Dealing with Temporal Inconsistency in Automated Computer Forensic Profiling

Andrew Marrington, George Mohay, Andrew Clark, Hasmukh Morarji
Information Security Institute, Queensland University of Technology

Abstract. Computer profiling is the automated forensic examination of a computer system in order to provide a human investigator with a characterisation of the activities that have taken place on that system. As part of this process, the logical components of the computer system – components such as users, files and applications - are enumerated and the relationships between them discovered and reported. This information is enriched with traces of historical activity drawn from system logs and 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 examines the impact of temporal inconsistency in such information and discusses two types of temporal inconsistency that may arise – inconsistency arising out of the normal errant behaviour of a computer system, and inconsistency arising out of deliberate tampering by a suspect – and techniques for dealing with inconsistencies of the latter kind. We examine the impact of deliberate tampering through experiments conducted with prototype computer profiling software. Based on the results of these experiments, we discuss techniques which can be employed in computer profiling to deal with such temporal inconsistencies.

Keywords: Computer profiling, digital forensics, digital evidence, event correlation, precondition event, happened-before.

1 Introduction

As the volume of digital evidence which must be examined in a computer forensic investigation increases, so too does the investment of a human forensic examiner's time and energy into that investigation. Marrington *et al* [1, 2] introduced an automated process (*computer profiling*) to conduct a forensic reconstruction of a computer system. The output of this process, the computer profile, allows a human examiner to make an informed decision regarding the likely value of the computer system to an investigation before undertaking a detailed manual forensic examination. In this paper, we consider the issue of temporal inconsistencies in digital evidence, and their impact on the computer profiling process and its output. 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.

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 [3]. In single-computer investigations, such as the automated profiling of a single computer, clock drift and skew can still lead to inconsistent timestamps in the evidence. Despite the fact that there is only one clock providing the timestamps in a single-computer system investigation,

severe cases of clock drift and skew can cause the timelines that are constructed to be misleading. In extreme cases – if there is a clock reset at reboot or some other mishap and time "goes backwards" – then events may appear out of the sequence in which they actually occurred.

In addition to inconsistency, incompleteness in digital evidence also poses a challenge. Automated processes such as computer profiling must be able to "fill-in" missing information which is necessary for the understanding of the data which does exist. As an example of incompleteness in digital evidence commonly caused by the normal operation of the computer system, consider a computer system which only records user login/logoff information going back six months. In a computer profile of such a computer system, a file created eight months ago will appear to have been created outside of a user session. To a human investigator this is easily explained, but in an automated time-lining process, the creation of the file may appear to be anomalous. This incompleteness is not suspicious as it is the result of normal operation of the computer system, not a deliberate attempt to obscure evidence.

The deliberate modification of computer records to obscure records of suspicious activity creates a second, and generally more concerning, 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, and event log records showing their login, opening the browser application, and logoff. 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. For an automated time lining process, deliberate alteration of evidence can appear very similar to the innocuous example above.

The rest of this paper is organised as follows. Section 2 summarizes computer profiling. 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 for testing the approaches discussed in Section 3. Section 5 describes the results of those experiments and evaluates the detection techniques. Section 6 is a detailed discussion of future work in the area of detecting inconsistency in computer profiling, and Section 7 is our conclusion.

2 Computer Profiling

The volume and heterogeneity of digital evidence which might be found on a computer system poses interrelated challenges to digital investigations which Carrier refers to as the *Quantity Problem* and *Complexity Problem* respectively [4]. These problems mean that it is desirable for forensic investigators to be able to narrow the scope of an investigation so that manual investigative effort can be applied in the most efficient and effective manner. An automated computer forensics process, designed to be run prior to a detailed manual investigation, can address the Quantity Problem by characterising the computer system's usage and discovering relationships between files, users and applications of interest. It can address the Complexity Problem by providing a description of the computer system at a level of abstraction appropriate to a digital investigation. By this is meant describing the computer system in terms useful for an investigator as opposed to in terms of systems architecture and the minutiae of various file formats. Marrington *et al* [1, 2] described a process to undertake such an automated examination, for which the term computer profiling was proposed. Computer profiling is not the only approach to addressing the Quantity and Complexity Problems. Alink *et al* describe the XIRAF framework, which separates data extraction from analysis, employing a common XML-based output format for evidence extraction tools to facilitate a human analysis of data from different extraction tools [5]. Beebe and Clark surveyed various data mining techniques to deal with very large forensic datasets [6]. We believe that, in practice, a combination of techniques will be required to address the Quantity and Complexity Problems. Additionally, it is important to note that the purpose of computer profiling is not to produce an analysis to be relied upon in a court or to make any other such final decision, but to produce an analysis which can guide a more detailed manual computer forensic investigation and indeed the decision to embark upon it in the first place.

In this work we briefly describe a formal representation of the computer profiling model described by Marrington *et al* [1, 2]. 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 objects contained in the set O may be related to each other, representing some relationship between the respective entities they represent. In addition to the discovery of the computer system's objects, the profiling process discovers relationships between those objects. We use logical predicates to express relationships. Let x and y be objects discovered on the computer system which share a relation R to one another according to some predicate "related(x,y)" which is true when x and y are related. The set R consists of all pairs of objects who are related according to the "related" predicate. This is the most generic type of relationship, which is convenient for automated reasoning but of little direct use to an investigator. Marrington *et al*'s model also describes relationships of specific types using more specific predicates than "related(x,y)", but the formal representation of these are not given here for the sake of brevity. All relationships have sets defined along similar lines to R:

$$R = \left\{ \left(x \in O, y \in O \right) \middle| \text{related}\left(x, y \right) \right\}.$$

All of the sets of different types of relationships, defined like R, together form the collection AR in a computer profile.

Finally, the computer profile 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 describe 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:

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

In our formal representation of 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.

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

The computer profile of a given computer system combines the set of all the discovered objects O, the collection of the different sets of relationships between objects AR, the set of all the times in the computer system's history T, and the set of all events, discovered and inferred, EVT. The sets R and $EVTI$, are created by automated reasoning based on the digital evidence, as opposed to identifying all the entities which can be found in the digital evidence, as is the case with O. The sets AR and $EVTI$ are 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 a relationship not being discovered or 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.

3 Detecting Inconsistency in Timelines

Marrington *et al* discussed a timestamp-based technique for building a timeline about a given object in the profile of the computer system [1]. 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 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 profiling 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 [7, 8]. However, this solution does not address the issue of clock skew varying over time prior to the examination of the computer system, and it is this variance which may lead to inaccuracies in timelines. Willassen proposes an algebra for the formal expression of falsifiable hypotheses about the discrepancy between a computer's clock and physical time [9]. 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. The techniques described in this paper are intended to detect internal inconsistency in timelines, which can assist an investigator to form clock hypotheses.

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 as the *happened-before* relation [10]. In [11], Gladyshev and Patel 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 profiling, the real time, if not the timestamp, of the execution event must be after the real time of the login event. Let $x \in P$, $y \in A$ and $t_n \in T \wedge t_m \in T$:

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

After the construction of a timeline (which is a sequence over the set *EVT*) in the computer profiling 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 [10, 11]:

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

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.

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

In [9], Willassen extends the use of the *happened-before* relation of Lamport [10], Fidge [12], and Gladyshev and Patel [11] to imply causality. Willassen's version of the *happened-before* relation is therefore equivalent to the "precondition" predicate. For the purposes of the research described in this chapter, it is preferable to maintain the *happened-before* relation as described by Lamport [10], Fidge [12], Gladyshev and Patel [11], 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:

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

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:

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

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. In computer profiling, events are inferred 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 computer profiling 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.
- 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

We developed prototype computer profiling software in order to conduct experiments relating to temporal inconsistency in computer profiles,. 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 prototype represents these as objects. It should be noted, however, that the prototype software does not discover all of the system objects on the target computer system. The set S consists of a single element, one System object to represent any logical entity on the target which should be represented by a system object. The software discovers relationships between the objects on the target 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, the set of objects O and relationships AR are examined and a set of events are inferred from the temporal data associated with each object and relationship. 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 querying the objects, relationships and events, and a specialised interface for the purpose of detecting temporal inconsistency in a given timeline, shown in Figure A.

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, and every event is implemented by a Java object. Rule objects have two event objects as fields, one called *evtA* and another called *evtB*. The objects *evtA* and *evtB* 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:

$$\text{precondition}\big((t_i \in T, u, s, \text{logon}, \text{success}), (t_k \in T, u, c \in C, \text{modified}, \text{success})\big).$$

In the object which represented this rule, *evtA* would represent the "logon" event, and *evtB* 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 *evtA* and *evtB* would be as follows:

$$evtA = (\text{null}, \text{null}, s, \text{logon}, \text{success})$$
$$evtB = (\text{null}, \text{null}, \text{null}, \text{modified}, \text{success}).$$

A limitation of the prototype's implementation of the detection techniques described in Section 3 is that it doesn't have a concept of a user session. A logon or logoff event is treated the same as any other event. Consequently, in the experiments described in Section 5, the timelines examined by the prototype software correspond to user sessions. 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.

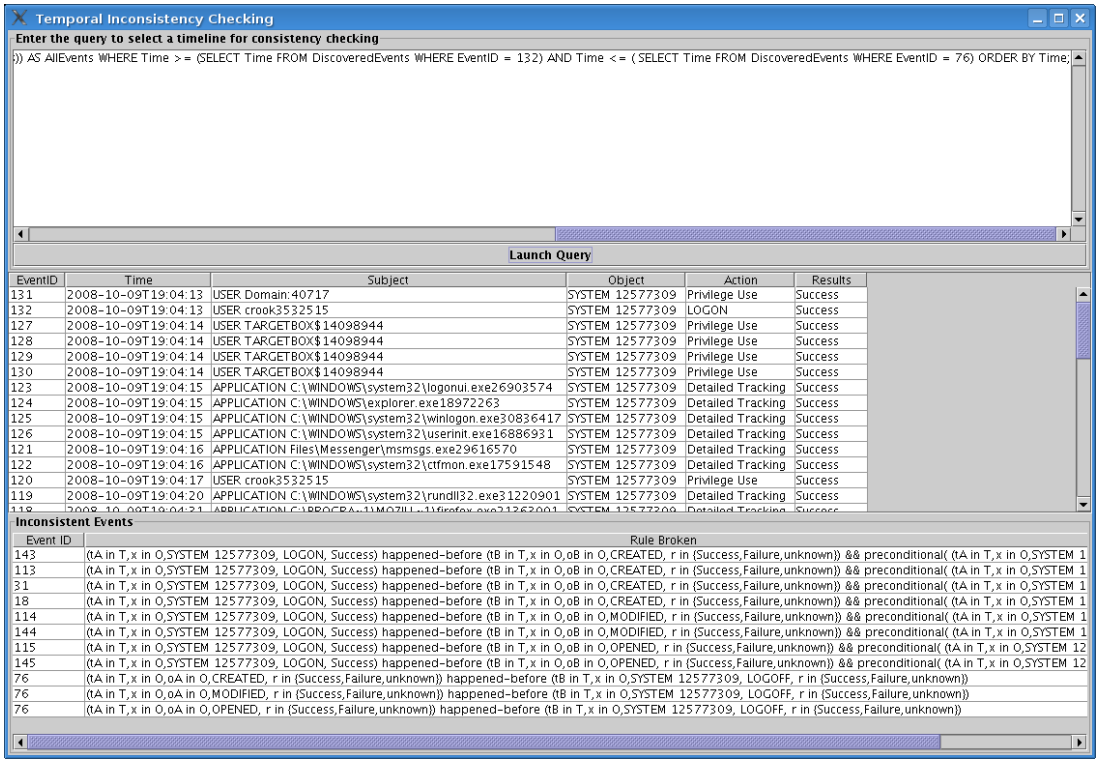

Figure A - Our prototype's inconsistency checking interface.

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:

$$\text{preconditional}(\text{userlogin}, \text{userlogout})$$
$$\text{preconditional}(\text{userlogin}, \text{filecreated})$$
$$\text{preconditional}(\text{userlogin}, \text{filemodified})$$
$$\text{preconditional}(\text{userlogin}, \text{fileaccessed})$$

$$\text{filecreated} \rightarrow \text{userlogout}$$
$$\text{filemodified} \rightarrow \text{userlogout}$$
$$\text{fileaccessed} \rightarrow \text{userlogout}$$

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.

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 computer profiling software, 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 where possible, when a non-canonical Principal object is the subject of an event, the prototype software attempts to find a canonical Principal object which may represent the same person. The prototype software posits an "isuser" relationship between non-canonical Principal objects and canonical Principal objects with the

same "name" field. In the case of the timelines presented in this section, the prototype software made the assumption that the non-canonical Principal object of type Individual with the name "baddie" represented the same person as the canonical Principal object of type User with the name "baddie". This assumption was made on the basis of the similarity of the names of the two objects.

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.

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). It is worth noting that for every inferred event describing an action by "baddie" on the object "WORDDOC invoice.doc19509473", there was a corresponding action on the object "WORDDOC Normal.dot3981922". This was the first time the user "baddie" had created a Word document, and consequently the normal template file was created for that user account.

An inconsistency check of timeline A against the rules provided in Section 4.2 demonstrated no inconsistencies. If the non-canonical Principal object "INDIVIDUAL baddie17975110" had not been linked to the canonical Principal object "USER baddie27660658" and the latter substituted for the former during the "infer events" stage of the prototype software's process, then the events around the authorship of "WORDDOC invoice.doc19509473" would have been highlighted as suspicious. As it was, 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".

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)

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

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.

Finally on the matter of timeline B, we note that it would have been possible to detect an inconsistency in the authorship attribution of the Word document "WORDDOC letter from baddie to nefarious.doc14850080" on the basis of the document's associated template. As noted in the discussion of timeline A, every user has their own instance of a Word template, which is created the first time they create a Word document according to that template. The relationship between the Word document "WORDDOC letter from baddie to nefarious.doc14850080" and its template "WORDDOC Normal.dot20348456" would suggest that the latter was the "baddie" user's normal template. That would be inconsistent with the relationship between "baddie" and the template created during timeline A, "WORDDOC Normal.dot3981922". The potential for using relationships between objects to detect inconsistency is discussed in Section 6.1.

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.

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)

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

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 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.

Time	Subject	Action	Rules
9/10/2008 18:51:23	USER baddie27660658	LOGOFF	filecreated → userlogout,filemodified → userlogout,fileaccessed → userlogout

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.

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 profiles constructed from realistic data is possible. These experiments applied a simple rules set to a real computer system's profile, 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 real computer system demonstrated that this approach to detecting temporal inconsistency is robust enough to be employed in real cases. The noise in realistic event data is a lesser problem to an automated process such as computer profiling than it is to a human investigator. By distilling event records down to the most important fields which are common to most events, computer profiling 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.

6 Future Work

There are two areas for future work in the handling of inconsistency in automated computer profiling which are immediately apparent. The first is another means of detecting inconsistency in computer profiles, based on inconsistency in relationships between objects rather than inconsistency in the chronology suggested by timestamps. In addition to detecting contradictory relationships, such an approach could supplement the techniques described in Section 3 in the detection of inconsistency in timelines. The second obvious avenue for future work is the development of a method for the construction of corrected timelines, incorporating missing events detected by the technique in Section 3.2, correctly chronologically ordered. We have begun work in each of these areas.

6.1 Relationships-based Approach

During the discussion of the experiment results for timeline B, an alternative approach to detecting the timeline's inconsistencies was noted. The section noted that the inconsistency between the WordDocument object's apparent authorship and the user session in progress at the time could have been detected on a relationship basis. The WordDocument object's relationship to the document's template object contradicted the WordDocument object's relationship to the document's apparent author "baddie", as a result of the template object's relationship with the user "crook". On the one hand, the template was created for the user "crook" the first time "crook" logged into the computer system as the user's personal document template. The document in question was created from that template. On the other hand, the document's authorship information indicates that the user "baddie" created the document. The composition of the relationship between the document and the template and the relationship between the template and "crook" could be regarded as being inconsistent with the relationship between the document and "baddie". If a rules base about the credibility of relationships was created, it might be used to give one relationship greater bearing than the other.

Reasoning based on relationships can supplement the techniques to dealing with inconsistencies we have described. Reasoning based on the relationships between objects could be used, in certain circumstances, to correct events which are detected as being inaccurate. An example of this sort of reasoning is shown below. An event evt_x is detected as inconsistent on the basis that its precondition (that the user who was the subject of evt_x is logged in) is not met. The relationship from which evt_x was inferred is thus discredited, and an alternative basis is used to propose a different subject for evt_x. After evt_x is revised, it would be reassessed against the rule, and the process completed if it was still found to be inconsistent.

The rule:

$$\text{precondition}(\text{userlogin}, \text{filecreated})$$

Is broken by the event:

$$evt_x = (t_x, john, document, \text{created}, \text{success})$$

Which was inferred from the relationship:

$$\text{author}(john, document).$$

These relationships were also discovered:

$$\text{template}(template, document)$$
$$\text{owner}(jane, template)$$
$$\therefore jane(\text{owner} \circ \text{template}) document$$

On this basis, evt_x could be corrected:

$$\neg \text{author}(john, document)$$
$$\wedge jane(\text{owner} \circ \text{template}) document$$
$$\Rightarrow evt_x = (t_x, jane, document, \text{created}, \text{success}).$$

Relationship-based reasoning as a basis for both detecting and handling inconsistency in computer profiles is a topic for further research. If it can provide a reliable process for automatically correcting inconsistent events, then the approach discussed in Section 6.2 below for creating consistent timelines becomes viable. A practical implementation of a combination of these approaches could prove very useful in providing investigators with thorough and accurate forensic reconstructions of computer systems.

6.2 Constructing Consistent Timelines

A consistent timeline in the context of computer profiling is defined as a sequence of events ordered by the time at which they occurred, with no significant missing events. The physical time at which an event occurred may or may not correspond to the computer-generated timestamp of an event, which may be missing or inaccurate. A consistent timeline must include events which are missing from the sets *EVTR* and *EVTI*, but which are detected using the techniques described in Section 3.2. *EVTM* is the set of all of the missing events detected on the basis of a precondition rule. This section discusses the basis of a technique for constructing such a timeline.

There are some events, especially members of *EVTM* (discovered by the approach in Section 3.2), for which there is no timestamp. There are other events for which there is a timestamp, but whose timestamp is provably incorrect (as determined by the process described in Section 3.1). Gladyshev and Patel describe the process of determining the time at which a given event takes place by bounding the event's time [11]. The upper and lower bounds for the time of an event can be determined if the event must have occurred between two other events. The range between these bounds (the time interval Δt) is the range of

possible times at which the event must have occurred. This range can be further narrowed if it is known that there is a minimal delay *d* which applies to a particular *happened-before* relation [11].

Given the problems associated with attempting to use time intervals, we propose then, instead of timestamps, to use Lamport logical clocks [10] to provide the basis of ordering the consistent timeline. The timestamp (or time interval in the case of events with indeterminate time) of an event will be used as a variable in the clock, but it will be the clock and not the timestamp which will be used as the basis for ordering events.

The clock C is defined to be a function which assigns a number to every event in the consistent timeline. The number produced by C has no bearing on physical time, but each event still has a timestamp t or a time interval Δt which can be used to determine the approximate physical time of the event. The number produced by C must be lower for events which occurred earlier in the history of the computer system than the number produced for events which occurred later. This will permit events to be sorted by the number produced by the clock C on an ascending order basis. Where the numbers produced by the clock C are the same for two or more events, these events will be ordered in the constructed consistent timeline on an arbitrary basis.

Given a complete set of rules to detect inconsistent and missing events, the number of unknown events in the computer system's history can be minimised. Each of the known events in the history of the computer system will have an associated timestamp, or, in the case of events with no timestamps or with provably incorrect timestamps, the narrowest possible time interval during which the event could have occurred. The clock function C will combine the timestamp or time interval for each known event with the rules relating that event to the other knowable events, and produce a number (a Lamport logical clock value) according to which the event may be sorted into the consistent timeline. Once completed, the consistent timeline will represent the best sequential ordering of the known events.

7 Conclusion

Inconsistencies in a computer profile can compromise the value of the computer profile as an investigative tool. If an automated forensics process accepts the original digital evidence from the target computer system uncritically, it may produce a history of the computer system which is unusable as a result of inaccuracy. Perhaps worse, it may itself 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 proposed techniques for detecting inconsistent 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 profile. The automatic detection of inconsistencies which might indicate deliberate tampering could assist a human investigator in a subsequent manual examination of the system.

Additionally, we have begun work on methods to create consistent timelines and to detect inconsistency in a computer profile's relationships. This work should provide automated computer profiling with a means to handle, as well as detect, chronological errors in timelines. Work into inconsistency in relationships should also provide a means for correcting the subject and target fields of events.

References

[1] A. Marrington, G. Mohay, A. Clark, and H. Morarji, "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, pp. 71-87.

[2] A. Marrington, G. Mohay, H. Morarji, and A. Clark, "Computer Profiling to Assist Computer Forensic Investigations," in *RNSA Security Technology Conference*, Canberra, 2006, pp. 287-301.

[3] B. Schatz, G. Mohay, and A. Clark, "A correlation method for establishing provenance of timestamps in digital evidence," *Digital Investigation - The Proceedings of the 6th Annual Digital Forensic Research Workshop (DFRWS '06),* vol. 3, pp. 98-107, 2006/9 2006.

[4] B. Carrier, "Defining Digital Forensic Examination and Analysis Tools Using Abstraction Layers," *International Journal of Digital Evidence,* vol. 1, 2003.

[5] W. Alink, R. A. F. Bhoedjang, P. A. Boncz, and A. P. de Vries, "XIRAF - XML-based indexing and querying for digital forensics," *Digital Investigation - The Proceedings of the 6th Annual Digital Forensic Research Workshop (DFRWS '06),* vol. 3, pp. 50-58, 2006/9 2006.

[6] N. L. Beebe and J. G. Clark, "Dealing with Terabyte Datasets in Digital Investigations," in *Advances in Digital Forensics*, M. Pollitt and S. Shenoi, Eds. Norwell: Springer, 2005, pp. 3-16.

[7] C. Boyd and P. Forster, "Time and date issues in forensic computing - a case study," *Digital Investigation,* pp. 18-23, 2004.

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

[9] S. Y. Willassen, "Hypothesis-Based Investigation of Digital Timestamps," in *Advances in Digital Forensics IV*. vol. 285 Boston: Springer, 2008, pp. 75-86.

[10] L. Lamport, "Time, clocks, and the ordering of events in a distributed system," *Commun. ACM,* vol. 21, pp. 558-565, 1978.

[11] P. Gladyshev and A. Patel, "Formalising Event Time Bounding in Digital Investigations," *International Journal of Digital Evidence,* vol. 4, 2005.

[12] C. Fidge, "Logical time in distributed computing systems," *Computer,* vol. 24, pp. 28-33, 1991.