Enviral: Fuzzing the Environment for Evasive Malware Analysis

Floris Gorter
Vrije Universiteit Amsterdam
The Netherlands
f.c.gorter@vu.nl

Cristiano Giuffrida
Vrije Universiteit Amsterdam
The Netherlands
giuffrida@cs.vu.nl

Erik van der Kouwe
Vrije Universiteit Amsterdam
The Netherlands
vdkouwe@cs.vu.nl

ABSTRACT

Analyzing malicious behavior is vital to effectively safeguard computer systems against malware. However, contemporary malware frequently contains evasive behavior, which allows it to hide its malicious intent from analysis. More specifically, if the malware detects it is being executed in an analysis environment, it resorts to evasive routines that exhibit benign behavior. Manually deactivating evasive checks requires significant effort, and is therefore not a scalable technique with regards to the increasing amount of evasive malware. Unfortunately, the existing systems that *automatically* analyze evasive malware are impractical, computationally inefficient, or incomplete by design.

In this paper, we introduce Enviral, an automatic evasive malware analysis framework that proposes a novel method to analyze evasive malware, combining the best elements of existing approaches. We achieve this by applying fuzzing techniques to repeatedly adapt the view of the execution environment, thereby iteratively defeating the evasive checks in the target application. We realize these adaptations by applying mutations to the outcomes of environment queries, which in turn leads to the exploration of multiple execution paths. Our experimental results demonstrate that Enviral can detect and overcome evasive behavior and thereby exposes previously hidden activity in malware. We evaluate our system against a similar framework, and conclude that Enviral can expose 39% more *interesting* hidden system call activity on average, and achieves productive explorations where previously unseen behavior is discovered in 67% more malware samples.

CCS CONCEPTS

• **Security and privacy → Malware and its mitigation**.

KEYWORDS

evasive malware analysis, fuzzing, system call hooks

ACM Reference Format:
Floris Gorter, Cristiano Giuffrida, and Erik van der Kouwe. 2023. Enviral: Fuzzing the Environment for Evasive Malware Analysis. In *16th European Workshop on System Security (EUROSEC '23), May 8, 2023, Rome, Italy*. ACM, New York, NY, USA, 7 pages. https://doi.org/10.1145/3578357.3589455

1 INTRODUCTION

Every year, billions of malware attacks cause enormous damage. Nowadays, malware analysts face malware that is increasingly evasive [4, 7, 36]. Such evasive malware thwarts analysis by concealing its malicious payload. In essence, this type of malware detects analysis platforms to appear harmless there, while it continues its harmful behavior on target systems. Existing solutions to automatically analyze evasive malware either apply preventive methods [9, 22, 25], which employ predefined rules to hide known artifacts used to identify the analysis environment, or explore multiple execution paths in a reactive manner [6, 19, 21, 32], where branch conditions are flipped as a response to input-dependent branches. Unfortunately, preventive methods can be evaded, as some artifacts always remain, while reactive approaches suffer from path explosion and rely on forced execution, which can result in impossible execution traces.

To overcome the limitations of these existing approaches, we introduce Enviral, a hybrid evasive malware analysis system that combines the best of both preventive and reactive methods. We merge the optimistic approach of preventive methods with the exploratory power of reactive techniques, balancing their combined application based on the evasive capabilities of the specific malware under analysis. We achieve this by applying fuzzing techniques to repeatedly mutate the outcomes of environment queries in order to defeat the evasive checks.

Enviral distinguishes between *definite* and *volatile* mutations to represent preventive and reactive methods, respectively. Environment queries that we can clearly identify as evasive, which we do by evaluating the query context against known artifacts as preventive systems do, result in a definite mutation being applied to hide the corresponding artifacts. Alternatively, if we are not certain about the evasive properties of a query, we apply the mutation in a *volatile* manner. This implies that we measure whether the mutation results in an increase in coverage (i.e., the mutation bypasses a check). Volatile mutations correspond to the exploration seen in reactive systems, as we attempt to explore alternative execution paths as an indirect response to input-dependent control flow.

In order to mutate the view of the environment, we dynamically capture the behavior of the malware by tracking its system calls. Afterwards, our mutation strategy decides which system calls receive a modification in the next execution. For example, if we record a system call that checks whether a file is present, we may apply a mutation that ensures that the file is marked as not present in the next execution. Whether this concerns a definite or volatile mutation depends on the context of the query, for instance, whether the file name contains a known artifact. We realize this mutation strategy by building a non-traditional fuzzer that takes a list of system calls as input, and consequently modifies the view of the environment by applying various mutation rules. After the exploration phase finishes, Enviral produces a descriptive analysis report containing the overall behavior and evasive expectations of the malware, along with the consequences of each individual mutation in terms of discovered activity.

We show that Enviral can expose more hidden behavior in the evaluated malware samples than the state-of-the-art, with a geomean increase in unique (i.e., *interesting*) system call activity of 1.63x, versus the 1.17x of BluePill [9]. Additionally, out of 338 potentially evasive malware samples, our system manages to discover hidden behavior in 36.7% of the cases, while BluePill only does so in 21.9% of the samples.

Contributions. We summarize our contributions as follows:

- We propose an automatic evasive malware analysis approach, which can explore multiple execution paths of a target application in a consistent manner by repeatedly applying mutations to the outcomes of environment queries.
- We build an implementation of our approach to showcase that fuzzing techniques can be applied to the field of evasive malware analysis.
- We evaluate Enviral against a recently published analysis framework, where our experimental results indicate that Enviral can detect a larger number of evasive samples and discover more hidden behavior.
- Code available at: https://github.com/vusec/enviral

2 BACKGROUND

Malware analysts construct dedicated analysis environments (i.e., sandboxes) to safely inspect malware. Common techniques are virtualization [10], emulation, and hardware-based analysis systems [27]. We consider a malware sample to be evasive (or, *environment-sensitive*) if it detects execution in such environments and attempts to trick automated malware analysis services (e.g., an anti-virus sandbox) with a benign execution. Detection techniques can be as simple as calling the `IsDebuggerPresent` API function, but commonly includes environment artifacts, timing [13], and CPU semantics [26]. Environment artifacts span a wide range of fingerprints that can be used to identify an analysis system. For example, malware can detect the presence of certain files, Windows registry values, usernames, or the network configuration. Malware authors can include many evasive conditions in their programs to increase the chance of detecting different analysis environments [36].

3 DESIGN

Enviral automatically detects, documents, and overcomes evasive behavior in malware. We repeatedly mutate the outcomes of environment queries, effectively fuzzing the malware to reveal additional evasive checks, malicious activity, or unwanted behavior. We limit our mutation input space to system calls, which represent the main source of information about the environment. We intercept Windows' Win32 and Native APIs (referred to as *system calls*).

Figure 1 provides an overview of the components of Enviral and their interactions. The *System Call Hooks* intercept system calls, logging their use and returning mutated results on later runs. The *Controller* repeatedly launches the target application, each time with a different set of mutations. It uses runtime coverage as feedback to decide which input to mutate.

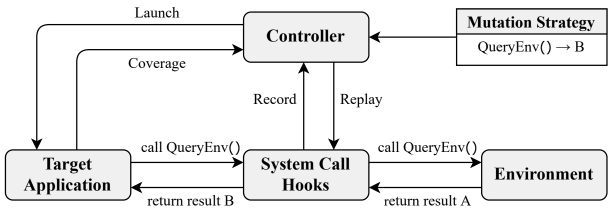

Figure 1: Overview of the components of Enviral.

3.1 Fuzzing the Environment

Enviral assumes malware generally follows an *abort if present* pattern. For example, if a malware sample detects the presence of a certain VirtualBox artifact, it aborts its malicious intent and resorts to evasive routines. In this case, a single mutation often suffices to determine if an environment query is evasive. That is, we observe whether negating the presence of an artifact results in the application advancing its (malicious) execution. We treat a recorded list of system calls as the input, and aim to mutate the right calls to progress the execution through the evasive behavior. Unlike traditional fuzzing, we do not mutate the call further after it manages to pass a branch, greatly reducing the search space.

3.2 System Calls

We use user-level system call hooks to intercept, record, and modify environment queries. We specifically intercept system calls that query the environment and those that signal possible malicious behavior. We record a total of 36 distinct calls (i.e., excluding Ex/A/W variants) to cover evasive behavior, and another 65 to capture possibly malicious activity. The evasion-related system calls are based on multiple existing sandbox detection tools [1–3, 16, 20]. We log context such as filenames where potentially relevant for mutations.

3.3 Mutations

We distinguish between two different types of mutations. *Definite* mutations modify systems calls that are clearly evasive. For example, an application checks if the process "VBoxTray.exe" is running, where the corresponding definite mutation indicates the process is not running. We are certain that the mutation has to be applied, because the process reveals the presence of VirtualBox. Definite mutations are similar to preventive evasive malware analysis systems [9, 22], but because we apply them adaptively we can report which evasion techniques the target uses. *Volatile* mutations are tentative adjustments to system calls that are only kept if they improve coverage. For example, consider a scenario where an application checks if the file "UnknownArtifact.txt" is present. The corresponding volatile mutation forces the outcome of the call to indicate that the file is not present. Afterwards, our mutation strategy consults the coverage metric to decide whether this volatile mutation was beneficial, and hence whether to keep it. Volatile mutations mimic reactive analysis systems [19, 21], but we require neither expensive constraint solving nor forced execution, which can create inconsistent or impossible states. Instead, we efficiently explore the search space using fuzzing techniques, and can mutate system call results consistently because our mutations are context-aware.

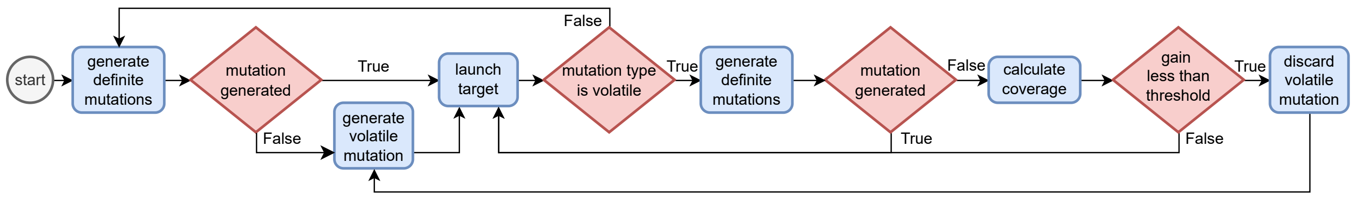

Figure 2: Flowchart of our exploration strategy.

Figure 2 shows our mutation strategy. We first apply any possible definite mutations, and then rerun until no additional definite mutations apply. We identify possible definite mutations from the system call contexts, for example searching for substrings like "VBox". We then apply volatile mutations one-by-one, starting from the system call at the end of the trace, closest to termination. We keep only those mutations that improve coverage.

3.4 Coverage

We measure coverage to decide whether volatile mutations are beneficial. We are specifically interested in malicious and evasive behavior, so we measure coverage in terms of system calls. We deduplicate system calls based on their call stacks to prevent additional loop iterations from affecting the coverage measurement. Using the number of *unique* system calls as coverage metric does not introduce any additional overhead or artifacts, since the call recordings are performed as a part of Enviral regardless. Although a more (complex) fine-grained tracking method (e.g., hardware tracing) can improve the feedback loop, we believe that the current evaluation metric is adequate for showcasing the purpose of the system.

4 IMPLEMENTATION

We implement Enviral for 32-bit Microsoft Windows systems using a total of 6678 lines of C++ code. We use Microsoft Detours [8] to insert user-level hooks on Win32/NT API calls. We apply our instrumentation to the target process using DLL injection, and set up a bi-directional pipe to communicate new mutations and the resulting coverage in terms of system call recordings.

5 EVALUATION

For the evaluation of Enviral, we run our experiments in a VirtualBox VM with Windows 7 Ultimate 32-bit, one Intel Core i7-8650U CPU core, 8 GB of RAM and 350 GB of SSD disk space. We leave the virtual machine as close to stock as possible, since we want the evasive malware to detect the virtualized environment in order to discover and defeat its evasive checks. Unfortunately, due to limitations in the current setup, the VM does not have an active internet connection. Nonetheless, one could argue that if we reach any networking behavior, we have likely already hit the malicious activity, although networking can also constitute evasive behavior.

5.1 Data Set

For our experiments we make use of a malware data set dump obtained from VirusTotal in 2019. The data set consists of roughly 35K Win32 executables. We deduplicate the data set using the attached *vhash* [29] clustering metric, which leaves a total of 8440 samples. This subset consists of 27 different types of malware (e.g., trojans and viruses), and 715 distinct malware families.

5.2 Exploration

To evaluate the exploratory capabilities of Enviral, we run our automated analysis on all of the 8440 deduplicated malware samples. This allows us to view to what degree Enviral manages to discover hidden system call activity in the malware. Unfortunately, there is no ground truth on which samples contain evasive behavior. We set the individual exploration time limit to 2.5 seconds, which means that each configuration of mutations is evaluated for a maximum of 2.5 seconds. We empirically decide this duration based on the assumption that malware generally contains evasive behavior at an early execution stage, and through initial results showing that the time is sufficient for malware to exit gracefully or end up in an infinite loop. The total exploration time is set to 50 seconds, meaning that we can assess at least 20 incremental mutation setups.

Out of the 8440 samples, Enviral produces an increase in system call activity in 2206 samples (26.1%). To achieve this, Enviral applies a total of 8694 mutations, consisting of 2738 distinct mutations. Of the 8694 mutations, 47.6% are of the volatile type, and the remaining 52.4% are definite mutations. As a result, we measure a geometric mean (geomean) increase of 1.47x more system calls. Note that the measured increase in system call activity concerns calls with a distinct origin, since we filter out loops and foreign activity, and hence encapsulate the increase in *unique* system calls. Additionally, since the system calls we record are constrained to calls that we select for their evasive or malicious properties, the increase in behavior represents *interesting* activity.

Since we observe that the execution of the malware samples can be noisy, we select all the gainful samples where the standard deviation of the number of system calls among three baseline runs is zero, and hence have a *stable* baseline. This selection procedure results in 1613 samples with a stable baseline. The geomean increase in system call activity for these samples is 1.58x.

Only considering the relative increase in the number of unique system calls does not accurately represent the ability of Enviral to discover previously hidden behavior in the malware. For example, a relative increase of 2.0x can concern a small number of system calls if the baseline is small. In order to provide a more complete view of the exploration, Figure 3 displays the absolute increases in the number of interesting system calls for the 1613 malware samples with a stable baseline. The majority of the samples fall into the 0 to 50 calls range, and we see increases up to 455 additional unique calls. While the increases in system call activity do expose hidden

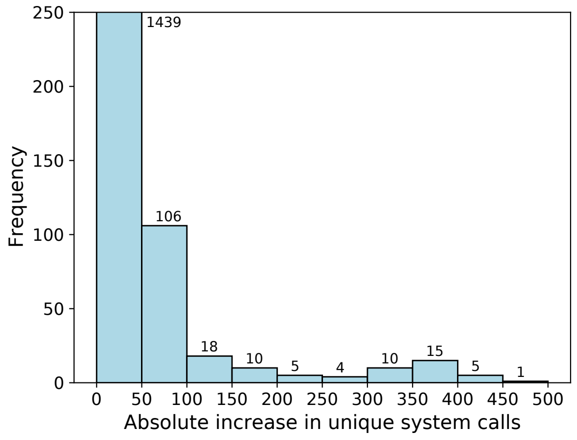

Figure 3: Histogram of the absolute increases in the number of unique system calls for the 1613 stable malware samples.

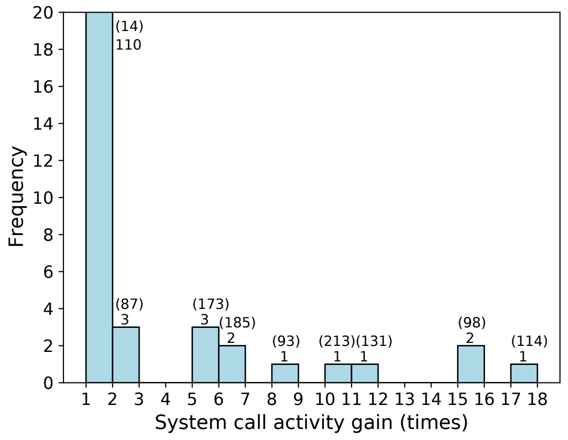

Figure 4: Histogram of the relative increases in the number of unique system calls for the 124 gainful samples that interact with VBox. (Mean absolute gain per bar in parentheses).

behavior, especially when the baseline is stable, the new behavior does not necessarily concern the malicious payload of the malware. Since there is no ground truth available regarding this aspect, we cannot report on the success of triggering the harmful routines. However, even if Enviral does not defeat all of the evasive checks up until the payload, the automatically generated analysis report can help in further analyzing the malware.

5.3 Virtual Machine Detection

Using the resulting system call logs of the previous exploration experiment, we select all the malware samples that exhibit calls where the context contains a reference to a VirtualBox artifact. For example, the logs may contain a `NtQueryAttributesFile` call with *VBoxTray.exe* in its context to detect the presence of this VirtualBox-specific file. This selection process produces 338 malware samples that appear to contain behavior to detect virtualized environments. We use this subset to evaluate the ability of Enviral to detect and overcome evasive behavior. However, we point out that even though the logs contain a reference to VirtualBox, some of the malware samples perform routines that scan the hard disk (e.g., ransomware). Such scans result in the virtual machine artifacts appearing in the logs without necessarily being concerned with an evasive check. The selected samples consist of 21 different types of malware, and 101 distinct malware families. The family distribution indicates that the potentially evasive samples concern a reasonable spread of families, with the largest representative, the *Neshta* virus, covering 9.5% of the 338 samples. For this aforementioned representative, we confirm (using Joe Sandbox) that the malware family contains capabilities to detect virtual machines [24].

Out of the 338 potentially evasive malware samples, Enviral achieves a gain in unique system call activity in 124 cases. As a result, we measure a geomean increase of 1.50x more interesting system calls. Selecting the targets with a stable baseline leaves 94 samples, for which the geomean increase is 1.63x. Figure 4 shows the distribution of the relative increases in unique system call activity for the 124 gainful samples. Most of the samples fall in the 1.0 - 2.0x range, however we reach up to a 17.3x increase for one particular sample. The 17.3x increase arises from a stable baseline of 7 calls, followed by 121 recorded calls when Enviral applies 6 mutations. More specifically, the application exhibits the largest increase in system call activity when Enviral mutates a registry call that detects the virtual hard disk (`DiskVBOX_HARDDISK`).

System Call	Context	Freq
NtQuerySystemInformation	SystemBasicInformation	249
NtQueryInformationProcess	Various DLLs	146
GetForegroundWindow	None	90
NtQuerySystemInformation	VBoxTray/VBoxService	72
NtQueryAttributesFile	[...]\CRYPTBASE.dll	72
Process32First	None	60
Process32Next	None	59
NtOpenKeyEx	[...]\VirtualDeviceDrivers	33
NtQueryAttributesFile	[...]\VBoxWHQLFake.exe	32
NtQueryAttributesFile	[...]\VBoxTray.exe	32
NtQueryAttributesFile	[...]\VBoxDrvInst.exe	32
NtQueryAttributesFile	[...]\VBoxControl.exe	32
NtOpenKey	[...]\Language Groups	31
NtCreateFile	\??\VBoxMiniRdrDN	28
LoadLibraryEx	[...]\VBoxMRXNP.dll	28

Table 1: Top 15 most frequent mutations in the 338 malware samples that interact with VirtualBox.

Moreover, Table 1 shows the 15 most frequent mutations that Enviral applied to the 338 potentially evasive malware samples. The frequency denotes to how many of the samples the mutation was successfully applied. The table highlights that the most frequent mutations often have a clear connection to an artifact that can expose virtualization or analysis environments.

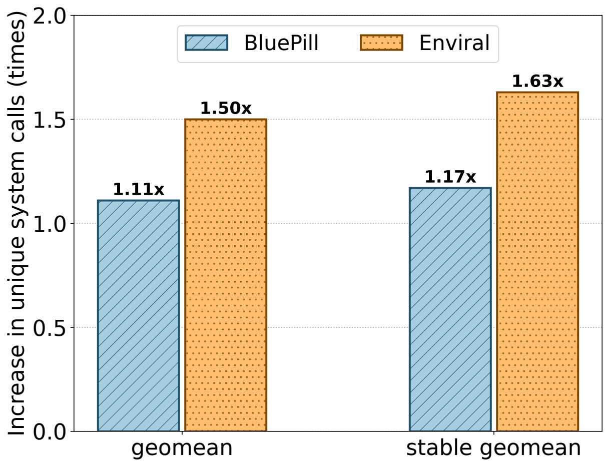

Figure 5: Comparison between the increases in unique system call activity of BLUEPILL and ENVIRAL for the 338 VirtualBox-related malware samples.

5.4 Comparison against the State-of-the-Art

To provide additional insight into the performance of ENVIRAL, we evaluate our system against a recently published automatic evasive malware analysis framework called BLUEPILL [9]. BLUEPILL is a preventive measure against evasive malware that relies on dynamic binary instrumentation to hide revealing artifacts on demand. The framework requires all artifacts and corresponding modifications to be defined statically beforehand.

Since BLUEPILL only produces a list of the captured evasive checks, and does not track the overall behavior of the malware, we extend the framework to accommodate a direct comparison. We reuse the call instrumentation of BLUEPILL to produce system call logs of the same functions as ENVIRAL. Since we want the comparison to be as accurate as possible, we implement the same optimizations present in ENVIRAL. That is, we detect loops, filter out foreign activity, and avoid nested hooks.

We compare the capabilities of ENVIRAL and BLUEPILL to detect and defeat evasive malware using the 338 (potentially evasive) malware samples described in Section 5.3. For each malware sample, we execute BLUEPILL thrice to generate a baseline, and another three times with the anti-evasive countermeasures enabled for the regular analysis. Just as in ENVIRAL, we select the baseline with the largest number of system calls, as well as the most successful countermeasure run. The execution duration of BLUEPILL is set to 10 seconds, which is greater than ENVIRAL, to take into account the overhead of Intel Pin and the added extensions.

Figure 5 shows the results of the comparison. To recall, ENVIRAL achieves an increase in unique system call activity in 124 of the samples (of which 94 are stable), with a geomean increase of 1.50x, and 1.63x for the stable samples. On the same 338 selected malware samples, BLUEPILL produces an increase in unique system call activity in 74 cases. More specifically, the geomean increase in recorded system calls for the 74 gainful samples is 1.11x. Out of these 74 samples, there are 35 cases that display a stable baseline. The geomean increase in activity of this subset is 1.17x. Comparing these results to ENVIRAL, we observe that ENVIRAL can expose hidden behavior in more samples while also revealing a greater magnitude of activity than BLUEPILL in the evaluated malware samples. Since both analysis systems are designed to deactivate evasive checks upon detection, the resulting increases in system call activity can be attributed to detecting and defeating evasive behavior, especially if the baseline is stable. Assuming that there is a correlation between disabling evasive checks and an overall increase in system call activity, we argue that ENVIRAL can detect more evasive behavior than BLUEPILL, since our system discovers more hidden behavior.

Although BLUEPILL in fact covers more evasive methods, such as the `cpuid` instruction to detect the 'hypervisor present' bit, from our experiments we conclude that ENVIRAL is able to discover more hidden behavior. Additionally, the fuzzing approach of ENVIRAL allows us to discover the dependencies and consequences of evasive checks, which in turn results in a descriptive analysis report with the corresponding increase in behavior of each defeated check. In contrast, the unmodified BLUEPILL framework focuses solely on evasive behavior, and hence is constrained to reporting on the observed evasive attempts. Additionally, a previously unseen system call based evasive technique would have to be discovered manually in order to be incorporated in BLUEPILL, whereas ENVIRAL may defeat the evasive check using a volatile mutation, after which the new method could become a definite mutation in future analyses.

6 LIMITATIONS

Although ENVIRAL actively tries to conceal its presence, for example by hiding the injected module from the target process, there are still ways for malware to detect the analysis. The current design of ENVIRAL has some limitations that negatively impact its detectability. First, since we insert user-level system call hooks, malware can detect the placed detours by checking for jump instructions at the start of functions. Second, the evasive countermeasures of ENVIRAL are constrained to system calls, which means malware can still detect the overhead imposed by the analysis using the `rdtsc` assembly instruction (and virtualization using `cpuid`). Both of these limitations can be prevented, for example by using kernel-level hooks and performing additional instrumentation at the instruction level, thereby improving the transparency of the system. Even though kernel-level hooks can still be detected, especially if the malware escalates privileges, they are stealthier than user-level hooks.

Malware can also apply anti-analysis techniques to bypass or deceive the system. For example, user-level hooks can be bypassed by directly invoking a system call via assembly. This results in the system call not being recorded, and therefore not being mutable. In addition, malware could guide ENVIRAL away from malicious code by performing decoy evasive checks in benign code. This can be addressed by adding backtracking to explore alternative paths.

Additionally, in recent studies we see the introduction of anti-fuzzing techniques, where programs are equipped with defenses specifically designed to hinder fuzzing [11, 12]. Malware authors can incorporate these anti-fuzzing techniques in their programs to ensure resilience against fuzzing-based malware analysis. We identify four distinct anti-fuzzing approaches: slowing down the exploration, tampering with coverage, preventing crashes, and obstructing symbolic execution. Since Enviral does not aim to detect crashes, and does not rely on symbolic execution (or taint analysis), the last two techniques are not of relevance. Our system is not very sensitive to anti-fuzzing techniques in general because, by design, it generates relatively few mutations.

7 RELATED WORK

7.1 Fuzzing Malware

Existing literature concerned with automatically analyzing evasive malware has introduced techniques such as reference platforms [5, 7, 13–15, 18, 26], forced execution [6, 19, 21, 28, 31, 32], and hardened environments [9, 17, 22, 25, 27, 33–37]. Furthermore, applying fuzzing techniques for the purpose of evasive malware analysis has already gained some traction for Android malware, though Windows malware has not received the same attention. For instance, FuzzDroid [23] and DirectDroid [30] are automatic analysis framework designed to expose malicious behavior in evasive Android malware. Similarly to Enviral, the main idea of these systems is to fuzz a set of APIs in order to repeatedly adapt the values that the target application obtains when interacting with its environment. These Android-based systems steer the application towards a configurable target location, where the distance of the executed path to the target location is used as exploration metric.

The main difference between Enviral and the approaches of FuzzDroid and DirectDroid lies with the assumptions regarding the exploration. More specifically, the existing systems assume that the analyst is able to specify a target location to explore towards, while this seems counterintuitive for evasive malware, where the target locations of interest still have to be determined by defeating the evasive checks. We point out that, perhaps, in Android malware the possible malicious acts (e.g., sending an SMS) are more constrained than in Windows malware. Therefore, it may be reasonable to expect the specification of a target location, for example to explore until the SMS API is reached. In contrast, Enviral explores the target binary in a more autonomous fashion, where it tries to maximize *interesting* behavior. This means the analysis is more independent, while it is, however, also more susceptible to deception and faulty exploration.

7.2 Reference Platforms

Evasive malware analysis systems that make use of a reference platform aim to detect the differences between executions of a malware sample on a virtualized and physical machine. By comparing the execution traces, the analysis can determine the divergence point (i.e., the point where the traces no longer match), which signifies the point of evasion. There exist multiple analysis systems that employ different types of reference platforms, such as stealthy virtualized environments and hardware-based machines [5, 13, 15, 18].

Using a reference platform is a method that is distinct from the preventive and reactive evasive countermeasures mentioned throughout this paper. In a sense, reference platforms introduce a third anti-evasion class; namely, while preventive measures are defined *before* the execution, and reactive analysis is applied *during* execution, reference platform analysis comes into play *after* the execution. In general, malware analysis based on a reference platform is mostly suitable for detecting evasive checks, and not mitigating (i.e., counteracting) the evasion. However, Kang et al. [13] demonstrate that if the complete instruction traces are available, patches against the evasive checks can be derived from the divergence points using data flow analysis.

The main drawback of this technique is the difficulty of recording the behavior of malware on a transparent reference platform. Since the malware may harm the system or infect others, the malware preferably is executed in a safe environment, such as a sandbox. Clearly, if the malware detects the reference platform, the analysis will not succeed. Creating a truly transparent and safe execution environment that can also record the behavior of the malware is a complicated task. In contrast, our system does not require a carefully crafted execution environment, since Enviral is designed to operate in any (analysis) environment, such that the evasive checks can be detected and defeated. However, reference platform analysis and fuzzing the environment do both share the principle of comparing the behavior of multiple executions of the malware to identify evasive routines.

8 CONCLUSION

In this paper, we introduce Enviral, an automated evasive malware analysis system. With Enviral, we showcase that fuzzing techniques can be beneficial in malware analysis. Existing evasive malware analysis solutions often disable evasive checks by applying either preventive or reactive measures. Unfortunately, preventive analysis can quickly become ineffective, since all of the detectable artifacts have to be defined beforehand. Similarly, reactive analysis suffers from scalability issues, due to the computational expense of exploring multiple execution paths. To overcome these limitations, Enviral employs a hybrid approach of preventive and reactive methods, which we achieve by combining fuzzing techniques with the repeated adjustment of the view of the execution environment. We realize this by defining two different types of mutations: *definite* and *volatile* mutations, which we implement using user-level system call hooks. The resulting analysis framework produces descriptive reports containing the overall behavior and evasive expectations of the malware, along with the additional system call activity induced by each individual mutation.

We evaluate our system on its exploratory capabilities, the ability to detect and overcome evasive behavior towards VirtualBox, and also in comparison to BluePill, a recently published evasive malware analysis tool. Using a malware data set obtained from VirusTotal, Enviral reaches a 1.58x geomean increase in *interesting* unique system call activity on 1613 malware samples with a stable baseline. Furthermore, using the behavioral logs of Enviral we identify 338 malware samples that contain a reference to VirtualBox artifacts. Considering only the samples with a stable baseline, our analysis system manages to expose hidden behavior in 94 of these potentially evasive samples, with a geomean increase in system call activity of 1.63x. In comparison, analyzing the same samples with BluePill results in 35 gainful samples, with a geomean increase in system call activity of 1.17x. From our experiments we infer that Enviral is able to detect more evasive malware samples, and exposes more hidden behavior in these samples.

ACKNOWLEDGEMENTS

We thank the anonymous reviewers for their feedback. This work was supported by EKZ through the AVR "Memo" project and by NWO through projects "TROPICS", "Theseus", and "INTERSECT".

REFERENCES

[1] [n. d.]. *Al-Khaser.* Retrieved January 31st 2023 from https://github.com/LordNoteworthy/al-khaser

[2] [n. d.]. *sems.* Retrieved January 31st 2023 from https://github.com/AlicanAkyol/sems

[3] [n. d.]. *VMDE.* Retrieved January 31st 2023 from https://github.com/hfiref0x/VMDE

[4] 2021. *Internet Security Report - Q1 2021.* Available Online: https://www.watchguard.com/wgrd-resource-center/security-report-q1-2021 (Accessed July 8th 2021).

[5] Davide Balzarotti, Marco Cova, Christoph Karlberger, Christopher Kruegel, Engin Kirda, and Giovanni Vigna. 2010. Efficient Detection of Split Personalities in Malware. In *Symposium on Network and Distributed System Security (NDSS).*

[6] David Brumley, Cody Hartwig, Zhenkai Liang, James Newsome, Dawn Song, and Heng Yin. 2008. Automatically Identifying Trigger-based Behavior in Malware. In *Botnet Detection.* Springer, 65–88.

[7] Xu Chen, Jon Andersen, Z. Morley Mao, Michael Bailey, and Jose Nazario. 2008. Towards an Understanding of Anti-virtualization and Anti-debugging Behavior in Modern Malware. In *In Proceedings of the IEEE International Conference on Dependable Systems and Networks With FTCS and DCC (DSN).* IEEE, 177–186.

[8] Microsoft Corporation. [n. d.]. *Detours.* Retrieved January 31st 2023 from https://github.com/microsoft/Detours

[9] Daniele Cono D'Elia, Emilio Coppa, Federico Palmaro, and Lorenzo Cavallaro. 2020. On the Dissection of Evasive Malware. In *IEEE Transactions on Information Forensics and Security 15 (TIFS).* IEEE, 2750–2765.

[10] Artem Dinaburg, Paul Royal, Monirul Sharif, and Wenke Lee. 2008. Ether: Malware Analysis via Hardware Virtualization Extensions. In *Proceedings of the ACM Conference on Computer and Communications Security (CCS).* ACM, 51–62.

[11] Emre Güler, Cornelius Aschermann, Ali Abbasi, and Thorsten Holz. 2019. ANTIFUZZ: Impeding Fuzzing Audits of Binary Executables. In *28th USENIX Security Symposium.* USENIX, 1931–1947.

[12] Jinho Jung, Hong Hu, David Solodukhin, Daniel Pagan, Kyu Hyung Lee, and Taesoo Kim. 2019. Fuzzification: Anti-Fuzzing Techniques. In *28th USENIX Security Symposium.* USENIX, 1913–1930.

[13] Min Gyung Kang, Heng Yin, Steve Hanna, Stephen McCamant, and Dawn Song. 2009. Emulating Emulation-Resistant Malware. In *Proceedings of the 1st ACM workshop on Virtual machine security (VMSec).* ACM, 11–22.

[14] Dhilung Kirat and Giovanni Vigna. 2015. MalGene: Automatic Extraction of Malware Analysis Evasion Signature. In *Proceedings of the 22nd ACM SIGSAC Conference on Computer and Communications Security (CCS).* ACM, 769–780.

[15] Dhilung Kirat, Giovanni Vigna, and Christopher Kruegel. 2014. BareCloud: Bare-metal Analysis-based Evasive Malware Detection. In *23rd USENIX Security Symposium.* USENIX, 287–301.

[16] Raman Ladutska. [n. d.]. *CheckPointSW.* Retrieved January 31st 2023 from https://evasions.checkpoint.com

[17] Kevin Leach, Chad Spenksy, Westley Weimer, and Fengwei Zhang. 2016. Towards Transparent Introspection. In *IEEE 23rd International Conference on Software Analysis, Evolution, and Reengineering (SANER).* IEEE, 248–259.

[18] Martina Lindorfer, Clemens Kolbitsch, and Paolo Milani Comparetti. 2011. Detecting Environment-Sensitive Malware. In *International Workshop on Recent Advances in Intrusion Detection (RAID).* Springer, 338–357.

[19] Andreas Moser, Cristopher Kruegel, and Ergin Kirda. 2007. Exploring Multiple Execution Paths for Malware Analysis. In *IEEE Symposium on Security and Privacy (S&P).* IEEE, 231–245.

[20] Alberto Ortega. [n. d.]. *Paranoid Fish.* Retrieved January 31st 2023 from https://github.com/a0rtega/pafish

[21] Fei Peng, Zhui Deng, Xiangyu Zhang, Dongyan Xu, Zhiqiang Lin, and Zhendong Su. 2014. X-Force: Force-Executing Binary Programs for Security Applications. In *23rd USENIX Security Symposium.* USENIX, 829–844.

[22] Mario Polino, Andrea Continella, Sebastiano Mariani, Stefano D'Alessio, Lorenzo Fontana, Fabio Gritti, and Stefano Zanero. 2017. Measuring and Defeating Anti-Instrumentation-Equipped Malware. In *International Conference on Detection of Intrusions and Malware, and Vulnerability Assessment (DIMVA).* Springer, 73–96.

[23] Siegfried Rasthofer, Steven Arzt, Stefan Triller, and Michael Pradel. 2017. Making Malory Behave Maliciously: Targeted Fuzzing of Android Execution Environments. In *IEEE/ACM 39th International Conference on Software Engineering (ICSE).* IEEE, 300–311.

[24] Joe Sandbox. [n. d.]. *Analysis Report Neshta virus.com.* Retrieved January 31st 2023 from https://www.joesandbox.com/analysis/305163/0/html

[25] Hao Shi and Jelena Mirkovic. 2017. Hiding Debuggers from Malware with Apate. In *Proceedings of the Symposium on Applied Computing (SAC).* ACM, 1703–1710.

[26] Hao Shi, Jelena Mirkovic, and Abdulla Alwabel. 2017. Handling Anti-Virtual Machine Techniques in Malicious Software. In *ACM Transactions on Privacy and Security (TOPS).* ACM, 1–31.

[27] Chad Spensky, Hongyi Hu, and Kevin Leach. 2016. LO-PHI: Low-Observable Physical Host Instrumentation for Malware Analysis. In *Symposium on Network and Distributed System Security (NDSS).*

[28] Xabier Ugarte-Pedrero, Davide Balzarotti, Igor Santos, , and Pablo G. Bringas. 2016. RAMBO: Run-time packer Analysis with Multiple Branch Observation. In *International Conference on Detection of Intrusions and Malware, and Vulnerability Assessment (DIMVA).* Springer, 186–206.

[29] VirusTotal. [n. d.]. *VirusTotal API v3 Overview.* Retrieved January 31st 2023 from https://developers.virustotal.com/v3.0/reference#files

[30] Xiaolei Wang, YueXiang Yang, and Sencun Zhu. 2018. Automated Hybrid Analysis of Android Malware Through Augmenting Fuzzing With Forced Execution. In *IEEE Transactions on Mobile Computing (TMC).* IEEE, 2768–2782.

[31] Jeffrey Wilhelm and Tzicker Chiueh. 2007. A Forced Sampled Execution Approach to Kernel Rootkit Identification. In *International Workshop on Recent Advances in Intrusion Detection (RAID).* Springer, 219–235.

[32] Zhaoyan Xu, Jialong Zhang, Guofei Gu, , and Zhiqiang Lin. 2014. GOLDENEYE: Efficiently and Effectively Unveiling Malware's Targeted Environment. In *International Workshop on Recent Advances in Intrusion Detection (RAID).* Springer, 22–45.

[33] Zhaoyan Xu, Jialong Zhang, Guofei Gu, and Zhiqiang Lin. 2013. AUTOVAC: Towards Automatically Extracting System Resource Constraints and Generating Vaccines for Malware Immunization. In *IEEE 33rd International Conference on Distributed Computing Systems (ICDCS).* IEEE, 112–123.

[34] Fengwei Zhang, Kevin Leach, Angelos Stavrou, Haining Wang, and Kun Sun. 2015. Using Hardware Features for Increased Debugging Transparency. In *IEEE Symposium on Security and Privacy (S&P).* IEEE, 55–69.

[35] Fengwei Zhang, Kevin Leach, Kun Sun, and Angelos Stavrou. 2013. SPECTRE: A Dependable Introspection Framework via System Management Mode. In *43rd Annual IEEE/IFIP International Conference on Dependable Systems and Networks (DSN).* IEEE, 1–12.

[36] Jialong Zhang, Zhongshu Gu, Jiyong Jang, Dhilung Kirat, Marc Ph. Stoecklin, Xiaokui Shu, and Heqing Huang. 2020. Scarecrow: Deactivating Evasive Malware via Its Own Evasive Logic. In *50th Annual IEEE/IFIP International Conference on Dependable Systems and Networks (DSN).* IEEE, 76–87.

[37] Lei Zhou, Jidong Xiao, Kevin Leach, Westley Weimer, Fengwei Zhang, and Guojun Wang. 2019. Nighthawk: Transparent System Introspection from Ring -3. In *European Symposium on Research in Computer Security (ESORICS).* Springer, 217–238.