Digital Forensics in Cloud Computing

Alecsandru PĂTRAŞCU, Victor-Valeriu PATRICIU
Military Technical Academy, Computer Science Department, Bucharest, 050141, Romania
alecsandru.patrascu@mta.ro, vip@mta.ro

Abstract—**Cloud Computing is a rather new technology which has the goal of efficiently usage of datacenter resources and offers them to the users on a pay per use model. In this equation we need to know exactly where and how a piece of information is stored or processed. In today's cloud deployments this task is becoming more and more a necessity and a must because we need a way to monitor user activity, and furthermore, in case of legal actions, we must be able to present digital evidence in a way in which it is accepted. In this paper we are going to present a modular and distributed architecture that can be used to implement a cloud digital forensics framework on top of new or existing datacenters.**

Index Terms—**application virtualization, cloud computing, cyberspace, forensics, multicore processing.**

I. Introduction

Cloud Computing represents a computing paradigm that involves multiple researches from various IT fields, such as networking, parallel processing, cryptography and many more. It is most interesting the fact that, although technically we find ourselves with various public services or software that we can use in our private environments, the field of Cloud Computing digital forensics is at its beginning.

In this context, cloud computing has become in the last years a paradigm that attracts more and more researchers. One of the main research areas in this field is the way in which common data and processing power can be shared and distributed across single or multiple datacenters that are spread across a specific geographical area or even the entire globe. A new need for IT experts is increasing: the need to know exactly how, where and in what condition is the data from the cloud stored, processed and delivered to the clients. We can say with great confidence that cloud computing forensics has become more and more a need in todays distributed digital world.

In case of classic computer forensics, the purpose is to search, preserve and analyze information on computer systems to find potential evidence for a trial. In cloud environments the entire paradigm changes because we don't have access to a physical computer, and even if we have access it is a great chance that the data stored on it is encrypted or split across multiple other computer systems.

Taking in account all the variables that have appeared in cloud computing technologies, modern hypervisors and virtualization technologies implement more or less simple mechanisms for data monitoring across datacenters. Starting from the basic building blocks composed by simple logs that are gathered from the entire cloud infrastructure, every monitoring module must have a precise target and must not affect the proper function of the systems from the datacenter. All virtualization technologies have a difficulty in this area.

In this paper we are going to present a new and novel way in which we can integrate a full forensics framework on top of a new or existing cloud infrastructure. We will talk about the architecture that stands at it ground and we will present its advantages for the entire cloud computing community. We will present also the impact that our technology proposal will have on existing cloud infrastructures and as a proof of concept we will present some particular implementation details over our own cloud computing framework that we have already implemented in [20].

We will present the basic architecture of a Cloud Computing enabled framework that is natively scalable across multiple work nodes and we will present the modifications that must be made to such a framework in order to monitor the entire user and cloud modules activity.

Our work is structured as follows. In section II we are going to talk about related work in the field of cloud computing and digital forensics. We also present the general context in which cloud forensics must be developed and in section III we will present in detail the basic architecture and the forensic enabled variant. In section IV we present some performance needs that must be resolved when implementing such a system and section V contains results gathered from the implementation of our framework. Finally in section VI we conclude our document.

II. Related Work

The area of cloud computing research is an active area and in it we can find a lot of work regarding the problem of efficiently organizing hardware in order to make it possible to run multiple virtual machines. In the field of classic incident response there are is a lot of active research and many books, guides and papers. Nevertheless, in the field of cloud computing incident response the papers are mostly theoretical and present only an ideal model for it.

In [1] Tayal talks about the perspective of Cloud Computing scheduling techniques. He considers that task based scheduling can be used with success in such distributed environments because they improve the flexibility and reliability of the entire structure. More exactly the tasks are split evenly across processing nodes, taking in consideration the hardware platform that is specific to each of them.

In [2] Buyya et al talk about the mixture of High Performance Computing (HPC) applications and Cloud Computing applications in modern software. This can help the development of new technologies and services offered to the end-users by enabling geographical aware deployments. Nevertheless, as the authors state, we must pay great attention to the processing power needed for such infrastructures and we must use a proper scheduler in order

Digital Object Identifier 10.4316/AECE.2014.02017

to balance computing across the nodes from the datacenter.

Also, Brandic et al [3] talk about what Cloud Computing means for end users and they present a model for a Cloud enabled architecture by using the power of Virtual Machines (VMs). They also present different scheduling strategies and present the need for a Service Level Agreement (SLA) for the offered services.

In the direction of classic incident response, one of the most interesting guides is the one from NIST [4]. In it we can find a summary containing a short description and recommendations for the field of computer forensics, along with the basic steps that must be made when conducting a forensic analysis: collection, examination, analysis and reporting. A great deal of attention is paid to the problem on incident response and how should an incident be detected, isolated and analyzed.

Bernd Grobauer and Thomas Schreck talk in their paper [5] about the challenges imposed by cloud computing incident handling and response. This problem is also analyzed by Chen [6] and in their paper they consider that incident handling should be considered a well-defined part of the security process. Also it is presented a description for current processes and methods used in incident handling and what changes can be made when migrating towards a cloud environment from the point of view of a customer or an security manager.

Furthermore, the integration of cloud incident handling and cyber security is presented in two papers, one written by Takahashi et al [7] and the other written by Simmons et al [8]. They talk about how Internet development lead to a widespread deployment of various IT technologies and security advances. In their paper they also propose an ontological approach for integration of cyber security in the context of cloud computing and present how information should be used and analyzed in such environments.

Cloud Computing and digital forensics is a rather new concept as it implies the usage of classic digital forensics rules on top of the new distributed approach adopted by cloud.Since twenty years ago, experts working in incident response field were taught that it is better to shutdown down an entire system in case of a breach. Today, with cloud computing surrounding us, this task is impossible and we need new approaches. For example, in a virtual machine memory can be stored important data for an investigator, such as network connections and running processes, which in case of a power failure will be lost.

Given these realities, we need to handle digital forensics (incident response and reporting) with great care. Incident responders and forensic investigators must now rely more and more on live collection and analysis of system RAM. Live response analysis has been used for the last six to eight years only in important high-end cases, under the command of highly qualified computer forensics specialists, but modern reality forces this approach to almost all data cases.

By live response analysis we understand the process in which digital data is collected from a running workstation, physical or virtual, *while* it is running. The process is even more interesting in virtualized environments as the notion of RAM is abstracted away by the hypervisor. Nevertheless there are tools such as Volatility [21] that are specialized to this task and can help investigators gather information from virtual machines. Representative examples in this direction are the usages of this tool in malware analysis - malware can be installed in a contained virtualized environment, analyzed and the memory dumps collected.

Current tools that are used in forensic analysis work in a simple way: you upload a remote agent to the target system, wait for and gather snapshots of data from memory and send it back to the investigators computers, where a human or a software program reviews them.

Furthermore, forensic investigators need to avoid copying the entire disk images. In today's cloud environments data can easily reach large sizes and it is not always technically possible or cost effective to do it. Even more, judges, agents and investigators were classically taught that only a perfect data copy is a “forensically sound” copy that can be used. This must change because there is no need to copy an entire disk image in order to retrieve only a few files representing evidence and to accomplish this the forensic investigator will need also a way to view and analyze “live remote” data.

We must be careful how we deal with incident response in the cloud because we must make sure how we will respond to it. Traditional incident response is easy to do because everything is located inside the same network or datacenter. It is an environment in which you can reach the physical devices and you can take forensic copies of them. Traditional incident response consists from seven basic steps and procedures that we are going to present briefly.

The first step is detection [9], in which we identify the threat(s). After we detected a certain threat, we must contain it so it cannot spread to the rest of the systems and eradicate it. To prepare ourselves for future attacks we must remediate the problem(s) and fix the vulnerabilities in order to close all gaps that can breach the system. The recovery is important because we must restore the system from backups to the point before the threat emerged [10]. Finally, after everything is back to normal, we must review the system and check it again and communicate what has been done to the administrators, key stockholders, media, etc.

The cloud model is different because we cannot “grab” the actual data because we cannot access the physical disks [11]. And if we look at the standard definition of cloud computing we can see that the data can be stored even across multiple datacenters. The problem in this case, and one of the main challenges in cloud, is how we can take a forensic image of a cloud instance. Containing the cloud environment is also a problem because we make sure that a potential threat does not spread to other cloud instances.

The only solution for these problems is to change our mindset and try seeing the big picture differently than before [12]. Of course we must still use the same basic principles existing in traditional incident response. The key is not to focus on the datacenter, but to focus on the actual data. We must move along datacenters and networks and pay attention to where is the data. An approach like the one used in airports to keep the airplanes secure can be used - starting from the first time a traveler enters an airport, up to getting in the plane, reaching the destination and safely getting down from the plane.

A step towards this is made in today's cloud computing environments, as we encrypt the data stored online [13-15]. This means that if something happens at the Cloud Service

Provider (CSP) location, the user data is kept safely, because only the user has the decryption key. This is a great responsibility for the user because he is now the manager of his own keys. The important fact is not to permit the CSP to handle the user key management because this represents a "backdoor" in the user's data.

Cloud computing incident reporting represents a way to monitor the data and its location across datacenters and present the report in a human readable form. Since in the following years almost 80% of organizations will store their data on various CSPs, they will be dependent of this technology. We can say for sure that large cloud providers will be serving tens of millions of end-users and it is also fair to say that a part of the public cloud computing services can be included as a part of the critical information infrastructure.

Our paper presents a solution to the problems mentioned in the previous sections. Since we gather data starting from the hypervisors, we can know for sure how and where the data is stored and processed. The main direction for our research is to concentrate on the user activity and his rented virtual machines. By user activity we understand the entire actions one can make while running a virtual machine, such as various network services or tools installation and configuration (HTTP server, FTP server etc), network and Internet communications, console command history and system logs. It is relevant in case of a forensic investigation to acquire all this activity as it can offer to the investigator a more complete view over the targeted system.

We focus on the need to collect and aggregate information from various sources existing in datacenters, starting from the hypervisors and ending with the networking equipments. As in the case of virtual machines, we need logs from all the datacenter modules because: a) this is the main way in which we can detect suspicious activity; b) in case of a security incident they are the only information that have legal meaning and c) we need this information every time a security audit is done over the datacenter.

In order to accomplish this we need a software infrastructure that is going to help us reach our goal.

In the rest of the paper we will talk about a cloud computing infrastructure that we have already implemented and deployed in our own private datacenter [14] and we will present the top-view modifications that we must do to it in order to support cloud computing digital forensics. We also emphasize the fact that the presented modification can be also applied to any other existing cloud infrastructures [11].

III. Cloud Computing Forensics Framework

In order to have a solid working platform, we must first introduce the concept of a software cloud computing framework.

As can be seen in Fig. 1 the top view of a cloud computing framework contains two main layers: the virtualization layer and the management layer.

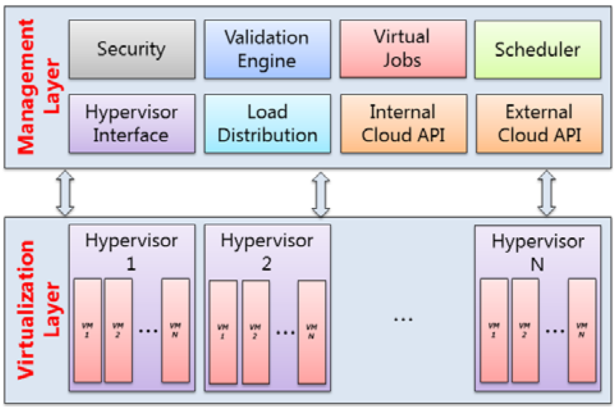

Figure 1. Basic cloud computing architecture

In the virtualization layer we find the actual workstations that host the virtual machines and have virtualization enabled hardware.

In the management layer we find the modules responsible with enabling all the operations specific to a cloud computing system. This layer has a modular architecture and it is easy to see that the whole ecosystem is actually pluggable and it can be extended with other modules or plugins. In Fig. 1 we can see the main components that must exist in this layer and we present each of them briefly. These modules are:

Security. This module is responsible with all security concerns related to the cloud system. For simplicity we can consider it as an intrusion detection and alarming module.

Validation engine. This module receives requests to add new jobs to be processed. It is responsible for checking that the actions that the user specified or requested are actually eligible for executing. The main goal is security, mainly anomaly request detection. Every new request is checked for consistency and it is validated and if it is legit, the new lease is transformed in a job for our system and it is properly inserted in a job queue.

Virtual jobs. This module creates an abstraction between the data requested by the user and the payload that must be delivered to the cloud system.

Scheduler. This is one of the most important modules in a cloud framework. It interacts with the cloud systems that it manages. It does a load balancing of the requests it receives, both in the same autonomous system or inter autonomous systems and it runs a lease based scheduler. Its main purpose is to efficiently schedule the jobs to the virtualization layer. It also must communicate with the other modules in order to find new instances, new services, virtual machine managers, load balancers in the system. For further details, the entire scheduler architecture is presented in detail in our previous work [20].

Hypervisor interface. This module acts like a translation layer that is specific to a virtualization software vendor. It must implement each vendor API specifications.

Load distribution. This module is responsible with both vertical and horizontal scaling of scheduler requests. It must run a distinct application framework in order to decouple the code from the existing underneath runtime. The algorithm must be applied automatically and in the process of this analysis, the number of workstations must be taken in account.

Internal cloud API. This module is intended as a link between the virtualization layer and the cloud system. In order to be more scalable and also maintain a high degree of abstraction, a common interface must be provided and every implementation of the specific API must implement this.

External cloud API. This module offers a way to the user to interact with the system. It must provide means to add new jobs in the cloud system. The requests are registered and sent to the validation engine module. This API must be flexible enough to permit adding details to the jobs, like the hardware specifications of the virtual machine, operating system to be used, packages to be installed.

Besides the properties presented in the previous sections we must introduce a new concept that stands at the ground of cloud computing: cloud native applications [16]. A cloud native application is designed and created to make use of specific engineering practices that have proven successful in some of the world's largest and most successful software applications. Many of these practices are unconventional, yet the need for unprecedented scalability and efficiency drove to the adoption in the environments that truly needed them.

For short, we need certain patterns in order to develop such applications. The most used patterns for cloud and their short descriptions are: scalability (creating applications that make full use of the single or multiple computing systems existent), queue-centric workflow (loose coupling the cloud application modules and focusing on asynchronous delivery of commands and data for processing), map-reduce computing paradigm, busy signal (determining when an application or a module must issue a "busy" signal and how can the other applications or modules respond to it), node failure (focusing on how an application should respond when the compute node on which it is running shuts down or fails) and valet key (focusing on efficiently using storage with untrusted clients).

Now that the notion of a cloud computing framework was presented, we talk about the modifications that must be made to it in order to create a forensic enabled cloud computing architecture. As can be seen in Fig. 2 the modification affects all the existing modules.

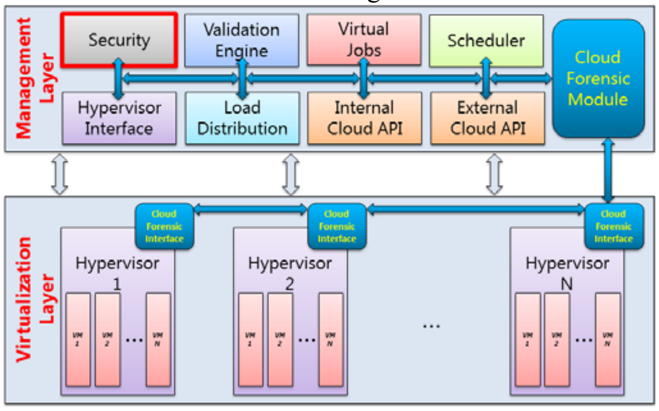

Figure 2. Forensic enabled cloud computing architecture

More exactly, we see a new module, the *Cloud Forensic Module*. Its main goal is to gather all forensic and log data from the virtual machines that are running inside the virtualization layer. Furthermore, we must attribute to the security module greater responsibilities and permit it communicate with all the other modules in the management layer.

Of course, in order to gather data reliably from the virtual machines we must interact with the hypervisors existing in the workstations kernel. In our paper we present only what modifications must be made to a Linux kernel. We have chosen this alternative because in a Linux kernel we can find at least two distinct, free and open source virtualization techniques: KVM and XEN. An image of a forensic enabled kernel can be seen in Fig. 3.

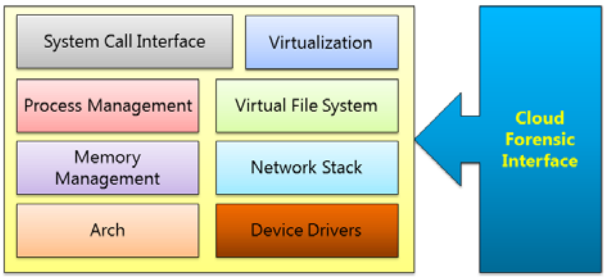

Figure 3. Forensic enabled kernel

On our research we will focus more on the KVM technology. KVM (Kernel-based Virtual Machine) represents a full virtualization software that can be found mainline distributions of Linux kernels running on hardware with Intel or AMD processors. It is implemented as a kernel module entitled *kvm.ko* and the main functionality is given by vendor specific processor modules - *kvm-intel.ko* or *kvm-amd.ko*. Using KVM a hardware platform can be used as a hypervisor and can run various virtual machines, each having its own virtual private disk, processor or network interface.

A generic C*loud Forensic Interface* must be implemented as a series of stand-alone kernel modules and user-space applications that can be activated or disabled at runtime. Our goal is to provide the users a way to manage it from the kernel building menu. We need this segregation because we want to have access to all the modules from the kernel that help in the entire process of virtualization. Parts like system calls, process management, virtual file system, memory management, networking management are extremely important to our research because they represent the basic building blocks between a virtual machine running on a host and the operating system. We will detail furthermore the main interest directions for our research.

The first step towards full kernel integration is to have a proper API both to the kernel and to the external system. We are interested mostly about KVM internal API. This API is a set of *ioctl* instructions that are issued in order to control different aspects of a running virtual machine. In computing, *ioctl* (short for "input-output control") represents a system call made to a specific device which cannot be done using regular system calls. Logically, this API is split across three different main parts: main system ioctls, meant to set proper KVM internal variables and used when creating a new virtual machine, virtual machine ioctls, meant to set different attributes of a virtual machine, like the memory size or layout and virtual CPU ioctls, meant to be used to control the operation of a virtual machine virtual CPU.

In order to be fully compatible with the Linux API, the entire KVM API is centered on the concept of file descriptors. This means that once activated, KVM creates a new device called "*/dev/kvm*". On initial open of the device we get a handle of the internal KVM system that we can use to issue proper ioctl commands. For example, sending a KVM_CREATE_VM command to the kernel, we get a response containing a virtual machine file descriptor that we can further use to set different values.

It is easy to see that we can get all details concerning about the status of the "virtual hardware" that is used for a certain virtual machine. This is useful because, for example, we can get details such as: the memory pages that are dirtied since a last call, getting processor registry values, setting processor registry values, and translation of a memory virtual address according to virtual CPU own translation mode.

From the point of view of live cloud computing forensics, a great impact is given by the Memory Management Unit (MMU). For a virtualization software it is very important to have a proper MMU module. In our case, KVM uses its own MMU modules with the purpose of translation from guest physical address to host physical address. This gives us a real advantage because interacting with these modules gives us a full map of what is going on inside a virtual machine memory space.

In order to be considered reliable, a virtual machine MMU must respect a set of particular requirements, like correctness (the virtual machine must not be able to determine that it is running using an emulated MMU) and security (the virtual machine must not allocate memory beyond the limits imposed by the MMU). These requirements are going to be monitored by our forensic module and at any time we will have a full memory footprint and the whole previous states.

It is important for a forensic investigator to have access to the network communication devices and to the storage devices. In case of virtual machines running under KVM it is done using the *virtio* interface. Virtio represents a virtualization aiding software that offers a standard driver to network or disk devices to the virtual environments that know that they are running in this mode. This is benefic because the virtual environment can cooperate with the driver in order to achieve maximum performance. Its design allows the hypervisor to export a common set of emulated devices and make them available through a common API. Using this interface we gain full access to everything related to the disk and/or network devices than an investigator can use.

To be able to collect data reliably from the hypervisor, in our framework we have implemented a kernel module that interacts with KVM and makes use of the previous presented data structures. The main utility is to collect information, besides the one that are given by default, that can help a forensic investigator when conducting an investigation. We have implemented hooks to the KVM API, which intercept all the calls and dump to the user this information, such as the virtual CPU registers and virtual RAM content.

Together with the custom kernel module running with the hypervisor we also implemented a light kernel module that is automatically installed in every virtual machine that the user creates. The combination of these two modules represents a good way to acquire detailed and real virtual machine activity

The light module has double purpose: 1) it helps our cloud framework to perform better as it can collect from the running virtual machine real-time information such as processor, memory load and read system file logs; and 2) it can intercept and forward the entire or partial network data flow directly to the outside modules by using the Netfilter API existing in every modern Linux kernel. The communication with the outside kernel module is simple and uses regular network sockets. Partial network data filtering is accomplished by implementing a source/destination IP address matching firewall.

IV. Performance Requirements

During the development of our forensic modules, besides the problem of intercepting all virtual machine activity we also analyzed the problem of performance. Since all the activity can be intercepted, there is the risk of severe time penalties and processing speed. In order to solve this problem, at this point we will offer the possibility for an investigator to choose the logging level for a certain virtual machine. This is helpful considering that, for example, an investigator only wants to analyze the virtual memory for its contents, and it is not interested in virtual disk images or virtual network activity.

There is also the problem of network transmission overhead. Since the data that is going to be sent from the physical virtualization host to the central forensic management unit can reach important size, we will implement a mechanism of "*diff*" between two pieces of data. For example, if an investigator will want to analyze a virtual machine memory over a period, the local forensic module will sent only one initial memory snapshot and after that only what has been changed will be sent. Of course we can use the full potential of the host and provide a local aggregation module that will pre-process the data collected before sending it to the central forensic module. This approach is new to the field of cloud computing forensics and we consider it a great way to reduce the impact over the network.

The process runs in the following manner. Initially the logging modules will send a reference file and then, at an user defined time period, the modules will send a delta file, that represents the difference between the previous reference file and the current state. Thus, it will implement a snapshot mechanism at the hypervisor level. We have chosen this approach because we want to offer to the forensic investigator the possibility to have an image of what is happening inside a virtual machine between two snapshots. This feature is currently not available in other hypervisors, such as VMware's; in their case we can have a snapshot at time t_0 and one at time t_i, but we cannot know the state of the virtual machine between the *0* and *i* step.

During our research we also have focused on choosing an intermediate representation of data that is sent between the local and central forensic modules. We have analyzed different existing metalanguages for logging.

The first one is the "Management metalanguage" [17] proposed by the UnixWare community. Its advantage is that it can be used as a transparent API in the kernel modules as it provides an interface for an external host. The downside is that it needs a lot of auxiliary binary data to be sent in order to re-create the entire picture at the other end, and using it we get quickly a traffic larger than the one that can be obtained by sending only the basic snapshots. This is due to the fact that this metalanguage is designed to be used only locally over a system.

On the other side, the CEE (Common Event Expression) organization [18] proposes a set of specifications using the JSON and XML markup languages for event logging. The advantage of this approach is that CEE expresses its interfaces and does not promote an actual implementation.

Using the information gathered at this step and taking in consideration that our project is novel in this field, we will provide our own logging metalanguage, based on the CEE specifications for compatibility. Also we will keep in mind suggestion made by previous research in this direction, such as the one by Sang et al [19], for making digital forensic logging easier. Due to advantages it offers, we are going to use the JSON markup language as data envelope because it is a simple, clean, concise and human-readable format. It is also good for decoupling our cloud forensic modules because we can implement each module using its own programming language and having only a common JSON interface for using it. Along with this format we intend to use it along with a storage module that is fit for our needs. We have chosen this approach in order to have the collected data from a host in a single database and only provide management messages to the above central forensic modules.

V. Results

In this section we are going to present details regarding the results collected after the implementation of our Cloud Logging modules.

A. Testbed configuration

For testing, we have implemented two scenarios, one for the basic cloud computing framework and one for the forensic enabled architecture.

In Fig. 4 we can see the way in which the modules have been implemented and split across multiple workstations in the basic scenario. For the second scenario, as can be seen in Fig. 5, we have simplified a bit the representation and we can clearly see the nodes responsible with virtual machines storage, virtualization, management and forensics layer.

They are represented as a cluster of servers, each having the functionality presented in detail in the architecture section. As it can be seen, the entire modules found in the dotted perimeter, called "Management modules", can also be ran all on one workstation. Elements like network switches are not represented in order not to burden the graphic, but the IP address of the hosts are kept. For the "*Hypervisor Interface*" three distinct hypervisor servers have been used, each having its own security and load policies.

The hardware platform used was composed from an AMD Phenom II X6, 6 cores, 8GB RAM, RAID0 configured hard-disks running KVM as hypervisor and QEMU as a hypervisor interface, an Intel DualCore, 4GB RAM as the storage layer and an AMD C-60 DualCore, 4GB RAM as the management layer. The network used is 10/100 MB.

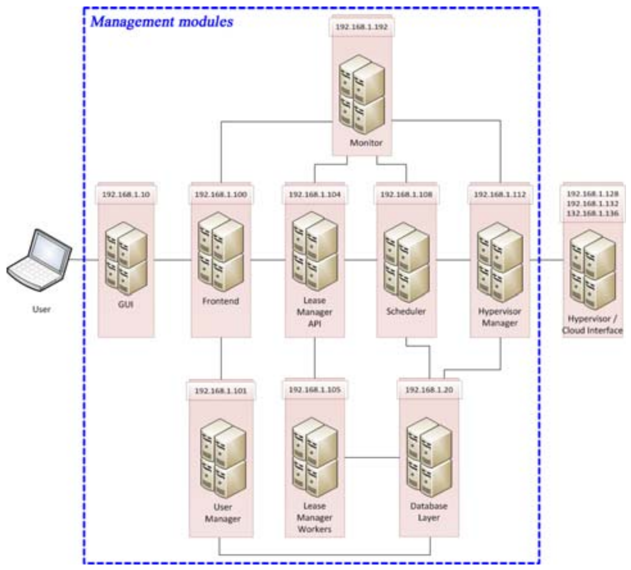

Figure 4. Mapping modules to workstations in a basic cloud computing framework

B. Experimental results

In order to test our implementation, besides the scheduler part which is thoroughly tested in our previous work published in [20], we used the Node.JS module called "node-inspector" which allowed us to get all parameters from the V8 virtual machine. In Table I we can see on the second column the time needed for a virtual job to be created on the system. On the third and fourth column we can see the amount of time needed to check the job for consistency and storage. In Table II we can see on the second column the time needed for the retrieval of a job from the internal processing queue and on the last column we see the amount of time needed by the system to check for resource availability.

Further experiments regarding the integration of our forensic modules that are responsible with collecting data from the users, were made using KVM as a hypervisor and QEMU and libvirt as drivers for the hypervisor. The tests that were made had the target set on the virtual machine used memory (RAM snapshot) and the virtual machine storage (DISK snapshot).

The testing has been conducted in several steps. The first one was to observe, measure and analyze the network communication and transmission overhead.

From our experiments we saw that using a hypervisor installed on a workstation/server with a single physical network interface has the worst time penalty as the bandwidth is split with all the virtual machine instances. In our testbed, with the hypervisor kernel module and the light virtual machine kernel module with an active/inactive rule to intercept all traffic, we saw a direct relation between the physical bandwidth and the virtual bandwidth.

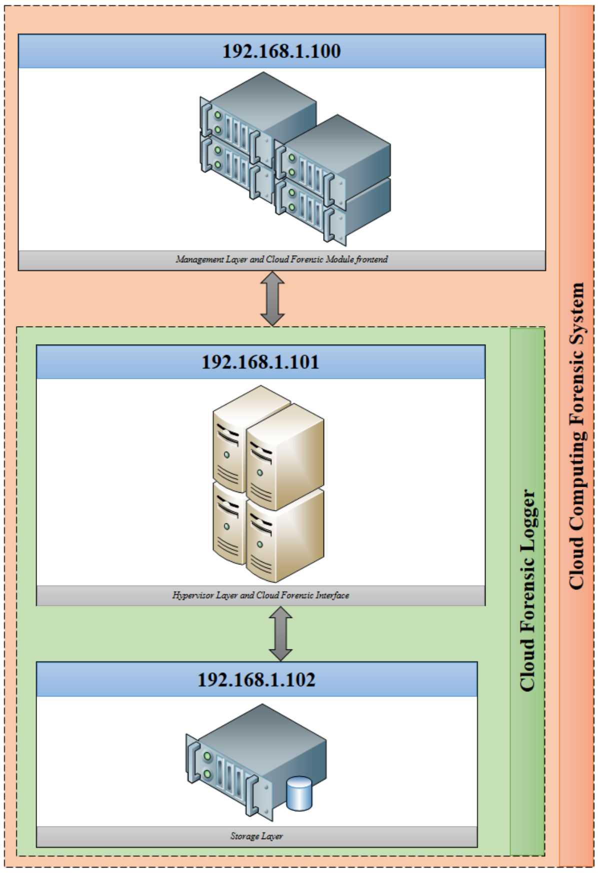
192.168.1.100
Management Layer and Cloud Forensic Module frontend
192.168.1.101
Hypervisor Layer and Cloud Forensic Interface
192.168.1.102
Storage Layer
Cloud Forensic Logger
Cloud Computing Forensic System

Figure 5. Mapping modules to workstations in a forensic enabled cloud computing framework.

TABLE I. LEASE MANAGER RESULT TABLE

#	Lease creation time (ms)	Lease check-up (ms)	Lease store time (ms)
1	204	10	1
2	205	11	1
3	289	11	1
4	208	10	1
5	262	10	1

TABLE II. HYPERVISOR MANAGER RESULT TABLE

#	Lease retrieve time (ms)	Lease resource check (ms)
1	3	25
2	3	28
3	3	51
4	3	26
5	3	39

More exactly, we have used the *iperf* tool for measuring bandwidth simultaneous between several virtual machines running in the hypervisor layer and the physical host running the management layer. The tool was ran every time with maximal bandwidth fill options using 20 sending threads, each having a TCP window size of 256KB.

With and without any rules applied inside the virtual machine kernel and having just the hypervisor monitor we have obtained results that can be seen in Table III.

We can clearly see that the available bandwidth for each virtual machine decreases as we activate/deactivate intercept modules and in our current implementation we obtain low performance metrics. The one rule was to match and intercept the entire traffic.

TABLE III. NETWORK BANDWIDTH AVAILABILITY FOR EACH VIRTUAL MACHINE WHEN RUNNING THE INTERCEPT MODULES

Virtual machine count	Bandwidth available to each virtual machine (%)	
	No rule applied	One rule applied
1	89	58
2	47	24
3	31	18
4	20	9
5	17	5

The next step was to conduct tests at the hypervisor level. The actual snapshots were taken using the following commands for RAM snapshot:

```
virsh snapshot-create-as
VM_NAME SNAPSHOT_NAME --atomic
```

and for the DISK snapshot

```
virsh snapshot-create-as
VM_NAME  SNAPSHOT_NAME  --disk-only  --
atomic
```

The process of recording the virtual machine activity was made over a period of several hours, at a time step of 10 minutes. The CPU load when conducting records using all the 6 cores was about 20%.

The results are interesting, if we take in consideration the technologies used internally by KVM. For example, RAM snapshots are made entirely from host machine RAM and do not contain necessarily consecutive RAM location. Nevertheless, in our experiments the RAM snapshots were the largest, reaching even gigabytes in size. On the other hand, the DISK snapshot is made efficiently by KVM each snapshot having a couple of tens or hundred of megabytes in size.

Bellow you can see the actual tests that were made. We have split the tests in two distinct zones, one up to 100 MB and one after this barrier. Table IV and Fig. 6 presents the data collected from our modules and the time needed to process the entire data. The transfer time between the Cloud Forensic Interface module and the Storage module is not taken in consideration, as being a constant time, of about 82 seconds for a 800 MB file. Table V and Fig. 7 presents the data collected from our modules and the time needed to process the entire data.

TABLE IV. TESTS UP TO 100 MB IN SIZE

Snapshot size (KB)	Time (ms)
4	296
454	517
1227	1136
5505	4929
10813	8000

TABLE V. TESTS OVER 100 MB IN SIZE

Snapshot size (KB)	Time (ms)
108036	58982
740032	401156
4251346	2277855

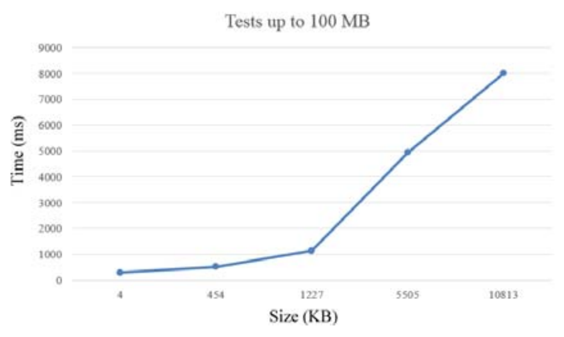

Figure 6. Tests up to 100 MB in size.

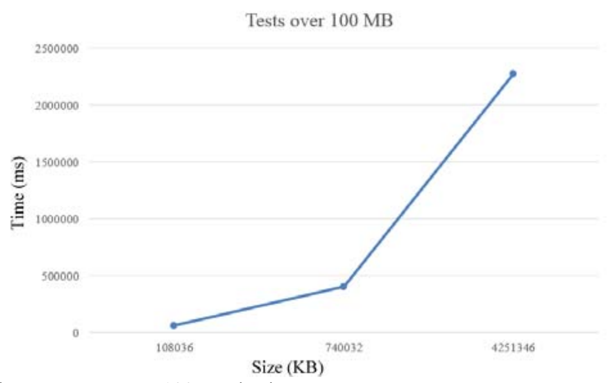

Figure 7. Tests over 100 MB in size.

VI. Conclusion

In this paper we presented a novel solution that provides to the digital forensic investigators a reliable and secure method in which they can monitor user activity over a Cloud infrastructure. Our approach takes the form of a complete framework on top of an existing Cloud infrastructure and we have described each of its layers and characteristics. Furthermore, the experimental results prove its efficiency and performance.

As we have seen, the field of cloud computing forensics and incident response is a new field for research that attracts more and more scientists. It poses a lot of challenges due to the distributed nature of the cloud but steps are starting to be made in this direction. In our paper we have presented a novel and new way in which user actions can be monitored and reproduced inside a cloud environment, even if it spreads over multiple datacenters.

Our work is focused on increasing reliability, safety, security and availability of Cloud Computing systems. The characteristics of such systems present problems when tackling with secure resource management due to its heterogeneity and geographical distribution. We presented the design of a hierarchical architectural model that allows investigators to seamlessly analyze workloads and virtual machines, while preserving scalability of large scale distributed systems.

As future work we intend to continue in this research direction in order to further optimize the snapshot and transfer algorithms, as well as the modules of the framework. Of course, further testing using more complex scenarios and a thin integration with other existing Cloud infrastructures would also help us to further improve our solution.

References

[1] S. Tayal, “Tasks scheduling optimization for the cloud computing system”, International Journal Of Advanced Engineering Sciences And Technologies, Vol5, 2011
[2] S. K. Garg, C. S. Yeo, A. Anandasivam, R. Buyya, “Energy efficient Scheduling of HPC application in Cloud Computing environments”, 2009
[3] R. Buyya, C. S. Yeo, S. Venugopal, J. Broberg, I. Brandic, “Cloud computing and emerging IT platforms: Vision, hype, and reality for delivering computing as the 5th utility”, 2008
[4] NIST SP800-86 Notes, “Guide to Integrating Forensic Techniques into Incident Response”, [Online]. Available: http://cybersd.com/sec2/800-86Summary.pdf
[5] B. Grobauer, T. Schreck, “Towards incident handling in the cloud: challenges and approaches”, in Proceedings of the 2010 ACM workshop on Cloud computing security workshop, New York, 2010
[6] G. Chen, “Suggestions to digital forensics in Cloud computing ERA”, in Third IEEE International Conference on Network Infrastructure and Digital Content (IC-NIDC), 2012
[7] M. Simmons, H. Chi, “Designing and implementing cloud-based digital forensics”, in Proceedings of the 2012 Information Security Curriculum Development Conference, pages 69-74, 2012
[8] T. Takahashi, Y. Kadobayashi, H. Fujiwara, “Ontological Approach toward Cybersecurity in Cloud Computing”, 2010
[9] S. Thorpe, I. Ray, T. Grandison, A. Barbir, “Cloud Digital Investigations Based on a Virtual Machine Computer History Model”, Future Information Technology, Application, and Service, 2012
[10] D. Birk, “Technical issues for forensic investigations in cloud computing environments”, IEEE Sixth International Workshop on Systematic Approaches to Digital Forensic Engineering, 2011
[11] A. Pătraşcu, V. Patriciu, “Beyond Digital Forensics. A Cloud Computing Perspective Over Incident Response and Reporting”, IEEE 8th International Symposium on Applied Computational Intelligence and Informatics, 2013
[12] B. Martini, K. R. Choo, “An integrated conceptual digital forensic framework for cloud computing”, Digital Investigation, November 2012
[13] Reilly, D., Wren, C., Berry, T., “Cloud computing: Forensic challenges for law enforcement”, Internet Technology and Secured Transactions, 2010 International Conference, vol., no., pp.1,7, 8-11 Nov. 2010
[14] A. Pătraşcu, C. Leordeanu, V. Cristea, „Scalable Service based Antispam Filters”, Proceedings of First International Workshop on the Service for Large Scale Distributed Systems(Sedis 2011) in conjunction with the EIDWT 2011 conference, Tirana, 2011
[15] A. Pătraşcu. D. Maimuţ, E. Simion, “New directions in cloud computing. A security perspective”, COMM International Conference, Bucharest, 2012
[16] B. Wilder, “Cloud architecture patterns”, O'Reilly Media, pp 10-14, 2012
[17] Management metalanguage, [Online]. Available: http://uw714doc.sco.com/en/UDI_specm_mgmt.html
[18] CEE Log Syntax (CLS) Specification, [Online]. Available: http://cee.mitre.org/language/1.0-beta1/cls.html
[19] Y. Du, P. Qin, J. Du, “A Log Based Approach to Make Digital Forensics Easier on Cloud Computing”, in Third conference on Intelligent System Design and Engineering Applications (ISDEA), 2013
[20] A. Pătraşcu, C. Leordeanu, C. Dobre, V. Cristea, “ReC2S: Reliable Cloud Computing System”, European Concurrent Engineering Conference, Bucharest, 2012
[21] Volatility framework, https://code.google.com/p/volatility/