RESEARCH

Dizang: A solution for collecting forensic evidences in cloud environments

Hamilton J S Fonte II* and Marcos A. Simplicio Jr.

*Correspondence:
hamiltonii@gmail.com
Escola Politécnica, Universidade de São Paulo (USP), Av. Prof. Luciano Gualberto, 380, 05508-010 São Paulo, SP, BR
Full list of author information is available at the end of the article

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.

Keywords: Cloud computing; containers; digital forensics; cloud forensics; evidence acquisition; chain of custody

Introduction

Virtualization techniques, replication of services and resource sharing among multiple users (multitenancy) are key enablers for the high scalability of computational clouds [**?**]. 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 [**?**]. 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 investiga-

tion 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 [**?**]. 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 [**?**].

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, [**?**] and [**?**] deal with factors such as multitenancy and multi-jurisdiction, discussing forms of collecting and preserving evidence outside the cloud. In comparison, [**?**] concentrates on forensic analysis to collect evidence from VMs while they are being executed, whereas [**?**] 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 [**?**], which would normally leave no traces when cloud-based virtual instances are deactivated and their memory is released [**?**, **?**].

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

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 [**?**]. Depending on which resources are provided to clients, three main cloud service models can be defined [**?**]: 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) [**?**]. 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 [**?**], 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 [**?**], 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 [**?**]. Examples include:

- *Remote library injection*: A malicious process forces the target process to load a library into its memory space [**?**]. 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 [**?**]. 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 [**?**]. 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.

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 1).

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) machine'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 [**?**]. 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 [**?**]. 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 [**?**], [**?**] and [**?**]. Unfortunately, however, they all have limitations when considering the particularities of the cloud scenario. Namely, [**?**] focuses on mobile environments, and has the disadvantage of not storing the history of memory changes; hence, it would not cope with the high volatility of the cloud. Conversely, the cloud-oriented work from [**?**] 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, [**?**] relies on indexing, pre-processing 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.

Chain of custody guarantees

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 [**?**]. 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 [**?**] and [**?**] address the issue of chain-of-custody, although not completely. Specifically, [**?**] 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, [**?**] 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 1 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.

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.

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 [**?**, **?**, **?**, **?**], involves collecting data pertinent to the investigation and storing them outside the cloud. The second, employed in [**?**] and consisting in a specific case of [**?**], 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 [**?**].

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.

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.

Memory copying

Memory copying is usually not an atomic activity. Instead, it is executed jointly with processes that may modify 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 and inspired on the process of cloud checkpointing [**?**], Dizang temporarily interrupts the execution of the container, makes a copies of its contents, and then resumes its execution. This technique is similar to that adopted by [**?**] for VMs, producing a photograph of the container's volatile memory for latter analysis.

As long as such data collection process is repeated periodically, in short time intervals, it is possible to build a reasonably 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., when it is not under attack. Memory copies reaching a given age are periodically discarded, opening space for new photographs. After an attack event is detected (e.g., by an intrusion detection system), though, Dizang stops discarding older pieces of memory from the monitoring log. This approach, illustrated in Fig. 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.

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., *Transport Layer Security* – TLS [**?**]).

Experimental analysis

The Dizang architecture was implemented in a testing platform, aiming to assess its efficacy and efficiency. The test bed 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. We developed a Java application, which is executed on the host OS to: (1) discover the process identifier associated to each container, and then (2) access the container's *Non-Uniform Memory Allocation descriptor* (**/proc/pid/numa_maps**). The latter contains the allocation of the memory pages, the nodes associated to these pages and what was allocated in their respective access policies [**?**]. The copy and recording of the file is such that, in every time interval t, 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 are older than a certain time interval, and discards them.

Using this test bed, two sets of experiments were performed: the first set 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 sub-sections.

Performance Analysis

For the first experiment, the system was configured to collect memory photographs in $t = 1$ minute intervals and send them to the physical machine outside the cloud. The data collection time window was set to $w = 5$ minutes, so samples that are older than w are erased from the physical machine's disk. The system was then executed for 30 minutes, and the following metrics were collected: (1) the disk 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.

Memory

In our test bed, each memory photograph takes 244-kbytes of storage. Fig. 3 shows the total disk space occupied during the experiment for all photographs, as obtained via the *Unix* command `du -sh *.mem`. 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. 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.

Latency overhead

A potential issue of the proposed solution is that pausing a container to collect data may, at least in principle, degrade the efficiency of the application being executed. To assess this overhead, the times for copying the container memory were measured along the experiment. The results are depicted in Fig. 4. This figure shows that 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 is unlikely to be perceptible by end users. Nevertheless, an even lower latency can be obtained if the collection procedure for each instance is performed in different moments of time. For example, instead of simultaneously suspending the execution of all computational resources to carry out the data collection task, one could interrupts them sequentially, in round-robin fashion. Therefore, the latency hereby shown can be considered a worst case scenario for our test bed.

Transfer process

If the time required for transporting the evidence to the storage server outside the cloud is longer than its generation interval, the system is prone to create a large backlog and, hence, evidences may be lost. To evaluate this factor, the evidence transport time was measured along the experiment, as illustrated in Fig. 5. The result obtained is that the time for transporting evidence from the AWS server in North America to our test server in South America is around 30 seconds on average. Obviously, the network itself plays an important role in those numbers, so servers located in closer locations should lead to smaller transfer times.

Identifying malware injection

Our second experiment aims to determine if it is possible, through the analysis of the collected evidence, to identify malware injection in the container memory. To this end, a library named **libexample.so** simulating a library-based malware was injected into one of the containers. After five minutes of evidence collection with Dizang, the aforementioned library was injected into the memory of one of the containers. Subsequently to the injection, Dizang was allowed to continue collecting memory snapshots for another 5 minutes. In addition to collecting the **/proc/pid/numa_maps** directory contents, a raw copy of the container's process memory was also acquired using the Linux *GDB* utility and the *nsenter* tool, via the commands shown in Fig. 6. Figs. 7 and 8 show part of the container's memory before and after the injection, respectively. As expected, it is possible to see the injected **libexample.so** library between addresses **7f85631b8000** and **7f85633b9000** in the snapshot after the injection. The library contents can also be identified when we analyze the raw copy of the container's process memory from the snapshots stored outs

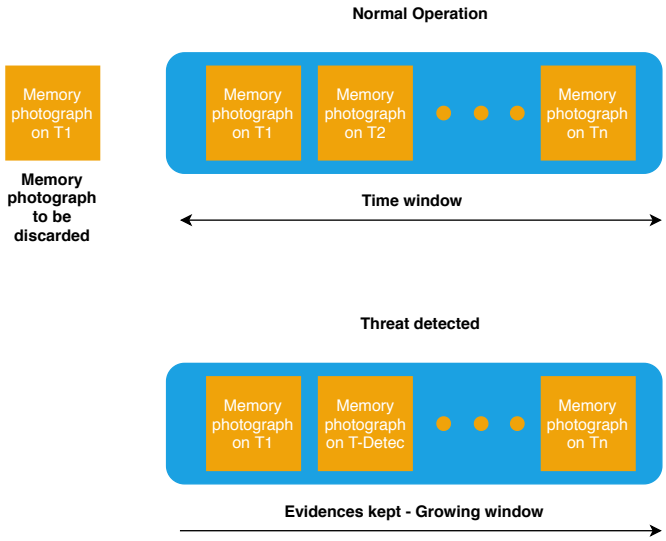
Conclusions and future work

Digital threats acting directly on the system memory may not leave any trace 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.

To address this scenario, we hereby propose Dizang, a solution for evidence collection in the cloud. Dizangallows the acquired memory photographs to be correlated to its origin, using the calculated hash of the container image as an identifier. In addition, the solutions employs a time window to prevent excessive disk usage for such photographs, while still allowing the memory before and after an attack to be analyzed. Our experimental analysis indicate that, when combined with intrusion detection mechanisms for identifying threats, Dizang can be used as a powerful incident management tool, improving evidence collection and forensic analysis in the cloud.

Finally, we note that the proposed architecture calls for the cloud resource to be uniquely identifiable, so collected evidences can be correctly associated to their origin. When developing Dizang, we determined that the easiest way to obtain this unique form of identification was via the hash of the container image. Indeed, and unlike what usually happens with VMs, containers built from the same recipe in the Docker Engine lead to images with the same hash. Nevertheless, and even though the implementation obtained is useful validating the viability of the proposed solution, it can only collect memory information in *user space*. Therefore, in principle Dizang does not provide support to malware investigation techniques based on information from the *kernel space*. For example, it would not support analysis that rely on comparing (1) information acquired from the *Process Environment Block* (PEB), which are in the user space, and (2) information from the *Virtual Address Descriptor*

Figure 1 Sliding window of evidence collection

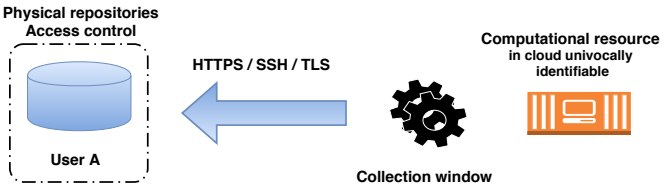


(VAD), which lies in the *kernel space*. Analyses of threats that directly manipulate the objects of the kernel (D.K.O.M – *Direct Kernel Object Manipulation*) do not benefit from the solution proposed, either. An open problem that remains is, thus, how to extend Dizang's functionalities so it can gather evidence from the *kernel space* associated with a container.

Competing interests
The authors declare that they have no competing interests.

Figures

Figure 2 Dizang's inner working for a single container.



Tables

Figure 3 Evolution of disk space usage while running Dizang.

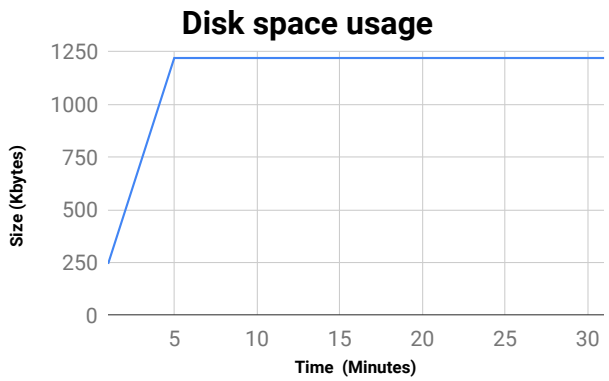


Figure 4 Average time for saving a container's snapshot.

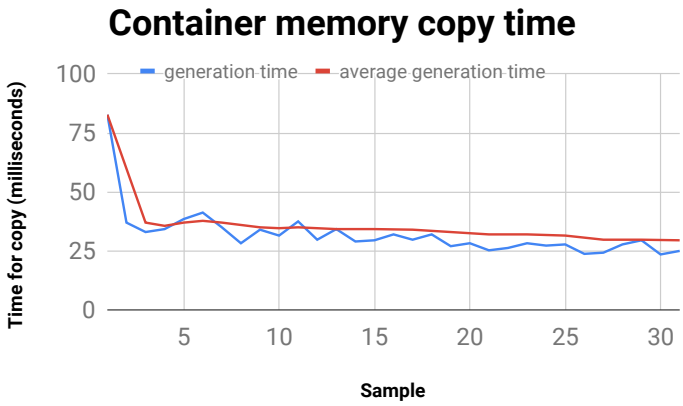


Figure 5 Time required for transporting evidences from the cloud to an external storage site.

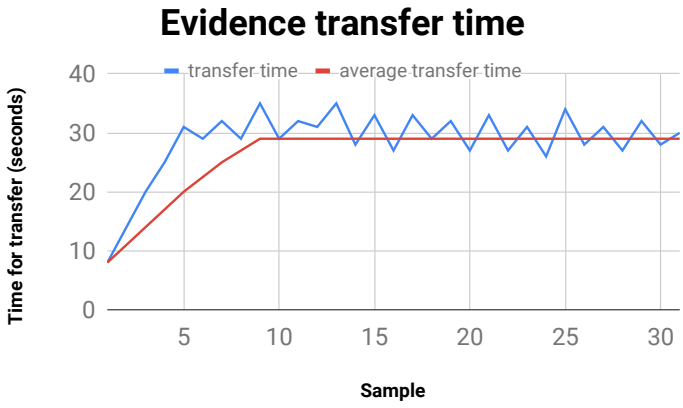


Figure 6 Commands to generate a raw copy of a container's memory.

```
sudo ./nsenter --target <pid> --mount --uts --ipc --net --pid

grep rw-p /proc/<pid>/maps | sed -n 's/^\([0-9a-f]*\)-\([0-9a-f]*\) .*$/\1 \2/p' |
while read start stop; do gdb --batch --pid <pid> -ex "x/4096wx 0x$start";
done > <pid>.dump
```

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

```
00400000 default file=/home/someuser/target mapped=1 N0=1
00600000 default file=/home/someuser/target anon=1 dirty=1 N0=1
00601000 default file=/home/someuser/target anon=1 dirty=1 N0=1
7fee0b0ed000 default file=/lib/x86_64-linux-gnu/libc-2.19.so mapp...
7fee0b2ab000 default file=/lib/x86_64-linux-gnu/libc-2.19.so
7fee0b4ab000 default file=/lib/x86_64-linux-gnu/libc-2.19.so anon...
7fee0b4af000 default file=/lib/x86_64-linux-gnu/libc-2.19.so anon...
7fee0b4b1000 default anon=3 dirty=3 N0=3
7fee0b4b6000 default file=/lib/x86_64-linux-gnu/ld-2.19.so mapp...
7fee0b6d1000 default anon=3 dirty=3 N0=3
7fee0b6d6000 default anon=2 dirty=2 N0=2
7fee0b6d8000 default file=/lib/x86_64-linux-gnu/ld-2.19.so anon...
7fee0b6d9000 default file=/lib/x86_64-linux-gnu/ld-2.19.so anon...
7fee0b6da000 default anon=1 dirty=1 N0=1
7ffc91a60000 default stack anon=2 dirty=2 N0=2
7ffc91b15000 default
7ffc91b17000 default
```

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

```
00400000 default file=/home/someuser/target map...
00600000 default file=/home/someuser/target ano...
00601000 default file=/home/someuser/target ano...
7f8562def000 default file=/lib/x86_64-linux-gnu/lib...
7f8562fad000 default file=/lib/x86_64-linux-gnu/lib...
7f85631ad000 default file=/lib/x86_64-linux-gnu/lib...
7f85631b1000 default file=/lib/x86_64-linux-gnu/lib....
7f85631b3000 default anon=3 dirty=3 N0=3
7f85631b8000 default file=/home/someuser/libexample.so ...
7f85631b9000 default file=/home/someuser/libexample.so
7f85633b8000 default file=/home/someuser/libexample.so ...
7f85633b9000 default file=/home/someuser/libexample.so ...
7f85633ba000 default file=/lib/x86_64-linux-gnu/ld-2.19.so...
7f85635d4000 default anon=3 dirty=3 N0=3
7f85635d9000 default anon=3 dirty=3 N0=3
7f85635dc000 default file=/lib/x86_64-linux-gnu/ld-2.19.so...
7f85635dd000 default file=/lib/x86_64-linux-gnu/ld-2.19.so...
7f85635de000 default anon=1 dirty=1 N0=1
7fff158b7000 default stack anon=4 dirty=4 N0=4
7fff159b2000 default
7fff159b4000 default
```

Figure 9 Sample of the library's memory inside the container

```
0x7f85633b9000: 0x00200e08 0x00000000 0x635db000 0x00007f85
0x7f85633b9010: 0x633d0570 0x00007f85 0x62e11760 0x00007f85
0x7f85633b9020: 0x62ede3b0 0x00007f85 0x631b8616 0x00007f85
0x7f85633b9030: 0x631b8626 0x00007f85 0x633b9038 0x00007f85
0x7f85633b9040: 0x00000000 0x00000000 0x75746e75 0x382e3420
0x7f85633b9050: 0x322d342e 0x6e756275 0x7e317574 0x302e3431
0x7f85633b9060: 0x29342e34 0x382e3420 0x0000342e 0x6d79732e
0x7f85633b9070: 0x00626174 0x7274732e 0x00626174 0x7368732e
0x7f85633b9080: 0x61747274 0x6e2e0062 0x2e65746f 0x2e756e67
0x7f85633b9090: 0x6c697562 0x64692d64 0x6e672e00 0x61682e75
0x7f85633b90a0: 0x2e006873 0x736e7964 0x2e006d79 0x736e7964
0x7f85633b90b0: 0x2e007274 0x2e756e67 0x73726576 0x006e6f69
0x7f85633b90c0: 0x756e672e 0x7265762e 0x6e6f6973 0x2e00725f
0x7f85633b90d0: 0x616c6572 0x6e79642e 0x65722e00 0x702e616c
0x7f85633b90e0: 0x2e00746c 0x74696e69 0x65742e00 0x2e007478
0x7f85633b90f0: 0x696e6966 0x6f722e00 0x61746164 0x68652e00
0x7f85633b9100: 0x6172665f 0x685f656d 0x2e007264 0x665f6865
0x7f85633b9110: 0x656d6172 0x6e692e00 0x615f7469 0x79617272
0x7f85633b9120: 0x69662e00 0x615f696e 0x79617272 0x636a2e00
0x7f85633b9130: 0x642e0072 0x6d616e79 0x2e006369 0x00746f67
0x7f85633b9140: 0x746f672e 0x746c702e 0x61642e00 0x2e006174
```

Table 1 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)	✓	✓	✓	✓
[?]	✘	✘	✘	✓
[?]	✘	✘	✘	✓
[?]	✘	✘	✘	✓
[?]	✘	✓	✘	✓
[?]	✓	✓	✘	✓
[?]	✘	✘	✘	✓
[?]	✘	✘	✘	✓
[?]	✓	✘	✘	✓
[?]	✓	✓	✘	✓