1-1-2011

CAT detect (computer activity timeline detection): A tool for detecting inconsistency in computer activity timelines

Andrew Marrington
Zayed University

Ibrahim Baggili
Zayed University

George Mohay
Queensland University of Technology QUT

Andrew Clark
Queensland University of Technology QUT

available at www.sciencedirect.com

ScienceDirect

journal homepage: www.elsevier.com/locate/diin

Digital Investigation

CAT Detect (Computer Activity Timeline Detection): A tool for detecting inconsistency in computer activity timelines

Andrew Marrington [a,*], *Ibrahim Baggili* [a], *George Mohay* [b], *Andrew Clark* [b]

[a] *College of Information Technology, Zayed University, United Arab Emirates*
[b] *Information Security Institute, Queensland University of Technology, Australia*

Keywords:
Timeline inconsistency
Event correlation
Precondition event
Happened-before
CAT detect

ABSTRACT

The construction of timelines of computer activity is a part of many digital investigations. These timelines of events are composed of traces of historical activity drawn from system logs and potentially from evidence of events found in the 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 (CAT Detect) for the detection of inconsistency within timelines of computer activity. We examine the impact of deliberate tampering through experiments conducted with our prototype software tool. Based on the results of these experiments, we discuss techniques which can be employed to deal with such temporal inconsistencies.

1. Introduction

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

The most common inconsistency in digital evidence is naturally occurring, that is to say, inconsistency which is not the result of deliberate tampering. This includes data pertaining to an event or file which simply is not recorded, or may have been over-written during the normal operation of the computer system. It also includes "naturally" erroneous or inaccurate data, perhaps due to a hardware characteristic, software misconfiguration or bug. Such natural inconsistencies pose difficulties for 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 behaviour of computer hardware clocks (that is to say clock skew and drift) will cause inconsistency between the different time sources. Schatz et al. discuss an approach for dealing with such inconsistencies. Their approach baselines the behaviour of inconsistent computer clocks via correlation with records from devices with more authoritative timestamps (Schatz et al., 2006). In 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

* *Corresponding author.*
E-mail addresses: andrew.Marrington@zu.ac.ae (A. Marrington), ibrahim.baggili@zu.ac.ae (I. Baggili), g.mohay@qut.edu.au (G. Mohay), a.clark@qut.edu.au (A. Clark).
1742-2876/$ – see front matter © 2011 Marrington, Baggili, Mohay & Clark. Published by Elsevier Ltd. All rights reserved.
doi:10.1016/j.diin.2011.05.007

extreme cases – if there is a clock reset at 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 computer records to obscure records of suspicious activity creates another, and generally more concerning, sort of inconsistency in 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 organised as follows. Section 2 introduces related work which informed our research. Section 3 examines approaches for the detection of inconsistency in timelines, dealing both with inconsistencies in event timestamps and events omitted from the system's record. Section 4 describes our experiments with our tool for testing the approaches discussed in Section 3. Section 5 describes the results of those experiments and evaluates the detection techniques. In Section 6 we list the limitations of the CAT Detect tool at the time of writing this paper, as well as limitations of the research described in this paper. Section 7 is a discussion of future work in the area of detecting inconsistency in computer activity timelines, and Section 8 is our conclusion.

2. Related work

This work employs some of the concepts from the computer profiling model described by Marrington et al. (2010). 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, there are 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 organisations 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 computer system. 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.

The model also includes the set of all times in the history of the computer system, T, 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, x be the object which instigated the event, y be the object which was the target of the event, ε describes the action of the event, and α describe the result of the event (either successful, unsuccessful, or unknown). An event evt in the set Evt consists of the quintuple:

$$evt = (t, x \in O, y \in O, \varepsilon, \alpha).$$

In the model, the finite set Evt 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[1] ($EvtR$) 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 artefact 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 an element 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 automated detection of inaccuracy or incompleteness leading to chronological inconsistency in timelines of computer activity.

In another work, Marrington et al. (2007) 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 time-lined 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

[1] Marrington et al. (2007) used the term "discovered events" instead of "recorded events". We prefer the latter term as it more accurately describes the nature of such events.

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 minimise 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 (Boyd and Forster, 2004; Nolan et al., 2005). 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 behaviour (Buchholz and Tjaden, 2007). Willassen (2008a,b, 2009) 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 clock hypotheses using the output of our tool.

3. Detecting inconsistency in timelines

This section describes the approaches our tool employs to detect inconsistency in timelines. Inconsistency in computer activity timelines can arise because events in the timeline are out of sequence, or because 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 Windows Event Logs (our tool is intended for the examination of Windows computers, although the approach could easily be adapted to other operating systems). Each event in each of the three logs is normalised 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 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 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 events are normalised according to the same pattern as recorded events extracted from logs. The tool then has enough data to both construct a computer activity timeline and to test it for internal inconsistency.

3.1. Detecting out-of-sequence events

It is self-evident that there are some events which can only take place after some other another event. This sort of relation is described by Lamport (1978) as the *happened-before* relation. Gladyshev and Patel (2005) discuss the application of the *happened-before* relation to a forensic context. An example of such a relation (represented by $\rightarrow$) between two events would be that a user x must log into the computer system before the user x can execute the application y. Applied to computer activity time-lining, the real time, if not the timestamp, of the execution event must be after the real time of the login event. Let $x \in P$, $y \in A$, $t_n \in T$ and $t_m \in T$:

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

After the construction of a timeline (which is a sequence over the set Evt) in the tool's execution process, an evaluation can be applied to all events ordered by their timestamp. If an event evt_A has a *happened-before* relation to evt_B, but the timestamp (t_B) of evt_B suggests that evt_B occurred before evt_A then we can say that t_A and t_B are inconsistent. In order to detect this inconsistency, a rules base must be created which describes the *happened-before* relations for the various types of events. When the timeline is evaluated against the rules base, the inconsistent events can be identified and assertions about their timestamps can be made. Consider two rules:

$$evt_A \rightarrow evt_B$$
$$evt_B \rightarrow evt_C$$

Where x is a User object, a is an Application object, and *system* is a System object representing the target computer system itself, and:

$$evt_A = (t_A \in T, x, system, \text{login}, \text{success})$$
$$evt_B = (t_B \in T, x, a, \text{execute}, \alpha \in \{\text{success}, \text{fail}, \text{unknown}\})$$
$$evt_C = (t_C \in T, x, system, \text{logout}, \text{success}).$$

Note that the *happened-before* relation is transitive (Lamport, 1978; Gladyshev and Patel, 2005):

$$(evt_A \rightarrow evt_B) \wedge (evt_B \rightarrow evt_C) \Rightarrow evt_A \rightarrow evt_C.$$

For the purposes of this example, let the time-lining function $H(x)$ produce a timeline corresponding to a single 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 evt_B occurs, the login event evt_A must *happen-before* it, and evt_B must *happen-before* the logout event evt_C. Therefore the physical time t_C at which the event evt_C must have occurred must be after the physical time t_B at which the event evt_B must have occurred, which must in turn be after the physical time t_A at which the event evt_A must have occurred. This is stated:

$H(x) \supseteq \{evt_A, evt_B, evt_C\} \Rightarrow (t_C > t_B > t_A)$.

If, given the two rules $evt_A \rightarrow evt_B$ and $evt_B \rightarrow evt_C$, it is not the case that $t_C > t_B > t_A$, then the timestamps t_A, t_B, and t_C do not reflect the physical times at which the events must have occurred. The 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 events is apparent. A hypothesised chronology of a computer system can be evaluated for internal inconsistencies by testing the hypothesised sequence of events against a set of *happened-before* rules.

3.2. Detecting missing events

There are some *happened-before* relations where the first event is a precondition for the second. In such relations, the presence of the second event necessarily implies the presence of the first. In the example in Section 3.1, the login event evt_A must occur before the application execution event evt_B, such that if evt_B occurred, then evt_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 evt_B must *happen-before* the logout event evt_C in order for evt_B to happen at all, the occurrence of the logout event evt_C does not imply that evt_B also happened. This is because evt_B is not a precondition for evt_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.

$$(evt_A \rightarrow evt_B) \wedge (\text{happened}(evt_B) \Rightarrow \text{happened}(evt_A)) \\ \therefore \text{precondition}(evt_A, evt_B).$$

Willassen (2008a,b) extends the use of the *happened-before* relation of Lamport (1978), Fidge (1991) and Gladyshev and Patel (2005) to imply causality. Willassen's version of the *happened-before* relation is therefore equivalent to the "precondition" predicate. For the purposes of the tool we developed, it is preferable to maintain the *happened-before* relation as described by Lamport (1978), Fidge (1991) and Gladyshev and Patel (2005), 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 event evt_A which "happened" does not exist in either the set of recorded events *EvtR* or the existing set of inferred events *EvtI*, then it is a missing event. It is a missing event because it was removed from or never recorded in the computer system's logs, and it was not previously inferred on the basis of relationships and object fields. These events could also be called inferred events, but it is convenient to preserve a distinction between events detected using this approach and other inferred events.

The rules base in the example in Section 3.1 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 3.1, is:

$$evt_A \rightarrow evt_B \\ evt_B \rightarrow evt_C \\ \text{precondition}(evt_A, evt_B)$$

The login event evt_A, the application execution event evt_B, and the logout event evt_C have the same definitions as in the previous example. The new rule states that if the event evt_B occurred in the timeline of the User object x, then the event evt_A must also have occurred. This is expressed:

$$(evt_B \in H(x)) \Rightarrow \text{happened}(evt_A) \\ \therefore evt_A \in EVT \\ \therefore (evt_A \notin (EVTR \cup EVTI)) \Rightarrow evt_A \in EVTM.$$

Detecting missing events is important, as such an event may have been deliberately deleted from system logs, which may in itself be suspicious. Detecting that an event is missing allows for the construction of a more complete timeline, helping the investigator gain a more complete understanding of the 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 event record was deliberately deleted, this may indicate that the user was attempting to conceal suspicious activity.

There are, of course, many instances where an event may be missing as a result of non-suspicious computer activity. Our tool infers events to describe an action by or on an object with associated temporal data. These inferred events are combined with events recorded in system logs in order to provide as complete a timeline as possible. In our experiments on computers running Microsoft Windows, our tool inferred many events which occurred prior to the enabling of many logging options in the Windows event logs. There were therefore very few recorded events from that early time period in the computer's history, and thus these inferred 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 investigator's assessment as to whether or not the event is suspicious. This scenario is an example of how the normal configuration of the computer system may make an event seem inconsistent.

4. Detection experiments

This section describes experiments in which the approach to detecting temporal inconsistency in user sessions described in Section 3 was tested. We examine timelines as developed by our prototype software in the following experiments:

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

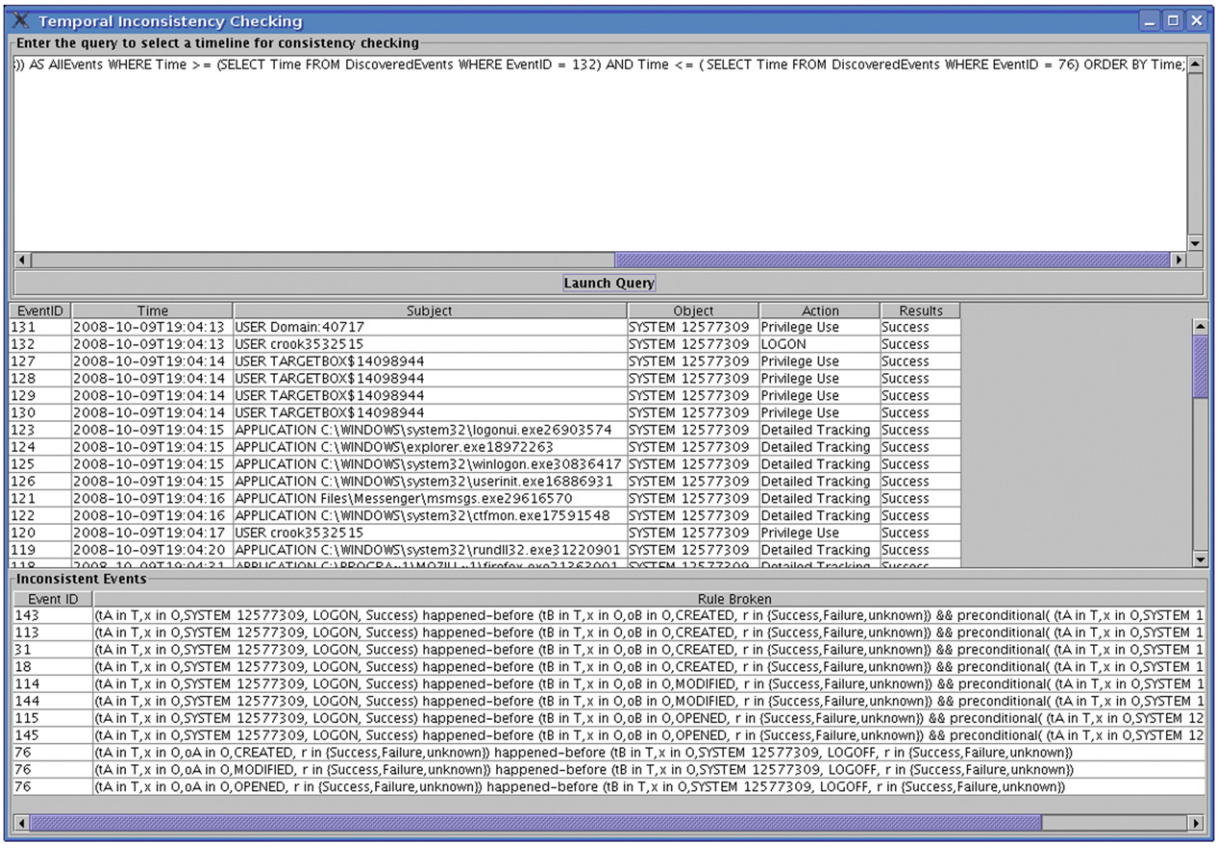

Fig. 1 – CAT Detect prototype for inconsistency checking.

- The unmodified timeline of a user session during which the user creates a document with deliberately misleading authorship information.
- Modified timelines of the above two user sessions where the system logs have been tampered with.

4.1. *Prototype software*

As mentioned in previous sections, we developed CAT Detect in order to detect inconsistency in computer activity timelines. The prototype software examines the target computer system's file system (which is mounted read-only) and enumerates the applications, files, and users of the target computer system. The Windows Event Logs are parsed, and the events described in those logs are stored as the set of recorded events (*EvtR*) in the database table *Recorded Events*. Finally, a set of events are inferred from the temporal data associated with each file. These events are the inferred 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 Fig 1.

The detection techniques described in Section 3 match the events in a timeline against the events in each rule being tested (as listed in Section 4.2). Programmatically, every rule is implemented by a Java object[2], and every event is implemented by a Java object. Rule objects have two event objects as fields, one called evt_A and another called evt_B. The objects evt_A and evt_B are archetype events, against which real events are compared. A real 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 real 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 real 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 events have the same subject or target field. The rule can also specify that 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. Consider the following example rule, which expresses the concept that a user object ($u \in PIU$) must log into the computer system ($s \in S$) before modifying any file:

$$\text{precondition}\left(\begin{array}{l}(t_i \in T, u, s, \text{logon}, \text{success}),\\ (t_k \in T, u, c \in C, \text{modified}, \text{success})\end{array}\right).$$

[2] We anticipate that in future versions, rules will be user-specifiable either through a graphical interface or XML configuration file. In the version used for the experiments described here, rules were hard-coded.

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

$evt_A = (\text{null}, \text{null}, s, \text{logon}, \text{success})$
$evt_B = (\text{null}, \text{null}, \text{null}, \text{modified}, \text{success}).$

The prototype CAT Detect software does not yet implement the concept of a user session. A logon or logoff 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 (Fig. 2).

4.2. Rules base for experiments

The software prototype incorporates a small set of rules to check for temporal inconsistency. It provides an interface which allows the user to specify a timeline to be checked for inconsistency. It then checks that timeline against the rules base. The rules built into the prototype software for the purposes of these experiments are as follows:

precondition(userlogin, userlogout)
precondition(userlogin, filecreated)
precondition(userlogin, filemodified)
precondition(userlogin, filemodified)
filecreated → userlogout
filemodified → userlogout
fileaccessed → userlogout

Where x is a Principal object representing the user, y is a Content object representing a file, *system* is the System object which represents the computer system, t_A through t_E are times in the history of the computer, and:

$\text{userlogin} = (t_A, x, \textit{system}, \text{logon}, \text{success})$
$\text{userlogout} = (t_B, x, \textit{system}, \text{logoff}, \text{success})$
$\text{filecreated} = (t_C, x, y, \text{created}, \text{success})$
$\text{filemodified} = (t_D, x, y, \text{modified}, \text{success})$
$\text{fileaccessed} = (t_E, x, y, \text{opened}, \text{success}).$

The data structures in our implementation which represented each of the archetype events in the rules base had null values in place of the fields x, y and t_A through t_E. As discussed in Section 4.1, null values are wild card values in our prototype software. Each rule had a Boolean field set to true, which specified that the subject of every event, x, had to be the same.

```
evtA = (null, null, s, "logon", "success")
evtB = (null, null, null, "modified", "success")

rule = evtA happened-before evtB
        where field 2 of evtA == x
        and where field 2 of evtB == x

for each evt in H(x)
        if evt = ( *, x, s, "logon", "success" )
                a = index of evt
        if evt = ( *, x, *, "modified", "success" )
                b = index of evt
next evt

if a > b then
        rule has been broken
```

Fig. 2 – Example inconsistency detection.

4.3. Data

In order to obtain data for these experiments, we employed a suspect test computer running Windows XP. All system logging options were turned on in order to give us as complete a set of Windows Event Logs as possible. We logged onto the computer twice for the purpose of generating two different 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 wanted to explore the detection of meddling with Windows Event Logs. For this purpose, we copied the case file and database about the test computer system generated by our tool, and then manually modified the database table containing the discovered events. As these discovered events are derived from the Windows Event Logs, the removal or modification of recorded events in the set *EvtR* effectively simulates the removal or modification of event records in the Windows Event Logs. We removed the log-on/log-off 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. The modified timelines are described below.

5. Evaluation of detection technique

This section describes each of the timelines examined in these experiments, and presents the results of the prototype software's analysis of inconsistency. There are four timelines (two unmodified, and two modified) which correspond directly to user sessions. Each of the timelines is a combination of the inferred events and the recorded events in the history of the computer system between two boundary events, ordered by timestamp. In regards to the inferred events, it should be noted that the software assumes that people can be assumed to be unique by their name. This means that when the tool extracted the author name "baddie" from some Microsoft Word documents on the target computer, the tool assumed that this person was the same as the user "baddie".

5.1. Timeline A: normal user session

Timeline A was a normal user session during which a Microsoft Word document was created. The user "baddie" logged into the computer system at 6:47pm on 9 October 2008, and created the file "invoice.doc" at 6:51pm. The user then browsed the Internet for a few minutes and logged off at 6:59pm. Nothing suspicious happened in the user session. The timeline consisted of all of the events which took place during the user session, both recorded and inferred. Our software inserted these events into its event database during its automated examination of the target system.

Table 1 – The inconsistent events detected in timeline B and the rules they violated.

Time	Subject	Target	Action	Rule
9/10/08 20:13:00	USER baddie27660658	WORDDOC letter from baddie to nefarious.doc14850080	CREATED	precondition(userlogin, filecreated)
9/10/08 20:13:00	USER baddie27660658	WORDDOC Normal.dot20348456	CREATED	precondition(userlogin,filecreated)
9/10/08 20:15:21	USER baddie27660658	WORDDOC letter from baddie to nefarious.doc14850080	MODIFIED	precondition(userlogin, filemodified)
9/10/08 20:15:21	USER baddie27660658	WORDDOC letter from baddie to nefarious.doc14850080	OPENED	precondition(userlogin, fileaccessed)
9/10/08 20:15:23	USER baddie27660658	WORDDOC letter from baddie to nefarious.doc14850080	CREATED	precondition(userlogin, filecreated)
9/10/08 20:15:23	USER baddie27660658	WORDDOC Normal.dot20348456	CREATED	precondition(userlogin, filecreated)
9/10/08 20:15:23	USER baddie27660658	WORDDOC Normal.dot20348456	MODIFIED	precondition(userlogin, filemodified)
9/10/08 20:15:23	USER baddie27660658	WORDDOC Normal.dot20348456	OPENED	precondition(userlogin, fileaccessed)

Most events in timeline A were discovered events (i.e. discovered in the Windows Event Logs), 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).

An inconsistency check of timeline A against the rules provided in Section 4.2 demonstrated no inconsistencies. The results of the analysis of timeline A were as expected.

5.2. *Timeline B: deliberate misattribution of authorship*

Timeline B was a user session during which the user created a Microsoft Word document with misleading authorship information, in an effort to shift responsibility for that document to an innocent third party. The user "crook" logged into the computer system at 7:04pm on 9 October 2008, and at 8:15pm a Word document was created with "baddie" as the listed author. The user "crook" then logged off.

Timeline B was analysed for inconsistency with our prototype software. Table 1 shows the inconsistent events detected in this timeline along with the rule from our rules base which were broken by each event. These events all related to the authorship of the Word document "WORDDOC letter from baddie to nefarious.doc14850080". The "baddie" user was not logged in at the time the Word document was created, and yet the author field listed "baddie" as the document's author. Therefore, "baddie" could not have been the author of "WORDDOC letter from baddie to nefarious.doc14850080".

It can be seen in Table 1 that there are two sets of "CREATED" events for both the suspect Word document and its template. This is because there are two sources of information which lead the prototype software to inferring such an event. The earlier timestamp is obtained from the Word document's metadata, and is the time at which the document was first created in Microsoft Word. 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" events derive their subject field from the same source, the Word document's author field.

5.3. *Timeline C: user session with logon/logoff events deleted*

Timeline C was derived from timeline A. The recorded and inferred events table in the prototype software's events database were copied and manually modified. The resulting timeline, timeline C, was identical to timeline A without the logon/logoff events. The removal of these two discovered events left user activity outside of a logon/logoff-bound user session.

The prototype software's temporal inconsistency check listed all of the inferred events with "USER baddie27660658" as the subject as inconsistent. These events were all listed as inconsistent on the basis of violating precondition rules with a user login event as the precondition. The inconsistent events from timeline C are listed in Table 2. These results were as expected. This demonstrates that removing user session information from the Windows Event Log will draw attention to the inferred events which took place during the session.

5.4. *Timeline D: user session with modified timestamps*

Timeline D was derived from timeline A, with the timestamp of the user's logoff event deliberately modified so as to appear to have taken place prior to the creation of the "WORDDOC invoice.doc19509473" document. The timestamp of "USER

Table 2 – Inconsistent events detected in timeline C, as a result of the login precondition not being met.

Time	Subject	Target	Action	Rule
9/10/2008 18:50:46	USER baddie27660658	WORDDOC invoice.doc19509473	MODIFIED	precondition(userlogin, filemodified)
9/10/2008 18:50:46	USER baddie27660658	WORDDOC invoice.doc19509473	OPENED	precondition(userlogin, fileaccessed)
9/10/2008 18:51:49	USER baddie27660658	WORDDOC Normal.dot3981922	CREATED	precondition(userlogin, filecreated)
9/10/2008 18:51:49	USER baddie27660658	WORDDOC Normal.dot3981922	MODIFIED	precondition(userlogin, filemodified)
9/10/2008 18:51:49	USER baddie27660658	WORDDOC Normal.dot3981922	OPENED	precondition(userlogin, fileaccessed)
9/10/2008 18:51:49	USER baddie27660658	WORDDOC invoice.doc19509473	CREATED	precondition(userlogin, filecreated)

baddie27660658"'s logoff was changed from 18:59:37pm to 18:51:23pm.

The prototype software's inconsistency check of timeline D listed "USER baddie27660658"'s logoff event as inconsistent, as shown in Table 3. The event was listed as breaking three rules, all of which ultimately 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 computer system.

The results of the analysis of timeline D were just as expected. The detection of this event demonstrates the suitability of this approach to detecting events whose timestamps are modified.

5.5. *Discussion of results*

The results of the experiments demonstrate that automatically detecting temporal inconsistency in computer activity timelines constructed from realistic data is possible using our tool. These experiments applied a simple rules set to a 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 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 file system. Timeline C showed that the deletion of system log entries pertaining to important 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, events can be detected which should have occurred in another sequence than their timestamps suggest.

The experiment's use of data from a computer system demonstrated that this approach to detecting temporal inconsistency is robust enough to be tested in real cases. The logical next step will be to perform experiments with CAT Detect using real case data, which will test the robustness and suitability of the approach with regards to real digital investigations. The noise in real event data is a lesser problem to a software tool than it is to a human investigator. By distilling event records down to the most important fields which are common to most events, our approach reduces the complexity and heterogeneity of the various types of events. This makes the testing of a set of simple logical predicates (such as the rules base employed in the experiments, described in Section 4.2) against a timeline of recorded and inferred events relatively straightforward. The results of these experiments demonstrate that this method of testing for inconsistency in timelines is effective in practical computer systems. Further experimentation, as noted in the future work section below, will be necessary to determine whether this effectiveness extends to real investigations.

Table 3 – The inconsistent event in timeline D, which was detected on the basis of breaking three rules. The target of the event is the system.

Time	Subject	Action	Rules
9/10/2008 18:51:23	USER baddie 27660658	LOGOFF	filecreated → userlogout, filemodified → userlogout, fileaccessed → userlogout

6. Limitations

As acknowledged at several points throughout this paper, the CAT Detect software has several limitations. We hope that these limitations will be addressed in future versions of the CAT Detect software.

The most serious of these limitations is the CAT Detect prototype software's inability to automatically detect user sessions. This requires the user to provide the boundaries (i.e. first and last 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 event under investigation. If this limitation could be overcome and CAT Detect could identify user sessions itself, then it could be used to assess entire 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 "case". Although CAT Detect performed well in this case, we are not yet able to validate its robustness or reliability with respect to real cases. Further, our experiments using the data from the test machine were limited in their extent only to the operating system and software installed on that machine (Windows XP, Microsoft Office 2007, and other common "office computer" software). Results with newer versions of Windows or non-Windows operating systems may vary. Further testing with different data sets is required. This further testing will allow investigators to establish confidence in the CAT Detect tool.

It is hoped that these limitations will be addressed by the future work described in Section 7 below, and through the release of the CAT Detect prototype as free and open source software. We hope that publicly releasing CAT Detect as open source software will achieve two things. First, we hope that CAT Detect will attract a community of users who will use it in a variety of cases and provide feedback. Second, we hope that CAT Detect will attract contributions from developers and researchers to address the tool's shortcomings and improve upon its functionality.

7. Future work

CAT Detect is still in the research in progress phase. There are five main areas in which we hope to improve CAT Detect. Primarily, we hope to create an interface in which rules for inconsistency can be created and saved in a configuration file. These rules can then be shared amongst investigator communities so they can use them during their investigative process using the CAT Detect tool.

Second, we hope to build an automatic Windows Event Log parser into CAT Detect to speed-up the overall process of acquiring Event Log data, and improve the quality of this data. This is primarily a development activity as opposed to a research one.

Third, we hope to improve CAT Detect so that the software can automatically detect user sessions. At the moment, the prototype software requires the user to specify the bounds (i.e. start and finish) of a user session before it is able to check the timeline of that session for internal consistency.

Fourth, we hope to extend the CAT Detect process and software to construct and consistency-check timelines of computers running non-Windows operating systems. We also would like to test and refine CAT Detect with data from Windows Vista and Windows 7 computers, and compare the results compared to otherwise similar cases where the computer involved ran Windows XP.

Lastly, and perhaps most importantly, in order to further validate and improve CAT Detect, it is important to conduct experiments on known cases which involve some timeline inconsistencies. We are particularly interested to test the robustness of the CAT Detect approach in real cases involving deliberate tampering. This can help validate our proposed method with relation to real-life scenarios.

8. Conclusion

Inconsistencies in a 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 computer system uncritically, a time-lining tool may produce a history of the computer system which is unusable as a result of inaccuracy. Perhaps worse, the investigator may fall victim to an adversary's deliberate modification of system logs and other temporal data, and create a misleading history of the adversary's own devising.

We have developed a tool, implementing techniques for detecting contradictory and missing 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 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 system.

REFERENCES

Boyd C, Forster P. Time and date issues in forensic computing – a case study. Digital Investigation; 2004:18–23.

Buchholz F, Tjaden B. A brief study of time. Digital Investigation 2007;4:31–42.

Fidge C. Logical time in distributed computing systems. Computer 1991;vol. 24:28–33.

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

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

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.

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.

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

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 workshop (DFRWS '06), vol. 3; 2006/9. p. 98–107.

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

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.

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

Dr. Andrew Marrington is an Assistant Professor at the College of Information Technology at Zayed University. Digital forensics is his primary field of research, although he is also interested in other aspects of information security. He is the primary researcher in the Advanced Cyber Forensics Research Laboratory at Zayed University's Dubai campus. He teaches in a range of forensics, security and general computer science courses at the undergraduate and graduate level. Prior to taking up his present position, Andrew was a Research Fellow at the Information Security Institute at Queensland University of Technology.

Dr. Ibrahim Baggili received his PhD with emphasis in Information Assurance and Cyber Forensics at Purdue University, West Lafayette, IN, USA. He was also a researcher at CERIAS (The Center for Educational Research and Education in Information Assurance). Currently, Ibrahim Baggili is an Assistant Professor at Zayed University in Abu Dhabi, UAE, at the College of Information Technology, at which he established and directs the first Advanced Cyber Forensics Research Laboratory in the Arab world. He consults, conducts research, trains and teaches in the areas of Information Assurance, Computer Forensics, Small Scale Digital Device Forensics, Penetration Testing and Network Forensics. Ibrahim has trained and consulted in both the private and public sectors. Besides his research interests in technical matters, he is also keen on studying the social and psychological aspects of cyber crime. Ibrahim is also on the editorial boards of journals. He has also held the general chair position of the International Conference on Digital Forensics and Cyber Crime (ICDF2C – www.d-forensics.org) in 2010, and is now on the steering committee of that international conference. To learn more about Ibrahim's work, and to get in touch with him you can visit http://baggili.weebly.com.

Dr. George Mohay is an Adjunct Professor in the Information Security Institute at the Queensland University of Technology,

Brisbane, Australia. Prior to this he was Head of the School of Computing Science and Software Engineering from 1992 to 2002. His current research interests lie in the areas of computer security, intrusion detection, and computer forensics. He has worked as a visiting researcher while on sabbatical leave at Stanford University in 1981, Loughborough University in 1986, Bristol University in 1990 and the Australian National University in 2000. He graduated BSc (Hons) (UWA) in 1966 and PhD (Monash) in 1970, and is a member of the ACM and of the IEEE Computer Society. He supervises PhD and Masters students in the above areas and is involved as chief investigator in a number of related funded research projects. His publications include the book Computer and Intrusion Forensics. He is a program committee member for a number of international conferences.

Dr. Andrew Clark is an Adjunct Professor at the Information Security Institute at Queensland University of Technology (QUT). Prior to his becoming adjunct, he was a Deputy Director of QUT's Information Security Institute where he led a team of researchers investigating various aspects of network security and computer forensics. His current research interests lie in the fields of intrusion detection, fraud detection and network forensics. He is the author of over 70 academic publications covering various aspects of information security and is currently supervising numerous postgraduate research students in related areas.