Evasive Malware via Identifier Implanting

Rui Tanabe[1(✉)], Wataru Ueno[1], Kou Ishii[1], Katsunari Yoshioka[1], Tsutomu Matsumoto[1], Takahiro Kasama[2], Daisuke Inoue[2], and Christian Rossow[3]

[1] Yokohama National University, YNU, Yokohama, Japan
{tanabe-rui-nv,ueno-wataru-tn,ishii-kou-yf}@ynu.jp,
{yoshioka,tsutomu}@ynu.ac.jp
[2] National Institute of Information and Communications Technology, NICT, Koganei, Japan
{kasama,dai}@nict.go.jp
[3] Center for IT-Security, Privacy, and Accountability, CISPA, Saarland University, Saarbrücken, Germany
rossow@cispa.saarland

Abstract. To cope with the increasing number of malware attacks that organizations face, anti-malware appliances and sandboxes have become an integral security defense. In particular, appliances have become the *de facto* standard in the fight against targeted attacks. Yet recent incidents have demonstrated that malware can effectively detect and thus evade sandboxes, resulting in an ongoing arms race between sandbox developers and malware authors.

We show how attackers can escape this arms race with what we call *customized malware*, i.e., malware that only exposes its malicious behavior on a targeted system. We present a web-based reconnaissance strategy, where an actor leaves marks on the target system such that the customized malware can recognize this particular system in a later stage, and only then exposes its malicious behavior. We propose to *implant* identifiers into the target system, such as unique entries in the browser history, cache, cookies, or the DNS stub resolver cache. We then prototype a customized malware that searches for these implants on the executing environment and denies execution if implants do not exist as expected. This way, sandboxes can be evaded without the need to detect artifacts that witness the existence of sandboxes or a real system environment. Our results show that this prototype remains undetected on commercial malware security appliances, while only exposing its real behavior on the targeted system. To defend against this novel attack, we discuss countermeasures and a responsible disclosure process to allow appliances vendors to prepare for such attacks.

1 Introduction

Malware, in its various forms, is equally a threat to consumers (e.g., banking trojans, ransomware), businesses (e.g., targeted attacks, denial-of-service bots),

C. Giuffrida et al. (Eds.): DIMVA 2018, LNCS 10885, pp. 162–184, 2018.
https://doi.org/10.1007/978-3-319-93411-2_8

and society (e.g., spambots). As a response, malware sandboxes have been widely deployed as a vital component in the fight against malware. Sandboxes help to obtain threat information, such as previously unseen malware, inputs for supervised detection mechanisms, malware C&C servers, targets of banking trojans, intelligence on spreading campaigns, or simply to assist in the manual processes of reverse engineering. In addition, organizations face so many security incidents that vendors use sandboxes as an integral part of malware security appliances. As a consequence, the anti-virus industry and security companies heavily rely on sandboxes to detect the maliciousness of an unknown program under analysis.

Seeing sandboxes' utility to defenders, malware authors naturally have been trying to evade sandbox analyses to fly under the radar. There are various types of sandbox implementations, most prominently sandboxes that use virtual machine technologies or CPU emulators. Whereas, malware may include indicators to detect a sandbox, e.g., using artifacts that show the presence of virtualization solutions. Consequently, sandbox designers have tried to design stealthier sandboxes, such as those hiding virtualization artifacts [21,24] or using bare-metal systems for malware analysis [33,34,54]. However, recent research demonstrates that characteristics other than virtualization hint at sandboxes. For example, Yokoyama *et al.* show that snapshotting features can reveal a sandbox [60], while Najmeh *et al.* search for wear-and-tear artifacts of user systems [42]. This demonstrates an ongoing arms race between sandbox maintainers and attackers.

We show that this cat-and-mouse game can be won by attackers once and for all if they implement malware that is tailored to a specific target system. Instead of malware that distinguishes between sandboxes and actual user systems, we envision *customized malware* that is carefully crafted such that it only reveals its malicious behavior on a specific target system. In fact, malware campaigns that follow a similar idea have already been observed [29]. For example, the Gauss malware computed MD5 hashes over directories, assuming a certain identifier only present on the target system, and using this information to unpack/decrypt itself [6]. The difference between these approaches and ours is that we do not use machine-specific keys that are difficult to exfiltrate before infection. Instead, we aim to *implant* marks in the target system. We propose an automated web-based reconnaissance phase, in which an actor uniquely marks a target system so that it can be reidentified later. After doing so, the customized malware is created such that it hides the malicious payload until it can verify that the previously placed characteristics match with the execution environment. This way, not only sandboxes, but any type of non-targeted systems are subject to evasion, without the need to enter the battle of sandbox detection. Such an attack might become especially attractive for targeted attacks, where attackers aim to compromise single individuals in sophisticated attacks.

We instantiate a reconnaissance strategy tailored towards both heterogeneous and homogeneous environments, i.e., a methodology that even works when most systems in the target environment (including the appliances' sandboxes) share the same configuration. Especially homogeneous environments are a highly-

controlled setting that are common in large organizations that aim to reduce maintenance costs. This way, traditional evasion attacks that aim to detect artifacts of sandboxes are ineffective in such a setting, as the users and the sandbox are cloned from each other. However, with our proposal to implant an identifier into the target system, we show that adversaries can even evade analysis in such homogeneous settings. We show that attackers can place unique and characteristic marks in the browser's history, cache, or cookies, or place delicate traces in the DNS stub resolver cache. Implants are, for example, triggered via email, where a target user is tricked into clicking on an attacker-controlled URL that performs a reconnaissance phase and places an implant.

To prototype the attack, we create customized malware that checks the execution environment and search for certain marks/traces. We experimentally show that this malware becomes active on a previously-explored target system, but remains silent and does not raise security warnings on any malware security appliance. Our results demonstrate that implants escape the battle of sandbox detection and remain fully undetected on modern anti-malware appliances. This provides a new angle to the ongoing arms race and calls for completely novel anti-evasion strategies. We discuss several such defenses, ranging from enhancing the sandbox with alerts upon detection of a reconnaissance phase, to how implants can be reliably destroyed. To allow appliance vendors to prepare for such attacks, we discuss a responsible disclosure process to notify affected vendors.

Summarizing, the contributions of this paper are as follows:

- We propose a stealth reconnaissance strategy that places unique *implants* in the target system. This methodology allows attackers to create target-specific malware for both heterogeneous and homogeneous environments.
- We envision the general concept of *customized malware* that presents a completely new angle in the context of malware sandbox evasion. We instantiate the concept with a web-based reconnaissance strategy and demonstrate perfect evasion of three popular commercial malware security appliances.
- We discuss several defenses and a responsible disclosure process. We inform appliances vendors about this new threat to collaborate on countermeasures.

2 Background

We first describe the terminology that we use throughout the paper. We use the term *sandbox* to refer to a dynamic analysis environment that executes unknown programs to observe their behavior. Sandboxes are extensively leveraged to obtain threat information, such as current campaigns [19], recent C&C servers and traffic patterns [43,49,51], or attack targets [26]. Similarly, sandboxes can be used to group behavior into malware families [16,50], or to identify suspicious behavioral patterns [36].

Sandbox Analysis: To cope with the daily feed of hundreds of thousands of previously-unseen malware samples, sandboxes are highly automated. To scale

the analysis, most sandboxes rely on some form of virtualization. To this end, sandboxes rely on various virtualization techniques such as VMWare [10] and VirtualBox [8] or CPU emulators [3,17]. Cuckoo Sandbox [5] is a popular open-source sandbox. However, many security organizations operate other sandboxes, either choosing from commercial sandboxes or designing their own solution. Virtualization offers the benefit that many virtual machines (VMs) can run in parallel on a single system, each analyzing one piece of malware. Egele *et al.* give a comprehensive overview of known sandboxes [25].

Malware Security Appliances: By now, sandbox analysis is widely used in academia, the anti-virus industry, and finally, also by commercial malware security appliances (henceforth simply "appliances"). Such appliances protect endpoints by analyzing unknown files and inspecting their behavior for suspicious actions. They are frequently deployed at the network layer and are used orthogonally as anti-virus scanners, e.g., to protect endpoints from opening malicious email attachments and malicious file downloads from the Internet. Internally, appliances also use a sandbox to analyze the malware behavior, such as its system interactions or network communication.

Given the importance of preventing malware intrusions, these appliances play an ever-increasing role as security defenses. This motivates malware authors to find ways to detect and evade sandboxes. Typically, malware detects sandboxes by checking the execution environment. For example, Barbosa *et al.* reported that about 80% of evasive malware detects VM environments [13]. By now, many appliances have resilience to such simple sandbox evasion techniques. Still, as we will show, it is fundamentally difficult to prevent targeted attacks that can gather information about the target system prior to compromising the target.

Targeted Attack: Adversaries with an outstanding interest to infiltrate a target will continuously and persistently attack the target using various types of malware in several attack phases. Security vendors have defined this type of attack as Advanced Persistent Threat (APT) [1,2]. The purpose of a targeted attack varies per attack campaign, such as leaking information related to financial and intellectual property, destroying a target's core system, or altering data saved in the target system. Although attacks are complex and can be divided into several phases, it is common that attackers gather pieces of information related to the target user(s) and their setup. This makes it possible to plan an attack scenario in which the targeted malware bypasses existing security measures. New vulnerabilities (zero-day attacks) and human weaknesses (social engineering) are then abused to infect a victim. Once the malware has infected a target system, backdoors or C&C channels are created. Further attacks may abuse this stepping stone to propagate to and monitor other target systems.

3 Sandbox Evasion via Implants

We use this section to explain the general concept of *customized malware* that presents a completely new angle in the battle of sandbox evasion. In our context, customized malware can identify a target-specific system in both heterogeneous and homogeneous environments. We will first explain the attack scenario that follows two phases, starting with a phase to place identifiers into the target system and then using them in the next phase. We then describe how to leave unique and characteristic features in the target system. To prototype the attack, we implement customized malware that checks the execution and search for certain marks/traces. Finally, we test these samples against commercial appliances and see if stealth evasion is possible in practice (Fig. 1).

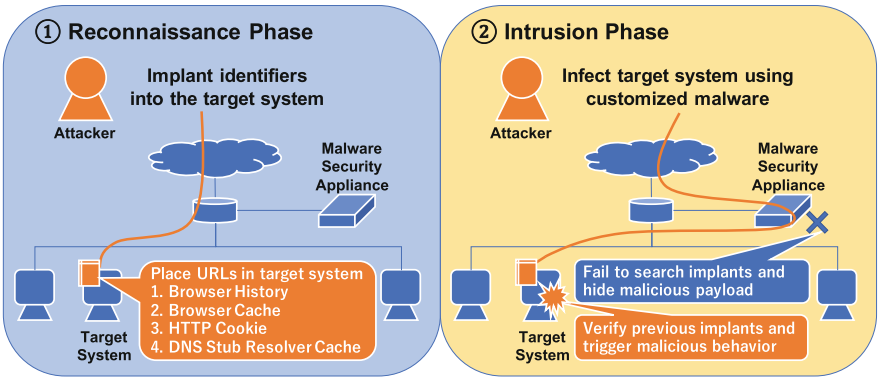

Fig. 1. Overview of our attack scenario. Adversaries first implant identifiers into the target system using an out-of-band channel. We propose to place URLs in the browser history, cache, cookie, and DNS stub resolver cache(①). Attackers then send customized malware that search for the previously placed characteristics, and trigger its malicious behavior only if the features match. This way, not only malware security appliances, but any type of not-targeted systems are evaded(②).

3.1 Attack Scenario

We first explain the threat model that we follow in our work and the assumptions that we make about the capabilities of sandboxes. We start with an overview of how sandboxes are used in practice, and then detail the setting of our new concept of customized malware.

To defend against the ever-increasing number of malware attacks, malware security appliances have become an integral part of many organizations' security strategies. Seeing their prevalence, malware authors put significant effort into detecting (and evading) such sandboxes. For example, malware can deploy detection routines for VM-based or emulation-based sandboxes and change its behavior if a sandbox environment is found. Similarly, malware that evades

sandbox analysis based on the lack of user interaction has been seen. In addition, recent academic works propose further more systematic sandbox detection strategies [42,60]. To cope with the problem, stealthier or even bare-metal sandboxes [7,34] have arisen, and security vendors have entered an ongoing arms race to become resilient against sandbox evasion techniques.

Whereas it seems that this evasion arms race will continue for a while, in this paper, we call attention to the next-stage problem of target-specific malware. That is, we envision *customized malware* that is tailored towards infecting one particular system. To this end, we assume that the customized malware first places an implant into the target system using an out-of-band channel (e.g., triggered via email). Later on, when executed, the customized malware first gathers information on the executed environment, matches these features against the previous implants of the target system, and triggers its malicious activities only if the features match. Regardless of the efforts put into hiding the presence of a sandbox, as long as the sandbox is not an exact copy of the target system, the customized malware will not reveal its normal behavior. That is, our work is fundamentally different from previous approaches in that our key idea is to identify the target system instead of separating sandboxes from normal systems.

Individual malware campaigns have already been observed to follow a similar idea, e.g., by computing MD5 hashes over directories, assuming a certain SID (a unique per-system identifier in Windows), and by comparing specific and current paths to decrypt themselves [6,29]. Chengyu *et al.* show some system unique properties that can be used to obfuscate malware samples [23]. However, these features are difficult to exfiltrate remotely (before infection) and thus pose costs for the attacker or even render such evasions infeasible. We hence revisit this attack pattern and augment it with new techniques. Most importantly, we propose to place unique marks on the target system prior to infection instead of merely using existing ones.

The novel concept of customized malware follows a two-phase approach, starting with the *reconnaissance phase* and then using its results in the *intrusion phase*. This attack scenario is common for targeted attacks such as Advanced Persistent Threats (APTs), in which the attackers aim to infiltrate specific systems after spying on their victims first. In our context, we require that the reconnaissance target and the attack target are the same (specific user). As we will show, this assumption can be easily satisfied, e.g., when both reconnaissance phase and infection phase share the same communication channel (e.g., email or HTTP). We will describe the two phases in the following:

Reconnaissance Phase: In this phase, adversaries first *implant* certain marks on the target system. The Internet has become a necessary technology for daily business, including email communication and web browsing. In fact, obtaining the email address of an attack target is already a vital step for targeted attacks. To perform the reconnaissance, we assume that a targeted email successfully tricks a user into clicking the URL of an attacker-controller reconnaissance web site. This web site will then use a web-based reconnaissance procedure that

implants features to the host, e.g., via email or via web sites. That is, if target users click on an attacker-provided URL in an email, an unique identifier is implanted on the target system. Such an implant can be stealthy, e.g., by leaving certain marks/traces in the browser or system. Later on, customized malware can recognize the target-specific system based on these implants.

During the reconnaissance phase, we assume that sandboxes either do not click on URLs provided in emails (e.g., as this would otherwise cause bad side effects, such as automated unsubscriptions from mailing lists, etc.). Or, if sandboxes indeed follow links provided in emails, we assume that the intrusion and reconnaissance phase would *not* be run within the same execution unit of the sandbox. This follows the reasoning that sandboxes have to differentiate the behavior of different inputs (emails, malware files, etc.) and restore their snapshot after analyzing a particular input. In fact, this relaxation allows us to place implants without even requiring the user to click on URLs. We observed that it is possible to carefully craft emails that automatically load external content from attacker-controlled URLs (e.g., loading embedded external images, as enabled by default in Apple Mail), which then can be used to place implants. In an attempt to collect implants, sandboxes could also automatically open emails and access the URLs, but again, if this happens on a freshly-restored snapshot, both phases would still not be linked. We will motivate this further in Sect. 4.

Intrusion Phase: After the reconnaissance phase, adversaries have sufficient characteristic information to reidentify a specific target. The purpose of the intrusion phase is thus to create stealth customized malware that only executes on the target system, while not triggering suspicious activity on other systems and thereby also evading anti-malware appliances. As we will show, we can even hide the actual malicious payload by encrypting/decrypting it with the characteristic information. That is, also a manual analyst lacking the true characteristics of the actual target system could not reverse engineer the malicious payload.

To assess how stealthy the customized malware is, we assume that the crafted customized malware is tested by any (combination of) appliance(s) and sandbox(es). We further assume that adversaries have no insights on which (if any) sandboxes are deployed, which also makes it difficult to implant features. Although some expert knowledge might be helpful to guess which activity in the intrusion phase might raise an alert in the sandbox, an attacker might use blackbox tests to identify viable strategies that survive sandbox checks.

Hetegerogeneous vs. Homogeneous Target Environments: Targeted attacks are usually easier to perform in heterogeneous environments compared to homogeneous environments, as systems in the heterogeneous setting can be easily distinguished from each other. Yet we propose a methodology for customized malware that also performs well if the target system has identically-configured clones. For example, organizations that deploy preconfigured configurations to their end users to minimize maintenance and license costs and to ease security

management create such a *homogeneous environment*. High-end security appliances adapt to such settings in that their sandbox operates on exact copies of the actual production systems of the organization they aim to protect. In such a setting, traditional fingerprinting methods may fail to distinguish target systems from any other system of the organization—including the sandbox.

3.2 Feature Implantation

We will now describe and evaluate our implant-based methodology to develop customized malware that implicitly evades sandboxes. The key idea of the proposed *implants* is to add unique and characteristic marks into the target system, such that the customized malware can recognize them during the infection stage. An important detail is that the implants should look benign to the target system (and to the sandbox when looking them up), while the customized malware has to be able to query implants. We now consider four non-invasive methods to realize such implants.

Browser History: Web browsers typically record histories of web accesses to ease lookups of visited sites in the future. By sending a unique (not necessarily attacker-controlled) URL to the target and leading him to access it, the URL will be recorded in the log of the target system. Since adversaries only need to implant the URL in the access history, the web site does not have to be malicious, so that no network appliances will be able to detect such implants. As the implant just has to be unique, it could even be a legitimate web site with a unique identifier added within the URL (e.g., `google.com/12345`). Another way to abuse browser history is to use the access date as an implant, if browsers record the last access date per URL.

Browser Cache: Most web browsers cache web content to speed up subsequent visits. By luring the target system into clicking a unique URL, one can place identifiers that are stored in the cache of the target system. The implant could be benign but unique URLs or resources (images, CSS files, etc.). As caches might be refreshed or deleted after a certain time and may thus not last long, attackers would likely aim to shorten the time between the reconnaissance phase and the intrusion phase.

HTTP Cookie: Cookies are a well-known technique for tracking browsers. They are stored on the user's computer as a file and store stateful information that is specific to the pair of a client and a specific web site. Cookies are usually saved when a new web site is loaded and can be queried by the web server or accessed by the client computer. By now, most legitimate web sites use cookies, so their usage is not suspicious. Although they can be destroyed when the current web browser is closed, the lifetime of cookies is configurable (unless overridden by manual browser configurations). Attackers can also use cookies to implant

an identifier in a stealth manner using attacker-controlled web sites. This way, malware authors could search for an implant in cookie databases of browsers.

Supercookies (Evercookies) are similar to cookies in that they are stored on the user's computer. They are usually harder to delete from the device, as they are realized using multiple storage mechanisms [31]. For example, a supercookie can implant a user-specific identifier using a Flash Local Shared Object. Alternatively, web servers can abuse a security improvement function like HSTS or HPKP for supercookies [9]. The data size can be larger than normal cookies, showing that supercookies are suited to track devices and place implants.

DNS Stub Resolver Cache: The Domain Name System (DNS) is widely used to resolve domains to IP addresses, e.g., when accessing web sites. On Windows, by default the Windows DNS stub resolver is used to query domains and cache results according to their lifetime, as specified in the Time-to-Live (TTL) value in DNS responses. Thus by sending a specific URL to the target and leading him to access it, the domain will be cached by the DNS stub resolver. In fact, the domain does not have to lead to a malicious web site, just feature a sufficiently long cache duration (i.e., TTL) to bridge the time between reconnaissance and infection. Malware authors could then check if the resolver cached a particular domain, e.g., using common DNS cache snooping techniques.

Beside implanting features, when it becomes important for malware authors to learn if the target has accessed a specific web site, the situation is trickier. The obvious solution is using an attacker-controlled URL. Alternatively, one could use unique benign web sites that state the date of the most recent visit, deploy a publicly-visible visitor counter, or even abuse a timing-based side channel to infer whether a certain page has been visited. By preparing an attacker-controlled URL, malware authors can also find out which web browser the target is using and make it easier to search for such implants. Therefore, malware authors may carefully send an attacker-controlled URL and effectively implant identifiers into the target system.

3.3 Customized Malware

We envision that customized malware first gathers information of the executed environment and then matches these features against the implants placed previously during the intrusion phase. Most basic, malware authors could simply check if the implant exists, and if so, follow a binary decision to either unpack the malicious payload or not. However, a manual analyst could then still reverse engineer the customized malware, reverse the decision to not unpack, and then obtain the malicious payload. In fact, malware sandboxes and appliances already have similar functionalities to scan malware binaries for malicious payloads, or to execute several branches (multi-path execution [41]).

To strengthen this naïve approach, malware authors can not only check for the existence of an implant. In fact, they can retrieve a value from the implant, which can then be used as decryption key for malicious payloads. This way, it

would be impossible to decrypt the malicious payload even for a human analyst or multi-path execution. Embedding such implant values is trivial for cookies, caches and browser history, where adversaries just need to configure the implant URLs in the reconnaissance phase accordingly. For DNS caches, storing implant values is bit more evolved, but also possible. For example, the value of a cached `AAAA` record (which stores an IPv6 address) could be used as a 128-bit AES key.

3.4 Malware Security Appliance Evasion

Seeing that one can implant identifiers through web-based techniques and implement customized malware, we now test if such implants can be used to evade commercial appliances.

Implementation: We first prepared a legitimate web site, which prototypes an attacker-controlled URL and gives an attacker the highest flexibility. We accessed our web site from a user machine that is used on a daily basis at a real organization (henceforth simply "target"). To compare the results, we accessed the web site from three different web browsers (Chrome, Firefox, and IE) and implanted a URL in the target system's browser history, cache, cookie, and DNS stub resolver cache. We then implemented Windows 32-bit PE programs written in C/C# that use the Windows API, commands, and custom functions to search for URLs that perfectly match our web site. We implemented samples that open the browser history by searching for `preferences` for Chrome, `sessionstore.js` for Firefox, and `%HISTORY%` for IE. Our web site has an image file embedded in the top page, and therefore the samples search for such cached items in `entries` for Firefox and `%TEMPORARY INTERNET FILES%` for IE. We furthermore implemented samples that read a cookie from `Cookies` for Chrome and `cookies.sqlite` for Firefox. To inspect the stub resolver's DNS cache, our prototype uses the Windows `ipconfig` utility and the undocumented Windows API `DnsGetCacheDataTable`.

Evaluation Setup: We first executed the samples on the target host and verified that all implants could be found. We then submitted the samples to three popular appliances from well-known vendors[1] (henceforth simply appliance A, B, and C). We gained access to various sandbox configurations (Windows 10, Windows 7, Windows XP, 32/64 bit, different service packs, etc.) of the appliances, totaling nine distinct sandboxes. Since the appliances did not allow for network communication (which supports our threat model that even collecting implants might be impractical for most sandboxes), we investigated the analysis report which was produced by the appliance after execution. Although the appliances were *not* cloned from actual user machines, this methodology would similarly work in a fully-homogeneous environment.

[1] We omit the vendor names not to pinpoint to weaknesses of individual appliances.

Table 1. Security alerts reported from the three appliances including nine sandboxes (Windows 10, Windows 7, and Windows XP) which executed samples that searched for implants of browser history, cache, cookie, and DNS cache.

Feature		Appliance A	Appliance B	Appliance C
History	Chrome	*Hardware Access*	no alert	no alert
	Firefox	*Hardware Access*	no alert	no alert
	IE	*Hardware Access*	no alert	no alert
Cache	Firefox	*Hardware Access*	no alert	no alert
	IE	*Hardware Access*	no alert	no alert
Cookie	Chrome	*Hardware Access* *Browser Access*	no alert	no alert
	Firefox	*Hardware Access* *Browser Access*	no alert	no alert
DNS		*Hardware Access*	*Directory Access*	no alert

Table 2. Security alerts reported from the three appliances including five sandboxes (Windows 10 and Windows 7) which executed samples that searched for implants of browser history, cache, and DNS cache and decrypts malicious payload when previous implants are found.

Feature		Appliance A	Appliance B	Appliance C
History	Chrome	no alert	no alert	no alert
	Firefox	no alert	no alert	no alert
	IE	no alert	no alert	no alert
Cache	Firefox	no alert	no alert	no alert
	IE	no alert	no alert	no alert
DNS		no alert	no alert	no alert

Evaluation Results: When manually inspecting the analysis reports, not surprisingly, the implants were not found in any of the sandboxes. After verifying the evasion capabilities, we then checked if our implant checks triggered any security alerts from the appliances. We summarize the results in Table 1. The first column contains the implant technique, and the last three columns show the result per vendor. We first implemented test samples that are executable on systems that are backwards-compatible to earlier Windows versions, down to Windows XP. We had a look at the reports produced by the appliances and found that a security alert about *Hardware access* was reported for every single sample. The alert was reported from Windows XP, by one of the nine sandboxes which belonged to appliance A. For samples that check HTTP cookies, we obtained another alert about *Browser access* from three sandboxes, which belonged to appliance A. The sample that checks the DNS stub resolver cache raised an alert about *Hardware access* from two sandboxes (including Windows

XP) which belonged to appliance A, and an alert about a Windows command (`ipconfig`) from three sandboxes which belonged to appliance B.

For further analysis, we implemented test samples that not only searched for implants, but also encapsulated an encrypted version of a malicious payload that all sandboxes would detect if it was not hidden. That is, we chose to use malware that was seen in a real attack campaign targeting organizations in Japan and Taiwan [4]. We submitted the malware sample to the appliances and verified that it indeed was detected as such by all sandboxes. We then wrapped the malware in the customized malware, protected by a decryption that would only trigger if implants were found. Technically, the sample searches for the implant, and when the URL is found, decrypts and executes the actual malware. In fact, adversaries could include the decryption key in implants, such that even multi-path execution or manual analysts would fail to obtain the malicious packed payload. In our evaluation, the implants are the same as for the previous experiment and searched for in the browser history (for Chrome, Firefox, and IE), browser cache (for Firefox and IE), and DNS stub resolver cache. Seeing that one of the three appliances reported alerts when accessing cookies, we excluded the HTTP cookie from further analysis. Assuming that the target host operates Windows 7 or newer, we implemented these samples using libraries that do not work on earlier Windows versions, leaving us to five sandboxes of three appliances in our test setting. Using this new API, only a single sandbox raised an alert when our sample tried to inspect the DNS stub resolver using `ipconfig`. To counter this alert, we used `DnsGetCacheDataTable`, which did not raise any alert. In summary, as Table 2 shows, our updated implant search did not trigger any alerts. We manually inspected the analysis reports of all sandboxes and verified that none of the sandboxes decrypted the malware sample, meaning that our implant checking mechanism worked as expected.

Evaluation Summary: To summarize, an attacker can *implant* several identifiers into the target-specific system using web-based techniques. The implanted features can be used to implement *customized malware* that can stealthily evade malware security appliances. Adversaries with insider knowledge on anti-malware appliances, or having an oracle that tells whether their customized malware is detected as such, can tweak their implant mechanism such that the evasion is stealthy and remains undetected.

4 Discussion and Limitations

4.1 Defenses Against the Attack

As we have shown, attackers could reliably distinguish between a target-specific system from others based on *implanted* identifiers. As an option to solve these problems, we consider three solutions. First, we envision raising the bar for creating implants. Second, sandboxes and appliances can be tuned against such attacks. Third, appliance could be included in the reconnaissance phase to learn implants. We will discuss those ideas and their limitations in the following.

Destroying Implants: Adversaries can implant features into the target system using web-based techniques. An important detail is that the implant looks benign to the target system and sandboxes when looking them up. Therefore, detecting the implant itself require additional care and cannot be easily thwarted. Potential targets could aim to destroy implants by periodically deleting the cookies and browser history, or disabling caching to destroy cache-based implants. However, especially disabling browser and DNS caches would likely degrade user experience due to increased communication latencies, showing a fine line between utility and security. It is possible to change where to store browser cache and make malware authors difficult to find. In our experiment, we found that Chrome stores browser cache randomly. Although, we note that an attacker could combine multiple implants, and if a single implant survives, any attempts to hide other implants would be rendered ineffective. Alternatively, one could use a proxy service to make the tracking difficult. This technique would even destroy most of the proposed implant strategies. Still, proxy services leaves records into the target system. An attacker can search for this identifier instead of tracing implants.

Reconnaissance Detection in Sandboxes: An orthogonal solution to destroy implants would be for sandboxes to become aware of the reconnaissance phases. Such detection would allow the sandbox to identify suspicious behavior in the customized malware, regardless of whether it unpacks the malicious payload. At the risk of raising false alerts, sandboxes could aim to identify the various strategies that customized malware has to use in order to find an implant. Our work can help sandbox developers to become aware of implant strategies and assist them in raising alerts when seeing such behavior. By now, appliances detect sandbox evasion techniques by monitoring access to credential files and the registry. We show that the monitoring search space needs to be significantly extended, including browser-related files (cookies, cache, history) and Windows internals such as the DNS stub resolver's cache.

An obvious challenge is to keep up with all potential implant techniques that attackers could use. Our list of possible implants is by no means complete. For example, during experiments, we also inspected the Windows system log that includes various system events. We found that one can create log entries by trying to resolve a domain for which the authoritative name server times out. This would be another excellent way to place an attacker-controllable implant in a system, and demonstrate how creative attackers might become when choosing implant techniques. By adapting to these techniques, sandbox operators can aim to *detect* potential customized malware samples. Unfortunately, this does *not* help in unpacking encrypted payloads that depend on the value of certain implants that are missing in the sandbox or are unknown to the manual analyst.

Including Sandboxes in the Reconnaissance Phase: At the core of our threat model, we assume that the target system undergoes the reconnaissance, whereas the appliance does not. Although the appliances did not allow for net-

work communication, we did not observe any actions that tries accessing to our web site, while the target host was protected by the appliances. This could be changed if appliances are included in the typical reconnaissance steps, e.g., when they automatically follow URLs listed in emails sent to targets. The dangers of such automated URL visits make them impractical, though, as they will cause undesired side effects such as unsubscribing from mailing lists, or mistakenly confirming email-based authentication requests. Other drawbacks, such as attackers being able to learn which email address is protected by an appliance, and requiring the appliance to have Internet access (which none in our testbed had), give us the impression that this idea is only a last resort.

Suppose that despite these drawbacks a sandbox would indeed visit the URLs, and by doing so, collect implants. Even then, one complication is that sandboxes typically use snapshots to clean the system after each analysis to properly differentiate and to avoid side-effects between two malicious inputs, respectively. As a sandbox cannot correlate reconnaissance and intrusion phases, both phases would be executed separately in a clean system snapshot. Therefore, the intrusion phase would not see implants placed in the reconnaissance phase.

4.2 Ethical Considerations

Our research may seem offensive in the sense that we reveal a methodology that adversaries can use for targeted attacks. However, with our insights, sandbox operators will have a heads-up to implement stealthier analysis systems. While it will always be possible to find artifacts that can identify an individual sandbox or user machines, it is significantly harder to distinguish the target-specific system, especially if vendors synchronize the characteristics of the sandbox with user systems. However, attackers may not need profiles of the target, but could *implant* an identifier. Therefore, it is important to be more sensitive to programs that conduct sensitive data acquisitions that is used to discover implants. A responsible disclosure process (see next subsection) informed vendors of the appliances to prepare for attacks described in this paper. We gave suggestions on which activities to monitor for detecting programs that conduct implant acquisitions to appliances vendors prior to the conference. Finally, We anonymize the appliance vendors to avoid exposing weaknesses of specific vendors or products.

4.3 Responsible Disclosure

Appliance vendors are immediately affected by our research results and we thus considered them as the target of our responsible disclosure process. We chose to disclose our result to the three appliance vendors from our experiments, and to additional eleven popular appliance vendors. To notify these organizations, we contacted them 120 days prior to the publishing date of this paper, detailing the proposed attack and including hints on how to protect against potential adversaries in the future. We used direct contacts whenever possible and available. Alternatively, we resorted to contact details stated on the organization's web sites, notably including Web-based contact forms. If we did not

hear back after four weeks, we retried to contact the organizations, if possible using alternative communication channels (e.g., using generic email addresses like info@organization.com or email addresses found in the WHOIS database for the organization's web site domain) and received a feedback. If we did not hear back after eight weeks from the first process, we contacted the national CERT(s) that are in the same country as the affected organization in order to notify the party via the CERT as a trusted intermediary.

We provided each organization an executive summary of our research results as well as a full description of our research methodology (i.e., a copy of this paper in the pre-print version). We made sure to highlight the implications of our work with respect to future operations of the appliance. We also specified our contact details for both research institutions, including physical address, phone number, and the email address of a representative for the research activities. We allowed organizations to download the latest version of the test samples and their source code. Such auxiliary data helps to build protection mechanisms against customized malware. We removed all organization and product names as well as precise messages (security alerts) that were reported by the appliances. To this end, we received feedbacks from most of the organizations and provided test samples and their source code.

4.4 Limitations of the Attack

Enforcement of Reconnaissance Phase: Our attack makes use of emails that contain a URL in order to create an implant. We assume that the targeted mail attracts the user, such that the target host will access the URL and complete the reconnaissance phase. We consider the way the attacker tricks the user into clicking the URL a separate research topic that is out of the scope of this work. In practice, however, sophisticated attacks involving social engineering would succeed with this task. Even if a user does *not* click, adversaries can make the target host automatically access the URL. For HTML mail, most email clients support techniques to prefetch web access or download images from the Internet. For example, Apple Mail has prefetching of external email content enabled by default. Attackers can easily create emails that use these techniques to complete the reconnaissance as soon as the user opens the email. On occasions where the email is opened through web browsers, the accessed URL or image is saved in caches. Even for some mailers, the accessed URL is saved in a temporary file, so the attacker can implant a specific URL. However, for services that use proxy servers for access (e.g., Gmail), serving implants is difficult. Similarly, the functionality of external email content prefetching is not always (and should not be) allowed, again requiring an active click on a URL as fallback. We leave the problem of luring a user into clicking such an implant URL as future work.

Stability of Implants: After the malware author has accomplished the reconnaissance phase, the intrusion phase is started. If the phases are far apart from each other, certain implants may get lost. However, we argue that the typical gap

between reconnaissance and intrusion is in the range of several days, which is a short enough period for implants to remain stable. Most of the implants do not decay over time, unless action is taken by the user or they are blocked from the beginning. For example, the browser history and cache file may not record access histories, which prevents implants from being created. Cookies could be configured to be deleted periodically, which disturbs the implants being recorded. The DNS stub resolver cache may be cleaned, which implicitly destroys implants. We argue that a combination of *multiple* implants would survive most of these individual deletions and decays. Note that this does not sacrifice accuracy, as any individual implant out of a set of multiple unique implants is still unique. That is, the customized malware could search for multiple implants and trigger its malicious behavior when *any* implant was found.

False Positives: It is not in question that implants can be made unique with an attacker-controlled (high) entropy. We thus consider that the possibility of a non-targeted host to be falsely recognized as the targeted host is low. Even if such a coincidental match happens, assuming that the malware spreads to hundreds of other systems (including sandboxes), it is still unlikely that the coincidentally matching system will ever get in contact with the customized malware.

5 Related Work

5.1 Sandbox Evasion Techniques

Seeing the wide use of sandboxes, malware authors have been trying to evade sandbox analyses. There are various types of sandbox implementations, and Egele *et al.* give a comprehensive overview [25]. Most sandboxes use virtual machine (VM) technology [8,10,14] or CPU emulators [3,17]. Techniques to check for artifacts that indicate the presence of virtualization solutions are seen in modern malware [13,15,18,20,35,53,55,56]. Accordingly, there have been a number of studies about how to distinguish between a real machine and a virtual environment. RedPill [52] is one of the most well-known methods, and determines whether it is executed on VMware using the `sidt` instruction. Many other detection methods have also been developed not only for VMware [32,48], but also for famous system emulators such as QEMU [11,28,32,40,45,48], and BOCHS [28,40,45]. There are also some detection methods for emulation-based Android sandboxes [30,47,59]. Garfinkel *et al.* [27] surveyed the wide range of dissimilarities between real and virtualized platforms, and Chen *et al.* [22] developed a taxonomy of anti-virtualization and anti-debugging techniques that are used by modern malware. Although these techniques work against classical sandboxes, malware security appliances are designed to protect endpoints from recent threats and may not be as susceptible.

On the other hand, evading sandboxes does not necessarily require searching for virtualization or emulation artifacts. Malware has already started to evade sandboxes based on the lack of user interaction, such as checking for mouse

events [53,55,56] or waiting for a user to close a dialog box [20,56]. Recent research identifies features that are common on user systems [44]. These types of evasions are based on seeing that the system is used by a real user, and thus may seem similar to our approach. The fundamental difference is that we do not try to identify if malware executes on just *any* (non-)sandbox system. Instead, we show that attackers can tailor their malware to specific target systems.

In addition to identifying sandboxes, fingerprinting has also become popular. Maier *et al.* [39] gathered several features of Android sandboxes and showed that Android malware can bypass the existing sandboxes by using the fingerprints. Regarding sandboxes for Windows malware, Yoshioka *et al.* [61] clustered and detected sandboxes by their external IP addresses. Yokoyama *et al.* clustered sandbox fingerprints and created a classifier that can distinguish user machines from sandboxes [60]. They gathered fingerprints from sandboxes to user machines and proposed sandbox-inherent features so that even appliances placed in real networks can be classified as sandboxes. Najmeh *et al.* collected user and sandbox fingerprints to assess the degree of system use and age [42]. They developed statistical models that capture how realistic the system's past use looks to aid sandbox operators in creating system images that exhibit a realistic wear-and-tear state. We were inspired by these works and swifted the problem to homogeneous target environments. To this end, we aim to use implant-based techniques to identify the *target* systems, instead of any sandboxes or any benign systems. Our work extends existing ideas in that we show how an attacker might escape the typical arms race of identifying sandboxes using customized malware.

5.2 Transparent/Bare-Metal Sandboxes

Seeing the threat of VM evasion, researchers started to explore transparent sandboxes that are stealthy against detection. Vasudevan and Yerraballi proposed Cobra [58], which is a first dynamic analysis system focused on countering anti-analysis techniques. Dinaburg *et al.* proposed Ether [24], a transparent sandbox using hardware virtualization extensions such as Intel VT. Those systems focus on how to conceal the existence of analysis mechanisms from malware. Pek *et al.* introduced a timing-based detection mechanism to detect Ether [46]. Orthogonal to stealth VM-based sandboxes, Kirat *et al.* proposed to use actual hardware to analyze malware [7,34]. The proposed system, called BareBox, is based on a fast and rebootless system restore technique. Since the system executes malware on real hardware, it is not vulnerable to any type of VM/emulation-based detection attacks. In the context of Android sandboxes, Bordoni *et al.* proposed Mirage [38], an architecture that aims to arm an Android analysis system to tackle evasion by reproducing characteristics of real devices as much as possible into the Android emulator. These sandboxes increase evasion resilience in the classical setting, but would be evaded by customized malware.

5.3 Evasive Malware Detection

Separately from our work, researchers have identified the threat of evasive malware and studied its evasion attempts. As evasive malware became more popular, the demand for distinguishing whether malware has evasive functionality increased. Many of the evasive malware detection methods are based on comparison of behaviors between analysis and non-analysis environments. Balzarotti *et al.* proposed a method to detect malware that behaves differently in an analysis environment vs. a bare-metal reference host [12]. They first execute a malware sample on a reference host and compare its system call behavior with the execution in a virtual environment, revealing split behaviors in malware. Sun *et al.* proposed the behavior distance algorithm [57], which is based on generic string matching, for calculating the difference between two environments for the same malware. DISARM [37] compares behaviors in four emulation-based analysis systems and detects those differences. Barecloud [33] is a similar method to DISARM, but uses four fundamentally different analysis platforms, including bare-metal sandboxes. All these studies support the conclusion that malware already deploys evasion attempts. As we have shown, it will be difficult to detect customized malware. In fact, even advanced techniques such as multi-path execution [41] do not help in our threat model, as attackers can derive (strong) keys from implants to decrypt the piggybacked malicious payload.

6 Conclusion

The novel concept of *customized malware* might gain attention from malware authors in the future, as it solves many of their daily problems of evading increasingly stealthy and professional sandboxes. A fundamental challenge of malware authors remains evading anti-malware solutions that are based on sandboxes. Whereas our focus was email as infection vector, the problem scope goes beyond this. In fact, similar concepts could be integrated (much more easily, and fully automatically) into other popular ways of infections such as browser exploit kits. We thus think it is important to shed light on this potential new area which might add another burden to malware sandbox engineers, and to aid sandbox operators with ideas for protecting against such attacks. This is not only relevant for targeted attacks; also consumer malware could facility similar ideas by slightly sacrificing the implant accuracy to broader the set of accepted systems.

Implant techniques that we presented have been fairly successful in evading even the most modern malware security appliances. In settings like small agile organizations (startups, academia, etc.), where employees bring and configure their own devices: heterogeneous environment, an attacker can distinguish target system from others. Furthermore, it helps to infiltrate all kinds of environments, even those that require strict homogeneity. Regardless of the precise customization strategy being used, it is the simplicity of the proposed attack concept and its ease of automation that make it appealing for attackers. To give sandbox operators a timely heads-up before publishing our work, we have planned a rigorous responsible disclosure process.

Acknowledgements. We would like to thank the anonymous reviewers for their constructive feedback. This work was supported by the European Union's Horizon 2020 research and innovation program, project SISSDEN, under grant agreement No. 700176. A part of this work was funded by the WarpDrive: Web-based Attack Response with Practical and Deployable Research Initiative project, supported by the National Institute of Information and Communications Technology (NICT).

References

1. Advanced persistent threats: how they work. https://www.symantec.com/theme.jsp?themeid=apt-infographic-1
2. APT [Advanced Persistent Threat]. http://www.trendmicro.com/vinfo/us/security/definition/advanced-persistent-threat
3. bochs: The open source IA-32 emulation project. http://bochs.sourceforge.net
4. Darwins favorite APT group. https://www.fireeye.com/blog/threat-research/2014/09/darwins-favorite-apt-group-2.html
5. Malwr - malware analysis by cuckoo sandbox. https://malwr.com/
6. The mystery of the encrypted gauss payload. https://securelist.com/the-mystery-of-the-encrypted-gauss-payload-5/33561/
7. NVMTrace: Proof-of-concept automated baremetal malware analysis framework. https://code.google.com/p/nvmtrace/
8. Oracle VM VirtualBox. https://www.virtualbox.org
9. Public key pinning extension for http. https://tools.ietf.org/html/rfc7469
10. VMware. http://www.vmware.com/
11. Detecting android sandboxes (2012). http://www.dexlabs.org/blog/btdetect
12. Balzarotti, D., Cova, M., Karlberger, C., Kruegel, C., Engin, K., Vigna, G.: Efficient detection of split personalities in malware. In: Proceedings of the Symposium on Network and Distributed System Security, ser. NDSS 2010 (2010)
13. Barbosa, G.N., Branco, R.R.: Prevalent characteristics in modern malware (2014). https://www.blackhat.com/docs/us-14/materials/us-14-Branco-Prevalent-Characteristics-In-Modern-Malware.pdf
14. Barham, P., Dragovic, B., Fraser, K., Hand, S., Harris, T., Ho, A., Neugebauer, R., Pratt, I., Warfield, A.: Xen and the art of virtualization. SIGOPS Oper. Syst. Rev. **37**(5), 164–177 (2003)
15. Bayer, U., Habibi, I., Balzarotti, D., Kirda, E., Kruegel, C.: A view on current malware behaviors. In: Proceedings of the 2nd USENIX Conference on Large-Scale Exploits and Emergent Threats: Botnets, Spyware, Worms, and More, ser. LEET 2009, p. 8 (2009)
16. Bayer, U., Comparetti, P.M., Hlauschek, C., Kruegel, C., Kirda, E.: Scalable, behavior-based malware clustering. In: Proceedings of the Symposium on Network and Distributed System Security, ser. NDSS 2009 (2009)
17. Bellard, F.: QEMU, a fast and portable dynamic translator. In: Proceedings of the Annual Conference on USENIX Annual Technical Conference, ser. ATEC 2005, p. 41 (2005)
18. Branco, R.R., Barbosa, G.N., Neto, P.D.: Scientific but academical overview of malware anti-debugging, anti-disassembly and anti-vm technologies (2012). http://research.dissect.pe/docs/blackhat2012-paper.pdf
19. Caballero, J., Grier, C., Kreibich, C., Paxson, V.: Measuring pay-per-install: the commoditization of malware distribution. In: Proceedings of the 20th USENIX Security Symposium (2011)

20. Candid, W.: Does malware still detect virtual machines? (2014). https://www.symantec.com/connect/blogs/does-malware-still-detect-virtual-machines
21. Carsten, W., Ralf, H., Thorsten, H.: CXPInspector: Hypervisor-Based, Hardware-Assisted System Monitoring (2012)
22. Chen, X., Andersen, J., Mao, Z., Bailey, M., Nazario, J.: Towards an understanding of anti-virtualization and anti-debugging behavior in modern malware. In: Proceedings of the 38th Annual IEEE International Conference on Dependable Systems and Networks, ser. DSN 2008, pp. 177–186 (2008)
23. Chengyu, S., Paul, R., Wenke, L.: Impeding automated malware analysis with environment-sensitive malware. In: Proceedings of the 7th USENIX Conference on Hot Topics in Security, ser. HotSec 2012 (2012)
24. Dinaburg, A., Royal, P., Sharif, M., Lee, W.: Ether: malware analysis via hardware virtualization extensions. In: Proceedings of the 15th ACM Conference on Computer and Communications Security, ser. CCS 2008, pp. 51–62 (2008)
25. Egele, M., Scholte, T., Kirda, E., Kruegel, C.: A survey on automated dynamic malware-analysis techniques and tools. ACM Comput. Surv. **44**(2), 6:1–6:42 (2008)
26. Freiling, F.C., Holz, T., Wicherski, G.: Botnet tracking: exploring a root-cause methodology to prevent distributed denial-of-service attacks. In: di Vimercati, S.C., Syverson, P., Gollmann, D. (eds.) ESORICS 2005. LNCS, vol. 3679, pp. 319–335. Springer, Heidelberg (2005). https://doi.org/10.1007/11555827_19
27. Garfinkel, T., Adams, K., Warfield, A., Franklin, J.: Compatibility is not transparency: VMM detection myths and realities. In: Proceedings of the 11th USENIX Workshop on Hot Topics in Operating Systems, ser. HOTOS 2007, pp. 6:1–6:6 (2007)
28. Hao, S., Abdulla, A., Jelena, M.: Cardinal pill testing of system virtual machines. In: Proceedings of the 23rd USENIX Security Symposium (2014)
29. Ishimaru, S.: Why corrupted (?) samples in recent APT? case of Japan and Taiwan. https://hitcon.org/2016/pacific/0composition/pdf/1201/1201%20R1%201500%20why%20corrupted%20samples%20in%20recent%20apt.pdf
30. Jing, Y., Zhao, Z., Ahn, G.-J., Hu, H.: Morpheus: automatically generating heuristics to detect android emulators. In: Proceedings of the 30th Annual Computer Security Applications Conference, ser. ACSAC 2014 (2014)
31. Mayer, J.R., Mitchell, J.C.: Third-party web tracking: policy and technology. In: Proceedings of the 33rd IEEE Symposium on Security and Privacy, ser. S&P 2012, pp. 413–427 (2012)
32. Jung, P.: Bypassing sandboxes for fun. https://www.botconf.eu/wp-content/uploads/2014/12/2014-2.7-Bypassing-Sandboxes-for-Fun.pdf
33. Kirat, D., Vigna, G., Kruegel, C.: BareCloud: bare-metal analysis-based evasive malware detection. In: Proceedings of the 23rd USENIX Security Symposium (2014)
34. Kirati, D., Vigna, G., Kruegel, C.: Barebox: efficient malware analysis on bare-metal. In: Proceedings of the 27th Annual Computer Security Applications Conference, ser. ACSAC 2011, pp. 403–412 (2011)
35. Kruegel, C.: Evasive malware exposed and deconstructed (2015). https://www.rsaconference.com/writable/presentations/file_upload/crwd-t08-evasive-malware-exposed-and-deconstructed.pdf
36. Lanzi, A., Balzarotti, D., Kruegel, C., Christodorescu, M., Kirda, E.: AccessMiner: using system-centric models for malware protection. In: Proceedings of the 17th ACM Conference on Computer and Communications Security, ser. CCS 2010 (2010)

37. Lindorfer, M., Kolbitsch, C., Milani Comparetti, P.: Detecting environment-sensitive malware. In: Sommer, R., Balzarotti, D., Maier, G. (eds.) RAID 2011. LNCS, vol. 6961, pp. 338–357. Springer, Heidelberg (2011). https://doi.org/10.1007/978-3-642-23644-0_18
38. Bordoni, L., Conti, M., Spolaor, R.: Mirage: toward a stealthier and modular malware analysis sandbox for android. In: Foley, S.N., Gollmann, D., Snekkenes, E. (eds.) ESORICS 2017. LNCS, vol. 10492, pp. 278–296. Springer, Cham (2017). https://doi.org/10.1007/978-3-319-66402-6_17
39. Maier, D., Müller, T., Protsenko, M.: Divide-and-conquer: why android malware cannot be stopped. In: Proceedings of the 9th International Conference on Availability, Reliability and Security, ser. ARES 2014 (2014)
40. Martignoni, L., Paleari, R., Roglia, G.F., Bruschi, D.: Testing CPU emulators. In: Proceedings of the Eighteenth International Symposium on Software Testing and Analysis, ser. ISSTA 2009, pp. 261–272 (2009)
41. Moser, A., Kruegel, C., Kirda, E.: Exploring multiple execution paths for malware analysis. In: Proceedings of the 28th IEEE Symposium on Security and Privacy, ser. S&P 2007 (2007)
42. Najmeh, M., Mahathi, P.A., Nick, N., Michalis, P.: Spotless sandboxes: evading malware analysis systems using wear-and-tear artifacts. In: Proceedings of the 38th IEEE Symposium on Security and Privacy, ser. S&P 2017 (2017)
43. Neugschwandtner, M., Comparetti, P.M., Platzer, C.: Detecting malware's failover C&C strategies with squeeze. In: Proceedings of the 27th Annual Computer Security Applications Conference, ser. ACSAC 2011 (2011)
44. Nikiforakis, N., Joosen, W., Livshits, B.: Privaricator: deceiving fingerprinters with little white lies. In: Proceedings of the 24th International Conference on World Wide Web, ser. WWW 2015, pp. 820–830 (2015)
45. Paleari, R., Martignoni, L., Roglia, G.F., Bruschi, D.: A fistful of red-pills: how to automatically generate procedures to detect CPU emulators. In: Proceedings of the 3rd USENIX Conference on Offensive Technologies, ser. WOOT 2009 (2009)
46. Pék, G., Bencsáth, B., Buttyán, L.: nEther: in-guest detection of out-of-the-guest malware analyzers. In: Proceedings of the 4th European Workshop on System Security, ser. EUROSEC 2011, pp. 3:1–3:6 (2011)
47. Petsas, T., Voyatzis, G., Athanasopoulos, E., Polychronakis, M., Ioannidis, S.: Rage against the virtual machine: hindering dynamic analysis of android malware. In: Proceedings of the 7th European Workshop on System Security, ser. EUROSEC 2014 (2014)
48. Raffetseder, T., Kruegel, C., Kirda, E.: Detecting system emulators. In: Garay, J.A., Lenstra, A.K., Mambo, M., Peralta, R. (eds.) ISC 2007. LNCS, vol. 4779, pp. 1–18. Springer, Heidelberg (2007). https://doi.org/10.1007/978-3-540-75496-1_1
49. Rieck, K., Schwenk, G., Limmer, T., Holz, T., Laskov, P.: Botzilla: detecting the phoning home of malicious software. In: Proceedings of the 2010 ACM Symposium on Applied Computing, ser. SAC 2010, pp. 1978–1984 (2010)
50. Rieck, K., Trinius, P., Willems, C., Holz, T.: Automatic analysis of malware behavior using machine learning. J. Comput. Sec. **19**(4), 639–668 (2011)
51. Rossow, C., Dietrich, C.J., Bos, H.: Large-scale analysis of malware downloaders. In: Proceedings of the 9th Conference on Detection of Intrusions and Malware & Vulnerability Assessment, ser. DIMVA 2012 (2012)
52. Rutkowska, J.: Red pill... or how to detect VMM using (almost) one CPU instruction (2004). http://www.securiteam.com/securityreviews/6Z00H20BQS.html

53. Shinotsuka, H.: Malware authors using new techniques to evade automated threat analysis systems (2012). http://www.symantec.com/connect/blogs/malware-authors-using-new-techniques-evade-automated-threat-analysis-systems
54. Simone, M., Yanick, F., Antonio, B., Luca, I., Jacopo, C., Dhilung, K., Christopher, K., Giovanni, V.: Baredroid: large-scale analysis of android apps on real devices. In: Proceedings of the 31st Annual Computer Security Applications Conference, ser. ACSAC 2015 (2015)
55. Singh A., Khalid, Y.: Don't click the left mouse button: introducing trojan upclicker (2012). https://www.fireeye.com/blog/threat-research/2012/12/dont-click-the-left-mouse-button-trojan-upclicker.html
56. Singh, A., Bu, Z.: Hot knives through butter: evading file-based sandboxes (2013). https://media.blackhat.com/us-13/US-13-Singh-Hot-Knives-Through-Butter-Evading-File-based-Sandboxes-WP.pdf
57. Sun, M.K., Lin, M.J., Chang, M., Laih, C.S., Lin, H.T.: Malware virtualization-resistant behavior detection. In: Proceedings of the 17th IEEE International Conference on Parallel and Distributed Systems, ser. ICPADS 2011, pp. 912–917 (2011)
58. Vasudevan, A., Yerraballi, R.: Cobra: fine-grained malware analysis using stealth localized-executions. In: Proceedings of the 27th IEEE Symposium on Security and Privacy, ser. S&P'06, pp. 264–279 (2006)
59. Vidas, T., Christin, N.: Evading android runtime analysis via sandbox detection. In: Proceedings of the 9th ACM Symposium on Information, Computer and Communications Security, ser. ASIA CCS 2014 (2014)
60. Yokoyama, A., et al.: SandPrint: fingerprinting malware sandboxes to provide intelligence for sandbox evasion. In: Monrose, F., Dacier, M., Blanc, G., Garcia-Alfaro, J. (eds.) RAID 2016. LNCS, vol. 9854, pp. 165–187. Springer, Cham (2016). https://doi.org/10.1007/978-3-319-45719-2_8
61. Yoshioka, K., Hosobuchi, Y., Orii, T., Matsumoto, T.: Your sandbox is blinded : Impact of decoy injection to public malware analysis systems. J. Inf. Process. **52**(3), 1144–1159 (2011)