IEEE Access
Multidisciplinary | Rapid Review | Open Access Journal

Received 26 May 2022, accepted 22 June 2022, date of publication 14 July 2022, date of current version 22 July 2022.

Digital Object Identifier 10.1109/ACCESS.2022.3190978

RESEARCH ARTICLE

Large-Scale Analysis on Anti-Analysis Techniques in Real-World Malware

MINHO KIM, HAEHYUN CHO, AND JEONG HYUN YI, (Member, IEEE)
Graduate School of Software, Soongsil University, Seoul 06978, South Korea

Corresponding author: Haehyun Cho (haehyun@ssu.ac.kr)

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

ABSTRACT To dynamically identify malicious behaviors of millions of Windows malware, anti-virus vendors have widely been using sandbox-based analyzers. However, the sandbox-based analysis has a critical limitation that anti-analysis techniques (i.e., Anti-sandbox and Anti-VM techniques) can easily detect analyzers and evade from being analyzed. In this work, we study on anti-analysis techniques used in real-world malware. First off, to measure how many Windows malware exhibits anti-analysis techniques, we collect anti-analysis techniques used in malware. We, then, design and implement an automated system, named EvDetector, that detects malware which employ anti-analysis techniques. EvDetector finds if malware uses an anti-analysis technique and monitors whether the malware changes its execution paths based on the result of the anti-analysis technique. By using EvDetector, we analyzed 763,985 real-world malware that emerged from 2017 to 2020. Our evaluation results show that 16.21% of malware use anti-analysis techniques on average. Also, we check the effectiveness of the analysis result by comparing EvDetector and static analysis. EvDetector analyzes up to 49.88% of malware detected by static analysis did not use anti-analysis techniques. In addition, we analyze that only up to 3.75% of the packed malware used anti-analysis techniques. Finally, we analyze the evasive malware trend through familial analysis and behavioral analysis. Our work implies that the research community needs to put more effort on defeating such anti-analysis techniques to automatically analyze emerging malware and respond with them.

INDEX TERMS Anti-analysis, anti-VM, anti-sandbox, dynamic analysis.

I. INTRODUCTION

There is an increasing volume of malware reported to anti-virus every day [30], [40]. According to FireEye's M-Trends report [19] in 2020, they analyze 1.1 million malware samples every day on average, and they show that the most of malware (roughly 95%) they analyzed runs on the Windows OS. To deal with such the large number of malware, anti-virus vendors have been using automated sandbox-based analysis techniques [9], [21], [22], [47]. The sandbox-based analyzers execute millions of malware to analyze malware's behavior every day. Such sandbox-based approaches become popular, malware writers started using evasion techniques to avoid being analyze [7]. Malware that uses anti-analysis techniques forcibly terminates execution on sandboxed execution environments or intentionally does not perform malicious behaviors [27]. Those malware can be alive in-the-wild longer than the others that do not use anti-analysis techniques by delaying time to be detected [38]. The emergence of such evasive malware has brought strong technical challenges to automatically analyze them based on their malicious behaviors.

To handle the large number of malware that use anti-analysis techniques, the security researchers have proposed various detection approaches and have reported promising evaluation results [14], [15], [39]. In general, they tried to find signatures related to anti-analysis techniques in malware that do not perform malicious behaviors on the sandbox-based

The associate editor coordinating the review of this manuscript and approving it for publication was Claudia Raibulet.

analysis environment. However, a lot of Windows APIs related to anti-analysis techniques are generally used in benign applications as well. For example, a program can check system information and display the environmental data to a user. Because the API used by such a program uses the same API as the anti-analysis techniques, the previous proposed approaches may appear to detect malware when they meet such benign applications. Therefore, the approaches for detecting anti-analysis techniques through finding signatures in malware inherently yield high false-positives.

To address this problem, techniques for analyzing the behavioral similarity have been proposed [25], [26]. They compare execution results of a given malware on several sandboxes. In contrast to the previous approaches, they analyzed the system calls and network packets to compare the behavioral similarity of malware processes. This line of work compares the behavior similarity of malware, but using those approaches in a small number of sandboxes can yield low accuracy. As an another line of work, recently, Lee *et al.* [28] proposed an approach using a bypassing module to analyze malware that use anti-analysis techniques such as anti-VM.

In this paper, we propose an approach for detecting evasive malware based on dynamic analysis results. By using our approach, named EvDetector, we automatically check whether the signatures used in the prior work are used and analyze whether the signatures affect the execution result of the malware. Our work targets Windows malware that has not been analyzed yet. Compared to the prior works, we performed the largest analysis to the best of our knowledge—we analyze the trend from 2017 to 2020 of in-the-wild malware. Our major contributions are, thus, as follows.

- We collect anti-analysis techniques used in real-world malware by analyzing the prior works.
- We propose an automated tool, EvDetector, to detect malware that use anti-analysis techniques. EvDetector dynamically inspects malware to check whether the anti-analysis techniques affect the execution result of the malware.
- We conduct a large-scale evaluation using the real-world malware dataset collected in 2017–2020 and show that how much malware are using anti-analysis techniques in the wild. We also conduct a comparative study, and show trends of evasive malware.

The rest of the paper is organized as follows: section II describes related work and why the analysis of evasive malware is complex. section III describes the anti-analysis techniques used by malware. section IV describes an overview of our work. section V describes the design of EvDetector and its implementation. section VI describes our evaluation results. section VII discusses the limitations faced during the research. section VIII concludes the work.
To foster future research, we have released the source code of EvDetector and signatures the anti-analysis techniques that we investigated for this study.[1]

[1]https://github.com/ssu-csec/EvDetector

II. BACKGROUND

Malware writers have been attempting to create malware that can avoid being analyzed from anti-virus vendors' detection systems. Naturally, the number of malware which equip anti-analysis techniques is increasing in the wild. To handle such evasive malware, previous studies proposed various approaches. In this section, we summarize the previous work in subsection II-A and introduce challenges to deal with evasive malware subsection II-B.

A. RELATED WORK

Kirat *et al.* [25] proposed BareBox, a bare-metal based system to analyze malware that detects VMware [48] and QEMU [42] virtual environments. To evaluate the malware that detects VMware and QEMU, BareBox used a malware dataset from Anubis [11], [12]. By classifying the malware family, BareBox analyzed malware's system call and compared whether malware created new processes or network packets in VMware, QEMU, and BareBox, respectively. BareBox observed that all malware was more active only in BareBox, proving the stealthiness was outstanding than VMware and QEMU. However, BareBox cannot identify anti-analysis techniques used in malware.

Branco *et al.* [14] categorized anti-analysis techniques into four categories: anti-debug, anti-disassembly, obfuscation, and anti-VM. They employed a static analysis to check whether specific APIs or instructions used in anti-analysis techniques exist. For finding if malware employs anti-VM techniques, they checked whether malware could have different execution results between a host machine and a guest VM. They measured that 81.4% of 4,030,945 malware samples used anti-VM techniques. They classified it as malware that detects virtual machines if it contains instructions such as SIDT (Store Interrupt Descriptor Table), SGDT (Store Global Descriptor Table), and SLDT (Store Local Descriptor Table). These instructions obtained the values of the CPU's descriptor table and returned different values from the host and guest machines. Existing detection techniques used these characteristics to detect virtual machines. On single-core processor systems, it is straightforward to detect guest machines using the IDTR because there is only one IDTR. However, on muticore systems, there are multiple IDTRs for each core and the host machine even may return a different value when the SIDT instruction executes. Since IDTRs exist for each processor, even the host may return a different value each time. The detection approach leveraging SLDT and SGDT is called as No Pill that exploits differences in execution results of the SLDT and SGDT instructions between on the host and guest machines. However, this approach can be easily bypassed by disabling the acceleration option of guest machines. These limitations are unsuitable for detecting virtual machines [8], [31].

To analyze the malware that detect the sandbox, BareCloud [26], a system that compares the behavior similarity of malware in various environments, used bare metal,

virtualization, emulation, and hypervisor-based environments. Specifically, it simultaneously collected execution logs by running the same malware on Ether [18], Anubis [11], [12], Cuckoo Sandbox [20], and BareCloud. They collected the execution logs of 234 evasive malware and 119 non-evasive malware to train the behavioral similarity comparison model. To evaluate the performance of the behavioral similarity comparison model, they tested 110,005 malware datasets. About 5,835 malware showed the results to detect the analysis environment. However, their system requires the overhead of running in four environments to collect and compare logs of one malware. For this reason, it can be a disadvantage for large-scale malware analysis.

Ping *et al.* [15] conducted a study on whether APT malware uses anti-debug and anti-VM techniques more than other malware families. They used static analysis to investigate whether the signature of instructions and APIs used for anti-analysis techniques exist in malware. For example, they examined strings related to virtual machines and sandboxes, such as ''VMware'' and ''sbiedll.dll'' in malware. They conducted a comparative analysis on the APT malware datasets (2009–2014: 1,037) and six general malware families datasets (2009–2014: 16,000). The evaluation results showed the anti-analysis detection rate and the anti-virus detection rate in each category. Their approach detected instructions with different execution results in hosts and guests and commonly used API such as ''Sleep''. However, there are limitations in detecting APIs that can reduce detection accuracy or instructions that are not valid in modern environments. [8].

Yokoyama *et al.* [50] analyzed collected features from 76 actual sandbox-based malware analysis services and represented refined features that distinguish between the sandbox services and user systems. They categorized the detection techniques into hardware, history, and execution. To prove their performance, they compared SandPrint with PAFISH [37]. PAFISH is an open-source tool that implements sandbox detection techniques. The evaluation results showed that SandPrint's single feature accuracy and combined accuracy are excellent than PAFISH. However, although SandPrint defined effective techniques for real-world sandboxes detection, they did not evaluate how much malware applied their feature to detect sandboxes.

Miramirkhani *et al.* [32] conducted research to analyze the characteristics of the sandbox compared to the real user system. To classify the Wear-and-Tears, they collected features of 270 real user systems and 16 malware analysis sandboxes. They showed the system's aging, which can distinguish the sandbox from the user system in the system, disk, network, registry, and browser categories. Since it contains only the degree of aging, which is data that does not violate privacy, if the sandbox generates a random fingerprint when it restores the snapshot, the disadvantage is that it is difficult to detect using a pre-trained model.

To evaluate malware that avoids the Cuckoo Sandbox's analysis, Oyama *et al.* [39] analyzed FFRI datasets (2016: 8,243) that recorded malware behavior in the Cuckoo sandbox. Due to the FFRI dataset did not record CPU instruction, they measured malware based on dynamically analyzed API Call sequences and statically extracted signatures. The evaluation results showed that 10.4% of malware use the anti-analysis technique. However, most signatures statically extracted from malware except for API calls. Although the datasets analyzed in the sandbox, a dynamic analysis environment, are used, their detection approach is not significantly different from the prior static analysis.

Choi *et al.* [16] proposed HybridEmu, a DBI framework for dynamically analyzing malware. They compared various DBI frameworks to prove resistance to 29 common anti-debug techniques and anti-debug techniques provided by 17 commercial protectors. HybridEmu prove to have resistance to anti-debug techniques. In particular, they showed that Intel Pin [29] had high resistance to anti-debug techniques than other DBI frameworks. From this result, we implement our approach based on Intel Pin.

Lee *et al.* [28] analyzed the anti-DBI and anti-VM techniques provided by commercial protectors and showed bypassing algorithms using Intel Pin. The commercial protectors are Themida [36], Enigma [45], VMProtect [2], Obsidium [35], and ACProtect [49]. To bypass each protection technique, they analyzed the protector's techniques in detail. To prove that it is possible to analyze malware applied with commercial protectors, they regarded the Juliet Test Suite Code [34] as malware. Then, they showed that their analysis tool successfully bypassed packed malware. However, detection and bypassing are possible only with the anti-analysis techniques provided by the analyzed version of the commercial protector. For example, it cannot counteract the protector's technique provided by the unanalyzed version or another technique used by the malware itself.

Yuhei *et al.* [23], [24] proposed a code tainting techniques-based analyzer, API Chaser, to identify the execution of monitored instructions. API Chaser gave different taint tags to the API, benign, and malware samples. First, they identified malware that called the API instructions through a taint tag. Then, they tagged the API execution result to respond to generated data during execution. Tracking the API execution result is roughly like our approaches. Their goal was to detect malware that used code injection and stolen code techniques, and they focused on monitoring the propagation of taint tags. They identified hook and target evasion malware and included it in the monitoring target. However, there is a limitation in not being able to respond to an API call of the Return Oriented Programming.

B. CHALLENGES TOWARDS ANALYZING EVASIVE MALWARE

Previous studies used various analysis methods to analyze malware that uses anti-analysis techniques. Table 1 summarizes the prior works investigated in subsection II-A. Although many studies have been conducted, many

TABLE 1. The summary of previous work against anti-analysis techniques.

Research	Year	Analysis	Detection	Evaluation Method
Kirat et al. [25]	2011	Dynamic	Analyze system calls to see if malware generates new processes or network packets.	Tested 200 evasive malware, all detected more activity in BareBox.
Branco et al. [14]	2012	Static	Check whether malware could have different execution results between a host machine and a guest VM.	Tested 4,030,945 malware, 81.4% of malware detected to use anti-VM techniques.
Kirat et al. [26]	2014	Dynamic	Analyze the behavioral similarity of malware in 4 different sandbox environments.	Tested 110,005 malware, 5.30% of malware detected to use anti-sandbox techniques.
Ping et al. [15]	2016	Static	Check whether malware could have different execution result betwwen a host machine and a guest VM. Check the APIs and strings used to virtual machines and sandboxes.	Tested 1,037 APT malware, 84.2% of malware detected to use anti-VM techniques. Tested 16,000 6 families malware, 68.6% of malware detected to use anti-VM techniques.
Yokoyama et al. [50]	2016	Fingerprint	Collect characteristics from real-world sandbox services.	Classified the sandbox detection techniques into 5 categories and 25 refined features.
Miramirkhani et al. [32]	2017	Fingerprint	Collect characteristics from real-world sandbox services and real-user system.	Classified the sandbox detection techniques into 5 categories and 44 refined features.
Oyama et al. [39]	2018	Hybrid	Analyze the malware analysis log of Cuckoo Sandbox.	Tested 8,243 malware, 10.4% of malware detected to use anti-analysis techniques.
Lee et al. [28]	2021	Dynamic	Analyze the anti-DBI and anti-VM techniques provided by the commercial protector and implement the bypass module.	Tested packed sample, bypass all samples to confirm normal operation.
EvDetector	2022	Dynamic	Conditional branches using variables containing execution results of APIs used for anti-analysis techniques.	Tested 763,935 of Windows malware, 16.21% of malware detected to use anti-analysis techniques.

challenges still make our analysis difficult. For the future research, we summarize the challenges.

The first common limitation that we discovered is the analysis scope: previous research work examined malware with only well-known anti-analysis techniques. It is impossible to detect unknown anti-analysis techniques, and thus, the limitation of such design makes analysis of new anti-analysis techniques infeasible. The second common limitation is the complicated design to analyze large-scale malware datasets: previous research work simultaneously compared logs generated in different conditions to validate the analysis. Building various environments to analyze and verify malware incurs overhead in terms of time and hardware.

The next limitation is in static analysis-based approaches. We have considered anti-analysis signatures such as instructions, APIs, and strings exist in the code or data sections of evasive malware. Therefore, detection is not possible if malware applies protection techniques such as packing and obfuscation. Windows APIs perform various roles depending on arguments passed when they execute. For example, APIs that are well-known for anti-analysis techniques may or may not be used in anti-analysis techniques, depending on the argument. Also, with static analysis-based approaches, it is impossible to accurately identify each API's arguments that may use in the anti-analysis techniques. If an analysis tool simply detected a malicious application that uses an anti-analysis technique based on APIs used in the malware, the result could be a false positive.

The last limitation is in dynamic analysis-based approaches. Similar to the previous limitation, when calling APIs that are likely to be used as anti-analysis technique, analyzing only the API and API's arguments can lead to false positives. We believe that, after calling the APIs, it is essential to investigate whether the results affect the execution result of the process. In analyzing the malware through a dynamic analysis tool, the malware can detect the analysis tool and change its behavior. In this work, we aim to resolve the last challenge by dynamically tracing return values of APIs widely used in anti-analysis techniques.

III. ANTI-ANALYSIS TECHNIQUES

In this section, we present anti-analysis techniques widely used in real-world malware.

MITRE ATT&CK [33] analyzes, categorizes, and discloses technologies used in real-world attacks. To prevent being analyzed from analysis frameworks, malware can use anti-analysis techniques. Anti-analysis technology is divided mainly into an approach that protects the binary itself and a method that manipulates the execution result according to the execution environment. Protection techniques such as packing, obfuscation, and encryption make it challenging to analyze malware binary. Existing analysis tools cannot usually extract malware's code or data at the binary level. Malware using such techniques should go through specific processes for unpacking and decrypting some code and data by itself to perform malicious actions. Previous studies have detected such moments when protected codes or data are unprotected. Other techniques are to stop execution in the middle of the performance if the malware writers don't want it. Antivirus vendors do automated analysis in a sandbox environment to counter a lot of malware. Malware writers want to prevent malware from being analyzed in analysis environments. For example, one of the general evasion techniques is virtualization and sandbox avoidance techniques (T1497, System Checks, User Activity-Based Checks, and Time Based Evolution). To perform malicious behaviors or detect analysis and forcibly terminate execution, evasive malware detects features that only appear in the sandbox-based analysis environments. In general, to avoid sandbox-based analysis, some techniques test the high-speed mouse

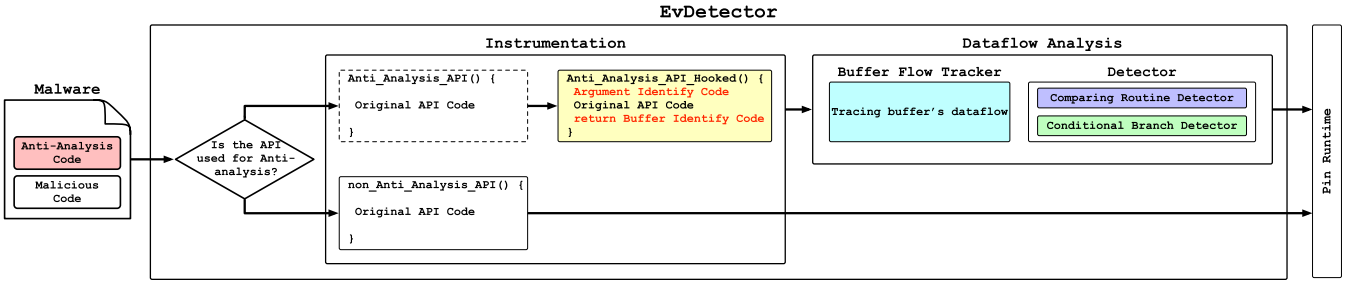

FIGURE 1. Overview of EvDetector: show that EvDetector analyzes the malicious application that use the anti-analysis techniques; EvDetector identifies the APIs used for the anti-analysis techniques and inserts a hooking code that pinpoints the dynamically-allocated buffer. After that, EvDetector traces the buffer's usage flow and performs the process of finding a conditional branch to see if the buffer comparison result affects the execution flow of the malicious application.

movements, check the number of clicks on certain icons, and wait for execution for minutes to days [43], [46]. These techniques are difficult to analyze and verify in the existing analysis environments. Therefore, in this work, we aim to analyze whether an malicious application uses an evasion technique that can deterministically detect sandboxes in a short time. To be specific, we investigated techniques such as detection using environmental information of a specific sandbox and virtualization solution's registry key, Driver and BIOS. The following subsections introduce each category of information that can be used in anti-analysis techniques.

A. HARDWARE AND ENVIRONMENTAL INFORMATION

To detect sandbox-based analysis, malware can retrieve the hardware and environmental information. Typically, anti-virus vendors need to employ a lot of sanboxes to deal with the millions of malware, and thus, each sandbox has limited hardware resources. For example, malicious applications find the number of CPU cores and the size of ram on a system [5], [13]. Then, they can compare the values with the average hardware specifications of real-world sandbox services for analyzing malware. If the retrieved value is less than the average value, malware recognize itself as running on a sandbox-based environment.

B. DRIVERS AND MODULE DETECTION

Malware can check the existence of drivers and system files provided by virtualization solutions such as VMware and VirtualBox [17]. For example, if there are drivers such as ''vmmouse.sys'' and ''VBoxMouse.sys'' on a system, malicious applications recognize the system as a virtual machine. These malware also can search modules or files found only in the sandbox environment. For example, it is a well-known fact that the Cuckoo sandbox uses files such as ''agent.py'' to communicate with the host and the guest environment. Malware, thus, can detect whether it runs on a sandbox by retrieving such files used in specific sandbox environments.

C. REGISTRY INFORMATION

Malicious applications can check the registry keys that exist only in a sandbox environment [10]. For example, the ''HARDWARE\ACPI\DSDT\VBOX__'' key exists only in a guest machine created by VirtualBox. Whey malware found such registry keys, they can consider that they are running on a virtualized guest machine. For example, the ''SystemBiosVersion'' key of ''HARDWARE\Description\System\'' stores a string value related to BIOS of a system. These registry keys store the string values related to the virtualization solutions, such as ''VMware,'' ''VirtualBox,'' and ''QEMU.'' In general Windows environments, these registry key values are totally different. Therefore, malware can detect a sandbox by reading such registry key values.

D. ANTI-DBI TECHNIQUES

A Dynamic Binary Instrumentation (DBI) framework such as Intel Pin enables the creation of dynamic program analysis tools by performing instrumentations at run time on the compiled binary files. Because the DBI framework is also widely-used to analyze malware, there are several techniques used in malicious applications to avoid being analyzed from the DBI framework and debuggers [41]. However, the analysis tools based on the DBI framework can have implement an automated bypassing module that recognizes and instrument such anti-DBI techniques to bypass them [28].

In this work, we focus on anti-DBI techniques, especially for detecting Intel Pin [41] because our approach is based on a dynamic analysis by using Intel Pin. In general, anti-DBI techniques (for detecting Intel Pin) work based on the NtQueryInformationProcess (ProcessDebugFlags), Single-Step exception, and PAGEGUARD exception [28], [41]. Among them, the traditional NtGlobalFlag detection technique uses a feature that the value of NtGlobalFlag, one of the variables in the PEB (Process Environment Block) structure, is always set to 0×70 when a debugger is attached to the process. This feature can be detected only when a Pin-based tool analyzes a 32-bit program on the 64-bit Windows OS. In addition, when the 64-bit Windows OS executes a 32-bit program, the OS creates a 64-bit PEB structure and a 32-bit PEB structure in the process memory for the compatibility. Therefore, when a pin-based tool analyzes a 32-bit program, the NtGlobalFlag values of 64-bit PEB and 32-bit PEB will be set to 0×70 and 0×0, respectively. As a result, the anti-DBI

technique can detect by examining the NtGlobalFlag value of the 64-bit PEB structure.

IV. OVERVIEW

In this work, we aim to understand evasive techniques used by Windows malware and effectively identify how many malicious applications try to avoid anti-virus vendor's investigation. Based on our approach, we believe that we can step forward to automate bypassing anti-analysis techniques. To this end, we employ Intel Pin, a representative DBI framework, to dynamically inspect whether malware uses anti-analysis techniques. Among the functions provided by Intel Pin, we use the Routine Replace to hook the Windows APIs used for anti-analysis techniques. Specifically, we insert code that identifies a dynamically-allocated buffer in which the Windows APIs stored APIs' execution results. After a malicious application calls the Windows APIs, we inspect that whether or not the malware compares the value in the buffer, executing conditional branches that affect the execution flow of the malware. Only when EvDetector identified malware executes conditional branches with the return values of the Windows APIs, EvDetector confirmed it as an malicious application that uses anti-analysis techniques. Figure 1 shows the overview of our EvDetector to detect malware that use anti-analysis techniques.

```
1  void sandbox_detect(...) {
2      ...
3      if(!detectAnalysisEnvironment()) {
4          do_normal_behavior();
5      }
6      else {
7          do_malicious_behavior();
8      }
9      ...
10 }
```

Listing 1. Concept of anti-analysis: The conditional branch we are looking for means that occurs due to the execution of like detectanalysisenvironment(), which affects the control flow of the malware.

By using EvDetector, we evaluate our dataset that consists of real-world Windows malware appeared from 2017 to 2020 (2017: 349,256, 2018: 119,952, 2019: 120,942, 2020: 173,785). Our evaluation results show that it is possible to analyze recent real-world malware that evades sandbox-based analysis (2017: 21.89%, 2018: 13.72%, 2019: 8.25%, 2020: 12.06%). It is worth to note that the half of detected signatures (generally known to be used as anti-analysis techniques) do not affect the execution results of the malware in our evaluation (2017: 48.83%, 2018: 49.88%, 2019: 48.22%, 2020: 42.37%). Therefore, our evaluation results imply that simply finding such signatures could yield very high false-positives. In addition, we shows only a tiny of packed malware detects the analysis environment (2017: 0.73%, 2018: 1.66%, 2019: 1.60%, 2020: 3.75%). EvDetector has the following advantages, overcoming the limitation of previous approaches:

- It is possible to detect more detailed results than the prior analyses that find only the presence of the signatures of anti-analysis techniques.
- To detect evasive malware, it consumes less computational resources and less time than approaches such as BareCloud [26] because it does not need to run the malware in multiple environments at the same time.

A. THREAT MODEL

By using EvDetector, our primary concern is how many real-world malicious applications can evade anti-virus vendor's analysis. To analyze real-world malware trends, we study malware collected over a period from 2017 to 2020 and we set up the following EvDetector's threat model:

- Evasive malware will detect whether it is being analyzed by using the anti-analysis techniques in the initial execution process, that is, before performing malicious actions.
- Evasive malware will have a conditional branch in which the control flow of the malware varies depending on the result of the anti-analysis techniques.

An example of our threat model is shown in Listing 1.

In the following sections, we show how we implement EvDetector to analyze malware (subsection V-B), how we collected our dataset (subsection VI-A), we evaluate EvDetector with the dataset (subsection VI-B), and we discuss the limitations of EvDetector in section VII.

V. DESIGN

The design goal of EvDetector is to automatically detect evasive malware.

A. ANTI-ANALYSIS TECHNIQUES TARGETED

For the automated analysis of evasive malware, it is necessary to investigate the anti-analysis techniques that malware can operate. We have collected various documents, such as analysis reports from anti-virus vendors about anti-analysis techniques that malware can use.

We surveyed the anti-analysis techniques discussed in section III and analyzed anti-analysis techniques and classified them by category as in Table 2, which shows examples of sandbox-based analysis detection techniques. In total, we targeted 38 classified Windows APIs and 370 strings that can be used as arguments.

B. EvDetector

To accurately detect anti-analysis techniques, we do not only find usages of Windows APIs used in the anti-analysis techniques but also focus on the dynamically allocated buffer where the return values of the APIs are stored. We dynamically trace the dataflow and analyze whether the return value affects the control flow of the malware. Dynamic analysis is required to identify buffer addresses, and track usage flows accurately. To this end, we implemented EvDetector using the DBI framework. Specifically, we employ Intel Pin that

TABLE 2. Examples of classified anti-analysis techniques: show that representative anti-analysis techniques. Our target is system checks and user activity-based checks defined by MITRE ATT&CK, and we investigated and classified techniques for detecting sandboxes in a short time. We use a total of 38 windows APIs and 370 strings to investigate malware that uses anti-analysis techniques.

Category	Type	Description	API	Practical Case
System Checks	Hardware & Environment	Check Hardware Spec	IWbemClassObject::Get	"SELECT NumberOfCores FROM Win32_Processor"
System Checks	Hardware & Environment	Check Hardware Spec	IWbemClassObject::Get	"SELECT TotalPhysicalMemory FROM Win32_ComputerSystem"
System Checks	Hardware & Environment	Check Hardware Spec	IWbemClassObject::Get	"SELECT Size FROM Win32_LogicalDisk"
System Checks	Hardware & Environment	Check Hardware Spec	GlobalMemoryStatusEx, GetDiskFreeSpaceEx	
System Checks	Hardware & Environment	Check System Properties	IWbemClassObject::Get	"SELECT CurrentSize FROM Win32_Registry"
System Checks	Hardware & Environment	Check System Properties	GetSystemFirmwareTable, GetMonitorInfo	
System Checks	Driver & Module	Check if a specific file or directory exists	GetFileAttributes, FindFirstFile, GetModuleHandle	"VMware\VMware Tools*", "Oracle\VirtualBox Guest Additions*"
System Checks	Driver & Module	Check if a specific file or directory exists	GetFileAttributes, FindFirstFile, GetModuleHandle	"agent.py", "C:\cuckoo"
System Checks	Driver & Module	Check if a specific file or directory exists	GetFileAttributes, FindFirstFile, GetModuleHandle	"vmhgfs.dll", "sbiedll.dll", "snxhk.dll"
System Checks	Registry	Check if a specific registry key exists	RegOpenKey	"SYSTEM\CurrentControlSet\Services\vmhgfs"
System Checks	Registry	Check if a specific registry key exists	RegOpenKey	"HARDWARE\ACPI\DSDT\VBOX__"
System Checks	Registry	Check the value of a specific registry key	RegQueryValue	"HARDWARE\Description\System"
System Checks	Registry	Check the value of a specific registry key	RegQueryValue	"SYSTEM\ControlSet001\Control\SystemInformation"
User Activity-Based Checks	Registry	Check the value of a specific registry key	RegQueryValue	"SOFTWARE\Microsoft\Office\16.0\Word\Place MRU"
User Activity-Based Checks	Registry	Check the value of a specific registry key	RegQueryValue	"SOFTWARE\Microsoft\Office\16.0\Word\File MRU"
User Activity-Based Checks	Registry	Check the value of a specific registry key	RegQueryValue	"SOFTWARE\Microsoft\Office\16.0\Word\User MRU"
User Activity-Based Checks	Registry	Check the value of a specific registry key	RegQueryValue	"SOFTWARE\Microsoft\Windows\CurrentVersion\Explorer\RunMRU"
User Activity-Based Checks	Registry	Check the value of a specific registry key	RegQueryValue	"SYSTEM\ControlSet001\Enum\USBTOR"
User Activity-Based Checks	Hardware & Environment	Check System Properties	IWbemClassObject::Get	"SELECT CurrentSize FROM Win32_Registry"
User Activity-Based Checks	Driver & Module	Check if a specific file or directory exists	GetFileAttributes, FindFirstFile, GetModuleHandle	"\AppData\Roaming\Microsoft\Windows\Recent*"

provides a function called Routine Replace, so it can insert code by hooking the APIs provided by the OS. Based on the functionality of Routine Replace, we designed a way to inject code that identifies the location of a dynamically-allocated buffer that stores the APIs' execution results, targeting the Windows API used in the anti-analysis techniques.

1) INSERTING THE API-HOOKING CODE

Our targets are the Windows APIs used for anti-analysis techniques. The most of Windows APIs can perform more then two functions. Depending on the parameters of each API, it can or cannot be used as an anti-analysis technique. Traditional approaches for anti-analysis techniques were to simply examine the values of APIs and parameters. A lot of studies statically disassemble the malware binary. They inspected the disassembly code to see if the malware calls APIs (with parameters) used for anti-analysis techniques. On the other hand, many approaches analyzed malware dynamically through the API hooking techniques. The existing API hooking method works by patching process memory, injecting loaded DLL files, etc. They logged the hooked APIs, which parameters passed during execution. Such method can be detected by API monitoring or file patching detection techniques. Therefore, we implement a API hooking technique via Intel Pin. Intel Pin executes instrumented code by the DBI framework. Intel Pin, also, supports replacing pre-instrumented code which cap wrap the actual API function. Before executing each API that can be used to implement anti-analysis techniques, EvDetector inserts analysis code that inspects the parameters values. After executing the API function, EvDetector analyzes where malware stored the return value. This approach has the advantage that EvDetector does not need to patch some areas of the process memory or inject files. As shown in Algorithm 1, when malware called a well-known API used for anti-analysis techniques, we insert code that checks the parameters of the API and code and the code identifies the address of a buffer that the API dynamically allocates. For example,

Algorithm 1 API Hooking for Anti-Analysis Techniques

Input: Current API(curRTN)
Output: Buffer's Address List

```
API_List ← 38 API used for anti-analysis techniques
Signature ← 370 signatures used for anti-analysis
if curRTN ⊂ API_List then
    if curRTN_Param = Signature then
        Replace curRTN with wrapped_curRTN
    end if
end if
```

```
// Syntax of GetSystemFirmwareTable API
// UINT GetSystemFirmwareTable(
//   [in]  DWORD FirmwareTableProviderSignature,
//   [in]  DWORD FirmwareTableID,
//   [out] PVOID pFirmwareTableBuffer,
//   [in]  DWORD BufferSize
// );

WINDOWS::UINT GetSystemFirmwareTable_Hooked(...){
    PIN_CallApplicationFunction(...);
    ...
    if (FirmwareTableProviderSignature == 'RSMB'){
        storeBufAddr(pFirmwareTableBuffer,
                     BufferSize, ...);
    }
    return retVal;
}
```

Listing 2. Example of the API Hooking: It shows that intel pin hooked getsystemfirmwaretable API for identifying the qpfirmwaretablebuffer's address.

a technique to check the SMBIOS of a system could call the GetSystemFirmwareTable API. To check SMBIOS, the first parameter (FirmwareTableProviderSignature) must be "RSMB." SMBIOS information is stored in the buffer pointed by the third parameter (pFirmwareTableBuffer). Therefore, as shown in Listing 2, we insert code that examines the parameters and identifies the buffer. Since the location of in/out parameters is different for each Windows API, we write hooking code for each API we want to analyze.

Algorithm 2 Trace the Usage Flow of the Buffer

Input: Current Instruction(curINS)

BufferAddrList
← Buffers related to anti-analysis API

if *curINS_Operand*2 ⊂ *BufferAddrList* **then**
  if *curINS* = *mov_INS* **then**
    storeBufferAddr(*curINS_Operand*1)
  end if
end if

if *curINS_Operand*1 ⊂ *BufferAddrList* **then**
  if *curINS* = *push_INS* **then**
    storeBufferAddr(*curINS_Operand*1)
  end if
  if *curINS* = *pop_INS* **then**
    deleteBufferAddr(*curINS_Operand*1)
  end if
end if

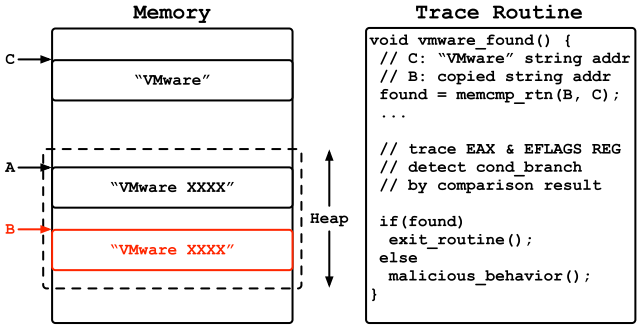

FIGURE 2. Depending on the detection result, the execution result of the conditional branch instruction is different. At this time, exit_routine or malicious_behavior may be operated by conditional branching, and the execution result may be completely opposite.

2) TRACKING THE DATAFLOW FROM THE BUFFER

Because the Windows API's dynamically-allocated buffers have a different address on each run, they must be identified each time correctly in the previous process. After the API hooking code injection, as in Algorithm 2, we track the dataflow of the stored value in the buffer. There is a possibility that the malware backs up the target buffer to another heap or stack address. The anti-analysis techniques can be used by referring to the backed-up buffer in the subsequent execution process. As shown in Figure 2, the malware can compare values using the copied string address. Because this case must be handled to detect an anti-analysis strategy, it is essential to accurately trace the dataflow. EvDetector, thus, traces the dataflow until it meets conditional branches using the return value of the APIs based on Algorithm 2.

3) DETECTING THE BUFFER AFFECT TO CONDITIONAL BRANCH

While tracing the dataflow from the stored values in the buffers, EvDetector inspects instructions that compares the values with other variables. EvDetector also inspects APIs that compare two data in memory. After an instruction or API compares buffers, it will change the value of the EFLAGS register. We look for a conditional branch affected by the EFLAGS value until it changes again. Algorithm 3 and Figure 2 shows how to find a conditional branch after comparing the buffer values.

Algorithm 3 Compare Buffers and Find Conditional Branch

Input: Current Instruction(curINS), Current API(curRTN)

findCondBr ← *False*
BufferAddrList
← Buffers related to anti-analysis API
EFLAGS_Value ← 0

if *curINS* = *cmp_INS* **then**
  if *curINS_Operand* ⊂ *BufferAddrList* **then**
    findCondBr ← *True*
    EFLAGS_Value ← *EFLAGS_REG*
  end if
end if

if *curRTN* = *memcmp_API* **then**
  if *curRTN_Param* ⊂ *BufferAddrList* **then**
    findCondBr ← *True*
    EFLAGS_Value ← *EFLAGS_REG*
  end if
end if

if *findCondBr* = *True* AND *curINS* = *cond_br_INS* **then**
  if *EFLAGS_Value* = *EFLAGS_REG* **then**
    identifyCondBr()
  end if
end if

4) BYPASSING MODULE FOR ANTI-DBI TECHNIQUES

When EvDetector analyzes malware, it should obtain the same result as the general execution result for the reliability. However, DBI frameworks such as Intel Pin can be detected by Anti-DBI techniques, as discussed in the previous section. EvDetector, thus, has to be resist against the anti-DBI techniques. For implementing the bypassing module, we are resistant to the following techniques. The ProcessDebugFlags of the NtQueryInformationProcess API can detect Intel Pin. This technique returns 0 in ProcessInformation if debugging is in progress. If the value of ProcessInformation is 0, the anti-DBI detection technique detects that the process is being debugged. The bypassing module detects the ProcessDebugFlags technique. We bypassed this technique by abusing the value of ProcessInformaton to be 1 instead of 0.

By default, all debuggers and DBIs ignore exceptions raised by debuggee. All exceptions have the characteristic of being handled directly by the analysis tool. To bypass

the detection techniques using the exception, analysis tools should handle exceptions without ignoring them. If the technique sets the value of Trap Flag in the EFLAGS register to 1, the CPU runs single-step mode. A single-step exception occurs when the program executes an instruction in single-step mode. There is a singularity that branch instructions do not work in this case. To bypass this, when the program called popfd instruction, we check the Trap Flags value. We manipulate the EIP and ESP registers to execute the single-step exception handling code if the single-step exception occurs. After that, Intel Pin handles the exception without ignoring it.

When accessing the memory area set as PAGEGUARD, an exception occurs. To bypass the PAGEGUARD exception technique, when accessing the PAGEGUARD memory, we handle an exception for ACCESS_INVALID_PAGE.

NtGlobalFlag is one of the variables of the PEB structure. During debugging, the debugger set NtGlobalFlag to 0×70. Intel Pin has the characteristic of storing 0×70 to NtGlobalFlag of 64-bit PEB structure when 32-bit Program runs in 64-bit OS. Since the detection technique can detect NtGlobalFlag value of the 64-bit PEB structure, we bypass it by manipulating this value as 0.

VI. EVALUATION

This section describes the results of analyzing real-world malware dataset using EvDetector. Specifically, we address the following research questions.

- **RQ1.** How many in-the-wild malware are using anti-analysis techniques?
- **RQ2.** Can EvDetector effectively find anti-analysis techniques implemented in real-world malware?
- **RQ3.** Compared to the previous approaches, can EvDetector find anti-analysis techniques more accurately in terms of the number of false-positives?

A. EXPERIMENTAL SETUP

1) MALWARE DATASET

To investigate anti-analysis techniques used in malware over the 4 years prior to the research (2017-2020). We collected 763,985 real-world malware samples provided by VirusShare (2017: 349,256, 2018: 119,952, 2019: 120,942, 2020: 173,785).

2) EXPERIMENTAL SETUP

We implemented EvDetector based on the Intel Pin (V3.20.98437) and EvDetector analyzed each malware on a Windows 10 VM (Windows 10 Pro, 21H1, V19043.1415). We conducted our experiments on three Ubuntu 18.04 host machines. We used 40 VMs in total to analyze the large dataset. Each VM was set to have 2 CPUs and 4GB of RAM. Also, we limit the execution time of each malware to be 3 minutes.

3) EXPERIMENTAL METHOD

VMware can control the guest VM on the host by using command line interface. We uploaded malware from the host machine to a guest VM via the shared directory. Then, EvDetector ran the malware and downloaded the execution log from the guest to the host. Using a script, we used 40 VMs running the malware sequentially.

TABLE 3. The detection result of packed malware by year.

Year	Total	Packed	Ratio (%)
2017	349,256	121,063	34.66
2018	119,952	34,173	28.49
2019	120,942	35,368	29.24
2020	173,785	55,895	21,16

TABLE 4. The detection detail result of packed malware: shows the number of detections per packer each year.

Packer	2017	2018	2019	2020
Unknown	83,309	17,856	24,063	29,719
UPX	9,341	12,098	4,881	7,820
Armadillo	17,086	3,197	4,086	15,176
PECompact	9,725	160	664	674
VMProtect	393	534	697	1,136
MPRESS	658	112	484	639
ASPack	357	140	344	227
FSG	28	4	20	284
Themida	1	12	26	167
NSPack	101	20	35	22
PEtite	20	10	18	19
NeoLite	35	13	14	5
ASProtect	7	16	9	3
Packman	1	0	27	1
eXpressor	1	1	0	2
PESpin	0	0	0	1

B. FINDING PACKED AND EVASIVE MALWARE

1) PACKED MALWARE

Before measuring the anti-analysis techniques used in malware, we evaluated the malware dataset to find how many malicious applications are packed with an open-source tool: PyPackerDetect [6] that provides an analysis function that integrates various heuristic analysis methods, such as the signature of PEiD [4] and the section names of well-known packers. Table 3 shows the packing detection results in our malware dataset: In our dataset, about 30% of malware was packed by known packers every year. Additionally, we analyzed which packers are mostly used for the malware. Table 4 shows that the analysis result. Unknown in the table means that malware used two or more packers together, or it is impossible to identify a packer. Notably, ''Unknown Packer'' accounts for more than half of the packed malware samples. Otherwise, the other malware used a single packer to pack an executable file. Malware used UPX [1] and Armadillo overwhelmingly compared to other packers. Next, PECompact, VMProtect, ASPack [3], and Themida are widely applied. These packers provide protection

TABLE 5. The detection result of anti-analysis techniques of malware by year.

Year	Total	Detected	Ratio (%)
2017	349,256	76,441	21.89
2018	119,952	16,476	13.72
2019	120,942	9,982	8.25
2020	173,785	20,978	12.06

techniques such as anti-Debug and anti-VM as well. Many studies have been conducted to analyze Packer's techniques for the packed malware that provides the self protection techniques. Those results made us question how many packing samples use evasion techniques to detect sandbox-based analysis. In subsection VI-C, we demonstrate evaluation results of how many packed malicious applications use anti-analysis techniques using EvDetector.

2) DETECTION RESULTS OF ANTI-ANALYSIS TECHNIQUES

We analyze how many malicious applications in our dataset equip the anti-analysis techniques with EvDetector. In addition, based on the detection results, we performed other analyses in subsection VI-C, subsection VI-D.

We first paid attention to the detection rate because the number of malware samples varies widely from year to year. Table 5 shows the results of malware that tries to evade analysis environments. On average, around 11% of malware in 2018–2020 is still detecting the analysis environment. In 2017, the ratio of malware that detects sandbox environments was about 21.89%, which is the highest rate among the dataset by year used in our experiments. Since then, in the 2018–2020 malware dataset, 13.72%, 8.25%, and 12.06% of malware are detecting sandbox environments, respectively. These results show a slight decline in the frequency of using the virtualization and sandbox detection techniques. However, this does not mean that malware using other anti-analysis techniques decreases. As mentioned in section III, because virtualization and sandbox detection techniques are a part of many anti-analysis techniques, we believe the research community needs to work on the other anti-analysis techniques continuously.

C. THE EFFECTIVENESS OF EvDetector

1) THE EFFECTIVENESS

As we discussed in section II, the previous approaches for detecting the anti-analysis techniques usually find only the existence of the signatures of anti-analysis techniques in malware. However, the existence of the signatures of anti-analysis techniques does not always mean a malicious application implements an anti-analysis techniques because Windows APIs used for evasive malware can have multiple functionalities depending on the arguments. Therefore, in this evaluation, we compared two anti-analysis detection results: One is detected by a static analysis that finds signatures related to APIs that can be used for implementing

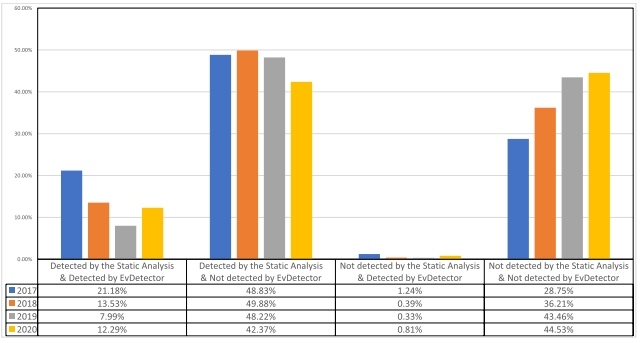

FIGURE 3. Comparison of detection results between the static analysis and EvDetector.

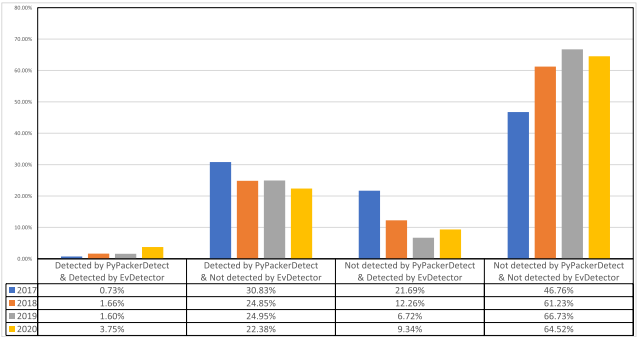

FIGURE 4. Comparison of detection results between the packing and EvDetector.

anti-analysis techniques that detect virtual machines, and sandboxes, the other one is detected by EvDetector. Figure 3 demonstrates the results of comparing the signature detection result with the EvDetector. The static analysis shows that many malware use anti-analysis techniques every year. On the other hand, when analyzing through EvDetector, it analyzed that most of the detected malware does not use the anti-analysis—when we use only the static analysis, the results can have a lot of false-positives. Also, as in Figure 3, even when a static analysis did not find any signature, EvDetector could detect anti-analysis techniques—the static analysis result can have false-negatives. Those results demonstrate the importance of the dataflow analysis that EvDetector employs to accurately detect anti-analysis techniques.

2) ANALYSIS RESULTS OF PACKED MALWARE

As mentioned earlier, packers such as PECompact, VMProtect, and Themida provide features such as Anti-VM and Anti-Sandbox. However, it is unknown that the other packers are also employ anti-analysis techniques. Therefore, we analyzed the correlation between packing techniques and anti-analysis techniques. As Table 3 shows, in our dataset, averagely 30% of malware by year use packers to protect themselves. However, our analysis results demonstrate that the most of the packed samples does not use anti-analysis techniques as shown in Figure 4.

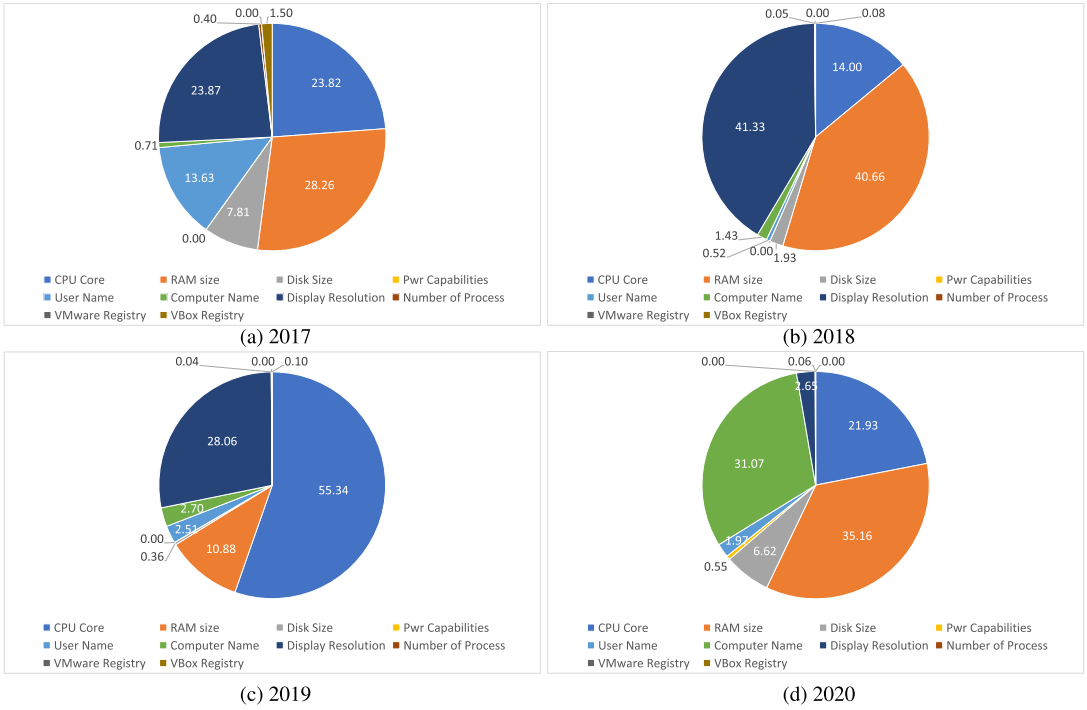

FIGURE 5. Analysis of evasive malware usage techniques each year: shows that malware steadily uses hardware detection, and other methods show a different distribution every year.

TABLE 6. The detection detail result of evasive malware: shows the number of techniques per malware each year.

Count	2017	2018	2019	2020
1	26,203	4,340	8,886	9,022
2	13,344	10,705	895	8,446
3	24,723	990	159	2,420
4	11,877	373	38	978
5	269	61	4	99
6	20	5	0	7
7	4	1	0	1
8	1	1	0	0
9	0	0	0	0
over 10	0	0	0	5

3) DETAILED RESULTS OF ANTI-ANALYSIS TECHNIQUES

Most malware appeared to use 1 to 4 techniques, as shown in Table 6. The number of malware using more than 5 techniques was significantly reduced. We found malware using more than 10 anti-analysis techniques only in the 2020 dataset. Next, we investigated what characteristics of the malware scans in the analysis environment. As shown in Figure 5, most malware inspected hardware characteristics such as CPU core and RAM size and OS environments such as display resolution and username.

D. ANALYSIS USING avclass2

We used avclass2 [44] and VirusTotal to further analyze evasive malware detected by EvDetector. Avclass2 is a tool that automatically classifies malware and categorizes it by a class, family, and its behavior. We collected VirusTotal's reports of only evasive malware detected by EvDetector 2018–2020. We, then, classified the collected analysis reports using Avclass2. Because we analyzed only Windows malware, we categorized them based on the families and behaviors of the malware detected.

1) BEHAVIORAL-ANALYSIS

We analyzed the classification results according to behaviors for evasive malware in 2018–2020 (2018: 24,767, 2019: 11,015, 2020: 24,585). We note that, as a result of the analysis, there are cases where one malware classified several overlapping actions. We found that evasive malware by year consisted of 2018: 29, 2019: 33, and 2020: 35 behaviors, respectively. Also, behaviors are becoming increasingly diverse. Figure 6 shows the detection results of the top 10 behaviors by year. The top five behaviors continue to rank in the top five annually. Based on this analysis result, we can observe that malware which exhibits "infosteal," "inject," "execdownload," and "filemodify" behaviors usually use anti-analysis techniques.

2) FAMILIAL-ANALYSIS

According to family classification, we analyzed the results for 2018–2020 (2018: 13,149/16,476, 2019: 7,532/9,982, 2020: 10,855/20,978). We found that evasive malware by year consisted of 2018: 121, 2019: 125, and 2020: 164 families, respectively. Families are becoming increasingly diverse.

TABLE 7. The analysis results of the top 30 family types out of 205 show that as time passes, various families are emerging. There were 40 newly discovered types in 2019 and 44 freshly discovered types in 2020. Among the newly discovered types, only sfone and palevo types in 2019 and only agenttesla and qbot types in 2020 ranked in top 30.

2018	Count	Ratio	2019	Count	Ratio	2020	Count	Ratio
fareit	9,511	72.33%	fareit	1,923	25.54%	fareit	1,895	17.53%
soft32downloader	864	6.57%	vittalia	1,406	18.67%	crossrider	1,778	16.45%
relevantknowledge	855	6.50%	soft32downloader	889	11.81%	emotet	1,312	12.14%
zusy	323	2.46%	dlhelper	807	10.72%	banload	547	5.06%
dlhelper	251	1.91%	zusy	633	8.41%	cobaltstrike	527	4.88%
amonetize	103	0.78%	relevantknowledge	290	3.85%	gozi	472	4.37%
vittalia	101	0.77%	razy	274	3.64%	remcos	389	3.60%
emotet	85	0.65%	amonetize	123	1.63%	revil	346	3.20%
lokibot	77	0.59%	azorult	105	1.39%	zusy	277	2.56%
hotbar	69	0.52%	sfone	87	1.16%	installcore	207	1.92%
softcnapp	68	0.52%	swizzor	85	1.13%	razy	199	1.84%
installmonster	63	0.48%	emotet	84	1.12%	soft32downloader	186	1.72%
uniblue	49	0.37%	installcore	57	0.76%	gamarue	181	1.67%
skeeyah	48	0.37%	hotbar	48	0.64%	nanocore	142	1.31%
gamarue	48	0.37%	softcnapp	48	0.64%	softcnapp	141	1.30%
remcos	37	0.28%	banload	35	0.46%	bladabindi	140	1.30%
installcore	34	0.26%	uniblue	29	0.39%	relevantknowledge	125	1.16%
razy	33	0.25%	gamarue	28	0.37%	lokibot	118	1.09%
wannacry	28	0.21%	presenoker	28	0.37%	azorult	116	1.07%
azorult	27	0.21%	onlinegames	27	0.36%	loadmoney	103	0.95%
speedingupmypc	25	0.19%	skeeyah	25	0.33%	agenttesla	100	0.93%
presenoker	23	0.17%	zbot	25	0.33%	qbot	89	0.82%
zbot	22	0.17%	conduit	23	0.31%	trickbot	86	0.80%
somoto	19	0.14%	ramnit	21	0.28%	mywebsearch	81	0.75%
carberp	18	0.14%	wapomi	19	0.25%	presenoker	77	0.71%
spigot	17	0.13%	trickbot	19	0.25%	netwiredrc	69	0.64%
banload	16	0.12%	revil	17	0.23%	dlhelper	53	0.49%
darkkomet	15	0.11%	cobaltstrike	16	0.21%	zbot	51	0.47%
opencandy	13	0.10%	palevo	16	0.21%	crysis	47	0.43%
wapomi	11	0.08%	lokibot	15	0.20%	vittalia	47	0.43%

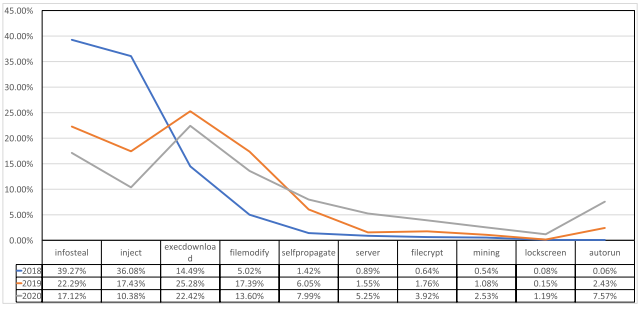

FIGURE 6. A result of the top 10 behavioral analyses of evasive malware: shows that they are used in various categories as time goes by.

Table 7 shows the detection results of the top 30 families by year. It is noteworthy that the "fareit" malware family accounts for 72.33% of evasive malware in 2018, but it is decreasing year by year, and it can see that various families use anti-analysis techniques. However, still, we can find that malicious applications in the "fareit" family widely use the anti-analysis techniques.

VII. LIMITATION

1) HARDWARE COMPATIBILITY

We use Intel Pin to analyze Windows malware in this work. Intel Pin can analyze binaries reliably on Intel CPU-based operating systems (e.g., Windows, Linux, and macOS). However, unlike the other DBI frameworks, Intel Pin only works with Intel CPUs. As a result, we should re-implement EvDetector for using it on the other CPUs.

2) OUT OF SYMBOL PROBLEM

In this work, we employ Intel Pin to trace Windows APIs used in the anti-analysis techniques and return values of them. Intel Pin provides the "Routine Replace" function to help implementing API hooking functions as shown in Listing 2. However, Windows OS does not provide all symbols of APIs and services needed to implement hooking functions. For example, the WMI, which is one of the frequently used service to check the system information. This service queries the system information through called ExecQuery and Get methods. Despite the WMI symbol provided by Windows OS, Intel Pin cannot hook the methods because the symbols of

the methods are not exported. Therefore, our analysis has a limitation that cannot hook all APIs which can be used to implement an anti-analysis technique.

3) API CALL CHAIN

Some of the evasion techniques detect through the linked call-chain while reusing the handle to call each API. The FindFirstFile-FindNextFile call-chain is one of the evasion techniques that finds files related to sandboxes or VMs. In this work, we did not consider how to track the handle, as we only focused on the APIs' dynamically allocated buffers. For this reason, EvDetector is limited to detect the anti-analysis techniques that work through such call-chains.

4) UNLABELED DATASET PROBLEMS

Our goal is to detect evasive malware and to analyze anti-analysis techniques trends. We collected in-the-wild malware dataset that does not have labels to show whether each malware equips anti-analysis techniques or not. We analyzed the trends of evasive malware. Our analysis results are meaningful in that we can classify them in more detail than the prior and analysis tools. However, it is impossible to have ground truth regarding anti-analysis techniques used in the malware. We leave this limitation as future work and would like to solve in the near future.

5) POSSIBLE DETECTION ERRORS AND THE SCOPE OF DETECTION

To track the data flow associated with the buffers generated by the APIs, we trace instructions such as MOV, PUSH, and POP. Also, we monitor the CMP-like instructions and MEMCMP-like APIs used to compare data in memory. However, since we do not know every Windows APIs that can modify memory, there can be missing APIs that we should trace. In addition, even though, we dynamically trace a malicious application to find whether or not the use of APIs used in anti-analysis techniques affects the execution flow of the malware, we did not completely validate our analysis results, and thus, false-positives may exist. For example, a conditional branch that EvDetector observed could not be the one to decide whether or not executing malicious behaviors. Also, we assume that evasive malware will first scan the execution environment on which it runs before performing malicious behaviors. We, thus, focused on anti-analysis techniques that detects sandboxes and virtual machine-based analysis environments. However, as discussed in section III, various anti-analysis techniques exist in addition to the sandbox and virtual machine detection techniques. We leave those limitation as our future work.

VIII. CONCLUSION

In this work, we investigated malware's anti-analysis techniques that effectively evade anti-virus vendor's analysis and how much Windows malware detects the analysis environment. To this end, we design and implement an analysis system, EvDetector, that finds if malware uses anti-analysis techniques and monitors whether the malware changes its execution path through anti-analysis results. By using EvDetector, we analyzed 763,385 in-the-wild malware from 2017 to 2020. To our knowledge, this is the largest dynamic analysis study involving anti-analysis techniques. Our evaluation results show that at least 8% to 21% of malware can detect sandbox-based analysis each year. To check the effectiveness of the analysis result, we compared the static analysis result and the result of EvDetector. Analyzing through EvDetector, most of the samples detected by the static analysis did not use anti-analysis techniques. We also analyzed samples that use commercial protectors, which provide a variety of anti-analysis techniques. Our evaluation results demonstrate that most packed malware does not detect sandbox. Finally, we analyzed the malware trend that uses anti-analysis techniques through familial analysis and behavioral analysis. Our analysis implies that the research community needs to put more effort into defeating such anti-analysis techniques to automatically analyze emerging malware and respond with them.

REFERENCES

[1] *UPX—The Ultimate Packer for Executables, 1996–2022*. Accessed: Sep. 7, 2020. [Online]. Available: https://upx.github.io/
[2] *VMProtect Software Protection, 2003–2022*. Accessed: Sep. 7, 2020. [Online]. Available: https://vmpsoft.com/
[3] *ASPACK Software, 2007–2022*. Accessed: Sep. 7, 2020. [Online]. Available: http://www.aspack.com/
[4] *PEiD, 2008–2022*. Accessed: Mar. 4, 2021. [Online]. Available: https://www.aldeid.com/wiki/PEiD
[5] *Al-Khaser, 2015–2022*. Accessed: Jun. 24, 2020. [Online]. Available: https://github.com/LordNoteworthy/al-khaser
[6] *PyPackerDetect, 2018–2022*. Accessed: Apr. 6, 2021. [Online]. Available: https://github.com/cylance/PyPackerDetect
[7] L. Abrams. *Malware Adds Online Sandbox Detection to Evade Analysis, 2020–2022*. Accessed: Jul. 14, 2021. [Online]. Available: https://www.bleepingcomputer.com/news/security/malware-adds-online-sandbox-detection-to-evade-analysis/
[8] A. Algawi, M. Kiperberg, R. Leon, A. Resh, and N. Zaidenberg, "Creating modern blue pills and red pills," in *Eur. Conf. Cyber Warfare Secur.*, Coimbra, Portugal, 2019, pp. 6–14.
[9] Any.Run. (2021). *Interactive Malware Hunting Service*. [Online]. Available: https://any.run/
[10] Y. Assor. *Anti-VM and Anti-Sandbox Explained, 2016–2022*. Accessed: Jul. 14, 2021. [Online]. Available: https://www.cyberbit.com/blog/endpoint-security/anti-vm-and-anti-sandbox-explained
[11] U. Bayer, I. Habibi, D. Balzarotti, E. Kirda, and C. Kruegel, "A view on current malware behaviors," in *Proc. LEET*, 2009, pp. 1–11.
[12] U. Bayer, C. Kruegel, and E. Kirda, *TTAnalyze: A Tool for Analyzing Malware*. 2006.
[13] J. Bolg. *Analyzing Azorult's Anti-Analysis Tricks With Joe Sandbox Hypervisor, 2020–2022*. Accessed: Jul. 14, 2021. [Online]. Available: https://www.joesecurity.org/blog/9048980422564630717
[14] 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*, vol. 1, pp. 1–27, 2012. Accessed: Mar. 8, 2021. [Online]. Available: https://kernelhacking.com/rodrigo/docs/blackhat2012-paper.pdf
[15] 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 *Proc. IFIP Int. Conf. ICT Syst. Secur. Privacy Protection*, Ghent, Belgium, May 2016, pp. 323–336.
[16] S. Choi, T. Chang, S.-W. Yoon, and Y. Park, "Hybrid emulation for bypassing anti-reversing techniques and analyzing malware," *J. Supercomput.*, vol. 77, no. 1, pp. 471–497, Jan. 2021.

[17] A. Dahan. *New Betabot Campaign Under the Microscope, 2018–2022*. Accessed: Jul. 14, 2021. [Online]. Available: https://www.cybereason.com/blog/betabot-banking-trojan-neurevt
[18] A. Dinaburg, P. Royal, M. Sharif, and W. Lee, "Ether: Malware analysis via hardware virtualization extensions," in *Proc. 15th ACM Conf. Comput. Commun. Secur.*, New York, NY, USA, 2008, pp. 51–62.
[19] FireEye. (2020). *M-Trends 2020*. [Online]. Available: https://content.fireeye.com/m-trends/rpt-m-trends-2020
[20] Stichting Cuckoo Foundation. *Cuckoo Sandbox—Automated Malware Analysis, 2010–2022*. Accessed: Apr. 7, 2021. [Online]. Available: https://cuckoosandbox.org/
[21] H. Triage. *Hatching Triage is a Fully Automated Solution for High-Volume Malware Analysis Using Advanced Sandboxing Technology, 2018–2022*. Accessed: Feb. 15, 2021. [Online]. Available: https://tria.ge/
[22] JoeSecurity. *Why Joe Sandbox? 2003–2022*. [Online]. Available: https://www.joesecurity.org/why-joe-sandbox
[23] Y. Kawakoya, M. Iwamura, E. Shioji, and T. Hariu, "API chaser: Anti-analysis resistant malware analyzer," in *Proc. Int. Workshop Recent Adv. Intrusion Detection*. Berlin, Germany: Springer, 2013, pp. 123–143.
[24] Y. Kawakoya, E. Shioji, M. Iwamura, and J. Miyoshi, "API chaser: Taint-assisted sandbox for evasive malware analysis," *J. Inf. Process.*, vol. 27, pp. 297–314, Jan. 2019.
[25] D. Kirat, G. Vigna, and C. Kruegel, "BareBox: Efficient malware analysis on bare-metal," in *Proc. 27th Annu. Comput. Secur. Appl. Conf.*, New York, NY, USA, Dec. 2011, pp. 403–412.
[26] D. Kirat, G. Vigna, and C. Kruegel, "BareCloud: Bare-metal analysis-based evasive malware detection," in *Proc. 23rd USENIX Secur. Symp. (USENIX Secur.)*, San Diego, CA, USA, Aug. 2014, pp. 287–301.
[27] KISA. (2021). *2021 1H Cyber Security Issue Report*. Accessed: Jul. 14, 2021. [Online]. Available: https://krcert.or.kr/filedownload.do?attach_file_seq=3431&attach_fil% e_id=EpF3431.pdf
[28] Y. B. Lee, J. H. Suk, and D. H. Lee, "Bypassing anti-analysis of commercial protector methods using DBI tools," *IEEE Access*, vol. 9, pp. 7655–7673, 2021.
[29] O. Levi. *PIN—A Dynamic Binary Instrumentation Tool, 2007–2022*. Accessed: Mar. 2, 2020. [Online]. Available: https://www.intel.com/content/www/us/en/developer/articles/tool/pin-a-dynamic-binary-instrumentation-tool.html
[30] Malwarebytes Labs. (2020). *2020 State of Malware Report*. [Online]. Available: https://www.malwarebytes.com/resources/files/2020/02/2020_state-of-malware-report.pdf
[31] M. Sikorski and A. Honig, *Practical Malware Analysis: The Hands-on Guide to Dissecting Malicious Software*. San Francisco, CA, USA: William Pollock, 2012.
[32] N. Miramirkhani, M. P. Appini, N. Nikiforakis, and M. Polychronakis, "Spotless sandboxes: Evading malware analysis systems using wear-and-tear artifacts," in *Proc. IEEE Symp. Secur. Privacy (SP)*, San Jose, CA, USA, May 2017, pp. 1009–1024.
[33] MITRE ATT&CK. *Virtualization/Sandbox Evasion, 2019–2022*. Accessed: Aug. 9, 2021. [Online]. Available: https://attack.mitre.org/techniques/T1497/
[34] NIST (National Institute of Standards and Technology). *Juilet Test Suites, 2010–2022*. Accessed: Jan. 14, 2021. [Online]. Available: https://samate.nist.gov/SRD/testsuite.php
[35] Obsidium Software. (2022). *Obsidium Software Protection System*. [Online]. Available: https://www.obsidium.de/
[36] Oreans Technologies. *Themida Overview—Oreans Technologies, 2004–2022*. Accessed: Mar. 2, 2020. [Online]. Available: https://www.oreans.com/Themida.php
[37] A. Ortega. *Pafish, 2012–2022*. Accessed: Jun. 23, 2020. [Online]. Available: https://github.com/a0rtega/pafish
[38] Y. Oyama, "Investigation of the diverse sleep behavior of malware," *J. Inf. Process.*, vol. 26, pp. 461–476, Jan. 2018.
[39] Y. Oyama, "Trends of anti-analysis operations of malwares observed in API call logs," *J. Comput. Virol. Hacking Techn.*, vol. 14, no. 1, pp. 69–85, Feb. 2018.
[40] Panda Security. (2020). *Panda Security Launches Its Threat Insights Report 2020*. [Online]. Available: https://www.pandasecurity.com/en/mediacenter/panda-security/threat-insights-report-2020/
[41] J. Park, Y.-H. Jang, S. Hong, and Y. Park, "Automatic detection and bypassing of anti-debugging techniques for Microsoft windows environments," *Adv. Electr. Comput. Eng.*, vol. 19, no. 2, pp. 23–29, 2019.
[42] QEMU. *QEMU, a Generic and Open Source Machine Emulator and Virtualizer, 2009–2022*. [Online]. Available: https://www.qemu.org/
[43] A. S. Sai Omkar Vashisht. *Turing Test in Reverse: New Sandbox-Evasion Techniques Seek Human Interaction, 2014–2022*. Accessed: Jul. 14, 2021. [Online]. Available: https://www.fireeye.com/blog/threat-research/2014/06/turing-test-in-reverse-new-sandbox-evasion-techniques-seek-human-interaction.html
[44] S. Sebastián and J. Caballero, "AVclass2: Massive malware tag extraction from AV labels," in *Proc. Annu. Comput. Secur. Appl. Conf.*, Dec. 2020, pp. 42–53.
[45] The Enigma Protector. *Enigma Protector, 2004–2022*. [Online]. Available: https://enigmaprotector.com/
[46] C. S. Thomas Roccia. *Evolution of Malware Sandbox Evasion Tactics—A Retrospective Study, 2019–2022*. Accessed: Jul. 14, 2021. [Online]. Available: https://www.mcafee.com/blogs/other-blogs/mcafee-labs/evolution-of-malware-sandbox-evasion-tactics-a-retrospective-study/
[47] VMRay. *VMRay—Cyber Security Threat Detection & Analysis Platform, 2015–2022*. Accessed: Apr. 9, 2021. [Online]. Available: https://www.vmray.com/
[48] VMware. *VMware, 2006–2022*. Accessed: Mar. 4, 2020. [Online]. Available: https://www.vmware.com/
[49] Yaldex. *Acprotect Standard, 2006–2022*. Accessed: Jan. 15, 2021. [Online]. Available: http://www.yaldex.com/Bestsoft/Utilities/acprotect.htm
[50] A. Yokoyama, K. Ishii, R. Tanabe, and Y. Papa, "SandPrint: Fingerprinting malware sandboxes to provide intelligence for sandbox evasion," in *Proc. Int. Symp. Res. Attacks, Intrusions, Defenses*, Paris, France, Sep. 2016, pp. 165–187.

MINHO KIM received the B.S. degree in electronic engineering and the M.S. degree in computer science and engineering from Soongsil University, in 2020 and 2022, respectively, where he is currently pursuing the Ph.D. degree with the School of Software. He is also a Research Staff with the Cyber Security Research Center. His research interests include binary analysis, software engineering, reverse engineering, and systems security.

HAEHYUN CHO received the B.S. and M.S. degrees in computer science from Soongsil University, Seoul, South Korea, in 2013 and 2015, respectively, and the Ph.D. degree from the School of Computing, Informatics and Decision Systems Engineering, Arizona State University, majoring in computer science, and especially concentrating on information assurance. He is currently an Assistant Professor with the School of Software and the Co-Director of the Cyber Security Research Center, Soongsil University. His primary research interests include the field of systems security, which is to address and discover security concerns stemmed from insecure designs and implementations. He is also passionate about analyzing, finding, and resolving security issues in a wide range of topics.

JEONG HYUN YI (Member, IEEE) received the B.S. and M.S. degrees in computer science from Soongsil University, Seoul, South Korea, in 1993 and 1995, respectively, and the Ph.D. degree in information and computer science from the University of California at Irvine, Irvine, in 2005. He was a Principal Researcher with the Samsung Advanced Institute of Technology, South Korea, from 2005 to 2008, and a member of Research Staff with the Electronics and Telecommunications Research Institute (ETRI), South Korea, from 1995 to 2001. From 2000 to 2001, he was a Guest Researcher with the National Institute of Standards and Technology (NIST), Maryland, USA. He is currently a Professor with the School of Software and the Director of the Cyber Security Research Center, Soongsil University. His research interests include mobile security and privacy, the IoT security, and applied cryptography.