Scarecrow: Deactivating Evasive Malware via Its Own Evasive Logic

Jialong Zhang†, Zhongshu Gu*, Jiyong Jang*, Dhilung Kirat*, Marc Ph. Stoecklin*, Xiaokui Shu*, Heqing Huang†
*IBM Research, †ByteDance AI Lab
†zhangjialong@bytedance.com, *{zgu, jjang, dkirat}@us.ibm.com,
*mtc@zurich.ibm.com, *xiaokui.shu@ibm.com, †huangheqing@bytedance.com

***Abstract*—Security analysts widely use dynamic malware analysis environments to exercise malware samples and derive virus signatures. Unfortunately, malware authors are becoming more aware of such analysis environments. Therefore, many have embedded evasive logic into malware to probe execution environments before exposing malicious behaviors. Consequently, such analysis environments become useless and evasive malware can damage victim systems with unforeseen malicious activities.**

However, adopting evasive techniques to bypass dynamic malware analysis is a double-edged sword. While evasive techniques can avoid early detection through sandbox analysis, it also significantly constrains the spectrum of execution environments where the malware activates. In this paper, we exploit this dilemma and seek to reverse the challenge by camouflaging end-user execution environments into *analysis-like* environments using a lightweight deception engine called SCARECROW. We thoroughly evaluate SCARECROW with real evasive malware samples and demonstrate that we can successfully deactivate 89.56% of evasive malware samples and the variants of ransomware (e.g., WannaCry and Locky) with little or no impact on the most commonly used benign software. Our evaluation also shows that SCARECROW is able to steer state-of-the-art analysis environment fingerprinting techniques so that end-user execution environments with SCARECROW and malware analysis environments with SCARECROW become *indistinguishable*.

***Index Terms*—Malware Analysis; Deceptive Execution Environments;**

I. INTRODUCTION

Malware has been widely distributed by attackers to gain illegitimate access to private computer systems, garner confidential information, and disrupt online services. In the past few years, malware has evolved rapidly to become more sophisticated and stealthy. It becomes more challenging to statically analyze advanced malware that adopts various obfuscation and packing techniques [28].

In order to effectively uncover malicious behaviors, security researchers and anti-virus companies largely rely on executing malware samples in controlled analysis environments, such as sandboxes, and dynamically tracing their malicious operations. In addition, many corporations start testing new software in analysis environments before deploying it in their enterprise machines [1]. Those analysis environments are usually deployed in an isolated network and are equipped with various comprehensive analysis techniques and tools. However, today's cybersecurity is a Cat-and-Mouse game. Recent malware variants often include *evasive logic* to detect underlying execution environments, i.e., evasive malware. If they find themselves running in an analysis environment, they stop executing any malicious code, e.g., they resort to exiting immediately or exhibiting only benign behaviors, in order to hinder further analysis. Over 80% of malware since mid-2015 has exhibited evasive behaviors, therefore most of the dynamic analysis environments become ineffective against such evasive malware [10].

Using evasive techniques is a double-edged sword for malware. Malware leverages evasive techniques to avoid being analyzed in the analysis environments. In turn, as defenders, we can also exploit the same evasive logic to deactivate evasive malware by camouflaging a regular end host as an *analysis environment*. In this paper, we systematically study the resources used by evasive malware. We design SCARECROW, a lightweight deception engine to deactivate evasive malware before the malware executes malicious code in end-user machines. We encapsulate specially crafted system resources that are commonly observed in malware analysis environments and deploy SCARECROW on the physical end host without incurring additional analysis performance overhead. Essentially, SCARECROW transforms the physical end host into an *analysis-like* environment from the view of evasive malware. In our setup, the predicates within evasive logic will be activated after detecting the deception engine and further restrain malware from conducting any malicious behavior.

There are several benefits of planting SCARECROW on the physical end hosts. First, since similar evasive techniques are often shared across different malware families, SCARECROW is able to deactivate previously unseen malware and sophisticated evasive malware that cannot be analyzed by the state-of-the-art analysis engines. For example, SCARECROW successfully deactivates recent WannaCry ransomware equipped with evasive logic before it encrypts file systems by deceiving network resources. Second, SCARECROW can proactively stop evasive malware on the end host before exposing malicious behaviors. Third, evasive malware often combines multiple evasive techniques to fingerprint the running environment, and it detects the environment as an analysis environment if any of the techniques works. It is a challenge for a transparent analysis system to defeat all fingerprinting techniques at the same time. However, SCARECROW only needs to successfully deceive one or a few out of many evasion checks.

978-1-7281-5809-9/20/$31.00 ©2020 IEEE
DOI 10.1109/DSN48063.2020.00027

To systematically assess the performance of SCARECROW, we evaluated it with 1,054 evasive malware samples from existing work [25] and 13 evasive malware samples from Joe Security [4]. We also evaluate SCARECROW with two state-of-the-art analysis environment fingerprinting systems developed by researchers. In summary, our work makes the following contributions:

- We design SCARECROW, a system to deactivate evasive malware in end-user execution environments before performing malicious behaviors. SCARECROW is malicious payload-agnostic such that it is able to deactivate highly obfuscated zero-day evasive malware.
- We develop a deception engine that encapsulates a wide spectrum of predicates in existing evasive logic to disguise a real end-user execution environment as a malware analysis environment. This deception engine incurs minimal performance overhead and can be easily deployed in end-user systems as a complement of existing anti-virus protection.
- We evaluate the effectiveness of SCARECROW through a large set of real-world evasive malware samples and benign software. Our evaluation demonstrates that SCARECROW can successfully deactivate 89.56% of evasive malware samples without impacting commonly used benign software.

II. SYSTEM DESIGN

A. System Overview

The goal of SCARECROW is to deactivate evasive malware before executing malicious code. Evasive malware can either terminate itself or camouflage as benign software in an analysis environment. We reversely exploit this specific characteristic to defend against evasive malware. By provisioning specific system resources that are unique in analysis environments, we transform real end-user systems into *sandbox-like* platforms to stop evasive malware from performing malicious behaviors. Furthermore, benign software typically does not behave differently on different platforms, thus SCARECROW has a negligible impact on benign software.

SCARECROW works as an on-demand service. When the target process tries to fingerprint the running environment, it provides an *analysis-like* environment to neutralize the malware with the following features:

- **Proactive Defense.** Deployed on end host, SCARECROW proactively stops malware before it exposes malicious behaviors. For example, SCARECROW deactivates evasive ransomware before it encrypts the whole file system.
- **Unknown Malware Defense.** SCARECROW is malicious payload-agnostic. It neutralizes advanced evasive malware that cannot even be analyzed in sandboxes. SCARECROW is transparent to obfuscation, such as packing.
- **Malware Variant Defense.** SCARECROW can defend against new variants of evasive malware because evasive techniques are often shared across different malware families. Malware developers could retrofit evasive logic to malware with a trivial effort. For example, the new variant of Cerber [6] explores new tricks to evade machine learning based detection; however, it still re-uses the same anti-VM check to determine the running environment and stop execution in a sandbox environment.

B. Deceptive Execution Environment

The goal of SCARECROW is to provide a target process with a deceptive execution environment view. For this purpose, we first learn the deterministic system resources that characterize dynamic malware analysis environments. We investigate known existing evasive techniques from existing research papers and articles [17, 30, 36] and group execution environment resources into three categories: software resources, hardware resources, and network resources.

Software resources are software-layer abstraction entities, including files and folders, processes, libraries, GUI windows, registries, and function hooks. Such resources are independent from the underlying hardware and are commonly used by evasive malware to fingerprint analysis environments [16, 29, 35].

(a) Files and Folders: certain files and folders are created during the installation of virtual machines, sandboxes, and deep analysis tools. Evasive malware exploits such information as the indications of malware analysis environments. For example, there are certain files that can be used as the indications of VMware (e.g., `vmmouse.sys`) [30] and evasive malware can check the presence of such files to detect a VMware virtual machine. Our deception engine provides deceptive files and folders available in different virtual machines, sandboxes, and security forensic tools.

(b) Processes: evasive malware checks specific processes associated with deep analysis tools and virtual machines. For example, processes `VBoxTray.exe` and `VBoxService.exe` observed in VirusTotal [14] sandbox, which reflects that VirusTotal sandbox runs on top of VirtualBox virtualized environment. In addition, evasive malware may terminate the process of forensic tools to prevent in-depth analysis; otherwise, malware stops further execution. We include 24 processes, such as `olydbg.exe`, `idap.exe`, and `PETools.exe`, in SCARECROW and protect them from being terminated by untrusted software.

(c) Libraries: the functionalities of virtual machines or forensic tools are often shipped in their unique dynamic link libraries (DLLs), and such customized DLLs could be used as the indications of analysis environments. For example, `SbieDll.dll` can be used as an indicator of sandbox analysis environment if it is loaded in memory. We incorporate 15 unique DLLs into SCARECROW.

(d) GUI windows: evasive malware also checks for GUI windows. For example, some evasive malware uses `FindWindow` API to look for active debugger windows as an indication of debugger presence. We embrace 6 debugger GUI windows and 4 sandbox related windows in SCARECROW.

(e) Registries: a registry is a hierarchical database that stores rich information about Windows operating system and

applications. Therefore, evasive malware often searches for the evidence of virtual machines, deep analysis tools, and sandbox configurations in a registry. For example, there are over 300 references in a registry to VMware. SCARECROW offers deceptive references to the forensic tools and virtual machines in a registry. In addition, evasive malware also explores system configuration entries in a registry to detect virtual machines. For example, registry key `HARDWARE\Description\System\SystemBiosVersion` could be used to detect VirtualBox when `VBOX` value is present. SCARECROW also fakes such configuration values by combining multiple virtual machine names. Furthermore, the recent analysis environment fingerprinting tool [29] also infers the user activities from certain registry entries and values. For example, `Software\Microsoft\Windows\CurrentVersion\Run` is used to evaluate if the system is actively used by users through collecting the number of programs that automatically run at system startup. SCARECROW only returns fewer entries to simulate a less active sandbox environment.

(f) Function Hooks: In order to hide the real system configuration and trigger more malicious behaviors of malware, some sandboxes utilize hooking techniques to hijack API calls from malware, and return manipulated values to malware. The normal execution on the end-user host usually does not have such hooks. Therefore, evasive malware employs anti-hooking techniques by monitoring if some function calls are hooked. For example, overwriting the first five bytes of a function could be an indication of inline hooking. SCARECROW utilizes inline hooking and we will discuss details in Section III-A.

(g) Exception processing: A transparent exception handling is one of the main requirements for transparent malware analysis [19]. Most dynamic analysis systems leverage exception handling to implement the analysis. For example, debuggers use software-based or hardware-based exceptions. Some shadow-page-based analysis systems are based on page fault exceptions. Evasive malware looks for the discrepancies in timing and other side effects to detect analysis environments. SCARECROW introduces deceptive timing discrepancies in default exception processing with minimal to no impact on benign applications.

Hardware resources reflect the properties of the hardware. Manipulating hardware resources requires extra care because benign software also retrieves deceptive hardware information and its behaviors might be affected. We note that sandboxes and virtual machines typically have some unique system configurations that are not widely observed in end-user machines. For example, the disk size of `C` drive in the Malwr [12] public sandbox is only 5GB, which is unusual on end-user systems. SCARECROW provides faked system configurations, such as disk size (50GB), memory size (1GB), and the number of cores (1)[1]. We acknowledge that these configurations may have an impact on benign software. For example, if benign software requires larger than 50GB of disk space, it may cause an error. However, our evaluation shows that such a configuration had a negligible impact on the majority of popular software (see Section IV), and specific values are easily adjustable by users if needed.

[1]We chose the values of deceptive system configuration based on public sandboxes.

Network resources denote the resources related to network traffic. Malware may generate and send requests to non-existent (NX) domains, e.g., via Domain Generation Algorithms (DGAs). Most sandboxes resolve such NX domains into some fake IP addresses to mimic "live" communications [27]. SCARECROW employs a similar approach for the non-existent domains. Specifically, it will always return the same reachable IP address for all the non-existent domain queries. As a result, it makes evasive malware believe that it is running in a sandbox-like environment.

C. Deceptive Resources Collection

We further collect information about the resources of public online sandboxes to complement our manually extracted resources. Public sandbox services accept the submission of suspicious samples, then execute the submitted samples in their sandboxes, and return the analysis reports to users. For example, VirusTotal allows users to upload and analyze the files in Cuckoo sandbox [5]. Malware authors often abuse those public sandbox services to test their malware before distribution [21]. Therefore, any system resources that are uniquely observed in those public sandboxes can be used by evasive malware for sandbox fingerprinting. To collect system resources on those sandboxes, we design a crawler and submit it to two popular public sandbox services: VirusTotal [14] and Malwr [12]. Our crawler collects the information about files, folders, registries, processes, and system configurations on these two public sandboxes, and sends it back to our server. Please note that the resources we collected from these pubic sandboxes may be also "deceptive resources" since those sandboxes may provide fake information to avoid being fingerprinted. However, as long as the "deceptive resources" are unique for the sandboxes, they can still be used as the indications of sandboxes. We then compare the crawled system resources with our clean bare-metal systems. The unique resources that are only present in public sandboxes are added to SCARECROW deception engine. As a result, 17,540 files, 24 processes, and 1,457 registry entries are added to SCARECROW.

One way to continuously learn new deceptive resources is to leverage the analysis results from MalGene [25]. MalGene automatically extracts evasion signatures by comparing the traces from two different environments where malware evades one of the environments while exposing malicious activities in another. One caveat is that MalGene reports a new evasive signature based on the *first* system resource that causes the deviation of the two traces, and other resources used by malware for fingerprinting would not be identified in case evasive malware uses multiple evasion techniques.

We acknowledge that our list of deceptive resources in each category is not exhaustive. However, we note that the system resources that could be used for fingerprinting analysis

environments are usually limited. Evasive malware typically uses multiples evasive techniques for fingerprinting. Malware across different malware families often shares the same evasive techniques. Like the Pareto principle, the unique advantage of SCARECROW is that a small subset of deceptive resources is enough to deactivate most evasive malware and we do not need to seek completeness. For example, if evasive malware wants to evade a debugger, it may look for related resources for multiple debuggers, including both popular and unpopular debuggers. From the perspective of malware, it is preferable to always check the popular debuggers because it gives evasive malware higher chances to detect commonly used debuggers. Therefore, SCARECROW deceives the resources relevant to popular debuggers, e.g., `Windbg` and `OllyDbg`. In our evaluation, we demonstrate that the current deception engine is already sufficient to deactivate most evasive malware.

III. REALIZATION OF SCARECROW

In this section, we discuss the realization of SCARECROW on end-user hosts. One can deploy the deception engine by simply creating many of the deceptive resources on an end-user machine. However, those deceptive resources are also visible to end users and other benign software, which may negatively interfere. Therefore, we consider the following four requirements when deploying SCARECROW. (a) SCARECROW should be transparent to users, i.e., users would not notice deceptive resources such as deceptive files and processes. (b) SCARECROW should have little or no impact on benign software behavior. It should only disrupt the behavior of potentially malicious software. (c) SCARECROW should have negligible performance overhead on a system. (d) SCARECROW should be easy to be deployed and set up on an end-user system without requiring special resources.

A. Hooking and DLL Injection

In order to address the above deployment requirements, we design SCARECROW based on function hooking using DLL Injection. When an application calls a function to access resources, the hooking technique allows us to intercept the call and redirect the control flow to our customized code (i.e., hook), which can manipulate the result of the original call. With this technique, instead of creating several deceptive resources on an end host, we intercept all function calls to access the resources in the SCARECROW deception execution environment and return the manipulated values to the caller. For example, evasive malware may call Windows API function `RegOpenKeyEx()` [13] to check the existence of the Virtual Box registry key `HKEY_LOCAL_MACHINE\SOFTWARE\Oracle\Virtual Box Guest Additions`. Since this registry key matches with one of the SCARECROW deceptive resources, we immediately return `SUCCESS` to the caller to deceive evasive malware that `VirtualBox Guest Additions` exists.

There are various hooking techniques to intercept function calls from the user-level to the kernel-level. For example, we can intercept the execution chain either inside the user process,

Original function	Hooked function
function_a:	function_a:
0x601000: **mov edi, edi**	0x601000: **jmp function_b**
0x601002: **push ebp**	0x601005: sub esp, 10
0x601003: **move ebp, esp**	*… snip …*
0x601005: sub esp, 10	
… snip …	

```
int check_hook (DWORD * dwAddress)
{
  BYTE *add = (BYTE *) dwAddress;
  return (*add == 0x8b) && (*(add+1) == 0xff)? FALSE: TRUE;
}
```

Fig. 1. in-line hooking and its detection

e.g., one or more parts of the Windows APIs, or inside the Windows kernel, e.g., the interrupt descriptor table (IDT) or the system service dispatch table (SSDT). In our implementation, we leverage user-level *in-line hooking* because it is efficient, less intrusive, and for many evasive malware samples, the sheer presence of such in-line hooking already makes the environment *analysis-like*. Specifically, in-line hooking overwrites the first few bytes of the target function with `JMP` instruction to the hook code. Figure 1 shows an example of in-line hooking and its detection method by checking if the first two bytes are intact (e.g., `mov edi,edi`). Since many analysis systems utilize such hooking techniques, the detection of the in-line hooking makes evasive malware believe that it is being monitored, which is the design goal of SCARECROW. Furthermore, the presence of SCARECROW does not guarantee that it is an end-user execution environment because SCARECROW can also be deployed in a sandbox environment. We further discuss this in Section VI. We hook 29 APIs that access SCARECROW deceptive resources to dynamically construct deceptive execution environments for evasive malware.

In order to minimize potential impacts on benign programs, it is preferable that SCARECROW is only visible to suspicious target programs, e.g., newly downloaded programs from the Internet, and E-mail attachments. We employ DLL code injection techniques to plant our hooking in the target process. A DLL code injection technique is to inject a DLL into the same memory space of the target process, and then to have it executed as a part of the target process. Specifically, we wrap our hooking code into a DLL and load the DLL into the memory space of the target process. Then, our hooks are only loaded into the address space of the specific target process, so they can only be called within the target process. We use EasyHook [9] to inject the DLL into the target process.

B. Deployment Framework

Figure 2 shows the overview of SCARECROW deployment framework. The framework consists of SCARECROW controller (`scarecrow.exe`) and SCARECROW library (`scarecrow.dll`). SCARECROW controller starts executing

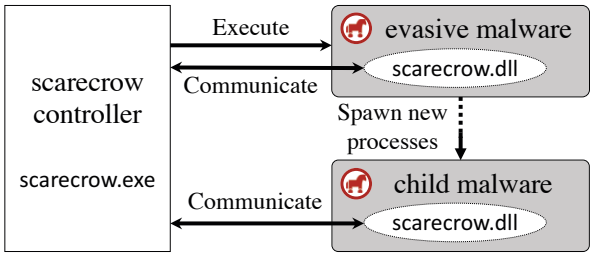

Fig. 2. SCARECROW deployment framework

the target program and injects `scarecrow.dll` into the target process. The DLL installs the API hooks and exchanges runtime information between the target process and the controller. Specifically, `scarecrow.dll` communicates with `scarecrow.exe` through interprocess communication (IPC) channels when a deceptive execution environment is fingerprinted by evasive malware. SCARECROW controller dynamically updates the hooks and configurations through IPC.

We use SCARECROW controller to initiate the execution of the target program because evasive malware may check the parent process of itself to detect an analysis environment. In the normal execution on an end-user host, the parent of the running process executed via double clicks is typically `explorer.exe`. We note that sandboxes often utilize an analysis daemon to run malware. The parent process of malware becomes the analysis daemon process. Using SCARECROW controller to execute a target program is to mimic the starting procedure commonly used in sandboxes and to deceive evasive malware in case malware checks the parent process. During the execution of a target program, SCARECROW dynamically reroutes the hooked API calls to the customized functions in `scarecrow.dll`, which inspects the call parameters and returns values. The return values are manipulated before returning to the caller if any resources in SCARECROW deceptive execution environment are queried. Since the target program may spawn new processes during the execution, `scarecrow.dll` hooks `CreateProcess` call and the functions related to deceptive resources. We suspend the running thread of the new process to inject `scarecrow.dll` into the address space of the new process and then resume it. This allows SCARECROW to work for the descendants of the target program.

IV. EVALUATION

A. Data Sources

For our evaluation, we collected evasive malware samples from two sources and benign programs from CNET [7].

Joe Security ($\mathcal{M}_{JS}$): Joe Security released its detailed analysis reports of 18 evasive malware samples [4], of which we collected all 13 Portable Executable (PE) evasive samples. Non-PE samples are excluded in $\mathcal{M}_{JS}$.

MalGene ($\mathcal{M}_{MG}$): We obtained 1,054 evasive malware samples from the authors of MalGene [25]. These evasive samples were collected from malware feed received by Anubis and were confirmed evasive malware based on the observation that they exhibited different runtime behaviors when running in different analysis environments, e.g., evading one environment while performing malicious activities in another environment.

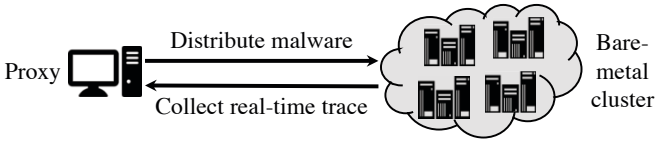

Fig. 3. Experiment environment

CNET Software ($\mathcal{B}_{CNET}$): In order to evaluate the potential impacts of SCARECROW on benign software, we also downloaded the top 20 most popular Windows programs from CNET.

B. Experiment Environment

Figure 3 depicts our experiment environment. The cluster consists of multiple bare-metal Windows 7 machines, each of which is reset to the clean state via Deep Freeze [8] before the execution of a malware sample. We did not conduct experiments on virtual machines since the majority of evasive malware samples were equipped with anti-VM techniques. We set up a python agent on each machine, which connected to the proxy to retrieve a new malware sample and a configuration (e.g., with or without SCARECROW) once the machine restarts. Then the agent ran the malware sample for one minute and rebooted/reset the machine. We used Fibratus [11] to trace Windows kernel activities, including process/thread creation and termination, file system I/O, registry, network activity, DLL loading/unloading and so on. All the activities were uploaded to the proxy in real time to avoid possible corruption of runtime traces.

C. Effectiveness of SCARECROW

1) Effectiveness of Defending against Evasive Malware: We first evaluated SCARECROW with the evasive malware samples collected from Joe Security ($\mathcal{M}_{JS}$), from which we manually constructed the ground truth of malicious behaviors and verified the effectiveness of SCARECROW. We executed each sample in both environments with and without SCARECROW enabled at about the same time (within one minute) to minimize other external factors that might affect the execution of samples. Table I shows the evaluation results on 13 evasive malware samples of $\mathcal{M}_{JS}$. The second column (without SCARECROW) lists examples of malicious behaviors we observed, such as process creation, process injection, and file system encryption, while running the malware samples on a bare-metal machine without SCARECROW. The next column (with SCARECROW) includes the results with SCARECROW planted. The fourth column describes the first triggers in the malware samples trying to identify analysis environments that were reported by SCARECROW. The last

column shows whether SCARECROW successfully deactivated the corresponding evasive malware.

Among 13 evasive malware samples, SCARECROW successfully deactivated 12 of them — the malware terminated without harm or executed non-malicious activities continuously (e.g., benign logic or sleep). Most evasive malware used multiple evasive techniques. For example, sample `9437eabf2fe5d32101e3fbf9f6027880` looked for multiple registry entries (e.g., `SOFTWARE\VMware,Inc.\VMware Tools`, `SYSTEM\CurrentControlSet\Enum\IDE`) through the system call `NtOpenKeyEx`, and files (e.g., `vmmouse.sys`, `vmhgfs.sys`, and `vboxmouse.sys`) using `NtQueryAttributesFile` system call. It was deactivated by just one deceptive resource. Sample `f504ef6e9a269e354de802872dc5e209` presented interesting behaviors. If we normally executed the sample, nothing showed up on a screen; however, it created two daemon processes: `FB_473.tmp.exe` and `FB_5DB.tmp.exe`. On the other hand, if we ran the sample with SCARECROW installed, the sample started a Windows form application without creating `*.tmp.exe`. By offering a Windows form application, the sample tried to exhibit benign behaviors. This means SCARECROW successfully deactivated its malicious behaviors and forced it to execute its benign components. SCARECROW failed to deactivate sample `cbdda646a20d95f078393506ecdc0796` which looked up Process Environment Block (PEB) to obtain `NumberOfProcessors` and terminated if it was smaller than 2. Since it accessed a memory to identify analysis environments instead of calling APIs, SCARECROW currently failed to deceive the evasive logic.

To evaluate the potential impacts on benign software, we tested SCARECROW on $\mathcal{B}_{CNET}$ to see if SCARECROW interfered with benign program execution.

We closely monitored the execution of the programs with SCARECROW installed. All of these software programs installed and operated without any issues. We acknowledge that we did not exhaustively explore all possible program paths in these benign software instances during our manual investigation and it is hard to quantify the impacts on benign software; however, most of SCARECROW deceptive resources were not required for or did not interfere with the execution of the programs, i.e., little or no impact to the benign programs. Hardware resources were typically queried only during the installation step where SCARECROW did not disrupt any benign software.

We further evaluated SCARECROW with a large evasive malware dataset $\mathcal{M}_{MG}$. We automatically ran these evasive samples on the experiment environment as shown in Figure 3. Although these samples were verified as evasive malware samples based on their different runtime traces on different analysis environments, detailed analysis results of their malicious behaviors were not available. Therefore, it was challenging to construct the ground truth of malicious behaviors to evaluate the effectiveness of SCARECROW.

We leveraged the observation from our empirical analysis to characterize the successful deactivation of evasive malware. We often observed self-spawning activities, which typically happened when evasive malware detected the presence of a debugger, spawned a new process to avoid being analyzed, and the new process continuously executed its malicious activities. This might allow evasive malware to bypass a debugger; however, in SCARECROW deceptive execution environment, such evasive logic had different consequences. For example, if malware checked for the presence of a debugger by using `IsDebuggerPresent()` API, SCARECROW returned true; then malware spawned itself and the new process also checked the debugging environment through `IsDebuggerPresent()` API for which SCARECROW returned true again. As a result, such evasive malware samples kept spawning new processes continuously. We considered this everlasting loop did not reach the code beyond the evasive logic and SCARECROW deactivated evasive malware.

We checked the traces with SCARECROW installed and found 823 (78.08 %) of evasive malware samples spawned itself more than 10 times. For example, sample `0827287d255f9711275e10bda5bda8c2` repeatedly spawned itself using system call `CreateProcessW` for 474 times in a minute and kept fingerprinting SCARECROW deceptive resources via function call `IsDebuggerPresent()`. In fact, we found `IsDebuggerPresent()` was the most commonly used method by evasive malware to identify analysis environments. 815 out of 823 malware samples spawning constantly themselves invoked `IsDebuggerPresent()`. To further verify whether such repeating self-spawning behavior will actually deactivate the malware samples, we first manually analyzed the behavior of randomly-chosen 10 samples. All of these samples only spawned new processes of themselves when SCARECROW was enabled. Without SCARECROW, however, the samples exhibited malicious behavior, such as creating malicious processes and modifying registries. We further examined the traces of other self-spawning samples and confirmed that all of them only created new processes of themselves without causing system changes. Therefore, we concluded that the malicious activities of such self-spawning samples were successfully deactivated by SCARECROW. We note that a self-spawning loop caused by SCARECROW might lead to fork bombs or high CPU usage on end-user systems. We currently only record such self-spawning loop behavior and raise an alarm without any interruptions; however, we can easily stop those samples with self-spawning loop behavior.

We also compared the trace generated *with* and *without* SCARECROW. We examined if there were any significant activities, such as creating new processes, writing files, and modifying registries, in the trace without SCARECROW but not in the trace with SCARECROW. If so, we considered that SCARECROW deactivated evasive malware.

In total, 944 (89.56 %) evasive samples were successfully deactivated by SCARECROW. Figure 4 depicts the evaluation results of top 10 malware families out of 61 malware families

TABLE I
EFFECTIVENESS (EFF.) OF SCARECROW ($\mathcal{M}_{JS}$)

Samples	Without SCARECROW	With SCARECROW	Trigger	Eff.
9fac72a	install a fake AV	terminate w/o installation	GlobalMemoryStatusEx()	✓
d80e956	create svchost.exe	sleep loop w/o injection	GetModuleHandleA()	✓
0af4ef5	create ffpp41.exe, scservices.exe	exit w/o creating executables	Hook detection	✓
3616a11	delete itself and create hh.exe	spawn, terminate w/o creating hh.exe	IsDebuggerPresent()	✓
f504ef6	create yfoye.exe	open winform w/o creating yfoye.exe	IsDebuggerPresent()	✓
cbdda64	create a copy of itself	create a copy of itself	N/A	✗
9437eab	bot dropper	exit after VM detection fails	NtQueryValueKey()	✓
40d19fb	create *.tmp.exe	sleep loop w/o creating *.tmp.exe	IsDebuggerPresent()	✓
ad0d7d0	create svchost.exe	terminate itself	GetTickCount()	✓
06a4059	create svchost.exe	terminate itself	NtQuerySystemInformation()	✓
f1a1288	create taskhost.exe	terminate w/o injection	IsDebuggerPresent()	✓
61f847b	encrypt file systems	keep sleeping	IsDebuggerPresent()	✓
564ac87	force restart immediately	keep sleeping	The name of malware	✓

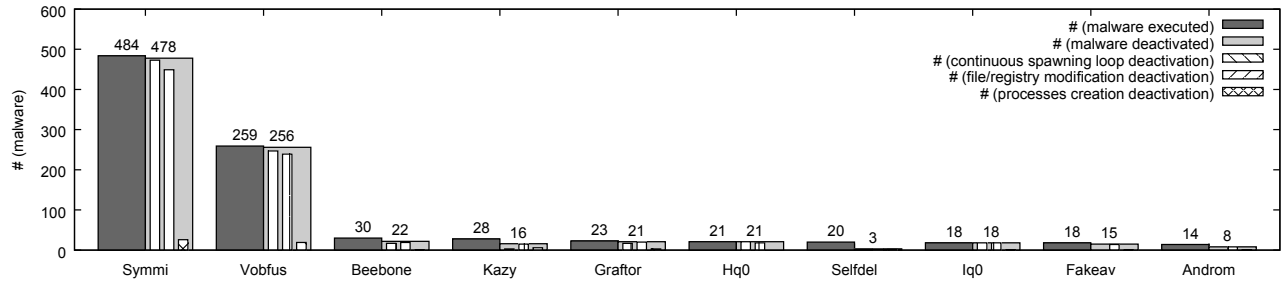

Fig. 4. Effectiveness of SCARECROW on $\mathcal{M}_{MG}$

in $\mathcal{M}_{MG}$. The x-axis denotes malware families[2], and y-axis denotes the number of samples. The first bar in each category represents the total number of the malware sample in the category. The second bar shows the number of the successfully deactivated malware samples. The successful deactivation is the union of the fully deactivated samples, deactivated samples creating processes, and deactivated samples modifying files/registries. For example, there were 484 samples in Symmi. SCARECROW successfully deactivated 478 (98.7 %) among which 473 samples kept spawning itself, 26 samples created new processes without deploying SCARECROW, and 449 samples modified files/registries without deploying SCARECROW. SCARECROW deactivated evasive malware in most families except Selfdel. Most of the samples in Selfdel automatically deleted and terminated itself promptly even without SCARECROW. It was not straightforward to determine the effectiveness of SCARECROW on such samples without observing any critical activities.

2) Effectiveness of Deceiving Existing Fingerprinting Solutions: To further assess the effectiveness of SCARECROW, we also evaluated it against two state-of-the-art analysis environment fingerprinting techniques: Pafish [2] (also known as Paranoid Fish) and wear-and-tear artifacts-based evasion [29].

Pafish [2] employs several fingerprinting techniques to detect analysis environments in the same way as malware does. It explores 54 pieces of evidence from 11 categories to identify analysis environments, including debuggers, sandboxes, virtual machines, and hooks. We executed Pafish on three different environments: (a) a bare-metal sandbox which is the same bare-metal machine used in the previous evaluation, (b) a virtual machine-based sandbox environment where we installed Cuckoo sandbox v2.0.3[3] on top of a VirtualBox virtual machine running Windows 7, and (c) a real end-user machine running Windows 7 which has been actively used and VMware is installed due to work requirements.

As shown in Table II, in the bare-metal sandbox without SCARECROW enabled, only one evidence (`mouse activity`) was triggered. Feature `mouse activity` checks the mouse activities during the 2 seconds of sleep. Since there was no mouse activity on our bare-metal sandbox during the execution, `mouse activity` was triggered. On the virtual machine sandbox, 3 pieces of evidence (`rdtsc_diff_vm exit`, `cpuid_hv_bit`, and `cpu_known_vm_vendors`) in CPU groups were triggered. Feature `rdtsc_diff_vmexit` executes `CPUID` between two `RDTSC` instructions. It exploits the fact that the hypervisor will cause a trap when executing `CPUID` and change context from the VM to the hypervisor, which severely delays the operation. Feature `cpuid_hv_bit` checks the bit 31 of `ECX` of `CPUID`, and exploits the fact that Intel and AMD CPUs have

[2]The malware families are presented based on Symantec labels from VirusTotal.

[3]We installed Cuckoo as it is one of the most popular sandboxes, which is used in both Malwr [12] and VirusTotal [14].

TABLE II
EFFECTIVENESS OF SCARECROW AGAINST PAFISH [2]: THE NUMBERS REPORTED INDICATE THE NUMBER OF EVIDENCE FEATURES TRIGGERED IN EACH CATEGORY.

Feature Categories	Bare-metal sandbox		Virtual machine sandbox		End-user machine	
(# of features)	w/ SCARECROW	w/o SCARECROW	w/ SCARECROW	w/o SCARECROW	w/ SCARECROW	w/o SCARECROW
Debuggers (1)	1	0	1	0	1	0
CPU information(4)	0	0	0	3	1	1
Generic sandbox (12)	10	1	9	3	9	1
Hook (2)	2	0	2	1	2	0
Sandboxie (1)	1	0	1	0	1	0
Wine (2)	2	0	2	0	2	0
VirtualBox (17)	14	0	14	16	14	0
VMware (8)	4	0	4	0	4	1
Qemu detection (3)	1	0	1	0	1	0
Bochs (3)	1	0	1	0	1	0
Cuckoo (3)	0	0	0	0	0	0

reserved bit 31 of `ECX` of `CPUID` leaf `0x1` as the hypervisor present bit. Hypervisors set this bit to one while physical CPUs set this bit to zero. Therefore, malware running in the virtual machine can test bit 31 to detect if it is running inside a virtual machine. Feature `cpu_known_vm_vendors` calls CPUID with `EAX=40000000` as an input, and the virtualization vendor string will be returned in `EAX`, `ECX`, and `EDX`. Through checking the known vendor strings, malware can easily decide whether it is running in a virtual environment. However, `cpuid_hv_bit` and `cpu_known_vm_vendors` are not reliable evidence since they can be easily manipulated. Both VMware and VirtualBox allow modifying the `CPUID` instruction results to generate customized outputs for the `CPUID` instruction to the guest. The evidence (`ShellExecuteExW`) in Hook group was triggered since Cuckoo sandbox hooks `ShellExecuteExW` API. 16 features in VirtualBox group were hit since we set up a virtual machine-based sandbox on top of VirtualBox. On the end-user system, the evidence (`rdtsc_diff_vmexit`) from the CPU group was triggered, and the evidence (`mouse activity`) from the Generic sandbox was also triggered since there was no mouse movement during the execution of Pafish. The evidence (`VMware device`) was triggered because VMware was installed on the tested user machine.

Overall, Pafish [2] performed well to detect a virtual machine-based sandbox; however, it was not able to detect a bare-metal sandbox. Furthermore, timing-based attacks were not reliable since `rdtsc_diff` and `rdtsc_diff_vmexit` were triggered in both the end-user system and the virtual machine sandbox, which should have been triggered only in a virtual machine sandbox.

We then enabled SCARECROW on all of the three execution environments. We also modified `CPUID` instruction results and updated the MAC address of the Cuckoo sandbox to make it more transparent to evasive malware. As a result, most of the features were triggered, which demonstrates the comprehensive coverage of SCARECROW. Some of the missed features were related to CPU timing information (e.g., `rdtsc_diff_vmexit` and `rdtsc_diff`), which was not handled by the current implementation of SCARECROW. However, as we observed before, such timing attacks are not reliable methods for evasive malware to detect analysis environments. Some missed features were not supported due to the unsupported system versions. For example, `IsNativeVhdBoot` was only supported in Windows 8 while Windows 7 was used for our evaluation. We note that, with SCARECROW enabled, all the three execution environments became *indistinguishable* for evasive malware as well as Pafish by demonstrating similar analysis environment resources.

Wear-and-tear artifacts-based evasion [29] is the most recent work to identify analysis environments based on the system usage. The key idea was that existing dynamic analysis systems were typically implemented using operating system images in an almost pristine condition while real devices usually under active use. Therefore, wear-and-tear artifacts can potentially be used by malware as indicators of the extent to which a system has been actually used. Miramirkhani et al. [29] proposed 44 wear-and-tear artifacts from 5 categories to characterize the "aging" patterns of an active end-user system and built a decision tree model to identify an analysis environment. Unlike Pafish [2], the proposed method could detect a bare-metal based sandbox since a bare-metal sandbox might also contain only a few wear-and-tear artifacts.

In order to evaluate the effectiveness of SCARECROW against [29], we further extended SCARECROW to support "aging"-related deceptive resources. Instead of deceiving all the 44 artifacts, we incorporated the top 5 artifacts plus all the artifacts from the largest category (registry category) from [29]. Table III describes the wear-and-tear artifacts we faked and the system call/API related to those resources. For example, artifact `sysevt` calculates the number of system events, and SCARECROW hooked `EvtNext()` API and only returned the top 8000 system events. For feature `deviceClsCount`, which calculates the number of the previously connected USB devices, SCARECROW hooked system calls, e.g., `NtOpenKeyEx` and `NtQueryKey`, and modified the value entries to 29. All these faked values of wear-and-tear artifacts were chosen based on the statistics of sandbox envi-

TABLE III
WEAR-AND-TEAR ARTIFACTS FAKED BY SCARECROW

Categories	Artifacts	Faked Resources	Associated APIs
Top 5	dnscacheEntries	Recent 4 entries	DnsGetCacheDataTable()
	sysevt	Recent 8K system events	EvtNext()
	syssrc	Number of sources in recent 8k events	EvtNext()
	deviceClsCount	`System\CurrentControlSet\Control\DeviceClasses` (29 subkeys)	NtOpenKeyEx(),NtQueryKey()
	autoRunCount	`Software\Microsoft\Windows\CurrentVersion\Run` (3 value entries)	NtOpenKeyEx(),NtQueryKey()
Registry related	regSize	SystemRegistryQuotaInformation 53M (bytes)	NtQuerySystemInformation()
	uninstallCount	`Software\Microsoft\Windows\CurrentVersion\Uninstall`	NtOpenKeyEx(),NtQueryKey()
	totalSharedDlls	`Software\Microsoft\Windows\CurrentVersion\SharedDlls`	NtOpenKeyEx(),NtQueryKey()
	totalAppPaths	`Software\Microsoft\Windows\CurrentVersion\AppPath`	NtOpenKeyEx(),NtQueryKey()
	totalActiveSetup	`Software\Microsoft\ActiveSetup\InstalledComponents`	NtOpenKeyEx(),NtQueryKey()
	totalMissingDlls	`Software\Microsoft\Windows\CurrentVersion\SharedDlls`	NtOpenKeyEx(),NtQueryKey(), NtCreateFile()
	usrassistCount	`Software\Microsoft\Windows\CurrentVersion\Explorer\UserAssist`	NtOpenKeyEx(),NtQueryKey()
	shimCacheCount	`SYSTEM\CurrentControlSet\Control\SessionManager\AppCompatCache`	NtOpenKeyEx(),NtQueryValueKey()
	MUICacheEntries	`Software\Classes\LocalSettings\Software\Microsoft\Windows\Shell\Muicache`	NtOpenKeyEx(),NtQueryKey()
	FireruleCount()	`SYSTEM\ControlSet001\services\SharedAccess\Parameters\FirewallPolicy\FirewallRules`	NtOpenKeyEx(),NtQueryKey()
	USBStorCount	`SYSTEM\CurrentControlSet\Services\UsbStor`	NtOpenKeyEx(),NtQueryKey()

ronments from [29]. Based on the results from [29], the top 5 artifacts were the most effective artifacts and were used by all of their decision trees. Our evaluation shows that SCARECROW was able to steer the values of those top 5 artifacts plus the artifacts from the largest registry category, which would largely affect the analysis environment fingerprinting decisions. We also manually verified that SCARECROW extended with deceptive wear-and-tear resources still had no impacts on the execution of benign software.

V. CASE STUDY

We discuss two case studies to demonstrate the capability of SCARECROW against comprehensive evasive logic and ransomware.

Case I: Comprehensive Evasive Logic

Malware instance (`md5:de1af0e97e94859d372be7fcf3a5daa5`) belongs to malware category `Worm:Win32/Kasidet.B` according to the AV report from Microsoft. In order to evade an analysis environment, malware used a combination of more than 10 evasive techniques. Essentially its evasive logic was a logical disjunction $\mathbb{D}$ of multiple propositions p_i, i.e., $\mathbb{D} = p_1 \vee p_2 \vee ... \vee p_i$. Concretely, p_i referred to the existence of a virtual machine (e.g., checking `VMwareTools` and `VirtualBox Guest Additions`), debugging frameworks (e.g., `IsDebuggerPresent()`), and sandboxes (e.g., `sandbox folder`) by checking software and hardware resources. If any of these checks detected a dynamic analysis environment (i.e., p_i is `True`), then malware terminated its process. To analyze the malware sample, existing sandbox systems need to mask *all* resources that are checked by the sample. Missing a single check may allow evasion.

However, in SCARECROW, we negate the logical disjunction $\mathbb{D}$, i.e., $\neg\mathbb{D} = \neg(p_1 \vee p_2 \vee ... \vee p_i) = \neg p_1 \wedge \neg p_2 \wedge ... \wedge \neg p_i$, which indicates that we only need one p_i to be `True` to deactivate evasive malware in end-user environments. For example, we simply returned `True` for the function call `IsDebuggerPresent()`) to stop the malicious behaviors of the malware sample. Therefore, one of the major advantages of SCARECROW is that we do not need to handle all evasive logic. By satisfying a small subset of conditions, we successfully deactivated a large number of evasive malware samples.

Case II: Deactivating Ransomware

Ransomware has been identified by the U.S. Department of Justice as the "biggest cyberthreat" of 2017 [3]. Ransomware is malicious software that targets and encrypts data until its owner pays the ransom. It cost businesses a total of $1 billion in 2016. In this section, we demonstrate how SCARECROW deactivated two well-known ransomware: WannaCry and Locky.

Sample (`md5:db349b97c37d22f5ea1d1841e3c89eb4`) is the latest variant of WannaCry ransomware cryptoworm, which has been observed infecting Windows computers world-

wide since May 12, 2017. Once infected, WannaCry malware encrypts the files into the `.WCRY` extension, and victims need to pay around $300 or $600 (via Bitcoin) to decrypt the data. The initial version of WannaCry does not have evasive logic and immediately encrypt the system. To avoid analysis in a sandbox, the new variant of WannaCry started to equip with a network-based evasion technique. Specifically, it uses a hard-coded non-existent domain. If it successfully gets an HTTP response from the non-existent domain, it exits without encrypting the data. This evasive technique is based on the fact that many sandboxes use a DNS sinkhole to resolve non-existent domains into controlled IP addresses. On a normal end-user machine, however, malware cannot get an IP address for a non-existent domain and will encrypt the user's data.

We ran this new variant sample with SCARECROW installed. Based on our configuration, all the non-existent domains will be resolved to the same IP address of our proxy. We successfully deceived WannaCry ransomware that it was being sinkholed in a dynamic analysis environment. Since we only manipulate the resolution of non-existent domains, SCARECROW has no impact on the behavior of benign software, which does not rely on connecting to non-existent domains. Evasive techniques have a limited applicable scope and malware authors typically enhance the evasive logic by adding more new checks while keeping existing checks to be backward compatible. As we observe in recent WannaCry ransomware, although the DNS-based evasion technique is a well-known evasive technique, it is still being adopted by emerging malware variants.

VI. DISCUSSION

A. Limitations

Evasive malware may exploit timing channels to detect the analysis environment. However, such timing channels are not reliable and may trigger both false positives and false negatives. Malware author typically combines the timing feature with other reliable and deterministic detection techniques together in the evasive logic. Based on the results from previous work [25], around 30% of evasive malware samples in our dataset explore the cumulative timing of system calls for evasion. However, we found that most of these samples also explored other evasive techniques, which SCARECROW used to deactivate them. During our evaluation, we found some malware can directly read from memory without using APIs to fingerprint the running system. Evasive malware may also bypass our current hooks by directly invoking Native Windows APIs. In the future, we plan to extend SCARECROW with kernel/hypervisor-based hooking. In theory, malware can use any evasive techniques and system resources to fingerprint the running system. However, in practice, most evasive techniques have been standardized and modularized. Malware often reuses and combines them to increase the chance to evade sandboxes. Thus, SCARECROW is able to deactivate a spectrum of evasive malware by covering the commonly used resources. In addition, as we discussed in II-C, some research such as [25] can also help to discover new evasive logic.

In summary, we admit that SCARECROW is not a silver bullet that can address all future evasive malware via deceptive resource fingerprinting, however, as a defender, SCARECROW has unique advantages by shifting risks to the attackers.

B. Detection of SCARECROW

Once the malware authors are aware of SCARECROW, they may detect the existence of SCARECROW and conduct malicious behaviors on systems with SCARECROW available. Because SCARECROW integrates multiple deceptive analysis environment resources together to guarantee the coverage of most evasive logic, the best way to detect SCARECROW is to check conflicting resources. For example, malware can check whether the underlying system bestows multiple VM features from different vendors, e.g., VMware and VirtualBox. This could be considered impossible because neither a production nor an analysis environment could belong to multiple VMs simultaneously. One possible solution is to prepare multiple profiles for different sandbox environments in SCARECROW. If one property of any individual profile is triggered, we can disable all other profiles immediately to avoid from being detected. We will leave this as our future work.

C. Active Mitigation

SCARECROW deactivates evasive malware by deceiving an *analysis-like* environment. However, evasive malware may perform unexpected behaviors in face of SCARECROW, which could affect an end-user system. For example, evasive malware samples might create a self-spawning loop and lead to fork bombs or high CPU usage as discussed in Section IV-C. While this is still more desirable than an actual infection of machines, it may affect overall system usages. Currently, we only passively record all the malware behaviors with SCARECROW enabled without any interruptions. Since such fork bomb behaviors already can be considered as suspicious program execution and could be further mitigated by killing its parent processes or directly blocking forking process.

VII. RELATED WORK

We summarize and discuss evasive malware analysis and defense techniques with respect to their execution environments. In Figure 5, we visualize the execution environment space with three bases: *virtualization and monitoring tools*, *wear-and-tear artifacts*, and *hardware diversity*. Approaches to make analysis environments more transparent can be represented as an arrow from the bottom right to the top left shown in Figure 5.

A. Transparent Sandbox

To evade fingerprinting from the recent emerging evasive malware samples, researchers started building transparent analysis systems to perform more stealthy analysis. Vasudevan et al. [32] proposed the first analysis system Cobra to defeat anti-debugging techniques. Dinaburg et al. [19], Jiang et al. [23], and Gu et al. [22] utilized the out-of-VM approach for monitoring and analysis. Although these systems can conceal their tracing and analysis, they can still be detected by fingerprinting

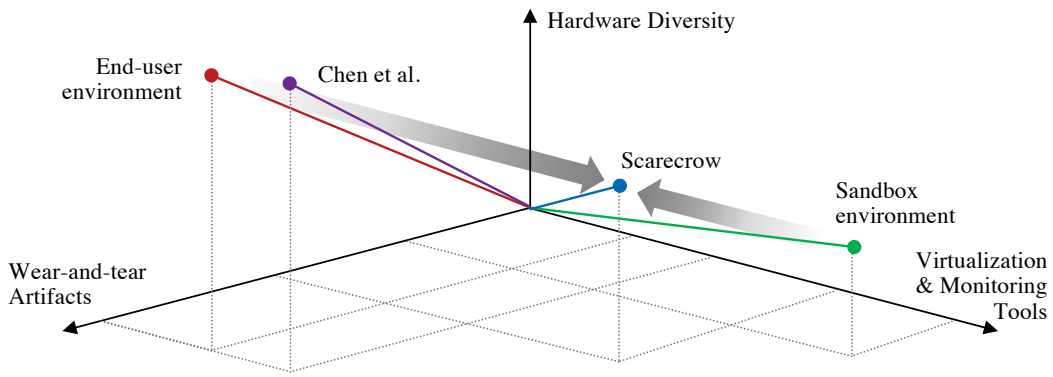

Fig. 5. Different execution environments

the virtual machines they built on. Lots of techniques have been proposed to detect VMware and QEMU [15, 20]. Pek et al. [31] also introduced a timing-based detection mechanism to detect hardware-assisted virtualization-based systems [19]. To further evade possible detection of virtualization and emulation based systems, Kirat et al. [26] proposed to use bare-metal machines to analyze malware. It explored a fast and rebootless system restore technique so that it was robust to VM/emulation-based detection attacks. Recently, Yokoyama et al. developed SandPrint [35], a sandbox fingerprinting tool to collect fingerprints from public online sandboxes and could even detect bare-metal sandboxes. Miramirkhani et al. [29] proposed a novel sandbox evasion technique that exploited the history and user activity information. In this way, it could effectively evade sandboxes without instrumentation indicators.

B. Evasive Logic Discovery

Kang et al. [24] proposed a scalable trace-matching algorithm to locate the point of execution diversion between two executions. The system dynamically modifies the execution of the whole-system emulator to defeat anti-emulation checks. Balzarotti et al. [15] designed a system to detect dynamic behavior deviation of malware by comparing behaviors between an instrumented environment and a reference host. Kirat et al. [27] designed a system to detect evasive malware by executing them on a bare-metal system and compared their behaviors when executing on other emulation and virtualization-based analysis systems. They further proposed an algorithm to automatically extract analysis evasion signatures [25].

C. Evasive Malware Defense

Chen et al. [18] proposed a taxonomy of evasion techniques used by malware against dynamic analysis systems and proposed a technique to deter evasive malware by imitating analysis systems. However, evasive malware has become much more diversified, and the limited scope and methods in [18] (only for anti-virtualization and anti-debugging malware) are not sufficient to cover the latest technical advancement in evasive malware. Wichmann et al. [33] introduced the concept of using malware infection markers to vaccinate systems against infections by a specific malware family. Xu et al. [34] proposed a method to stop malware execution by faking environment-related resources. However, these two systems mainly explored malware specific resources. If the malware fingerprints analysis environment, it cannot generate resources.

SCARECROW transforms an end-user environment into a sandbox-like environment to deter evasive malware as represented as an arrow from the top left to the bottom right In Figure 5. Compared with Chen et al. [18], which only imitates a few characteristics of virtual machines and debuggers, we systematically explore the resources exploited by evasive malware, such as software, hardware, and network resources, covering virtual machine environments, debugging environments, security products, and sandbox-specific components. SCARECROW provides a more comprehensive view of an indistinguishable "analysis environment" to deactivate the most recent evasive malware.

VIII. CONCLUSION

We observed that increasingly more malware variants integrate evasive logic to counteract malware analysis techniques. Evasive logic is built on the hypothesis that end-user execution environments and malware analysis environments are fundamentally distinguishable; therefore semantic gaps between heterogeneous platforms are exploited to enable environment-aware malware execution. However, from the defender's perspective, by breaking this premise and minimizing the semantic gaps, execution of malware on end-user systems can be prevented by means of the malware's very own evasive logic. In this paper, we introduce SCARECROW, a system designed to deactivate evasive malware on end-user environments. We associate an untrusted application with a deceptive execution environment, which consists of crafted system resources typically available only in analysis environments. Such an analysis-like environment effectively sensitizes malware's evasive logic to turn on the self-destruction mechanism and discontinue exhibiting malicious behaviors.

REFERENCES

[1] Biggest cybersecurity threats in 2016, 2016. http://www.cnbc.com/2015/12/28/biggest-cybersecurity-threats-in-2016.html.

[2] Pafish, 2016. https://github.com/a0rtega/pafish.

[3] 38 of consumers affected by ransomware pay up, 2017. https://www.helpnetsecurity.com/2017/04/18/ransomware-payouts/.

[4] Analysis reports of evasive malware, 2017. https://www.joesecurity.org/joe-sandbox-reports-evasive.

[5] Automated malware analysis - cuckoo sandbox, 2017. https://www.cuckoosandbox.org/.

[6] Cerber starts evading machine learning, 2017. http://blog.trendmicro.com/trendlabs-security-intelligence/cerber-starts-evading-machine-learning/.

[7] Cnet, 2017. http://download.cnet.com/windows/.

[8] Deep freeze, 2017. http://www.faronics.com/products/deep-freeze/.

[9] Easyhook, 2017. http://easyhook.github.io/.

[10] Emulators ineffective for malware detection & more, 2017. http://blog.blackducksoftware.com/top-4-tidbits-emulators-ineffective-for-malware-detection-more.

[11] Fibratus, 2017. https://github.com/rabbitstack/fibratus.

[12] Malwr, 2017. https://malwr.com/.

[13] Regopenkeyex, 2017. https://msdn.microsoft.com/en-us/library/windows/desktop/ms724897(v=vs.85).aspx.

[14] Virustotal, 2017. https://www.virustotal.com/.

[15] Davide Balzarotti1, Marco Cova, Christoph Karlberger, Christopher Kruegel, Engin Kirda, and Giovanni Vigna. Efficient detection of split personalities in malware. In *NDSS*, 2010.

[16] Jeremy Blackthorne, Alexei Bulazel, Andrew Fasano, Patrick Biernat, and Bülent Yener. AVLeak - Fingerprinting Antivirus Emulators through Black-Box Testing. In *USENIX Workshop on Offensive Technologies*, 2016.

[17] Rodrigo Rubira Branco, Gabriel Negreira Barbosa, and Pedro Drimel Neto. Scientific but not academical overview of malware anti-debugging, anti-disassembly and anti-vm technologies. In *BlackHat USA*, 2012.

[18] Xu Chen, Jon Andersen, Z. Morley Mao, Michael Bailey, , and Jose Nazario. Towards an understanding of anti-virtualization and anti-debugging behavior in modern malware. In *DSN*, 2008.

[19] Artem Dinaburg, Paul Royal, Monirul Sharif, and Wenke Lee. Ether: Malware analysis via hardware virtualization extensions. In *ACM CCS*, 2008.

[20] Peter Ferrie. Attacks on virtual machine emulators. In *Symantec Advanced Threat Research*, 2007.

[21] Mariano Graziano, Davide Canali, Leyla Bilge, Andrea Lanzi, and Davide Balzarotti. Needles in a haystack: mining information from public dynamic analysis sandboxes for malware intelligence. In *USENIX Conference on Security Symposium*, 2015.

[22] Zhongshu Gu, Zhui Deng, Dongyan Xu, and Xuxian Jiang. Process implanting: A new active introspection framework for virtualization. In *2011 IEEE 30th International Symposium on Reliable Distributed Systems*, pages 147–156. IEEE, 2011.

[23] Xuxian Jiang and Xinyuan Wang. "out-of-the-box" monitoring of vm-based high-interaction honeypots. In *RAID*, 2007.

[24] Min Gyung Kang, Heng Yin, Steve Hanna, Stephen McCamant, and Dawn Song. Emulating emulation-resistant malware. In *ACM workshop on Virtual machine security*, 2009.

[25] Dhilung Kirat and Giovanni Vigna. Malgene: Automatic extraction of malware analysis evasion signature. In *ACM CCS*, 2015.

[26] Dhilung Kirat, Giovanni Vigna, and Christopher Kruegel. Barebox: efficient malware analysis on bare-metal. In *Annual Computer Security Applications Conference (ACSAC)*, 2011.

[27] Dhilung Kirat, Giovanni Vigna, and Christopher Kruegel. Barecloud: Bare-metal analysis-based evasive malware detection. In *USENIX conference on Security Symposium*, 2014.

[28] llsun You and Kangbin Yim. Malware obfuscation techniques: A brief survey. In *BWCCA*, 2010.

[29] Najmeh Miramirkhani, Mahathi Priya Appini, Nick Nikiforakis, and Michalis Polychronakis. Spotless sandboxes: Evading malware analysis systems using wear-and-tear artifacts. In *IEEE Symposium on Security and Privacy*, 2017.

[30] Yoshihiro Oyama. Trends of anti-analysis operations of malwares observed in API call logs. *Journal of Computer Virology and Hacking Techniques*, 2017.

[31] Gábor Pék, Boldizsár Bencsáth, and Levente Buttyán. nether: in-guest detection of out-of-the-guest malware analyzers. In *EUROSEC*, 2011.

[32] Amit Vasudevan and Ramesh Yerraballi. Cobra: Fine-grained malware analysis using stealth localized-executions. In *IEEE Symposium on Security and Privacy*, 2006.

[33] Andre Wichmann and Elmar Gerhards-Padilla. Autovac: Automatically extracting system resource constraints and generating vaccines for malware immunization. In *IEEE International Conference on Green Computing and Communications*, 2012.

[34] Zhaoyan Xu, Jialong Zhang, Guofei Gu, and Zhiqiang Lin. Autovac: Automatically extracting system resource constraints and generating vaccines for malware immunization. In *IEEE International Conference on Distributed Computing Systems (ICDCS)*, 2013.

[35] Akira Yokoyama, Kou Ishii, Rui Tanabe, Yinmin Papa, Katsunari Yoshioka, Tsutomu Matsumoto, Takahiro Kasama, Daisuke Inoue, Michael Brengel, Michael Backes, and Christian Rossow. Sandprint: Fingerprinting malware sandboxes to provide intelligence for sandbox evasion. In *Research in Attacks, Intrusions, and Defenses (RAID)*, 2016.

[36] Fengwei Zhang, Kevin Leach, Angelos Stavrou, Haining Wang, and Kun Sun. Using Hardware Features for Increased Debugging Transparency. In *IEEE Security & Privacy*, 2015.