IET Information Security

Research Article

IET Journals
The Institution of Engineering and Technology

Event reconstruction using temporal pattern of file system modification

ISSN 1751-8709
Received on 2nd December 2017
Revised 10th May 2018
Accepted on 17th December 2018
E-First on 29th January 2019
doi: 10.1049/iet-ifs.2018.5209
www.ietdl.org

Somayeh Soltani[1], Seyed Amin Hosseini Seno[1] ✉, Hadi Sadoghi Yazdi[1]

[1]*Department of Computer Engineering, Ferdowsi University of Mashhad, Mashhad, Iran*

✉ *E-mail: hosseini@um.ac.ir*

Abstract: Nowadays, several digital forensic tools extract a lot of low-level information from different parts of the system. Constructing high-level information from low-level ones is very challenging. This study reconstructs high-level events by using the traces of applications that are found in the file system metadata. In this regard, an event reconstruction framework is proposed that determines which applications have been run on a compromised system. The proposed framework works in two phases. In the training phase, the signatures of various applications are constructed. The signature of an application is the temporal pattern of file system modification of the application. In the detection phase, at first, the temporal pattern of file system modification of the hard disk (TPFSM-D) of the compromised system is constructed. Then in order to determine whether a particular application has been run on the compromised system, the distance between the signature of the application and the TPFSM-D of the hard disk is calculated by using a proposed distance measure. Finally, a decision engine decides whether the application has been run on the compromised system. The proposed event reconstruction framework has been tested on different scenarios. The empirical results suggest that the framework is effective in reconstructing events.

1 Introduction

Digital forensic is the science of detection, extraction, and analysing the pieces of evidence from the digital media. Its goal is to prepare accepted reports for the courts [1]. Digital forensic is a critical requirement in cyber space. Hard disk, memory, and network forensic are three important components of digital forensic since they record some tracks from behaviours of cyber criminals. Each of these components contains large volume of low-level data as evidence. Examining millions of pieces of raw data to extract high-level information is a time-consuming and exhausting work. Thus, some automatic methods are required to generate high-level information from raw and low-level data [2–4].

Various types of evidence can be extracted from different parts of the system which makes the automatic reconstruction of events a very challenging process [2]. For example, physical memory image, captured network traffic, log files, registry, file system metadata, and other parts of the system have different formats. Integrating these large-volume multi-format data and extracting meaningful information require many works to be done.

Running each application leaves some traces on different parts of the system, which can be used to reconstruct events. NTFS file system is a valuable source of forensically sound information. It stores various metadata for each file and folder in a specific file called master file table (MFT) [5]. For example, the MACB timestamps (the last Modification time, the last Accessed time, time of Changing MFT entry, and time of Born) of each file and folder are stored in MFT. By running an application, the MACB timestamps of many files and folders are updated. Besides the EXE file of an application, some other files and folders are accessed, modified, or even created during running the application. Common libraries, configuration files, and log files are some files that their MACB timestamps might be changed [6].

It is possible to find files and folders modified or accessed by an application by comparing copies of the hard disk before and after running the application. However, updates on the metadata timestamps are not necessarily caused by a specific application. For example, Windows is a multi-process operating system and many tasks and processes would run concurrently. Therefore, timestamps might be updated by several applications and not by a particular application.

In this paper, the events are reconstructed by using the traces of applications stored in MACB timestamps of NTFS in a Windows operating system. NTFS is the default file system for Windows NT family, and it is the most widely used file system on private computers nowadays [2, 6]. Therefore, the NTFS file system is the focus of this work. In this paper, an event reconstruction framework is presented that determines which applications have been run on a compromised system. The proposed framework consists of a training phase and a detection phase. In the training phase, the signatures of various applications are constructed. The signature of an application is called the temporal pattern of file system modification of the application (TPFSM-A). TPFSM-A is a time series of file system metadata modifications which results from the run of the application. The detection phase of the proposed event reconstruction framework compares the hard disk of the compromised system against different signatures for deciding which applications have been executed on it. This phase consists of three components: (i) TPFSM-D constructor, (ii) distance calculator, and (iii) decision engine. The first component constructs the TPFSM-D of the hard disk of a system. TPFSM-D is the temporal pattern of file system modification of the hard disk. It is a time series of file system metadata modifications that contains traces of various applications and operating system. The second component calculates the distance between the signature of an application and the TPFSM-D of the hard disk of the system using a proposed distance metric. The third component uses the calculated distance to decide whether the particular application has been run on the system. When dealing with a compromised system, it is useful to know which applications have been run on it, and the proposed event reconstruction framework helps to understand it.

The contributions of this study are as follows:

i. The signature or the temporal pattern of file system modification of an application (TPFSM-A) is defined.
ii. The temporal pattern of file system modification of a hard disk (TPFSM-D) is defined.
iii. A method is presented for calculating the distance between TPFSM-A and TPFSM-D.
iv. A method is presented to determine whether a particular application has been run on a system.

The rest of the paper is organised as follows. Section 2 presents a literature review on the subject. Section 3 elaborates the proposed event reconstruction framework. The performed experiments, results, and evaluation are described in Section 4. Finally, Section 5 concludes the paper and presents some future works.

2 Literature review

Several researches in the area of digital forensic have proposed useful tools to gather raw data from the system [5, 7–11]. However, analysing the collected low-level data to understand the occurred events is less paid attention. Nowadays, there exist millions of low-level records of information obtained by various digital forensic tools. Extracting high-level events from these raw data accelerates the forensic process.

Several researchers have addressed the problem of event reconstruction using different extracted data. Many of event reconstruction methods in the literature use extracted data from hard disk [2–4, 6, 9, 10, 12–25]. However, there are event reconstruction techniques which are based on the extracted pieces of evidence from memory image [26–29] or network traffic [30–32]. It should be noted that, because of the permanent nature of hard disk data, the pieces of evidence which are extracted from hard disk are more reliable.

Event reconstruction methods which are based on hard disk are categorised into two groups: (i) signature-based event reconstruction [2–4, 12–14] and (ii) correlation-based event reconstruction [15–19, 23, 24]. Signature-based event reconstruction methods try to find some fingerprints or signatures of applications using extracted pieces of evidence. Correlation-based event reconstruction methods try to find some correlations among low-level pieces of evidence. Moreover, there are some miscellaneous event reconstruction methods in the literature [6, 9, 10, 20–22].

2.1 Signature-based event reconstruction methods

Some methods reconstruct events using the signature of applications or events. These methods first try to find the signatures of various applications. Then, by using these signatures, they can detect which events have been occurred or which applications have been run on the compromised system.

In order to create the fingerprint of an application, James *et al.* [12] extract file and registry timestamps of the application using Microsoft Process Monitor [33]. Then they define four categories of timestamp updates: timestamps updated in each run, timestamps updated only on the first run, timestamps updated irregularly, and timestamps updated based on the usage. The timestamps of the first category are the most important part of the fingerprint. Other timestamps are considered as supportive evidence.

Similarly, the technique proposed by Kalber *et al.* [2] finds fingerprints of different applications using timestamp metadata related to files and folders of the application. In order to obtain the fingerprint of an application, this technique compares the file system metadata just before and after running the application and then considers their difference as the fingerprint of the application. For this comparison, the Fiwalk digital forensic tool is used [14]. The technique of Kalber *et al.* has been applied to Windows operating system in which several programs can be run simultaneously. Thus, in order to remove the impacts of other programs on the generated fingerprint, they have repeated their experiment 100 times.

Hargreaves and Patterson [3] have presented a python digital forensic timeline tool named PyDFT. PyDFT is capable of extracting timing information from various types of files and reconstructing high-level events from low-level ones. In addition, it preserves the provenance of the high-level events and shows the low-level building block events. PyDFT works in two steps: low-level event extraction and high-level event reconstruction. In the low-level event extraction, for each type of file, its concerned extractor is invoked to add the timing information within the file to the low-level event timeline. In the high-level event reconstruction phase, for each high-level event, some rules are defined. Then an analyser script searches for these rules in the low-level event timeline.

Khan *et al.* [4] have introduced a neural network-based method for event reconstruction. In order to generate train and test data, different applications are run separately on a virtual machine. The patterns of the file system activity of various applications are captured using FileSystemWatcher library of.NET technology. Then two feed forward and recurrent neural networks are trained using nine inputs from the file system and registry values. In another paper, to reconstruct events, Khan [13] has presented two neural network-based and Bayesian network-based methods and then has compared the performance of these two ones.

At the end of this section, it should be noted that signature-based event reconstruction methods in [2, 12] are time-consuming. In addition, they can be bypassed by attackers. In these methods, only a small number of updated timestamps are considered as the signature of application. As a result, the attacker can hide his activities by changing timestamps of files and folders using some tools such as BulkFileChanger [34]. Furthermore, the event reconstruction method in [3] defines some rules for different events which is an exhausting work. The methods in [4, 13] define different neural networks for different applications. However, as Khan *et al.* have stated, the constraint of these methods is that separate neural networks are required to be trained for different applications which is a time-consuming task.

2.2 Correlation-based event reconstruction methods

FACE [16] is a framework which provides automated digital evidence discovery and correlation. FACE provides some parsers for extracting data from various resources such as memory dumps, network traces, hard disk images, log files, and configuration files. It also has one correlation engine which uses an internal database to store and correlate various conceptual units (such as user identities, filenames etc.).

ECF [15] is another event correlation method which extracts pieces of evidence from varied resources. Although these pieces of evidence have different formats, some common attributes of them such as Time, Action, Subject, and Object are stored in a canonical table. ECF implements some parsers for several types of files such as Apache Server Log, Browser Cache Logs, and *NIX SYSLOG which insert the common attribute values into the canonical table.

Raghavan *et al.* [24] have presented a forensic integration architecture which provides a framework for abstracting evidence sources and integrating evidence information from multiple sources. In another research, Raghavan and Raghavan [17] have proposed AssocGEN which is an engine for analysing associations in pieces of evidence. AssocGEN uses metadata of extracted digital artefacts to determine the correlations between them. It seeks metadata matches between digital artefacts and groups the related artefacts.

FORE [18] is a forensics ontology which is based on the Web Ontology Language (OWL). The ontology can provide strong representational model for forensics data. It also may provide automated methods of correlating these data. FORE ontology has two base classes, an entity class which represents tangible ‘objects’ in the world and an event class which represents changes in state over time. FORE constructs rules to identify the causal relationships between events. If event A happens before event B, A causes B.

Chabot *et al.* [23] have introduced a formalised knowledge representation model for digital forensics timeline analysis. Their approach reconstructs scenarios from suspect data and analyses them using experts’ knowledge and semantic tools. Then a formal incident modelling and timeline reconstruction is presented to guarantee the correctness of the whole investigation process. In another paper, Chabot *et al.* [19] have introduced an ontology-based approach to reconstruct and analyse the digital forensics timelines. In this regard, they have proposed the ORD2I ontology (Ontology for the Reconstruction of Digital Incidents and Investigation) which is based on OWL 2 (http://www.w3.org/TR/owl2-overview/) that is a description logic language. The ORD2I ontology has three layers: (i) the common knowledge layer, (ii) the

specialised knowledge layer, and (iii) the traceability knowledge layer. The task of the common knowledge layer is to store common knowledge about events. Although events have different types, they have a number of common characteristics such as timing information, resources, and involved subjects. The specialised knowledge layer stores technical information about different objects that may be found in a cyber environment such as files, user accounts, web-related objects, communication-related objects, and registry keys. Finally, the aim of the traceability knowledge layer is to ensure the reproducibility of the results. This layer stores information about investigative activities, data used during investigation, and the involved users.

At the end of this section, it should be noted that event reconstruction methods such as ECF [15] and FACE [16] which store data in databases lack the semantic of data. Moreover, the correlations of events in these methods are not automatic which pose the burden of finding the correlations on the investigator. FORE [18], another correlation-based event reconstruction method, uses ontology to represent forensics data. However, constructing the rule set is a time-consuming and hard task.

2.3 Miscellaneous event reconstruction methods

The event reconstruction method proposed by Kalber *et al.* [6] first extracts file system timestamps and then sorts them in an ascending order. Then if the gap between two consecutive timestamps is >20 s, it assumes that a new event has taken place. Finally, the timestamps within each event are clustered according to their filenames and paths using DBSCAN algorithm [17]. It should be mentioned that this method does not properly detect the occurrence of the applications since it just considers 20 s gaps between timestamps as the occurrence of a new event.

CFTL [10] is a computer forensic timeline visualisation tool which extracts FAT and NTFS timestamps. It also extracts timing information stored within various types of files including EXIF files, Link files, MBOX archives, and registry files. The CFTL uses specific extractor for each type of file. log2timeline [9] is another efficient digital forensic timeline tool which extracts timing information from 26 different files such as Chrome browser history, Event logs, Firefox bookmark files, Firefox browser history, Internet Explorer browser history, and McAfee antivirus log files. While methods in [9, 10] extract various low-level timing information from different parts of the hard disk, they do not usually construct high-level events. It should be mentioned that Plaso, which is a backend engine for log2timeline, provides tagging rules to support event reconstruction partially.

The event reconstruction method presented by Zhu *et al.* [22] extracts ShellBag information from some registry keys and then reconstructs the user activities using this information. Some information about Windows default shell, i.e. Windows Explorer, is stored in different locations in the Windows registry. In this regard, some of users' window viewing preferences such as the window size, the position of the window on desktop, view mode, and sort method for items within the window are used. In this method, in order to reconstruct user activities, the registry snapshots are compared and the presence or absence of user actions between two snapshots are detected using nine different detection rules.

The event reconstruction method in [20] describes the compromised system as a finite state machine (FSM). Then, in order to determine all possible scenarios of the incident, it back-traces transitions from the state in which the system was discovered. Finally, it discards the scenarios which disagree with the available evidence. Unfortunately, this method cannot be used in real-world systems. The reason is that a real-world system has thousands of states, and thus, the state space grows exponentially.

Similarly, the event reconstruction method in [21] describes the system as a deterministic finite automaton (DFA) which encodes the set of system computations as a set of strings. Then it defines witness statements as restrictions on strings which are accepted by the DFA. Although the witness statements restrict the state space, in a real-world system, the state space still grows exponentially.

Table 1 summarises the related work on event reconstruction. Several issues including the type of event reconstruction, timing and non-timing information which are used as pieces of evidence, reproducibility, and being resistant against anti-forensic attempts are stated in this table. For example, record 1 in Table 1 states that the event reconstruction method by James *et al.* [12] is a signature-based method which gathers and analyses timing information of file system metadata and registry. Moreover, this method is reproducible, i.e. it is able to explain how the results are obtained. However, this method is not resistant against anti-forensic attempts. The signature of application that is produced by this method contains only a few number of items which can be easily bypassed by an attacker. For another example, consider the method proposed by Hargreaves and Patterson [3] in record 3 of Table 1. This method manipulates various timing information within file system, registry, various log files, and some other files. This method also preserves the provenance of produced high-level events. Therefore, its results are clearly reproducible. Here, we consider that this method is resistant against anti-forensic attacks. The argument behind this is that the obtained signatures are based on low-level data from different resources, and it is harder for an attacker to clear all of his traces on these resources. It is important to note that there are various anti-forensic attempts. However, here we just consider the anti-forensic attempts which try to bypass the signature matching techniques.

Furthermore, as record 4 of Table 1 shows, the neural network-based method presented by Khan *et al.* [4] lacks the reproducibility. The reason is that some parameters used during the learning phase of neural networks remain unknown, and thus, the results of neural networks are not easily interpretable, a point originally raised by Chabot *et al.* [19]. With regard to anti-forensic attacks, as mentioned above, we just consider those that are related to signature matching schemes. Therefore, the anti-forensic resistant fields in records 6–16 are denoted by the symbol •. Moreover, as mentioned earlier, methods in [9, 10] do not usually construct high-level events. Therefore, the reproducibility fields in records 12 and 13 are denoted by •. Furthermore, the event reconstruction methods in [20, 21] differently model the system as an FSM. Since the types of information are not important in these methods, the related fields in records 15 and 16 are denoted by •.

The event reconstruction method presented in this paper is similar to signature-based methods, but with the difference that it does not rely on a fix and small number of updated timestamps. Therefore, it is more resistant against anti-forensic efforts. It is harder for attackers to change the timestamps of various files and folders than changing a small number of files and folders. Moreover, the proposed method does not simply match the signature of an application against a compromised system. Instead, it introduces a distance measure and calculates the distance between the signature of the application and the compromised system. If the computed distance is lower than a predefined threshold, it is concluded that the application has been run on the compromised system.

3 Proposed event reconstruction framework

In this section, a framework is proposed that determines which applications have been run on a compromised system. The proposed event reconstruction framework has two phases: a training phase and a detection phase. In the training phase, the signatures or TPFSM-As of applications are constructed. The detection phase decides whether a particular application has been run on a system. While the training phase of the proposed framework has one component called TPFSM-A constructor, the detection phase consists of three components: (i) TPFSM-D constructor, (ii) distance calculator, and (iii) decision engine. TPFSM-A constructor in the training phase constructs the TPFSM-A of the application. In the detection phase, TPFSM-D constructor constructs the TPFSM-D of the hard disk of the compromised system, then distance calculator calculates the distance between TPFSM-A of the application and TPFSM-D of the hard disk, and finally, decision engine decides whether the application has been run on the compromised system. Fig. 1 shows the phases and the components of the proposed event reconstruction framework.

Table 1 Summary of event reconstruction methods

	Event reconstruction method	Signature-based	Correlation-based	Timing information				Non-timing information	Reproducibility	Anti-forensic resistant
				File system	Registry	Log files	Other files			
1	James *et al.* [12]	✓	×	✓	✓	×	×	×	✓	×
2	Kalber *et al.* [2]	✓	×	✓	×	×	×	×	✓	×
3	Hargreaves and Patterson [3]	✓	×	✓	✓	✓	✓	×	✓	✓
4	Khan *et al.* [4]	✓	×	✓	✓	✓	×	✓	×	✓
5	Khan [13]	✓	×	✓	✓	✓	×	✓	×	✓
6	Case *et al.* [16]	×	✓	×	×	✓	✓	✓	✓	•
7	Chen *et al.* [15]	×	✓	×	×	✓	×	✓	✓	•
8	Raghavan and Raghavan [17]	×	✓	✓	×	✓	✓	✓	✓	•
9	Schatz *et al.* [18]	×	✓	×	×	✓	×	×	✓	•
10	Chabot *et al.* [19]	×	✓	✓	✓	✓	✓	✓	✓	•
11	Kalber *et al.* [6]	×	×	✓	×	×	×	×	×	•
12	Olsson *et al.* [10]	×	×	✓	✓	✓	✓	×	•	•
13	Guðjónsson [9]	×	×	✓	✓	✓	✓	×	•	•
14	Zhu *et al.* [22]	×	×	×	✓	×	×	×	×	•
15	Pavel Gladyshev and Patel [20]	×	×	•	•	•	•	•	✓	•
16	James *et al.* [21]	×	×	•	•	•	•	•	✓	•

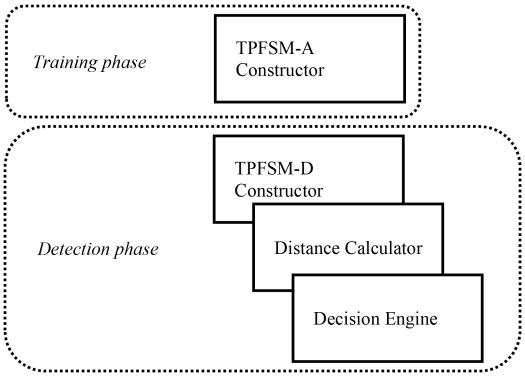

Fig. 1 *Proposed event reconstruction framework*

In the training phase, each application is run in different scenarios, and for each run, the TPFSM-A of the application is constructed. Then one of these TPFSM-As should be considered as the signature of the application. Moreover, some other parameters should be configured in this phase. The reason is that our event reconstruction method, unlike other signature-based methods, is not based on exact signature matching; rather, it is based on a distance measure. In the detection phase, if the distance between the signature (TPFSM-A) of the application and the hard disk of the compromised system is near to a specific value, it is said that the application has been run on the system. This specific value is an indicator of existence of the application in the copy of hard disk which is defined in the training phase. Moreover, in order to exactly define the concept of 'near', a threshold is considered for the distance. This threshold is also defined in the training phase. The following subsections describe some preliminaries, details of the training phase, and details of the detection phase, respectively.

3.1 Preliminaries

The proposed event reconstruction framework needs to calculate the distance between TPFSM-A of an application and TPFSM-D of a hard disk. Since TPFSM-A and TPFSM-D are both time series, it is needed to measure the distance between time series. In this regard, the dynamic time warping (DTW) distance measure is used.

DTW is a technique for computing the minimum distance between two time series of different lengths. Let $T_1 = \{a_1, a_2, \ldots, a_N\}$ and $T_2 = \{b_1, b_2, \ldots, b_M\}$ be two time series. In order to compute the distance between these two time series, using DTW technique, first, it is required to compute the distance between elements of two series, one by one. The Euclidean distance method or other methods can be used to calculate this distance. For example, the Euclidean distance between the ith element of T_1, i.e. a_i, and the jth element of T_2, i.e. b_j, is computed as

$$d_{i,j} = |a_i - b_j| \tag{1}$$

Then dynamic programming can be used to compute the DTW distance. For this purpose, a cumulative distance matrix $\boldsymbol{D}_{N\times M}$ is calculated. In this matrix, $D_{i,j}$ is the cumulative distance between ith element of T_1 and jth element of T_2. Thus, $D_{i,j}$ is calculated using the following equation

$$D_{i,j} = d_{i,j} + \min\left(D_{i,j-1},\ D_{i-1,j},\ D_{i-1,j-1}\right) \tag{2}$$

The distance between the two time series is equal to the last element of matrix $\boldsymbol{D}$, i.e. $\boldsymbol{D}_{N,M}$. The final step is the calculation of the optimal warped path. The optimal warped path $p^* = (p_1, p_2, \ldots, p_L)$ is calculated in reverse order of indices of the matrix. The last element of the path is equal to the last index of matrix $\boldsymbol{D}$, i.e. $p_L = (N, M)$. Thus, assuming $p_l = (n, m)$ is obtained, p_{l-1} is calculated using the following equation

$$p_{l-1} = \begin{cases} D(1,\ m-1) & \text{if } n = 1 \\ D(n-1,\ 1) & \text{if } m = 1 \\ \arg\min\{D(n-1,m-1), D(n-1,m), D(n,m-1)\} & \text{otherwise} \end{cases} \tag{3}$$

3.2 Training phase

In the training phase, each application is run several times in different scenarios, and the TPFSM-As of the application are constructed for each scenario. Moreover, for one of these scenarios, the TPFSM-D of the hard disk is constructed. Next, one of these TPFSM-As is considered as the signature of the corresponding application. Determining the ideal signature for each application is beyond the scope of this paper. As explained earlier, detection of a running application on a compromised system depends on the

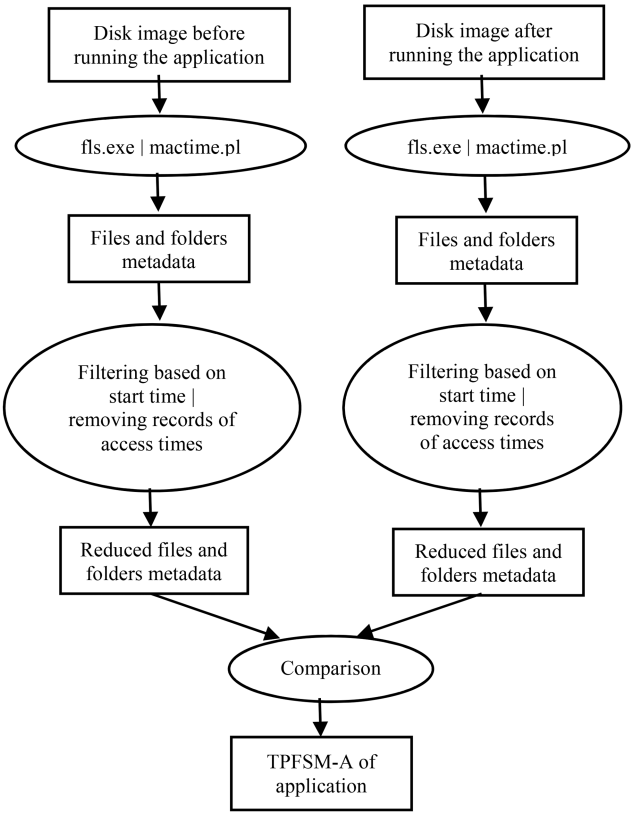

Fig. 2 *Constructing the TPFSM-A of an application*

distance between the signature of the application and the TPFSM-D of the hard disk. Moreover, as said before, this distance is compared to the indicator of the existence of the application. For each application, this indicator is defined in the training phase as the mean distance between the TPFSM-As of the application in different scenarios and the aforementioned TPFSM-D of the hard disk. Note that the constructed TPFSM-D in this phase is for the purpose of training and is related to the hard disk of the training system. It should not be confused with the TPFSM-Ds of the compromised systems which are dealt with in the detection phase. The real experiments in Section 4 help to clarify these concepts. The next subsection describes how a TPFSM-A of an application is constructed.

3.2.1 TPFSM-A construction: As mentioned before, running an application leaves some traces on file system metadata and modifies MACB timestamps of several files and folders. These file system metadata modifications can be sorted by time to map a time series to the running application. Thus, the run of an application can be shown by a time series which in this paper is referred to as TPFSM-A. This section describes the way of constructing the TPFSM-A of an application which plays the role of the signature of the application.

In the training phase, in order to construct the TPFSM-A of an application, it is needed to take a copy of the hard disk just before and after running the application, using a disk imaging tool such as the FTK Imager [8]. After taking copy of the hard disk, a list of files and folders within the copy is extracted, using the fls.exe tool from the Sleuthkit forensic toolset [7]. The output of this tool is a list which contains some information such as name, size, MACB timestamps, and metadata address of each file and folder within the copy of the hard disk. Since this output is not human readable, another tool from the Sleuthkit toolset, named mactime.pl, is used to convert the output of the fls.exe to a human-readable one. Mactime.pl sorts the information of files and folders according to the time field. Therefore, it is possible to see which file or folder has been modified, accessed, or created at each time point.

The output list contains a large number of files and folders. For example, consider an experiment in which the Microsoft Word application is run after the system boot up and then a copy of hard disk is taken. In this experiment, there exist almost 250,000 records in the output list of the copy of hard disk. In order to reduce the volume of the output list and concentrate on the time of event, a start time or a time range can be defined. There are some ways to guess approximately the start of application execution. For example, for each application, the related.pf file in the Prefetch directory contains a timestamp indicating the last time the application was run. Thus, in the previous example, by confining the output list to the time of running Microsoft Word, the number of records is reduced to 1500. The reason is that in the preliminary set of records, just a small portion of records is related to the Microsoft Word application and the rest one is related to the boot process and the background running applications.

The number of output records can be further reduced. Many of these records are related to the files and folders which are accessed. For example, clicking on the Start menu or opening a window by Windows Explorer updates the access timestamps of all files and folders within the window. Thus, the length of the output list can be reduced by removing the records related to access updates of files and folders. In the previous experiment, the number of records reduces to ∼150 by removing records related to access times.

Now there exist two output lists, one contains the timestamps of files and folders before running the application and another one contains the timestamps of files and folders after running the application. Then these two output lists can be compared with some tools such as *Compare It* [35]. This comparison is equivalent to comparing two copies of the hard disk. By comparing these two output list, the set of files and folders related to our application can be found. In this paper, this obtained set is referred to as TPFSM-A of the application. The TPFSM-A of the aforementioned experiment has ∼60 records. However, as mentioned before, because of the multi-process nature of Windows operating system, these files and folders are not necessarily related to our application. Fig. 2 shows the process of constructing the TPFSM-A of an application.

Some parts of TPFSM-A of running the Word application are shown in Fig. 3. This run of Word application is named *Word1* (see Table 2 for some description about this experiment). The *Date* field of TPFSM-A indicates the time of modification of a file or folder. The *Type* field stands for the type of timestamp (*m* for

modification, *a* for access, *c* for change of the related MFT entry, and *b* for born). For example, in the first record of TPFSM-A of *Word1*, *mac.* indicates that '*/WINDOWS/SoftwareDistribution/DataStore/Logs/edb.log'* is modified and accessed and the related MFT entry is changed at 2017-04-21T04:33:30Z.

3.3 Detection phase

When the investigator deals with a compromised system, she may want to know which applications have been run on it. Having a database of applications' signatures achieved from the training phase, she is able to check this issue. To this end, she can take a copy of the hard disk of the compromised system. Then she can use the TPFSM-D constructor to construct the TPFSM-D of the hard disk. Next, she can use the distance calculator to calculate the distance between the desired application and the copy of the hard disk. Finally, she can employ the decision engine of the proposed event reconstruction framework to determine whether the desired application has been run on the compromised system. The following subsections describe each of the components of the detection phase.

3.3.1 TPFSM-D construction: The first component of the detection phase of our framework is TPFSM-D constructor. The construction of TPFSM-D is similar to the construction of TPFSM-A, except that just one copy of hard disk is taken. In order to construct the TPFSM-D of the hard disk of the system, it is sufficient to run the FTK Imager to take a copy of the hard disk. Next, a list of files and folders within the copy is extracted, using

Number	Date	Size	Type	Meta	File_Path
1	2017-04-21T04:33:30Z	131072	mac.	14769-128-3	/WINDOWS/SoftwareDistribution/DataStore/Logs/edb.log
2	2017-04-21T04:33:30Z	56	mac.	29504-144-5	/WINDOWS/SoftwareDistribution/DataStore/Logs
3	2017-04-21T04:33:30Z	8192	mac.	29510-128-3	/WINDOWS/SoftwareDistribution/DataStore/Logs/edb.chk
4	2017-04-21T04:33:30Z	24125440	mac.	3328-128-3	/WINDOWS/SoftwareDistribution/DataStore/DataStore.edb
5	2017-04-21T04:33:31Z	1646041	mac.	24365-128-4	/WINDOWS/WindowsUpdate.log
6	2017-04-21T04:33:35Z	11780	mac.	29509-128-4	/WINDOWS/Prefetch/JAVA.EXE-1E21D4DA.pf
7	2017-04-21T04:33:35Z	56	..c.	29539-144-6	/Documents and Settings/Administrator/Local Settings/Temporary Internet Files
8	2017-04-21T04:33:35Z	56	..c.	29540-144-5	/Documents and Settings/Administrator/Local Settings/Temporary Internet Files/Content.IE5
9	2017-04-21T04:33:35Z	256	..c.	29546-144-1	/Documents and Settings/Administrator/Local Settings/History
10	2017-04-21T04:33:35Z	56	..c.	29547-144-5	/Documents and Settings/Administrator/Local Settings/History/History.IE5
11	2017-04-21T04:33:35Z	56	..c.	29567-144-5	/Documents and Settings/Administrator/Cookies
12	2017-04-21T04:33:35Z	212992	..c.	29625-128-4	/Documents and Settings/Administrator/Local Settings/Temporary Internet Files/Content.IE5/index.dat
13	2017-04-21T04:33:35Z	32768	..c.	29628-128-4	/Documents and Settings/Administrator/Local Settings/History/History.IE5/index.dat
14	2017-04-21T04:33:35Z	32768	..c.	29659-128-4	/Documents and Settings/Administrator/Cookies/index.dat
15	2017-04-21T04:33:37Z	118746	mac.	29756-128-3	/Documents and Settings/Administrator/Local Settings/Temp/jusched.log
16	2017-04-21T04:38:33Z	1024	mac.	3743-128-3	/WINDOWS/system32/config/SECURITY.LOG
17	2017-04-21T04:43:05Z	347432	..c.	13840-128-3	/Program Files/Microsoft Office/Office12/WINWORD.EXE
18	2017-04-21T04:43:05Z	2515	m.c.	14736-128-4	/Documents and Settings/Administrator/Desktop/Microsoft Office Word 2007.lnk
19	2017-04-21T04:43:05Z	90	macb	5262-48-2	/System Volume Information/_restore{76E4AA8D-B4CB-4EB7-9870-63A396AB60C6}/RP114/A0139562.lnk ($FILE_NAME
20	2017-04-21T04:43:08Z	8206	.ac.	14356-128-3	/Documents and Settings/All Users/Application Data/Microsoft/OFFICE/DATA/opa12.dat
21	2017-04-21T04:43:09Z	1024	...b	14694-128-4	/Documents and Settings/Administrator/Local Settings/Temporary Internet Files/Content.Word/~WRS{1A14CDE2-EE6
22	2017-04-21T04:43:09Z	158	macb	14694-48-2	/Documents and Settings/Administrator/Local Settings/Temporary Internet Files/Content.Word/~WRS{1A14CDE2-EE6

Fig. 3 *Parts of TPFSM-A of Word1*

Table 2 Experiments of phase 1

Experiment name	Experiment description
Word1	clicking on the Word shortcut on the desktop, creating a new document named *doc1*, and then closing the application
Word2	clicking on the Word shortcut on the Start menu, creating a new document named *doc1*, and then closing the application
Word3	clicking on the Word shortcut on the desktop, opening *doc1* document, but do not closing the application
Word4	opening *doc1* by clicking on it and then closing it
Adobe Reader1 (Adobe1)	clicking on the Adobe Reader shortcut on the desktop, opening *file1.pdf*, and then closing the application
Adobe Reader2 (Adobe2)	clicking on the Adobe Reader shortcut on the Start menu, opening *file1.pdf*, and then closing the application
Adobe Reader3 (Adobe3)	clicking on the Adobe Reader shortcut on the desktop, opening *file1.pdf*, saving the file as another file in drive C, and then closing the application
Adobe Reader4 (Adobe4)	opening *file2.pdf* by clicking on it and then closing the application
Firefox1	clicking on the Firefox shortcut on the Start menu, visiting *yahoo.com*, and then closing the application
Firefox2	clicking on the Firefox shortcut on the desktop, trying to visit google.com, an error occurred because the connection is not set up yet. Then visiting *google.com*, searching 'Elsevier journals', opening www.elsevier.com/journals in a new tab, and finally closing the application
Firefox3	clicking on the Firefox shortcut on the Start menu, visiting *mail.yahoo.com*, logging in, opening an email, downloading its attachment which is an audio file, and finally closing the application
Firefox4	clicking on the Firefox shortcut on the desktop, opening a bookmarked site (ieeexplore.com) from Bookmarks→Recently Bookmarked, clicking a link on this website and the following five consecutive links, and finally closing the application
Windows Media Player1 (Player1)	clicking on the Windows Media Player shortcut on the Start menu, opening and playing the *audio1.mp3*, and finally closing the application
Windows Media Player2 (Player2)	opening *video1.mp4* by clicking on it, playing of the video, and then closing the application
Windows Media Player3 (Player3)	clicking on the Windows Media Player shortcut on the desktop, creating a playlist named *favorite* including *audio1.mp3* and *video1.mp4*, running the playlist, and finally closing the application
Windows Media Player4 (Player4)	clicking on Start Menu→My Music→My Playlists→favorite, running the *favorite* playlist, and then closing the application

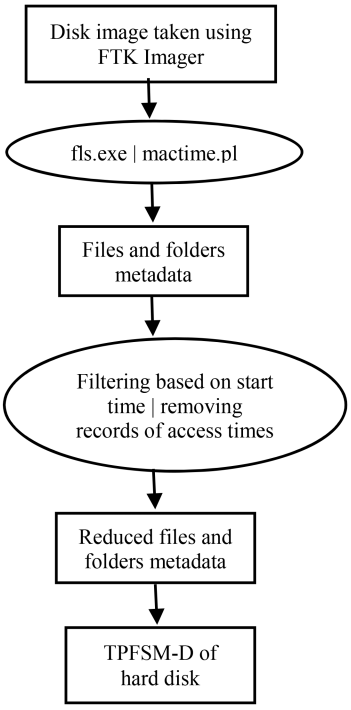

Fig. 4 *Constructing the TPFSM-D of a hard disk*

Number	Date	Size	Type	Meta	File Name
1	2017-05-01T04:10:57Z	4456448	mac.	1775-128-3	/WINDOWS/system32/config/system
2	2017-05-01T04:10:59Z	2048	mac.	1631-128-1	/WINDOWS/bootstat.dat
3	2017-05-01T04:10:59Z	2048	mac.	1631-128-1	/WINDOWS/system32/migisol.exe (deleted-realloc)
4	2017-05-01T04:10:59Z	2206	mac.	27-128-3	/Documents and Settings/All Users/Application Data/VMware/Compatibility/virtual/wpa.dbl
5	2017-05-01T04:10:59Z	80	mac.	27-48-2	/Documents and Settings/All Users/Application Data/VMware/Compatibility/virtual/wpa.dbl ($FILE_NAME)
6	2017-05-01T04:10:59Z	24	mac.	29523-128-1	/System Volume Information/_restore{76E4AA8D-B4CB-4EB7-9870-63A396AB60C6}/_driver.cfg
7	2017-05-01T04:10:59Z	352	mac.	36829-144-1	/Documents and Settings/All Users/Application Data/VMware/Compatibility/virtual
8	2017-05-01T04:10:59Z	0	mac.	3750-128-173	/WINDOWS/Debug/PASSWD.LOG
9	2017-05-01T04:10:59Z	524288	..c.	3754-128-1	/WINDOWS/system32/config/SysEvent.Evt
10	2017-05-01T04:10:59Z	8200	mac.	3761-128-3	/Documents and Settings/All Users/Application Data/VMware/Compatibility/virtual/OPA12.BAK
11	2017-05-01T04:10:59Z	8200	mac.	3761-128-3	/WINDOWS/pchealth/helpctr/DataColl/CollectedData_1254.xml (deleted-realloc)
12	2017-05-01T04:10:59Z	8200	mac.	3761-128-3	/WINDOWS/pchealth/helpctr/DataColl/CollectedData_1414.xml (deleted-realloc)
13	2017-05-01T04:10:59Z	84	mac.	3761-48-2	/Documents and Settings/All Users/Application Data/VMware/Compatibility/virtual/OPA12.BAK ($FILE_NAME)
14	2017-05-01T04:10:59Z	84	mac.	3761-48-2	/WINDOWS/pchealth/helpctr/DataColl/CollectedData_1254.xml ($FILE_NAME) (deleted-realloc)
15	2017-05-01T04:10:59Z	84	mac.	3761-48-2	/WINDOWS/pchealth/helpctr/DataColl/CollectedData_1414.xml ($FILE_NAME) (deleted-realloc)
16	2017-05-01T04:10:59Z	8206	mac.	3762-128-3	/Documents and Settings/All Users/Application Data/VMware/Compatibility/virtual/opa12.dat
17	2017-05-01T04:10:59Z	8206	mac.	3762-128-3	/WINDOWS/pchealth/helpctr/DataColl/CollectedData_1256.xml (deleted-realloc)
18	2017-05-01T04:10:59Z	8206	mac.	3762-128-3	/WINDOWS/pchealth/helpctr/DataColl/CollectedData_1416.xml (deleted-realloc)
19	2017-05-01T04:10:59Z	84	mac.	3762-48-2	/Documents and Settings/All Users/Application Data/VMware/Compatibility/virtual/opa12.dat ($FILE_NAME)
20	2017-05-01T04:10:59Z	84	mac.	3762-48-2	/WINDOWS/pchealth/helpctr/DataColl/CollectedData_1256.xml ($FILE_NAME) (deleted-realloc)
21	2017-05-01T04:10:59Z	84	mac.	3762-48-2	/WINDOWS/pchealth/helpctr/DataColl/CollectedData_1416.xml ($FILE_NAME) (deleted-realloc)
22	2017-05-01T04:11:00Z	9386	...b	14419-128-3	/System Volume Information/_restore{76E4AA8D-B4CB-4EB7-9870-63A396AB60C6}/RP117/change.log

Fig. 5 *Parts of TPFSM-D of Run_Word*

the fls.exe tool. Then the mactime.pl is used to convert the output of the fls.exe to a human-readable one. As mentioned before, in order to reduce the volume of the output list and concentrate on the time of event, a start time or a time range can be defined. Moreover, to further reduce the number of output records, it is possible to remove the records related to access updates of files and folders. Fig. 4 shows the process of constructing the TPFSM-D of a hard disk.

Some parts of TPFSM-D of hard disk of a system on which Word application has been run are shown in Fig. 5. This copy of hard disk is named *Run_Word* and it is described in Table 3.

3.3.2 Distance calculation: The second component of the detection phase of our framework is distance calculator which calculates the distance between signature (TPFSM-A) of an application and TPFSM-D of a hard disk. This section describes the details of this component. The values of the File_Path attribute and their locations in TPFSMs are used to compute the distance between two TPFSMs. In order to compute the distance between two File_Path attributes in two TPFSMs, a string distance metric such as Levenshtein distance [36] can be used. However, since, in our problem, all the strings are file paths, another metric has been defined. In this regard, each file or folder name gets a unique code. Then these codes are assigned to different parts of a file path. For example, in Fig. 3, for file path '/WINDOWS/SoftwareDistribution/DataStore/Logs/edb.log', code 1 is assigned to WINDOWS, code 2 is assigned to SoftwareDistribution, code 3 is assigned to DataStore, code 4 is assigned to Logs, and finally, code 5 is assigned to edb.log. Thus, code 1–2–3–4–5 is assigned to the total file path. As another example, code 1–2–3–4–6 is assigned to file path '/WINDOWS/SoftwareDistribution/DataStore/Logs/edb.chk'. Fig. 6 shows the codes assigned to different parts of the File_Path attribute in TPFSM-A of *Word1*.

The similarity of the aforementioned file paths is 4, since they have four codes in common. In order to compute the distance between two file paths, it is possible to just subtract the obtained similarity from a constant value (such as the length of the longest file path). For example, the file path '/WINDOWS/WindowsUpdate.log' has code 1–8 and its similarity to the file path '/WINDOWS/SoftwareDistribution/DataStore/Logs/edb.log' is 1. By assuming that the length of the longest file path is 10, the distance between these two file paths is 9.

After computing the distances between file paths, a method should be used to compute the distance between one TPFSM-A

Table 3 Experiments of phase 2

Experiment number	Experiment name	Experiment description
Ex.1	*Run_Word*	clicking on the Word shortcut on the desktop, creating a new document named *doc1*, and then closing the application
Ex.2	*Run_Adobe*	clicking on the Adobe Reader shortcut on the desktop, opening *file1.pdf*, and then closing the application
Ex.3	*Run_Firefox*	clicking on the Firefox shortcut on the desktop, visiting *yahoo.com*, and then closing the application
Ex.4	*Run_Player*	opening *audio1.mp3* by clicking on it, playing the audio, and then closing the application
Ex.5	*Run_Word_Adobe*	clicking on the Word shortcut on the desktop, opening *doc1*, closing the application. Clicking on the Adobe Reader shortcut on the desktop, opening *file1.pdf*, and then closing the application
Ex.6	*Run_Word_Adobe_Firefox*	opening *doc1* by clicking on it, modifying the opened file, saving it, and then closing the application. Opening *file1.pdf* by clicking on it and then closing the application. Clicking on the Firefox shortcut on the desktop, visiting *google.com*, searching 'digital forensic', opening *digitalforensicsmagazine.com* in a new tab, and finally closing the application
Ex.7	*Run_Word_Adobe_Firefox_Player1 (Run_All1)*	clicking on the Firefox shortcut on the desktop, trying to visit *amazon.com*, an error occurred because the connection was not set up yet, then trying again, visiting *amazon.com*, following two consecutive links on the website, and finally closing the application. Opening *file2.pdf* by clicking on it and then closing the application. Clicking on the Word shortcut on the desktop, opening *doc2*, modifying the opened file, saving it, and then closing the application. Clicking on the Windows Media Player shortcut on the Start menu, opening and playing the *audio1.mp3*, and finally closing the application
Ex.8	*Run_Word_Adobe_Firefox_Player2 (Run_All2)*	clicking on the Adobe Reader shortcut on the desktop, opening *file1.pdf*, but not closing the application. Clicking on the Windows Media Player shortcut on the desktop, opening and playing the *audio1.mp3*, but not closing the application. Clicking on the Firefox shortcut on the desktop, visiting *google.com*, searching 'external hard disk', opening the link of the first result in a new tab, but not closing the application. Clicking on the Word shortcut on the desktop, opening *doc1*, but not closing the application

	1	2	3	4	5	6	7
1	'windows'	'softwaredis...	'datastore'	'logs'	'edb.log'	[]	[]
2	'windows'	'softwaredis...	'datastore'	'logs'	[]	[]	[]
3	'windows'	'softwaredis...	'datastore'	'logs'	'edb.chk'	[]	[]
4	'windows'	'softwaredis...	'datastore'	'datastore.e...	[]	[]	[]
5	'windows'	'windowsu...	[]	[]	[]	[]	[]
6	'windows'	'prefetch'	'java.exe-1e...	[]	[]	[]	[]
7	'documents...	'administrat...	'local settin...	'temporary ...	[]	[]	[]
8	'documents...	'administrat...	'local settin...	'temporary ...	'content.ie5'	[]	[]
9	'documents...	'administrat...	'local settin...	'history'	[]	[]	[]
10	'documents...	'administrat...	'local settin...	'history'	'history.ie5'	[]	[]
11	'documents...	'administrat...	'cookies'	[]	[]	[]	[]
12	'documents...	'administrat...	'local settin...	'temporary ...	'content.ie5'	'index.dat'	[]
13	'documents...	'administrat...	'local settin...	'history'	'history.ie5'	'index.dat'	[]
14	'documents...	'administrat...	'cookies'	'index.dat'	[]	[]	[]
15	'documents...	'administrat...	'local settin...	'temp'	'jusched.log'	[]	[]
16	'windows'	'system32'	'config'	'security.log'	[]	[]	[]
17	'program fil...	'microsoft ...	'office12'	'winword.exe'	[]	[]	[]
18	'documents...	'administrat...	'desktop'	'microsoft ...	[]	[]	[]
19	'system vol...	'_restore{76...	'rp114'	'a0139562.l...	[]	[]	[]
20	'documents...	'all users'	'application...	'microsoft'	'office'	'data'	'opa12.dat'

	1	2	3	4	5	6	7
1	1	2	3	4	5	0	0
2	1	2	3	4	0	0	0
3	1	2	3	4	6	0	0
4	1	2	3	7	0	0	0
5	1	8	0	0	0	0	0
6	1	9	10	0	0	0	0
7	11	12	13	14	0	0	0
8	11	12	13	14	15	0	0
9	11	12	13	16	0	0	0
10	11	12	13	16	17	0	0
11	11	12	18	0	0	0	0
12	11	12	13	14	15	19	0
13	11	12	13	16	17	19	0
14	11	12	18	19	0	0	0
15	11	12	13	20	21	0	0
16	1	22	23	24	0	0	0
17	25	26	27	28	0	0	0
18	11	12	29	30	0	0	0
19	31	32	33	34	0	0	0
20	11	35	36	37	38	39	40

Fig. 6 *Codes assigned to the File_Path attribute of TPFSM-A of Word1*

(e.g. *Word1*) and one TPFSM-D (e.g. *Run_Word*). This method works as follows: first, select top n records from TPFSM-D of *Run_Word* (name it $TPFSM_1$), n being the number of records of TPFSM-A of *Word1*. Second, construct a DTW matrix of dimension $n \times n$, named **DTW1**. Each element of this matrix is equal to the distance between two File_Path attribute values in TPFSM-A of *Word1* and $TPFSM_1$. For example, the first File_Path attribute value in TPFSM-A of *Word1* is '/WINDOWS/SoftwareDistribution/DataStore/Logs/edb.log', while the first File_Path attribute value in $TPFSM_1$ is '/WINDOWS/system32/config/system'. Thus, the first element of **DTW1**, which denotes the distance between the first records in two TPFSMs, is equal to 9 (it is assumed that the length of the longest file path is 10). Finally, the last element of **DTW1** is considered as the distance between TPFSM-A of *Word1* and $TPFSM_1$ which is named dis_1.

Then select top n records, starting from the second record of TPFSM-D of *Run_Word* (name it $TPFSM_2$) and calculate the distance between TPFSM-A of *Word1* and $TPFSM_2$. Similarly, in step i, in a sliding window manner, select top n records, starting from the ith record of TPFSM-D of *Run_Word* and compare it with TPFSM-A of *Word1*. The total number of steps is k, where $k = (m - n) + 1$, m being the number of records of TPFSM-D of *Run_Word*. Finally, the minimum obtained distance is considered as the distance between TPFSM-A and TPFSM-D, i.e. the distance between *Word1* and *Run_Word*. Fig. 7 shows the method of calculating the distance between TPFSM-A and TPFSM-D.

3.3.3 Decision engine: The last component of the detection phase is a decision engine which determines whether an application has been run on a system. In order to determine whether an application x has been run on the system, TPFSM-D constructor builds the TPFSM-D of the hard disk of the system. Then distance calculator computes the distance between the signature of x and the constructed TPFSM-D. If this distance is near to the indicator of x, i.e. if the difference of this distance and the indicator is lower than the threshold, decision engine concludes that x has been run on the system. Fig. 8 shows the decision engine which determines whether an application x has been run on the system.

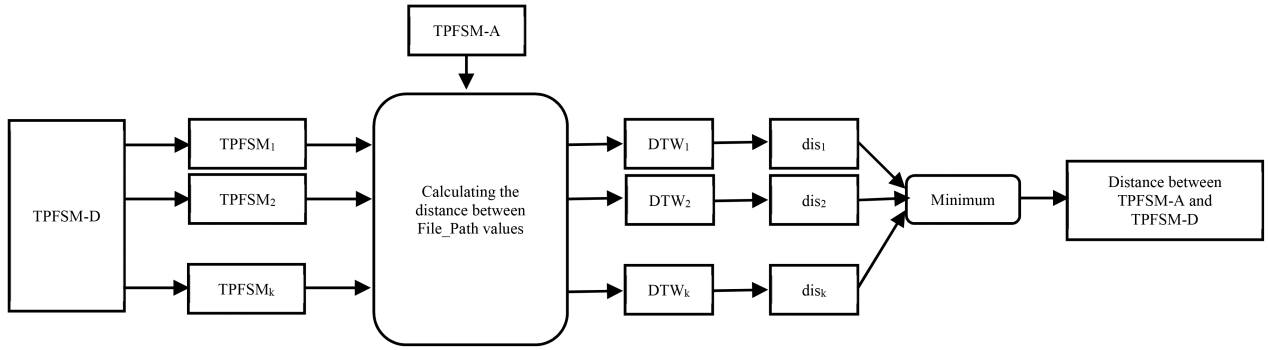

Fig. 7 *Calculating the distance between TPFSM-A and TPFSM-D*

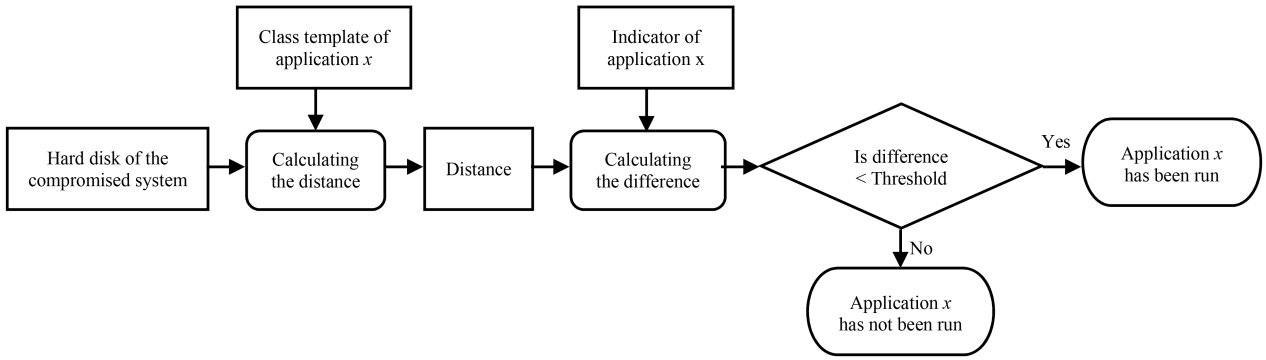

Fig. 8 *Determining whether application x has been run on a compromised system*

Table 4 Experiments of training phase, constructing TPFSM-Ds of the training system

Experiment name	Experiment description
Run_Word	clicking on the Word shortcut on the desktop, creating a new document named *doc1*, and then closing the application
Run_Adobe	clicking on the Adobe Reader shortcut on the desktop, opening *file1.pdf*, and then closing the application
Run_Firefox	clicking on the Firefox shortcut on the desktop, visiting *yahoo.com*, and then closing the application
Run_Player	opening *audio1.mp3* by clicking on it, playing the audio, and then closing the application

Table 5 Distances between Word experiments and *Run_Word*

Distance	*Run_Word*
Word1	133
Word2	138
Word3	164
Word4	146
mean	145.25

4 Experiments, results, and evaluation

By having a copy of hard disk of a compromised system, it is interesting to determine if specific applications have been run on the system. In this section, some experiments have been done to demonstrate the capability of the proposed event reconstruction method in detecting which applications have been run on the system. The experiments are performed in two phases. In phase 1, i.e. the training phase, several different applications are run separately in different scenarios, and the TPFSM-As of these applications are constructed. Next, for each application of phase 1, a signature, an indicator, and a threshold are defined. For phase 2, i.e. the detection phase, several copies of hard disks are presented. Traces of various applications can be found in these copies. In this phase, at first, for each copy of hard disk, the TPFSM-D is constructed. Then the distances between different signatures of applications of phase 1 and TPFSM-Ds of phase 2 are calculated. Next, it is shown that the decision engine decides whether one application of phase 1 has been run on a system of phase 2. Finally, the performance of the proposed event reconstruction method is evaluated.

4.1 Phase 1 experiments: training phase

In phase 1, Microsoft Word, Adobe Reader, Firefox, and Windows Media Player applications are run separately in different scenarios. In these experiments, first, a copy of hard disk is taken before running the application, using FTK Imager tool (this tool is not closed after taking copy). Then after terminating the application, a copy of hard disk is taken again, using the same opened tool. Finally, the TPFSM-A of them are calculated, as described in Section 3.2.1. Table 2 describes the process of launching, running, and terminating these applications.

For each of these applications, it is needed to define a signature of the application, an indicator of the existence of the application, and a threshold. For each application, as said before in Section 3.2, one of the TPFSM-As of that application is selected as the signature of the application. Here, the TPFSM-As of *Word2*, *Adobe1*, *Firefox2*, and *Player3* have been considered as the signature of the related applications.

In order to define an indicator of the existence of the application, it is needed to have a TPFSM-D of the hard disk of the training system. To this end, once again each of Microsoft Word, Adobe Reader, Firefox, and Windows Media Player applications are run separately, and for each experiment, the TPFSM-D of the hard disk is constructed. Table 4 elaborates details of these experiments. For each application, its indicator is defined as the mean distance between the TPFSM-As of the application in different scenarios and the aforementioned TPFSM-D of the hard disk. Tables 5–8 show the distances between the TPFSM-As of each application in different scenarios and the related TPFSM-D. Therefore, the indicators of the existence of Word, Adobe Reader, Firefox, and Windows Media Player are 145.25, 108.75, 738.25, and 144.75, respectively.

As it can be seen, values for the indicator of application in these experiments range from ∼100 to ∼750. Therefore, it is not appropriate to simply choose a constant threshold for all applications. In this section, for each application, the threshold is defined as 1/10 of the related indicator. Therefore, the thresholds for Word, Adobe Reader, Firefox, and Windows Media Player are 14.5, 10.9, 73.8, and 14.5, respectively. However, in order to achieve high accuracy and precision, the threshold will be adjusted in the evaluation phase in Section 4.3.

4.2 Phase 2 experiments: detection phase

In phase 2, Microsoft Word, Adobe Reader, Firefox, and Windows Media Player applications are run in different scenarios. In these experiments, no copy is taken before running the applications and the FTK Imager is run just after finishing the run of applications. Then the TPFSM-Ds of hard disk are calculated, as described in Section 3.3.1. Table 3 elaborates details of these experiments. As this table shows, in the first four experiments, just one application is run after the system boot up, then a copy of hard disk is taken. In the fifth experiment, two applications, first Word and then Adobe Reader, are run. In the sixth experiment, three applications, Word, Adobe Reader, and Firefox, are run, respectively. In the last two experiments, all of the four applications are run, one by one.

Now, the distances between TPFSM-Ds of these copies of hard disks and the signatures of various applications are calculated. Table 9 shows these distances. Using the calculated distances, the decision engine can determine whether Word, Adobe Reader, Firefox, or Windows Media Player applications have been run on the systems of phase 2 or not. For example, in order to determine whether the Word application has been run on a system such as *Ex.7*, the distance between the signature of Word application and TPFSM-D of *Ex.7* is calculated. The calculated distance is 144. Then the decision engine calculates the difference between this calculated distance and the indicator of Word (145.25). The difference is 1.25, and it is less than the threshold for Word, which is 14.5. Thus, it can be concluded that Word has been run in the *Ex.7* experiment. As another example, in order to determine whether the Firefox application has been run on a system such as *Ex.8*, the distance between the signature of the Firefox application and *Ex.8* is calculated, which is 802. Then the difference between this distance and the indicator of Firefox (738.25) is calculated, which is 63.75. The calculated difference is less than the threshold for Firefox, which is 73.8. Therefore, it is concluded that Firefox has been run on *Ex.8* system.

However, there are cases in which the decision engine does not decide correctly. For example, it cannot determine correctly that there are traces of Adobe Reader application in *Ex.6*. The distance between the signature of Adobe Reader application and *Ex.6* is 125. The difference between this distance and the indicator of Adobe Reader (108.75) is 16.25, which is greater than the threshold for Adobe Reader (10.9). Therefore, the decision engine decides that Adobe Reader has not been run in the *Ex.6* experiment, which is incorrect.

Table 6 Distances between Adobe experiments and *Run_Adobe*

Distance	*Run_Adobe*
Adobe1	109
Adobe2	90
Adobe3	121
Adobe4	115
mean	108.75

Table 7 Distances between Firefox experiments and *Run_Firefox*

Distance	*Run_Firefox*
Firefox1	460
Firefox2	813
Firefox3	783
Firefox4	897
mean	738.25

Table 8 Distances between Player experiments and *Run_Player*

Distance	*Run_Player*
Player1	174
Player2	94
Player3	152
Player4	159
mean	144.75

4.3 Evaluation

In order to evaluate the performance of the proposed event reconstruction method, several standard metrics including true positive rate (TPR), false positive rate (FPR), precision, and accuracy are employed. In this regard, the following definitions are presented:

- True positive (TP) is the number of applications which are truly decided as being run on the system.
- True negative (TN) is the number of applications which are truly decided as not being run on the system.
- False positive (FP) is the number of applications which are falsely decided as being run on the system.
- False negative (FN) is the number of applications which are falsely decided as not being run on the system.

The evaluation metrics achieved by the proposed event reconstruction method using different threshold values are illustrated in Table 10. The results show that when the threshold is equal to 1/10 of the related indicator, the TPR is 0.71, the FPR is 0, the precision is 1, and the accuracy is 0.84. As it can be seen in Table 10, the greater the threshold, the greater the TPR. The TPR reaches to 1 when the threshold is equal to 1/5 of the related indicator. On the other hand, as the results show, the greater the threshold, the greater the FPR and the smaller the precision. For example, when the threshold equals to 1/4 of the related indicator, the FPR is 0.2 and the precision is 0.85. The results suggest that the threshold which is equal to 1/6 of the related indicator is an appropriate one, because the TPR, FPR, precision, and accuracy are acceptable.

As Table 10 shows, when the threshold is equal to 1/6 of the related indicator, the precision and accuracy of the proposed method are both 0.94. Therefore, the proposed event reconstruction method can precisely and accurately determine whether an application has been run on a compromised system.

The proposed event reconstruction framework is compared with two signature-based event reconstruction methods presented in [2, 12]. The proposed event reconstruction method is similar to methods in [2, 12], with the difference that it does not rely on a

Table 9 Distances between signature of different applications and TPFSM-Ds of hard disks of phase 2

Distance	*Ex.1*	*Ex.2*	*Ex.3*	*Ex.4*	*Ex.5*	*Ex.6*	*Ex.7*	*Ex.8*
signature of Word	138	168	170	188	139	148	144	140
signature of Adobe Reader	141	109	134	141	114	125	119	125
signature of Firefox	1225	1189	813	1136	1424	726	699	802
signature of Windows Media Player	209	214	206	152	203	205	161	171

Table 10 Evaluation metrics achieved by the proposed event reconstruction method

Threshold	TPR	FPR	Precision	Accuracy
threshold = 1/10 of the related indicator	0.71	0	1	0.84
threshold = 1/9 of the related indicator	0.76	0	1	0.88
threshold = 1/8 of the related indicator	0.82	0	1	0.91
threshold = 1/7 of the related indicator	0.82	0	1	0.91
threshold = 1/6 of the related indicator	0.94	0.07	0.94	0.94
threshold = 1/5 of the related indicator	1	0.13	0.89	0.94
threshold = 1/4 of the related indicator	1	0.2	0.85	0.91

small number of updated timestamps. The event reconstruction methods in [2, 12] consider a small number of updated timestamps as the signature of an event. For example, the signature of 'opening Internet Explorer 8' in [12] has just six items, two of which are from file system and four are from registry. As another example, the event of 'opening Firefox 3.6' in [12] has just one item as the signature, which is the standard Windows pre-fetch file for Firefox. The proposed event reconstruction method, in contrast, does not consider a small number of updated timestamps as the signature of an application. For example, the signature of Firefox in the proposed method has >100 items.

Moreover, the proposed method does not simply try to match the signature of an application against the hard disk of the compromised system. Instead, it introduces a distance measure and calculates the distance between the signature of the application and the hard disk of the compromised system. Furthermore, for each type of application, it defines an indicator and a threshold. Then if the difference of the calculated distance and the indicator is lower than the threshold, it concludes that the application has been run on the system. Therefore, the proposed event reconstruction method is more resilient against anti-forensics attacks. While an attacker can easily bypass the signature-based methods in [2, 12] and hide his activities on a compromised system by changing the timestamps of a small number of files and folders, it is harder for him to subvert the proposed method.

5 Conclusion and future works

Event reconstruction is one of the most important phases of digital forensic. In this regard, proper high-level events should be reconstructed from low-level data. These data are extracted from different parts of a system including memory, hard disk, and network, using digital forensic tools. In this research, we have focused on event reconstruction using file system metadata. Running an application leaves some traces on the file system metadata, such as MACB timestamps. These timestamps can be represented as a time series, which in this paper is referred to as TPFSM.

In this paper, an event reconstruction framework has been proposed which determines whether an application has been run on a compromised system. The proposed framework has constructed the signature or the TPFSM-A and the TPFSM-D. Moreover, the framework has presented a distance metric which is used to calculate the distance between the signature of the application and TPFSM-D of the hard disk. Finally, the decision engine of the framework has used the calculated distance to decide whether the application has been run on the compromised system. The evaluation results show the effectiveness of the proposed framework in reconstructing events. The precision and accuracy of the proposed framework reached 94%.

Further work should be done on the distance metric which is used to calculate the distance between the signature of an application and a copy of hard disk. Moreover, in order to enhance the event reconstruction framework, we suggest integrating some traces from other parts of the system, such as memory, registry, and network. For example, the Plaso forensic tool, presented by Guðjónsson [9], which is used to automatically create a super timeline of various artefacts, including various log files, browsers history, registry entries, and file system metadata, can be used to integrate some pieces of evidence from different parts of the system. Although the proposed event reconstruction framework only focuses on Windows operating system, it can be applied to other operating systems. Further work should be done on this issue.

6 References

[1] Daniel, L., Daniel, L.: '*Digital forensics for legal professionals: understanding digital evidence from the warrant to the courtroom*' (Syngress Publishing, Rockland, Massachusetts, USA, 2011)

[2] Kalber, S., Dewald, A., Freiling, F.C.: 'Forensic application – fingerprinting based on file system metadata'. Proc. of the 2013 Seventh Int. Conf. on IT Security Incident Management and IT Forensics, Nuremberg, Germany, 2013, pp. 98–112

[3] Hargreaves, C., Patterson, J.: 'An automated timeline reconstruction approach for digital forensic investigations', *Digit. Invest.*, 2012, **9**, (Suppl.), pp. 69–79

[4] Khan, M.N.A., Chatwin, C.R., Young, R.C.D.: 'A framework for post-event timeline reconstruction using neural networks', *Digit. Invest.*, 2007, **4**, (3–4), pp. 146–157

[5] Carrier, B.: '*File system forensic analysis*' (Addison Wesley Professional, USA, 2005)

[6] Kalber, S., Dewald, A., Idler, S.: 'Forensic zero-knowledge event reconstruction on filesystem metadata'. Lecture Notes in Informatics, Vienna, Austria, 2014, pp. 331–343

[7] Carrier, B.: 'The Sleuth kit 2003'. Available from http://www.sleuthkit.org/index.php

[8] AccessData Group.: FTK Imager. Available from: http://accessdata.com/

[9] Guðjónsson, K.: 'Mastering the super timeline with log2timeline'. SANS Institute, InfoSec Reading Room, USA, 2010, pp. 1–84

[10] Olsson, J., Boldt, M.: 'Computer forensic timeline visualization tool', *Digit. Invest.*, 2009, **6**, (Suppl.), pp. 78–87

[11] Cohen, M.I.: 'Pyflag – an advanced network forensic framework', *Digit. Invest.*, 2008, **5**, (Suppl.), pp. 112–120

[12] James, J.I., Gladyshev, P., Zhu, Y.: 'Signature based detection of user events for post-mortem forensic analysis'. Digital Forensics and Cyber Crime, Abu Dhabi, United Arab Emirates, 2010, pp. 96–109

[13] Khan, M.N.A.: 'Performance analysis of Bayesian networks and neural networks in classification of file system activities', *Comput. Secur.*, 2012, **31**, (4), pp. 391–401

[14] Garfinkel, S.: 'Automating disk forensic processing with SleuthKit, XML and python'. Proc. of the Fourth Int. IEEE Workshop on Systematic Approaches to Digital Forensic Engineering, Berkeley, CA, USA, 2009, pp. 73–84

[15] Chen, K., Clark, A., De Vel, O., *et al.*: 'ECF – EVENT CORRELATION FOR FORENSICS'. 1st Australian Computer, Network & Information Forensics Conf., Perth, Western Australia, 2003, pp. 1–10

[16] Case, A., Cristina, A., Marziale, L., *et al.*: 'FACE: automated digital evidence discovery and correlation', *Digit. Invest.*, 2008, **5**, (Suppl.), pp. 65–75

[17] Raghavan, S., Raghavan, S.V.: 'AssocGEN: engine for analyzing metadata based associations in digital evidence'. Eighth Int. Workshop on Systematic Approaches to Digital Forensic Engineering (SADFE), Hong Kong, China, 2013, pp. 1–8

[18] Schatz, B., Mohay, G., Clark, A.: 'Rich event representation for computer forensics'. Proc. of the Fifth Asia-Pacific Industrial Engineering and Management Systems Conf. (APIEMS 2004), Gold Coast, Queensland, 2004, pp. 1–16

[19] Chabot, Y., Bertaux, A., Nicolle, C., *et al.*: 'An ontology-based approach for the reconstruction and analysis of digital incidents timelines', *Digit. Invest.*, 2015, **15**, pp. 83–100

[20] Gladyshev, P., Patel, A.: 'Finite state machine approach to digital event reconstruction', *Digit. Invest.*, 2004, **1**, (2), pp. 130–149

[21] James, J., Gladyshev, P., Abdullah, M.T., *et al.*: 'Analysis of evidence using formal event reconstruction'. Digital Forensics and Cyber Crime, Albany, NY, USA, 2009, pp. 85–98

[22] Zhu, Y., Gladyshev, P., James, J.: 'Using shellbag information to reconstruct user activities', *Digit. Invest.*, 2009, **6**, (Suppl.), pp. 69–77

[23] Chabot, Y., Bertaux, A., Nicolle, C., *et al.*: 'A complete formalized knowledge representation model for advanced digital forensics timeline analysis', *Digit. Invest.*, 2014, **11**, (2), pp. 95–105

[24] Raghavan, S, Clark, A, Mohay, G.: 'FIA: an open forensic integration architecture for composing digital evidence', *Forensics Telecommun., Inf. Multimed.*, 2009, **8**, pp. 83–94

[25] Hilgert, J.-N., Lambertz, M., Plohmann, D.: 'Extending the Sleuth kit and its underlying model for pooled storage file system forensic analysis', *Digit. Invest.*, 2017, **22**, (Suppl.), pp. 76–85

[26] Okolica, J., Peterson, G.L.: 'Extracting the windows clipboard from physical memory', *Digit. Invest.*, 2011, **8**, (Suppl.), pp. 118–124

[27] White, A., Schatz, B., Foo, E.: 'Surveying the user space through user allocations', *Digit. Invest.*, 2012, **9**, (Suppl.), pp. 3–12
[28] Stevens, R.M., Casey, E.: 'Extracting windows command line details from physical memory', *Digit. Invest.*, 2010, **7**, (Suppl.), pp. 57–63
[29] Bridge, A.: 'Obtaining forensic value from the cbWndExtra structures as used by windows common controls, specifically for the Editbox control', *Digit. Invest.*, 2017, **20**, pp. 54–60
[30] Xie, G., Iliofotou M, ., Karagiannis, T., *et al.*: 'Resurf: reconstructing web-surfing activity from network traffic'. IFIP networking Conf., Brooklyn, NY, USA, 2013, pp. 1–9
[31] Gugelmann, D., Gasser, F., Ager, B., *et al.*: 'Hviz: HTTP(S) traffic aggregation and visualization for network forensics', *Digit. Invest.*, 2015, **12**, (Suppl. 1), pp. 1–11
[32] Spiekermann, D., Keller, J., Eggendorfer, T.: 'Network forensic investigation in OpenFlow networks with ForCon', *Digit. Invest.*, 2017, **20**, (Suppl.), pp. 66–74
[33] Russinovich, M.: 'Process monitor v3.2 2015', Available at https://technet.microsoft.com/en-us/sysinternals/bb896645.aspx
[34] NirSoft.: BulkFileChanger v1.33. Available at http://www.nirsoft.net/utils/bulk_file_changer.html
[35] Grig Software.: Compare It! 4. Available at http://www.grigsoft.com/wincmp3.htm
[36] Levenshtein, V.: 'Binary codes capable of correcting deletions insertions and reversals', *Dokl. Phys.*, 1966, **10**, (8), pp. 707–710