Laccolith: Hypervisor-Based Adversary Emulation with Anti-Detection

Vittorio Orbinato, Marco Carlo Feliciano, Domenico Cotroneo, Roberto Natella

Abstract—Advanced Persistent Threats (APTs) represent the most threatening form of attack nowadays since they can stay undetected for a long time. Adversary emulation is a proactive approach for preparing against these attacks. However, adversary emulation tools lack the anti-detection abilities of APTs. We introduce Laccolith, a hypervisor-based solution for adversary emulation with anti-detection to fill this gap. We also present an experimental study to compare Laccolith with MITRE CALDERA, a state-of-the-art solution for adversary emulation, against five popular anti-virus products. We found that CALDERA cannot evade detection, limiting the realism of emulated attacks, even when combined with a state-of-the-art anti-detection framework. Our experiments show that Laccolith can hide its activities from all the tested anti-virus products, thus making it suitable for realistic emulations.

Index Terms—Cybersecurity, MITRE ATT&CK, TTPs, Adversary Emulation, APT, Virtualization

1 Introduction

Advanced Persistent Threats (APTs) have become a severe threat in several domains where the impact can be exceedingly high (e.g., in terms of service outages, private data breaches, and intellectual property theft), such as healthcare, manufacturing, telecom, energy, and transportation. In APT, attackers accomplish their goals through a carefully-planned sequence of malicious actions performed while being careful to stay undetected. These attacks are getting even more challenging to counter, as they are carried out by cybercriminal and state-sponsored groups. Well-known examples include the Stuxnet attack, which has been sabotaging Iran's nuclear centrifuges since 2005 and was uncovered in 2010 [58], GhostNet [8], and Carbanak [28]. The time an APT attack goes undetected ("dwell time") has been estimated to be up to 700 days [34]. With all this time to act, APTs can cause irreparable damage.

Adversary emulation is the most effective prevention against APTs [1]. Adversary emulation is a proactive approach that reproduces the actions of an APT inside a target computer infrastructure. It grants several advantages for security training and assessment purposes, such as understanding threats, realistic and comprehensive testing, identification of weaknesses, and validation of security controls. This approach focuses on post-compromise scenarios, assuming that the APT has already gained access to the target infrastructure and is infecting hosts and devices silently, escalating privileges, and exfiltrating data. Adversary emulation is usually exercised in virtualized environments (*cyber-ranges*) and, in general, it encompasses time-consuming activities that engage human personnel (*red teams*). Several tools aim to automate adversary emulation [67], which offer automated procedures that implement APT techniques, as learned from threat intelligence sources. These tools typically automate actions for information gathering, lateral movement across hosts, connections to command-and-control servers, and privilege escalation.

A fundamental characteristic of APTs is the adoption of *anti-detection* techniques, to hide their traces from AV (anti-virus) products and EDR (endpoint detection and response), a modern evolution of AVs, and to persist in the target infrastructure as long as possible. Therefore, adversary emulation also needs to apply anti-detection techniques, in order to perform realistic and fruitful security assessments and training activities. Examples of anti-detection techniques include unhooking probes used by EDRs to instrument and monitor DLL and API uses [25]; disabling or hampering event tracers, such as the Event Tracing for Windows (ETW) subsystem [3]; using malicious kernel modules to hide processes and files [9]; obfuscating malicious payloads (e.g., shellcodes) [20], [26]. However, it is challenging for adversary emulation tools to automate anti-detection techniques, since both APTs and AV solutions are continuously evolving, in a "cat-and-mouse" game. Currently, adversary emulation tools have to be customized for the specific AV/EDR to evade, which requires considerable skills and development efforts [14], [55], and is prone to become outdated and ineffective. For these reasons, emulating anti-detection techniques in automated ways in adversary simulations is still impractical. This is a significant limitation in realistically emulating APTs.

In this work, we investigate a novel solution for adversary emulation with anti-detection capabilities, to avoid the previously mentioned "cat-and-mouse" game. We tested state-of-the-art solutions for adversary emulation (*MITRE CALDERA* [67] , *Atomic Red Team* [51], *Invoke-Adversary* [13]) and for anti-detection (*Inceptor* [29]) against multiple AV/EDR products. We found that several malicious actions (and even the installation of the emulation agent) cannot evade detection, thus limiting the realism of emulated attacks. Laccolith is based on a novel *hypervisor-based* architecture for adversary emulation. This design choice was led by the fact that cybersecurity exercises typically happen in virtualized environments [2], [7], [21], [62], [63].

Our solution (*Laccolith*) enables the non-detectable execution of malicious actions, by injecting them from the lowest layers of the software stack. We experimented with Laccolith against several AV solutions for Microsoft Windows. The results showed that Laccolith was able to execute all of the malicious actions, which evaded all of the tested AV products.

- *V. Orbinato, D. Cotroneo, and R. Natella are with the Department of Electrical Engineering and Information Technology (DIETI), Università degli Studi di Napoli Federico II, Naples, Italy.*
  E-mail: {vittorio.orbinato, cotroneo, roberto.natella}@unina.it
- *M.C. Feliciano is with Secureware s.r.l., Naples, Italy.*
  E-mail: mfeliciano@secware.it

Manuscript received April 19, 2005; revised August 26, 2015.

Our solution does not require customizations for the specific version of the target system, as it can reliably execute non-detectable actions across different versions of the guest OS and AV products.

In summary, the research contributions of this work are:

- An innovative architecture for adversary emulation with anti-detection capabilities;
- A working prototype of the proposed architecture;
- An experimental analysis of anti-detection capabilities in state-of-the-art solutions for adversary emulation;
- An evaluation of the proposed solution compared to the state-of-the-art.

The remainder of the paper is structured as follows. In Section 2, we elaborate on background concepts about APTs and adversary emulation. In Section 3, we present the proposed solution for adversary emulation. In Section 4, we illustrate experiments on the anti-detection of existing solutions and our proposed one. In Section 5, we discuss related work. Section 6 concludes the paper.

2 Background

APTs are complex attack campaigns that eventually gain access to a target infrastructure due to inevitable weaknesses, such as the exploitation of an unknown ("zero-day") software vulnerability, a known vulnerability in outdated software, or human mistakes that result in weak credentials, information disclosure, and execution of untrusted code [57]. Initial access represents just the preliminary part of the attack, followed by several other activities that depend on the objectives and skills of the attackers.

In recent years, several models were introduced to describe the actions and behavior of attackers. ATT&CK [10] is a security framework introduced by the MITRE Corporation and has rapidly become the *de facto* standard in the security community. ATT&CK provides a detailed classification of the adversaries' tactics, techniques, and procedures (TTPs), organized as a matrix, continuously updated by observing and analyzing threat intelligence sources [56], [67]. *Tactics* describe **why** an adversary performs an action, and *techniques* describe **how** they do it. Techniques are described in the ATT&CK model from both offensive and defensive points of view, so they are a useful reference and pivot between both disciplines. The descriptions also include references to known usage examples of the specific techniques and links to public threat reporting on adversary groups and their campaigns. Moreover, techniques can further specialize into sub-techniques, which usually provide more specific versions of an attacker's actions. Finally, *procedures* represent the actual implementation of techniques for specific target systems. At the time of writing, ATT&CK encompasses 193 techniques and 401 sub-techniques. The richness of this framework makes it suitable for modeling APT campaigns. It is worth noting that a great number of combinations of TTPs is possible, making it relevant to focus on the TTPs of specific APTs of interest. Typically, APTs are classified according to the targeted domain, such as education, finance, government, and healthcare.

As an example, we will refer to the OilRig APT [11]. OilRig is a cyber threat actor operating since 2014, interested in several domains, such as finance, government, energy, chemical, and telecommunications [12]. It primarily leverages social engineering as an initial attack vector. The Center for Threat-Informed Defense (CTID) [19] defined the model of its behavior and actions by collecting and analyzing threat and incident reports from several sources, such as Cyware [15], Mandiant [35], and Malwarebytes Labs [33]. Figure 1 shows OilRig's operational steps for *exfiltrating data from a targeted server*. The first tactical goal is *Initial Access*, to identify the target to exploit: OilRig achieves it using *spearfishing attachment* (T1566[1] [45]) (step ① in Figure 1). Once the target opens the malicious attachment (T1204.002 [47]), a backdoor is installed on the host machine (step ②). Then, it performs *enumeration* (T1087 [41]) (step ③), leading to the discovery that the user is a member of the administrator group on an Exchange Web Server (EWS). Using the aforementioned backdoor, OilRig *obtains credentials* to EWS (T1003 [44]) (step ④). Leveraging these credentials, the attackers connect to EWS and install a new backdoor (step ⑤) to perform further *enumeration* (T1087 [41]) (step ⑥): this leads to the discovery of an SQL server. Using the backdoor, OilRig *dumps the credentials* (T1003 [44]) (step ⑦) to access the SQL server. The next step is *lateral movement* towards the server (step ⑧) by passing-the-hash (T1550.002 [46]), where the attackers will copy the database backup files (step ⑨) and exfiltrate them to a controlled mailbox (T1048 [43]) (step ⑩).

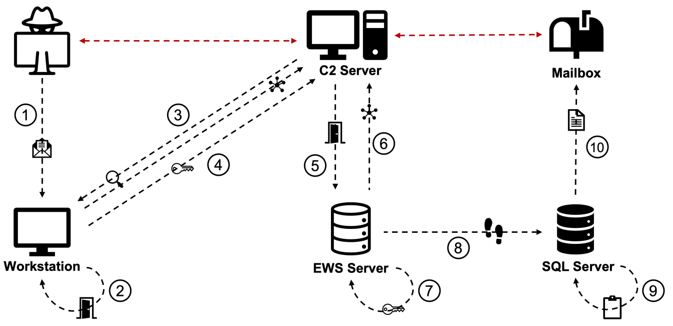

Fig. 1: OilRig Operational Steps. The red dotted arrows represent C2 communication, the black ones the operational steps.

Once a model of an APT has been created using the ATT&CK framework, it can be used for testing and assessing the security posture of a target infrastructure through adversary emulation. Several tools have been developed to facilitate and automate adversary emulation. They provide automated low-level procedures that implement the techniques described by the ATT&CK framework. These procedures emulate the actions of actual adversaries in a typical attack scenario. In particular, we will refer to CALDERA [1], [42], an open-source adversary emulation tool developed by MITRE. CALDERA has gained popularity due to its feature-richness, maturity, and ease of setup and use [67]. In addition, it provides high coverage of TTPs of the MITRE ATT&CK framework. In contrast, other adversary emulation tools consist of a small set of scripts (e.g., in Python and Powershell) that automate individual malicious actions (e.g., the Atomic Red Team), leaving human red teams to string these single actions into a complete APT campaign.

CALDERA is based on a client/server architecture, as shown in Figure 2: the command-and-control (C2) server is responsible for administrating the operations, i.e., the simulations, planning the actions to execute, and making all the information available to the user. On the other hand,

[1] **Txxxx** indicates a technique from the MITRE ATT&CK framework

This article has been accepted for publication in IEEE Transactions on Dependable and Secure Computing. This is the author's version which has not been fully edited and content may change prior to final publication. Citation information: DOI 10.1109/TDSC.2024.3376129

the clients are the agents installed on the target machines: the agent is responsible for the actual execution of the actions and for sending the results back to the C2 server.

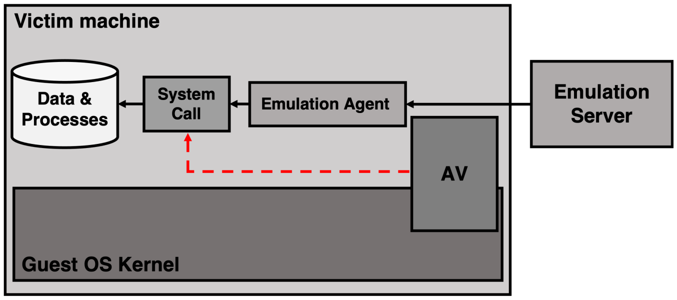

Fig. 2: Adversary emulation traditional architecture.

Adversary emulation tools, such as CALDERA, do not provide anti-detection techniques, hence the need to turn off the AVs during the emulations. Turning off the AV causes the system to run in a different environment. This aspect constitutes a relevant concern in critical domains (such as defense), which need to accurately emulate advanced adversaries with evasion abilities. For instance, for forensic analysis purposes, it is best not to alter the lists of processes and files seen by the security teams, which would be polluted by adversary emulation tools that run in user-space. Moreover, the emulation designers may want some specific malicious activities (not all) to be detected, to guide the participants in solving pre-planned cyber-attack scenarios. As an extreme, if none of the activities were detectable, the blue teams would not have any element to learn from. Our solution allows the designer to selectively configure which activities to hide from the AVs and the participants.

To perform non-detectable actions, red teams hide their traces using separate tools. A state-of-the-art toolkit for anti-detection is Inceptor [29]. Inceptor is a well-known template-driven framework for AV evasion. A template is a generic, customizable loader with placeholders for evasion techniques and the actual payloads. There are many templates for three different types of payloads: .NET, PowerShell, and native code. Inceptor offers the possibility of chaining encoding techniques to evade static code analysis. It is also possible to plugin additional source code writing techniques to evade the Windows Antimalware Scanning Interface's (AMSI) dynamic analysis [24]. AMSI also analyses in-memory artifacts, for example, text areas the code is going to jump to. For this reason, Inceptor includes AMSI bypass techniques [30]. In our experimental evaluation of the state-of-the-art on adversary emulation and anti-detection, we consider MITRE CALDERA in combination with Inceptor.

3 Proposed Solution

In this section, we present our proposed solution for adversary emulation with anti-detection. We first provide an overview, including our system model and assumptions for our design. We then describe in detail the proposed design. Finally, we provide information about our implementation.

3.1 Overview

We based the design of Laccolith on a set of assumptions applied in general for adversary emulation. The assumptions include:

- Adversary emulation focuses on post-compromise scenarios where the attacker has already gained a foothold inside the system (e.g., through phishing or exploiting a vulnerability) and is performing more malicious actions, such as gaining more privileges and stealing information.
- Adversary emulation is performed in the context of cybersecurity exercises, which are authorized and overseen by a "white team" in the organization (e.g., system administrators). Adversary emulation can be warranted system privileges as needed.
- Laccolith is designed to perform adversary emulation in a virtualized environment, where hosts are deployed using virtual machines. This is typically the case in cybersecurity exercises, where dedicated networks and hosts are deployed using virtualization infrastructures [2], [31], [54]. For example, all the most popular Cyber Ranges run in the cloud, leveraging virtualization technologies [7], [21], [63]. Moreover, the solution applies to organizations that run their own private data centers, which is often the case for high-criticality domains such as defense, critical infrastructures, and healthcare [16], [53]. Instead, the proposed solution is not meant for networks and services running on bare-metal hardware.
- Endpoints in the environment can be equipped with AV products, as in the case of real computer networks. This assumption is not met by traditional adversary emulation tools, which require turning off AV products to install and run the tools [67]. Laccolith is designed to overcome this limitation.

The driving idea for our design is to leverage kernel- and hypervisor-level privileges to execute malicious actions, which would otherwise be detected if performed from user level (i.e., from an application process).[2] AV solutions typically use "hook" functions to intercept invocations of the system calls. When applications (including malicious ones) invoke system calls, the hook functions are executed instead (e.g., by replacing pointers in the system call table), which can check the invocation and detect suspicious activities. In traditional adversary emulation (Figure 2), an agent process executes malicious actions on behalf of a red team by issuing system calls that access OS resources, such as files, processes, connections, and others. Without any anti-detection technique, AV products can detect these actions by checking system calls.

To perform non-detectable actions, it is necessary to use system calls not monitored by the AVs, hence from the kernel level. Consequently, the hypervisor level is the ideal choice to accomplish that, leveraging virtual machine introspection (VMI) techniques. In our design, we circumvent the anti-virus checks by introducing an emulation agent from the hypervisor. Since the hypervisor has full privileges on a physical machine, it can warrant full read and write access to the state of a virtual machine. We use these privileges to install an emulator into the kernel of the guest OS of the virtual machine. This way, it is possible to directly access guest memory, without issuing user-space system calls. From the agent installed in the guest OS kernel, we can perform malicious actions by calling kernel-level APIs. Such calls cannot be detected by AV products since they are not subject to security checks.

[2] The name *Laccolith* reflects the injection of malicious actions from the lower layers of the software stack. A laccolith is a volcanic phenomenon where magma rises through the Earth's crust, forcing rock strata upward.

Moreover, we designed the kernel-level agent with the ability to run user-level commands, as in traditional adversary emulation tools. Therefore, red teams can combine kernel-level and user-level actions to perform both detectable and non-detectable actions. This flexibility enables red teams to perform realistic training exercises for security teams, where the emulated APT leaves only a few selected traces of malicious activity.

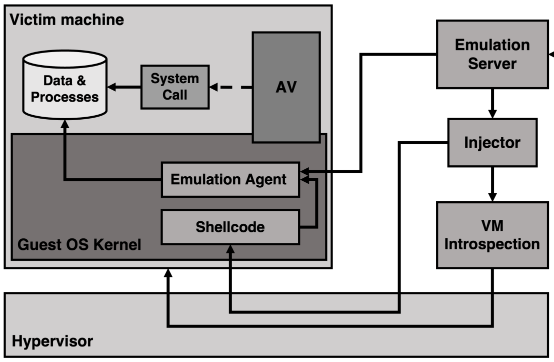

Fig. 3: Laccolith architecture.

Laccolith offers a complete architecture for adversary emulation: a central *emulation manager* orchestrates *emulation agents* across the network and issues actions to execute on the victim hosts. Therefore, the red team controls the emulation agents to reproduce the actions of an APT campaign. The red team can interact with the emulation manager through web and command-line interfaces. On each physical machine, Laccolith installs an *emulation server*, which runs at hypervisor-level, and manages the emulation agents on each physical machine. The emulation server installs the emulation agents on each virtual machine by leveraging VM introspection techniques to modify guest memory and create a new execution flow inside the guest kernel. Moreover, the emulation server forwards communication between the emulation agents and the manager. The communication traffic between the components of Laccolith is invisible to the participants of the cybersecurity exercise since this traffic flows at the physical level, beyond the virtual networks of the virtual machines.

Laccolith is neutral concerning post-facto analysis: the designer of the attack scenario can decide which artifacts should be left inside the system for forensics purposes, thanks to the possibility of configuring which specific actions are executed (un)detectably. For instance, the designer may want to stealthily poison some entries in the system registry for forensic analysis. Moreover, the designer can launch processes or connections in a non-detectable fashion from kernel space. In these cases, the participant is challenged to use event correlation techniques and SIEM technology to analyze the system state and understand what is happening.

In summary, our design addresses the following technical challenges: *1)* to run an emulation agent in a victim machine, with no intrusive modifications of the system under analysis; *2)* to perform adversarial actions without triggering AVs; and *3)* to coordinate such adversary emulation campaigns from outside of the victim machine through a C2 infrastructure.

3.2 Emulation server

The *emulation server* is the trickiest component in our architecture. It is responsible for injecting *emulation agents* inside the VMs, by leveraging read/write (R/W) access permissions to modify their state. It relies on installing a kernel-level agent to avoid detection from AV products.

3.2.1 Overview of the injection method

The emulation agent needs to run in a dedicated area of the virtual memory of the VM. However, the allocation of virtual memory cannot be directly performed from the hypervisor but needs to be performed by the guest OS kernel, which is aware of the current layout of memory allocations. Consequently, the injection of the emulation agent consists of two stages. The first stage consists of the injection of a small program (a “shellcode”) in kernel space, responsible for allocating the code region for the emulation agent, while the second stage entails bootstrapping the asynchronous execution of the agent in that area. The injector overwrites a piece of existing kernel code with the shellcode to make it executed by the kernel. Differently from the *injector* (which runs in the hypervisor and can only work with “guest physical” memory addresses), the shellcode can allocate and access the “guest virtual” memory addresses since it runs within the guest OS kernel. Before selecting the target kernel code to overwrite, we need to address two requirements for the shellcode:

1) *Space*: the shellcode should be small enough to fit into the target kernel code;
2) *Behavior*: if the overwritten kernel code is invoked, the shellcode needs to handle the call without hanging the calling process. Moreover, it should also handle the case of concurrent calls from different processes.

We consider the code of system calls for the injection of the shellcode since their position in the virtual memory of the VM can be identified with *virtual machine introspection* (VMI) techniques. VMI is an approach to gain visibility and control over VMs without modifying the guest OS [22]. Moreover, system calls are regularly invoked by applications, thus assuring that the shellcode will be eventually executed. The previously mentioned requirements are the reason why we cannot trivially overwrite a system call with the code of the *emulation agent*: the first one would restrict the size of the agent, which would hamper the implementation of malicious actions; the second requirement implies that the system call code needs to be restored at some point, thus removing the injected code.

To choose a target system call to overwrite, we look for a *linear region of code*, i.e., a memory region that meets the following requirements:

1) The memory region is from the code area of the guest OS kernel; this code is only executed by starting from the initial address of the memory region. For example, this requirement is satisfied if the memory region exactly matches the code of an individual kernel function; in this case, other kernel code only jumps to the initial address of the memory region (i.e., there are no jumps to addresses in the middle of the memory region).
2) The code in the memory region does not call other functions. This requirement avoids that code not belonging to the memory region returns in the middle of the region.
3) The region fits inside a memory page without crossing page boundaries.

The first and second requirements allow us to inject arbitrary code in the memory region without risking the kernel jumping in the middle of the region, which would likely raise CPU exceptions (e.g., executing an invalid opcode). The shellcode will execute when the kernel executes the memory region, in place of the original code. The third requirement is necessary to make it easier to modify the VM memory from the hypervisor since code over multiple pages in virtual memory does not necessarily map to contiguous pages in physical memory. An example of an eligible memory region in Microsoft Windows OS is the *MmQueryVirtualMemory* function (about $3,800$ bytes), which is called by the *NtQueryVirtualMemory* system call. In general, linear regions of code are plentiful and easy to find: given the function call graph of the kernel code (e.g., *ntoskrnl.exe* in Microsoft Windows), a linear region of code is a function without fan-out that fits within a single memory page.

3.2.2 Injection method

The injection method is shown in Figure 4 and summarized as follows. First, the *injector* searches the code of the target system call in memory (step ①). Since the *injector* can only access VM memory through guest physical addresses, it scans memory by looking for code that matches the initial unique bytes of the target system call. The injector leverages VMI to make this process more portable and efficient: it determines from VMI the unique bytes of the system call from the kernel binary, and the relative offset of the system call within the kernel virtual address space. Then, it only scans memory pages on that relative offset. Once the position of the target system call has been found, the *injector* overwrites it with the shellcode (step ②).

Then, the shellcode will eventually execute when the system call is invoked. The shellcode will execute for a limited time, long enough to allocate a memory area and communicate to the emulation server that the memory area is ready for loading the emulation agent (the last step of the method). The shellcode uses the APIs of the guest OS kernel (step ③) to allocate a contiguous area of guest virtual memory (for example, the *MmAllocateContiguousMemory* function from the Windows Driver Kit (WDK) [38]). The shellcode then obtains pages of contiguous virtual memory (e.g., 16KB in our implementation) from the guest virtual address space (step ④), without the constraints that previously applied for the shellcode.

During this time window, any other concurrent call to the target system call must immediately return, since the shellcode should execute only once. For this purpose, the shellcode acquires a spinlock in a *non-blocking* way, without busy-waiting if the spinlock is already held (e.g., using the *xchg* instruction on Intel architectures [27]). Once acquired, the shellcode holds the spinlock to avoid concurrent executions.

Next, the shellcode writes a pre-defined value ("egg"), also known by the injector, within the allocated memory area. This way, the injector can find the allocated memory area using only guest physical addresses. The injector overwrites the allocated memory with the code of the emulation agent (step ⑤). Finally, the shellcode sets up an execution context to run the emulation agent in the kernel. Again, the shellcode uses APIs of the guest OS kernel to create a kernel thread (step ⑥), such as using the *PsCreateSystemThread* function in Windows. The new kernel thread will be configured to run the code of the emulation agent in the allocated memory area (step ⑦), e.g., using the *StartRoutine* of the Windows kernel. As a result, the emulation agent will execute with high privileges since it will run in kernel mode. After the agent starts, the injector restores the original system call code to leave no traces in the target VM other than the emulation agent (step ⑧).

3.3 Emulation agent

The *emulation agent* is responsible for receiving and executing commands from the emulation server. Since it runs with high privileges, it can call kernel APIs as if it were a kernel module. The agent can also access the OS resources, such as the process descriptors and the system registry. Through these resources, the agent can access the memory of user-level processes since a process descriptor indicates the memory regions where the code and data of a process are. It can both read (e.g., process dumping to steal passwords, tokens, and other data) or write (e.g., process/DLL injection to hide malicious code in system processes) the processes' memory, and modify the system configuration, e.g., changing the registry for persistence purposes. Moreover, the agent can read/write the file system, e.g., navigating through folders, creating new files, and deleting existing ones. It can also create new user-level processes to execute commands, such as simulating fileless malware by running system binaries (e.g., PowerShell, WMI) and creating new OS resources. For example, the execution of user-mode commands can be accomplished using worker factories [65], [66], a Windows mechanism that allows the kernel to create user-space thread pools and let them execute specific tasks.

3.4 Emulation manager

The *emulation manager* communicates with the agents. Its role is critical to orchestrating the low-level actions performed by the agents into emulating the complex behavior of APTs, as in the example of Section 2. It also offers a user interface to manage Laccolith. In particular, it offers interfaces to inject agents into specific targets, to access C2 functions (e.g., listing connections to agents, sending a command to an agent, reading its output, starting an autonomous operation), to customize parameters (e.g., choosing which payload to inject), and upload and download files (e.g., to exfiltrate data and information gathering). The emulation manager is also responsible for the management of the *facts*. A fact is a piece of information about the target system, helpful to run an ability. The manager communicates the fact values to emulation agents when they need specific information to perform an action. For example, suppose the goal of the emulation is to perform lateral movement to another machine in the local network. The emulation agent will perform a network scanning action, whose outcome will be a fact containing the username and IP address of the target machine. The agent will then use this fact to perform lateral movement.

3.5 Implementation

We implemented the design of Laccolith for the QEMU hypervisor for x86_64 [50], running on a Linux host managed by Libvirt [32], and with hardware-supported virtualization based on KVM [52].[3]Laccolith runs as a privileged process on the Linux host and accesses VM memory through a virtual

[3] The source code and documentation of the prototype is available at https://github.com/Shotokhan/adversary-emulation for research purposes. A commercial version of the tool and support is provided by the Secureware s.r.l. spin-off company, see https://www.secware.it/.

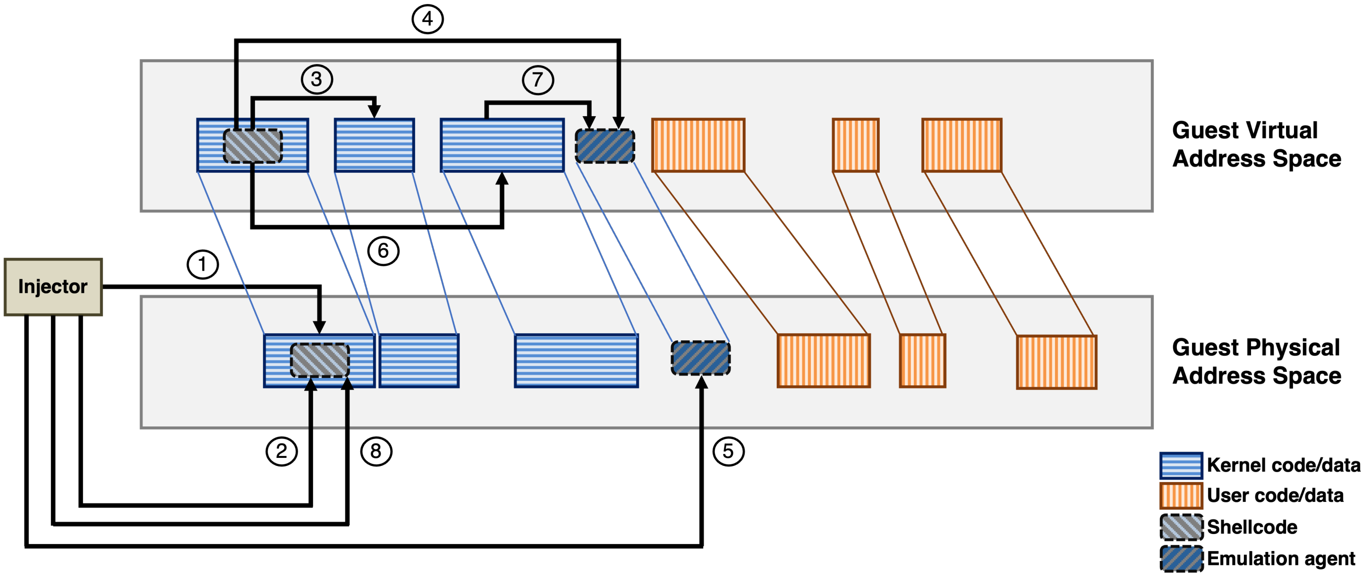

Fig. 4: Injection method in Laccolith.

device file. Our implementation targets the Microsoft Windows OS as the guest OS for the victim machine. We used Volatility [61], a virtual machine introspection (VMI) framework, to get Windows kernel symbols. We chose Volatility since it is a widespread and well-known framework that offers several profiles to inspect the memory of several OSes. Using a popular VMI framework facilitates the portability of Laccolith to new versions of the guest OS (e.g., builds of Microsoft Windows) and new guest OSes since the community continuously provides multiple profiles to align VMI with new releases of the OSes.

4 Experimental Analysis

We compare Laccolith with MITRE CALDERA since it is a state-of-the-art solution for adversary emulation. Compared to other adversary emulation tools, CALDERA provides greater coverage of APT tactics and techniques, a complete client-server architecture, and the ability to orchestrate complex APT campaigns (see also Section 5 for a detailed comparison). We evaluate adversary emulation concerning its *detectability*.

To evaluate detectability, we test the adversary emulation tools against a set of AV solutions, described in Section 4.1. As a metric to measure detectability, we consider the **adversary profile execution progress**. An adversary profile is a sequence of atomic steps that represent malicious activities. The metric represents the progress of the execution of an adversary profile in terms of atomic steps successfully executed until the AV raises an alarm. The *successful execution* of an atomic step means that the agent (from either CALDERA or Laccolith) can execute the action without being detected by the AV. The steps that compose an adversary profile are called *abilities* for CALDERA and *actions* for Laccolith. This metric is expressed as a fraction:

$$\text{Adversary Profile Execution Progress} = \frac{N_{EA}}{N_{PA}}$$

N_{EA} is the number of executed actions of an adversary profile, while N_{PA} is the total number of its actions. For example, suppose a profile has 10 actions, and the AV detects the execution of its eighth action. In that case, the adversary profile execution progress will be $7/10$ since the profile managed to execute 7 actions out of 10.

To gain additional insights into the detectability, we also analyze the initial loading of the emulation agent in the victim machine. Since AV products perform rigorous checks before an executable launches, the emulation agent also needs to evade these checks. For this purpose, we introduce an additional metric, the **injection success**. This metric represents the probability of successfully injecting the agent into the victim without triggering detection. The metric is defined as a fraction:

$$\text{Injection Success} = \frac{N_{SI}}{N_I}$$

where N_{SI} is the number of successful injections attempts and N_I is the total number of attempts.

Finally, we also include *atomic tools* in our experimental analysis of state-of-the-art, i.e., tools that enable the execution of single (atomic) adversarial actions, outside the context of a complete attack campaign. In particular, we focused on *Atomic Red Team* [51] and *Invoke-Adversary* [13], two popular atomic toolkits [67] as discussed in Section 5. These tools are designed to perform single adversarial actions from the MITRE ATT&CK matrix. Their atomic nature allows us to overcome the problem of not testing techniques skipped when running an entire campaign with multiple actions, which could still be detected and stopped by the AVs. Specifically, we analyze the percentage of actions that can be detected by AVs.

4.1 Experimental Setup

We used the following configuration for the experimental analysis, shown in Figure 5:

- Victim Windows 10 VM (VM1): the target of the adversary emulation;
- Additional Windows 10 VM (VM2): it is a machine necessary to perform some malicious actions, e.g., network shares enumeration performed by the *Shares Hunter* profile, as shown in Table 6.

Both VMs are equipped with 4 CPU cores and 2 GB RAM and run Microsoft Windows 10. Before our experiments, we tested the portability of Laccolith across different versions of the guest OS. We deployed Laccolith on seven Windows 10 versions, which span over three years, with significant changes across the versions [64]. Laccolith leverages Volatility to apply the injection method in a portable way. Indeed, Laccolith could correctly inject the emulation agent in all the tested versions of Windows without the need for special customizations. In our experiments, we focus on Microsoft Windows 10 build 19044.

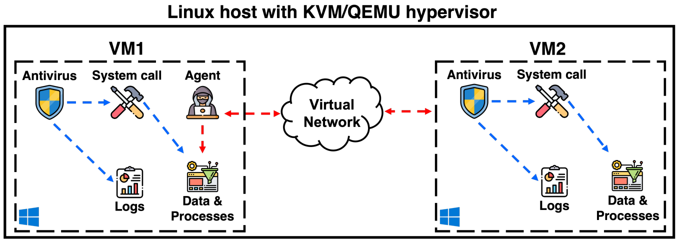

Fig. 5: Experimental setup.

We selected five popular AV products for our detectability analysis: Windows Defender, Avast, AVG, Kaspersky, and Avira. These products qualify as modern EDR solutions and cover state-of-the-art detection techniques [5]. For instance, they monitor system calls using user-space and kernel-space hooks; they use both signature-based and real-time behavioral detection; and they implement self- and system-protection techniques. An example of such protections is preventing filesystem access to specific directories, such as the main system directories and the AV directory itself.

4.2 Detectability evaluation of MITRE CALDERA

To study the detectability of CALDERA, we selected its twelve default adversary profiles. These profiles can be classified into two categories: *reconnaissance & information gathering* and *advanced*. The first category encompasses the first eight profiles listed below, which perform basic operations: user identification, process enumeration, anti-virus discovery, screenshot capture, and file search. The advanced profiles perform more invasive actions, like process injection, lateral movement, and malicious payload execution. As a consequence, their activities will be noisy and more likely to be flagged by the anti-viruses. We report a short description for each profile:

- *Discovery*: collects detailed information from a host, such as local users, user processes, admin shares, and anti-virus programs;
- *Hunter*: performs Discovery operations, then tries to exfiltrate files from the working directory;
- *Check*: collects information about the configuration of the host (e.g., installation of common software packages, such as Chrome, Go, and Python), and the configuration of its network interfaces;
- *Collection*: collects personal information from the host, such as company emails, IP addresses, and personal files;
- *Enumerator*: enumerates the presence of different types of processes on the host, such as WMIC, PowerShell, and SysInternals utilities;
- *Nosy Neighbor*: finds the preferred Wi-Fi networks, and tries to disrupt the Wi-Fi connection;
- *Signed Binary Proxy Execution*: executes malicious code through signed and trusted binaries;
- *Super Spy*: monitors the active user by capturing screenshots, copying data from the clipboard, and scanning preferred Wi-Fi networks;
- *Undercover*: swaps the built-in PowerShell with PowerShell Core to stop PowerShell processes;
- *Stowaway*: injects Sandcat (the default emulation agent in CALDERA) into another process;
- *Worm*: runs PowerKatz to steal user credentials, then moves laterally in multiple ways;
- *You Shall (Not) Bypass*: bypasses User Account Control (UAC).

In addition to these profiles, we implemented a custom *Ransomware* profile for completeness. This profile was developed from scratch, similar to the implementation of the predefined profiles. It encompasses the following abilities:

- Find files;
- Stage sensitive files;
- Compress the staged directory;
- Exfiltrate the staged directory;
- Encrypt sensitive files (e.g., *docx* and *pdf* files).

The first four abilities re-use the ones of the predefined profiles to find the target files. The last ability has been implemented with a PowerShell script [37] to encrypt the sensitive files, by silently skipping the files if it does not have permission to rewrite them. Using a PowerShell script with standard APIs for encryption makes the process more similar to "legitimate" programs (as in fileless malware), similar to other profiles in CALDERA. In this way, the Ransomware profile should be as detectable as the other reconnaissance & information gathering adversary profiles.

TABLE 1: Adversary Profile Execution Progress for MITRE CALDERA. **Bold** values represent incomplete progress.

Profile	Windows Defender	Avast	AVG	Kaspersky	Avira
Discovery	9/9	9/9	9/9	9/9	9/9
Hunter	14/14	14/14	14/14	14/14	14/14
Check	6/6	6/6	6/6	6/6	6/6
Collection	2/2	2/2	2/2	2/2	2/2
Enumerator	5/5	5/5	5/5	5/5	5/5
Nosy Neighbor	7/7	7/7	7/7	7/7	7/7
Signed Binary Proxy Execution	3/3	3/3	3/3	3/3	3/3
Super Spy	11/11	11/11	11/11	11/11	11/11
Undercover	**1/2**	**1/2**	**1/2**	**1/2**	2/2
Stowaway	**1/2**	**1/2**	**1/2**	**1/2**	2/2
Worm	**1/9**	**1/9**	**1/9**	**1/9**	9/9
You Shall (Not) Bypass	**2/4**	**2/4**	**2/4**	**1/4**	**1/4**
Ransomware	5/5	5/5	5/5	5/5	5/5

Table 1 shows the execution progress for each adversary profile of MITRE CALDERA against the five anti-viruses. It is possible to notice that the reconnaissance & information gathering profiles execute without being detected since they do not perform harmful actions. For the advanced profiles, the results of the experiments are quite different. During the execution, all the anti-viruses, except for Avira, flagged the activities of the profiles as suspicious. Moreover, the detected abilities are always the first ones in the atomic order: once an ability has been detected, it is not possible to complete the operation, since the prerequisites for the following ones will not be satisfied. As a consequence, the last abilities of the advanced profiles could not be executed. For instance, the Stowaway profile uses two abilities: since the execution of the first one is stopped by the anti-virus, the second one cannot be tested. Therefore, this profile scored 1/2 in terms of detectability. We reported the detected abilities for each profile in Table 2. This experimental analysis shows that the emulation agent of MITRE CALDERA can stealthily execute most Discovery, Collection, and Exfiltration techniques, but is easily blocked when it tries to perform more intrusive actions, such as Credential Access and Privilege Escalation.

It is worth noting that the AVs always detect the injection of CALDERA's agent. After performing 20 repetitions of the injection, we conclude that the injection success of CALDERA is $0/20$ against each AV. Consequently, the results in Table 1 only refer to the execution of the adversary profiles, assuming that the agent has already been loaded on the target machine.

4.2.1 Integrating CALDERA with anti-detection

To overcome the limitations of MITRE CALDERA in terms of anti-detection, we combined it with Inceptor [29], a state-of-the-art evasion framework introduced in Section 2. In this experimental analysis, we focus on Windows Defender, and on the injection stage of the emulation agent of CALDERA (i.e., the initial installation of the agent). We enhance the agent with multiple anti-detection techniques from Inceptor.

The injection needs to bypass three layers of detection: User Account Control (UAC), signature-based static analysis, and dynamic sandbox analysis. UAC is a protection that involves user interaction, by displaying a message through a GUI frontend (Microsoft SmartScreen [40]). Sandbox analysis is a technique used by anti-virus software to analyze potentially malicious files and programs, by running them in a safe and isolated environment. It is particularly useful for detecting new and unknown malware, which can often evade static signature-based detection. In Microsoft Windows, *Windows Defender Application Guard* [39] is responsible for performing the sandbox analysis, using a virtual container with a specialized version of the Windows OS.

We used the following combination of anti-detection techniques to make the injection stealthier: a native binary template, an encoding chain composed of *Shikata-Ga-Nai* [17], a popular polymorphic binary encoder [36], XOR encoding, 120 seconds of execution delay, and an unhooking technique for EDR bypass. Moreover, we signed the resulting binary with a Microsoft signature using CarbonCopy [49].

We experimentally evaluate the injection of the CALDERA agent with anti-detection by performing 20 repetitions. The injection is unable to evade UAC. This would require the exploitation of known vulnerabilities in Microsoft Windows (*UAC bypass*), which are regularly fixed by updates of the OS. Therefore, Inceptor is unable to provide a reliable solution to escape UAC, and we manually allow the execution of the CALDERA agent through the UAC. Even neglecting the problem with UAC, the agent was successfully injected approximately 5 times out of 20, thus with an injection success equal to $5/20$ and 25% probability. The encoding chain allows the binary to bypass static analysis since it does not match any known signature. The execution delay and unhooking technique heuristically help to elude the dynamic sandbox analysis: the delay is helpful to make the behavior seem benign while unhooking prevents EDRs from inspecting function calls. The binary signature helped to evade the dynamic analysis: when the binary has a valid signature, Windows Defender performs less detailed dynamic checks. However, since the binary needs to decode itself from the encryption chain at some point, it is still challenging to bypass the sandbox analysis, making the injection detectable. The need for additional UAC bypass techniques further complicates the process of making CALDERA stealthier.

4.3 Detectability evaluation of atomic tools

After CALDERA, we evaluate the detectability of individual actions from atomic tools (*Atomic Red Team* [51] and *Invoke-Adversary* [13]). For the sake of clarity, we focused on testing a subset of the actions provided by these tools. We sampled the actions uniformly across the tactics of the MITRE cyber kill chain. Table 3 provides the percentage of actions detected by the AVs. Table 4 and 5 provide detailed results for Atomic Red Team and Invoke-Adversary, respectively. These tools show the same drawbacks as CALDERA, where the most intrusive actions all raise alerts from the AVs. Looking at the tables, it is possible to notice how non-intrusive actions are the only ones not detected by any of the AVs, namely *GetCurrent User with PowerShell Script*, *Prompt User for Password*, *Activate Guest Account* for Atomic Red Team, and *System Owner Discovery* and *Screen Capture* for Invoke-Adversary. It is worth noting that even the deployment of the tools triggers an alert from the AVs. So, it is not possible to use them without turning AVs off in the deployment phase.

4.4 Detectability evaluation of Laccolith

Our previous analysis showed that traditional adversary emulation is cumbersome and unable to perform intrusive actions without being detected. We here present an experimental analysis of detectability for Laccolith. For this analysis, we implemented four adversary profiles with Laccolith: *Thief*, *Op-2*, *Ransomware*, and *Shares Hunter*, as described in Table 6. Since we had to develop adversary profiles from scratch for Laccolith, we did not aim for a verbatim reimplementation of the adversary profiles of CALDERA. Instead, the adversary profiles in Laccolith match the CALDERA profiles in terms of high-level strategy of the attackers. Moreover, we relate the adversary profiles for Laccolith with real-world APTs (threat-informed adversary emulation). The new profiles in Laccolith cover all the tactics covered by CALDERA profiles, except Privilege Escalation (covered by the *You Shall (Not) Bypass* profile) since Laccolith already has high privilege. The Ransomware profile also covers the Impact tactic, which is not covered by the default profiles of CALDERA. We introduced this profile to show the ability of Laccolith to support complex operations and cover a relevant cybersecurity threat.

Table 7 provides details on which commands of the emulation agent are involved in each profile, in order to get more insight about the adversary profiles of this experiment. In total, the profiles cover 7 commands implemented by the emulation agent, including actions to access the filesystem, the system registry, and to execute user-space commands. Moreover, the Laccolith agent provides additional commands for managing adversary emulation campaigns, not shown in the table for the sake of brevity. Overall, the newly implemented profiles allowed us to test the functionalities of Laccolith in depth.

To compare Laccolith to CALDERA, we performed the same experimental analysis for detectability. Table 8 shows the adversary profile execution progress for the four profiles against all the chosen AVs. It is possible to notice that all profiles achieve complete execution progress, meaning that the AVs do not detect any of their actions. It is worth noting that the Shares Hunter profile has seven actions according to Table 8, instead of the four mentioned in Table 6. This happens because this profile does not have an *a priori* planning, but a dynamic one in which some actions depend on the outcome of the previous ones. In particular, since this profile

TABLE 2: Abilities of MITRE CALDERA detected by AV products.

Profile	Ability	Windows Defender	Avast	AVG	Kaspersky	Avira
Undercover	Install PowerShell Core 6	✓	✓	✓	✓	
Stowaway	Inject Sandcat into Process	✓	✓	✓	✓	
Worm	Run PowerKatz	✓	✓	✓	✓	
You Shall (Not) Bypass	Wow64log DLL Hijack	✓	✓	✓	✓	✓
You Shall (Not) Bypass	Bypass UAC Medium	✓	✓	✓		

TABLE 3: Atomic techniques detection results for each AV product.

Profile	Windows Defender	Avast	AVG	Kaspersky	Avira
Atomic Red Team	70%	70%	70%	50%	30%
Invoke-Adversary	75%	75%	75%	63%	50%

TABLE 4: Technique detectability for Atomic Red Team.

Technique	Windows Defender	Avast	AVG	Kaspersky	Avira
Execute base64-encoded PowerShell from Windows Registry	✔	✔	✔	✔	
GetCurrent User with PowerShell Script					
Thread Execution Hijacking	✔	✔	✔	✔	✔
Prompt User for Password					
Mimikatz	✔	✔	✔	✔	✔
Clear Logs	✔	✔	✔		
Activate Guest Account					
Disable Windows Security Center Notifications	✔	✔	✔	✔	✔
Bypass UAC using Event Viewer (PowerShell)	✔	✔	✔		
Disable Microsoft Defender Firewall	✔	✔	✔	✔	

TABLE 5: Technique detectability for Invoke-Adversary.

Technique	Windows Defender	Avast	AVG	Kaspersky	Avira
System Owner Discovery					
PowerShell Encoded Mimikatz	✔	✔	✔	✔	✔
Screen Capture					
Add local firewall rule exceptions	✔	✔	✔	✔	
Create local administrator	✔	✔	✔	✔	✔
Capture Lsass Memory Dump	✔	✔	✔	✔	✔
Clear Security Log	✔	✔	✔		
PSExec	✔	✔	✔	✔	✔

performs network discovery operations, the result of its actions depends on whether there are any neighbors in the network. We assumed that the target VM interacted with the other VM recently, so it has the IP address of VM2 in its ARP cache alongside the gateway one. Both VM1 and VM2 have the NetBIOS sharing option active.

The injection relies on overwriting the code of a system call or a kernel function called by a system call. Then, the shellcode executes only once, and the other concurrent calls return, as described in Section 3.2.1. After loading the emulation agent, the original code is restored. This injection method can fail since the original code may be restored concurrently with a thread executing that function, which could encounter invalid opcodes due to misalignment or valid code that returns errors because of invalid values in the CPU registers. This failure results in an assertion failure (e.g., *bug check*) within the kernel. Moreover, a critical system process (e.g., *SYSTEM svchost.exe*) may crash because the system call does not exhibit the expected behavior.

We performed preliminary experiments to gain insights into these events. In these initial experiments, we executed the injection after an increasing amount of time after the boot of the victim machine. We found that these events are more likely if the injection is performed within a few minutes right after the boot. If the injection is performed after a few minutes have passed since boot, the injection becomes more reliable. This behavior can be explained by the higher activity of system processes in the early phases of the start-up, which make high use of system calls and can expose these processes to failures.

Therefore, the success of the injection method depends on the fraction of time the system spends executing the injected system call, and the percentage of cases that the system call is invoked by system processes. To quantify the effectiveness of the injection method, we performed more experiments by focusing on the favorable case of injection after boot has been completed, and the system has stabilized. The experiments consisted in repeating the injection method multiple times, each time from a clean condition (e.g., a full restart of the victim machine), and measuring how many times the injection was successful. We conducted 20 experiments for each AV, re-booting the target VM each time to have independent samples. The timing of the injection was set to one minute after the Windows login prompt appeared. To verify if the injection was successful, we sent the agent an *echo* and a *close* command to check if it could execute commands and terminate gracefully. The injection fails if the connection is received, but it is impossible to perform these operations. If there is some minor error from user applications, which cannot be attributable to a security alarm (e.g., a generic "unknown exception in explorer.exe"), we still consider the injection successful.

Table 9 summarizes the experimental results. The AV does not impact the injection success since all the setups with the different AVs exhibit similar behavior. The overall success rate of the injection method has been $90/100$, hence 90% of experiments. Considering a margin of error of $1/\sqrt{N}$, where N is the overall number of repetitions [6], the success rate of the injection method ranges between 80% and 100%. Even in the worst case, the success rate is still higher than the probability of running CALDERA and Inceptor without being detected, which had approximately a 25% probability of success, neglecting the issue of bypassing UAC which makes the process even more uncertain.

It is worth noting that the injection success for Laccolith can be further improved by injecting into a target linear region of code that is executed less frequently, in order to reduce the probability that the injection clashes with the execution of the target system call. This can be achieved by a preliminary profiling of the execution frequency of system calls. Since the profiling depends on the workload of the specific system under evaluation, and since this represents an engineering problem, we consider this beyond the scope of our research work.

4.5 Threats to validity

Threats to external validity. We targeted Microsoft Windows as **guest OS** for the victim machine, which may affect the generality of our solution. We focus on Windows since it

TABLE 6: Adversary profiles in Laccolith.

Profile	Description	Tactics	Commands	High-level actions	Referenced APTs
Thief	Exfiltrate files from local user desktop	Discovery, Collection, Exfiltration	Directory listing (to find local users and to list desktop files), Read file	(1) Find local users, (2) List user desktop, (3) Exfiltrate a list of staged files	APT1, OilRig, APT3
Op-2	Upload a Powershell script in a system folder and install a scheduled task that executes that script at boot, get system version and dump LSASS memory	Persistence, Credential access	Write file, Write to registry, Version, Dump process memory	(1) Write file on remote file system, (2) Install a scheduled task on the remote Windows target, (3) Get system version, (4) Dump lsass credentials	Remsec (Strider), Ke3chang
Ransomware	Discover and exfiltrate sensitive files, encrypt them and leave a message	Discovery, Collection, Exfiltration, Impact	Directory listing, Read file, Write file	(1) Find local users, (2) Find sensitive files, (3) Exfiltrate a list of staged files and encrypt them, (4) Encrypt remote files, (5) Write ransom message	APT3, Bad Rabbit (multiple APTs)
Shares Hunter	Read ARP cache to find neighbors, scan them to see which has Netbios/SMB sharing enabled, enumerate shares	Discovery, Lateral Movement	User-mode commands: ipconfig, arp, nbtstat, net view	(1) Find local IP address, (2) Read ARP cache, (3) Scan hosts for SMB/NetBIOS, (4) Enumerate network shares	Conti, APT32

TABLE 7: Coverage of commands of the Laccolith agent, with respect to the adversary profiles.

Profile	Commands
Thief	dir, read
Op-2	write, setkey, version, dump
Ransomware	dir, read, write
Shares Hunter	read, usermode

TABLE 8: Adversary profile execution progress for Laccolith.

Profile	Windows Defender	Avast	AVG	Kaspersky	Avira
Thief	3/3	3/3	3/3	3/3	3/3
Op-2	4/4	4/4	4/4	4/4	4/4
Ransomware	5/5	5/5	5/5	5/5	5/5
Shares Hunter	7/7	7/7	7/7	7/7	7/7

TABLE 9: Injection success for Laccolith.

Anti-virus	Injection Success
Windows Defender	19/20
Avast	17/20
AVG	17/20
Kaspersky	19/20
Avira	18/20
Overall	90/100

is a common target for APTs and is widely used in the adversary emulation landscape. For example, the MITRE ATT&CK framework started as a project to gather information about TTPs against Windows-based systems [56]. It exhibits a diverse attack surface, encompassing a wide range of services, applications, and configurations, which enables a comprehensive exploration of attack vectors. This complexity makes Windows a realistic target for enterprise systems. Other guest OSes can also be targeted by our solution, as discussed later in the paper. We performed experiments with **three state-of-the-art open-source adversary emulation tools**: MITRE CALDERA, Atomic Red Team, and Invoke-Adversary. According to a recent survey [67], these tools are the most mature and are aligned to the MITRE ATT&CK matrix, which allow us to reproduce threat behavior in a reliable way. We evaluated the detectability of adversary emulation against five popular AV products. The **choice of antivirus solutions** can be a limitation of our evaluation. We focused on antivirus solutions that represent those commonly used in real-world scenarios [5], considering the ones that aligned better with the typical deployment context of adversary emulation.

Threats to internal validity. Inconsistent configuration settings for the AVs may threaten the internal validity of the study. We used standard configurations of the five AV solutions, to avoid variations of results not due to the effectiveness of the adversary emulation tools.

Threats to construct validity. The **choice of detection metrics** can impact the validity of the study. We adopted three different metrics for the evaluation of detectability, which encompass all the key aspects of detectability: perimeter breach detection (*injection success*), malicious actions detection (*atomic technique detection*), and timeliness in the APT campaign detection (*adversary profile execution progress*).

5 Related Work

Zilberman *et al.* [67] analyze and classify several adversary emulators. The authors defined a set of criteria and a methodology to evaluate the threat emulators; a taxonomy of the qualities of the threat emulators; and guidelines for choosing an appropriate adversary emulator, given the specific environment and the security assessment tasks. Among the criteria defined in this work are OS compatibility, changes needed in the security array, ATT&CK TTPs coverage, procedures configuration, and required security expertise. In this section, we further analyze and discuss the most popular and mature adversary emulation tools according to the survey [67].

Atomic Red Team [51] is a library of scripts to emulate adversary behavior. Every script implements a single ATT&CK technique or sub-technique (277 out of 719, 231 out of 507 for Windows). It can be used for specific/atomic tests but is not suitable to emulate complex scenarios. *Red Team Automation*

TABLE 10: Adversary emulation solutions comparison.

Tool	C2 Server	Complex Attacks	ATT&CK Tactics Coverage	Needs Pre-Installed Agent	Anti-detection
CALDERA	✓	Built-in, Custom	Collection, Credential Access, Defense Evasion, Discovery, Execution, Exfiltration, Lateral Movement, Persistence, Privilege Escalation	✓	✗
Atomic Red Team	✗	✗	Collection, Credential Access, Defense Evasion, Discovery, Execution, Exfiltration, Impact, Initial Access, Lateral Movement, Persistence, Privilege Escalation	✗	✗
Red Team Automation	✗	Built-in, Custom	Defense Evasion, Discovery, Execution, Lateral Movement, Persistence, Privilege Escalation	✗	✗
APTSimulator	✗	Built-in, Custom	Collection, Credential Access, Defense Evasion, Discovery, Execution, Persistence	✗	✗
Infection Monkey	✓	Built-in	Collection, Credential Access, Defense Evasion, Discovery, Execution, Exfiltration, Lateral Movement, Persistence, Privilege Escalation	✓	✗
Metta	✗	Built-in	Collection, Credential Access, Defense Evasion, Discovery, Execution, Exfiltration	✗	✗
DumpsterFire	✗	Built-in, Custom	Collection, Credential Access, Discovery, Exfiltration, Persistence, Privilege Escalation	✗	✗
Invoke-Adversary	✗	✗	Credential Access, Defense Evasion, Discovery, Execution, Persistence	✗	✗
Sliver	✓	✗	Collection, Defense Evasion, Discovery, Exfiltration, Privilege Escalation	✓	✗
Laccolith	✓	Built-in, Custom	Collection, Credential Access, Defense Evasion, Discovery, Execution, Exfiltration, Impact, Initial Access, Lateral Movement, Persistence, Privilege Escalation	✗	✓

Key:

Built-in	Collection	Execution	Lateral Movement
Custom	Credential Access	Exfiltration	Persistence
	Defense Evasion	Impact	Privilege Escalation
	Discovery	Initial Access	

[18] is a script framework that implements single techniques from the ATT&CK framework (around 50) for security assessment purposes. It does not offer built-in scripts for multi-procedure attacks. *APTSimulator* [48] is a batch script-based tool for Windows that offers around 30 attack techniques to emulate post-compromise scenarios. These techniques leverage external tools such as Mimikatz and PowerSploit. *Infection Monkey* [23] is an adversary emulation tool composed of two main elements, the server (Monkey Island) and the agents (Monkeys). Infection Monkey implements a few ATT&CK techniques, mostly for Initial Access and Lateral Movement. As CALDERA, Infection Monkey does not offer any anti-detection capability to hide malicious actions, and AVs easily detect its agents' activities [67]. *Metta* [60] is an adversary emulator developed by Uber Technologies to assess endpoint security. It provides techniques for various tactics, such as Discovery, Credential Access, and Defense Evasion. However, it does not emulate complete attacks but focuses on the assessment of specific targets. *DumpsterFire* [59] offers a collection of actions (fires) that can be chained together into complex attacks (dumpster fires). Fires are not based on ATT&CK framework techniques and are organized into categories: Network Scans, File Downloads, Websurfing, Account Bruteforcing, Filesystem Activities, Malware, Custom OS Commands, and Shenanigans. It can only execute on Linux. *Invoke-Adversary* [13] is a PowerShell script to test security mechanisms. It offers 39 techniques grouped by ATT&CK tactics. It does not offer multi-procedure scripts and does not support Lateral Movement. Moreover, its procedures are usually detected by AVs, making it not suitable for usage in real-world scenarios. *Sliver* [4] is a C2 framework for adversary emulation. It offers multi-platform agents that communicate with the C2 server using different protocols (e.g., HTTP, DNS, Mutual TLS). It also provides attacker capabilities through plugins called armories. Sliver does not offer any AV-evasion capability as explicitly stated in its documentation.

Table 10 summarizes the comparison among the mentioned adversary emulation solutions. It is possible to notice how the tools that provide an advanced architecture with a C2 server (i.e., CALDERA, Infection Monkey, and Sliver) need to install an agent on the target system. None of the other tools, which do not use an agent, offer command-and-control capabilities. In terms of ATT&CK tactics coverage, Atomic Red Team is the best-performing tool, covering 11 tactics. CALDERA has a similar coverage to Atomic Red Team and is the most complete among the C2 frameworks. CALDERA does not cover Initial Access, because it is not relevant in adversary emulation, and the Impact tactic, unlike our solution which encompasses it in the Ransomware profile. Moreover, CALDERA surpasses the other C2 frameworks regarding complex attacks, providing built-in attacks and capabilities to develop custom ones. CALDERA also provides a plugin to leverage Atomic Red Team tactics and techniques, making it the best-performing tool among the existing ones. It is worth noting that none of the analyzed tools provide anti-detection techniques to evade AV/EDR products.

Our proposed solution differs from the previous ones. Laccolith represents a C2 framework that does not need to explicitly install an agent on the victim machine, unlike CALDERA, Infection Monkey, and Sliver. It also offers profiles based on real-world APTs and mapped to the ATT&CK matrix, covering the same tactics of Atomic Red Team. Most importantly, Laccolith enables the emulation of malicious actions without being detected by AV and EDR solutions. Moreover, it is portable across different versions of the target guest OS and does not require any customization related to the specific AV/EDR. To achieve all these abilities, Laccolith leverages virtualization, widely used in infrastructures for cybersecurity exercises, as illustrated in the following.

Yamin *et al.* [63] performed a literature review on Cyber Range platforms and security testbeds, highlighting that most solutions leverage virtualization to set up security training and assessment environments. The use of virtualization spans multiple domains, such as Internet-of-Things (IoT), autonomous systems, SCADA systems, and critical infrastructures. Beuran *et al.* [2] proposed *CyTrONE*, a framework for cybersecurity training. CyTrONE is equipped with a Learning Management System (LMS) for the trainees and offers a Cyber Range that relies on virtualization technologies. The Cyber Range is instantiated using a variable number of VMs, depending on the specific training scenario. *KYPO* [7] is a platform for cyber defense exercises developed by Masaryk University. It aims to emulate attacks on critical infrastructures in a controlled environment. For this purpose, KYPO leverages cloud computing technologies (e.g., OpenStack), and relies on virtualization for hosts and networks. *DETERlab* [62] is a solution for cybersecurity experimentation developed in the context of the *DETER* project. It provides a flexible and realistic environment for security training and assessment. These goals are fulfilled by leveraging virtualization technologies, which enable easy reconfiguration and scalability. The *National Cyber Range* (NCR) [21] is a facility for cybersecurity testing established by the Defense Advanced Research Projects Agency (DARPA). Its purpose is to provide an environment to design and test new ways to respond to the most recent threat actors. The range combines physical and virtualized resources and networks according to the nature of the simulation.

6 Conclusion

In this work, we presented Laccolith, a hypervisor-based solution for adversary emulation equipped with anti-detection capabilities. Laccolith introduces a new approach to inject an agent into the target machine based on virtualization, an enabling technology for cybersecurity assessment and training infrastructures. Laccolith supports the emulation of advanced cyber-attacks (APTs) that adopt sophisticated techniques to hide their traces. This architecture enables the development of a new generation of cyber-range simulations for critical domains, such as in defense. Moreover, Laccolith can support future research on solutions to detect stealthy adversaries.We performed an experimental analysis to compare Laccolith with state-of-the-art tools for adversary emulation. Our analysis showed that existing tools cannot evade detection by AV/EDR solutions, limiting the usefulness of adversary emulation. We also integrated anti-detection solutions, namely Inceptor, to find out that this combination was still unable to evade defense mechanisms. Instead, Laccolith evaded all the AVs with high reliability, making it suitable for usage in actual emulation scenarios.

A limitation of this work is the focus on a specific guest OS (the victim) and hypervisor. Currently, our prototype supports Microsoft Windows as guest OS and Linux-KVM as hypervisor, since they represent popular solutions for enterprise systems. For the portability of our solution to a different hypervisor, we need APIs from the hypervisor to read and modify the memory of the guest OS. For example, in the case of the XEN hypervisor, the LibVMI library can be leveraged by our solution. If such APIs are not available for the target hypervisor, an alternative is to save and modify a snapshot of the VM memory. For the portability to a different guest OS, we need to rewrite the injector to use kernel-level APIs. This can be easily achieved by taking advantage of the modular architecture of modern OSes, such as, using the APIs for Linux Kernel Modules. About the scalability, our current prototype requires a few minutes for injecting into VMs with a small amount of RAM. For systems with more RAM, the performance can be improved by optimizing the current prototype. In particular, the most expensive step in the injection process is the analysis of memory through Virtual Machine Introspection (VMI). Our prototype leverages Volatility for VMI, which collects large amounts of information that may be unnecessary for the injection. Therefore, our prototype can be further optimized by tailoring Volatility for the injection phase.

Acknowledgments

This work has been partially supported by the Italian Ministry of University and Research (MUR) under the programme "PON Ricerca e Innovazione 2014-2020 – Dottorati innovativi con caratterizzazione industriale" and by MUR PRIN 2022, project FLEGREA (CUP E53D23007950001).

References

[1] Andy Applebaum, Doug Miller, Blake E. Strom, Chris Korban, and Ross Wolf. Intelligent, automated red team emulation. *Proceedings of the 32nd Annual Conference on Computer Security Applications*, 2016.

[2] Razvan Beuran, Dat Tang, Cuong Pham, Ken-ichi Chinen, Yasuo Tan, and Yoichi Shinoda. Integrated framework for hands-on cybersecurity training: Cytrone. *Computers & Security*, 78:43–59, 2018.

[3] Binarly. Design issues of modern EDRs: bypassing ETW-based solutions. https://www.binarly.io/posts/Design_issues_of_modern_EDRs_bypassing_ETW-based_solutions/index.html.

[4] Bishop Fox. Sliver. https://github.com/BishopFox/sliver.

[5] Marcus Botacin, Felipe Duarte Domingues, Fabrício Ceschin, Raphael Machnicki, Marco Antonio Zanata Alves, Paulo Lício de Geus, and André Grégio. Antiviruses under the microscope: A hands-on perspective. *Computers & Security*, 112:102500, 2022.

[6] Brian Caffo. Statistical inference for data science. *British Columbia, UK: Leanpub*, 2016.

[7] Pavel Čeleda, Jakub Čegan, Jan Vykopal, Daniel Tovarňák, et al. Kypo–a platform for cyber defence exercises. *M&S Support to Operational Tasks Including War Gaming, Logistics, Cyber Defence. NATO Science and Technology Organization*, 2015.

[8] Citizen Lab. Tracking GhostNet: Investigating a Cyber Espionage Network. https://citizenlab.ca/wp-content/uploads/2017/05/ghostnet.pdf.

[9] Cornelis De Plaa. Bypass EDR's memory protection, introduction to hooking. https://outflank.nl/blog/2019/06/19/red-team-tactics-combining-direct-system-calls-and-srdi-to-bypass-av-edr/.

[10] MITRE Corporation. MITRE ATT&CK. https://attack.mitre.org/.

[11] CTID. OilRig Adversary Plan. https://github.com/center-for-threat-informed-defense/adversary_emulation_library/tree/master/oilrig.

[12] CTID. OilRig Intelligence Summary. https://github.com/center-for-threat-informed-defense/adversary_emulation_library/blob/master/fin6/Intelligence_Summary.md.

[13] CyberMonitor. Invoke-Adversary. https://github.com/CyberMonitor/Invoke-Adversary.

[14] CyberStruggle. FireEye EDR Bypassed with Basic Process Injection. https://cyberstruggle.org/fireeye-edr-bypassed-with-basic-process-injection/.

[15] Cyware. APT34: The Helix Kitten Cybercriminal Group Loves to Meow Middle Eastern and International Organizations. https://cyware.com/blog/apt34-the-helix-kitten-cybercriminal-group-loves-to-meow-middle-eastern-and-international-organizations-48ae.

[16] R Dhaya, R Kanthavel, and Kanagaraj Venusamy. Dynamic secure and automated infrastructure for private cloud data center. *Annals of Operations Research*, pages 1–21, 2021.

[17] Ege Balci. Shikata-Ga-Nai. https://github.com/EgeBalci/sgn.

[18] Endgame Inc. Red Team Automation. https://github.com/endgameinc/RTA.

[19] MITRE Engenuity. Center for threat-informed defense. https://ctid.mitre-engenuity.org/.

[20] Evan Pena and Casey Erikson. Staying Hidden on the Endpoint: Evading Detection with Shellcode. https://www.mandiant.com/resources/staying-hidden-on-the-endpoint-evading-detection-with-shellcode.

[21] Bernard Ferguson, Anne Tall, and Denise Olsen. National cyber range overview. In *2014 IEEE Military Communications Conference*, pages 123–128. IEEE, 2014.

[22] Tal Garfinkel, Mendel Rosenblum, et al. A virtual machine introspection based architecture for intrusion detection. In *Ndss*, volume 3, pages 191–206. San Diego, CA, 2003.

[23] Guardicore. Infection Monkey. https://www.guardicore.com/infectionmonkey/.

[24] Danny Hendler, Shay Kels, and Amir Rubin. Amsi-based detection of malicious powershell code using contextual embeddings. In *Proceedings of the 15th ACM Asia Conference on Computer and Communications Security*, pages 679–693, 2020.

[25] Hoang Bui. Bypass EDR's memory protection, introduction to hooking. https://medium.com/@fsx30/bypass-edrs-memory-protection-introduction-to-hooking-2efb21acffd6.

[26] InfoSec. Evade EDR with Shellcode Injection and gain persistence using Registry Run Keys. https://infosecwriteups.com/evade-avs-edr-with-shellcode-injection-159dde4dba1a.

[27] Intel. XCHG - Exchange Register/Memory with Register. https://www.cs.princeton.edu/courses/archive/spr18/cos217/reading/x86-64-2.pdf.

[28] Kaspersky. Carbanak APT: The Great Bank Robbery. https://media.kasperskycontenthub.com/wp-content/uploads/sites/43/2018/03/08064518/Carbanak_APT_eng.pdf.

[29] klezVirus. Inceptor. https://github.com/klezVirus/inceptor.

[30] klezVirus. Inceptor – Bypass AV-EDR solutions combining well-known techniques. https://github.com/klezVirus/inceptor/blob/main/slides/Inceptor%20-%20Bypass%20AV-EDR%20solutions%20combining%20well%20known%20techniques.pdf.

[31] Daniel Kouril, Tomáš Rebok, Tomas Jirsik, Jakub Cegan, Martin Drasar, Martin Vizváry, and Jan Vykopal. Cloud-based testbed for simulation of cyber attacks. In *2014 IEEE Network Operations and Management Symposium (NOMS)*, pages 1–6. IEEE, 2014.

[32] LibVirt. LibVirt. https://libvirt.org/.

[33] Malwarebytes Labs. APT34 targets Jordan Government using new Saitama backdoor. https://www.malwarebytes.com/blog/threat-intelligence/2022/05/apt34-targets-jordan-government-using-new-saitama-backdoor.

[34] Mandiant. M-Trends 2021. https://www.mandiant.com/resources/m-trends-2021.

[35] Mandiant. New Targeted Attack in the Middle East by APT34, a Suspected Iranian Threat Group, Using CVE-2017-11882 Exploit. https://www.mandiant.com/resources/blog/targeted-attack-in-middle-east-by-apt34.

[36] Mandiant. Shikata Ga Nai Encoder Still Going Strong. https://www.mandiant.com/resources/blog/shikata-ga-nai-encoder-still-going-strong.

[37] Markus Fleschutz. Collection of Powershell scripts. https://github.com/fleschutz/PowerShell/blob/master/Scripts/encrypt-file.ps1.

[38] Microsoft. API reference docs for Windows Driver Kit (WDK). https://learn.microsoft.com/en-us/windows-hardware/drivers/ddi/.

[39] Microsoft. Microsoft Defender Application Guard overview. https://learn.microsoft.com/en-us/windows/security/threat-protection/microsoft-defender-application-guard/md-app-guard-overview.

[40] Microsoft. Microsoft Defender SmartScreen. https://learn.microsoft.com/en-us/windows/security/threat-protection/microsoft-defender-smartscreen/microsoft-defender-smartscreen-overview.

[41] MITRE. Account Discovery, T1087. https://attack.mitre.org/techniques/T1087/.

[42] MITRE. CALDERA. https://github.com/mitre/caldera.

[43] MITRE. Exfiltration Over Alternative Protocol, T1048. https://attack.mitre.org/techniques/T1048/.

[44] MITRE. OS Credential Dumping, T1003. https://attack.mitre.org/techniques/T1003/.

[45] MITRE. Phishing, T1566. https://attack.mitre.org/techniques/T1566/.

[46] MITRE. Use Alternate Authentication Material: Pass the Hash, T1550.002. https://attack.mitre.org/techniques/T1550/002/.

[47] MITRE. User Execution: Malicious File, T1204.002. https://attack.mitre.org/techniques/T1204/002/.

[48] Nextron Systems GmbH. APTSimulator. https://github.com/NextronSystems/APTSimulator.

[49] Paranoid Ninja. CarbonCopy. https://github.com/paranoidninja/CarbonCopy.

[50] Qemu. Qemu. https://www.qemu.org/.

[51] Red Canary. Atomic Red Team. https://atomicredteam.io/.

[52] RedHat. Linux-KVM. https://www.linux-kvm.org/page/Main_Page.

[53] Nathan Regola, Nitesh V Chawla, et al. Storing and using health data in a virtual private cloud. *Journal of medical Internet research*, 15(3):e2076, 2013.

[54] Kevin Schoonover, Eric Michalak, Sean Harris, Adam Gausmann, Hannah Reinbolt, Daniel R Tauritz, Chris Rawlings, and Aaron Scott Pope. Galaxy: a network emulation framework for cybersecurity. In *11th {USENIX} Workshop on Cyber Security Experimentation and Test ({CSET} 18)*, 2018.

[55] Spot the Planet. Bypassing Cylance and other AVs/EDRs by Unhooking Windows APIs. https://www.ired.team/offensive-security/defense-evasion/bypassing-cylance-and-other-avs-edrs-by-unhooking-windows-apis.

[56] Blake E Strom, Andy Applebaum, Doug P Miller, Kathryn C Nickels, Adam G Pennington, and Cody B Thomas. Mitre att&ck: Design and philosophy. In *Technical report*. The MITRE Corporation, 2018.

[57] Blake E Strom, Joseph A Battaglia, Michael S Kemmerer, William Kupersanin, Douglas P Miller, Craig Wampler, Sean M Whitley, and Ross D Wolf. Finding cyber threats with att&ck-based analytics. *The MITRE Corporation, Bedford, MA, Technical Report No. MTR170202*, 2017.

[58] Symantec Security Response. W32.Stuxnet Dossier. https://docs.broadcom.com/doc/security-response-w32-stuxnet-dossier-11-en.

[59] TryCatchHCF. DumpsterFire. https://github.com/TryCatchHCF/DumpsterFire.

[60] Uber Technologies Inc. Metta. https://github.com/uber-common/metta.

[61] Volatility Foundation. Volatility. https://www.volatilityfoundation.org/.

[62] John Wroclawski, Terry Benzel, Jim Blythe, Ted Faber, Alefiya Hussain, Jelena Mirkovic, and Stephen Schwab. Deterlab and the deter project. *The GENI Book*, pages 35–62, 2016.

[63] Muhammad Mudassar Yamin, Basel Katt, and Vasileios Gkioulos. Cyber ranges and security testbeds: Scenarios, functions, tools and architecture. *Computers & Security*, 88:101636, 2020.

[64] Pavel Yosifovich, David A Solomon, and Alex Ionescu. *Windows Internals, Part 1: System architecture, processes, threads, memory management, and more*. Microsoft Press, 2017.

[65] Zerosum0x0. "Heresy's Gate": Kernel Zw*/NTDLL Scraping + "Work Out": Ring 0 to Ring 3 via Worker Factories. https://zerosum0x0.blogspot.com/2020/06/heresys-gate-kernel-zwntdll-scraping.html#workout.

[66] Zerosum0x0. Zerosum0x0 GitHub repository. https://github.com/zerosum0x0-archive/archive.

[67] Polina Zilberman, Rami Puzis, Sunders Bruskin, Shai Shwarz, and Yuval Elovici. Sok: A survey of open-source threat emulators. *arXiv preprint arXiv:2003.01518*, 2020.

Vittorio Orbinato is a PhD Student at Università degli Studi di Napoli Federico II, Naples, Italy. His research interests include adversary emulation and software security. He received his M.Sc. Degree from Università degli Studi di Napoli Federico II, Naples, Italy.

Marco Carlo Feliciano is a Security Researcher at Secureware s.r.l. His research interests are in adversary emulation. He received his M.Sc. Degree from Università degli Studi di Napoli Federico II, Naples, Italy.

Domenico Cotroneo (PhD) is Full Professor at Università degli Studi di Napoli Federico II, Naples, Italy. His research interests include software fault injection, dependability assessment, and field-based measurement techniques. He is co-founder and scientific consultant at Secureware s.r.l.

Roberto Natella (PhD) is Associate Professor at Università degli Studi di Napoli Federico II, Naples, Italy. His research interests are in software security and dependability, with main recurring theme on the experimental injection of faults, attacks, and stressful conditions in software systems. He is co-founder and scientific consultant at Secureware s.r.l.