Dizang : A solution for collecting forensic evidences on 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 factors 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 balance the load among them; this behavior is expected to meet the demand without harming the quality of the service offered. 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 forensic perspective when attacks occur. For example, if a temporary virtual processing instance undergoes an attack that directly affects its memory, without leaving traces in permanent storage (e.g., log files), 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, information relevant to the investigation may be allocated in different countries. In practice, this encumbers the acquisition of such information, especially when there are no cooperation agreements among the entities involved [3]. Combined, these characteristics hinder the implementation of a evidence collection process with the necessary credibility so that they can be accepted in legal processes. In particular, it becomes harder to comply with privacy, jurisdiction and chain of custody requirements, as well as to ensure the reproducibility of the collection process [4].

Although 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 removed. 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 removed.

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 V 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) MARCOS: CITACAO BEM VINDA: algum trabalho que explica VMs. 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 [12], 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. Therefore, containers allow a simpler implementation and better resource usage, eliminating layers between the application being executed and the physical hardware. MARCOS: TODO: seria bom inclur 1 ou 2 frases com benchmarks de conteineres vs. VMs, ou qualquer outra coisa que motive melhor o uso de conteineres (números seriam o ideal)

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 [13]. 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 [14]. 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 [15]. 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 and for a post-mortem analysis of such attacks. On the other hand, balancing the conflicts between this security need and the performance requirements met via rapid elasticity is a challenging task. We discuss some of the proposals in the literature that address this issue in the next section.

III. Related works

MARCOS: TODO: Existem 4 critérios na sua tabela, mas apenas 3 deles são discutidos nas sub-seções: PRECISA DISCUTIR O 4o também!!!

There is a number of aspects that need to be taken into account when performing forensic analysis in the cloud, going from information collection to guaranteeing the evidence's chain of custody. For a structured discussion of the works available in the literature, which are summarized in Table I, we present them considering their approach to each of such aspects.

A. Accessing and collecting memory-related information in the cloud

Different works on forensic analysis in the cloud concentrate on collecting data "after the fact", i.e., after an intrusion has been detected [6], [16], [5], [7], [8]. The collection processes described in those works can be started manually, or automatically by integrating them with an intrusion detection system. In the specific case of volatile memory, that form of collection cannot describe the memory contents before the intrusion, since the process is only activated after the attack has been detected. This limitation can be harmful to investigations, given that some analyses depend exactly on their ability to compare two memory snapshots [9].

Among the works studied, the only proposal found that considers this need is [20], which proposes that the data are stored in the very equipment under analysis. Unfortunately, however, the application of this approach to the cloud scenario is not completely viable. The reason is that it can lead to the loss of important information in case the VM or the container comes to be deactivated and has its resources released.

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

	Continuous collection	Collection process reproducible without VM	Chain of custody guarantees	Jurisdiction and privacy preservation
Dizang (this proposal)	✓	✓	✓	✓
[7]	✘	✘	✘	✓
[16]	✘	✘	✘	✓
[5]	✘	✘	✘	✓
[17]	✘	✘	✘	✓
[6]	✘	✘	✓	✓
[8]	✓	✘	✓	✓
[18]	✘	✘	✘	✓
[19]	✘	✘	✘	✓
[20]	✓	✘	✘	✓
[21]	✓	✘	✓	✓

There are also works aimed at collecting information during the system's execution. In this case, data are constantly collected with no distinction between what happened before or after the event of interest. This is the case of the proposals described in [16], [5], [8], which adopt the strategy of isolating and halting the VM, and only then conducting the collection process. Albeit interesting, the approaches described in these works may lead to a massive volume of data collected. In addition, they do not treat the scenario in which the evidence to be collected refers to data in virtual resources that have been released.

B. Ability to reproduce the process and to obtain the same results

If, during a forensic analysis, different analysts obtain distinct results when executing the same collection procedure, the evidence generated has no credibility and may come not to be accepted 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.

C. Avoiding violating privacy or jurisdiction of parties not involved in the investigation

In a public cloud environment, removing the hardware for later analysis may lead to violating the privacy of users. 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 issue using two major strategies. The first, adopted by [6], [7], [16], [5], involves collecting data pertinent to the investigation and in 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, 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 [22].

IV. PROPOSED SOLUTION: DIZANG

This proposal aims mainly at collecting memory from virtual computational resources so as to succeed in: (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 memory 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 solution here presented, called Dizang, is described in detail as follows.

A. Description

In computational systems executed on a physical infrastructure (i.e., non-virtualized), a direct association can be made between any resource, such as memory information, disk image or packets traveling in the network, and their corresponding origin. In systems built in a virtual infrastructure, especially when it is self-scalable, the computational resources are highly volatile and can thus be removed at any time. MARCOS: Dizer aqui que focamos em conteineres por ser mais fácil rotulá-los de forma única. A frase original era (e precisa ser mudada para melhorar o contexto: In this work, we focus primarily on containers for our implementation of Dizang , although a VM-based implementation of the proposed architecture would also be possible To succeed in correlating a piece of evidence to its volatile origin, this implementation uses Linux containers (LXC) to persist in the source-evidence relationship. Basically, LXC takes advantage of functionalities such as *cgroups* and *namespacing* of the Linux kernel to help managing and isolating virtual resources. Although the resulting container is still a software artifact, and is thus also volatile, each image compiled and its execution in the form of a container are normally linked to a hash that univocally identifies this relationship. The container also allows more

precisely identifying the source of a piece of evidence, once it is possible to divide the parts of a system into containers; for example, one container for the engine of dynamic web pages (e.g. *Apache*), another with business logic (e.g. *Golang*) and a third one for a database (e.g. *Cassandra*).

Memory copy is not an atomic activity, since it is executed jointly with other processes. Hence, in case one of these processes is a malicious code erasing traces of its existence from the container memory, possibly important information may eventually be lost. Aiming to make the memory copy process more atomic, Dizang temporarily interrupts the execution of the container, makes a copy of its memory and then resumes its execution. This technique, similar to that adopted by [23] for VMs, produces a photograph of the container volatile memory; this allows its analysis at a repose state, that is, without the need of having the container at work. When conducting the data collection at adequate time intervals, it is possible to build the history of the memory state during its execution in the container.

Most of the forensic techniques currently most widely employed are directed to obtaining information in its totality, be it via bit by bit copy, be it by obtaining the physical hardware [24] [25]. Even though these techniques may sound interesting at first, many times they are eventually responsible for a problem: the growing volume of information investigators have to analyze [26]. To mitigate this difficulty, Dizang adopts two strategies. The first is defining a volume of data that may be considered enough for conducting an investigation; the second is defining a maximum age for the evidence while the system works under normal conditions, i.e., when it is not under attack. To detect and to analyze intrusions in the processes memory, it is necessary to have a copy of the memory before and after the intrusion [9]. The solution proposed thus implements a window of memory photographs covering a pre-defined time interval, as illustrated in Fig. 1. In normal operation conditions, the pieces of evidence are collected with a certain periodicity and collections reaching a given age are discarded. In contrast, after an attack event is detected, (e.g. by an intrusion detection system), Dizang stops discarding the oldest collections from the monitoring log, enabling to know the system before and after the attack and hence assess its evolution.

To persist in the evidence-origin relationship and to guarantee its integrity, this proposal calculates the pair hash [evidence, container image identifier] and stores the triple [hash, container image identifier, evidence]. To prevent possible problems with the storage of these data in countries with different jurisdictions from those in which they must be applied in the investigation in question, the evidence collected is stored in a physical place outside the cloud, after being transported by a safe channel (e.g. via *Transport Layer Security* – TLS [27]).

B. Implementation

The methods proposed were implemented in a test platform aiming to assess the Dizang efficacy in collecting the memory information from the containers in a reproducible way, without violating jurisdictions or users' privacy. The solution, illustrated in Fig. 2, consisted in creating a t2.micro instance in the Ohio zone of the AWS with 3.3 Mhz, 1Gb RAM and 64-bit operational system. At this AWS instance, Docker Engine 1.10 and API Docker 1.21 were installed, from which 3 containers were created, executing nginx 1.0 at different ports. Using a Java application that discovers the process identifier associated to each container, it was possible to copy the content of the *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 [28]. The copy and recording of the file is such that, every minute, the application (1) pauses the container in question, (2) copies

Figure 1. Sliding window of evidence collection

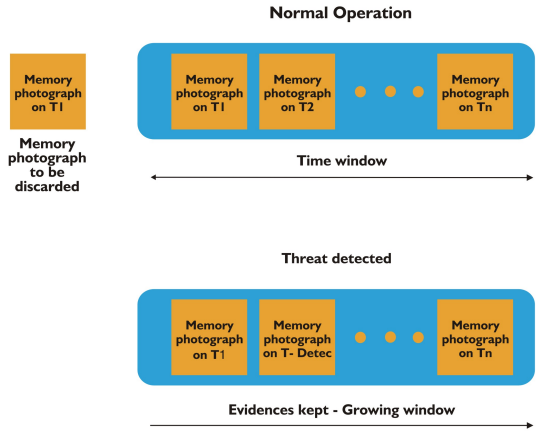
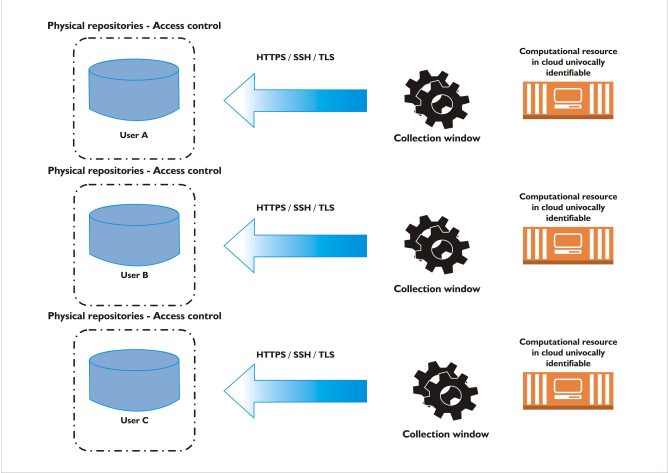


Figure 2. General architecture of the Dizang solution



the **numa_maps** directory, (3) concatenates the data obtained with the container image identification hash, (4) calculates the set hash, and (5) saves the result in a file whose name is the container image identifier and whose extension is **.mem**. The safe transport of the evidence to a physical storage outside the AWS was implemented using a t2.micro instance in the AWS Ohio zone whereby an OpenVPN server had been installed. The EC2 instance containing the evidence was configured to accept connections solely from machines in the VPN. A physical machine outside the AWS used the OpenVPN client to establish a VPN connection with the instance containing the evidence and transported it to the disk of the physical machine. After concluding the transport process, the physical machine verifies whether there are .mem files in disk older than a certain "t" time interval, and discards them.

C. Experimental results

To assess the Dizang efficacy in collecting evidence, two experiments were performed using the environment implemented (described in Section IV-B). For the first experiment, the system was configured for memory collections at 1-minute intervals, to save them in a disk in a physical machine outside the cloud and to erase the samples collected over 5 minutes before. The system was then executed for 30 minutes, a time along which the following were collected as metrics: (1) the use of space in disks used by the memory photographs saved, (2) the time necessary for pausing the container for copying them, and (3) the time taken for transporting the evidence to the physical machine outside the cloud. At each collection, *Unix* command `du -sh *.mem` was executed in the physical storage disk outside the cloud to return the list of the files in which the memory photographs were stored and the disk space they occupied.

The disk space occupied by the memory photographs captured during the experiment is shown in Fig. 3. The graph shows that the increase in use of the disk space is linear and the growth ceases when the time limit configured for the window is reached, once the collections with a lifespan larger than that limit are erased from the disk. The solution thus keeps the disk space occupied by the samples collected under control. Concurrently, memory photographs saved by the solution after the containers and the AWS instance are removed and kept in the disk of the physical machine outside the cloud; they may be associated to their origin, as expected for a forensic analysis. This ability is kept after a threat has been detected since, in this case, older collections stop being erased so it is possible to describe the state of the system before and after the incident [9], allowing, for example, code injection attacks to the memory to be analyzed.

A potential limitation of the proposed solution is that the pause of a container to collect data may, in principle, cause performance losses of the application being executed. To assess this impact, the times for copying the container memory were measured along the experiment. The results are shown in the graph of Fig. 4. Note that, after the application is initialized, the time for making a copy is very small, 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.

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 Dizang collecting 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 Dizang from 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 [29], 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*,

Figure 3. Evolution of disk space usage while running Dizang . TODO: Melhor reduzir o tamanho da figura pra 1 coluna, aumentando a fonte do texto mas reduzindo a distância horizontal. Senão passa a impressão de desperdício de espaço.

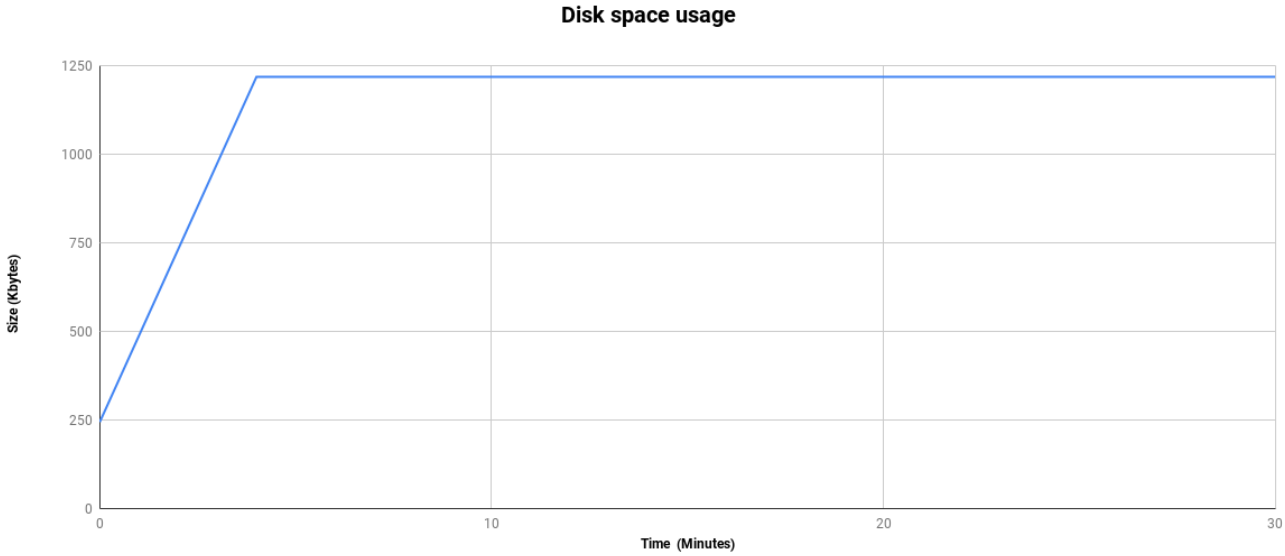
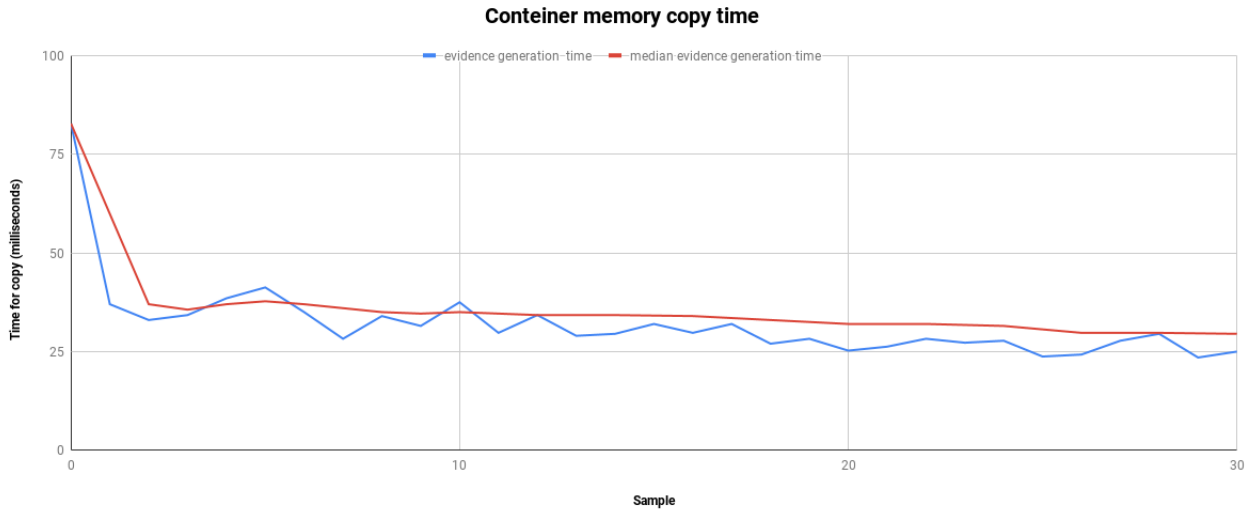


Figure 4. Time for copying the container. MARCOS: Está escrito "conteiner" na figura... deveria ser "container".



the kernel sends a memory access violation signal, the result is shown in Figure 9. The Docker Engine [30] documentation mentions the command `--cap-add=SYS_PTRACE --security-opt-seccomp=unconfined` which 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 [29] 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/**.

D. 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, Dizang does 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

Figure 5. Evidence transport time

Figure 6. Command to generate a raw copy of the process conteiner 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>ò<80><9d>¯›Ñ*Í~^Z<9b>íEkI^Fß, ü0¡ê®½>
```

kernel (D.K.O.M – *Direct Kernel Object Manipulation*) do not benefit from the solution proposed, either.

V. 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

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
ïi1<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 póssível acessar a memória no endereço 0xba9d5000
/root/target: Arquivo ou diretório não encontrado
Não é póssível acessar a memória no endereço 0xba9d7000
/root/target: Arquivo ou diretório não encontrado
Não é póssível acessar a memória no endereço 0xbabdd000
/root/target: Arquivo ou diretório não encontrado
Não é póssível acessar a memória no endereço 0xbadf8000
/root/target: Arquivo ou diretório não encontrado
Não é póssível acessar a memória no endereço 0xbadfd000
/root/target: Arquivo ou diretório não encontrado
Não é póssível acessar a memória no endereço 0xbae01000
/root/target: Arquivo ou diretório não encontrado
Não é póssível acessar a memória no endereço 0xbae02000
/root/target: Arquivo ou diretório não encontrado
Não é póssível acessar a memória no endereço 0x6f2e8000
hamilton@Linux-VirtualBox:~/Documentos/pos-memreader$
```

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

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

[14] 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.

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

[16] 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

[17] 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.

[18] 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.

[19] 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.

[20] 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.

[21] 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

[22] 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.

[23] 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

[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] D. Bem, F. Feld, E. Huebner, and O. Bem, "Computer forensics - past , present and future," *Journal of Information Science and Technology*, vol. 5, no. 3, pp. 43–59, 2008.

[26] 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

[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.