Journal of Information Assurance and Security.
ISSN 1554-1010 Volume 7 (2012) pp. 24-31

Detecting Temporal Inconsistency in Virtual Machine Activity Timelines

Sean Thorpe[1], Indrajit Ray[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

***Abstract*: The construction of timelines of Virtual Machine (VM) Computer activity is a part of many digital investigations. These timelines of VM events are composed of traces of historical activity drawn from the hypervisor kernel system logs found on the VM host computer file system. A potential problem with the use of such information is that some of it may be inconsistent and contradictory thus compromising its value. This work introduces a software tool called the virtual machine log auditor for the detection of inconsistency within timelines of virtual machine activities. This is an important first step towards the resolve of Cloud Computing Digital Investigations. We examine the impact of deliberate tampering through experiments conducted with our prototype. Based on the results of these experiments, we discuss techniques which can be employed to deal with such temporal inconsistencies.**

***Keywords*:** Virtual Machine, logs, cloud, temporal, forensics**.**

I. Introduction

In this paper, we consider the issue of temporal inconsistencies in digital evidence, and their impact on virtual machine digital investigations. Our work is motivated by earlier work done in [13,14,18]. By temporal inconsistency, we mean an incongruity in the digital evidence pertaining to the sequence of VM events in the history of the computer system, which could lead to the VM history being inaccurately reconstructed. Temporal inconsistencies can impede VM digital investigations in which timelines of computer activity are an important part of the VM digital evidence under consideration. This includes any sort of cloud investigation in which determining an accurate sequence of VM events is crucial to understanding the crime and building a case.

Managing the cloud computing environment has become a very pertinent matter. As more users become attracted to the economies of scale benefits of launching cloud services, the level of crime that now faces these logical service domains is always increasing.

The most common inconsistency in VM digital evidence is naturally occurring, that is to say, inconsistency which is not the result of deliberate tampering with the hypervisor (virtual operating system) logs, that monitor the underlying physical operating system disk running within the storage area networks (SAN) of the private data centre. This may include data pertaining to a VM event log or application file instance which simply is not recorded, or may have been over-written during the normal operation of the underlying computer system that host the VM. It also includes "naturally" erroneous or inaccurate data, perhaps due to a hardware characteristic, software misconfiguration or bug. Such natural inconsistencies pose difficulties for forensic investigators, even if they are not the result of deliberate action taken by a suspect.

Timestamps generated by computer clocks are an example of data of such unreliable accuracy. Where multiple clocks pertaining to a case generate timestamps, the normal behavior of computer hardware clocks (that is to say clock skew and drift) will cause inconsistency between the different time sources. Schatz [9] discuss an approach for dealing with such inconsistencies. Their approach baselines the behavior of inconsistent computer clocks via correlation with records from devices with more authoritative timestamps. In the traditional single-computer investigations, clock drift and skew can still lead to inconsistent timestamps in the evidence. Despite the fact that there is only one clock providing the timestamps in a single-computer system investigation, severe cases of clock drift and skew can cause the timelines that are constructed to be misleading.

In extreme cases if there is a clock reset at VM reboot or some other mishap and time "goes backwards" then events may appear out of the sequence in which they actually occurred.

The deliberate modification of VM log records to obscure records of suspicious activity creates another, and generally more concerning, sort of inconsistency in the VM digital evidence. For example, a user who downloads illegal material may attempt to obscure that fact by deleting web browser history and caches, event log records showing their login, opening the browser application, and logging off. If, in a subsequent forensic investigation, the illegal material is discovered, but the user was successful in his/her destruction of log data, then the illegal material will appear to have been downloaded outside of a user session.

The rest of this paper is organized as follows. Section II introduces related work which informed our research. Section III 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 IV describes our experiments with our tool for testing the approaches discussed in Section III. Section V describes the results of those experiments and evaluates the detection techniques. In Section VI we list the limitations of the log auditor detect tool at the time of writing this paper, as well as limitations of the research described in this paper. Section VII is a discussion of future work in the area of detecting inconsistency in virtual machine computer activity timelines, and Section VIII is our conclusion.

II. Related Work

This work employs some of the concepts from the computer profiling model described by Marrington [9]. This model of a computer system consists of objects representing the various entities which 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 (together comprising the set O) are classified according to their type.

In Marrington et al's model [9], there is four broad types of objects (Application, Principal, Content and System) with increasingly specific subtypes. We represent each of these categories as sets. The set of Application objects, A, consists of all the application software on the 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 which 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 System objects, S, consists of all the configuration information, system software and hardware devices on the computer system. A, S, C, and P are all subsets of O. The model also describes relationships between these objects, but these are unrelated to this work.

We adopt from the model the set of all times in the history of the virtual machine computer system, T_v, and the set of all events, Evt, which have taken place in the history of the computer system. Let t be a time in T_v, x being the object which instigated the event, y be the object which was the target of the event, describes the action of the event, and describe the result of the event (successful, unsuccessful, or unknown). An event evt in the set Evt consists of the quintuple:

evt = (t, x ∈ O, y∈O, ,).

We also adopt from the model, the finite set Evt which consists of two enumerable subsets, and one subset which cannot be enumerated. The first subset consists of events which are recorded in the computer system's logs. The second consists of events which are not recorded in logs, but which can be inferred on the basis of other digital evidence on the system (such as relationships between different objects). These are the recorded events (EvtR) and the inferred events (EvtI) respectively. These two sets do not exhaustively describe the complete history of the computer system. There may be other events which took place which were unrecorded and left no artifact from which they could be inferred. These 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 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 computer system's logs, and consequently, will incorporate any inaccurate event records in the system logs. Further, if an event is not logged, and cannot be inferred, it will not be anelement of either EvtR or EvtI. Such an event will therefore be an unknown event, and the more unknown events in the history of the computer system, the less complete the timeline of the target computer's activity will be. This paper provides a means for the semi-automated detection of inaccuracy or incompleteness leading to chronological inconsistency in timelines of virtual machine computer activity.

In another work, Marrington [6] discussed a timestamp-based technique for building a timeline about a given object in the profile of the computer system. A timeline is a sequence over the set Evt ordered by the timestamp t of each event where the subject or target of the event was the object being time-lined o. Such a timeline is constructed by querying a database of all the recorded events and all the inferred events in the computer system's history with the object being timelined as either the subject or target of the event, and then ordering the results by the event timestamp.

This approach is not resilient to inaccuracies in timestamps, which may cause events to appear out of sequence. Missing events, whether removed manually or simply never recorded, lead to timelines which may present events out of the context in which they actually occurred. Consequently, this approach to constructing timelines of computer activity must be supplemented with techniques to detect and deal with inconsistency and incompleteness.

We note that as a general principle, the failure to detect an inconsistency in a timeline is a greater problem for the purposes of computer 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.

An obvious 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 "natural", that is to say, it is the result of the normal operation of the computer system.

The solution for addressing this issue 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 varying over time prior to the examination of the computer system, and it is this variance

which may lead to inaccuracies in timelines. Studies of large numbers of hosts on the Internet suggest that many computer clocks are significantly inaccurate (by a margin of more than 10s) and that the clocks of many hosts do not conform to the existing models of clock behavior. [2,10] proposes an 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 hypothesis 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 computer 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.

III. VM log Auditor detection of Inconsistent Timelines

This section describes the approaches our log auditor tool employs 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 which should be in the timeline are missing. The approaches we describe in this section address both of these scenarios.

Before testing for inconsistency employing the approaches described in this section, our tool has to perform several tasks. First, it parses the copied hypervisor event logs from which our test VMWare essx3i host runs. Although our tool is intended for the examination of VMs running on Windows computers, the approach could easily be adapted to other physical operating systems with different vendor specific variants of virtual machines that may run on them.

Each event in each of the three logs (i.e. system, error, and application) is normalized and stored in a database table of recorded events. Each event is stored as a database row including an ID, a timestamp, the user/application which instigated the VM event, the object of the event, the action of the event, and the result of the event. Second, our tool walks the computer's hard drive and extracts MAC (modified-accessed-created) times and file metadata containing VM timestamps. Third, our tool creates a table of inferred events in the database for each of the timestamps found in the walk of the file system. These VM events are normalized according to the same pattern as recorded events extracted from logs. The log auditor then has enough data to both construct a VM computer activity timeline and to test it for internal inconsistency.

A. Detecting out of sequence VM Timelines

It is obvious that there are some events which can only take place after some other another event. This sort of relation 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 log into the computer system before the user x can execute the application y. Applied to our VM activity time-lining, the VM real time, if not the hypervisor kernel system log timestamp, of the execution VM event must be after the real time of the VM login event.

Let $x \in P$, $y \in$, $t_n \in T_v$ and $t_m \in T_v$

$(t_n , x , \text{VM system, login , success}) \rightarrow (t_m , x, y, \text{execute , success}) \rightarrow t_m > t_n$

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 timestamps can be made. Consider two rules:

$vmevt_a \rightarrow vmevt_b$

$vmevt_b \rightarrow vmevt_c$

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, and:

$vmevt_a = (t_a \in_c T_v$ x, VM system, login, success)

$vmevt_b = (t_b \in_c T_v$, x, a, execute, a, $\in$ {success,fail, unknown }

$vmevt_c = t_c \in_c T_v$, x, VM system,logout; success}

Note that the happened-before relation is transitive [4, 5]:

$(vmevt_a \rightarrow vmevt_b) \wedge (vmevt_b \rightarrow vmevt_c) \rightarrow vmevt_a \rightarrow vmevt_c$

For the purposes of this example, let the time-lining function VH(x) produce a timeline 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, 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:

$VH(x) \supseteq \{vmevt_a, vmevt_b, vmevt_c\} \rightarrow \{t_c > t_b > t_a\}$

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.

B. 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 $vmevt_b$, such that if $vmevt_b$ occurred, then $vmevt_a$ should also have occurred. This does not hold true for all happened-before relations, however. 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. This is because $vmevt_b$ is not a precondition for $vmevt_c$. Where such a precondition does exist, it is expressed with the predicate "precondition", as shown below. A second predicate, "happened", is employed to assert that some event occurred.

$$(vmevt_a \rightarrow vmevt_b) \wedge (happened(vmevt_b) \rightarrow happened(vmevt_a)$$

$$precondition(vmevt_a, vmevt_b)$$

[10] extends the use of the happened-before relation of [3,5] to imply causality. Willamssen's version [10] of 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 which 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 $vmevt_a$ which "happened" does not exist in either the set of recorded VM events EvtR or the existing set of inferred VM events EvtI, then it is a missing event. It is a missing event because it was removed from or never recorded in the VM's hypervisor computer system's 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.

The rules base in the example in Section IIIA can be expanded to include all pairs of events for which the "precondition" predicate is true. If an event evt_x has a precondition event specified by a rule, then the presence of the precondition event can be inferred, even if it is absent from EvtR and EvtI.

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

The new rules base, expanded from that in Section III , is:

$vmevt_a \rightarrow vmevt_b$

$vmevt_b \rightarrow vmevt_c$

$precondition(vmevt_a, vmevt_b)$

The login event $vmevt_a$ and the 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 User object x, then the event $vmevt_a$ must also have occurred. This is expressed:

$$(vmevt_b \in VH(x)) \rightarrow happened(vmevt_a)$$

$$vmevt_a \in EVT$$

$$vmevt_a \notin (EvtR \cup EvtI) \rightarrow vmevt_a \in EvtM$$

Detecting missing VM events is important; as such an event may have been deliberately deleted from the hypervisor kernel system logs, which may in itself be suspicious. 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 a particular point in the timeline an event was either not recorded or its record was deleted, such software could provide a lead for subsequent manual investigation, which may determine why the record is missing. If the VM event record retrieved from the hypervisor logs was deliberately deleted, this may indicate that the user was attempting to conceal suspicious activity.

There are many instances where an event may be missing as a result of non-suspicious computer activity.

Our log auditor infers VM events to describe an action by or on an object with associated temporal data. These inferred VM events are combined with events recorded in the hypervisor kernel system logs in order to provide as complete a timeline as possible. In our experiments on computers running VMware sessions on Microsoft Windows, our log auditor inferred many VM events which occurred prior to the enabling of many logging options. There were very few recorded VM events from that early time period in the computer's history, and thus these inferred VM events were out-of-context. Such inferred events may appear to have occurred outside of 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.

IV. VM Event Detection Experiments

This section describes experiments in which the approach to detecting temporal inconsistency in user sessions described in Section III was tested. We examine timelines as developed by our log auditor prototype software in the following experiments:

- The unmodified timeline of a VM user session during which the user creates a document, and does not attempt to obscure his/her actions.

- The unmodified timeline of a VM user session during which the user creates a document with deliberately misleading authorship information.

- Modified timelines of the VM user sessions where the hypervisor kernel system logs have been tampered with.

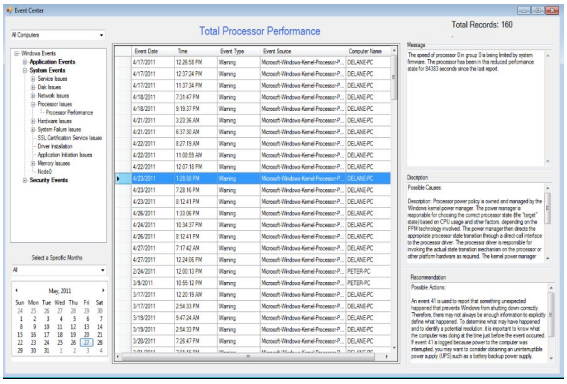


Figure 1. Files organized according to VM System Log timestamp information

A. *VM Log Auditor Software*

As mentioned in previous sections, we developed the log Auditor in order to detect inconsistency in virtual machine hosted computer activity timelines. The prototype software examines the suspect target VM host file system (which is mounted read-only) and enumerates the applications, files, and users of the target VMware essx3i computer system. We achieve this by performing a SAN disk image of the suspected VM host to our third party evidence server in our test bed [16,17]. The VMware hypervisor logs are parsed, and the events described in those logs are stored 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 file. These events are the inferred VM events (EvtI), and are saved in a separate table in the database called Inferred Events. After conducting this automated process, the software prototype provides a basic interface for the purpose of detecting temporal inconsistency in a given timeline, shown in Figure 1.

The detection techniques described in Section III match the 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 which 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.

In the object which 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 $_a$ = {null, null, s,logon, success }
vmevt $_b$ {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

B. *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 back end functionality which 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 = vm**evtA** *happened-before vm***evtB**
where field 2 of **evtA** == **x**
and where field 2 of **evtB** == **x**
for each **vmevt** in V**H(x)**
if **evt** = (*, **x**, **s**, "logon", "success")
a = index of **evt**
if **vmevt** = (*, **x**, *, "modified", "success")
b = index of vm**evt**
next **evt**
if **a** > **b** then
rule has been broken

Figure 2. VM inconsistency detection algorithm

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

C. *Data*

In order to obtain data for these experiments, we employed a suspect test VMWare essx 3i hosted computer running on Windows 7. All system logging options were turned on in order to give us as complete a set of hypervisor event logs. 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, we copied the case file and database about the test VM computer system inspected by our log auditor tool, and then manually modified the database table containing the discovered events. As these discovered 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. We 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.

V. Evaluation of VM Detection Techniques

This section describes the hypothesis each of the timelines examined in these experiments. There are four timelines (two unmodified, and two modified) which 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.

A. Timeline A - Normal VM User Session

Timeline A was a normal user session during which a text document was created. The user "thorpe" logged into the computer system at 9:48 pm on 12 October 2011, and created the file "vmsyslog.doc" at 9:51pm. The user then browsed the Internet for a few minutes and logged off at 10:00 pm. Nothing suspicious happened in the user session. The timeline consisted of all of the events which 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).

B. 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 "Vmuser " as the listed author. The user "Vmuser " then logged off.

Timeline B was analysed for inconsistency with our prototype software. The "Vmuser" was not logged in at the time the text document was created, and yet the author field listed "Vmuser" as the document's author.

Therefore, "VMuser" could not have been the author of the text document.

This is so because there are two sources of information which 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 computer's file system, and is the time at which the document was first saved as a file on the disk. Both sets of "CREATED" VM events derive their subject field from the same source, the Word document's author field originated.

C. 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 which took place during the session.

D. 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.

E. 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 which 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 real 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 which are common to most events, our approach is likely to reduce the complexity and heterogeneity of the various types of VM events.

VI. Limitations

As suggested and reinforced at several points throughout this paper, the log auditor software has several limitations. We hope that these limitations will be addressed in future versions of the tool.

The most serious of these limitations is the log auditor's inability to automatically detect VM user sessions. This requires the user to provide the boundaries (i.e. first and last VM event) of the computer activity timeline whose consistency they wish to evaluate. This is a serious limitation as it requires the investigator to have some knowledge, at least with respect to the period, of the VM event under investigation.

If this limitation could be overcome and the log auditor could identify VM user sessions itself, then it could be used to assess entire VM computer histories for inconsistency with no prior knowledge.

The experiments described in this paper were limited as they were not conducted using data from real cases. Instead, the experiments were conducted using a simplistic test scenario performed on a test machine as a hypothetical "case". Although the log auditor performed advisably well in this case, we are not yet able to validate its robustness with respect to real investigations.

Further, our experiments using the data from the test machine were limited in their extent only to the prescribed operating system and the software installed on that machine (Microsoft Office 2007, and other common "office computer" software). Results with newer versions of Windows or non-Windows operating systems running VMWare or otherwise may vary. Further testing with different data sets is required. This further testing will allow investigators to establish confidence in the log auditor tool. In a complementary paper we hypothesize as to using the VM log auditor to simulate cloud investigations using linux based file timestamp information[14].

It is hoped that these limitations will be addressed by the future work described in Section VII below, and through the release of the log auditor prototype as free and open source software. We hope that releasing log auditor as an open source forensic software will achieve two things. First, the auditor can attract a community of users who will use it in a variety of cases and provide feedback. Secondly, we hope researchers can use log auditor to address shortcomings and improve upon its functionality.

VII. Future Work

The VM Log Auditor is still a work in progress. There are five main areas in which we hope to improve VM log auditor.

Primarily, we hope to create an interface in which rules for inconsistency can be created and saved in a VM host configuration file.

These rules can then be shared amongst digital investigator communities so they can use them during their investigative process. Second, we hope to build an automatic hypervisor event log parser into log auditor to speed-up the overall process of acquiring event log data.

Third, we hope to improve log auditor so that the software can automatically detect user sessions [15]. 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 session for internal consistency.

Fourth, we hope to extend the VM log auditor process and software to construct consistency-check timelines of VM hosted computers running non-Windows operating systems. Although we have already started to explore this under linux [14], we also would like to test and refine the log auditor with data from Windows azure computers, and compare the results to otherwise similar cases where the computer involved ran Windows 7.

Finally, in order to further validate and improve the log auditor, it is important to conduct experiments on known cases which involve some timeline inconsistencies. We are particularly interested to test the robustness of the log auditor approach in real cases involving deliberate tampering over public clouds. This can help validate our proposed method with relation to real-life scenarios considering the various dimensional complexities and configurations likely on multi-tenant VMs running within these abstract logical domains.

VIII. Conclusion

Inconsistencies in a VM computer activity timeline can compromise the value of the timeline as an investigative tool. If an investigator accepts the original digital evidence from the target VM hosted computer system unbiased, a time-lining tool may produce a history of the VM computer system which is unusable as a result of inaccuracy. Perhaps worse, the investigator may fall victim to an adversary's deliberate modification of hypervisor kernel system logs and other temporal data, and create a misleading VM history of the adversary's own devises.

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

References

[1] Boyd C, Forster P. Time and date issues in forensic computing ea case study. Digital Investigation; 2004:18-23.

[2] Buchholz F, Tjaden B. A brief study of time. Digital Investigation 2007; 4:31-42.

[3] Fidge C. Logical time in distributed computing systems. Computer 1991;vol. 24:28-33.

[4] Gladyshev P, Patel A. "*Formalising event time bounding in digital investigations.* International Journal of Digital Evidence 2005; vol. 4.

[5] Lamport L. "Time, clocks, and the ordering of events in a distributed system. Communications of the ACM 1978; 21:558-65.

[6] Marrington A, Mohay G, Clark A, Morarji H. Event-based computer profiling for the forensic reconstruction of computer activity. In: AusCERT Asia Pacific Information Technology Security Conference 2007 Refereed R&D Stream, Gold Coast; 2007. p. 71-87.

[7] Marrington A, Mohay G, Morarji H, Clark A. A Model for Computer Profiling. In: Third International Workshop on Digital Forensics at the International Conference on Availability, Reliability and Security, Krakow; 2010. p. 635-640.

[8] Nolan R, O'Sullivan C, Branson J, Waits C. First responder's guide to computer forensics. Pittsburgh: Software Engineering Institute, Carnegie Mellon University; 2005.

[9] Schatz B, Mohay G, Clark, A. A correlation method for establishing provenance of timestamps in digital evidence. In: Digital Investigation. The *Proceedings of the 6th Annual digital forensic research worksho*p (DFRWS '06), vol. 3; 2006/9.p. 98-107.

[10] Willassen SY. Hypothesis-based investigation of digital timestamps. In: Advances in Digital Forensics IV, vol. 285.Boston: Springer; 2008a. p. 75-86.

[11] 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: ICST (Institute for computer sciences, social-Informatics and Telecommunications Engineering); 2008b.

[12] Willassen SY. A model based approach to timestamp evidence interpretation. International *Journal of Digital Crime and Forensics* 2009;1:1-12

[13] Thorpe S, Ray I, Grandison T. Towards a Formal Temporal Log Model for the synchronized Virtual Machine Environment. *Proceedings of the Journal of Information Assurance and Security (JIAS), Volume 6, No 2*. 2011.

[14] Thorpe S, Ray I. File timestamps in Cloud Digital Investigations. To appear in the *Journal of Information and Assurance, Vol 7 issue, March 2012*.

[15] Thorpe S, Ray I, Barbir A, Grandison T. Towards a Formal Parameterized Context for a Cloud Computing Forensic Database. *Proceedings of the 3rd Digital Forensics and Cybercrime Conference, October 2011. To appear in the Springer –Verlag LNCS series.*

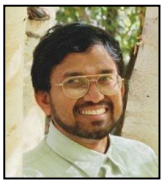
[16] Thorpe S, Ray I, Grandison T. Associative Mapping Techniques for the synchronized virtual machine environment. *Proceedings of the 4th CISIS Conference, June 8 2011.*

[17] Thorpe S, Ray I, Grandison T. Enforcing Data Quality Rules for the synchronized virtual machine environment. *Proceedings of the 4th CISIS Conference June 8 2011.*

[18] Thorpe S, Ray I. The Theory of a Cloud Computing Digital Investigation using synchronized virtual machine kernel logs. (Unpublished PhD Thesis).

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.