Cloud Forensics

Nikola Slavov
Computer Science Department
UMBC
Baltimore, MD, U.S.A.
niko3@umbc.edu

Joshua Mpere
Computer Science Department
UMBC
Baltimore, MD, U.S.A.
jmpere1@umbc.edu

Sam Mendimasa
Computer Science Department
UMBC
Baltimore, MD, U.S.A.
sam34@umbc.edu

***Abstract*—The cloud forensics project aims to address information volatility in cloud infrastructure. Based on current Cloud Service Providers (CSPs) policies, memory is erased when Virtual Machines (VMs) are turned off. This can be done to ensure that compromised VMs do not spread malware to the network. However, this is not always the case, as often critical information on VMs that are not compromised are always loss, when a VM is turned off. To address this problem, we will setup a small private cloud using devstack, a single machine variant of openstack. Our private cloud will consist of three or more VMs running various operating systems (OS). We will infect one of the VMs with malware, use dumpIT, a memory acquisition to send a memory dump to a storage server. We will then perform memory forensics using tools such as volatility and frost to see possible information that can be recover. Afterwards, we will evaluate information that is recovered with the forensics tools.**

***Index Terms*—memory forensics, cloud, service provider, virtualization, volatility, virtual machines, devstack, openstack, dumpit**

I. Introduction

A. Memory Forensics

Memory forensics is the field of Digital Forensics that focuses on the physical memory of the machine for finding artifacts left by a malicious agent. In our framework this will mean dumping the memory of the compromised virtual machine and transferring it to another secure machine where it will be analyzed using the tool Volatility. In this way a potential investigator can gain insightful knowledge about many of the things a malicious process or user could have done while being active on that machine.

B. The nature of the cloud

We first want to establish a general understanding of what the cloud is and how it works. A cloud setup has several major components. First, it has one or more physical machines, called servers, on which it is running. Second, it has an operating system, called a host, which will organize the resources that are given by the physical machines into 'pools' which can be handed out to clients. And for the type of cloud structure that we are examining, it has Virtual Machines(VMs), also called instances, each of which manages the resource pool given to one client. That being said, a single instance can be drawing its resources from multiple physical machines and one physical machine can contain data and give resources to multiple VMs. Each VM can belong to a different client and each client can have more than one VM. Final thing needed to consider, resource pools can be, and often are, changed by the CSP to fit the clients' needs, meaning resources that were given to one VM at one point could have belonged to another VM previously and could yet get assigned to a third VM in the future.

C. Problem Statement

Our research seeks to build upon existing cloud forensics research, specifically focusing on performing digital forensics on data stored in cloud environments. In the current state of cloud computing, VMs are provisioned by CSPs and based on their current policies, volatile data is erased when the VM gets restarted or shut down unless there is a external storage provided by the CSPs. Hence, critical information is often lost. For example, if a malicious actor has access to a VM and the CSP decides to shut down the instance, critical memory would be lost and accessing the data from the incident that occurred would be extremely difficult.

D. Purpose Statement

In order to address the problem of volatile data in cloud infrastructures, we plan to first recreate the problem. We will create a small private cloud using Devstack, and compromise one of the VMs in our network with malware. We will then write a script and use dumpIt, a memory acquisition tool to send a memory dump to a storage server on the same network before shutting down the compromised VM. We will then perform memory forensics with several tools, such as Volatility, and Frost to evaluate and see the amount of information that can be recovered with each tool. Afterwards, we will develop metrics for each tool based on the information that is recovered. We plan to make this solution scalable, such that once a compromised VM is identified, CSPs can use our solution to collect information about the malicious incident, which will be useful for identifying cyber criminals. This proposal will resolve the problem statement as it will add onto existing cloud forensics research and it will also help to recover critical volatile information, that may otherwise be lost.

E. Motivation

In the current state of cloud computing, there are many challenges that a potential digital forensics investigation can encounter. We list some of them below:

- Physical Inaccessibility: The cloud servers are multi-tenant, so a cloud crime scenario differs from a typical cybercrime scenario in the sense that investigators cannot seize the machine on which the data resides. According to current regulations, the investigators instead have to rely on the CSP to acquire the correct client's information and send it to them over a secure connection.
- Trust: Since investigators rely on the CSP to send them the dumped data, additional doubts can be raised in any potential resulting court case on whether the data was handled correctly.
- Chain of Custody: Another issue related to the multi-tenancy of clouds is establishing the chain of custody of the gathered evidence in a potential court case. For the case of any data that came from the cloud, the investigators again have to rely on the CSP to provide a correct and complete list of the clients that have had access to the examined data.
- Decentralization: One of the crucial steps in a digital forensic investigation is to seize and analyze logs from the processes that were running on the machine and from any network communications the machine has had. This is challenging on a new level in cloud forensics, because the logs for any single instance or VM can be scattered across many physical devices.
- Cross-border scenarios: Considering the previous issue, it is also likely that the require evidence is located in two or more different jurisdictions, meaning the investigators will have to follow all of the laws from each of those regions. Potentially, there could be enormous road blocks here for an investigation, if the different laws conflict on what to do with the data.
- Data Volatility: When a malicious actor commits a crime in a VM on a cloud infrastructure, often critical evidence are lost when the VM is turned off by the CSPs.
- Attribution: Nowadays, the capability to obtain attribution is in high demand in the cyber forensics operational community and volatile data in cloud infrastructure is a huge hindrance to that.

Tackling all of these problems will have to be done in many steps. We wish to focus on retaining as much of the data from an incident as possible. Therefore, our solution is designed to address the volatility of information stored in the cloud, which will help in determining if there is malicious activity in the cloud infrastructure.

II. Related Works

As cloud technology is still new, little has been written about the applicability of forensics to cloud computing environments. This is even further exacerbated as CSPs do not provide a lot of information about their infrastructure. This all helps to hinder any methodology of conducting forensic investigation that occur in cloud environments. There are however tools such as Frost and UFED Cloud Analyzer that can be used to perform cloud forensics (Naaz 3). However, these are dependent on knowledge of the cloud framework. Due to such obscurity, we intent to develop a framework that is easy to understand and that forensics can easily be conducted with available open source tools. Hence, out project differs in the sense that we are developing our own framework that will be user friendly and we will also evaluate open source forensics tools.

Our proposed solution is inspired by the work detailed in Forensics framework for cloud computing by Alex and Kishore. The method proposed in that paper involves collecting forensic data in a forensic server outside the cloud providers network. It also involves a forensic monitoring plane which collects data on all inbound and outbound traffic from the providers network and sends it to the server. Alex and Kishores implementation automatically collects snapshots of the cloud providers VMs and sends them to the forensic server. Similarly to that implementation, our approach is to have a separate forensic server where the forensic data will be sent without needing cooperation from the cloud provider. There are two key distinctions between that approach and ours. First, our team will not be implementing the forensic monitoring plane for the current project, though it is a possible extension for any works that build on this research. That has the benefit that we wont need a permission from the International Telecommunication Union (ITU). The other distinction is that our approach will not have automatic collection of snapshots, but it will instead be triggered by certain system calls the cloud provider would invoke when they detect a security concern. We believe this is a benefit for our approach, since it will greatly reduce the burden placed on the cloud providers servers and bandwidth.

Another research that closely relates to our work is An efficient approach to forensic investigation in cloud using VM snapshots by Rani, DeeviRadha, and G. Geethakumari. The approach detailed in that paper involves placing Intrusion Detection Systems (IDSs) on the VMM and on every single VM. The difference here is that our approach does not involve any IDSs, which means the cloud provider will not have to worry about the extra overhead coming from them. However, our solution still implements kernel modules which hook certain system calls that could be a sign of an intrusion. Whats similar to our approach is the performing of the forensic investigation using VM snapshots. The above referenced paper does a quick overview of the exponential growth of the collection time for the snapshots in the event that we dont know which VM is the infected one. This further supports our approach, since our approach will gather a snapshot from one VM and it will be the infected one.

III. System Framework

The first major milestone we faced as we begain development was understanding cloud computing and setting up a cloud environment. Cloud computing is basically a network of servers that provides on -demand delivery of computing services. Currently, there are three types of cloud computing, Infrastructure as a Service (IaaS), Software as a Service (Saas), and Platform as a Service (PaasS). Infrastructure as a Service

is a model of cloud computing in which the CSPs provides virtualized computing services, such as servers and storage over the internet. Software as a Service is another cloud computing model, where 3rd party software such as Microsoft 365 is hosted and made available to clients over the internet. Lastly, Platform as a Service is a cloud computing model where CSPs host platforms that are then used by clients to develop, run, and manage various applications. The framework which we propose for our research is similar to a PaaS model of cloud computing.

After conducting research on cloud computing, we decided to use the OpenStack framework, specifically Devstack for our cloud environment. Our decision was based on the fact that OpenStack is open source and it is used by over 500+ companies as the basis for their own private cloud. Devstack is a single server variant of OpenStack, which basically simulates having a distributed cloud environment on one machine. The Devstack model also has all of the services as the full OpenStack. Please see figure 1, for a graphical overview of Devstack. Our framework focuses specifically on Nova, the compute service that creates VMs, and Neutron, the Software Defined Network (SDN) service that creates and manages networks.

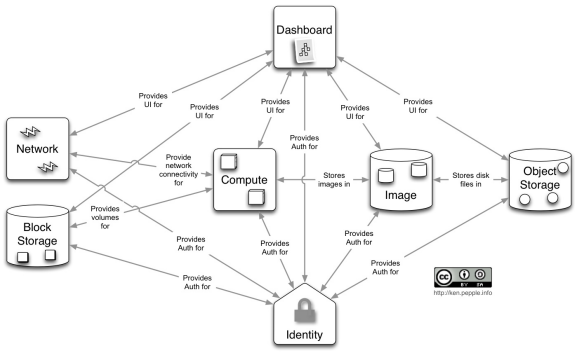

Fig. 1. Graphical Layout of Devstack Framework.

The Hardware used to set up our Cloud Framework was a Toshiba Computer, running Ubuntu 16.0, with 500 GB storage and 6+ GB RAM. We initially installed Devstack version Stein, the current and most up to date development version of Devstack. As we encountered many issues, we switched to Devstack Rocky, the currently most recent stable version of Devstack.

Once Devstack was installed and configured, we set up our cloud framework, which entails the following: Two Networks, one public and one private, and both allocate IP addresses from a pool when a new VM is created. The public and the private networks are connected with a switch, such that you are able to send data across them. On the Public Network, we installed a VM, to represent the Malware Analyst VM that would receive the memory dump files for forensic analysis. On the Private network, we installed two VMs to represent the malicious actor and the compromised client. Please see figure 2 and 3 below for the frameworks graphical representation. This current setup represented what we imagine our framework would be on Devstack, hence it was more of an experimental setup that used the default cirros image that is preinstalled on Devstack.

Displaying 3 items

Instance Name	Image Name	IP Address	Flavor	Key Pair	Status	Availability Zone	Task	Power State	Time since created	Actions
vm3	cirros-0.3.5-x86_64-disk	172.24.4.16, 2001:db8::18	cirros256	vm3-2	Shutoff	nova	None	Shut Down	2 days, 13 hours	Start Instance
vm2	cirros-0.3.5-x86_64-disk	130.85.247.3	cirros256	vm2	Shutoff	nova	None	Shut Down	2 days, 13 hours	Start Instance
vm1	cirros-0.3.5-x86_64-disk	130.85.247.22	m1.tiny	vm1	Shutoff	nova	None	Shut Down	2 days, 13 hours	Start Instance

Fig. 2. VMs Network overview, including IPs addresses and image flavors.

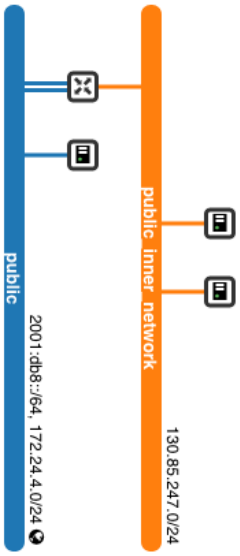

Fig. 3. Cloud Infrastructure network setup.

As we began changing our experimental setup into our actual setup with the correct VMs and images, we ran into difficulties and ultimately, we kept the structure of our same framework but instead, we set it up in VirtualBox and performed the memory forensics from there. Please reference section 5 Issues, for additional details.

The VirtualBox design is a copy of the initial design that was proposed for the cloud environment. Virtualbox is a free and open source hypervisor, which worked well for our project as we could do virtualization and create the correct VMs and network needed for routing data and sending and receiving the memory dump from a compromised system. Please see figure 4 which contains the graphical overview of the VirtualBox design.

The VirtualBox setup unlike the cloud experimental setup has the correct VMs. For the setup, the VMs for the Malicious actor and the Forensic Analyst had Kali Linux installed as the Operating System, and the Clients VMs to be exploited ran Windows 7 and Ubuntu 16. The windows 7 VM was the compromise VM in this design.

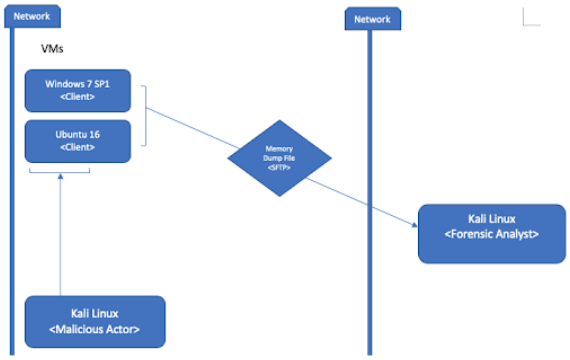

Fig. 4. Virtualbox Design showing networks and flow of data.

IV. Forensic Analysis and Report

Our Forensic procedure begins with us collecting the evidence from the compromised machine. We use a tool called Dumpit. This is an executable that simply takes a snapshot of the host's physical memory. Basically that mean the ram is saved to a .raw file. Then we sent the ram file to the forensic server using the SCP protocol to make sure the file was transferred securely.

In the Organization phase we analyzed the ram file using Volatility. Volatility is a completely open collection of tools written in Python for the extraction of digital artifacts from volatile memory samples. The extraction techniques offer visibility into the runtime state of a system. Each tool that extracts information from a memory sample is called a plugin. To start with the imageinfo plugin was used and that gathered all the necessary information Volatility needs to perform a proper analysis. In order to analyze an image with certain Volatility plugins it needs to to which Operating system its looking at. We get this information as a Suggested profile which is in the output of the imageinfo plugin. In our case the Suggested profile was Win7SP1x86 this means the Operating system was a Windows 7 Service Pack 1 32 bit architecture.

```
Volatility Foundation Volatility Framework 2.6
ERROR   : volatility.debug    : Please specify a location (-l) or filename (-f)
root@kali:~/Downloads# volatility -f IE10WIN7-20181023-185341.raw imageinfo
Volatility Foundation Volatility Framework 2.6
INFO    : volatility.debug    : Determining profile based on KDBG search...
          Suggested Profile(s) : Win7SP1x86_23418, Win7SP0x86, Win7SP1x86
                     AS Layer1 : IA32PagedMemoryPae (Kernel AS)
                     AS Layer2 : FileAddressSpace (/root/Downloads/IE10WIN7-2018
1023-185341.raw)
                      PAE type : PAE
                           DTB : 0x185000L
                          KDBG : 0x82968c78L
          Number of Processors : 2
     Image Type (Service Pack) : 1
                KPCR for CPU 0 : 0x82969d00L
                KPCR for CPU 1 : 0x807c6000L
             KUSER_SHARED_DATA : 0xffdf0000L
           Image date and time : 2018-10-23 18:53:43 UTC+0000
     Image local date and time : 2018-10-23 11:53:43 -0700
root@kali:~/Downloads#
```

Fig. 5. This is the imageinfo plugin which is capturing the operating system profile Volatility uses to scan memory samples.

Our Next step was to find out what exactly was running on the system at the time we acquired this memory. We used the psscan plugin which scans the kernel for processes and can also find hidden processes. Since we were enumerating the processes we wanted to immediately apply a visualization to the output in order to better understand the parent child relationships between processes. Therefore we used the dot graph renderer which is a tool in Volatility that created a dot graph. This proved useful for 2 reasons; first we were able to see which process was spawned by which second we were able to see which processes tried to hide themselves by the ones shaded in gray

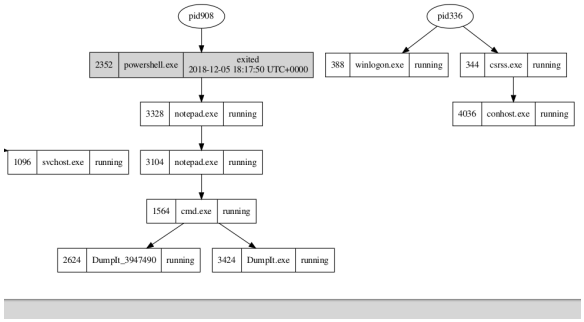

Fig. 6. Here we see the processes in parent child relationship form.

Once we finished that step we wanted to follow the clues we found about the malicious processes on the system. We knew that the powershell plugin was malicious therefore we honed in on it. The next plugin I used was the privs plugin. A privilege is the permission to perform a specific task like debugging a process or shutting down a computer. From a forensic perspective We were most concerned with explicitly enabled privileges because that shows awareness and intent. So I used that plugin and applied the grep unix command in order to find the powershell.exe process using its process ID which I found in the last step. One privilege I found explicitly enabled was the SeDebugPrivilege which allows one process to read from or write to another processes memory space. Which allows to malware to perform code injection. This is a privilege commonly found in malware.

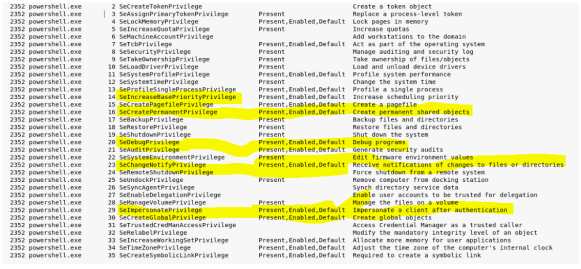

Fig. 7. Highlighted are the privileges which trigger some red flags for instance the SeDebugPrivilege or the SeChangeNotifyPrivilege and SeImpersonatePrivilege.

The last plugin I used was the malfind plugin because this one allowed me to see which processes injected code where. The malfind plugin is designed to hunt down remote code injection. First one process allocates memory in another process with the "PAGE EXECUTE READWRITE" protection bit set. The first process then does WriteProcessMemory and then CreateRemoteThread and boom its injected. When I performed malfind I found that the notepad.exe process injected code into

itself to spawn a command prompt this was confirmation of that same information displayed in the process dot graph.

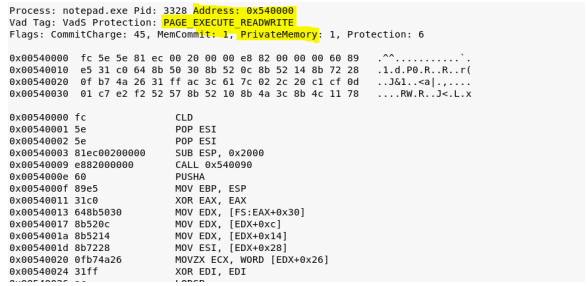

Fig. 8. Here we see the notepad.exe have the PAGE EXECUTE READWRITE bit set which is how we know it was used for remote code injection.

V. Issues

Although we intended to set up our framework on a Devstack environment, that did not prove to be feasible. Our attempted setup was able to contact the Internet from the host, to launch instances, and to have the instances communicate with each other, but it had troubles when it came to having those instances be accessible from outside the cloud.

Additional challenges presented themselves when we attempted to work around this issue. Our workaround fix was to try to launch instances that were malicious or virtual machines that were compromised, and once a part of the cloud we would extract data from them to analyze. That failed, however, since Devstack was not properly launching the instances that we tried to give it. The launch attempts resulted in an error state that didn't produce very meaningful feedback to us, due to which we did not gain a good understanding of why the instances are not starting up.

VI. Conclusions and Findings

Our research produced a few meaningful findings. First, we were able to successfully compromise the victim VM, getting nearly all privileges in it. Second, upon analyzing the RAM from the investigator's perspective, we were able to identify a process that acted suspiciously by producing child notepad processes that themselves produced terminal processes. Third, upon looking at the process privileges, we found that the suspected process had indeed gained privileges that normal non-system processes wouldn't have. Fourth, by looking at the process logs, we were able to locate a process that had been code injected by the suspect process by using its debug privilege. Finally, we concluded that the framework is feasible and can produce meaningful indicators of compromise on a host in a cloud environment.

VII. Future Work

To continue our work on this project, we plan to first run benchmarks to measure the time and computing power cost that the dumping of memory and sending it off has on each instance. Our approach would be to use the system clock in the script that dumps the memory, which will tell us in how much time it was accomplished. And in conjunction with that, we'd use the system logs that will tell us how much memory was given to the dumping process. As for the sending of the data, our approach could involve sending the data across different distances and different types of networks.

Additionally, we plan to add a third machine to our framework, a forensic server. The server will be the new destination for the memory dump data and will be outside the cloud. Once the data is there, it will be stored by the server for several days, allowing investigators to extract it to their own machines. The intended setup will also involve automatic deletion after the data has been extracted, so as to save the server's space.

We also intend to resolve all Devstack issues so that the framework can be tested on an actual cloud. For starters, we want to work more on getting the instances to be accessible from outside the cloud, which is a basic requirement for a CSP to function. Once we have that down, our focus will be to get instances to launch with commonly used operating systems like Windows, Mac OS, and Linux.

Finally, we intend to design another framework after we are done optimizing this one. As there are many issues other than data volatility with the cloud, we believe a new framework will have to be made in the future to make the digital forensics process go more smoothly.

References

[1] Case, Andrew. "Memory Forensics: The Path Forward."
[2] Naaz, Sameena. "Comparitive Study of Cloud Forensics Tools."
[3] Popovi, Kreimir. "Cloud Computing Security Issues and Challenges - IEEE Conference Publication." Design and Implementation of Autonomous Vehicle Valet Parking System - IEEE Conference Publication, Wiley-IEEE Press.
[4] S.Subashini. "A Survey on Security Issues in Service Delivery Models of Cloud Computing." Egyptian Journal of Medical Human Genetics, Elsevier, 15 July 2010.
[5] Ligh, Michael. Case, Andrew. Levy, Jamie. Walters, AArons. "The Art of Memory Forensics." Wiley. July 2014.