Information Assurance and Security Letters.
ISSN 2150-7996 Volume 2 (2011) pp. XXX-XXX
© MIR Labs, www.mirlabs.net/iasl/index.html

The Virtual Machine Log Auditor

Sean Thorpe[1], Indrajit Ray[2] Tyrone Grandison[3] Abbie Barbir[4]

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

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

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

[4] Bank of America,
abbie.barbir@bankofamerica.com

***Abstract*: With the increased use of compute clouds, forensic science requires tools that enable investigation and discovery. The Virtual Machine Log Auditor (VMLA) is one such tool. It is a graphical tool that allows a cloud computing forensic investigator to create a timeline of virtual machine (VM) hypervisor log events that were gathered from one or more physical operating system (OS) sources. This paper describes the design, implementation and use of the VMLA. The VM timestamp hypervisor log information visualized by the VMLA tool refers to VM hosted physical OS Modification, Access and Creation (MAC) times, copied from the storage area network (SAN) disks. The paper also gives an overview on how to improve the existing prototype.**

Keywords: Cloud, Log, Auditor, Forensic, hypervisor**.**

I. Introduction

Cloud computing services have become an attractive model for accessing the traditional shared pool of network resources [1]. The buzz around clouds arguably has created more interest than its predecessors of Grid and Mainframe computing because of the on-demand economies of scale benefits from these rental service models. In the same breadth the potential for criminal activity is ripe, and presents significant concerns for the IT forensics community, especially as it relates to law enforcement.

Existing tools are available that aid a computer forensic investigator in analyzing SAN storage media and the overlying file systems. The literature however shows very little evidence anywhere on the use of cloud forensic auditing tools to support investigations within the virtualization layers, i.e. meta abstraction layer, that runs within the existing SAN of your data center.

Arguably some of the likely cloud adopters to watch are Encase [5], the Sleuth Kit [4], and the Forensic Toolkit [2]. They focus mainly on evidence recovery, i.e. recovering deleted or hidden data. Beyond simple searching and indexing functionality, these tools however have limited abilities to further analyze the data that can be recovered from cloud storage, regardless of it being in the private or public domain. Searching for keywords, file types, or file hashes might be sufficient when trying to locate incriminating material on an existing physical system, but clearly insufficient when trying to reconstruct virtual machine operating system (OS) or hypervisor events that have taken place on a cloud system. The VMLA allows a forensic investigator to construct and view a timeline of virtual machine log events based on the information found in the system under cloud digital investigation.

In the current prototype, the timeline consists of log events that come from VM hypervisor sources running Xen Citrix and VMWare essx3i operating on both Windows 7 and Linux Mandriva 10 test platforms. MAC times are retrieved from the hypervisor system, application, error and security logs. Currently, an investigator needs to analyze the output from these hypervisor log sources separately and make notes to correlate events. The hypervisor logs are copied from the VM host source via the file transfer protocol (FTP) method to an Oracle11g target evidence server for further examination and analysis [11,12]. In due course, the VMLA can incorporate events from multiple VM host sources into a single list of time-sorted entries.

The VMLA is the only tool that lets a user import hypervisor log events from distinct OS sources and order them according to time values without the need for setting up special monitoring. This paper is motivated by prior work done in [10,13]. Furthermore, it is the only tool that lets a user generate a hierarchy of hypervisor log events: starting with events at discrete times that were retrieved from the hypervisor FTP log server sources.

II. Background and Related Work

As mentioned earlier, the current set of related tools include the Encase tool [6], the Forensic Toolkit [2] and the Sleuth Kit [4].

The main functionality of the Encase tool [6], from Guidance Software, lies in the retrieval of data from a physical system to locate specific data easily. Encase provides the functionality to sort file information by various fields, which includes timestamps. It is also possible to retrieve and search within log files, such as system logs, and log data from security software and application logs [6]. It should be noted that there is no intent to use VMLA at this time to combine data from different sources across time zones as one cannot synchronize NTP servers for which you as the administrator or investigator have no control over the temporal or logical ordering of the log evidence collected [14].

Access Data's Forensic Toolkit (FTK) [2] focuses primarily on securing information from an operating system and then providing the ability to locate and examine specific files. Files can be sorted by their attributes, including the file timestamp values. There are extensive search capabilities, as well as a large number of known file formats whose contents can be displayed and searched using FTK.

Brian Carrier's Sleuth Kit [4] also has the ability to view file system events. The focus lies primarily in the recovery of the information as opposed to its analysis. Basic timelines of the file system events are generated with the mactime tool, which generates a sorted list of modification, access, and creation (MAC) timestamp hypervisor log events. One approach the investigator can use to organize events with the Autopsy forensic browser is to generate annotated bookmarks for events. At this time, it is not possible to group the hypervisor log events into a detailed hierarchy and perform complex searches on them.

The FileList Pro tool from New Technologies [9] can create timelines of file activity on a system. A user can choose to sort the files and the information associated with them by various timestamps. Timelines can be created for file access, creation, modification, and deletion for DOS and Windows systems. A user is not able to use the tool to introduce log events from other unknown sources or group events together outside of the test LAN.

The nFX open Security Platform from netForensics [8] provides a mechanism to gather event information from various sources of a distributed network. The events are normalized and synchronized and displayed real-time for intrusion detection purposes. Furthermore, statistical analysis techniques may be utilized for event correlation. The events are gathered via agents that need to be installed on the systems from which to gather data. There are a large number of devices supported directly by the product, and a \Quick Connect" feature provides the ability for custom agents for other data sources.

Given that agents need to be installed and active on all systems that are monitored, the usefulness for nFX for forensic purposes is limited. When the platform is deployed on a system that needs to be investigated it may well be used for a forensic investigation. It is not possible to group events into hierarchies to graphically build a timeline of events. Also, only those events are captured through some kind of logging or reporting mechanism by the system or application are accessible. Events from MAC times for example cannot be captured in this fashion.

III. The Design of the VMLA

The VMLA is an open-sourced graphical tool written in Java that allows the VM forensic investigator to import various hypervisor log event files from the VM host operating system. The logs are then ordered and classified into one or more timelines of events. The organization of events and timelines are hierarchical tree views that allow the investigator to display and hide specific VM log events. This is intended to support relevant aspects of the cloud investigation one at a time. This is further supported by the capability to filter VM hypervisor log events based on start and ending times. These VM log events serve as a unified data structure to bring together potential VM log forensic evidence from different OS sources. It is now possible to combine data from those sources and analyze them within a single framework. Given the Exabyte and Zettabyte storage limits of the production SAN, the VMLA by assumption is only engaged in snapshot hypervisor log evidence compilation as the preferred data reduction method [10].

The design objectives for VMLA are as follows:

- Arbitrary Generation of VM log events from the specified local VM host operating systems data sources
- Enabling the grouping of hypervisor log events together into logical groups recursively
- Seamless filtering of the VM log data that is displayed
- Enabling the easy location of specific VM log events
- Providing an intuitive interface
- Being a platform independent implementation

The development of the VMLA was punctuated with several design challenges. Among them are fast and efficient data structures, efficient importing of arbitrary VM logs, and intuitive user interface design.

A. The VM Log Event data Structure

Choosing an appropriate data structure was our first hurdle. The Java Development Kit (JDK) in version 1.4 offers several data structure classes. As the anticipated size of the data set was in the hundreds of thousands of events, the ability for a rapid search and retrieval of data items is important. Also, the project assumed that events could be added to the data structure at ten thousand to one hundred thousand at a time. This necessitated a quick build time of the data structure. Given that that the log events needed to be sorted, it was decided that any data structure would require a run time of O(log(n)) for any of its operation (lookup, add, remove), which would result in an overall build-time of O(n log n). JDK 1.5 offers a TreeSet class, which is an implementation of a balanced search tree. Due to the mappings required for a custom TreeModel class for the JTree GUI component, one of the lookup operations needed could only be run in O(n) time, which resulted in a build-time of $O(n^3)$ due to meta layer abstraction run time overheads.

To implement a custom data structure that utilizes Adelson-Velskii and Landis (AVL) trees [2], a variant of a balanced binary search tree was adapted. The main data structure of the VMLA is the Log event. There are two types of log events in the VMLA: atomic events (primitive) are the events that are directly imported from the host OS system i.e. File MAC times, logs, etc. VM Complex log events are events that are comprised of atomic events or other complex events, that is they act as a container for other VM log events from which it derives some of its properties. There is an abstract class TimeEvent, which defines common fields and methods of the two kinds of events, such as the start time, name, description, and parent event fields and methods to retrieve them. The class AtomicEvent further adds a reference to the source of the event. The source contains information about from where the event was imported and what the time granularity is.

The class ComplexEvent adds fields for an end time and its child log events. The start time of a complex log event is defined as the smallest start time among its children, while the end time is the largest end time of the children (for atomic events, the end time can be considered the same as the start time). That is, whenever child log events are added or removed from the complex log event its start and end times potentially change. Note that a VM complex log event does not have a source associated with it. Instead, the sources of its children define its "\source" from the corresponding SAN disk cluster. The child events of a complex event are organized in a balanced binary AVL search tree. The sorting key for the children is their start time combined with a unique identifier so that the events are sorted by their start time. Events that have the same start time and are siblings are sorted in an arbitrary fashion (actually, an event that was created earlier is considered "\smaller" because its unique identifier will be smaller. Complex events that are children of a VM complex log event in turn may contain children, which are also organized in the AVL tree, and so on. Figure 1 depicts the structure of a VM log complex event.

Thus a VM log timeline is nothing but a VM log complex event as a root that contains the hierarchy of VM log events with its children. VMLA uses a subclass of the java.swingx.JTree class. Here is a summary treatment of the behavior of the VM Complex log Event class container properties. The structure of a VM complex log event place timelines in a tree view. Complex events are collapsible nodes, whereas atomic events are the leaf nodes. This is analogous to a file systems browser with directories and files. A complex event must at least contain one child, which means that at the lowest level of the hierarchy there must be an atomic event. The exception is an empty timeline, where the complex event serves as the root node of the tree view.

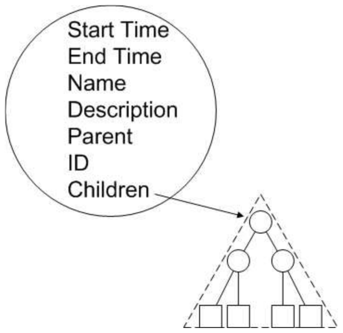


Figure 1. Hierarchial Structure of a VM hypervisor Log Complex Event

B. Importing VM Log Events

One of the essential features of the VMLA is its ability to generate log events from multiple physical OS data source. This is similar to the \Quick Connect" feature of the nFX Open Security Platform [9], but given that data is not gathered in real-time, one has the ability to gather more types of data, such as file MAC times or other information that cannot be actively monitored by agents installed on a VM host system. VMLA's import capabilities can be dynamically extended by supplying a Java class that implements our VMLog InputFilter interface and generates log events from the desired data source. The fact that VMLA dynamically loads the VM log input filter classes at start-up means that a user can extend the functionality without having to re-compile the VMLA classes. It is sufficient to compile the new input filter class and put it into a special directory.

The capability to create your own input filters to generate VM log events is one of the most desirable features of the VMLA. A brief discussion of the Java InputFilter class that extends this functionality to VMLA is discussed here. The use of VMLogInputFilter class was adapted to implement an input filter. The interface requires the following methods:

```
public abstract class VMLogInputFilter {
public abstract Source init(String location, Component
parent);
public abstract AtomicEvent getNextEvent();
public abstract FileFilter getFileFilter();
public abstract String getName();
public abstract String getDescription();
public abstract long getExactCount();
public abstract long getTotalCount();
public abstract long getProcessedCount();
}
```

Classes that implement the InputFilter interface must provide an implementation for the required methods. init() will initialize the data source (e.g. open a file and gather information about the source) and return an object describing the source. getNextEvent() will return the next event that comes from an initialized source. If there is no next event, then null is returned. This allows loops such as:

```
while ((event = filter.getNextEvent()) != null){
/* process the event */
}
```

The getFileFilter method tells the tool how (and if) to filter files or file names when the file chooser dialog is opened (e.g. *.txt" for text files). It may return null if no such filtering is desired.

The getName() and getDescription methods return a name and a description for the filter, respectively. The name will appear in the filter type selection when importing, while the description will be used in future versions to give the user more feedback when selecting a filter.

The three methods getExactCount(), getTotalCount(), and getProcessedCount() are used for progress bar updates. If the filter knows how many events will be generated, then it returns that number as the exact count. Otherwise, the exact count should return 0 and the total count can be returned, which represents the amount of data that is processed (number of lines or bytes, for example).

The methodgetProcessedCount() will then return the amount of data (of the total count) that has already been processed by the filter during the import process. The next step is to explain how the VMLogInputFilter class works, which may serve as a proof-of-concept for other filter classes.
The VMLogInputFilter class processes data similar to the fls tool used by Sleuth Kit [4]. It is required that the VMLog output to be in machine text readable format. The init method is fairly straightforward. The attempt to open the input stream is specified by the file name and onsuccess which returns a new Source object or null otherwise:

```
public Source init(String filename) {
try {
  file_input = new BufferedReader(
  new FileReader(filename));
}
   catch (IOException ioe) {return null;}
return new Source("Log filter",
filename,Source.GRANULARITY_SEC);
}
```

Each line of the log output can create between one and three separate events, depending on whether the timestamp are the same or differ. For this, one adopts the FIFO queue called logevent queue in which the extra log events resulting from the processing of the line to be read in subsequent getNextEvent() calls. Thus the algorithm for processing lines and returning events is as follows:

- If logevent queue is empty, read the next line from the input. Else dequeue and return the next logevent.
- Return null is end of file is reached.
- Process the line and compare the log timestamps.
- If more than one log event is created, queue all but one.
- Return the remaining logevent.

The following is pseudo-Java code of the getNextEvent() method, glossing over unimportant parts:

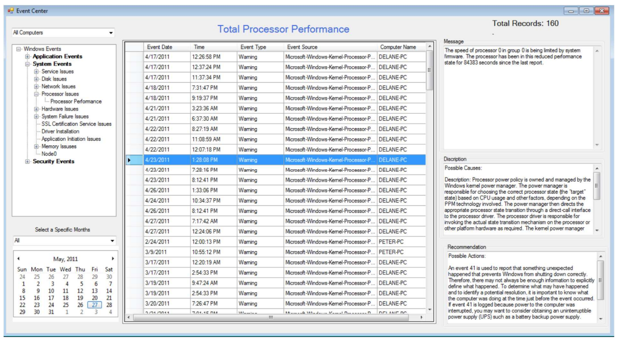
```
public AtomicEvent getNextEvent() {
if (event_queue.isEmpty()) {
read the line from the input stream
if (line == null) return null;
fields = line.split("\\|");
// get timestamps, one has second granularity
// but need to convert to ms
long mtime =
Long.decode(fields[12]).intValue()*1000;
long atime =
Long.decode(fields[11]).intValue()*1000;
long ctime =
Long.decode(fields[13]).intValue()*1000;
String name = fields[1];
String description = "User: " + fields[7] +
"\n" + "Group: " + fields[8] + "\n" +
"Mode: " + fields[5];
if ((mtime == atime) && (mtime == ctime))
return new AtomicEvent("MAC " + name,
description, new Timestamp(mtime));
if (mtime == atime) {
event_queue.add(
new AtomicEvent("MA. " + name,
description, new Timestamp(mtime)));
return new AtomicEvent("..C " + name,
description, new Timestamp(ctime));
}
/* and so on for all the MAC combinations ...*/
}
else
return (AtomicEvent) event_queue.removeFirst();
}
```

The progress bar methods are implemented as follows: because one does not have an exact count of the number of VM log events that will be generated when one initializes the filter, getExactCount() returns 0. The total count is simply the size of the log file system in bytes, and the processed count the byte position of the open file handle:

```
public long getExactCount() {
return 0;
}
public long getTotalCount() {
return file_input.length();
}
public long getProcessedCount() {
return file_input.getFilePointer();
}
```

C. The VM Graphical User Interface

Java offers several options for producing GUIs. Among them are the Abstract Windows Toolkit (AWT) and more recent Swing classes. Briefly considered was the Eclipse project, whose Simple Window Toolkit (SWT) allows Java GUI functionality to be handled by OS-native APIs. Eventually Eclipse was discontinued from consideration because of its poor performance when building a Tree object. Swing proved to be the optimum choice for GUI design because of its native GUI objects such as the tree list, an object ideally suited for the hierarchical display of events such as those dealt with in VMLA. A conscious thought while designing the GUI was to make it as easily understandable as possible. Because the general audience of the program may include law enforcement professionals with little prior background in computers, the program should be as simple as possible to approach and understand. With this thinking in mind the GUI was constructed to imitate the functionality of other pervasive applications, such as Microsoft's Windows Explorer, so as to take advantage of the user's innate understanding of user interface functions.



Event Date	Time	Event Type	Event Source	Computer Name
4/17/2011	12:26:58 PM	Warning	Microsoft-Windows-Kernel-Processor-P...	DELANE-PC
4/17/2011	12:37:24 PM	Warning	Microsoft-Windows-Kernel-Processor-P...	DELANE-PC
4/17/2011	11:37:34 PM	Warning	Microsoft-Windows-Kernel-Processor-P...	DELANE-PC
4/18/2011	7:31:47 PM	Warning	Microsoft-Windows-Kernel-Processor-P...	DELANE-PC
4/18/2011	9:19:37 PM	Warning	Microsoft-Windows-Kernel-Processor-P...	DELANE-PC
4/21/2011	3:23:36 AM	Warning	Microsoft-Windows-Kernel-Processor-P...	DELANE-PC
4/21/2011	6:37:30 AM	Warning	Microsoft-Windows-Kernel-Processor-P...	DELANE-PC
4/22/2011	8:27:19 AM	Warning	Microsoft-Windows-Kernel-Processor-P...	DELANE-PC
4/22/2011	11:08:59 AM	Warning	Microsoft-Windows-Kernel-Processor-P...	DELANE-PC
4/22/2011	12:07:18 PM	Warning	Microsoft-Windows-Kernel-Processor-P...	DELANE-PC
4/23/2011	1:28:08 PM	Warning	Microsoft-Windows-Kernel-Processor-P...	DELANE-PC
4/23/2011	7:28:16 PM	Warning	Microsoft-Windows-Kernel-Processor-P...	DELANE-PC
4/23/2011	8:12:41 PM	Warning	Microsoft-Windows-Kernel-Processor-P...	DELANE-PC
4/26/2011	1:33:06 PM	Warning	Microsoft-Windows-Kernel-Processor-P...	DELANE-PC
4/24/2011	10:34:37 PM	Warning	Microsoft-Windows-Kernel-Processor-P...	DELANE-PC
4/26/2011	8:12:41 PM	Warning	Microsoft-Windows-Kernel-Processor-P...	DELANE-PC
4/27/2011	7:17:42 AM	Warning	Microsoft-Windows-Kernel-Processor-P...	DELANE-PC
4/27/2011	12:24:06 PM	Warning	Microsoft-Windows-Kernel-Processor-P...	DELANE-PC
2/24/2011	12:00:13 PM	Warning	Microsoft-Windows-Kernel-Processor-P...	PETER-PC
3/9/2011	10:55:12 PM	Warning	Microsoft-Windows-Kernel-Processor-P...	PETER-PC
3/17/2011	12:20:19 AM	Warning	Microsoft-Windows-Kernel-Processor-P...	DELANE-PC
3/17/2011	2:54:33 PM	Warning	Microsoft-Windows-Kernel-Processor-P...	DELANE-PC
3/19/2011	9:47:24 AM	Warning	Microsoft-Windows-Kernel-Processor-P...	DELANE-PC
3/19/2011	2:54:33 PM	Warning	Microsoft-Windows-Kernel-Processor-P...	DELANE-PC
3/20/2011	7:26:47 PM	Warning	Microsoft-Windows-Kernel-Processor-P...	DELANE-PC

Message

The speed of processor 0 in group 0 is being limited by system firmware. The processor has been in this reduced performance state for 84383 seconds since the last report.

Discription

Possible Causes:

Description: Processor power policy is owned and managed by the Windows kernel power manager. The power manager is responsible for choosing the correct processor state (the "target" state) based on CPU usage and other factors, depending on the PPM technology involved. The power manager then directs the appropriate processor state transition through a direct-call interface to the processor driver. The processor driver is responsible for invoking the actual state transition mechanism on the processor or other platform hardware as required. The kernel power manager

Recommendation

Possible Actions:

An event 41 is used to report that something unexpected happened that prevents Windows from shutting down correctly. Therefore, there may not always be enough information to explicitly define what happened. To determine what may have happened and to identify a potential resolution, it is important to know what the computer was doing at the time just before the event occurred. If event 41 is logged because power to the computer was interrupted, you may want to consider obtaining an uninterruptible power supply (UPS) such as a battery backup power supply.

Figure 2. VMLA Graphical hierarchial tree log event timeline window

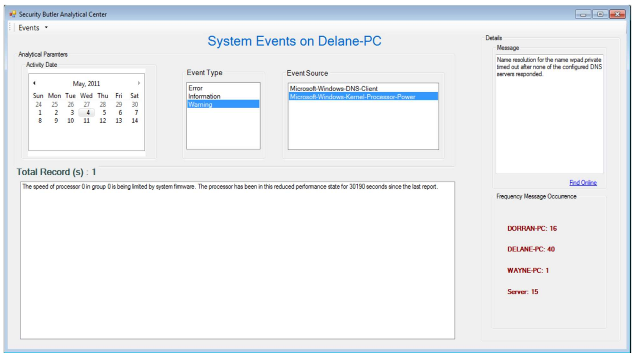
The timeline on the right represents the unmodified hypervisor timestamp logs retrieved from the local VM ftp server along with descriptions of the log event activity as adopted from the VM host OS source. The timeline on the left contains a drop down menu selection log events list (i.e. Application, System, Security, and Error Logs). In the left lower panel, the current system access date for these evidence target server hypervisor logs.

IV. Features of the VMLA

The initial prototype of VMLA that is described in this paper concentrates on a small set of basic features that are important when constructing a timeline of log events. They can be divided into three categories: managing log events via the GUI, maintaining the integrity of VM log digital evidence, and being able to quickly locate log events.

A. Managing Events

Being able to manage log events efficiently is the most important feature of VMLA. After hypervisor log events from one source have been imported into a single timeline, a user has several options to group them into complex events. New complex events can only be created from a selection of other events (atomic and/or complex): the selected events are moved into a newly created complex event, which, in turn, is inserted into the timeline at the parent of the node(s) highest in the hierarchy among the selected events. Once a complex event exists, events can be transferred to and from it via drag and drop. This can be within the same timeline or between two timelines. Furthermore, events may be transferred using cut and paste actions. Nodes in the tree that represent a complex event may be expanded or collapsed, allowing the user display and hide information as needed.
A user can also create empty timelines and then populate it with log events, or s/he can create a new timeline from selected events of another one. This way, a user can make a hypothesis and then look for log events supporting it while at the same time building the timeline for it. Timelines are displayed either within a single JTabbedPane or in two such panes next to each other. The arrangement of timelines in tabbed panes allows the user to easily switch view between timelines. The double-pane model lets the user view timelines next to each other and a drill down functionality to analyze a hypervisor log event based on the OS machine source user as seen in Figure 3.0. This is especially beneficial when constructing a timeline of events from different sources: one side is used to construct the VM log event hierarchy, and the other side to search for the relevant VM log events supporting the timeline under construction.
This mode is also ideal for moving these VM hypervisor log events between timelines via drag and drop. The single-panel mode offers more viewing space for the tree view and may be helpful when grouping log events within the same timeline.


Security Butler Analytical Center
Events
System Events on Delane-PC
Analytical Paramters
Activity Date
May, 2011
Event Type
Error
Information
Warning
Event Source
Microsoft-Windows-DNS-Client
Microsoft-Windows-Kernel-Processor-Power
Details
Find Online

Total Record (s) : 1

The speed of processor 0 in group 0 is being limited by system firmware. The processor has been in this reduced performance state for 30190 seconds since the last report.

Message

Name resolution for the name wpad.private timed out after none of the configured DNS servers responded.

Frequency Message Occurrence

DORRAN-PC: 16

DELANE-PC: 40

WAYNE-PC: 1

Server: 15

Figure 3. Hypervisor Log Analytical center

B. Maintaining Forensics Integrity

The final design challenge faced by VMLA is it's conformance to the special needs as potential cloud digital forensics software. Data utilized by forensic programs are often presented in a court of law. As a result of this usage, the handling of data by the software is subject to certain restrictions to prevent unauthorized manipulation. One specific example of this is how a single source of information such as output from the VM Host OS hypervisor log provides thousands of megabytes of potential data evidence. An unscrupulous investigator could remove from the data set biased log evidence.
As the developer the approach was to have the VMLA program use every single event from a log source file or none at all. An investigator would not be allowed to delete from the project any log events. The solution to this was to create an unseen "orphan" log timeline that contains any "deleted" log events. The log events themselves are not deleted from the project but merely shifted into the orphan log timeline. In this way the log events can be moved out of the field of view but without compromising the integrity of the log source of evidence.
Edit functionality also affects the source integrity of a data set. Take for example a log event being cut and never pasted into a timeline. If this happens the program closes and the VM log event stored in the cut buffer is lost. In order to avoid this, special care is taken when the program is saving to first dump the contents of the cut buffer into the orphan log timeline before writing out the data.

C. Queries

VMLA uses Log query objects to match against VM hypervisor log events. Queries can then be used to filter and locate events. During filtering only those log events that match the query are displayed. When locating log events the (first) event(s) that match the query are displayed in their current context.

Currently, queries only support a limited keyword search in combination with a time interval in which the VM log events must occur. The keyword search allows the use of regular expressions as supported with the Java's String.matches() method to determine a match. The keyword is initially padded with wildcard (.*) matchers at the start and end, but the keyword itself can contain Java regular expression characters. This also means that certain characters have to be escaped for a literal match.
All the logic needed to perform query matches is contained in the Query class. More sophisticated types of queries may thus be easily added to VMLA by either modifying the Query class, or by sub-classing it and overriding its matches() method. Some of the query features that are planned for future releases are: search by VM Log event source, end time, and type (once a typing system for VM Log events exists).

V. Using the VMLA

To date the VMLA has been used to detect temporal hypervisor log inconsistencies within the timelines of a digital investigation as supported by the University research environment of the authors. Based on these inconsistencies inference methods have been developed to this end to evaluate these inconsistencies. The experimental rules and results can be further explored in a recent journal article [10]. In a second article [13], the VMLA is used to support incident analysis within the SAN, by providing evaluations on how to formulate hypothesis with the log file stamps for a potential cloud investigation.

VI. Conclusions and Future Work

The virtual machine log auditor is a prototype tool that allows the forensic investigator to import VM hypervisor log events from distinct OS sources, and let him group these log events together into complex log events. This grouping supports a hierarchy of VM log events. The primary goal of this tool is to assist the investigator in creating hypothesis about what VM events took place on the physical OS by way of these hypervisor sources.

For future work the plan is to introduce a machine learning step that intelligently filters out unimportant events reducing the amount of log events that need to be kept in the GUI data structures and in main memory. Other expected prototype improvements include indexing on hypervisor log searches, and usability task improvements.

References

[1] P. Mell and T.Grance. “NIST Definition of Cloud Computing V15”, July 10, 2009. Retrieved from http://csrc.nist.gov/groups/SNS/cloud-computing/

[2] AccessData Corp. Forensic Toolit. http: //www.accessdata.com/Product04_Overview.htm.

[3] G.M. Adelson-Velskii and E.M. Landis. An algorithm for the organization of information. Dokladi Akademia Nauk SSSR, 146(2):1259{1262, 1962.

[4] Brian Carrier. Sleuthkit and Autopsy forensic browser. http://www.sleuthkit.org.

[5] Guidance Software. Encase forensic software. http://www.guidancesoftware.com.

[6] Guidance Software. EnCase Enterprise Edition Detailed Product Description. http://www.guidancesoftware.com/corporate/whitepapers/downloads/EEEDeta%iled.pdf, July2004.

[7] HoneyNet Project. HoneyNet Scan 15. http://www.honeynet.org/scans/scan15, May 2001.

[8] netForensics Inc. nFX Open Security Platform http://www.netforensics.com/nfxosp.asp.

[9] New Technologies Inc. FileList Pro Computer Timeline Software. http://www.forensics-intl.com/filelist.html

[10] Sean Thorpe , Indrajit Ray. “*File Timestamps in a Cloud Digital Investigation.*” To Appear in the Journal of Information Assurance and Security , Vol 7, March 2012 issue.

[11] Sean Thorpe, Indrajit Ray, Tyrone Grandison. “Enforcing Data Quality Rules for a Synchronized VM Log Audit Environment using Transformation Mapping Techniques". The *Proceedings of the 4th Intl Conference on Computational Intelligence in Security for Information Systems, Torremolinos, Malaga, Spain.* June 8-10, 2011.

[12] Sean Thorpe, Indrajit Ray , Tyrone Grandison. "Use of Schema Associative Mapping for synchronization of the Virtual Machine Audit Logs". The *Proceedings of the 4th International Conference on Computational Intelligence in Security for Information Systems, Torremolinos, Malaga, Spain. June 8-10, 2011.*

[13] Sean Thorpe , Indrajit Ray. “Detecting Temporal Inconsistency in Virtual Machine Activity Timelines.” To Appear in the Journal of Information Assurance and Security, Vol 7, March 2012 issue.

[14] Sean Thorpe, Indrajit Ray , Indrakshi Ray , Tyrone Grandison. Towards a Formal Temporal Log Model for the Virtual Machine synchronized cloud computing environment.” Proceedings of the Journal of Information Assurance and Security, Volume 6, No 2. 2011, March 2011.

Author Biographies

Sean Thorpe holds an M.S. and B.S. degrees in Information Security and Computer Science respectively from the University of Westminster, London, UK in November 2002 and from the University of the West Indies, Mona Campus Jamaica in November 2000. He joined the University of Technology (UTECH) as a Lecturer in January 2003 with responsibility for teaching System Security at the undergraduate level. Mr. Thorpe has worked extensively in the IT industry since 1995 as a System Programmer Analyst and Oracle DBA before joining academia. He is a 2009 recipient of the Fulbright Visiting faculty Scholarship award to Harvard University, where he explored collaborative research work in the area of Security Metrics. He is also the 2009 winner of the OOPSLA Educational Symposium Award for his innovative computer science teaching methods, and the recent 2011American National Science Foundation (NSF) awardee for Caribbean based research in the area of Cloud Computing. His specific research interest includes cloud forensics, and security policies. In 2010 he started his PhD in Computer Science.

Indrajit Ray is an Associate Professor at Colorado State University since 2002. Prior he was an Assistant Professor at the University of Michigan Melbourne from 1997 to 2001. He earned his PhD from George Mason University, Virginia in summer 1997. He obtained his BSc Computer Science Degree from Bengal Institute in India in 1984 and then his MSc from Jadvapur University in 1991, also in India. His primary research interest is digital forensics, security policies, access controls, and intrusion detection.

Tyrone Grandison is a Research Staff Member at the Thomas J. Watson Research Center. He received a B.S. degree in Computer Studies (Computer Science and Economics) from the University of the West Indies in 1997, a M.S. degree in Software Engineering in 1998 and a Ph.D. degree in Computer Science from the Imperial College of Science, Technology & Medicine in London. He then joined IBM at the Almaden Research Center in California, where he worked on data privacy and security. In 2010, he joined the Global Healthcare Transformation team as Program Manager for Core Healthcare Services. Dr. Grandison is a Distinguished Engineer of the Association of Computing Machinery (ACM), a Senior Member of the Institute of Electrical and Electronics Engineers (IEEE), a Fellow of the British Computer Society (BCS), has been recognized by the National Society of Black Engineers (as Pioneer of the Year in 2009), the Black Engineer of the Year Award Board (as Modern Day Technology Leader in 2009 and Minority in Science Trailblazer in 2010) and has received the IEEE Technical Achievement Award in 2010 for "pioneering contributions to Secure and Private Data Management". He has authored over 70 technical papers and co-invented over 20 patents.

Abbie Barbir holds a PhD from Arizona State University since 1991. He currently heads the OASIS Security Study group 17 responsible for policy formations and standards for open access environments. He has significant industry experience within the telecomm sector including Nortel and the ITU. He is currently a consultant to Bank of America.