Hypervisor Event Logs as a Source of Consistent Virtual Machine Evidence for Forensic Cloud Investigations

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

[1] Faculty of Engineering & Computing, University of Technology, Kingston, Jamaica,
thorpe.sean@gmail.com
[2] Department of Computer Science, Colorado State University, Fort Collins, USA
indrajit@cs.colostate.edu, france@cs.colostate.edu
[3] Proficiency Labs Intl, Ashland, Oregon
tgrandison@proficiencylabs.com
[4] Bank of America,
abbie.barbir@bankofamerica.com

Abstract. Cloud Computing is an emerging model of computing where users can leverage the computing infrastructure as a service stack or commodity. The security and privacy concerns of this infrastructure arising from the large co-location of tenants are, however, significant and pose considerable challenges in its widespread deployment. The current work addresses one aspect of the security problem by facilitating forensic investigations to determine if these virtual tenant spaces were maliciously violated by other tenants. It presents the design, application and limitations of a software prototype called the Virtual Machine (VM) Log Auditor that helps in detecting inconsistencies within the activity timelines for a VM history. A discussion on modeling a consistent approach is also provided.

1 Introduction

Temporal event logs meticulously record events that have occurred in the history of the computer system, and therefore, constitute a valuable source of digital evidence. Event logs are generated by the operating system as well as by other subsystems and their applications. There has been a significant amount of research about the examination and the auditing of such logs for post incident forensic purposes [1, 2]. With cloud computing service environments, the expectations for post incident forensics is no different. Cloud computing is predicated on the well known service oriented architecture (SOA) and harnesses the power of the virtualization stack. The known services offered across the virtual stack layers are Infrastructure as Service (IAAS), Platform as a Service (PAAS) and Software as a Service (SAAS). In our work the PAAS provision, which handles the hypervisor logs, is our primary concern. By studying the hypervisor logs, this work represents the first body of work in the literature that seeks to explore closing the semantic

gap of how eye witness forensic data that can be collected between the lower layers and higher virtualization layers and is motivated by prior work [1, 2, 21].

In this work the log categorization used is the hypervisor event logs, which contains a hierarchy of application logs, security logs, error logs and system logs. Some have likened a log file in computer forensics to an eyewitness in a traditional crime scene [1]. The analogy seems befitting when using the hypervisor logs to manage virtual crime scenes for which the multi-tenant VMs are all co-located.

Providing a picture of what happened through the analysis of the hypervisor event logs at a virtual crime scene is no trivial task. Detecting patterns that may be events of interest by hand in a large log file is not feasible. Managing the sheer volume of log data is a well known quantity problem. Logs must be parsed programmatically and even this can take a very long time, with the exact amount of time varying significantly, and is often dependent on the type of algorithm employed to detect the patterns of interest [2].

As log data are explicitly a record of events, establishing their reliability is of particular importance. Log files are written to very frequently and hence may get corrupted or could be difficult to understand; as records may be saved in an unexpected sequence as a result of unusual system behavior, e.g. software bug or power outage. As log files are an obvious record of events, they are also an obvious target for tampering. Suspicious events indicating that something is wrong may be deliberately removed, rendering all or a part of the log potentially fraudulent.

Our work addresses the issue of deliberate tampering, internal contradictions and inconsistent entries with these hypervisor event logs within the storage area network (SAN) data centre environments. We attempt to improve on the state of the art by providing a forensic platform that transparently monitors and records data access events using these PAAS logs as a form of static snapshot state analysis for a post incident VM cloud investigation. This approach complements the traditional statistical trace analysis methods and the VM memory introspection methods established in prior work [21, 22, 25]. As it relates to using the PAAS logs to detect VM attacks, particularly session hijacking, this represents independent ongoing work using several data mining techniques [23, 24] that unearth ground truth forensic evidence based on anomalous patterns detected from such logs. As it is now, the timestamps recorded in the hypervisor event logs may be unreliable, as result of both flaws in the clocks that generate them and the nature of the software that records entries to these data cloud logs.

The rest of this paper is organized as follows. Section 2 introduces our VM profiling model. Section 3 examines approaches for the detection of inconsistency in timelines, dealing both with inconsistencies in VM event timestamps and VM events omitted from the hypervisor kernel system's record. Section 4 describes the experiments with the tool for testing the approaches discussed in Section 3. Section 5 describes the rule base for the experiments and evaluates the detection techniques based on the rule base. Section 6 provides a formal discussion on constructing the consistency approach, and Section 7 provides the discussion of the experimental results. Section 8 presents a discussion of future work, Section 9 provides the related work and we conclude in Section 10.

2 Virtual Machine Profiling Model

This work is based on concepts from the computer-profiling model described by Marrington et al. [9]. The authors' model of a computer system consists of objects representing the various entities that form part of the computer system's operation. These entities include users, data files, system software, hardware devices, and applications. The objects discovered on the computer system under examination, collectively referred to as the set ***O***, are classified according to their type. In their model [9], there are four broad types of objects (Application, Principal, Content and System) with increasingly specific subtypes.

We adopt in our work each of these categories as VM sets. The set of Application objects, ***A***, consists of all the application software on the VM host computer system. The set of Principal objects, ***P***, consists of all the computer system's users and groups, and all of the people and organizations otherwise discovered in the examination of the computer system. Of these objects, some Principal objects are described as *canonical* if they represent definite entities on the computer system that are actors in their own right, such as users and groups. Principal objects may be described as *non-canonical* if they represent people or groups of people who may not be actors on the system, but may be, for instance, people mentioned in documents. The set of Content objects, ***C***, consists of all the documents, images and other data files on the host computer system from which the VMs are running. The set of VM System objects, ***S***, consists of all the VM configuration information, system software and hardware devices on the VM host computer system. ***A***, ***S***, ***C***, and ***P*** are subsets of ***O***, the set of objects on the cloud system under investigation.

We characterize our model with the set of all times in the history of the VM host computer system, $\boldsymbol{T}_v$, and the set of all events, ***Evt***, which have taken place in the history of the VM host computer system. Let t be a time in $\boldsymbol{T}_v$, x be the VM object that triggered the VM event, y be the object that was the target of the event, ε describe the action of the event, and α describe the result of the event (successful, unsuccessful, or unknown). An event evt in the set ***Evt*** consists of the quintuple $evt = \left(t, x \in O, y \in O, \varepsilon, \alpha\right)$.

This same finite set ***Evt*** consists of two enumerable subsets, and one subset which cannot be enumerated. The first subset – the recorded events set ***EvtR*** – consists of VM events that are recorded in the VM host computer hypervisor system's logs. The second subset – the inferred events set ***EvtI*** – consists of events that are not recorded in logs, but that can be inferred on the basis of other digital evidence on the system, such as relationships between different objects. These two sets do not necessarily describe the complete history of the cloud system in an exhaustive manner. There may be other events that took place and were unrecorded and left no artifact from which they could be inferred. These VM events are obviously unknown, and comprise the final subset of ***Evt***.

The set ***EvtI*** is particularly vulnerable to inconsistency or incompleteness in the data obtained from the VM target computer's file system. Contradictory, inaccurate or missing information can lead to an incomplete timeline of a user's activity. ***EvtR*** is a direct

representation of the contents of the VM host hypervisor system's logs, and consequently, incorporates any inaccurate event records in the system logs. Further, if a VM event is not logged, and cannot be inferred, it will not be an element of either ***EvtR*** or ***EvtI***. Handling unknown events within the VM history of the computer system is a challenge and hence the less accurate the timeline of the target computer's activity will be. We address this challenge in an independent paper. For this work, we focus on the declared events and making the inferences from these stated events. This work provides a means for the semi-automated detection of inaccuracy or incompleteness leading to chronological inconsistency in timelines of VM computer activity.

Marrington et al. [6], discussed a timestamp-based technique for building a timeline about a given object in the profile of the computer system. However, their approach is not resilient to inaccuracies in timestamps, which may cause VM events to appear out of sequence. Missing events, whether removed manually or simply never recorded, lead to timelines that may present events out of the context they actually occurred. We posit that as a general principle, the failure to detect an inconsistency in a timeline is a greater problem for the purposes of VM activity time-lining than falsely identifying an event as inconsistent. This is simply because false positives can be manually investigated and dismissed, whereas false negatives will never receive further attention. Nevertheless, it is obviously desirable to minimize the rate of false positives in all detection techniques.

A limitation of any time-lining activity based on timestamps provided by a computer's system clock is the inaccuracy inherent in such clocks. This inaccuracy in computer-generated timestamps is normal, and the solution suggested most frequently in the literature is to note the system clock time of a computer under investigation at the time of its examination and to determine the discrepancy between that time and the time of a reference clock [1, 8]. However, this solution does not address the issue of clock skew. A few works, notably [2, 10], propose algebra for the formal expression of falsifiable hypotheses about the discrepancy between a computer's clock and physical time. The term proposed for such a phenomenon is a clock hypothesis. In practice, it would be necessary to form a clock hypothesis for every moment in time throughout the history of the VM host system. Our tool is intended to detect internal inconsistency in timelines. An investigator could potentially be assisted in the formation of VM clock hypotheses using the output of our tool.

3 VM Log Auditor for Timelines

We now describe the approaches employed by our log auditor software to detect inconsistency in timelines. Inconsistency in virtual machine computer activity timelines can arise because hypervisor kernel log events in the timeline are out of sequence, or VM events that should be in the timeline are missing.

3.1 Detecting out of Sequence VM Timelines

There are some VM events that need to occur before some other event can happen. This sort of relation between events is described as the *happened-before* relation [4]. Gladyshev and Patel [5] discuss the application of the *happened-before* relation to a forensic context. An example of such a relation between two events would be that a VM user x must "login" successfully to the computer system before the user x can "execute" the application y. So the *happened-before* ($\rightarrow$) relation implies that in the VM activity timeline, the time of the VM login event must be before the time of the execution event. We express this as follows. Let $x \in \boldsymbol{P}$, $y \in \boldsymbol{A}$, and $t_m, t_n \in \boldsymbol{T}_y$. Then (($t_m$, x, y, login, success) $\rightarrow$ (t_n , x, VM system, execution , success)) $\Rightarrow$ ($t_m > t_n$), where $\Rightarrow$ is the logical implication operator . Note that the *happened-before* relation is transitive [4, 5]:

After the construction of a VM log timeline (which is a sequence over the set ***Evt***) in the log auditor's execution process, an evaluation can be applied to all VM events ordered by their timestamp. If a VM event $vmevt_a$ has a *happened-before* relation to $vmevt_b$, but the VM kernel log timestamp (t_b) of $vmevt_b$ suggests that $vmevt_b$ occurred before $vmevt_a$ then we can say that t_a and t_b are inconsistent. In order to detect this inconsistency, a rule base must be created which describes the *happened-before* relations for the various types of events [15]. When the VM timeline is evaluated against the rules base, the inconsistent events can be identified and assertions about their time stamps can be made.

Consider the two rules $vmevt_a \rightarrow vmevt_b$ and $vmevt_b \rightarrow vmevt_c$ with

$$vmevt_a = \left(t_a \in T_y,\ x,\ system,\ \text{login}, \text{success}\right)$$

$$vmevt_b = \left(t_b \in T_y,\ x,\ a,\ \text{execute},\ \alpha \in \{\text{success}, \text{fail}, \text{unknown}\}\right)$$

$$vmevt_c = \left(t_c \in T_y,\ x,\ system,\ \text{logout}, \text{success}\right)$$

where x is a User VM object, a is an Application VM objects, and *system* is a VM System object representing the target VM computer system itself. Then, by the transitivity property of the *happened-before* relation,

$$\left(\left(vmevt_a \rightarrow vmevt_b\right) \wedge \left(vmevt_b \rightarrow vmevt_c\right)\right) \Rightarrow \left(vmevt_a \rightarrow vmevt_c\right)$$

For the purposes of this example, let the time-lining function $VH(x)$ produce a timeline (where a timeline is an ordered set of discrete time instances) corresponding to a single VM user session of the user x. The first rule states that a user x must be logged in before executing any application. The second rule states that user x cannot have logged out before performing that execution. If the execution event $vmevt_b$ occurs, the login event $vmevt_a$ must happen before it, and $vmevt_b$ must happen before the logout event $vmevt_c$. Therefore, the physical time t_c at which the event $vmevt_c$ must have occurred must be after the physical time t_b at which the event $vmevt_b$ must have occurred, which must in turn be after the physical time t_a at which the event $vmevt_a$ must have occurred.

This is stated as: $\left(VH(x) \supseteq \langle vmevt_a, vmevt_b, vmevt_c \rangle\right) \Rightarrow \left(t_a > t_b > t_c\right)$ where $\langle \bullet \rangle$ denotes an ordered set.

If, given the two rules $vmevt_a \rightarrow vmevt_b$ and $vmevt_b \rightarrow vmevt_c$, it is not the case that $t_c > t_b > t_a$, then the timestamps (t_a, t_b, t_c) do not reflect the physical times at which the VM events must have occurred. The VM timestamps are therefore inaccurate, as they suggest an internally inconsistent chronology. From this example, the utility of the *happened-before* relation as a basis for proposing rules for the detection of inconsistent VM events is apparent. A hypothesized chronology of a VM computer system can be evaluated for internal inconsistencies by testing the hypothesized sequence of events against a set of *happened-before* rules.

3.2 Detecting Missing VM Events

There are some *happened-before* relations where the first VM event is a precondition for the second. In such relations, the presence of the second VM event necessarily implies the presence of the first. In the example in Section 3.1, the VM login event $vmevt_a$ must occur before the VM application execution event $vmevt_b$; in other words, if $vmevt_b$ occurred, then $vmevt_a$ must also have occurred. Note that this does not hold true for all *happened-before* relations. This can be seen in the same example, where although the execution event $vmevt_b$ must happen before the logout event $vmevt_c$ in order for $vmevt_b$ to happen at all, the occurrence of the logout event $vmevt_c$ does not imply that $vmevt_b$ also happened; $vmevt_b$ is not a precondition for $vmevt_c$.

[10] extends the use of the *happened-before* relation of [3, 5] to imply causality. The *happened-before* relation is therefore equivalent to the "precondition" predicate. For the purposes of the log auditor, it is preferable to maintain the *happened-before* relation as described by [3, 5] and to employ the "precondition" predicate to imply a causal relationship. The *happened-before* relation allows for the detection of events that are listed in the timeline out of the sequence in which they must have occurred, whereas the "precondition" predicate allows for the detection of missing events.

If the VM event $vmevt_a$ that "happened" does not exist in either the set of recorded VM events, ***EvtR***, or in the existing set of inferred VM events, ***EvtI***, then it is a missing VM event. It is a missing VM event because it was removed from, or never recorded in the VM's hypervisor kernel logs, and it was not previously inferred on the basis of relationships and object fields. These VM events could also be called *inferred* VM events, but it is convenient to preserve a distinction between events detected using this approach and other VM inferred events.

Precondition VM events which are absent from ***EvtR*** and ***EvtI*** can be added to the set of missing VM events, which we call ***EvtM***.

The VM login event $vmevt_a$ and the VM application execution event $vmevt_b$, and the logout event $vmevt_c$ have the same definitions as in the previous example. The new rule

states that if the event $vmevt_b$ occurred in the timeline of the VM User object x, then the event $vmevt_a$ must also have occurred.

Missing VM events are suspicious and hence important. They are important because it is possible to deliberately delete them from the hypervisor kernel system logs. Detecting that an event is missing allows for the construction of a more complete timeline, hence helping the VM investigator gain a more complete understanding of the VM computer system. By automatically indicating that at a particular point in the timeline an event was either not recorded or its record was deleted, the forensic software could provide a lead for the subsequent manual investigation. This, in turn, may determine why the record is missing. However, it should be noted that there are many instances where an event may be missing as a result of non-suspicious VM host computer activity.

The log auditor infers VM events to describe an action by or on an object with associated VM temporal data. These inferred VM events are combined with VM events recorded in the hypervisor kernel system logs in order to provide as complete a timeline as possible. In the experiments on computers running VMware sessions on Microsoft Windows, our software prototype inferred many VM events that occurred prior to the enabling of many logging options. There were very few recorded VM events from that early time period in the VM's computer history, and thus these inferred VM events were out-of-context. Such inferred VM events may appear to have occurred outside of the VM user sessions, or in an otherwise inconsistent fashion, however, the absence of complete information must obviously be considered in the VM investigator's assessment as to whether or not the event is suspicious.

This scenario is an example of how the normal configuration of the VM computer system may make an event seem inconsistent.

4 VM Log Auditor Software

A summary of the VM Log Auditor architectural components are provided in Fig. 1. As a VM log inconsistency checking profile tool it maps all hypervisor events, as they are extracted from the kernel logs. It uses a set of shell script parsers to provide an associative and transformational mapping [17,18] of these logs into a normalized database with a set of discovered and inferred VM event tables. The prototype software examines the suspect target VM host file system (which is mounted read-only) and enumerates the set of VM objects of applications, files, and users of the target VMware essx3i computer system. We achieve this by performing a Storage Area Network (SAN) disk image of the suspected VM host to our evidence server in our test bed. The parsed hypervisor log events are described as the set of recorded VM events (***EvtR***) in our Oracle database. Finally, a set of events are inferred from the temporal data associated with each log file which may be as a result of modification, access or creation (MAC) events which have occurred. These events are the inferred VM events (***EvtI***), and are saved in a separate table in the database called Inferred VM Events. It is useful to note that the inferred VM events ***EvtI*** is

particularly vulnerable to inconsistency or incompleteness of data obtained from the VM host file system. The two sets ***EvtR*** and ***EvtI*** do not exhaustively describe the complete history of the VM hosted computer system. Hence where an event is not logged or it cannot be inferred we described this as an unknown or missing event. It stands to reason that the more missing or unknown events the less complete the timeline of the suspect VM host computer system being investigated. Due space limitation we omit the formal completeness proof. After conducting this automated process, the software prototype provides a basic interface for the purpose of detecting temporal inconsistency in a given timeline.

The detection techniques described in this paper match the VM events in a timeline against the events in each rule being tested. Programmatically, every rule is implemented by a C# object, and every event is implemented by a C# object. Rule objects have two event objects as fields, one called $vmevt_a$ and another called $vmevt_b$. The objects $vmevt_a$ and $vmevt_b$ are archetype VM events, against which known VM events are compared. A known VM event is compared against the archetypes on the basis of the fields of each. The fields of the archetype events can have a specific value, or be null. If the archetype has a specific value for a particular field, then any known VM event that matches the archetype must have the same value. If the archetype has a null value for a particular field, it can match any value for the known VM event's corresponding field.

The rule object can also be set to match subject and target fields, that is to say, to require that both matching VM events have the same subject or target field. The rule can also specify that the subject field of one event is the target of the other event, or vice versa. This allows for the definition of generic rules.

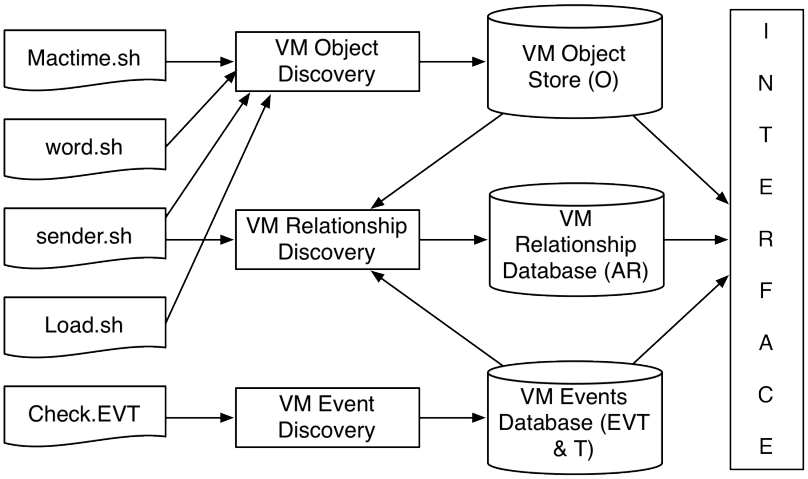

Fig. 1: Architectural components of the VM log auditor prototype

In the object that represented this rule, $vmevt_a$ would represent the "logon" VM event, and $vmevt_b$ would represent the "modified" VM event. A Boolean field of the rule object would be set to true to indicate that the subject of each VM event had to be the same object. Given this, the values of the fields of the objects $vmevt_a$ and $vmevt_b$ would be as follows:

$$vmevt_a = \{\text{null, null, s, logon, success}\}$$
$$vmevt_b = \{\text{null, null, null, modified, success}\}:$$

The prototype log auditor software does not yet implement the concept of a user VM session. A logon or logoff VM event is treated the same as any other event. This means that the user needs to specify which events are to be treated as the beginning and end of the user session timeline. In order to check timelines of a computer system's complete history, the prototype software would need to have a concept of user session built into it. This is an item of future work.

5 Rule Base for Experiments

The VM log auditor software prototype incorporates a small set of rules to check for VM temporal inconsistency. It provides a backend functionality that allows the user to specify a timeline to be checked for inconsistency. It then checks that timeline against the rule base. The rules built into the prototype software for the purposes of these experiments use the following algorithm.

vmevtA = (null, null, **s**, "logon", "success")
vmevtB = (null, null, null, "modified", "success")
rule = **vmevtA** *happened-before* **vmevtB**
where field 2 of **vmevtA** == **x** and where field 2 of **vmevtB** == **x**
for each **vmevt** in V**H(x)**
    if **vmevt** = (*, **x**, **s**, "logon", "success")
        a = index of **vmevt**
    if **vmevt** = (*, **x**, *, "modified", "success")
        b = index of **vmevt**
next **vmevt**
if **a** > **b** then
    rule has been broken

Fig. 2: VM Inconsistency Algorithm

The data structures in our implementation that represented each of the archetype VM events in the rules base, had null values in place of the fields *x*, y.

5.1 Experimental Setup

In order to obtain data for these experiments, we employed a suspect test VMWare essx3i private network hosted computer running on Windows 7 within the University's local area network. This deployment represents our private cloud test environment for the course of the timeline experiments carried out. All system logging options were turned on in order to give us as complete a set of hypervisor event logs which are all stored in a comma delimited format (.csv) on the host oracle databases. We logged onto the VM test host computer twice for the purpose of generating two different VM user sessions: the first, an "innocent" user session, and the second, a user session in which a document was created with misleading authorship information. The details of these two sessions are described below.

We also tinkered with the detection outcomes of meddling with the hypervisor logs. For this purpose, a copy of the case file and database about the suspicious test VM host computer system inspected by the log auditor tool was provided, and then manually modified the database table containing the discovered VM events. As these discovered VM events are derived from the VMWare essx3i kernel system event logs, the removal or modification of recorded VM events in the set *EvtR* effectively simulates the removal or modification of VM event records in the same. The investigator removed the log-on/log-off VM events from the first user session, and modified the timestamps of these events on the second user session so that they would be presented out of their real sequence if ordered by timestamp.

5.2 Evaluation of VM Detection Techniques

This section describes each of the VM timelines examined in each these experiments and provides the results of the log auditor's analysis of inconsistency. There are four VM log timelines (two unmodified, and two modified) that correspond directly to user sessions. Each of the timelines is a combination of the VM inferred events and the recorded VM events in the history of the VM hosted computer system between two boundary events, ordered by timestamp. Due to space limitations the detailed timeline trace tables that capture the recorded and inferred events have been omitted.

Timeline A - Normal VM User Session

Timeline A was a normal user VM session during which a text document was created. The user "thorpe" logged into the VM host computer system at 9:48 pm on 12 October 2011, and created the file "vmsyslog.doc" at 9:51pm. The VM user then browsed the Internet for a few minutes and logged off at 10:00 pm. Nothing suspicious happened in the user VM session. The timeline consisted of all of the events that took place during the user VM session, both recorded and inferred. Our software inserted these VM events into its Oracle 11g VM event database during its automated examination of the target system.

Most events in timeline A were discovered events (i.e. discovered in the VMWare essx3i hypervisor kernel event logs running under Windows 7), however, the events with "CREATED", "MODIFIED" or "OPENED" as their actions were inferred events (i.e. inferred on the basis of an object, its relationships, or other information about the object).

Timeline B - Deliberate Misattribution of Authorship

Timeline B represents a user VM session during which the user created a text document with misleading authorship information, in an effort to shift responsibility for that document to an innocent third party. The user "VMuser" logged into the computer system at 9:51pm on 13 October 2011, and at 9:55pm a Word document was created with "anoynmous " as the listed author. The user "VMuser " then logged off.

Timeline B was analyzed for inconsistency with our prototype software. The events are all related to the authorship of the word document entitled "WORDOC letter from anonymous". The anonymous user was not logged in at the time the text document was created, and yet the author field listed "anonymous" as the document's author.

There are two sets of "CREATED" events for both the suspect Word document and its template. Hence "anonymous" could not have been the author of the text document. This is so because there are two sources of information that lead the log auditor inferring such an event. The earlier timestamp is obtained from the text document's metadata, and represents the time at which the document was first created. The later timestamp is obtained from the target VM host computer's file system, and is the time at which the document was first saved as a file on the host VM physical disk. Both sets of "CREATED" VM events derive their subject field from the same source, the Word document's author field .

Timeline C - VM user Session with Logon/Logoff Events Deleted

Timeline C was derived from timeline A. The recorded and inferred VM events in the prototype's events database were copied and manually modified. The resulting timeline, timeline C, was identical to timeline A without the logon/ logoff VM events. The removal of these two discovered VM events left user activity outside of a logon/logoff-bound VM user session. This demonstrates that removing VM user session information from the hypervisor event log will draw attention to the inferred VM events that took place during the session.

Timeline D - With VM user Modified Timestamps

Timeline D was derived from timeline A, with the timestamp of the user's logoff VM event deliberately modified so as to appear to have taken place prior to the creation of the text document.

The event was listed as breaking three rules, all of which assert that if a file is modified, accessed or created, it must be modified, accessed or created prior to the user logging out of the VM host computer system. The results of the analysis of timeline D were just as expected.

The detection of this VM event demonstrates the suitability of this approach to detecting events whose timestamps are modified.

6 Constructing Consistent Timelines

The temporal inconsistencies can be handled by the creation of a consistent VM timeline. A consistent VM timeline, in the context of VM computer profiling, is defined as a sequence of VM events ordered by physical time at which they occurred with no obviously missing VM events. The physical time at which the event occurred may or may not correspond to the VM hosted computer generated timestamp of a VM event, which may be missing from the sets ***EVTR*** and ***EVTI***, but which are detected using the techniques that establish VM relationships via hypervisor log object profiles. ***EVTM*** is the set of all of the missing VM events detected on the basis of a precondition rule. The sequence ***EVTC*** is a sequence over ***EVT*** ordered into a consistent VM timeline. This section describes a technique for constructing such a timeline.

There are some VM events, especially members of ***EVTM***, for which there is no timestamp. There are other VM events for which there is a timestamp, but whose timestamp is provably incorrect (as determined by detecting the out of sequence experiments). Gladyshev and Patel describe the process of determining the time at which a given event takes place by bounding the event's time [4], and we adopt this approach for our VM log auditor tool. The upper and lower bounds for the time of an event can be determined if the VM event must have occurred between two other VM events. The range between these bounds, i.e. the time interval Δt, is the range of possible times at which the VM event could have occurred. The range can be further narrowed if it is known that there is a minimal delay d , which applies to a particular *happened-before* relation [4, 5]. If it is known that there is at least a ten seconds greater than the time of the first event. The range of possible times at which a VM event $vmevt_b$ might have occurred can be calculated.

This approach can only provide a range of times in lieu of a missing or inaccurate timestamp, but such a range is the best possible indication of when the event occurred. As it is impossible to obtain perfect timestamps for every VM event in the history of the VM computer system, the sequence ***EVTC*** cannot be ordered on the basis of the available timestamps. The available timestamps will not be precise enough for ordering the events in and themselves, although they might be useful in determining some other basis for ordering events.

Instead of timestamps, the use of a Lamport logical clock [7] is proposed to provide the basis of ordering the consistent timeline ***EVTC*** . The timestamp (or time interval in the case of VM events with indeterminate time) of a VM event will be used as a variable in the clock, but it will be the clock and not the timestamp which will be used as the basis for ordering VM hosted events.

The VM hosted computer BIOS clock C is defined to be a function which assigns a number to every VM event in the consistent timeline ***EVTC***. The number produced by C has no bearing on physical time, but each VM event has a timestamp t for a time interval Δt which can be used to determine the physical time of the VM event. The number produced by C must be lower for VM events which occurred earlier in the VM history of the computer system than the number produced for VM events which occurred later. This will permit events to be sorted by the number produced by the clock C on the ascending order basis.

Given a complete set of rules to detect inconsistent and missing VM events, the number of unknown VM events in the VM computer system's history can be minimized. The proportion of the set of all of the VM events that occurred in the VM history of the VM hosted computer system ***EVT*** that is known can be maximized. A VM event is a known event if it is an element of the sets ***EVTR***, ***EVTI***, or ***EVTM***. Each of the known events will have an associated timestamp, or, in the case of VM events with no timestamps or with provably incorrect timestamps, and the narrowest possible time interval during which the event could have occurred. The clock function C will combine the timestamp or time interval for each known event with the rules relating that event to the other knowable VM events, and produce a number (i.e. Lamport timestamp) according to which the VM event may be sorted into the consistent timeline ***EVTC***. In the case of potentially concurrent events (which have no *happened-before* relations to other VM events) whose timestamps or time intervals are inconclusive relative to other VM events, the number produced by C will be identical. In such cases, some arbitrary mechanism can be used to sort VM events into the consistent timeline ***EVTC***. Once completed, the consistent timeline ***EVTC*** will represent the best sequential ordering of known events that make up the associated VM history profile.

This permits an investigator to view both a consistent timeline, and the original inconsistent timeline along with the reasoning as to why the original hypervisor temporal log data was inconsistent. Then the investigator could then make an informed judgment about the cause of the inconsistencies in the VM computer system timeline.

7 Discussion of Results

The results of the experiments demonstrate that automatically detecting temporal inconsistency in VM hosted computer activity timelines constructed from realistic data is possible using our tool. These experiments applied a simple rule set to a VM hosted computer system's activity timeline, and the results demonstrate that inconsistency can be detected in several basic scenarios. The *happened-before* relation and the precondition predicate can be used together to construct effective rules to draw an investigator's attention to suspicious VM events. Timeline B demonstrated that such rules can be

applied to detect an event (in this case, the creation of a document) initiated by a different user than first suggested by the VM file system.

Timeline C showed that the deletion of a hypervisor kernel system log set of entries pertaining to important VM events can be detected. If the deleted events are preconditions for other events, which are recorded or inferred, then they can be detected. Timeline D demonstrated that, by applying a rational set of rules in an automated analysis of a timeline, VM events can be detected that should have occurred in another sequence than their timestamps suggest.

The experiment's use of data from a VM hosted computer system demonstrated that this approach to detecting temporal inconsistency on VM log data is robust enough to be tested in real cases. The ideal next step will be to perform experiments with the log auditor using large scale live incident case data, which will test the robustness and suitability of the approach with respect to real investigations.

The noise in real VM event data is a lesser problem to the VM log auditing tool than it is to a human investigator. By distilling VM event records down to the most important fields that are common to most events, our approach is likely to reduce the complexity and heterogeneity of the various types of VM events.

8 Future Work

We plan to improve the log auditor so that the software can automatically detect user sessions given our ongoing work [16]. At the moment, the prototype software requires the user to specify the bounds (i.e. start and finish) of a user VM session before it is able to check the timeline of that VM session for internal consistency.

We would like to extend the VM log auditor process and software to construct consistency-check timelines of VM hosted computers running non-Windows operating systems.

The basic model presented in this paper doesn't detect hidden nested states that may exist on the VM host as extracted from the hypervisor logs particularly as a remote or distributed logging service. We plan to extend the existing model to now explore and formalize this concurrency problem.

9 Related Work

Although there has not been any formal accepted definition for cloud forensics, its fundamentals are still entrenched in a view that the data provided as case evidence has to be court admissible. The provision of a time-lining technique together with the practical approaches for gathering and inferring VM events comprise a technique for tracing the history of the VM hosted computer system as possible source of such potential forensic

evidence is motivated by prior work [20, 21, 22, 25]. In this paper, we attempt to improve on the state of the art by providing a forensic platform that transparently and distinctively monitors and records data access events using the hypervisor kernel event logs. This work adopts a static state snapshot log approach to support post incident off line forensic investigations. Our work complements the live trace analysis and VM introspection methods [21, 22, 25] and the static snapshot finite state hypothesis computational models [5, 6, 11].

10 Conclusion

This work has produced a tool, implementing techniques for detecting contradictory and missing VM events in the history of the computer system. The experiments with this software demonstrate that the techniques that have been proposed can be used successfully to detect temporal inconsistencies in a VM computer activity timeline. The automatic detection of inconsistencies that might indicate deliberate tampering could assist a human investigator in a subsequent manual examination of the VM hosted system running within the data center.

References

1. Rodgers, M.: The role of criminal profiling in the computer forensics process. Computers & Security **22**(4) (2003) 292-298
2. Rodgers, M., Goubalt–Larrecq, J.: Log auditing through model checking. In: Proceedings of the 14th IEEE Computer Security Foundations Workshop, Cape Breton, Nova Scotia (June 2001).
3. Boyd, C., Forster, P.: Time and date issues in forensic computing a case study. Digital Investigation **1**(1) (2004)18-23
4. Buchholz, F., Tjaden, B.: A brief study of time. In: Proceedings of the 7th Digital Forensics Workshop, Pittsburg, Pennsylvania, USA, (August 2007).
5. Fidge, C.: Logical time in distributed computing systems. Computer 24(1) (1991)28-33
6. Gladyshev, P., Patel, A.: Formalizing event time bounding in digital investigations. International Journal of Digital Evidence **4**(2) (2005) 1-14
7. Lamport, L.: Time, clocks, and the ordering of events in a distributed system. Communications of the ACM **21**(1)(1978) 558-565
8. Marrington, A., Mohay, G., Clark, A., Morarji, H.: Event-based computer profiling for the forensic reconstruction of computer activity. In: Proceedings of the AusCERT Asia Pacific Information Technology Security Conference,Gold Coast,Australia, (May 2007).
9. Marrington A., Mohay G., Morarji H., Clark A.: A Model for Computer Profiling. In: Proceedings of the 5th International Workshop on Digital Forensics at the International Conference on Availability, Reliability and Security, Krakow, Poland, (February 2010).

10. Nolan R., O'Sullivan C., Branson J., Waits C.: First responder's guide to computer forensics. Software Engineering Institute, Carnegie Mellon University, Pittsburg, USA, (May 2005).
11. Schatz B., Mohay G., Clark, A.: A correlation method for establishing provenance of timestamps in digital evidence. In: Proceedings of the 6th Annual digital forensic research workshop, West Lafayette, Indiana, USA, (August 2006).
12. Willassen, SY.: Hypothesis-based investigation of digital timestamps. Advances in Digital Forensics IV **285** (1) (2008) 75-86
13. Willassen, SY.: Timestamp evidence correlation by model based clock hypothesis testing. In: Proceedings of the 1st international conference on forensic applications and techniques in telecommunications, information, and multimedia and workshop. Adelaide, Australia, (January 2008).
14. Willassen, SY.: A model based approach to timestamp evidence interpretation. International Journal of Digital Crime and Forensics **1**(2) (2009) 1-12
15. Thorpe, S., Ray, I., Grandison, T.: A Formal Temporal Log Model for the synchronized Virtual Machine Environment. Journal of Information Assurance and Security,6 (5) (2011) 398-406
16. Thorpe, S., Ray, I., Barbir A., Grandison, T.: Towards a Formal Parameterized Context for a Cloud Computing Forensic Database. In: Proceedings of the 3rd Digital Forensics and Cybercrime Conference, Dublin, Ireland, (October 2011).
17. Thorpe, S., Ray, I., Grandison, T., Associative Mapping Techniques for the synchronized virtual machine environment. In: Proceedings of the 4th Computational Intelligence in Security for Information Systems Conference, Torremolinos, Spain, (June 2011).
18. Thorpe, S., Ray, I., Grandison, T.: Enforcing Data Quality Rules for the synchronized virtual machine environment. In: Proceedings of the 4th Computational Intelligence in Security for Information Systems Conference, Torremolinos, Spain (June 2011).
19. Thorpe, S.: PhD Thesis -The Theory of a Cloud Computing Digital Investigation using the Hypervisor kernel logs, University of Technology Jamaica, (February 2013).
20. Thorpe, S., A Virtual Machine History Model Framework for a Data Cloud Investigation. Journal of Convergence **3**(4) (2012) 9-14
21. Srinivas, K., Snow, K., Monrose, F.: Trail of Bytes: Efficient support for Forensic Analysis. In: Proceedings of the ACM Conference on Communication Security, Chicago, Illinois, USA, (October 2010).
22. Gidwani, T., Argano, M., Yan, W., Issa, F.: A Comprehensive Survey of Event Analytics. International Journal of Digital Crime and Forensics, **4**(3) (2012) 33-46
23. Thorpe, S., Ray, I., Grandison, T., Barbir, A.: A Model for Compiling Truthful Forensic Evidence from the Log Cloud Hypervisor Databases. In: Proceedings of the 28th Annual Computer Security Applications Conference (ACSAC), Work in Progress Session, Orlando, USA, (December 2012).
24. Thorpe, S., Ray, I., Grandison, T., Barbir, A.: Log Audit Explanation Templates with Private Data Clouds. In : Proceedings of the 28th Annual Computer Security Applications Conference (ACSAC), Work in Progress Session, Orlando, USA, (December 2012).
25. Pauw, W., Heisig, S., Visual and algorithmic tooling for system trace analysis: A case study. In: ACMSIGOPS Operating system review **44**(1)(2010) 97-102.