. Based on this, they proposed anti-sandbox techniques and demonstrated the performance of techniques. To avoid anti-sandbox techniques proposed in previous studies, Feng et al. proposed a system that automatically generates user activity artifacts to solve the problem that the sandbox is easy to detect because there are no artifacts related to user activity in the sandbox service [19].

3 Method and Analysis Results

In this section, we demonstrate our approach to study anti-analysis techniques used in real-world malware and preset analysis results.

3.1 Our Goal

First off, we aim to analyze known anti-VM and anti-sandbox techniques. To this end, we investigated the anti-analysis techniques from malware reports and previous work. We, then, define signatures and patterns of existing anti-analysis techniques. Based on the analysis result, we analyzed the trend of the anti-analysis techniques used in real-world malware.

3.2 Anti-VM Detection Signatures

Table 4 contains a list of major virtual machine detection signatures that we found from multiple sources. Among the anti-VM detection methods, the detection by instruction includes the `IN` instruction that uses an I/O port that exists between the host and the guest machine and the `CPUID` instruction that checks the hypervisor vendor. The above instructions have in common is that execution results are returned differently in the host and guest environments. These instruction execution results can detect the virtual machine. In additio, Branco et al. employed instructions related to the Interrupt Descriptor Table (IDT) [13]. IDT-related instructions' execution results are different between a host machine and a guest similar to the `IN` and `CPUID` instructions. However, in these days, with the advancement of virtualization technologies, it is impossible to distinguish between a host and a guest machine by using the IDT-related instructions [8]. Therefore, we do not use such instructions as anti-analysis signatures in this work. Furthermore, there are other ways to detect virtual machines by using processes, drivers, registries, the BIOS, and hardware information in a system. Each technique to detect a virtual machine exploits such characteristics that clearly distinguishes it from a host environment. For example, if a string related to a specific virtual machine such as "VMware" or "VirtualBox" is stored, malware can detect it as a virtual machine.

Table 5: Anti-Sandbox Detection Signature

Category	Description	Fingerprint
Hardware	Check the number of CPU core	"SELECT NumberOfCores FROM Win32_Processor"
Hardware	Check size of ram	"SELECT TotalPhysicalMemory FROM Win32_ComputerSystem"
Environment	Check known DLL file exists	"sbiedll.dll", "pstorec.dll" "snxhk.dll"
Environment	Check Display Resolution	"GetSystemMetrics(SM_CX/YSCREEN)"
Environment	Check known host name	"GetComputerName()", "GetUserName()"
User-interative	Check mouse movement	"GetCursorPos()"
User-interactive	Check mouse click	"GetAsyncKeyState(0x1/0x2/0x4)"
History	Check user's temp files exists	"GetEnvironmentVariable("TEMP")"
History	Check the number of connected USB devices	"SYSTEM\ControlSet001\Enum\USBSTOR"
History	Check the number of installed Programs	"SOFTWARE\Microsoft\Windows\CurrentVersion\Uninstall"

3.3 Anti-Sandbox Detection Signature

Table 5 shows a list of major signatures used for sandbox detection. Sandbox detection methods are divided into 4 categories: a method of checking hardware; a method of checking execution environment; a method of checking user interaction; and a method of checking history [34, 47, 4, 11, 3, 5, 1, 2, 10, 16, 43]. The method of checking hardware takes advantage of the fact that the most of sandbox environments are built and used to analyze much malware in parallel. To be specific, this method utilizes hardware limitations such as the number of CPU cores and RAM size are clearly distinguished from the user environment. The method to check the execution environment information is to find a module such as a specific library file that exists only in a sandbox, to check the display resolution such as 4:3 observed in a virtualization environment, or to check the hostname of a well-known sandbox environment. The method of examining user interaction uses abnormally fast mouse movement and fast click speed measured during the reverse Turing test in the motion control module implemented in the script form. It is detected using the point that the user controls input devices such as mouse and keyboard and the reverse Turing test is distinguished. The method of checking the history is to check temporary files, the number of connected USB devices, and the number of installed programs. This is because a snapshot machine restores the sandbox state before the execution of the malware after analyzing malware, removing records accumulated as the malware executes.

3.4 Methodology

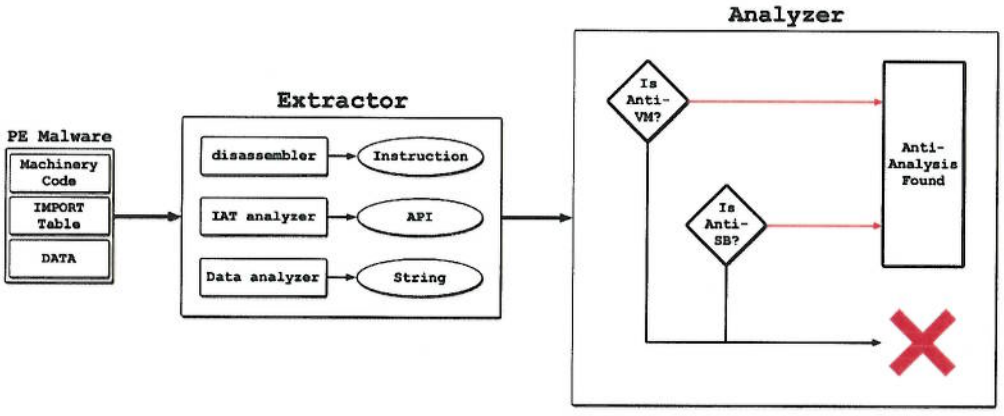

Figure 1: The proposed framework to study anti-analysis techniques.

We collected malware as executable files, from which we can find machine instructions, APIs and

Table 6: Packing Detection Result.

Year	Total	Packed	Ratio (%)
2018	119,952	34,173	28.49
2019	120,942	35,368	29.24
2020	173,785	55,895	32.16

strings used in malware. Based on such information, we analyzed the malware for checking whether they use anti-VM and anti-sandbox techniques or not by using anti-analysis signatures and patterns that we defined. Figure 1 illustrates our framework to study anti-analysis techniques in the wild. The extractor module extracts instructions translated from machine language code into assembly language, API information recorded in the import table, and string information existing in the data section. The analyzer module checks whether the extracted information contains anti-analysis signatures and patterns related to anti-VM and anti-sandbox techniques. When the analyzer finds the signatures and patterns in malware, we classify it as anti-VM or anti-sandbox malware.

3.5 Dataset

In this work, we collected Windows malware from VirusShare (from January 2018 to September 2020) and analyzed the dataset based on the anti-analysis signatures that we discovered from multiple sources [34, 47, 4, 11, 3, 5, 1, 2, 10, 16, 43]. Our dataset consists of 414,679 Windows malware (PE files) in total (2018: 119,952, 2019: 120,942, 2020: 173,785).

3.6 Packing Detection Result

Before we analyze anti-analysis techniques from our dataset, we first found packed malware to classify them.

As shown in Table 6, packed malware accounted for 28,49%, 29.24%, and 32.16% of all malware from 2018 to 2020. The proportion of packed malware shows a trend that gradually increases as time goes by. In Table 7, Unknown Case is a case where well-known packers are duplicated or unknown packers are applied. We found that unknown case accounted for 52%, 68%, 53% of all packed malware from 2018 to 2020. Next, we found UPX(2018: 12,098, 2019: 4,881, 2020: 7,820), Armadillo(2018: 3,197, 2019: 4,086, 2020: 15,176), VMProtect(2018: 534, 2019: 697, 2020: 1,136), and PECompact(2018: 160, 2019: 664, 2020: 674).

3.7 Anti-VM Detection Result

From 2018 to 2020, malware using virtual machine detection was detected at 2911, 3011, and 5609, with an average of 2.78%. In the virtual machine detection experiment, VMware, VirtualBox, QEMU, Parallels, Xen, VirtualPC, Hyper-V, and Bochs were used to analyze whether malware detects virtual machines by targeting various virtualization solutions. As a result, the most detected virtualization solutions were analyzed in order of VirtualBox-VMware-QEMU. Although the share of VMware virtual machines is overwhelmingly high, it seems that malware creators analyzed the fact that there are many cases where malware analysis sandboxes have been implemented using open sources virtualization solutions such as VirtualBox and QEMU, and this seems to have been reflected in the malware.

Table 7: Packing Detection Detailed Result.

Packer	2018	2019	2020
Unknown	17,856	24,063	29,719
UPX	12,098	4,881	7,820
Armadillo	3,197	4,086	15,176
VMProtect	534	697	1,136
PECompact	160	664	674
MPRESS	112	484	639
ASPack	140	344	227
FSG	4	20	284
Themida	12	26	167
NSPack	20	35	22
PEtite	10	18	19
NeoLite	13	14	5
ASProtect	16	9	3
Packman	0	27	1
eXpressor	1	0	2
PESpin	0	0	1
Enigma	0	0	0

Table 8: Anti-VM Detection Result.

Year	Total	Detected	Ratio (%)
2018	119,952	2,911	2.42
2019	120,942	3,011	2.49
2020	173,785	5,609	3.23

Table 9: Anti-Sandbox Detection Result.

Year	Total	Detected	Ratio (%)
2018	119,952	73,382	61.18
2019	120,942	66,290	54.81
2020	173,785	91,975	52.92

3.8 Anti-Sandbox Detection Result

From 2018 to 2020, 73382, 66290, and 91975 malware using sandbox detection were detected in an average of 55% of malware. In the sandbox detection experiment, we analyzed whether the malware is using many detailed techniques classified into hardware, execution environment, user interaction, and history. In the hardware category, it can be seen that the number of CPU cores is checked the most, and the weight of RAM size measurement is gradually increasing. In the execution environment category, display resolution, hostname check, and known DLL module check are frequently used in the order. The display resolution seems to take advantage of the unusual environment, such as the default display ratio of 4:3 in sandboxes. In addition, the proportion of methods for verifying the hostname is increasing significantly. Finally, it can be seen that the cases of checking well-known DLL module names are greatly reduced. As for the method of checking well-known DLL modules, many module names can be detected in sandboxes that have been released in the past, so it can be seen that the cases of detecting old sandboxes even in malicious code decrease.

In the case of history, it can be seen that there are many uses only in certain techniques, such as user's temporary files, the number of connected USB devices, the number of installed browsers, and the number of installed programs. It was analyzed that techniques such as system uptime check, last logged in time check, and last accessed file check was not actually used in malicious code in which detection performance was verified in previous studies.

Table 10: Anti-VM Detection Result per Each Technique.

Category	VMware			VirtualBox			QEMU			Parallels			Xen			Virtual PC			Hyper-V			Bochs		
	2018	2019	2020	2018	2019	2020	2018	2019	2020	2018	2019	2020	2018	2019	2020	2018	2019	2020	2018	2019	2020	2018	2019	2020
I/O Port	269	193	440	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0
CPU Vendor	5	14	105	1	4	7	92	4	19	0	5	4	7	6	46	4	5	42	0	0	0	0	0	0
Process	23	99	94	27	55	131	1	2	14	3	12	40	2	13	9	3	15	43	0	0	0	0	0	0
Service	0	5	11	38	85	126	0	0	15	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0
Driver	16	74	34	45	92	216	0	0	0	2	0	1	0	0	0	1	0	0	0	0	0	0	0	0
File	1	6	11	0	1	12	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0
Module	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	5	5	47	0	0	0	0	0	0
Registry	1	6	30	61	43	56	0	1	23	0	1	0	1	4	4	0	0	0	0	1	3	0	0	2
Firmware	173	343	432	94	163	486	0	0	0	0	0	2	0	0	0	0	0	0	0	0	0	0	0	0
Hardware	391	569	1,046	1,702	1,608	3,317	1,086	1,150	1,913	14	18	29	37	32	94	4	16	22	1	2	27	0	2	2
Total	879	1,309	2,203	1,969	2,061	4,371	1,179	1,157	1,984	19	36	76	47	55	153	17	41	154	1	3	30	0	2	4

Table 11: Anti-Sandbox Detection Result per Each Technique.

Category		2018	2019	2020
Hardware	CPU Core	15,003	10,673	20,829
Hardware	RAM Size	2,639	5,966	10,974
Hardware	Disk Size	53	40	82
Hardware	MAC Address	0	0	0
Hardware	WMI Query	4	13	9
Environment	DLL	10,718	9,519	4,890
Environment	Product ID	4	3	2
Environment	Process Name	0	0	0
Environment	Display Resolution	60,520	46,475	64,564
Environment	File Name	0	1	0
Environment	Host Name	13,212	15,256	20,342
User-Interactive	Mouse Movement	24,541	23,670	31,218
User-Interactive	Mouse Click	5,354	3,204	5,839
History	Explorer	7	13	25
History	Internet Explorer	4	64	66
History	User's Temp File	24,438	11,354	18,513
History	Admin's Temp File	24	29	39
History	System Uptime	0	0	7
History	Windows Update	0	0	0
History	USB Device	66,372	51,432	69,383
History	Registry Size	0	0	0
History	Installed Browser	9,706	8,075	3,025
History	Last Login Time	0	0	0
History	Last File Access	3	2	16
History	Installed Program	11,016	9,498	4,901

4 Discussion

4.1 Limitation

Firstly, in this work, we statically analyzed real-world malware, and thus, we could not analyze the packed malware (roughly 30% of our dataset). Also, among the other 70% of the dataset, we could not analyze encrypted or encoded malware. We leave this limitation as future work where we will extend our framework to deal with such cases. Secondly, among the sandbox detection techniques investigated in previous studies and reports, there were cases difficult to define as anti-analysis signatures and patterns. For example, we can check the number of CPU cores by using the GetSystemInfo() API. However, we cannot conclude that this API checks the number of CPU cores because the API uses the parameters of the structure variables. Lastly, in this work, we could not verify out analysis results and find false positives from our dataset. However, we believe our findings clearly demonstrate the need of studies on automated analysis framework for anti-analysis techniques. Based on our findings, we expand this study and conduct future work as in subsection 4.2.

4.2 Future Work

The anti-analysis techniques exploit distinctive characteristics of execution environments in analysis systems. Hence, the advances in hardware, patches from virtual machine vendors, and major updates to sandbox environments can make existing anti-analysis techniques unavailable [8]. As such, anti-analysis techniques should be changing and evolving according to the changes of analysis systems. Consequently, to detect and thwart anti-analysis techniques, we have to continuously collect and find emerging anti-analysis techniques used in malware. In our future work, we will develop an automated framework that continuously collects malware to find anti-analysis techniques for evading detection. In addition, we plan to use a dynamic analysis method to find anti-analysis techniques in packed malware. Furthermore, we will find false positives to discover possible mid-detection by conducting malware's behavior comparison experiments.

5 Conclusion

In this paper, we conducted an analysis study of evasive malware. Especially, we investigated on the anti-analysis methods. To this end, we first surveyed known detection techniques and effective analysis prevention techniques proposed in previous studies. We, then, performed the static analysis on real-world Windows malware collected from 2018 to 2020 to find the trend of anti-analysis techniques. As results, we found that malware that detects virtual machines accounted for about 2.92%, and about 55% of malware that recognize sandboxes. In our future work, we will expand our system and conduct extensive study on the anti-analysis techniques used in malware to support the anti-malware ecosystem.

6 Acknowledgments

This work was supported by Institute of Information & communications Technology Planning & Evaluation (IITP) grant funded by the Korea government(MSIT) (No.2017-0-00168, Automatic Deep Malware Analysis Technology for Cyber Threat Intelligence)

References

[1] al-khaser. `https://github.com/LordNoteworthy/al-khaser`, last viewed July 2021.

[2] Paranoid fish. `https://github.com/a0rtega/pafish`, last viewed July 2021.

[3] The kutaki malware bypasses gateways to steal users' credentials. `https://cofense.com/kutaki-malware-bypasses-gateways-steal-users-credentials/`, last viewed July 2021, 2019.

[4] Malware behavior catalog. `https://github.com/MBCProject/mbc-markdown`, last viewed July 2021, 2019.

[5] Analyzing azorult's anti-analysis tricks with joe sandbox hypervisor. `https://www.joesecurity.org/blog/9048980422564630717`, last viewed July 2021, 2020.

[6] L. Abrams. Malware adds online sandbox detection to evade analysis. `https://www.bleepingcomputer.com/news/security/malware-adds-online-sandbox-detection-to-evade-analysis/`, last viewed July 2021, 2020.

[7] A. Afianian, S. Niksefat, B. Sadeghiyan, and D. Baptiste. Malware dynamic analysis evasion techniques: A survey. *ACM Computing Surveys (CSUR)*, 52(6):1–28, 2019.

[8] A. Algawi, M. Kiperberg, R. Leon, A. Resh, and N. Zaidenberg. Creating modern blue pills and red pills. In *European Conference on Cyber Warfare and Security*, pages 6–14. Academic Conferences International Limited, 2019.

[9] Ö. Aslan and R. Samet. Investigation of possibilities to detect malware using existing tools. In *2017 IEEE/ACS 14th International Conference on Computer Systems and Applications (AICCSA)*, pages 1277–1284. IEEE, 2017.

[10] Y. Assor. Anti-vm and anti-sandbox explained. `https://www.cyberbit.com/blog/endpoint-security/anti-vm-and-anti-sandbox-explained/`, last viewed July 2021, 2016.

[11] BENKOW. Win32/atrax.a. `https://thisissecurity.stormshield.com/2014/08/20/win32atrax-a/`, last viewed July 2021, 2014.

[12] Bochs. The cross platform ia-32 emulator. `https://bochs.sourceforge.io`, last viewed July 2021, 2001–2021.

[13] R. R. Branco, G. N. Barbosa, and P. D. Neto. Scientific but not academical overview of malware anti-debugging, anti-disassembly and anti-vm technologies. *Black Hat*, 1:1–27, 2012.

[14] P. Chen, C. Huygens, L. Desmet, and W. Joosen. Advanced or not? a comparative study of the use of anti-debugging and anti-vm techniques in generic and targeted malware. In *IFIP International Conference on ICT Systems Security and Privacy Protection*, pages 323–336. Springer, 2016.

[15] CWSandbox. Understanding the sandbox concept of malware identification. `http://cwsandbox.org`, last viewed July 2021, 2013–2021.

[16] A. Dahan. New betabot campaign under the microscope. `https://www.cybereason.com/blog/betabot-banking-trojan-neurevt`, last viewed July 2021, 2018.

[17] DECAF. Decaf is a binary analysis platform based on qemu. `https://github.com/decaf-project/DECAF`, last viewed July 2021, 2014–2021.

[18] A. Dinaburg, P. Royal, M. Sharif, and W. Lee. Ether: malware analysis via hardware virtualization extensions. In *Proceedings of the 15th ACM conference on Computer and communications security*, pages 51–62, 2008.

[19] P. Feng, J. Sun, S. Liu, and K. Sun. Uber: Combating sandbox evasion via user behavior emulators. In *ICICS*, pages 34–50, 2019.

[20] R. Hat. What is a hypervisor? `https://www.redhat.com/en/topics/virtualization/what-is-a-hypervisor`, last viewed July 2021, 1993–2021.

[21] D. Inoue, K. Yoshioka, M. Eto, Y. Hoshizawa, and K. Nakao. Automated malware analysis system and its sandbox for revealing malware's internal and external activities. *IEICE transactions on information and systems*, 92(5):945–954, 2009.

[22] A. Jadhav, D. Vidyarthi, and M. Hemavathy. Evolution of evasive malwares: A survey. In *2016 International Conference on Computational Techniques in Information and Communication Technologies (ICCTICT)*, pages 641–646. IEEE, 2016.

[23] S. Jamalpur, Y. S. Navya, P. Raja, G. Tagore, and G. R. K. Rao. Dynamic malware analysis using cuckoo sandbox. In *2018 Second international conference on inventive communication and computational technologies (ICICCT)*, pages 1056–1060. IEEE, 2018.

[24] JoeSecurity. Parallels desktop virtual machine. `https://www.parallels.com/`, last viewed July 2021, 1999–2021.

[25] JoeSecurity. Why joe sandbox? `https://www.joesecurity.org/why-joe-sandbox`, last viewed July 2021, 2003–2021.

[26] D. Kirat, G. Vigna, and C. Kruegel. Barebox: efficient malware analysis on bare-metal. In *Proceedings of the 27th Annual Computer Security Applications Conference*, pages 403–412, 2011.

[27] D. Kirat, G. Vigna, and C. Kruegel. Barecloud: Bare-metal analysis-based evasive malware detection. In *23rd {USENIX} Security Symposium ({USENIX} Security 14)*, pages 287–301, 2014.

[28] KVM. Kernel virtual machine. `https://www.linux-kvm.org/page/Main_Page`, last viewed July 2021, 1996–2021.

[29] C. Lim et al. Mal-eve: Static detection model for evasive malware. In *2015 10th International Conference on Communications and Networking in China (ChinaCom)*, pages 283–288. IEEE, 2015.

[30] M. Lindorfer, C. Kolbitsch, and P. M. Comparetti. Detecting environment-sensitive malware. In *International Workshop on Recent Advances in Intrusion Detection*, pages 338–357. Springer, 2011.

[31] Microsoft. Hyper-v on windows 10. `https://docs.microsoft.com/en-us/virtualization/`

hyper-v-on-windows/, last viewed July 2021, 2008–2021.

[32] A. Mills and P. Legg. Investigating anti-evasion malware triggers using automated sandbox reconfiguration techniques. *Journal of Cybersecurity and Privacy*, 1(1):19–39, 2021.

[33] N. Miramirkhani, M. P. Appini, N. Nikiforakis, and M. Polychronakis. Spotless sandboxes: Evading malware analysis systems using wear-and-tear artifacts. In *2017 IEEE Symposium on Security and Privacy (SP)*, pages 1009–1024. IEEE, 2017.

[34] A. Mushtaq. The dead giveaways of vm-aware malware. https://www.fireeye.com/blog/threat-research/2011/01/the-dead-giveaways-of-vm-aware-malware.html, last viewed July 2021, 2011.

[35] Oracle. Oracle vm virtualbox. https://www.virtualbox.org, last viewed July 2021, 2007–2021.

[36] PANDA. Platform for architecture-neutral dynamic analysis. https://github.com/panda-re/panda, last viewed July 2021, 2013–2021.

[37] A. Pektaş and T. Acarman. A dynamic malware analyzer against virtual machine aware malicious software. *Security and Communication Networks*, 7(12):2245–2257, 2014.

[38] T. X. Project. Hyper-v on windows 10. https://docs.microsoft.com/en-us/virtualization/hyper-v-on-windows/, last viewed July 2021, 2003–2021.

[39] PyREBox. Python scriptable reverse engineering sandbox, a virtual machine instrumentation and inspection framework based on qemu. https://github.com/Cisco-Talos/pyrebox, last viewed July 2021, 2017–2021.

[40] QEMU. Qemu the fast! processor emulator. https://www.qemu.org, last viewed July 2021, 2009–2021.

[41] S. A. Roseline and S. Geetha. A comprehensive survey of tools and techniques mitigating computer and mobile malware attacks. *Computers & Electrical Engineering*, 92:107143, 2021.

[42] S2E. S2e: A platform for in-vivo analysis of software systems. https://s2e.systems/, last viewed July 2021, 2010–2021.

[43] A. S. Sai Omkar Vashisht. Turing test in reverse: New sandbox-evasion techniques seek human interaction. https://www.fireeye.com/blog/threat-research/2014/06/turing-test-in-reverse-new-sandbox-evasion-techniques-seek-human-interaction.html, last viewed July 2021, 2014.

[44] G. Severi, T. Leek, and B. Dolan-Gavitt. M alrec: compact full-trace malware recording for retrospective deep analysis. In *International Conference on Detection of Intrusions and Malware, and Vulnerability Assessment*, pages 3–23. Springer, 2018.

[45] D. Shoemake. Dynamic behavioral analysis of malicious software with norman sandbox. 2010.

[46] B. Taubmann and B. Kolosnjaji. Architecture for resource-aware vmi-based cloud malware analysis. In *Proceedings of the 4th Workshop on Security in Highly Connected IT Systems*, pages 43–48, 2017.

[47] C. S. Thomas Roccia. Evolution of malware sandbox evasion tactics - a retrospective study. https://www.mcafee.com/blogs/other-blogs/mcafee-labs/evolution-of-malware-sandbox-evasion-tactics-a-retrospective-study/, last viewed July 2021, 2019.

[48] VirusShare. Virusshare.com. https://virusshare.com/, last viewed July 2021.

[49] VMware. Vmware. https://www.vmware.com, last viewed July 2021, 1998–2021.

[50] VMware. Virtual machine. https://www.vmware.com/topics/glossary/content/virtual-machine.html, last viewed July 2021, 1999–2021.

[51] J. Wei, L. K. Yan, and M. A. Hakim. Mose: Live migration based on-the-fly software emulation. In *Proceedings of the 31st Annual Computer Security Applications Conference*, pages 221–230, 2015.

[52] A. Yokoyama, K. Ishii, R. Tanabe, Y. Papa, K. Yoshioka, T. Matsumoto, T. Kasama, D. Inoue, M. Brengel, M. Backes, et al. Sandprint: Fingerprinting malware sandboxes to provide intelligence for sandbox evasion. In *International Symposium on Research in Attacks, Intrusions, and Defenses*, pages 165–187. Springer, 2016.

[53] K. Yoshioka, D. Inoue, M. Eto, Y. Hoshizawa, H. Nogawa, and K. Nakao. Malware sandbox analysis for secure observation of vulnerability exploitation. *IEICE transactions on information and systems*, 92(5):955–

966, 2009.