CLOUDOSCOPE: Detecting Anti-Forensic Malware using Public Cloud Environments

Mordechai Guri
gurim@post.bgu.ac.il
Ben-Gurion University of the Negev
Beer-Sheva, Israel

ABSTRACT

Many modern malware employs runtime anti-forensic techniques in order to evade detection. Anti-forensic tactics can be categorized as anti-virtualization (anti-VM), anti-debugging, anti-sandbox, and anti forensic-tools. The detection of such malware is challenging since they do not reveal their malicious behavior and are therefore considered benign.

We present CLOUDOSCOPE, a novel architecture for detecting anti-forensic malware using the power of public cloud environments. The method we use involves running samples on bare metal machines, then running and monitoring them in multiple forensic environments deployed in the cloud. That includes virtual machines, debugging, sandboxes, and forensic environments. We identify anti-forensic behavior by comparing results in forensic and non-forensic environments. Anti-forensic malware would expose a difference between bare-metal, non-forensic, and virtualized forensic executions. Furthermore, our method enables the identification of the specific anti-forensic technique(s) used by the malware. We provide background on anti-forensic malware, present the architecture, design and implementation of CLOUDOSCOPE, and the evaluation of our system. Public cloud environments can be used to identify and detect stealthy, anti-forensic malware, as shown in our evaluation.

CCS CONCEPTS

• **Computer systems organization** → **Embedded systems**; *Cyber-Security*; Malware analysis; • **Networks** → Cloud.

KEYWORDS

Anti-forensic, evasion, anti-VM, anti-debug, anti-sandbox, malware, APT, public cloud, detection

ACM Reference Format:
Mordechai Guri. 2023. CLOUDOSCOPE: Detecting Anti-Forensic Malware using Public Cloud Environments. In *European Interdisciplinary Cybersecurity Conference (EICC 2023), June 14–15, 2023, Stavanger, Norway*. ACM, New York, NY, USA, 8 pages. https://doi.org/10.1145/3590777.3590793

1 INTRODUCTION

The ongoing arms race between attackers and defenders constantly requires both sides to develop new tools and techniques to gain an advantage over the other. As a result, attackers constantly evolve sophisticated forms of malware, social engineering tactics, and creative attack techniques to bypass security defenses and gain unauthorized access to sensitive information. On the other hand, defendants are developing and implementing enhanced security technologies and strategies to detect and prevent these attacks. This includes antivirus (AV) software, intrusion detection and prevention systems (IDS/IPS), security best practices and policies, automated patch management systems, and user awareness training.

Meanwhile, the evasive techniques used by attackers have proven to be highly successful, and security firms report millions of new malware variants every year [10][11]. For instance, previous research showed that by changing a small number of bytes in the original malware executable, a sample could be rendered undetected by security products [38].

1.1 Evasive Malware

Evasive malware refers to a class of malware designed to evade detection. Such malware might include code encryption [51], data hiding [41], system calls obfuscation [39], polymorphic code [29], and metamorphic engines [46]. Malware authors often use these techniques to bypass AV scanners and cover their tracks. Additionally, it makes it difficult for forensic investigators, security experts, and even experienced malware analysts to detect the attack and deploy the necessary mitigation.

1.2 Anti-Forensic Malware

Anti-forensic malware is a special type of evasive malware specifically designed to bypass forensic analysis, statically or at runtime, [50]. Anti-forensic malware tries to identify whether they are executed in a forensic environment, such as under a debugger, in a virtual machine, or inside a sandbox and changes their behavior accordingly. These techniques pose a significant challenge to security tools and malware analysts and require high levels of expertise and time-consuming, step-by-step manual inspection.

Advanced persistent threats (APT) commonly employ anti-forensic techniques to remain stealthy and evade detection. For example, Remcos malware detects whether it runs on VMware or Sandboxie to alter its behavior [17]. The SUNBURST malicious DLL used in the Orion SolarWinds attack in 2020 examined the host's domain name to check whether it operated in a natural environment [19]. Similarly, FinFisher [2], and Darkhotel [21] APTs employ virtual machine and sandbox detection mechanisms by checking hardware resources and file names to determine whether they are being analyzed. Ransomware often employs anti-forensic techniques as well. TeslaCrypt and Locky ransomware detect whether they are running inside a virtual machine or under a debugger and terminate their execution accordingly [9].

EICC 2023, June 14–15, 2023, Stavanger, Norway

ACM ISBN 978-1-4503-9829-9/23/06.
https://doi.org/10.1145/3590777.3590793

1.3 Anti-Forensic Challenge

Detecting anti-forensic malware is particularly challenging since they are designed to remain stealthy and bypass security products. The predominant strategy to counter anti-forensic malware today is hiding the markers of forensic environments, such as eliminating virtual machine and sandbox artifacts or running debuggers in hidden mode. However, these approaches are proven to have only a limited effect and can be easily bypassed if the attacker knows them [40][32][34][49][31].

1.4 Our Contribution

We present CLOUDOSCOPE, a system that allows the detection of anti-forensic malware using the power of public cloud environments. Our method involves running samples on bare metal machines and then monitoring the samples in a variety of cloud-based forensic environments, including virtual machines, sandboxes, debuggers, and forensic environments. We identify anti-forensic behavior by comparing the results of multiple executions. We present the design, implementation, and evaluation of CLOUDOSCOPE as a generic, cloud-based solution for detecting anti-forensic malware. To the best of our knowledge, this work presents the first architecture that uses public cloud capabilities to counter anti-forensic attacks.

2 ANTI-FORENSIC TECHNIQUES

The anti-forensic techniques are classified into two main categories; (1) those that bypass static analysis tools and malware scanners and (2) those that evade detection and security products at runtime.
Static anti-forensic techniques consist of methods that aim to challenge security tools when a sample is statically examined. This also includes strategies for preventing or delaying the successful reverse engineering of malware. Some static techniques are anti-disassembly [35], code obfuscation [55], stenography [41], and code encryption [51]. Static anti-forensic techniques are considered less challenging to mitigate since they can be resolved at runtime [50]. For example, malware may employ anti-disassembly to hide the actual rough code from the disassembler. However, the real code will always be exposed to the reverse engineer at runtime when it is being debugged. In another example, the malware uses code encryption to hide the use of suspicious API functions and evade anti-virus scanners. As in the previous case, these calls can be intercepted and revealed at runtime when the actual API function call is executed.

Due to the inherent weakness of static anti-forensic techniques, malware authors tend to move to runtime evasion [50]. In this approach, malware may employ a set of techniques at runtime in order to evade detection, security tools, and forensic investigation. In particular, malicious code may be designed to change its behavior or content at runtime, making it difficult to detect and analyze.

2.1 Runtime Anti-Forensic Techniques

Runtime Anti-forensic techniques aim to detect the existence of forensic environments while executing them to evade them. The malware will take one or more of the following strategies if such an environment setting is identified; (1) kill itself and disappear, (2) hide the malicious behavior by eliminating certain activities - to avoid detection, (3) sleep for a particular period or (4) exploit vulnerabilities to circumvent the security product.

There are four main categories of anti-forensic techniques at runtime.

2.1.1 Anti-VM. Anti-VM (Virtual Machine) techniques are designed to detect when a program runs in a virtual environment, such as within a virtual machine (VM). Technically, malware determines whether it executes on bare metal, on top of a Virtual Machine Monitor (VMM), or on a hypervisor and changes its behavior accordingly. Malware analysts widely use virtual machines due to their ability to prevent malware from spreading and their easy backup, revert, and rollback capabilities. Anti-VM techniques are difficult to detect because they can avoid detection by security tools that are specifically designed to run in virtual environments. There are many ways to identify virtual machines and hypervisors based on virtualization artifacts, using special instructions, timing measurements, and operating system markers [27].

2.1.2 Anti-debugging. Dynamic analysis of malware consists of debugging the sample with debuggers. Malware uses anti-debugging and anti-tampering techniques to detect and evade debuggers. When a debugger is detected during execution, malware may alter its behavior. Debuggers can be detected using specific API calls (e.g., the `IsDebuggerPresent()` Win32 API [8]), checking operating system structures, timing measurements, and searching for breakpoint instructions in the code [27][50].

2.1.3 Anti-sandbox. Sandboxes monitor malware in a controlled environment to determine its behavior and impact on a system. Various parameters of the process are monitored. Network activity of the process is traced and logged, including inbound and outbound connections, DNS requests, and so on. Filesystem activities, reading/writing from/to files, registry operations (in Microsoft Windows OS). Processes and threads creation and termination are also monitored. Many sandboxes monitor API calls of a process to determine a suspicious sequence of API calls. Sandboxes automate malware analysis on a per-sample basis. Commercial sandboxes are deployed in the form of a gateway that can scan bulk samples and grade them as benign or suspicious. Malware can detect whether it executes in a sandboxed environment. If a sandbox is detected, malware alters its behavior. Another tactic is that malware sleeps or performs dummy operations for some time. Since sandbox scanners terminate the process after a timeout, the sample will pass monitoring and be classified as benign. Malware can detect sandboxes by checking artifacts such as drivers and user names, product keys, injected DLL names, and environmental settings [50].

2.1.4 Forensic tools detection. Malware forensics experts commonly use a variety of tools. These tools include process monitor tools such as process exploiter and process monitor, PE file analysis, and debuggers [30]. As part of dynamic analysis, these tools are used to monitor and log malware's impact on the system. Malware may search for the existence of such tools in the environment, deduce that it runs in a forensic environment, and behave accordingly. Detecting forensic tools is done via scanning files, searching for certain processes' names, and looking for specific registry files [50].

Notably, many types of malware today use anti-forensic techniques, including advanced persistent threats (APTs), ransomware,

Table 1: Anti-forensic techniques

Method	Techniques	Mitigation
Static anti-forensic	Anti-disassembly, code obfuscation, hiding and stenography, code encryption	Dynamic analysis
Dynamic anti-forensic	Anti-virtualization (anti-VM), anti-debugging , anti-sandbox, forensic tools detection	Dynamic analysis+ anti-anti-forensic

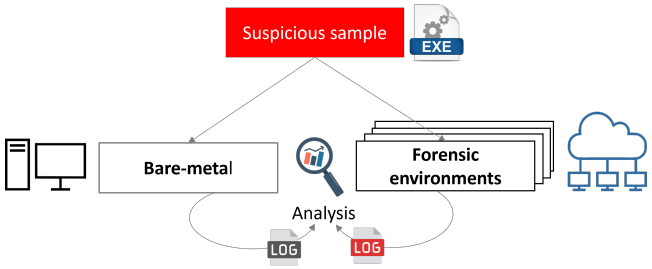

Figure 1: Suspicious samples are run on bare-metal and in the cloud on multiple machines. Trace logs are collected, indexed, and analyzed to determine whether anti-forensic techniques were used.

rootkits, fileless attacks, and exploit kits. The most common anti-forensic techniques used by malware are summarized in Table 1.

3 DESIGN AND IMPLEMENTATION

Anti-forensic malware research focuses mainly on finding new anti-forensic, especially anti-VM and anti-debugging techniques, either manually or by automated tools [32][50]. Researchers previously described methods for locating anti-forensic code via static analysis and countermeasures that prevented them from being effective during dynamic analysis. For example, it is possible to configure a VM and hide its virtualization artifacts to reduce its detectability [28]. Some debuggers provide special capabilities to counter and evade anti-debugging techniques [48].

Our approach is based on monitoring the execution of a suspicious sample in a bare-metal environment in a controlled and monitored manner. Concurrently, we deploy multiple forensic environments in the cloud and examine the malware effect on the forensic systems. The core of our detection technique is based on the fact that a benign sample would behave the same way in both setups; forensic environments on the cloud and non-forensic, bare metal environments. In order to create multiple forensic setups, we utilized the capability of public clouds to provide highly manageable environments with different operating systems, applications, and frameworks.

The architecture of our system is presented in figure 1. The suspicious samples are executed on bare-metal machines and on multiple heterogeneous forensic machines in the public cloud. The trace logs are collected and then gathered into a database. Finally, the traces are analyzed to identify differences at which points the sample employs anti-forensic techniques.

3.1 Bare-Metal

The sample must be run on a bare metal machine as the base reference point. Technically, the sample is executed on a physical computer that runs an operating system directly on the physical hardware, without intermediate layers like a hypervisor or virtual machine monitors. This execution provides traces of the process in a non-forensic environment. The sample mustn't be executed on any hypervisor to neutralize its anti-VM code if it exists.

One of the main challenges of running the sample on bare metal is the lack of advantages virtual machines offer, which are crucial to malware analysis. In virtual machines, for instance, the operating system is started from a 'clean' state at every reboot to eliminate any persistent effects of the malware analyzed. Virtual machines are also capable of taking disk and memory snapshots at certain checkpoints. Finally, virtual machines allow controlling the network settings at a logical layer using the virtual network interface card (NIC). It can be used to maintain inbound and outbound traffic, redirect or block IP addresses, and disconnect machines, creating a virtual air gap.

To enable these features, we used Faronics Deep Freeze [5], which allows us to restore the computer to its checkpoint configuration at each reboot. Note that this type of solution (and other commercial alternatives) does not use virtualization and hence won't trigger the anti-VM mechanisms. To control the network, we use traffic control at the physical router. It enables us to maintain inbound and outbound traffic or disconnect the machine from the Internet.

3.2 Bare-Metal on Cloud

The public cloud provider uses the virtualization layer to manage virtual machines, hardware, computation, and network resources as part of its infrastructure. That prevents using the public cloud for bare metal monitoring due to the persistent virtualization layer in the underlying infrastructure, which would trigger the anti-VM mechanisms of the sample under analysis. However, during 2021/2, the major cloud providers started providing so-called 'bare metal instances' in the cloud, allowing customers to manage services on bare metal servers without a virtualization layer. In particular, bare metal in the cloud enables the use of technologies that aren't fit to run in a typical virtualized setting. This restriction might be due to performance issues, the need for direct hardware access, and so on. Some of the bare metal services in the public cloud are listed below.

- Azure 'BareMetal' Infrastructure [23].
- Amazon EC2 bare metal instances [6].
- Google Cloud bare metal instances [1].

With this brand-new type of infrastructure, the cloud providers enable the customer with direct access to the hardware and resources of the underlying machines while eliminating the hypervisor layer.

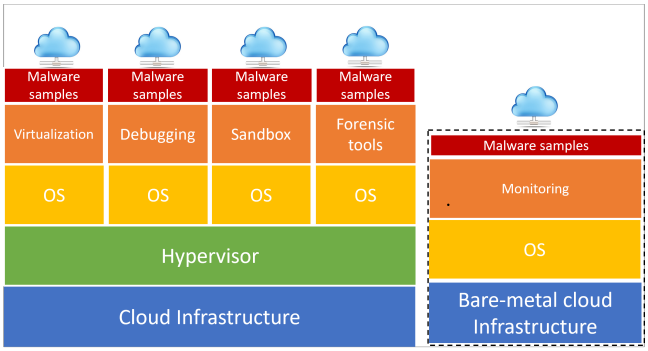

Figure 2: The system architecture with on-cloud bare-metal infrastructure.

This service allows the management of bare metal machines in the cloud and preconfigures them with non-virtualized system restoration solutions such as Deep Freeze. Specifically, it enables to implementation of bare metal malware monitoring in public clouds, as shown in Figure 2.

3.3 Forensic Environments on Cloud

CLOUDOSCOPE architecture consists of multiple forensic environments.

3.3.1 Virtual Machines/hypervisors. There are three main types of virtual machine monitors that malware targets in their anti-VM checks.

- VMWare [22]. VMware is the hypervisor most targeted by Anti-VM techniques. We deployed a machine with a VMware workstation, which is a type-2 hypervisor, to represent a workstation with a virtual machine. Note that type-2 hypervisors are more commonly targeted by malware than type-1 hypervisors, which are mainly used on servers. We also tested a machine with VMWare ESXi type-1 hypervisors in an enterprise-class virtual environment. Note that most commonly, anti-VM techniques don't target a specific type of VMWare hypervisor but are relevant to both types.
- Oracle VM VirtualBox [12]. This is a type-2 hypervisor for Intel/AMD virtualization developed by Oracle Corporation. VirtualBox is widely used for user-grade virtualization on workstations for malware analysis. Many anti-VM malware programs target this VMM specifically in their VM detection.
- Microsoft Hyper-V [7]. This is a type-1 hypervisor that is installed on bare-metal servers or the Windows 10 operating system. Hyper-V supports different operating systems, including FreeBSD, Linux, and Windows.

3.3.2 Debuggers. To simulate a debugging environment, we used Windbg [24], a debugger developed by Microsoft for Windows operating systems. It supports both user-mode and kernel-mode debugging and provides memory dump analysis, symbol loading, source-level debugging, and remote debugging capabilities. We also tested the x64xdbg [25], an open-source x64/x32 debugger for Windows OS with command-line capabilities. Note that anti-debugging techniques are usually resilient to the specific debugger type and only check whether the process runs under any debugger.

Table 2: The trace logs representation matrix

Environment	Files	Registry	Network	Process/threads	API
Bare-metal	BM_f	BM_r	BM_n	BM_p	BM_c
VM/Hypervisor	VM_f	VM_r	VM_n	VM_p	VM_c
Debugging	DB_f	DB_r	DB_n	DB_p	DB_c
Sandbox	SB_f	SB_r	SB_n	SB_p	SB_c
Forensic tools	FT_f	FT_r	FT_n	FT_p	FT_c

3.3.3 Sandboxes. To emulate a sandbox environment, we operated Cuckoo Sandbox [13], an open-source software for automated malware analysis. This dynamic malware analysis tool allows analysts to analyze malicious software in a controlled and isolated environment. For analysis, sandboxes log system activity, network traffic, and other relevant information.

3.3.4 Forensic tools. We deployed machines representing the workstation of a forensic investigator. It runs the Sysinternals ProcMon [15], ProcessExplorer [14], HexRay IDA Pro [4], and RegShot [16] utilities that forensic investigators commonly use.

3.4 System Monitoring

Our architecture is based on comparing the effect of the process on the system. We monitor the system at different levels using the Microsoft Sysinternals monitoring tools [20] and store the following trace logs.

- File-system activity. The activity in the file system includes information about file creation, deletion, modification, and access. The logs include file information and the time of the event.
- Registry operations. The system registry operations, key and value creation, deletion, modification, and access. The logs include key/value information and the time of the event.
- Network operations. The network activity, including outbound and inbound communication in layers 2/3. The logs include DNS, TCP, UDP, IP addresses, and port numbers.
- Process/threads. The creation and termination of processes and threads in the system.
- API calls. Win32 API calls made by a process. This parameter is optional and can be disabled. It can be used to monitor specific API calls in `kernel32.dll` when the files, registry, network, and process/thread trace logs do not provide sufficient information.

4 DATA COLLECTION AND ANALYSIS

All the trace logs are collected in a database. We represent the trace logs in a matrix ('representation matrix') pointing to a set of logs for each process, as presented in Table 2. After the traces are collected and indexed, the data analysis script is executed, which checks whether the suspicious sample employs anti-forensic mechanisms.

Algorithm 1 outline the main function `DetectAF()`. This function receives five documents representing the event logs in each

parameter; files (f), registry (r), network (n), processes/threads (p), and API monitor (c). Note that the event logs are received for each environment; bare metal (BM), Virtual Machine (VM), debuggers (DB), Sandbox (SB), and forensic tools (FT).

For each type of event log, the algorithm compares the bare metal event vector with the event vector of a specific forensic environment. Suppose the event difference is greater than a threshold of T_d. In that case, it indicates that the sample behaves differently in a forensic environment, and therefore anti-forensic techniques were employed. We set the T_d to be 5% to eliminate false positives. This is because execution trace commonly displays natural (non-malicious) minor differences due to execution on different systems. This could result from varying timing measurements, synchronization issues, OS versioning, and environmental configuration. The `DetectAF()` function returns a vector indicating whether anti-forensic was identified for each of the five examined anti-forensic techniques $\langle AntiVM, AntiDebug, AntiSandbox, AntiForensicTools\rangle$.

Table 3 shows a trace log representation matrix for an anti-forensic sample. The API calls row is eliminated in this example due to space constraints. In this case, Anti-VM and Anti-Sandbox techniques are detected. As can be observed, in virtualization and sandbox environments, the effect on the system is minimal; no files, network, and registry activity and single process/thread creation and termination. This is since VM and sandbox detection was used in this case.

Algorithm 1 DetectAF(BM,VM,DB,SB,FT)

$BM_a = \langle BM_f, BM_r, BM_n, BM_p, BM_c\rangle$
$VM_a = \langle VM_f, VM_r, VM_n, VM_p, VM_c\rangle$
$DB_a = \langle DB_f, DB_r, DB_n, DB_p, DB_c\rangle$
$SB_a = \langle SB_f, SB_r, SB_n, SB_p, SB_c\rangle$
$FT_a = \langle FT_f, FT_r, FT_n, FT_p, FT_c\rangle$
$AntiVM, AntiDebug, AntiSandbox, AntiForensicTools = 0$
if $(|VM_a - BM_a|) \geq T_n$ **then**
    $AntiVM = true$
end if
if $(|DB_a - BM_a|) \geq T_n$ **then**
    $AntiDebug = true$
end if
if $(|SB_a - BM_a|) \geq T_n$ **then**
    $AntiSandbox = true$
end if
if $(|FT_a - BM_a|) \geq T_n$ **then**
    $AntiForensicTools = true$
end if
$v = \langle AntiVM, AntiDebug, AntiSandbox, AntiForensicTools\rangle$
return v

5 EVALUATION

We evaluated the system with 30 malware samples (executable files) known to employ anti-forensic techniques. The results are presented in Table 4. Twenty-three samples were identified as with anti-VM, 7 with anti-debugging, and a single sample employed anti-sandbox (in addition to anti-VM).

We also generated anti-forensic scripts to test our system against a set of anti-forensic techniques. Our tests were based on Pafish [3], an open-source tool that can identify virtual machines, debuggers, and specific forensic environments like Sandboxie, Wine, VMware, Bochs, and Cuckoo Sandbox. Pafish uses methods such as timing checks via `GetTickCount()` Win32 API, debugger detection using process environment block (PEB) information, thread local storage (TLS) hooks, kernel drivers information, and so on.

We implemented anti-forensic scripts based on Pafish to mimic the behavior of anti-forensic malware artificially. The script works as follows: If a VM, debugger, or sandbox is found, terminate; otherwise, perform random I/O operations (writing files and initiating outbound connections) and terminate. Since Pafish doesn't provide forensic tools detection functionalities, we added a specific function to detect the existence of HexRay IDA Pro in the system by relying on the process name.

5.1 False Positives and Limitations

There are types of benign applications that behave differently in bare metal and forensic environments. For example, debuggers commonly check for other debuggers in the system via the Win32 API, such as `IsDebuggerPresent()` and its variations. In addition, some applications don't run properly on virtual machines or within a nested virtualization environment. It is usually appended when direct access to the underlined hardware is required. Some examples are provided below.

- Graphics-intensive applications such as video editing programs, video games, and complex CAD/CAM applications.
- Defragmentation and partitioning applications that manipulate disk images.
- Applications that demand strict real-time performance.
- Applications that require processor-level capabilities, such as TPMs, and software enclaves.
- Other hypervisors that may not support nested virtualization.

These applications might be incorrectly identified as malicious and sent for further dynamic analysis. In these cases, the applications behaved differently in a bare metal environment and were identified as anti-forensic. In the context of malware detection, the solution is to maintain a whitelist of benign forensic-aware applications. These applications would be excluded from anti-forensic checking according to their hash or signature values.

5.2 False Negatives

False-negative results may occur when the investigated sample is not identified as anti-forensic. For example, attackers may delay malicious operations to avoid (any) forensic investigation. This is especially relevant in automated analysis environments where malware sleeps for a certain amount of time via functions such as the Win32 API function `Sleep()` [18] or by using dummy operations. These techniques enable the malware under investigation to reach the sandbox timeout without detection. Previous work proposed several approaches to mitigate such evasion techniques, e.g., monitoring specific sleeping system calls or virtualizing the system clocks [26]. In the context of our defense, the detection

Table 3: A trace log representation matrix (partial) for an anti-forensic sample. In this case, Anti-VM and Anti-Sandbox techniques are detected.

Environment	Files	Registry	Network	Process/threads
Bare-metal	read: 5, write: 4	read: 9, write: 3	TCP connections: 2	creation:3, termination:1
VM/Hypervisor	read: 0, write: 0	read: 0, write: 0	TCP connections: 0	creation:1, termination:1
Debugging	read: 5, write: 4	read: 9, write: 3	TCP connections: 2	creation:3, termination:1
Sandbox	read: 0, write: 0	read: 0, write: 0	TCP connections: 0	creation:1, termination:1
Forensic tools	read: 5, write: 4	read: 9, write: 3	TCP connections: 2	creation:3, termination:1

Table 4: Evaluation results with 30 anti-forensic samples

Environment	Detected
VM/Hypervisor	23 (23 samples used anti-VM)
Debugging	7 (7 samples used anti-debugging)
Sandbox	1 (1 samples used anti-sandbox)
Forensic tools	0 (0 samples used anti forensic-tools)

systems will be combined in the CLOUDOSCOPE architecture to identify samples that employ evasion techniques.

6 DISCUSSION

Anti-forensic malware are known to be challenging in the domain of malware analysis at both practical and theoretical levels.

Branco et al. presented a non-academic overview of anti-forensic malware and provided extended statistics based on a database of 4 million samples [27]. Chen et al. presented a comparative overview of anti-VM and anti-debugging techniques used by malware authors [31]. More recently, Gallardo et al. [32], and Mafia et al. [40] studied the prevalence of dynamic evasive mechanisms in modern malware. Notably, most of the proposed methods for detecting and mitigating anti-forensic malware focus on specific anti-forensic techniques [42]. Polino et al. introduced a runtime system based on the Intel PIN tool for instrumentation detection. They estimated whether a process uses anti-forensic techniques, such as timing measurements and environmental settings [45]. There are multiple solutions for existing debuggers that try to hide the debugger artifacts from the sample to counter anti-debugging techniques [36]. However, these techniques can be easily bypassed once the attacker knows the stealth technique. Smith introduced an assembly-level solution named REDIR that detects and highlights anti-debugging techniques in the code based on static code analysis [52].

Several previous academic works discussed the concept of enumerating the differences between a given forensic environment and a physical machine. However, the prior work mainly focused on virtual machines and omitted other environments such as debuggers and sandbox [49]. Gilboy introduced DVasion, a framework that can expose evasive behavior using so-called multiple execution environments [33]. The proposed system is based on a monitor in a virtual machine and is not designed to counter other types of anti-forensic techniques. Ding et al. described a way to detect anti-VM capabilities via collected behavioral information and a specified distance algorithm, which estimates the difference between VM and physical machines [53]. Guri et al. introduced a method for testing the so-called split behavior of malware in a generic way [34]; nonetheless, they didn't utilize the power of modern public clouds to manage multiple forensic environments and instead used on-prem servers for their evaluation. Lee et al. presented an empirical study of the top commercial anti-VM and anti-instrumentation tools and provided guidelines to bypass them using traces modules [37].

Note that due to their specified approaches, most of the previous research works can detect anti-VM techniques, and some are extended to anti-debugging [33] [37]. Genetic approaches taken by [34] have been evaluated for their anti-VM and anti-debugging capabilities and are theoretically capable of working against anti-sandbox malware, but no evaluation has been conducted for their anti-sandbox and anti-forensic tools capabilities. Our work is relevant to all four anti-forensic techniques and was evaluated on them. The previous work and their relevancy are summarized in Table 5.

6.1 Public clouds vs. existing approaches

In addition to the contribution mentioned above, CLOUDOSCOPE is the first architecture that uses public cloud capabilities to perform manageable, extensible, and practical detection of anti-forensic malware generically. In addition, we propose to use the newly introduced bare metal infrastructure provided by major cloud providers today to evaluate the sample in the cloud environment. There are several advantages of using cloud environments over previous approaches, in terms of engineering and practical aspects. The main benefits of the cloud environment to the CLOUDOSCOPE architecture are listed below.

- **Deployment.** The approach requires deployments of tens of runtime environments with multiple OS, hypervisors, debuggers, and forensic tools. It is possible in the public cloud environment via VM deployment tools all major cloud providers offer [54].
- **Automation.** The approach requires automation of multiple environments to upload, execute, log, and analyze. Such automation is supported by the cloud infrastructure available today [45].
- **Orchestration.** The orchestration of forensic environments, such as initiating, taking memory snapshots, and terminating the system under investigation, is an integral part of modern clouds [43].
- **Scalability.** The cloud architecture enables greater scalability of the solution, which is one of the crucial aspects in automated malware analysis; execution of multiple samples concurrently on cloned environments is possible and greatly supported by the public cloud vendors [47].

Table 5: Previous work comparison

#	Work	Technique	VM	Debugging	Sandbox	Tools	Proposed environment
2011	Pek et al. [44]	Timing and CPU virtualization	YES	NO	NO	NO	On-prem, Xen monitor
2011	Sun et al. [53]	Process behavioral analysis	YES	NO	NO	NO	On-prem, VMware monitor
2013	Guri et al. [34]	API and CPU traces comparisons	YES	YES	YES	NO	On-prem, Hypervisors
2016	Gilboy et al. [33]	Dynamic binary instrumentation	YES	YES	NO	NO	On-prem, Intel PIN
2021	Lee et al. [37]	API/instruction traces	YES	YES (partial)	NO	NO	On-prem, Intel PIN
2023	CLOUDOSCOPE	Public cloud multiple environments	YES	YES	YES	YES	Cloud

This makes CLOUDOSCOPE a more practical, automated and scalable solution compared to all on-prem approaches presented so far.

7 CONCLUSION

To remain undetected, malware authors arm their code with anti-forensic mechanisms. It includes anti-VMs, anti-debuggers, anti-sandbox evasion, and forensic tools detection. These techniques enabled malware to remain stealthy and proved highly efficient against Anti-Virus products, automated sandboxes, intrusion detection systems, and even manual forensic investigators. We present CLOUDOSCOPE, an architecture that enables the detection of anti-forensic malware by using the capabilities of public clouds. Our approach is based on the fact that anti-forensic malware behaves differently in regular and forensic settings. We presented an architecture for deploying heterogeneous forensic environments in the cloud. With the aggregated data, we analyze bare-metal and forensic execution traces to identify anti-forensic malware's operation. We also showed that this technique could detect specific anti-forensic methods, even where they were unknown in advance. We presented related work, an overview of anti-forensic methods, the design and architecture of the system, and experimental results.

REFERENCES

[1] [n. d.]. Egressing from Google Bare Metal Solution | by Ben King | Google Cloud - Community | Medium. https://medium.com/google-cloud/egressing-from-google-bare-metal-solution-aa459389436c. (Accessed on 02/03/2023).

[2] [n. d.]. FinFisher exposed: A researcherâ€™s tale of defeating traps, tricks, and complex virtual machines - Microsoft Security Blog. https://www.microsoft.com/en-us/security/blog/2018/03/01/finfisher-exposed-a-researchers-tale-of-defeating-traps-tricks-and-complex-virtual-machines/. (Accessed on 02/03/2023).

[3] [n. d.]. GitHub - a0rtega/pafish: Pafish is a testing tool that uses different techniques to detect virtual machines and malware analysis environments in the same way that malware families do. https://github.com/a0rtega/pafish. (Accessed on 02/03/2023).

[4] [n. d.]. Hex Rays - State-of-the-art binary code analysis solutions. https://hex-rays.com/ida-pro/. (Accessed on 02/03/2023).

[5] [n. d.]. Instant System Restore Software for Multiple Computers | Deep Freeze Enterprise. https://www.faronics.com/en-uk/products/deep-freeze/enterprise. (Accessed on 02/03/2023).

[6] [n. d.]. Introducing two new Amazon EC2 bare metal instances. https://aws.amazon.com/about-aws/whats-new/2021/11/amazon-ec2-bare-metal-instances/. (Accessed on 02/03/2023).

[7] [n. d.]. Introduction to Hyper-V on Windows 10 Microsoft Learn. https://learn.microsoft.com/en-us/virtualization/hyper-v-on-windows/about/. (Accessed on 02/03/2023).

[8] [n. d.]. IsDebuggerPresent function (debugapi.h) - Win32 apps | Microsoft Learn. https://learn.microsoft.com/en-us/windows/win32/api/debugapi/nf-debugapi-isdebuggerpresent. (Accessed on 02/03/2023).

[9] [n. d.]. Locky ransomware adds anti sandbox feature (updated) | Malwarebytes Labs. https://www.malwarebytes.com/blog/news/2017/08/locky-ransomware-adds-anti-sandbox-feature. (Accessed on 02/03/2023).

[10] [n. d.]. Malware Statistics & Trends Report | AV-TEST. https://www.av-test.org/en/statistics/malware/. (Accessed on 02/02/2023).

[11] [n. d.]. mwb_threatreview_2022_ss_v1.pdf. https://www.malwarebytes.com/resources/malwarebytes-threat-review-2022/mwb_threatreview_2022_ss_v1.pdf. (Accessed on 02/02/2023).

[12] [n. d.]. Oracle VM VirtualBox. https://www.virtualbox.org/. (Accessed on 02/03/2023).

[13] [n. d.]. Privacy error. https://cuckoosandbox.org/. (Accessed on 02/03/2023).

[14] [n. d.]. Process Explorer - Sysinternals | Microsoft Learn. https://learn.microsoft.com/en-us/sysinternals/downloads/process-explorer. (Accessed on 02/03/2023).

[15] [n. d.]. Process Monitor - Sysinternals | Microsoft Learn. https://learn.microsoft.com/en-us/sysinternals/downloads/procmon. (Accessed on 02/03/2023).

[16] [n. d.]. regshot download | SourceForge.net. https://sourceforge.net/projects/regshot/. (Accessed on 02/03/2023).

[17] [n. d.]. Sandbox detection and evasion techniques. How malware has evolved over the last 10 years. https://www.ptsecurity.com/ww-en/analytics/antisandbox-techniques/. (Accessed on 02/03/2023).

[18] [n. d.]. Sleep function (synchapi.h) - Win32 apps | Microsoft Learn. https://learn.microsoft.com/en-us/windows/win32/api/synchapi/nf-synchapi-sleep. (Accessed on 03/09/2023).

[19] [n. d.]. SUNBURST: Attack Flow, C2 Protocol, and Prevention. https://www.cynet.com/attack-techniques-hands-on/sunburst-backdoor-c2-communication-protocol/. (Accessed on 02/03/2023).

[20] [n. d.]. Sysinternals - Sysinternals | Microsoft Learn. https://learn.microsoft.com/en-us/sysinternals/. (Accessed on 02/03/2023).

[21] [n. d.]. Virtualization/Sandbox Evasion, Technique T1497 - Enterprise | MITRE. https://attack.mitre.org/techniques/T1497/. (Accessed on 02/03/2023).

[22] [n. d.]. VMware - Delivering a Digital Foundation For Businesses. https://www.vmware.com/. (Accessed on 02/03/2023).

[23] [n. d.]. What is BareMetal Infrastructure on Azure? - Azure Baremetal Infrastructure | Microsoft Learn. https://learn.microsoft.com/en-us/azure/baremetal-infrastructure/concepts-baremetal-infrastructure-overview. (Accessed on 02/03/2023).

[24] [n. d.]. WinDbg - Wikipedia. https://en.wikipedia.org/wiki/WinDbg. (Accessed on 02/03/2023).

[25] [n. d.]. x64dbg. https://x64dbg.com/. (Accessed on 02/03/2023).

[26] Amir Afianian, Salman Niksefat, Babak Sadeghiyan, and David Baptiste. 2019. Malware dynamic analysis evasion techniques: A survey. *ACM Computing Surveys (CSUR)* 52, 6 (2019), 1–28.

[27] Rodrigo Rubira Branco, Gabriel Negreira Barbosa, and Pedro Drimel Neto. 2012. Scientific but not academical overview of malware anti-debugging, anti-disassembly and anti-vm technologies. *Black Hat* 1, 2012 (2012), 1–27.

[28] Matthew Carpenter, Tom Liston, and Ed Skoudis. 2007. Hiding virtualization from attackers and malware. *IEEE Security & Privacy* 5, 3 (2007), 62–65.

[29] Silvio Cesare, Yang Xiang, and Wanlei Zhou. 2012. Malwiseâ€"an effective and efficient classification system for packed and polymorphic malware. *IEEE Trans. Comput.* 62, 6 (2012), 1193–1206.

[30] S Sibi Chakkaravarthy, D Sangeetha, and V Vaidehi. 2019. A survey on malware analysis and mitigation techniques. *Computer Science Review* 32 (2019), 1–23.

[31] Ping Chen, Christophe Huygens, Lieven Desmet, and Wouter Joosen. 2016. Advanced or not? A comparative study of the use of anti-debugging and anti-VM techniques in generic and targeted malware. In *ICT Systems Security and Privacy Protection: 31st IFIP TC 11 International Conference, SEC 2016, Ghent, Belgium, May 30-June 1, 2016, Proceedings 31*. Springer, 323–336.

[32] Nicola Galloro, Mario Polino, Michele Carminati, Andrea Continella, and Stefano Zanero. 2022. A Systematical and longitudinal study of evasive behaviors in windows malware. *Computers & Security* 113 (2022), 102550.

[33] Matthew Ryan Gilboy. 2016. *Fighting evasive malware with DVasion.* Ph. D. Dissertation. University of Maryland, College Park.

[34] Mordehai Guri, Gabi Kedma, Tom Sela, Buky Carmeli, Amit Rosner, and Yuval Elovici. 2013. Noninvasive detection of anti-forensic malware. In *2013 8th International Conference on Malicious and Unwanted Software:" The Americas"(MALWARE)*. IEEE, 1–10.

[35] Christopher Jämthagen, Patrik Lantz, and Martin Hell. 2013. A new instruction overlapping technique for anti-disassembly and obfuscation of x86 binaries. In *2013 Workshop on Anti-malware Testing Research*. IEEE, 1–9.

[36] Yuhei Kawakoya, Makoto Iwamura, and Mitsutaka Itoh. 2010. Memory behavior-based automatic malware unpacking in stealth debugging environment. In *2010 5th International Conference on Malicious and Unwanted Software*. IEEE, 39–46.

[37] Young Bi Lee, Jae Hyuk Suk, and Dong Hoon Lee. 2021. Bypassing anti-analysis of commercial protector methods using DBI tools. *IEEE Access* 9 (2021), 7655–7673.

[38] Keane Lucas, Mahmood Sharif, Lujo Bauer, Michael K Reiter, and Saurabh Shintre. 2021. Malware makeover: Breaking ml-based static analysis by modifying executable bytes. In *Proceedings of the 2021 ACM Asia Conference on Computer and Communications Security*. 744–758.

[39] Weiqin Ma, Pu Duan, Sanmin Liu, Guofei Gu, and Jyh-Charn Liu. 2012. Shadow attacks: automatically evading system-call-behavior based malware detection. *Journal in Computer Virology* 8 (2012), 1–13.

[40] Lorenzo Maffia, Dario Nisi, Platon Kotzias, Giovanni Lagorio, Simone Aonzo, and Davide Balzarotti. 2021. Longitudinal Study of the Prevalence of Malware Evasive Techniques. *arXiv preprint arXiv:2112.11289* (2021).

[41] Wojciech Mazurczyk and Luca Caviglione. 2015. Information hiding as a challenge for malware detection. *arXiv preprint arXiv:1504.04867* (2015).

[42] Pham Ri Nep and Nguyen Tan Cam. 2022. A Research on Countering Virtual Machine Evasion Techniques of Malware in Dynamic Analysis. In *Intelligent Computing & Optimization: Proceedings of the 5th International Conference on Intelligent Computing and Optimization 2022 (ICO2022)*. Springer, 585–596.

[43] Yao Pan, Ian Chen, Francisco Brasileiro, Glenn Jayaputera, and Richard Sinnott. 2019. A performance comparison of cloud-based container orchestration tools. In *2019 IEEE International Conference on Big Knowledge (ICBK)*. IEEE, 191–198.

[44] Gábor Pék, Boldizsár Bencsáth, and Levente Buttyán. 2011. nEther: In-guest Detection of Out-of-the-guest Malware Analyzers. In *Proceedings of the Fourth European Workshop on System Security*. 1–6.

[45] J Prassanna, AR Pawar, and V Neelanarayanan. 2017. A review of existing cloud automation tools. *Asian J Pharm Clin Res* 10 (2017), 471–473.

[46] Babak Bashari Rad, Maslin Masrom, and Suhaimi Ibrahim. 2012. Camouflage in malware: from encryption to metamorphism. *International Journal of Computer Science and Network Security* 12, 8 (2012), 74–83.

[47] Aaqib Rashid and Amit Chaturvedi. 2019. Cloud computing characteristics and services: a brief review. *International Journal of Computer Sciences and Engineering* 7, 2 (2019), 421–426.

[48] Hao Shi and Jelena Mirkovic. 2017. Hiding debuggers from malware with apate. In *Proceedings of the Symposium on Applied Computing*. 1703–1710.

[49] Hao Shi, Jelena Mirkovic, and Abdulla Alwabel. 2017. Handling anti-virtual machine techniques in malicious software. *ACM Transactions on Privacy and Security (TOPS)* 21, 1 (2017), 1–31.

[50] Michael Sikorski and Andrew Honig. 2012. *Practical malware analysis: the hands-on guide to dissecting malicious software*. no starch press.

[51] Jagsir Singh and Jaswinder Singh. 2018. Challenge of malware analysis: malware obfuscation techniques. *International Journal of Information Security Science* 7, 3 (2018), 100–110.

[52] Adam J Smith, Robert F Mills, Adam R Bryant, Gilbert L Peterson, and Michael R Grimaila. 2014. Redir: Automated static detection of obfuscated anti-debugging techniques. In *2014 International Conference on Collaboration Technologies and Systems (CTS)*. IEEE, 173–180.

[53] Ming-Kung Sun, Mao-Jie Lin, Michael Chang, Chi-Sung Laih, and Hui-Tang Lin. 2011. Malware virtualization-resistant behavior detection. In *2011 IEEE 17th international conference on parallel and distributed systems*. IEEE, 912–917.

[54] Michael Wurster, Uwe Breitenbücher, Michael Falkenthal, Christoph Krieger, Frank Leymann, Karoline Saatkamp, and Jacopo Soldani. 2020. The essential deployment metamodel: a systematic review of deployment automation technologies. *SICS Software-Intensive Cyber-Physical Systems* 35 (2020), 63–75.

[55] Ilsun You and Kangbin Yim. 2010. Malware obfuscation techniques: A brief survey. In *2010 International conference on broadband, wireless computing, communication and applications*. IEEE, 297–300.