Cloud Computing Log Evidence Forensic Examination Analysis

Sean Thorpe[1], Tyrone Grandison[2], Indrajit Ray[3]

[1]Faculty of Engineering and Computing
University of Technology, Kingston, Jamaica
sthorpe@utech.edu.jm

[2]IBM Research,
York Town Heights, New York, USA
tyroneg@us.ibm.com

[3]Department of Computer Science
Colorado State University, Fort Collins, USA
indrajit@cs.colostate.edu

Abstract. Forensic analysis in the context of physical evidence is a relatively mature field. The computerization of society has led to the emergence of digital forensics and now the popularity of cloud computing has sparked interest into cloud forensics. Our goal in this paper is to enable cloud forensics, by using the theory of abstraction layers to describe the purpose and goals of virtual machine (VM) forensic digital examination analysis tools. Using VM abstraction as a meta abstraction layer, we identify how VM log forensic audit tools by generalization can introduce errors and provide requirements that such tools must follow to avoid these errors. Categories of VM log forensic analysis types are also defined based on the VM abstraction layers.

Keywords: abstraction, virtual machines, tool, log, cloud, forensic, requirement.

1 Introduction

While cloud forensics is a field that is still in its infancy, it is gaining increased relevance as a growing number of companies look to leverage cloud technologies to unlock the advantages from economies of scale and increased focus on their core strategic mission. VM abstraction is based on the underlying theory of abstraction layers and is a fairly new concept that builds on prior work [9, 10]. At the University of Technology in Jamaica, we explore several case studies for designing and testing VM log forensic audit tools that considers the VM abstraction properties.

We expand on the earlier work started by Carrier [1]. Ideally any good digital investigator or system administrator must continuously remind himself as to what it means to have a cloud digital forensic analysis tool? We adopt from [1] the argument that we must find ways to categorize the different types of analysis tools, in our case

VM log analysis tools. For the cloud computing environment, this is a pertinent matter, as file fragments within the data cloud are located over distributed network domains in various jurisdictions.

We contend that viewing file metadata within the cloud has to be conscientiously supported through the use of the hypervisor system logs. Bear in mind that the hypervisor within the storage area network (SAN) of the private data center is nothing more than a meta operating system (OS) layer, i.e. a para-virtualization layer, that allows one to observe the behavior of the physical OS. Using raw device mapping (RDM) techniques, the hypervisor synchronizes the behavior of the physical OS using the standard BIOS clock setting running across NTP servers. In a follow-up paper we will discuss the details of the RDM service layer functions. For example, we posit that the VM investigator can view the hypervisor files and directories of a suspect VM host system by using either specialized forensic software like our VM log auditor [12, 13, 15] or by using the underlying physical operating system (OS) of an analysis system and viewing the files by mounting the drives provided that support the running VM instances. Both methods allow the investigator to view evidence in allocated files, but only the specialized forensic software allows him to easily view unallocated files. The latter becomes preferable when considerations for examination analysis are required within a public cloud where no one system administrator or VM investigator has physical jurisdiction. Additional tools will be required when relying on the OS. Clearly both approaches allow the VM investigator to find digital evidence and therefore should be considered forensic tools. It is however unclear as to how we should compare and categorize these VM tools especially when considering a cloud computing digital investigation, where a multiplicity of meta abstraction layers exist and have to be carefully defined.

The high-level process of cloud digital forensic investigations, unlike any other traditional digital forensic investigation, includes the acquisition of data from a VM host hypervisor log source, analysis of the log data, extraction of the potential log evidence, preservation and presentation of the potential log evidence [9]. Previous work has been done on the general theory and requirements of data acquisition [7] and the preservation of evidence [4]. This paper proposes the need for VM forensic tools that can be used for the analysis of data and extraction of evidence as we see applicable to virtualized cloud computing environments.

This paper examines the nature of tools in the VM digital forensics setting and proposes definitions and requirements. In our considerations for a cloud forensic investigation, our observations take into account existing digital forensic tools which have produced results that have been successfully used in prosecutions, but were not designed with forensic science needs as base requirements.

In an existing digital investigation, the tools provide the investigator with access to evidence, but typically do not provide access to methods for verifying that the evidence is reliable. This is necessary when approaching any digital forensics investigation albeit a physical crime scene or within the distributed data centers from which these VMs are running. From a scientific point of view, such a concern is a good candidate for a standard legal requirement for VM data cloud service environments in the future.

The core concept of this paper is the basic notion of VM abstraction layers. Abstraction layers exist in all forms of digital data and therefore abstraction layer

functions should be available from the tools used to analyze them. The idea of using tools for layers of abstraction is not new, but a discussion of the definitions, properties, and error types of VM abstraction layers when used with cloud digital forensics has not occurred. The concepts proposed here are applicable to any cloud/digital forensic analysis type, which will be defined later in this paper.

This paper begins with definitions regarding digital forensic analysis tools, followed by a discussion of abstraction layers. The abstraction layer properties are used to define analysis types and propose requirements for digital forensic analysis tools. We present these abstraction properties as a metadata layer from which the VM hypervisors can extract evidence. We purport this argument consideration and adapt its merit as a part of a VM abstraction layer design. Finally, we conclude the paper along with identifying opportunities for future work.

2 Definitions

As defined in [8], digital forensic science is:

"*The use of scientifically derived and proven methods toward the preservation, collection, validation, identification, analysis, interpretation, documentation and presentation of digital evidence derived from digital sources for the purpose of facilitating or furthering the reconstruction of events found to be criminal, or helping to anticipate unauthorized actions shown to be disruptive to planned operations*".

This definition covers the broad aspects of digital forensics from data acquisition to legal action. This paper is limited in scope to the phases of identification and analysis that can be adopted for the cloud digital investigation. These phases come after the collection and validation phases, which handle the acquisition of valid data from the VM host suspect system. The identification and analysis phases examine the acquired data to identify evidence. Using the broad definition from [8], one can define the goal of the identification and analysis phases as adopted for the cloud digital forensics environment.

To identify VM log digital evidence using scientifically proven methods can be used to facilitate or further the reconstruction of log events in a cloud investigation. As with any investigation, to find the truth one must identify data that:

- Verifies existing data and theories, i.e. Inculpatory Evidence.
- Contradicts existing data and theories, i.e. Exculpatory Evidence

For the hypothetical cloud forensic investigation, to find both evidence types, all acquired data must be analyzed and identified. Analyzing every bit of data is a daunting task especially when confronted with the potential size of distributed VM storage cloud disk systems. Furthermore, the acquired data is typically only a series of byte values from the underlying hard disk or network wire from which the virtual machines gain access. Raw data like this are typically difficult to understand. In cases of multi-disk systems, such as RAID and Volume Management, acquired data from a single disk cannot be analyzed unless they are merged with the data from other disks using complex algorithms. This problem compounds itself when the VM investigator has to now consider distributed disk cluster volumes over which the VM file instances are running.

We further support the argument presented by Carrier [1] that summarizes the complexity problem in digital forensics. In order to solve the complexity problem, tools are used to translate data through one or more layers of abstraction until it can be understood. We believe that this approach is very necessary for the cloud forensic environment. Our approach strongly favors the need to rely on the integrity of the hypervisor logs to do this [9, 12, 13, 15]. For example, to view the contents of a directory from a file system image, we design VM log audit forensic tools that aim to process the file system structures so that the appropriate values are displayed. The data that represents the VM source files in a directory exist in formats that are too low-level to identify without the assistance of specific VM log auditing tools. The directory is a layer of abstraction in the file system. For the virtual machine hypervisor, the directory is a meta-layer of abstraction for the physical host OS file systems. Examples of non-file system layers of abstraction include ASCII, ML Files, Windows Registry, Network Packets, and Source Code.

Similarly, the well known *Quantity Problem* [14] in existing digital forensics places significant emphasis on the fact that the amount of data to analyze can be very large. Without realizing it, the cloud computing deployment models (i.e. public, private, hybrid, and community) multi-folds the size of the scale of data that needs to be analyzed to Exabyte storage amounts. We suspect in the very near future that Zettabyte VM storage limits will become a reality in a typical cloud forensics investigation. It is inefficient to analyze every single piece of data. Therefore data reduction techniques are required to solve this, by grouping data into larger events or by removing known (potentially less useful) data. Data reduction techniques are examples of abstraction layers, for example:

- Identifying known network packets using Intrusion Detection System (IDS) signatures
- Identifying unknown entries during log processing
- Identifying known files using hash databases
- Sorting files by their type

This paper is concerned with the adoption of VM analysis tools that translate data from one layer of VM abstraction to another. As we experiment with VM log forensic audit tools, it is proposed that the purpose of the cloud digital forensic log analysis tool is to accurately present all data at a layer of meta abstraction and format that can be effectively used by an investigator to identify evidence [9]. The needed layer of abstraction is dependent on the skill level of the VM investigator and the type of investigation requirements. In our work particular to this skill level requirement is the need for the investigator to understand the architectural abstraction design parameters of the SAN within the data center from which these VM instances are running. The raw data is stored within the logical unit number (LUN) addressed clusters on the physical disk segment blocks that host the VM instances. For example, in some cases viewing the raw contents of a SAN LUN VM host disk block location is appropriate whereas other cases will require the disk block to be processed as a file system structure. Tools are required to provide these options. The next section will cover VM abstraction layer properties in more detail.

3 Layers of VM Abstraction

We posit that layers of VM abstraction can be used to analyze large amounts of hypervisor log data in a more manageable format which are compliant with the well known advanced digital data forensic (ADDF) formatting. This is a necessary feature in the design of modern cloud digital systems because all data, regardless of application, are represented on an underlying physical disk or network in a generic format, where bits are set to one or zero. To use this generic storage format for custom cloud applications, the bits are translated by the VM log audit tool applications to a structure that meets its needs. The custom format should be treated as a layer of VM abstraction.

A basic VM abstraction example is ASCII or EBCIDIC in data format. In this work we will explore only the US ASCII format. Every letter of the US English alphabet is assigned to a number between 32 and 127. When a text file is saved, the letters are translated to their numerical representation and the value is saved on the media as bits. Viewing the raw file shows a series of ones and zeros. By applying the ASCII layer of abstraction, the numerical values are mapped to their corresponding characters and the file is displayed as a series of letters, numbers, and symbols. A text editor is an example of a tool operating at this layer of abstraction. Ideally this means VM log text editors will be new requirements in the very near future for a typical log audited cloud domain.

Each VM abstraction layer can be described as a VM log function of inputs and outputs. The VM layer inputs are data and a translation rule set [13]. The rule set describes how the input data should be processed, and in many cases is a design specification of the VM object. The outputs of each VM layer are the data derived from processing the VM input log data and a margin of error. In the ASCII example, the inputs are the binary data and the ASCII mapping rule set. The output is the alphanumeric representation.

The output data of a VM layer can be fed as input to another VM layer, as either the actual data to be translated or as descriptive meta-data that is used to translate other input data. In the ASCII example, if the file was an HTML document then the output of the first layer, the characters, would be used as the input data to the HTML layer of abstraction. This layer takes the ASCII data and the HTML specification as input and outputs a formatted document. An HTML browser is an example of a tool that performs this translation.

A well known example of descriptive metadata as output is the block pointer and type fields in a UNIX file system inode structure. The inode structure describes a file and includes a descriptor that indicates if the inode is for a file, directory, or some other special type. Another inode field is the direct block pointer that contains an address of where the file content is stored. Both values are used as descriptive data when processing the next VM layer of abstraction in say a UNIX VM host files system. The address is used to identify where to read data from in the file system and the type value is used to identify how to process it, since a directory is processed differently than a file. In this case, the output of the inode layer is not the only input to the next layer because the entire file system image is needed to locate the block address.

VM abstraction layers can occur in multiple levels. The file system itself is a layer of abstraction for the stream of bytes from the disk media. Within the file system are additional layers of abstraction and the end result is a smaller stream of bytes that represents a file, which is then applied to an application level of abstraction and it is processed further. Multiple levels of abstraction layer characteristics are discussed further in Section 3.2.

3.1 VM Abstraction Layer Errors

We posit that each layer of VM abstraction can introduce errors and therefore a margin of error can be identified as an output value. The errors discussed in this paper are not a comprehensive list of errors that exist during the cloud investigation process. Errors introduced from the attacker covering his tracks, from faulty imaging tools, or from an investigator misinterpreting the results of a tool are not covered. Such results in existing digital investigative tools are explored in [2].

We would like to formulate the argument that VM abstraction layers can introduce two forms of errors: VM Tool Implementation Error and VM Abstraction Error. VM Tool Implementation Error is introduced because of programming and tool design errors for software written specifically to aid the virtualization evaluations. Examples of this include programming errors for two reasons: (a) because the tool uses an incorrect specification which is unknown to the host VM systems, and (b) errors because the tool uses the correct VM host system specifications but the original application did not. This latter error is the most difficult to calculate because it requires extensive testing and code review. Efforts by the NIST Computer Forensics Tool Testing Group [6] can help recognize and fix this type of error. Ideally, one can assume that if a fault (or bug) has been detected, it will be fixed and a new version of the tool will be released. Therefore, a VM investigator can keep this value minimal by keeping up to date on VM tool fixes.

We also consider the argument that in order to help identify the risk of unknown faults; a VM Tool Implementation Error could be calculated for each VM tool used. The calculation would be based on the number of faults found in recent years and the severity of each. The range of this number in terms of the lower and upper bounds would be dependent on the a priori historical value of the VM complex event log data. As a basic step we can let this number be in the range 0 to 100 and the operations are range-preserving operations whose combinatorial function which can be defined later. Ideally it would be in a VM vendor's best interest to have this value as small as possible, it could be difficult to calculate with closed source applications because faults that are not publicized could be quietly fixed and not added to the calculation.

The second type of suggested error for consideration is the VM Abstraction Error, which is introduced because of simplifications used to generate the layer of VM abstraction. This type of error occurs when a layer of VM abstraction is not part of the original design. For example, a VM host file system image has several layers of VM abstraction in its design. Going from one layer to another would introduce no Abstraction Error. Alternatively, a VM Abstraction Error could exist in an IDS system that reduced multiple VM network packets into a specific attack. As the IDS did not know with certainty that the packets were part of an attack, it introduced a margin of

error. The error value for the IDS should be different for the different attacks that it was trying to detect. This error value could be improved with research and better VM abstraction techniques.

Using these ideas, we can define the VM Abstraction Layer Error Problem as the errors that are introduced by the layers of VM abstraction. Calculating a margin of error for each VM layer and taking it into account while analyzing the resulting metadata could solve this problem. To help mitigate the risk associated with this problem, one needs access to the VM layer inputs, VM rule set, and outputs to verify the translation. Similar to the tool implementation error , the translation error will also be a range preserving value between 0 to 100 .We still need however to perform a sequence of hypothesis based experiments with the existing hypervisor historical log data to derive certainty values.

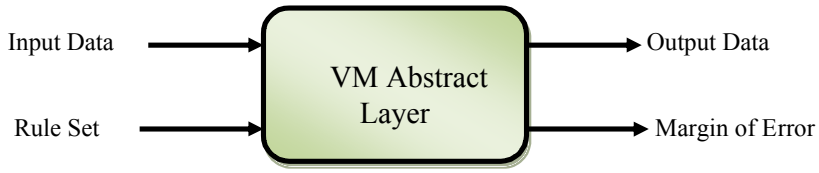

Fig. 1. VM Abstraction Layer

3.2 VM Abstraction Layer Characteristics

Not all layers of VM abstraction or VM tools are the same. This section will suggest four characteristics that can be used to describe a layer and the tools that process them as adopted from [1].

We posit that the VM Abstraction Error can be used to describe a layer by identifying it as a VM Loss prone Layer or a VM Lossless prone Layer. A VM Loss prone Layer is one that has a greater than zero margin of VM Abstraction Error associated with it. A VM Lossless prone Layer is one that has zero margin of VM Abstraction Error. VM Tool Implementation Error is not included in these definitions because it relates to a tool, not a layer, with specific value. VM File system abstraction layers and ASCII are examples of VM Lossless prone Layers, whereas IDS alerts are an example of a VM Loss prone Layer.

A layer can also be described by its VM mapping attributes [11, 12]. A one-to-one layer has a unique mapping so that there is a one-to-one correlation between any input log and output log (i.e. on the VM target log evidence server). The ASCII example and many layers of a file system fall into this category. The input of these VM layers can be determined given the output and rule set. A multi-to-one VM layer should have a non-unique VM mapping where an output can be generated by multiple VM input values. The SHA1 hash is an example of this. Two inputs can generate the same SHA1 checksum value, although it is difficult to find them. Another example of

multi-to-one is with IDS alerts. One can generally not recreate the entire packet sequence that generated an alert.

There can be layers of VM abstraction within a higher-level layer of abstraction. In the case of VM SAN disk storage, there are at least four high-level layers of abstraction. The first is the VM physical media layer, which translates the unique on-disk format to the general format of sectors and LBA and CHS addressing that the hardware interface provides. The second layer is the VM media management layer that translates the entire disk to smaller partitions. The third layer is the VM file system layer that translates the partition contents to files. The fourth layer is the VM application layer that translates the file content to the needs of an application.

We posit that the last layer in a level of VM abstraction can be described as the VM Boundary Layer. The output of this VM layer is not used as input to any other VM layers in that level. For example, the raw content of a file is a VM Boundary Layer in the VM file system level. The translation to ASCII and HTML is done in the VM application layer level.

We further consider that the purpose of VM translation tools is to convert the data to the next VM layer of abstraction. A VM presentation tool should be one that takes the data from the VM translation tool and display it in a way that is useful to the VM investigator. From the VM investigator's point of view, these tools should not be separate. VM Layers that produce a large amount of output data may separate the tools for efficiency.

As an example, we recommend that a VM Translation Tool could analyze a VM file system image and display the hypervisor log file and directory listings in the order that they existed in the image. One VM presentation tool could take that data and sort it by VM hypervisor log directory to display just the files within a given directory, similar to the output of 'ls' or 'dir' in UNIX and Windows respectively. A second recommendation is that any VM presentation tool should sort the entries by the Modified, Access, and Changed (MAC) times of each file and display a timeline of file activity [13, 14]. The same data may exist in each result, but in a format that achieves different needs.

4 The Process

Our approach for VM hypervisor kernel log extraction demands of us to first synchronize the logs from the existing VM Host source Operating system environment to our target. We enforce this synchronization policy by applying schematic and transformation mapping techniques as a part of our virtual machine log auditor tool kit discussed in our prior work[11,12]. We represent this log extraction process as a part of a cloud computing digital investigation process model[9].

5 Conclusion

This paper examined the role of VM tools during a proposed cloud digital forensic examination analysis and has documented the use of VM abstraction layers to support this task. The use of VM abstraction layers is an adopted idea that builds on the theory of abstraction layers in the existing literature, but little has been written about it by way of the work that is now on in earnest in the field of cloud computing security and forensics. The paper proposed definitions and error types associated with VM abstraction layers so that they can be refined and expanded upon by the cloud digital forensics community. Our ongoing work looks at further case study examples of how the VM investigator can benefit from VM forensic visualization tools that automates the abstraction layer properties and functions described in this paper. Through VM tool visualization [15], we expect to navigate the hypervisor log directory structures in such a way that explores quick turnaround time on examination and analysis of potential log evidence.

References

1. Carrier, Brian. : Defining Digital Forensic Examination and Analysis Tools. In Digital Research Workshop II, 2002. Available at: http://www.dfrws.org
2. Casey, Eoghan. : Error, Uncertainty, and Loss in Digital Evidence. International Journal of Digital Evidence, 1(2), Summer 2002
3. Gutmann, Peter. : Secure Deletion of Data from Magnetic and Solid-State Memory. In Proceedings of the 6th USENIX Security Symposium, 1996Foster, I., Kesselman, C.: The Grid: Blueprint for a New Computing Infrastructure. Morgan Kaufmann, San Francisco (1999)
4. Hosmer, Chet. : Proving the Integrity of Digital Evidence with Time. International Journal of Digital Evidence, 1(1). Spring 2002.
5. Microsoft Organization. FAT: General Overview of On-Disk Format, 1.03 edition, December 2002.
6. NIST. Computer Forensic Tool Testing (CFTT). Available at: http://www.cftt.nist.gov
7. NIST CFTT. Disk Imaging Tool Specification, 3.16 edition, Oct 2001. www.ijde.org 11 International Journal of Digital Evidence Winter 2003, Volume 1, Issue 4 www.ijde.org 12
8. Palmer, Gary, A Road Map for Digital Forensic Research. Technical Report DTR-T0010-01, DFRWS, November 2001. Report from the First Digital Forensic Research Workshop (DFRWS).
9. Thorpe, Sean, Ray, Indrajit, Grandison, Tyrone. : Towards a Synchronized Cloud Forensic Framework. Proceedings of the 1st Cyber Forensics Conference 2011, University of Stratcylde Publishers (Editor George Weir).
10. Thorpe, Sean, Ray, Indrajit. : The Theory of a Cloud Computing Digital Investigation using synchronized virtual machine kernel logs (Unpublished PhD thesis).
11. Thorpe, Sean, Ray, Indrajit, Grandison, Tyrone. : Towards Associative Mapping for the virtual machine log synchronized environments. Proceedings of CISIS 2011. And Springer Verlag Edition Preprint.
12. Thorpe, Sean, Ray, Indrajit, Grandison, Tyrone. : Towards a Transformational Mapping of the virtual machine log synchronized environment. Proceedings of CISIS 2011. And Springer Verlag Edition Preprint.

13. Thorpe, Sean, Ray, Indrajit. : Detecting Temporal Inconsistency in Virtual Machine Activity Timelines. Submitted to the Journal of Information Assurance and Security (Awaiting Review)
14. Thorpe, Sean, Ray, Indrajit. : File Timestamps for Cloud Digital Investigations. Submitted to the Journal of Information Assurance and Security (Awaiting Review).
15. Thorpe, Sean. : The Virtual Machine Log Auditor. Submitted to the 1st IEEE International Workshop on Security and Forensics in Communication Systems (Awaiting Review).