Dizang: A solution for collecting forensic evidences in cloud environments

Hamilton J. S. Fonte II, Marcos A. Simplicio Jr., Edson S. Gomi
Escola Politécnica, Universidade de São Paulo (USP) São Paulo, SP, Brasil
Email: hamiltonii@gmail.com, mjunior@larc.usp.br, gomi@usp.br

***Abstract*—Cloud architectures are increasingly more common, as is the number of security issues involving this technology. Unfortunately, due to the volatile nature of virtualized resources in the cloud, the task of gathering evidences for forensic analysis currently faces practical and legal challenges. In this work, we address this issue by analyzing proposals aimed at meeting such challenges, discussing their limitations and then presenting a solution to overcome them. The proposal specifically focuses on the reproducibility of the collection process in virtualized environments, without violating jurisdictions or the privacy of those not involved in the investigation. As such, it should be a useful tool for analyzing the causes of security incidents in cloud computing systems.**

I. Introduction

Virtualization techniques, replication of services and resource sharing among multiple users (multitenancy) are key enablers for the high scalability of computational clouds [1]. At the same time, however, these mechanisms also lead to the volatility of the virtual resources executing cloud-based applications. After all, when submitted to a high load, a cloud application may create clones of the virtual machines (VMs) or containers hosting it, and then balance the load among those copies; this auto-scaling behavior is expected to avoid any degradation of the quality provided by the service. After the load subsides, the cloned instances are usually deactivated, their resources are released and the system returns to the previous capacity, thus avoiding unnecessary costs.

Despite interesting from the efficiency and cost viewpoints, such volatility of the cloud is likely to causes problems from the perspective of attack response teams and forensic experts. For example, suppose that a temporary virtual processing instance undergoes an attack that directly affects its memory, without leaving traces in permanent storage (e.g., log files). In this case, the evidences of this event may be completely lost after such instances are deactivated and their resources are released. This issue is further aggravated by aspects such as multitenancy and multi-jurisdiction, typical of cloud solutions [2]. Particularly, the multitenancy aspect makes it harder to isolate the hardware executing the applications of interest: since each piece of hardware is shared by a number of users, physically removing them for analysis could lead to privacy violation of users not related to the investigation. Moreover, due the distributed nature of the cloud, data relevant to the investigation may be allocated in different countries. In practice, this encumbers the acquisition of the corresponding information, especially when there are no cooperation agreements among the entities involved [3]. Combined, these characteristics hinder the implementation of an evidence collection process. As a result, the response to memory-oriented attacks may be delayed, and evidences collected afterward may not have the necessary credibility so that they can be accepted in legal processes. In particular, it becomes harder for forensic experts to comply with privacy, jurisdiction and chain of custody requirements, as well as to ensure the reproducibility of the collection process [4].

Even though the literature include solutions aimed at collecting information in the cloud for forensic analysis, most of them handle collection, transport and storage in an isolated manner. For example, [5] and [6] deal with factors such as multitenancy and multi-jurisdiction, discussing forms of collecting and preserving evidence outside the cloud. In comparison, [7] concentrates on forensic analysis to collect evidence from VMs while they are being executed, whereas [8] deals with ensuring the chain of custody when transporting evidences. However, none of the proposals identified in the literature (1) describe how data are collected and stored observing the chain of custody, and (2) create the required conditions for reproducing the evidence collection process even if a virtualized resource is decommissioned.

Aiming to overcome these limitations, this work presents Dizang, a proposal focusing on: (1) the reproducibility of the collection process; (2) establishing a link between the evidence collected and its origin (assuming the cloud resource is univocally identifiable); and (3) preserving the jurisdiction and the privacy of those not involved in the investigation. As such, the proposed solution can be used as tool for incident management, safeguarding evidences for analysis during a post-incident stage. In addition, these characteristics contribute to the credibility of collected evidences and, thus, to their acceptability in a possible legal process. When compared with similar-purpose solutions, Dizang is particularly useful against code injection attacks [9], which would normally leave no traces when cloud-based virtual instances are deactivated and their memory is released [10], [9].

The rest of this paper is organized as follows. Section II briefly discusses cloud solutions and their characteristics. Section III analyzes the related works, in particular those focused on forensic analysis of (virtualized) memory information. Section IV details the proposed solution and its features. Section VI presents our final considerations.

II. Problem statement: virtualization vs. forensics in the cloud

Cloud computing refers to a model that provides network access to a configurable amount of computing resources, in such a manner that users can allocate and release computational resources on demand and with minimal management effort [11]. Depending on which resources are provided to clients (also called tenants), three main cloud service models can be defined [11]: software as a service (SaaS), in which the software to be used by the clients is provided; platform as a service (PaaS), in which an environment is provided for clients to develop, test and execute their software; and infrastructure as a service (IaaS), in which basic computational resources are provided (e.g., processing, memory and storage), usually by means of virtualization. In this work, we focus on the IaaS service model, where clients have further control over the underlying cloud resources.

The traditional way of virtualizing IaaS resources in the cloud is to rely on virtual machines (VMs) [12]. More recently, however, there has been an increasing interest in using containers for this purpose. Indeed, according to a study conducted in 2016 with 235 companies involved with software development [13], 76% of the respondents used containers to improve the efficiency of their development process and of their cloud micro-services architecture. However, whereas VMs involve instantiating a virtual hardware and also an operating system (OS) on top of the native system, virtualization with containers is conducted at the level of the native OS. According do [12], containers allow for a simpler implementation and better resource usage, eliminating layers between the application being executed and the physical hardware. They also have a lower total cost and a more predictable performance.

Whichever the virtualization technology employed, the result is a highly volatile memory environment. This happens because the cloud's on-demand nature implies that computational resources are allocated and released following the actual system's load. Therefore, it is not uncommon that attack response teams are unable to access all possible evidences of a breach because the pieces of volatile memory containing those evidences have already been released as part of the cloud's automatic scaling policies. From a forensic perspective, this is specially troublesome when dealing with injection attacks that operate directly on the target's volatile memory, without affecting the long term storage of VMs or containers [9]. Examples include:

- *Remote library injection*: A malicious process forces the target process to load a library into its memory space [14]. As a result, that code inside that library is executed wit the same privileges as the target process. This strategy, commonly used for enabling a malware to be installed in a victim's machine, may also involve storing the malicious library in the system so it can be loaded by different processes.
- *Inline Hooking*: A malicious process writes a piece of code directly into the target process's memory space, as a sequence of bytes, so the victim that code as if it was part of its own design [15]. This is commonly employed to force the execution of shell scripts, giving the attacker remote control over the target machine.
- *Reflective library injection*: a malicious process directly access the target process's memory and writes into it the bytecode corresponding to a library, forcing the victim to execute the injected instructions [16]. In this case, the malicious library is not stored in the system and its loading is not registered by the operation system's logs, making this kind of attack harder to detect.

The ability to analyze the state of volatile memory is, thus, an important requirement for enabling an effective incident response procedure, as well as post-mortem analyses of such attacks. However, it is challenging task to balance the conflicts between (1) this security need and (2) the cloud's performance requirements, usually met via rapid elasticity. In the next section, we discuss some of the proposals in the literature aimed at addressing this issue.

III. Related works

There are a few aspects that need to be taken into account when performing forensic analysis in the cloud, including: their approach for information collection, the reproducibility of this process, which guarantees are provided for the evidence's chain of custody, and how privacy and jurisdiction are preserved. For a structured discussion, in this section we analyze the works available in the literature considering these four aspects (see Table I).

A. Continuous collection of relevant data

The evidence-gathering process in traditional forensics consists basically in isolating a crime scene and then collecting all data that seem relevant for later analysis. A straightforward translation of this practice to the context of digital forensics is, thus, performing the bitwise copy of the (virtual or physical) machines's memory. In the past, when computing services manipulated much smaller amounts of memory, disk and traffic than what is seen today, such practice was not considered very problematic. However, in today's cloud systems and applications, the volume of data is considerably larger [23]. After all, one of the main advantages of cloud solutions is exactly to facilitate access to storage and processing resources, such as virtual machines, load balancers, firewalls and other evidence-generating elements. The result is that digital forensics laboratories are often overloaded, with backlogs that may reach a few months [23]. An important step for enabling more efficient incident response and forensic investigations is, thus, finding a way to store less, more relevant information for analysis.

Among the solutions found in the literature, the only proposals for digital forensics that clearly consider this need are [21], [8] and [22]. Unfortunately, however, they all have limitations when considering the particularities of the cloud scenario. Namely, [21] focuses on mobile environments, and has the disadvantage of not storing the history of memory changes;

Table I
SOLUTIONS FOR COLLECTING MEMORY INFORMATION FROM CLOUD MACHINES FOR FORENSIC ANALYSIS

	Continuous collection of relevant data	Chain of custody guarantees	Ability to reproduce the evidence collection process	Jurisdiction and privacy preservation
Dizang (this proposal)	✓	✓	✓	✓
[7]	✘	✘	✘	✓
[17]	✘	✘	✘	✓
[5]	✘	✘	✘	✓
[18]	✘	✘	✘	✓
[6]	✘	✓	✘	✓
[8]	✓	✓	✘	✓
[19]	✘	✘	✘	✓
[20]	✘	✘	✘	✓
[21]	✓	✘	✘	✓
[22]	✓	✓	✘	✓

hence, it would not cope with the high volatility of the cloud. Conversely, the cloud-oriented work from [8] constantly collects information from virtual machines, not distinguishing between what happened before or after the fact of interest. This is likely to generate a large amount of (not necessarily useful) data for analysis. Finally, [22] relies on indexing, preprocessing and knowledge sharing of the evidence to optimize analysis time. Although this approach limits the amount of data gathered in the short term, it still allows its continuous growth; therefore, in the long run, backlogs are still likely to accumulate.

B. Chain of custody garantees

One important aspect in (digital) forensics is how the process by means of which evidence is handled. Specifically, evidences must collected, transported and stored in such a manner that it would be acceptable in a legal process, rather than having their credibility questioned later. To achieve this goal, digital forensic tools must ensure the evidence's chain of custody.

In a physical infrastructure, the collection of evidence can be done by removing the physical resource, transporting it to a laboratory and thereby analyzing the data. To limit the evidence's exposure to tampering, it may be kept in a safe room, to which access is controlled.

Conversely, in a cloud computing scenario, a number of new challenges appear. In contrast with traditional infrastructures, physical resources shared among different cloud tenants cannot be removed due to privacy issues. Preserving evidence's integrity while it is collected, transported and stored becomes, thus, another task that requires some care to ensure modification attempts do not go unnoticed. Finally, due to the volatility of computational resources, the verification of an evidence's origin becomes a complex process, in particular after the resource that generated the evidence ceases to exist [24]. Violation of any of these characteristics may put the evidence's credibility into question, hindering its admissibility by a judging authority.

Among the papers analyzed, only [8] and [22] address the issue of chain-of-custody, although not completely. Specifically, [8] employs hashes to check the integrity of the evidence, allowing the detection of changes. One limitation, however, is that it does not clarify the mechanisms that should be employed to prevent unauthorized access (and, thus, potential change) to the hashes themselves. Meanwhile, [22] display some concerns with who is allowed access to the evidence, but does the solution does not describe how the evidence is transported or its integrity is secured. In comparison, the other proposals listed in Table I focus only on the technical aspects of data collection, without discussing in detail the chain of custody. In general, these works only mention the need of a forensically acceptable collection process, without discussing how that would be conceived.

C. Ability to reproduce the evidence collection process

If, during a forensic analysis, different analysts obtain distinct results when executing the same collection procedure, the evidence generated has no credibility and may be considered unacceptable in a legal process. For this reason, the reproducibility of the collection process is an important part in evidence generation for forensic analysis.

None of the proposals found in the literature allows such reproducibility in cloud scenarios in which virtual instances are deactivated and their physical resources are released. More precisely, they all depend on the existence of the virtual resource for repeating the collection process.

D. Jurisdiction and privacy preservation

As aforementioned, one challenges of digital forensics in a public cloud environment is that removing the hardware for later analysis may lead to privacy violations. After all, in such a multitenant scenario, the same physical machine stores information about different clients, some of which may not be involved in the investigation in course.

Different works in the literature adequately deal with this specific issue by means of two main strategies. The first, adopted by [6], [7], [17], [5], involves collecting data pertinent to the investigation and storing them outside the cloud. The second, employed in [8] and consisting in a specific case of [7], depends on the cooperation of the cloud service provider for obtaining the information necessary to the investigation.

Depending on the cloud service provider, however, the latter is not a highly recommended strategy, for at least two reasons: (1) the volume of user's data may force the providers to limit the size of the logs stored; and (2) in case of unavailability due to attacks, the goal of the provider will most likely be restoring the service, not necessarily preserving evidences [25].

IV. Proposed solution: Dizang

The main goal of the hereby proposed solution, called Dizang, is to enable evidence collection in highly volatile cloud environments while: (1) identifying the evidence source, even if the virtual resource no longer exists; (2) describing the system before and after the incident; (3) transporting and storing the data collected so as to guarantee its integrity and confidentiality; and (4) not violating the jurisdiction and privacy of other users that might have resources allocated in the same physical server.

The mechanisms employed by Dizang for these purposes are described in detail in this section.

A. Resource identification

In a physical (i.e., non-virtualized) infrastructure, a direct association can usually be made between any resource and its corresponding origin. This is the case, for example, of memory information, disk images and packets traveling in the network. In contrast, when the system is built upon a virtual infrastructure, especially when it is self-scalable, the high volatility of computational resources can lead to their removal at any time. When that occurs, it becomes hard to trace the origin of any data generated by such resources.

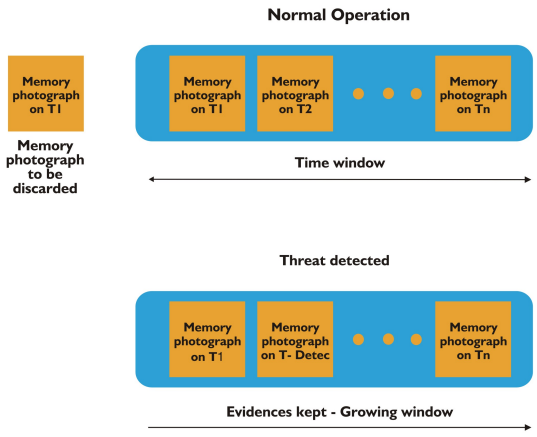
To address this issue, cloud-oriented digital forensic solutions require cloud virtualized resources to be unequivocally identifiable. Fortunately, most container-based cloud environments already provide such capability: usually, a container's identifier corresponds to a hash calculated over that container's contents. This contrasts with VM-based implementations, in which random alphanumeric numbers are commonly used as identifiers. Hence, and even though a VM-based implementation of Dizang would also be possible, the proposed architecture focus on container-oriented clouds. In particular, Dizang can take advantage of Linux containers (LXC) to persist the source-evidence relationship. Basically, LXC uses *cgroups* and *namespacing* of the Linux kernel for managing and isolating virtual resources; the resulting containers, albeit still volatile, univocally identify the compiled images being executed.

B. Memory copying

Memory copying is usually not an atomic activity, since it is executed jointly with processes that may be modifying the very memory parts being copied. Hence, if one of such processes maliciously erases traces of its existence from the container memory, important information may eventually be lost.

Aiming to improve the atomicity of a memory copy process, Dizangtemporarily interrupts the execution of the container, makes a copies its contents, and then resumes its execution. This technique is similar to that adopted by [27] for VMs, producing a photograph of the container's volatile memory for latter analysis.

Figure 1. Sliding window of evidence collection



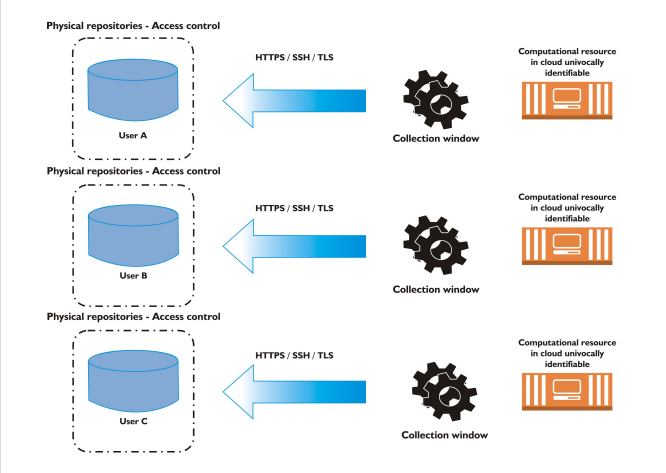
As long as such data collection process is repeated periodically, in short time intervals, it is possible to build an accurate history of the container's memory state during execution. However, as discussed in Section **??**, this may also lead to a huge volume of information to be stored and analyzed. To mitigate this issue, Dizang employs a time window that limits the number of memory photographs stored while the system operates under normal circumstances, i.e., while it is not under attack. Memory copies reaching a given age are periodically discarded, opening space for new photographs. However, after an attack event is detected (e.g., by an intrusion detection system), Dizang stops discarding older pieces of memory from the monitoring log. This approach, illustrated in Figure 1, gives forensic analysts the ability to analyze memory copies from before and after the intrusion, as well as assess the system's evolution during this period.

C. Secure data storage

Aiming to persist the evidence-origin relationship, as well as to guarantee data integrity, Dizang calculates the hash H of the pair {memory photograph, resource identifier} and stores the triple {H, memory photograph, resource identifier}. The place of storage should be defined by the client and the cloud provider, and included in a service-level agreement (SLA). If such SLA dictates that evidences must be stored outside the cloud environment, it is important to have the data transported using a secure channel (e.g. via *Transport Layer Security* – TLS [29]).

V. Experimental analysis

Figure 2. Dizang's general architecture. MARCOS: Não precisa copiar 3x os mesmos blocos para explicar a solução... 1x basta... E por favor (1) aumente a fonte do texto na figura e (2) crie uma imagem vetorizada em pdf (e.g., no Visio dá para exportar como pdf)



The Dizang architecture was implemented in a testing platform, aiming to assess its efficacy and efficiency. The testbed employed, illustrated in Fig. 2, consists in creating a t2.micro instance (3.3 Mhz, 1GiB RAM and Linux 64-bit) at Amazon Web Services (AWS). In this AWS instance, we installed Docker Engine 1.10 and API Docker 1.21, and then created 3 containers for executing the nginx 1.0 web server in different ports. Using a Java application that discovers the process identifier associated to each container MARCOS: o que seria essa "a Java application"? Você que desenvolveu? Tem que deixar claro se você usou algo ou desenvolveu esse algo, we accessed the content MARCOS: of that container's (é isso mesmo?? É do container ou é do sistema como um todo?) *Non-Uniform Memory Allocation descriptor* (**/proc/pid/numa_maps**), containing the allocation of the memory pages, the nodes associated to these pages and what was allocated in their respective access policies [30]. The copy and recording of the file is such that, every minute, the application (1) pauses the container in question, (2) copies the **numa_maps** directory, (3) concatenates the data obtained with the container's image identifier, (4) calculates the hash H for this set, and (5) saves the result in a **.mem** file named after the container's identifier.

The secure transportation of the evidence to a physical storage outside AWS employs another t2.micro instance, whereby an OpenVPN server is installed. As a basic form of access control, the EC2 instance containing the evidence is configured to accept connections solely from machines in the VPN. A physical machine outside AWS then uses an OpenVPN client to establish a VPN connection with the instance containing the evidence, obtaining the corresponding data. After concluding the data acquisition process, the physical machine verifies whether there are local .mem files that older than a certain time interval, and discards them.

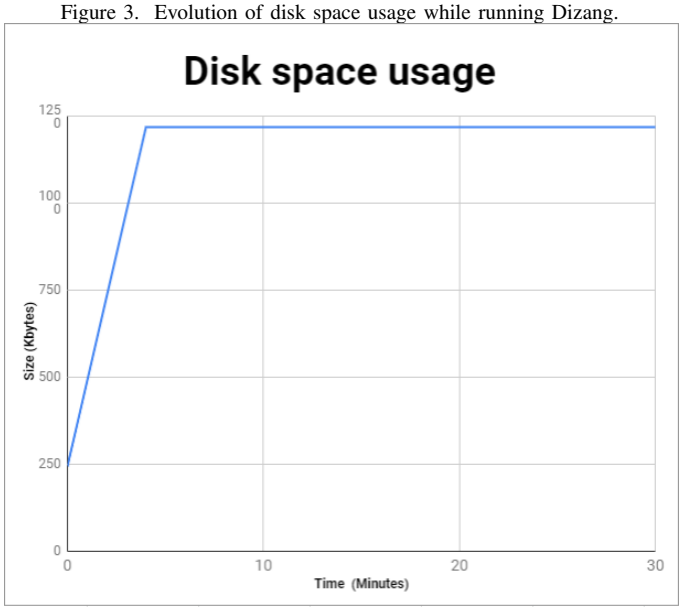
Using this testbed, two experiments were performed: the first focus on performance aspects, whereas the second evaluates Dizang's ability to aid in identifying malware injection attacks. Both experiments are described in the following subsections.

A. Performance

For the first experiment, the system was configured to collect memory photographs in 1-minute intervals and send them to the physical machine outside the cloud. The data collection time window was set to 5 minutes, meaning that samples older than that are erased from the physical machine's disk. The system was then executed for 30 minutes, and the following metrics were collected: (1) the disks space required for storing the memory photographs; (2) the time necessary for pausing a container and copying its contents; and (3) the time taken for transporting the evidence to the physical machine outside the cloud.

The disk space occupied by the memory photographs captured during the experiment, obtained via the *Unix* command `du -sh *.mem`, is shown in Fig. 3. This figure shows that, as expected, the disk space usage increases linearly until the time window is reached, and then stays constant due to older samples being discarded MARCOS: Qual o tamanho de 1 sample? Essa info me parece relevante mencionar aqui. The discarding process can be halted at any time by the physical machine itself (e.g., after an attack is detected). It is, thus, possible to describe the state of the system before and after an incident is detected, even after the corresponding cloud instances are erased.

Figure 3. Evolution of disk space usage while running Dizang.



A potential issue of the proposed solution is that pausing a container to collect data may, at least in principle, cause efficiency losses of the application being executed. To assess this overhead, the times for copying the container memory

Figure 4. Time for copying the container MARCOS: Aqui é “median” (=mediana) mesmo, ou é “average” (=média)? São duas coisas diferentes do ponto de vista de estatística.

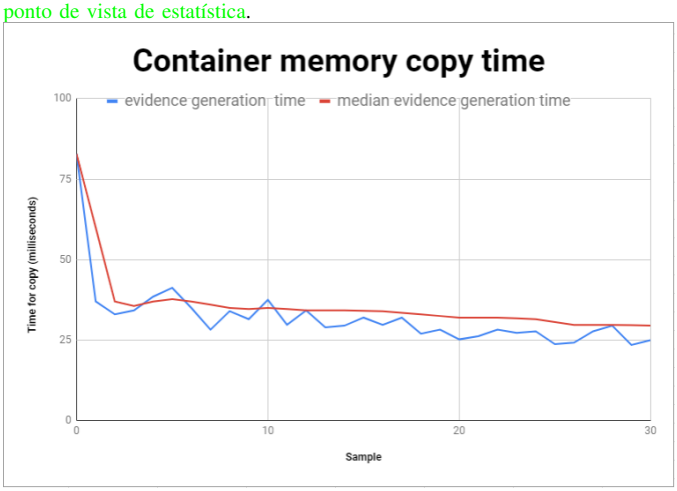


Figure 5. Evidence transport time

were measured along the experiment. The results are shown in Figure 4. Note that, after the application is initialized MARCOS: O que você quer dizer com “after the application is initialized”? Qual application (Dizang? WebServer? Outro?)? Aliás, se remover esse pedaço da sentença dá na mesma não? Enfim, é lógico que o tempo para fazer a cópia é medido depois de inicializado o processo de cópia (donde a dúvida de qual é a “aplicação” que você menciona), the time for making a copy is quite short, varying between 20 and 40 milliseconds. Especially for containers executing the engine of dynamic web pages, as is the case of the experiment in question, this latency can be not very perceptible by end users. For the cases in which the interruption of the computational resource's execution for a short time causes problems of availability, it is possible to perform the collection procedure in separate moments of time. Thus, instead of suspending the execution of all computational resources to carry out the collection simultaneously, the procedure interrupts them sequentially. MARCOS: Você quer dizer que aproveitaria o load-balancing e pararia 1 instância das 3 a cada instante, em round-robin? Eu entendi isso do texto, mas não ficou 100% claro se é isso mesmo

Another concern is the time for transporting the evidence to the storage outside the cloud. The evidence transport to the storage outside the cloud can take longer than generation interval, leading to evidence loss during transport. To assess this impact, the evidence transport times for storage outside the cloud were measured along the experiment, as shown by the results in the graph of Fig. 5. The transport time is verified to stabilize after the window size is reached. The time for transporting evidence is approximately 30 seconds on average. The topology is a factor that may contribute both positively and negatively to the transport time. In this experiment, the evidence generator is in North America, whereas the physical machine where the evidence was transported to is in South America.

A second experiment aimed to determine if it is possible, through the analysis of the collected evidence, to identify malware injection in the container memory was made. To this end, a library named **libexample.so** simulating a malware in the form of a library was injected into one of the containers. After five minutes of Dizangcollecting evidence, the library mentioned berfore was injected into the memory of one of the containers. After the injection the solution was allowed to continue collecting memory snapshots for another 5 minutes. In addition to collecting directory content **/proc/pid/numa_maps**, a raw copy of the process memory of the container was also performed using the Linux *GDB* utility via the command described in Figure 6.

Two distinct moments were compared in the life of the container, before and after the injection of the library. Figure 7 shows part of the memory before the injection and Figure 8 shows part of the memory after the injection. It is possible to see the injected **libexample.so** library between addresses **7fbb52a50000** and **7fbb52c51000** in the snapshot after the injection. Therefore, it is possible to identify the injection of a malware in the form of a library via evidence collected by Dizangfrom the **/proc/pid/numa_maps** directory via comparison between two memory snapshots. This allows code injection attacks to be identified.

Despite the successful collection by copying the **/proc/pid/numa_maps**, it was not possible to copy the contents of the process memory from the container via *ptrace*. According to [31], this happens because the system calls used by tools such as *ptrace* and *htop* were created before the implementation of *cgroups* in the linux *kernel* and thus are not aware of the existence of isolation between processes. When *ptrace* tries to access an area of memory isolated by *cgroups*, the kernel sends a memory access violation signal, the result is shown in Figure 9. The Docker Engine [32] documentation mentions the command `--cap-add=SYS_PTRACE --security-opt-seccomp=unconfined` which

Figure 6. Command to generate a raw copy of the process container memory

```
#reading the memory of a process inside a conteiner
grep rw-p / proc/$1/maps | sed -n's/^\([0-9a-f]*\)-\([0-9a-f]*\) .*$/\1 \2/p'|while read sta
rt stop; do gdb --batch -- pid $1 - ex "dump memory $1- $start-$stop.dump 0x$start 0x$stop";done
```

Figure 7. Part of **/proc/pid/numa_maps** file BEFORE injection

```
00400000 default file=/root/ target mapped=1 N0=1
00600000 default file=/root/ target anon=1 dirty=1 N0=1
00601000 default file=/root/ target anon=1 dirty=1 N0=1
7f6ea0013000 default file=/lib/x86_64-linux-gnu/libc-2.19.so mapped=219 mapmax=2 N0=219
7f6ea01d1000 default file=/lib/x86_64-linux-gnu/libc-2.19.so
7f6ea03d1000 default file=/lib/x86_64-linux-gnu/libc-2.19.so anon=4 dirty=4 N0=4
7f6ea03d5000 default file=/lib/x86_64-linux-gnu/libc-2.19.so anon=2 dirty=2 N0=2
7f6ea03d7000 default anon=3 dirty=3 N0=3
7f6ea03dc000 default file=/lib/x86_64-linux-gnu/ld-2.19.so mapped=35 mapmax=2 N0=35
7f6ea05f7000 default anon=3 dirty=3 N0=3
7f6ea05f7000 default
7f6ea05fe000 default file=/lib/x86_64-linux-gnu/ld-2.19.so anon=1 dirty=1 N0=1
7f6ea05ff000 default file=/lib/x86_64-linux-gnu/ld-2.19.so anon=1 dirty=1 N0=1
7f6ea0600000 default anon=1 dirty=1 N0=1
7ffef8084000 default stack anon=3 dirty=3 N0=3
7ffef80ed000 default
7ffef80ef000 default
Ô•Ù_^T<87>o<80><9d>~›Ã*Í~^Z<9b>íEkI^Fß, ü0¡ê®½>
```

Figure 8. Part of **/proc/pid/numa_maps** file AFTER injection

```
00400000 default file=/root/target mapped=1 N0=1
00600000 default file=/root/target anon=1 dirty=1 N0=1
00601000 default file=/root/target anon=1 dirty=1 N0=1
7fbb52687000 default file=/lib/x86_64-linux-gnu/libc-2.19.so mapped=215 mapmax=2 N0=215
7fbb52845000 default file=/lib/x86_64-linux-gnu/libc-2.19.so
7fbb52a45000 default file=/lib/x86_64-linux-gnu/libc-2.19.so anon=4 dirty=4 N0=4
7fbb52a49000 default file=/lib/x86_64-linux-gnu/libc-2.19.so anon=2 dirty=2 N0=2
7fbb52a4b000 anon=3 dirty=3 N0=3
7fbb52a50000 default file=/root/ libexample.so mapped=1 N0=1
7fbb52a51000 default file=/root/ libexample.so
7fbb52c50000 default file=/root/ libexample.so anon=1 dirty=1 N0=1
7fbb52c51000 default file=/root/ libexample.so anon=1 dirty=1 N0=1
7fbb52c52000 default file=/lib/x86_64-linux-gnu/ld-2.19.so mapped=35 mapmax=2 N0=35
7fbb52e6c000 default anon=3 dirty=3 N0=3
7fbb52e72000 default anon=1 dirty=1 N0=1
7fbb52e74000 default file=/lib/x86_64-linux-gnu/ld-2.19.so anon=1 dirty=1 N0=1
7fbb52e75000 default file=/lib/x86_64-linux-gnu/ld-2.19.so anon=1 dirty=1 N0=1
7fbb52e76000 default anon=1 dirty=1 N0=1
7ffe5507b000 default stack anon=3 dirty=3 N0=3
7ffe550bf000 default
7ffe550c1000 default
ïï1<9e>>>~âç<82><86>1pïF<9b>Hx"ÕÒx^D7 ^BBeÔv<8b>^U<8c>
```

Figure 9. Error while attempting to copy the memory of a process inside a container

```
hamilton@Linux-VirtualBox:~/Documentos/pos-memreader$ sudo ./dumper .sh 4465
/root/target: Arquivo ou diretório não encontrado
/root/target: Arquivo ou diretório não encontrado
Não é póssivel acessar a memória no endereço 0xba9d5000
/root/target: Arquivo ou diretório não encontrado
Não é póssivel acessar a memória no endereço 0xba9d7000
/root/target: Arquivo ou diretório não encontrado
Não é póssivel acessar a memória no endereço 0xbabdd000
/root/target: Arquivo ou diretório não encontrado
Não é póssivel acessar a memória no endereço 0xbadf8000
/root/target: Arquivo ou diretório não encontrado
Não é póssivel acessar a memória no endereço 0xbadfd000
/root/target: Arquivo ou diretório não encontrado
Não é póssivel acessar a memória no endereço 0xbae01000
/root/target: Arquivo ou diretório não encontrado
Não é póssivel acessar a memória no endereço 0xbae02000
/root/target: Arquivo ou diretório não encontrado
Não é póssivel acessar a memória no endereço 0x6f2e8000
hamilton@Linux-VirtualBox:~/Documentos/pos-memreader$
```

allows *ptrace* to access the memory of a process inside the container but does not allow the host machine or other container to access it. Still according to [31] an alternative to enable the monitoring and access to memory information would be to expose such information in the structure of **/sys/fs/cgroup/** in the same way that it is made to **/proc/pid/**.

B. Limitations

The described proposal calls for the cloud resource to be uniquely identifiable in order to make the association between evidence and its origin. During the course of this project this unique form of identification was only possible through the hash of the container image. This was the only feature that, submitted to the build process from the same recipe resulted in the same hash of the image. So, the implementation for validating the proposed solution can only collect memory information in *user space*. In principle, therefore, Dizangdoes not provide support to malware investigation techniques based on information from the kernel space, such as comparing the information from the *Process Environment Block* (PEB), which are in the user space, with information from the *Virtual Address Descriptor* (VAD), which lies in the kernel space. Analyses of threats that directly manipulate the objects of the kernel (D.K.O.M – *Direct Kernel Object Manipulation*) do not benefit from the solution proposed, either.

VI. Final Considerations

Digital threats acting directly on the system memory do not usually leave traces in disk after their corresponding resources are removed, hindering later forensic analyses. This problem is especially notable in cloud computing systems, in which the allocation and removal of virtualized resources (e.g. VMs and containers) are frequent. This characteristic, together with aspects such as multitenancy and multi-jurisdiction of computational clouds, hinders evidence collection for investigating incidents.

In this scenario, the proposal presented aims to relate the memory photograph to its origin, using the calculated hash of

the container image as a stored evidence identifier. To prevent an excessive use of disk space, the volume of data stored uses a storage window, which allows describing the memory before and after an attack (e.g. memory injection). Combined with a tool for identifying threats, these Dizangcharacteristics make it a robust solution to provide evidence and thus make forensic analyses in the cloud viable.

As mentioned before, it was not possible to copy the contents of the process running inside the container. Due to the complexity and knowledge required to perform the suggested modification and the process for accepting it in the linux code base, it is recommended for future work to incorporate into Dizanga way of solving access to the contents of the memory of a container from the host system.

REFERENCES

[1] A. M. Morsy, J. Grundy, and I. Muller, "An Analysis of the Cloud Computing Security Problem," in *APSEC Cloud Workshop*. Sydney, Australia: Cornell University, 2010. [Online]. Available: https://arxiv.org/abs/1609.01107

[2] P. Gilbert and S. Sujeet, *Advances in Digital Forensics IV*, 1st ed. Orlando: Springer-US, 2008, vol. 1.

[3] J. Dykstra and A. Sherman, "Acquiring forensic evidence from infrastructure-as-a-service cloud computing: Exploring and evaluating tools, trust, and techniques," *Digital Investigation*, vol. 9, pp. S90–S98, 2012, (Proc. of the 12th Annual DFRWS Conference). [Online]. Available: dx.doi.org/10.1016/j.diin.2012.05.001

[4] S. Rahman and M. N. A. Khan, "Review of live forensic analysis techniques," *International Journal of Hybrid Information Technology*, vol. 8, no. 2, pp. 379–388, 2015. [Online]. Available: www.sersc.org/journals/IJHIT/

[5] J. Dykstra and A. T. Sherman, "Design and implementation of FROST: Digital forensic tools for the OpenStack cloud computing platform," *Digital Investigation*, vol. 10, pp. S87–S95, 2013, (Proc. of 13th Annual DFRWS Conference). [Online]. Available: dx.doi.org/10.1016/j.diin.2013.06.010

[6] Z. Reichert, K. Richards, and K. Yoshigoe, "Automated forensic data acquisition in the cloud," *IEEE Int. Conf. on Mobile Ad Hoc and Sensor Systems*, pp. 725–730, 2015.

[7] S. George, H. Venter, and F. Thomas, "Digital Forensic Framework for a Cloud Environment," in *IST Africa*. Tanzania: IIMC, 2012, pp. 1–8.

[8] T. Sang, "A log-based approach to make digital forensics easier on cloud computing," *Intelligent System Design and Engineering Applications (ISDEA)*, pp. 91–94, 2013.

[9] A. Case, M. Ligh, L. Jamie, and A. Walters, *The Art of Memory Forensics: Detecting malware and threats in Windows, Linux and Mac memory*. Hoboken, NJ: Wiley, 2014.

[10] S. Vömel and J. Stüttgen, "An evaluation platform for forensic memory acquisition software," *Digit. Investig.*, vol. 10, pp. S30–S40, 2013, elsevier Science Publishers. [Online]. Available: http://dx.doi.org/10.1016/j.diin.2013.06.004

[11] P. Mell and T. Grance, "The NIST definition of cloud computing," NIST SP 800-145, p. 7, 2011. [Online]. Available: csrc.nist.gov/publications/nistpubs/800-145/SP800-145.pdf

[12] Diamanti, "Five reasons you should run containers on bare metal, not vms," Diamanti, Tech. Rep., 2018, available: https://diamanti.com/wp-content/uploads/2018/07/Diamanti{_}WP{_}Five{_}Reasons{_}You{_}Should{_}Run{_}Containers{_}on{_}Bare{_}Metal{_}071918.pdf.

[13] DevOps and ClusterHQ, "Container market adoption survey 2016," https://clusterhq.com/assets/pdfs/state-of-container-usage-june-2016.pdf, 2016.

[14] M. Miller and J. Turkulainen, "Remote library injection," NoLogin, Tech. Rep., 2004, available: http://www.hick.org/code/skape/papers/remote-library-injection.pdf.

[15] H. Yin, Z. Liang, and D. Song, "HookFinder: Identifying and understanding malware hooking behaviors," in *Proceedings of the Network and Distributed System Security Symposium (NDSS'08)*, 2008, pp. 1–16.

[16] S. Fewer, "Reflective DLL injection – v1," Harmony Security, Tech. Rep., Oct 2008.

[17] R. Poisel, E. Malzer, and S. Tjoa, "Evidence and cloud computing: The virtual machine introspection approach," *Journal of Wireless Mobile Networks, Ubiquitous Computing, and Dependable Applications*, vol. 4, no. 1, pp. 135–152, 2013. [Online]. Available: citeseerx.ist.psu.edu/viewdoc/summary?doi=10.1.1.469.937

[18] D. Barbara, "Desafios da perícia forense em um ambiente de computação nas nuvens," Univ. do Planalto Catarinense, Tech. Rep., 2014, revista. uniplac.net/ojs/index.php/tc_si/article/view/1911.

[19] B. Dolan-Gavitt, T. Leek, M. Zhivich, J. Giffin, and W. Lee, "Virtuoso: Narrowing the semantic gap in virtual machine introspection," in *IEEE Symposium on Security and Privacy*. Plymouth, UK: IEEE, May 2011, pp. 297–312.

[20] A. Aljaedi, D. Lindskog, P. Zavarsky, R. Ruhl, and F. Almari, "Comparative analysis of volatile memory forensics: Live response vs. memory imaging," in *IEEE 3rd Int. Conf. on Privacy, Security, Risk and Trust*, 2011, pp. 1253–1258.

[21] F. Dezfouli, A. Dehghantanha, R. Mahmoud, N. Sani, and S. Shamsuddin, "Volatile memory acquisition using backup for forensic investigation," in *Int. Conf. on Cyber Security, Cyber Warfare and Digital Forensic (CyberSec)*. Plymouth, UK: IEEE, June 2012, pp. 186–189.

[22] R. B. van Baar, H. M. A. van Beek, and E. J. van Eijk, "Digital Forensics as a Service: A game changer," *Digital Investigation*, vol. 11, pp. S54–S62, 2014. [Online]. Available: dx.doi.org/10.1016/j.diin.2014.03.007

[23] D. Quick and K. K. R. Choo, "Impacts of increasing volume of digital forensic data: A survey and future research challenges," *Digital Investigation*, vol. 11, no. 4, pp. 273–294, 2014. [Online]. Available: dx.doi.org/10.1016/j.diin.2014.09.002

[24] S. Simou, C. Kalloniatis, E. Kavakli, and S. Gritzalis, "Cloud forensics: Identifying the major issues and challenges," in *Advanced Information Systems Engineering (CAiSE 2014)*, vol. 8484. Cham, CH: Springer International Publishing Switzerland 2014, 2014, pp. 271–284.

[25] S. Alqahtany, N. Clarke, S. Furnell, and C. Reich, "Cloud forensics: A review of challenges, solutions and open problems," in *Int. Conference on Cloud Computing (ICCC)*. Plymouth, UK: IEEE, April 2015, pp. 1–9.

[26] M. Rafique and M. N. A. Khan, "Exploring Static and Live Digital Forensics: Methods, Practices and Tools," *IJSER*, vol. 4, no. 10, pp. 1048–1056, 2013. [Online]. Available: www.ijser.org/researchpaper{\%}5CExploring-Static-and-Live-Digital-Forensic-Methods-Practices-and-Tools.pdf

[27] R. E. Dierks T, "The Transport Layer Security (TLS) Protocol," IETF: https://tools.ietf.org/html/rfc5246, Fremont, CA, 2008.

[28] Unix Man Pages, "Numa Maps - Non Uniform Memory Architecture," man7.org/linux/man-pages/man7/numa.7.html, acessado em: 24-06-2017.

[29] Fabio Kung, "Memory inside LInux containers," https://fabiokung.com/2014/03/13/memory-inside-linux-containers/, acessado em: 30-11-2018.

[30] Docker, "Runtime privilege and Linux capabilities," https://docs.docker.com/engine/reference/run/, acessado em: 30-11-2018.