Formal Parameterization of Log Synchronization Events within a Distributed Forensic Compute Cloud Database Environment

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

[1] Faculty of Engineering and Computing, University of Technology, Kingston, Jamaica
thorpe.sean@gmail.com
[2] Department of Computer Science, Colorado State University Fort Collins, USA
{iray,indrajit,france}@cs.colostate.edu
[3] IBM Research
YorkTown Heights, NY, USA
tyroneg@us.ibm.com
[4] Bank of America
abbie.barbir@bankofamerica.com

Abstract. Advances in virtual server internetworking and the Internet have been accompanied by increased incidences of computer related crimes for such domains. At the same time, the number of sources of potential evidence in any particular cloud computing forensic investigation has grown considerably, as evidence of the occurrence of relevant events can potentially be drawn not only from multiple computers, networks, and electronic systems but also from disparate personal, organizational, and governmental contexts. Potentially, this leads to significant improvements in forensic outcomes but is accompanied by an increase in complexity and scale of the event information, particularly since such information is treated as composite events. In order for digital investigators to effectively administer the virtual machine(VM) environments they need to have automated methods for correlating and synchronizing such event data as a critical concern. The contribution of the paper is the provision of a University case study of our ongoing work that integrates an automated detection of a computer forensic scenario for virtual network server clouds. This is work based upon facts derived from digital events synchronized within the VM environment. We use our preliminary case evaluations to present the formal parameterized context for which such VM log events are likely to occur based on the event condition action (ECA) paradigm adopted from work done in [16][19].

Keywords: Cloud, Forensic, log, parameterized, event.

1 Introduction

As more enterprises seek to capitalize on the economies of scale of virtual machine compute clouds and their efficiencies, the increase in malicious activity seemingly is waiting to provide even greater opportunities for those who recognize the huge security risk of entrusting the virtual data centres to third party providers for which you have no physical jurisdiction. This security challenge overlaps with the fact that the forensics community also shares it's own concerns for auditing , searching , and providing analysis for victims of miscreant behaviour within this abstract domain. This is particularly true in that a VM object within a data centre may be subject to several eventualities through network distribution before reaching its final user(s).

On the premise of these eventualities, our work introduces the need to look at log event correlation activities within the VM domain to better address the forensic scenarios that are possible. Event correlation is a term that is used to describe an array of techniques applied to comprehend the dynamic behaviour of systems, based on events and patterns of events in their history. Considerable effort has been expended on event correlation in a number of computer security application domains, in particular in the areas of network management and intrusion detection. Events can be primitive or they can be composite. When a composite event occurs, it is possible that many instances of some constituent primitive event occur. The context determines which of these primitive events should be considered for evaluating the composite event.

At the same time, developments in computer forensics have seen an increasing reliance upon event logs generated by computer systems as a source of evidence. We simply adopt in our work on compute cloud forensics these core formalisms. Recent trends have resulted in considerable growth in both the number of sources of event oriented evidence, and the volume of events. This is due to the potential for drawing evidence not only from multiple computers, networks, and electronic systems, but also from disparate personal , organizational and government domains. This potentially leads to very significant improvements in forensic outcomes but is accompanied by an increase in both complexity and scale of the processing involved.

When comparing the number of security related events to the total number of events logged by modern computer systems, we find in practice, security related events comprise only something like 1% of logged information in general. This means that there is a huge amount of event log information (the other 99%) which is not related to security but which is available to the computer forensics investigator for use in identifying activities and events of potential forensic interest. In addition, forensic event correlation may consider event logs from other disparate sources, which are not computer event logs per se, such as electronic door logs, telephone call records, and bank transactions records etc.

In order for forensic investigators within the virtual machine environment to effectively investigate this mass of data, some means of addressing both the conceptual complexity, and the volume of events is becoming a necessity. Methods of representation search and reasoning with the quite heterogeneous semantics implicit within incident oriented evidence from disparate source types is a grave concern. Automated methods of analyzing and correlating these events are needed for forensic investigation to become widely used and cost effective process.

Our motivation for this work is based on the background of exploring methods of forensic investigation within the heterogeneous VM, using event logs [16,17,18,19]. This type of requirement is useful to the detection and notification phase, within a compute cloud forensic framework.[14] We provide for the investigation methods such as human guided search, automated correlation and hypothetical reasoning based on event ordering mechanisms within the Virtual machine server network domain.

Existing approaches to event correlation have focused on single domains of interest only, and have employed models of correlation very specific in nature. Repurposing existing approaches to the task of more general forensics is made difficult for a number of reasons especially in the area of cloud database forensics. Existing specific models underlying event pattern languages don't necessarily generalize to application wider domains. For example, it is well understood that state machine based event pattern languages may work well for events related to protocols, however they don't work well for patterns where time and duration are uncertain. Most approaches focus exclusively on events, but ignore context related information such as environmental data and configuration information. Furthermore, few approaches have available implementations in a form that is readily modifiable.

For the VM environment we need a means to rapidly integrate knowledge from new types of event logs, in a manner that makes explicit the environmental or implicit concepts associated with those logs. This is in order to facilitate synchronization of human understanding, and also automated inference. A general solution is still needed.

In order to generate manageable synchronized VM log events, associating a single context with a complex composite of log events is often times not adequate. Many real world scenarios cannot be expressed if a composite event is associated within a single context.

Our approach is to associate different VM log contexts for various constituent primitive VM events over a distributed compute cloud. We show that this can be done by providing a formal semantics for associating contexts within primitive events.

The authors who have related work in traditional active database environments[16] have identified four different contexts, namely recent, chronicle, continuous, and cumulative that can be associated with a composite event. We simply transcribe these contexts to the VM environment as a basis for evaluating atomic consistency for VM events within these abstract federated networks.

In this work we explore the design of these log events by creating a synchronized VM Oracle log database that maps the virtual services running within the physical data centre. In the interest of space we'll not discuss the prototype here.

In our treatment of this paper we analogize the VM database environment to that of the traditional database management systems. i.e. they are passive: the database system executes commands when requested by the user or application program. However, there are many applications where this passive behaviour is inadequate. Consider for example as in[19], an accounting application: whenever the price of a stock for a company falls below a given threshold, the user must sell his corresponding stocks. One solution is to add monitoring mechanisms in all the relevant application programs. The alternate option is to poll periodically and check the stock prices. Polling too frequently incurs a performance penalty; polling too infrequently may result in not getting the desirable functionalities. A better solution is to use active databases.

Active databases move the reactive behaviour from the application into the database. This reactive capability is provided by triggers also known as event condition action rules. In other words, triggers give active databases the capability to monitor and react to specific circumstances that are relevant to an application. Adopted from [19] the authors argue that an active database system must provide trigger processing capabilities in addition to providing all the functionalities of a passive database system.

Different types of events are often supported by a database application: (1) data modification/ retrieval events caused by database operations, such as insert, delete, update, or select,(ii) transaction events caused by transaction commands, such as, begin , abort , and commit, (iii) Application defined events caused by application programs (iv) temporal events which are raised at some point of time , such as December 31, 2010 at 12:00 p.m. or two hours after the database is updated, and (v) external events that occur outside the database, such as , sensor recording temperature changes. Various mechanisms are needed for detecting these different types of events. The types of events that are of interest depend on the specific application and the underlying data model for these VM clouds. Our synchronized VM log files are able to capture these different types of events, however we do not distinguish the type of events due to lack of space.

An application may be interested in an occurrence of a single VM log event or in a combination of such VM log events. The occurrence of a single log event is referred to as a primitive VM log event. Primitive VM log events are atomic in nature - they cannot be decomposed into constituent events. For example a student gets 95 on a Math test or the temperature reaching 90 degrees Fahrenheit. Sometimes an application is more interested in the occurrence of the combination of multiple VM primitive events using event composition operators. Such events are described as VM composite events. For example, an application may be interested in the event that occurs when the stock prices at Google drop after Facebook . This generally could be seen as an example of a composite of event. It is made up of two primitive events: Google stock market prices drop , and the stock market prices of Face book drop. The event composition operator in this case is the temporal operator “after”.

Many instances of a constituent VM primitive event may occur before the occurrence of a VM composite event. In such cases, we need to identify which VM primitive event(s) to pair up to signal the occurrence of the composite event. An example will help to explain this. Say, that we are interested in detecting the occurrence of the composite event e defined as follows: $e = e_1 ; e_2$. This means that event e occurs whenever primitive event e_2 occurs after primitive event e_1. Suppose e_1 = Google Stock Market prices drop and e_2 = Facebook stock Market prices drop. In such cases, which instance(s) of the event e_1 should we consider in the evaluation of the composite event e. Should we consider the first occurrence of e_1 or the most recent occurrence or both ?

Chakravarthy et.al [6] have solved this problem by formalizing the notion of context. They proposed four kinds of contexts, namely recent, chronicle, continuous and cumulative, can be associated with composite events. Recent context requires that only the recent occurrences of primitive events be considered when evaluating the composite event. In the previous example, if recent context were to be used , then the last occurrence of e_1 will be paired up with the occurrence of e_2 and the composite event e will be signalled once. Chronicle requires that the primitive events be considered in a chronological order when evaluating the composite event. In this context, the composite event will consist of the first occurrence of e_1 followed by the occurrence of e_2 will be signalled only once. Continuous requires that the primitive events in a sliding window be considered when evaluating a composite event. In this case, the composite event will be signalled twice. In other words, there will be two occurrence of the composite event. In the first case, the first occurrence of e_1 will be paired with $e_{2.}$. In the second case, the next occurrence of e_1 will be composed with e_2 . Cumulative requires that all the primitive events be considered when evaluating the composite event. In this case, only one composite event will be signalled. However, it will take into account both the occurrences of $e_{1.}$ Although the four contexts proposed by Chakravarthy et.al [6] can model a wide range of scenarios, they fail to model many situations. Consider for example the situation of managing multiple VMs running within a school environment scenario. The following ECA rule is used in the school event : admit a new student and the admissions administrator enters registration centre, condition: true , and action: alert administrator about student's registration requirements. We assume the scenario described is logged on the local VM server as a coordination of documents that arise from the prescribed events. Here event is a composite one made up of two primitive events E1 = admit student and E2= administrator enters registration centre. We want this rule to be triggered every time a new student is admitted and the admissions administrator enters the registration centre. For E1 we want to use the cumulative context and for event E2 we want the recent context. Such possibilities cannot be expressed by Chakravarthy work as indicated in [16], as the composite event is associated with a single context. By analogy monitoring within the VM environment as a function of a forensic log auditing framework, demonstrate the same characteristics when managing database transaction logs, system logs , error logs and so on.

We argue that associating a single context with a composite event albeit in the physical world or on a virtual machine is restrictive for such environments. More so synchronization for a single context becomes an NP Hard problem given that instances within a time interval (t) are used to influence the occurrence of an event. We see that applications will have composite events made up of different types of primitive events. Often times, the type of event determine which context should be used. For example, it makes sense to use recent context for events based on streaming data, chronicle context for events involving say students or even general customer orders. Since the constituent primitive events in composite events are of different types, requiring them to be associated with the same context is placing unnecessary restrictions on the composite event and prohibiting them from expressing many real world situations.

We adopt the solution in [7] that a single context should be associated with an atomic primitive event. For the virtual machine environment, maintenance of synchronized logged activities between the Physical Data centre and the VM users are predicated on the need to have this type of atomic consistency within the data sets. Hence at a record level each VM user suggested is denoted with his own BLOCKID within the file system. Bearing in mind, that a virtual server maps to its own unique physical MAC address and a logical address called the CPUID. The virtual server with the same CPUID can have many run time user instances and hence several primitive or composite events are likely for each of these server instances. This adopted approach, by observation allows for abstraction of each synchronized VM context separately from the other. At the same time such independence will allow correlation of each VM log context. Equally how different types of primitive events can be associated to form a composite VM log event is also clearly delineated with such an approach. We discuss how this can be done and provide the formal semantics of associating contexts with primitive events for each VM log composition operator. We adopt by proof the expressiveness of the approach. We also adopt algorithms showing how VM composite event detection can take place when the primitive events have varying contexts as a premise to synchronization evaluation within such environments.

The rest of this paper is organized as follows. Section 2 describes related work in this area. Section 3 provides a case study of how we synchronize VM log events, using our own automated of a VM log auditor. Section 4 formally presents our notion of the Synchronized VM log context model. Section 5 provides context definitions for detecting VM composite events. Section 6 provides the conclusion along with pointers to future direction.

2 Related Work

We must admit that the compute cloud offers a powerful abstraction that provides virtualized infrastructure, platform or software as a service where the complexity of fine grained resource management is hidden from the user. In the traditional network storage environment, users often store those data on local storage devices ,while service consumers in public cloud store all of their data in the cloud's common storage pool. The challenges of monitoring the events within these abstract domains are a non trivial problem. Our work represents nascent material on event specification and detection in a synchronized VM hybrid cloud database environment. In our approach we use the hypervisor system logs to capture the events. A number of works [8-13] have been done in traditional event specification and detection in active databases. Some active databases detect events using detection based semantics.[10,11]; whereas others use interval based semantics [8, 9]. Work in the area of parameterized contexts for both traditional and cloud computing environments is very limited based on the published literature available.

In COMPOSE[12,13] and SAMOS [8] systems, the parameters of composite events detected are computed using the unrestricted context. In the unrestricted context, the occurrences of all primitive events are stored and all combinations of primitive events are used to generate composite events. For instance, consider the composite event E = P;Q; Let p1 and p2 be instances of P and q1, q2 be instances of Q. Consider the following sequence of events: p1,p2,q1,q2. This will result in the generation of four composite events in the unrestricted context : (p1,q1),(p2,q1),(p1,q2) and (p2,q2). Unrestricted contexts have two major drawbacks. The first is that not all occurrences may be meaningful from the viewpoint of an application. The other, is the big computation and space overhead associated with the detection of events.

The SNOOP system [8-12] discusses an event specification language for active databases. It classifies the primitive events into database, temporal, and explicit events. Composite events are built up from these primitive events using the following five event constructors; ***disjunction***, ***sequenc***e, ***any***, ***aperiodic, periodic*** and cumulative periodic operators. One important contribution of SNOOP is that it proposes the notion of parameter contexts. The parameter context defines which instance of the primitive events must be used in forming a composite event. The SNOOP authors had proposed four (4) different parameter contexts which were discussed earlier. Although the SNOOP system discussed how to specify consumption of events in four (4) different parameter contexts, a parameter context can be specified only at the top level event expression, which means that the entire composite event is associated with a single context for which we argue otherwise.

From the perspective of reconstruction of a digital crime scene within the compute cloud, the ability to understand event ordering constraints and semantics is clearly a discernible point of note. It is easily argued that within the multi tenant cloud domain, several hybrid composite events exist. The ability to disentangle these events into their composite and atomic types is a relevant synchronization concern and requirement to the digital investigator seeking to provide analysis of time dependent case evidence.

We also embrace the literature by Zimmer and Unland [15], who provide an in depth discussion of the semantics of complex events. They provide a meta model for describing various types of complex events. In a complex event, it is possible that many event instances belonging to a particular type occurs. The event instance selection decides which instance to consider in the composite type. They have the operators first, ***la*****st** and cumulative. Event instance consumption is of three types: ***shared***, ***exclusive***, ***ext-exclusive*** The shared mode doesn't delete any instance of the event. The exclusive mode deletes only those event instances that were used in triggering the composite event. The ext-exclusive deletes all occurrence of the event. Although the Zimmer and Unland provide many different possibilities using the combinations of event instance selection and event instance consumption , their formal semantics are not presented. We leverage the work done by [16] as a basis of our own formalisms for the synchronized event based compute clouds. Moreover, it is critical to use these formalisms to understand the impact of these semantics on the underlying VM log auditor implementation.[17].

3 Case Study Analysis of Our VM Log Synchronization Scenario

We have currently setup a test environment to demonstrate the VM log context events[17]. In this test environment, we designed a log auditor cloud prototype that maps the existing system virtual logs running on the production Vmware essx3i , and hosted on a 40 terabyte Physical Storage Area Network (SAN)disk. Essx3i server instances load as apart of a Windows 7 Operating System. Our case study is coordinated at the University of Technology, Jamaica [UTECH]. Our UTECH private log cloud auditor manages student records along with limited accounting functionalities on campus. For the purposes of our test bed, the UTECH log auditor analyzes the parameters of event log files generated by the production VM servers. We have mapped the existing event disk logs on these Virtual servers and store them in an archival Oracle 11g relational database, which is running on a separate and local test VMware essx3i server host. In summary of our present approach ,we use local ftp sessions over periodic intervals to retrieve and stage data from the production VM SAN disk event logs onto our test VM Oracle 11g database(DB). The data sets on the test Oracle 11g DB are then evaluated by our log auditor for synchronized event activities of the production VMware servers. Seen below in Figure 1.0 is a snapshot of one of our VM log auditor's, log file report that demonstrates the parameterized context of events discussed in this paper.

```
SQL*Loader: Release 11.2.0.1.0 - Production on Fri Jan 21 16:43:55 2011
Copyright (c) 1982, 2009, Oracle and/or its affiliates. All rights reserved.

Control File:
C:\cygwin\home\kfarquharson\data\utechvmlogger\work\vmware_kernel.ctl
Data File:     C:\cygwin\home\kfarquharson\data\utechvmlogger\work\DATA.txt
Bad File:      C:\cygwin\home\kfarquharson\data\utechvmlogger\work\DATA.bad
 Discard File:
C:\cygwin\home\kfarquharson\data\utechvmlogger\work\vmware_kernel_log.log
(Allow all discards)
Number to load:   ALL
Number to skip:   0
Errors allowed:   50000
Bind array:       10000 rows, maximum of 5000000 bytes
Continuation:     none specified
Path used:        Conventional

Table KAVAN.ST_UTECH_SYS_LOG_IMPORT, loaded from every logical record.
Insert option in effect for this table: APPEND
TRAILING NULLCOLS option in effect

   Column Name           Position       Len Term      Encl Data type
 DATE_CODE               FIRST            *  |        CHARACTER
 LOG_TYPE                NEXT             *  |        CHARACTER
 EVENT                   NEXT             *  |        CHARACTER
 LOG_DATE                NEXT             *  |        CHARACTER
 LOG_SOURCE              NEXT             *  |        CHARACTER
 COMPUTERNAME            NEXT             *  |        CHARACTER
 LOG_CATEGORY            NEXT             *  |        CHARACTER
 ACTIVE_USER             NEXT             *  |        CHARACTER
 DESCRIPTION             NEXT             *  |        CHARACTER
 FILENAME                NEXT             *  |        CHARACTER
 DATE_LOADED             NEXT             *  |        CHARACTER
```

```
SQL string for column : "SYSDATE"
value used for ROWS parameter changed from 10000 to 1761
Record 154: Rejected - Error on table KAVAN.ST_UTECH_SYS_LOG_IMPORT,
column DESCRIPTION.
ORA-12899: value too large for column
"KAVAN"."ST_UTECH_SYS_LOG_IMPORT"."DESCRIPTION"    (actual:    223,
maximum: 200)

Record 155: Rejected - Error on table KAVAN.ST_UTECH_SYS_LOG_IMPORT,
column DESCRIPTION.
ORA-12899: value too large for column
"KAVAN"."ST_UTECH_SYS_LOG_IMPORT"."DESCRIPTION"    (actual:    236,
maximum: 200)

Table KAVAN.ST_UTECH_SYS_LOG_IMPORT:
499 Rows successfully loaded.
2 Rows not loaded due to data errors.
0 Rows not loaded because all WHEN clauses were failed.
0 Rows not loaded because all fields were null.

Space allocated for bind array:    4997718 bytes (1761 rows)
Read  buffer bytes: 5000000

Total logical records skipped:        0
Total logical records read:           501
Total logical records rejected:        2
Total logical records discarded:      0

Run began on Fri Jan 21 16:43:55 2011
Run ended on Fri Jan 21 16:43:55 2011

Elapsed time was:     00:00:00.68
CPU time was:         00:00:00.11
```

Fig. 1. A Snapshot of the VM Log auditor's sysLog Event file report

From the above snapshot of our auditor's log file report , we have multiple primitive event set occurrences. Each subsequent log file report when compiled and archived presents multiple composite events about the state of the VM. We seek to use these hypervisor log report files as evidence objects for establishing integrity of these VMs. What we have found particularly useful which has not been available before to the System Administrator is the ability to categorically identify and capture those VM records which have been thrown or rejected by the VM servers. Previously such events go by unnoticed despite their frequency of occurrence. In an independent paper we elaborate on the statistical importance of the event frequency data. Although the mapped log represents a recent event context , our aim is to have the auditor map both recent and continuous event contexts, as apart of the run time environment.

4 Our VM Event Log Synchronized Model

As a basis of VM event synchronization and detection , time can be treated using both a partial or total ordered set of integers. We adopt from [16] the use of temporal intervals and use an interval based semantics to describe our work. We begin by giving a definition of event occurrence and detection and how we apply them to a synchronized VM log event instance. The occurrence of an event typically occurs over a time interval. The detection of an event by our VM log auditor toolkit[17] within a hybrid cloud typically occurs over a time interval, as assumed within the physical environment. Hence the detection of such VM event occurs at a particular point in time albeit a partial order or a total order instance.

Definition 1: [Lamport Partial order of VM log events] - For a set of primitive distributed VM log events running on several hybrid virtual servers we seek to synchronize the log of their activities over a time interval using Lamport timestamps. For every two(2) primitive VM log events say a and b occurring in the same process, and C(x) being the timestamp for a certain event x, it is necessary that C(a) never equals C(b). if a $\rightarrow$bthen *C(a) < C(b)*, where $\rightarrow$means "happened before." A Lamport clock may be used to create a partial causal ordering of the VM log events(both composite and primitive) between processes.

Definition 2: [Vector Partial order of VM log events] - Each time a VM server process experiences an internal primitive event, it increments its own system counter in a vector array by one. The vector clock is an algorithm that allows for the partial causal ordering of VM events and promotes strong consistency within such event context.

As a precursor to Definitions (3-14) we assume that the VM log events like traditional event ordering associations /mappings can be total order relations , as adopted from earlier work by Ray et.al[16]. These arguments are particularly useful and intuitive if we assume that our VM log audited cloud is a private cloud environment. Partial orders otherwise assume both highly distributed public and hybrid data cloud domains.

Definition 3: [Overlap of VM log Events] : Two Events E_i and E_j whose occurrences are denoted by O(E_i [t_{si} t_{ei}] and O(E_j , [t_{sj} , t_{ej}] respectively are said to overlap if the following condition is true : $\exists t_p$ such that ($t_{si} \leq t_p \leq t_{ei} \wedge t_{sj} \leq t_p \leq t_{ej}$). Otherwise the two VM log events are said to be non-overlapping. We posit that this consideration is critical as to prevent duplicate basic primitive events. The overlap relation is reflexive and symmetric but transitive.

Definition 4: [Detection of a VM log Event] The detection of an event E_i is denoted by the predicate D(E_i , t_{di}) where $t_{si} \leq t_{di} \leq t_{ei}$ and t_{si} , t_{di} , t_{ei} represent the start time, detection time , end time of the event E_i respectively. In almost all events(except the composite ones connected by a ternary operator), the detection time is the same as the system notification time. Once the notification is given , we consider the termination of the event. From here onwards we refer to this as the notice termination time and this is denoted by t_{ni} . Hence $t_{ni} = t_{di}$.

Lets assume that the detection time of all events in the system form a total order. This is not a unrealistic assumption , since the detection of an event occurs at an interval in time.

Definition 5: [VM log Event Ordering] We define two ordering relations on events. We say an event E_i occurs after event E_j if $t_{di} > t_{dj}$. We say an event E_n follows event E_m if $t_{sn} > t_{em}$, that is event E_n starts after E_m completes.

Definition 6: [Primitive VM log Event] - A primitive event is an atomic event which cannot be decomposed.

Definition 7: [Composite Event] - A composite event E is an event that is obtained by applying an event composition operator op to a set of constituent events denoted by E_1 , E_2 ....E_n . This is formally denoted as follows E =op(E_1 , E_2 ....E_n) . The VM log event composition operator op may be binary or ternary. The constituent events E_1 , E_2 ....E_n may be primitive or composite events.

Definition 8: **[Occurrence of a VM log Event]** - The occurrence of an event E_i is denoted by the predicate O(E_i (t_{si} , t_{ei}) where $t_{si} \leq t_{ei}$ and t_{si} , t_{ei} denote the start time , end time of E_i respectively. The predicate has the value true when the event E_i has occurred within the time interval [t_{si} , t_{ei}] and is false otherwise. Primitive VM log events are often instantaneous – in such cases the start time t_{si} and t_{ei} ,that is $t_{si} = t_{ei}$. Hence within our VM log synchronized model, events occur over a temporal interval. Since it is not always possible to define a total order on temporal events , we adapt for the synchronized log VM the notion of overlapping and non overlapping events.

Definition 9: [Initiator] - Consider a composite event E =op(E_1 , E_2 ....E_n) that occurs over the time interval [t_s , t_e]. The first detected constituent event instance e_i (i ε $_{[}$1,2,........n]) that starts the process of parameter computation for this composite event is known as the initiator.

Definition 10: [Terminator] Consider a composite event Consider a composite event E = op(E_1 , E_2 ....E_n) that occurs over the time interval [t_s , t_e]. The constituent event instance e_i (i ε $_{[}$1,2,........n]) that detects the occurrence of the composite event E is called the detector.

Definition 11: [Terminator] Consider a composite VM log event E = $E_1 op_1$ E_2 op_2 ......op_x .E_n) that occurs over the time interval [t_s , t_e]. The constituent event instance e_i (i ε $_{[}$1,2,........n]) that ends in time t_e which terminates the composite event E is called the terminator.

The VM log initiator of each event can be associated with different parameter contexts. The contexts specify which instance(s) of the VM log initiator should be paired with a given VM log terminator instance. The terminator instance determines whether the same instance can be used with one or more terminators. The formal definition of the VM log initiator and terminator contexts are given below.

VM Log Initiator in Recent Context – An instance of the initiator event starts the composite event evaluation. Whenever a new instance of the initiator event is detected, it is used for this composite event and the old instance is discarded. After an instance of initiator event has been used for a composite event, it is discarded.

VM Log Initiator Chronicle Context – Every instance of the initiator event starts a new composite event. The oldest initiator event is used for the composite event. The instance of initiator event is discarded after using it for composite event calculation.

VM Log Initiator in Continuous Context - Every new event instance of the initiator starts a composite event after discarding the previous instance of the initiator. An instance of the initiator event can be used multiple times in the composite event evaluation. The same initiator can be paired with multiple terminators. The initiator is discarded only after another initiator event occurs.

VM Log Initiator in Cumulative Context - The first instance of the initiator event starts the composite event . The subsequent occurrences of the initiator event start the composite event. The subsequent occurrences of the VM log initiator events will be used in this same composite event. The instances of initiator events are discarded after use.

VM Log Terminator in Continuous Context - Each terminator can be used multiple times in the composite event calculation. That is, a terminator can be paired with multiple initiators.

VM Log Terminator in Chronicle Context - Each terminator is used only once in the composite event evaluation. A terminator can be paired with only one initiator. Here we do not distinguish the contexts. This is because the very first occurrence of the terminator will terminate the event.

For this paper, we consider only three binary event composition operators, namely disjunction, sequence , and conjunction. In our follow up work we explore negation , and contraposition operations for the semantic.

Disjunction $E_1 \vee E_2$
This VM event is triggered whenever an instance of E_1 or E_2 occurs. Since only one event constitutes the composite event, there is no context associated with this single event. This is because the very first instance of E_1 or E_2 will be the initiator as well as the terminator of this composite event.

Sequence E_1 ; E_2
This VM log event is triggered whenever an instance of E_2 follows an instance of E_1. In this case, an instance of E_1 will be the initiator and an instance of E_2 will be the terminator. Since there may be multiple instances of initiators involved, context

determines which instance of E_1 gets paired with which instance of E_2 will be the terminator. Since there may be multiple instances of initiators involved, context determines which instance of E_1 gets paired with which instance of E_1 gets paired with which instance of E_2 .

Conjunction $E_1 \wedge E_2$
This VM log event is triggered whenever both instances of E_1 and E_2 occur. For composite event with "and" operator $E(A \wedge B)$, either event A or e vent B can be an initiator event or a terminator event. When no instance of event B occurs before any instance of event A, event A is considered as an initiator event and event B is considered as a terminator event. When event B occurs before event A, event B is considered as an initiator event and event A is considered a terminator event.

5 Contextual VM Log Semantic Definitions

In this section we give the formal definition of each operator when different contexts are associated with each constituent event. Due to lack of space, we do not provide the formal semantics for the conjunction operator.

Disjunction Operator E1 $\vee$ E2
For the disjunction operator, the context of the operand is not taken into account for defining the VM log event. This is because the very first event occurrence of either of the events is the initiator as well as the terminator. The operator is commutative.

$$O(E_1 \vee E_2 , [t_s , t_e]) = (O(E_1 , [t_{s1} , t_{e2}]) \wedge (t_s \leq t_{s1} \leq t_{e1} \leq t_e)) \vee (O(E_2 , [t_{s2} , t_{e2}]) \wedge (t_s \leq t_{s2} \leq t_{e2} \leq t_e))$$

Sequence Operator E1: E2
Different contexts will result in different semantics for the sequence operator. Since an initiator can be associated with four(4) different contexts and a terminator can be associated with 2 contexts, we have eight(8) possibilities. Each VM log event occurrence associated with the context is described using the following notation: $O(E1_{Context} ; E2_{Context} , [t_{s1} , t_{e2}])$. Their formal definitions are given below:

$$O(E1_R ; E2_{TC}) , [t_{s1} , t_{e2}]) = \exists t_{e1} , t_{s2} (t_{s1} \leq t_{e1} < t_{e2} \wedge O(E1_R , [t_{s1,} t_{s2}]) \wedge O(E2_{TC} , [t_{s2} , t_{e2}]))$$

VM log detection of "A$\wedge$B" in occurrence say "a1,a2,b1,b2,b3,b4,a3,a4" can be represented as :

$$\begin{aligned} = \exists t_{e1} ,t_{s2} (t_{s1} \leq t_{e1} < t_{s2} \leq t_{e2} \wedge O(E1,[t_{s1} , t_{e1}]) \wedge O(E2,[t_{s2} , t_{e2}]) \\ \wedge (\neg\exists(O(E1' ,[t'_{s1} , t'_{e1}]) \wedge (t_{s1} \leq t'_{s1} \leq t'_{e1}) \wedge (t_{e1} \leq t'_{e1} \leq t'_{s2}))) \\ \wedge (\neg\exists(O(E2' ,[t'_{s2} , t'_{e2}]) \wedge (t_{e1} \leq t'_{s2} \leq t'_{e2}) \wedge (t'_{e2} \leq t_{e2})))) \end{aligned}$$

$O(E1_R ; E2_{TO}) , [t_{s1} , t_{e2}])$
The same as $O(E1_R ; E2_{Tc}) , [t_{s1} , t_{e2}])$

Definition 12: [VM log Event tree] - A VM event log tree ET_e = (N.E) corresponding to a composite VM log event type E is a directed tree where each node E_i represents an event type. The root node corresponds to event type E , the internal nodes represent the constituent composite event types and the leaf nodes correspond to the primitive event types that make up E. The edge (E_i , E_j) signifies that node E_i is a constituent of the composite event E_j .E_i in this case is referred to as the child node and E_j is the parent node.

A VM log event can trigger multiple rules. Thus different event trees can have nodes corresponding to the same VM log event. In such cases to save storage space, the event trees can be merged to form an event graph. An event graph may contain events belonging to different rules. The nodes corresponding to the events which fire one or more rules are labelled d with the rule-ids of the corresponding rule.

Definition 13: [Identical Composite VM log event types] - Two composite event types C = op($E1_{cond1}$, $E2_{cond2}$ **……….,** En_{condn}) and C' = op'($E1'_{cond1'}$, $E2'_{cond2'}$ **……….,** $En'_{condn'}$) are said to be identical if they satisfy the following conditions:

1. The constituent VM events and their associated contexts are identical in both the cases, that is , $Ei_{coni} = Ei'_{coni'}$ for i∈ {1,2,…..n }
2. the constituent events are composed using the same event composition operator, that is op = op' .

The two major VM log event trees can be merged to form a VM log event graph if they have any common nodes.

Definition 14: [VM log Event Graph] - A VM log event graph EG= (N,E) is a directed graph where node E_i represents a log event.and edge (E_i , E_j) indicates that the event corresponding to the node E_i is a constituent of the composite event corresponding to node Ej . Each node E_i is associated with a label. $label_{Ei}$ is a set of rule-ids(possibly empty) that indicate the rules that will fire when the VM log event corresponding to node $label_{Ei}$ happens.

6 Conclusion

As many applications migrate towards a compute cloud, we are already starting to see how real world event processing of the same is riddled with a multiplicity of abstract composite events and primitive events. In this paper we have provided a case study application for which a formal parameterization context can be considered within the virtual machine cloud environment. These formalisms are particularly important if the digital investigator/system administrator is to use these VM database logs as a source of credible forensic case evidence. We have adopted the use of binary operators as the basis of our synchronization parameters for event composition. Future work seeks to focus on expanding these formalisms to look at similarities between event log contexts as a data mining concern within our synchronization model. It is evident that log events as used by the VMs today are rapidly changing given that the area of cloud computing is still an emergent field. We also agree that VM server localities within a distributed public ,private or hybrid cloud influence how we evaluate the parameterized event contexts , albeit a total order or partial order of such events. Further work also examines model checking formalisms that traces the hypervisor log event state behaviour as a proof of correctness for inferences on multiple events.

References

1. Borgida, A., Bracham, R.J., McGuiness, D.L., Resnick, L.A.: CLASSIC - A Structural Data Model of Objects. In: ACM SIGMOD International Conference on Management of Data, Portland, Oregon (May-June 1989)
2. Chen, H., Finin, T., Joshi, A.: An Ontology for Context Aware Pervasive Computing Environments, Adjunct. In: Proceedings of the 6th International Conference on Ubiquitous Computing, Seattle, Washington, October 12-15 (2003)
3. Chen, K., Clark, A., DeVel, O., Mohay, G.: ECF-Event Correlation for Forensics. In: 1st Australian Computer, Network and Information Forensics Conference, Perth, Western Australia, November 25 (2003)
4. Cuppens, F., Miege, A.: Alert Correlation in a Cooperative Intrusion Detection Framework. In: IEEE Symposium on Security and Privacy, Berkley, California, May 12-15 (2002)
5. Doyle, J., Kohane, I., Long, W., Shrobe, H., Szolovits, P.: Event Recognition Beyond Signature and Anomaly. In: IEEE Workshop on Information Assurance and Security, June 5-6. United States Military Academy, West Point (2001)
6. Chakravarthy, S., Krishnaprasad, V., Anwar, E., Kim, S.K.: Composite Events for Active Databases: Semantics, Contexts and Detection. In: The Proceedings of the 20th International Conference on very Large Databases, Santiago de Chile, pp. 606–617 (September 1994)
7. Grandison, T., Maxmillen, M., Thorpe, S., Alba, A.: Formal definition towards Cloud Computing. Proceedings of IEEE Services (2010)
8. Adaikkalavan, R., Chakravarthy, S.: Formalization and Detection of Events Using Interval Based Semantics. In: Proceedings of the 11th International Conference on Management of Data, Goa, India, pp. 58–69 (January 2005)
9. Adaikkalavan, R., Chakravarthy, S.: SnoopIB: Interval based Event Specification and Detection for Active Databases. Data and Knowledge Engineering 59(1), 139–165 (2006)
10. Chakravarthy, S., Krishnaprasad, V., Anwar, E., Kim, S.K.: Composite Events for Active Databases: Semantics, Contexts and Detection. In: Proceedings of the 20th International Conference on Very Large Databases, Santiago de Chile, Chile, pp. 606–617 (September 1994)
11. Chakravarthy, S., Mishra, D.: Snoop: An Expressive Event Specification language for Active Databases. Data and Knowledge Engineering 14(1), 1–26 (1994)
12. Gatziu, S., Dittrich, K.R.: Detecting Composite Events in Active Database Systems Using PetriNets. In: Proceedings of the 4th International Workshop on Research Issues in Data Engineering: Active Database Systems, Houston, TX, pp. 2–9 (February 1994)
13. Gehani, N., Jagadish, H.V., Shmueli, O.: Event Specification in an Active Object Oriented Database. In: Proceedings of the ACM SIGMOD International Conference on Management of Data, pp. 81–90. ACM Press, San Diego (1992)
14. Thorpe, S., Ray, I.: Compute Cloud Forensic Framework (unpublished)

15. Zimmer, D., Unland, R.: On the Semantics of Complex Events in Active Database Management Systems. In: Proceedings of the 15th International Conference on Data Engineering, pp. 392–399. IEEE Computer Society Press, Sydney (1999)
16. Ray, I., Huang, W.: Event Detection in Multi level Secure Databases. In: Proceedings of the 1st International Conference on Secure Systems, Kolkata, India (December 2005)
17. Thorpe, S., Farquharson, K.: Design of an enterprise VM log auditor tool kit. In: Proceedings of the 1st International Forensic Conference and Journal of Arts and Technology. University of Technology, Kingston (2011)
18. Thorpe, S., Ray, I., Ray, I., Grandison, T., Barbir, A.: Global Virtual Machine Auditor for a Federated Digital Identity. Proceedings of the Journal of Information Assurance and Security 6(4), 322–330 (2011)
19. Ray, I., Huang, W.: Increasing Expressiveness of Composite Events Using Parameter Contexts. In: Atzeni, P., Caplinskas, A., Jaakkola, H. (eds.) ADBIS 2008. LNCS, vol. 5207, pp. 215–230. Springer, Heidelberg (2008)