Chapter 6

A CONSISTENCY STUDY OF THE WINDOWS REGISTRY

Yuandong Zhu, Joshua James and Pavel Gladyshev

Abstract This paper proposes a novel method for checking the consistency of forensic registry artifacts by gathering event information from the artifacts and analyzing the event sequences based on the associated timestamps. The method helps detect the use of counter-forensic techniques without focusing on one particular counter-forensic tool at a time. Several consistency checking models are presented to verify events derived from registry artifacts. Examples of these models are used to demonstrate how evidence of alteration may be detected.

Keywords: Windows forensics, registry analysis, counter-counter-forensics

1. Introduction

Electronic devices often contain large amounts of data of evidentiary value. However, since it is possible for a suspect to alter the devices through software or hardware, it is extremely important in digital investigations to determine whether or not the evidence collected from seized devices has been modified [5].

This paper advocates the use of information obtained from the Windows registry to verify the consistency of the collected evidence. The Windows registry is a database that stores information about the hardware, software and user profiles of a Windows machine [6]. As such, it an important resource for identifying events that occurred during the use of the machine [1]. During the normal execution of an operating system, certain events always occur in the same order. Therefore, if information about some events is obtained from the registry, then correlating known sequences of these events in temporal order of their timestamps gives a clue as to whether or not the evidence is internally consistent. For example, the event involving the installation of a particular software system for the first time must occur before the event involving the running of the software for the first time. If the timestamp associated with installing the software is later than the timestamp associated with running the software, either the timestamp itself or other information about the event was tampered with.

This paper proposes a method to verify the consistency of registry artifacts by obtaining event information from the artifacts and examining their associated timestamps in event sequences. This provides a generic way to detect the use of counter-forensic techniques without focusing on one particular counter-forensic tool at a time. The method also helps detect when timestamps were altered, which gives a digital forensic investigator additional information about the user's activities.

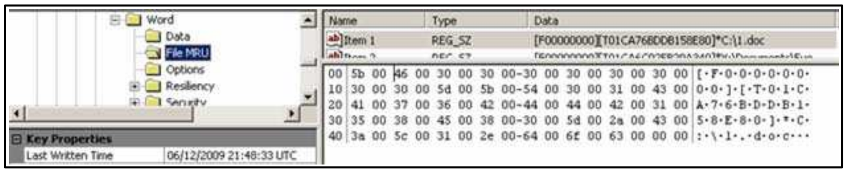
2. Related Research

Previous research on detecting counter-forensic techniques has proceeded along two lines. One approach (see, e.g., [2]) tends to solve the problem from very specific perspective. Experiments are performed to understand the behavior of a particular counter-forensic tool; forensic investigators are then provided with information about where traces of the tool may be found. Although the approach is effective, it suffers from the limitation that the results may not apply to other tools. Indeed, it is almost impossible to develop an individual method against each tool because there are so many counter-forensic tools available [5].

The second, generic approach engages methods that check the consistency of system properties. For example, Gladyshev and Enbacka [3] developed an algorithm to check the consistency of log files. Willassen [7] proposed a technique for discovering evidence of "antedating" by studying the sequence number allocation properties of storage systems. Motivated by these techniques, this paper proposes a general method for verifying the consistency of collected evidence. Although this method may not always be as effective as searching for traces of a counter-forensic tool, it can be used against a wide range of counter-forensic techniques because inconsistencies are detected regardless of the specific technique used to tamper with evidence.

3. Consistency Checking Method

The consistency checking method proposed in this paper involves two steps: (i) obtain events and their timestamp information from the Windows registry; and (ii) verify the consistency of the identified events using the appropriate consistency checking model.



Name	Type	Data
Item 1	REG_SZ	[F00000000][T01CA76BDDB158E80]*C:\1.doc

```
00  5b 00 46 00 30 00 30 00-30 00 30 00 30 00 30 00  [·F·0·0·0·0·0·0·
10  30 00 30 00 5d 00 5b 00-54 00 30 00 31 00 43 00  0·0·]·[·T·0·1·C·
20  41 00 37 00 36 00 42 00-44 00 44 00 42 00 31 00  A·7·6·B·D·D·B·1·
30  35 00 38 00 45 00 38 00-30 00 5d 00 2a 00 43 00  5·8·E·8·0·]·*·C·
40  3a 00 5c 00 31 00 2e 00-64 00 6f 00 63 00 00 00  :·\·1·.·d·o·c···
```

Key Properties	
Last Written Time	06/12/2009 21:48:33 UTC

Extracted Event 1: File MRU key was updated at 06/12/2009 21:48:33 UTC

Extracted Event 2: Value "[F00000000][T01CA76BDDB158E80]*c:\1.doc" was updated into the File MRU key before 06/12/2009 21:48:33 UTC

Figure 1. Event extraction.

3.1 Events and Timestamps

All events obtained from the registry are categorized as extracted events or inferred events. Extracted events are associated with information that was updated at specific registry locations at specific times; these events can be directly extracted from the registry. Inferred events are deduced from the registry contents based on known relationships between registry information and user and system actions.

Extracted Events

An extracted event is either an update of a registry key or a data value corresponding to a key. The time of the update is determined by the LastWrite timestamp of the associated registry key, which specifies the most recent modification time of the key. Figure 1 illustrates the extraction of events from *HKCU\Software\Microsoft\Office\12.0\Word\File MRU*. If the extracted event (Ext Event) corresponds to the updating behavior of the key, then the timestamp can be written as:

$$T^{Ext\ Event(Key)} = T^{LastWrite} \tag{1}$$

Since each data value causes the LastWrite time of the associated key to be updated, the timestamp of the event is estimated by:

$$T^{Ext\ Event(Value\ Data)} \leq T^{LastWrite} \tag{2}$$

Upon combining Equations 1 and 2, the timestamp estimation equation that applies to all possible extracted events is given by:

$$T^{Ext\ Event} \leq T^{LastWrite} \tag{3}$$

A better way to estimate the timestamp for an extracted event associated with a particular record is to use the registry snapshot comparison method [9]. This method helps bind an extracted event to a specific time interval. The comparison method is based on the fact that many Microsoft Windows versions automatically back up system registry hives, creating "system restore points" approximately every 24 hours. By considering each registry snapshot as a previous state of the system registry and comparing a given state of the registry with previous registry snapshots, changes made to the registry between the creation times of two consecutive restore points can be identified. When previous registry snapshots are available, comparing each registry snapshot with its preceding snapshot can identify extracted events that are known to have occurred before the LastWrite timestamp of the key and also after the creation time of the preceding registry snapshot:

$$T^{Prec\,Snapshot} \leq T^{Identified\,Ext\,Event} \leq T^{LastWrite} \tag{4}$$

Inferred Events

Carvey [1] has described the inference of user and system events from the registry. In general, the time interval between an inferred event and the action that triggered the corresponding update in the registry (i.e., the corresponding extracted event) influences the selection of the consistency checking model that is applied. Thus, inferred events are divided into three groups:

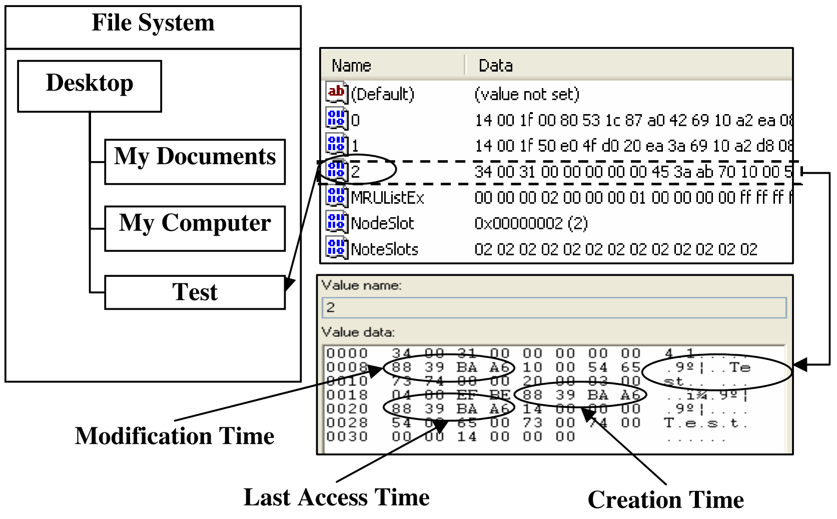
- Inferred events that occurred before the corresponding extracted event.
- Inferred events that occurred "at the same time" as the corresponding extracted event.
- Inferred events that occurred after the corresponding extracted event.

Note that an inferred event and its associated extracted event do not occur simultaneously. However, if the time interval is short enough, the extracted event can be assumed to have occurred "at the same time" as the action that precipitated it.

In the following, we present four examples of inferred events to demonstrate how events may be inferred from registry information.

The first example involves ShellBag information associated with the "Test" folder on a local computer (Figure 2). As described in [8], the registry value associated with this folder includes the timestamp when the ShellBag information was first updated in the registry. Therefore, three events can be inferred:

- Folder "Test" located on the Desktop was created at `08/12/2008 20:53:52`.



Name	Data
(Default)	(value not set)
0	14 00 1f 00 80 53 1c 87 a0 42 69 10 a2 ea 08
1	14 00 1f 50 e0 4f d0 20 ea 3a 69 10 a2 d8 08
2	34 00 31 00 00 00 00 00 45 3a ab 70 10 00 5
MRUListEx	00 00 00 02 00 00 00 01 00 00 00 00 ff ff ff f
NodeSlot	0x00000002 (2)
NoteSlots	02 02 02 02 02 02 02 02 02 02 02 02 02

Value name:
2
Value data:

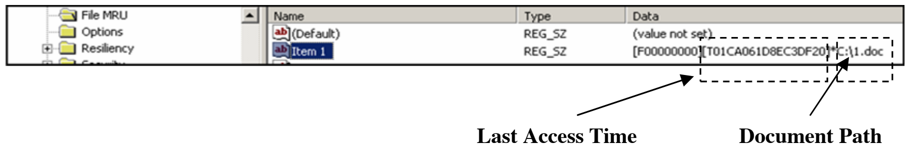
```
0000   34 00 31 00 00 00 00 00   4.1.....
0008   88 39 BA A6 10 00 54 65   .9º¦..Te
0010   73 74 00 00 20 00 03 00   st......
0018   04 00 EF BE 88 39 BA A6   ..ï¾.9º¦
0020   88 39 BA A6 14 00 00 00   .9º¦....
0028   54 00 65 00 73 00 74 00   T.e.s.t.
0030   00 00 14 00 00 00         ......
```

Figure 2. ShellBag information associated with the "Test" folder.

- Folder "Test" located on the Desktop was accessed at `08/12/2008 20:53:52`.

- Folder "Test" located on the Desktop was modified at `08/12/2008 20:53:52`.

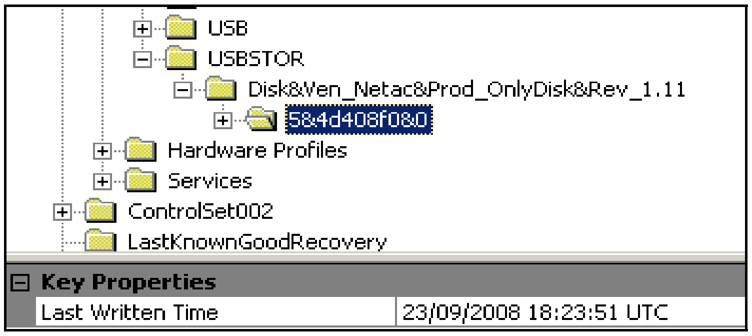
In this example, more than one event is inferred from a single registry record. Since the consistency checking method is based on the events and timestamps that are found, it is important to infer as many events as possible from the registry. Note that all three events are known to have occurred before the corresponding extracted event.



Name	Type	Data
(Default)	REG_SZ	(value not set)
Item 1	REG_SZ	[F00000000][T01CA061D8EC3DF20]*C:\1.doc

Figure 3. FileMRU key.

The second example involves the FileMRU key, which is located at *HKCU\Software\Office\12.0\Microsoft\Word\FileMRU* (Figure 3). It stores the paths of Word documents that have recently been opened by Microsoft Office. The information in Figure 3 implies the (inferred) event that a Word document *C:\1.doc* was accessed at `13:58:55 16/07/2009` corresponding to the Windows 64-bit timestamp `01CA061D8EC3DF20`.



Last Written Time 23/09/2008 18:23:51 UTC

Figure 4. USBSTOR subkey.

The third example involves the USBSTOR subkey located at *HKLM\System\ControlSet01\Enum\USBSTOR* (Figure 4). The inferred event is that a USB device Disk&Ven_Netac&Pro_OnlyDisk&Rev_1.11 was connected to the system at `23/09/2008 18:23:51`. The timestamp is estimated using the extracted event: Registry key *HKLM\System\ControlSet-01\Enum\USBSTOR\Disk&Ven_Netac&Pro_OnlyDisk&Rev_1.11\5&4d408f08&0* was updated at `23/09/2008 18:23:51`. The timestamp of the extracted event can be used for this inferred event because the inferred event belongs to the second type of inferred event (defined above), where the extracted event is assumed to have occurred at the same time as the inferred event.

The fourth example corresponds to the third type of inferred event that occurred after the corresponding extracted event. The value Lease TerminatesTime under the registry key *HKLM\SYSTEM\ControlSet001\Services\Tcpip\Parameters\Interfaces\AdapterID* gives the time that the IP address of the adapter expired. This definitely occurred after the adaptor was connected.

3.2 Consistency Checking Models

After obtaining event information from the registry, it is necessary to verify the events and their associated timestamp information. This section describes several consistency models that may be used for this purpose. Because each event can be placed in a different position in a time sequence when grouped with other events, it is difficult, if not impossible, to define a consistency checking model for an event without considering the details of other events. We address this issue in a generic manner using a context-based model. Each model is defined with respect to a specific context; the model is applied when the events meet the associated conditions.

Basic Model

The basic event-time bounding model is used to estimate the time frame during which a particular event without time information occurred. This is done by considering its relation in time to other events that are known to have occurred either before or after the event [4]. The same concept can also be applied to user data. Since this model also represents the relationships between a sequence of multiple events and their timestamps, it is the basis of other consistency checking models. The event-time bounding model utilizes two rules:

- **Rule 1:** If Event A occurred before Event B, and Event B occurred before Event C, then the time that Event B occurred is bounded by the times that Events A and C occurred:

$$T^A < T^B < T^C \tag{5}$$

- **Rule 2:** If several Events A_1, A_2, ..., A_m occurred before Event B, and Event B occurred before several Events C_1, C_2, ..., C_n, then the time that Event B occurred is bounded by:

$$Max(T^{A_1},\ T^{A_2}, \ldots, T^{A_m}) < T^B < Min(T^{C_1},\ T^{C_2}, \ldots, T^{Cn}) \tag{6}$$

Checking Inferred Events and Extracted Events

The basic model was developed to verify the consistency of inferred events and extracted events. As mentioned earlier, there are three types of inferred events. Therefore, based on their relationship with the corresponding extracted event, three equations can be derived using Equations 3 and 5.

- For an inferred event (Inf Event) that occurred before the extracted event, we have:

$$T^{Inf\ Event} < T^{Ext\ Event} \leq T^{LastWrite}$$

- For an inferred event that occurred at the same time as the extracted event, we have:

$$T^{Inf\ Event} = T^{Ext\ Event} \leq T^{LastWrite}$$

- For an inferred event that occurred after the extracted event, we have:

$$T^{Inf\ Event} > T^{Ext\ Event}$$

or, if the time difference Δ between the action and a future action is known, we have:

$$T^{Inf\ Event} = T^{Ext\ Event} + \Delta$$

In the example in Figure 2, if the LastWrite timestamp of the key is `07/12/2008 11:00:00`, then based on the model for inferred events that occurred before the corresponding extracted event, the information conflicts with the inferred event: Folder "Test" was created at `08/12/2008 20:53:52`.

Additionally, if multiple inferred events are identified as being associated with the same extracted event, the timestamp of the extracted event will be affected by all the associated inferred events. Therefore, the consistency checking models must be extended according to Equations 3 and 6.

- For inferred events that occurred before the extracted event, we have:
$$Max(T^{Inf\ Events}) < T^{Ext\ Event} \leq T^{LastWrite}$$

- For inferred events that occurred at the same time as the extracted event, we have:
$$Max(T^{Inf\ Events}) = T^{Ext\ Event} \leq T^{LastWrite}$$

- For inferred events that occurred after the extracted event, we have:
$$Min(T^{Inf\ Events}) > T^{Ext\ Event}$$

The application of the model is illustrated using the UserAssist key at *HKCU\Software\Windows\CurrentVersion\Explorer\UserAssist.* This key stores a list of values that record information about the execution of software on the computer. Each time a particular software executes, the corresponding value is updated under the UserAssist key. This falls under the model for inferred events that occurred at the same time as the corresponding extracted event. Assume that three values are found under the UserAssist key as shown in Table 1. Upon applying the model, Max($T^{Inf\ Events}$) is equal to `15:35:12 14/05/2009`. If the LastWrite timestamp is `20:04:31 25/04/2009`, then the model proves that the event information is inconsistent.

Table 1. Interpreted information for the UserAssist key.

Software Path	**Timestamp**
C:\Program Files\Internet Explorer\iexplorer.exe	`10:03:01 05/03/2009`
C:\WINDOWS\system32\notepad.exe	`15:35:12 14/05/2009`
C:\Program Files\Windows Media Player\wmplayer.exe	`19:11:52 22/01/2009`

Checking Inferred Events

When examining the registry, it is possible to infer events from different locations in the registry that point to the same user or system action. This is because information is sometimes saved at multiple locations in the registry. The consistency of inferred events can be verified using these related pieces of data.

Zhu, *et al.* [8] have proposed several rules that use ShellBag information to determine if a folder was accessed. Specifically, the information under the BagMRU registry key (located at *HKCU\Software\Microsoft\Windows\ShellNoRoam\BagMRU*) and the Bags key (located at *HKCU\Software\Microsoft\Windows\ShellNoRoam\Bags*) can be used to determine if the folder was accessed during a particular period. If one event, i.e., the folder was accessed at `10/07/2009 7:30:00` is inferred from information in the Bags key, and another event, i.e., the same folder was not accessed between `09/07/2009 9:12:00` and `11/07/2009 18:20:30` is inferred from the BagMRU key, then it follows that the two inferred events are not consistent.

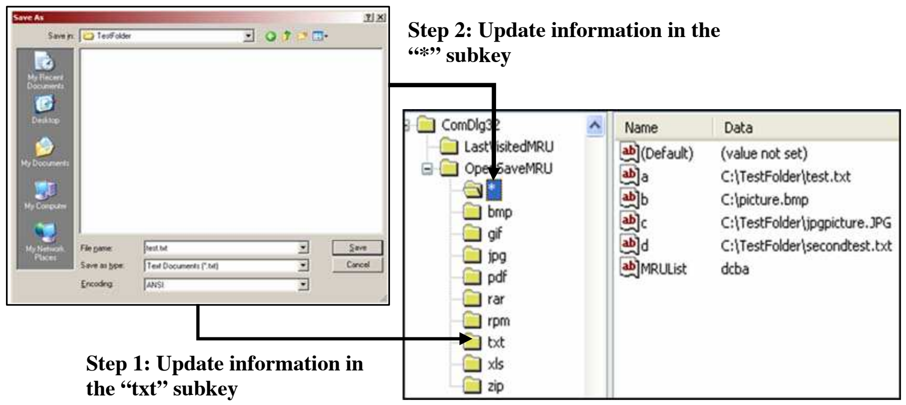
Checking Extracted Events

The consistency between extracted events is due to the fact that some registry keys are always updated after other keys. When events pertaining to registry operation are extracted, their timestamps should appear in the same order. By applying the basic model to this relationship it is possible to say that: if Key B is always updated after Key A, then the most recent extracted event of Key B is definitely after the most recent extracted event of Key A:

$$Max(T^{Ext\ Events(Key\ A)}) < Max(T^{Ext\ Events(Key\ B)})$$

Because the most recent extracted event of a registry key is equal to the LastWrite timestamp of the key, the consistency check can be expressed as:

$$T^{LastWrite(KeyA)} < T^{LastWrite(KeyB)}$$



Save in: TestFolder

File name: test.txt

Save as type: Text Documents (*.txt)

Encoding: ANSI

Name	Data
(Default)	(value not set)
a	C:\TestFolder\test.txt
b	C:\picture.bmp
c	C:\TestFolder\jpgpicture.JPG
d	C:\TestFolder\secondtest.txt
MRUList	dcba

Figure 5. OpenSaveMRU update example.

Similarly, if Key B is always updated after other Keys A_1, A_2, ..., A_n, then the timestamp of Key B is constrained by the most recent extracted events:

$$Max(Max(T^{Ext\ Events(Key\ A_1)}), \ldots, Max(T^{Ext\ Events(Key\ A_n)})) \\ < Max(T^{Ext\ Events(Key\ B)})$$

Upon replacing the most recent extracted event with the LastWrite timestamp of the key, the consistency check can be written as:

$$Max(T^{LastWrite(Key\ A_1)}, \ldots, T^{LastWrite(Key\ A_n)}) < T^{LastWrite(Key\ B)}$$

This model can be used to check the consistency of event information stored in the OpenSaveMRU key at *HKCU\Software\Microsoft\Windows\CurrentVersion\Explorer\ComDlg32\OpenSaveMRU*. The key is known to record the most recently used history of the open and save dialog of the Windows system. For example, downloading a file and using the open and save window to save the file in a local directory updates this key with information about the selected directory.

Figure 5 presents the structure of the OpenSaveMRU key. Subkeys in OpenSaveMRU correspond to files with extensions (e.g., "jpg" corresponding to a `.jpg` file and "rar" corresponding to a compressed `.rar` file). The OpenSaveMRU key itself saves information about files without extensions. The only exception is that the subkey "*" records directories for every file.

The algorithm used to update the OpenSaveMRU key and its subkeys checks if the open and save window is used. If it is, the path of the file is stored in the "*" subkey after updating the other file extension

Table 2. Interpreted information of OpenSaveMRU.

Registry Key Path	**Timestamp**
OpenSaveMRU\bmp	`07:42:16 13/07/2009`
OpenSaveMRU\jpg	`22:12:28 21/07/2009`
OpenSaveMRU\txt	`21:15:42 09/07/2009`
OpenSaveMRU\*	`19:27:09 14/07/2009`

type keys. In the example in Figure 5, the file *C:\TestFolder\test.txt* was first stored in the "txt" subkey, and then saved in the "*" subkey. That is to say, the "*" key is always updated after other OpenSaveMRU keys or subkeys. Thus, the "*" key satisfies the conditions described above. Table 2 presents the timestamps of all the keys; the information is inconsistent because the timestamp of the OpenSaveMRU\txt key is later than the timestamp of the OpenSaveMRU* key.

Checking Registry Events and Other Events

Apart from the registry database, there are other sources that provide information about events that have occurred. Two important sources of events that should be checked for consistency are file timestamps and registry snapshots.

- **File Timestamps:** In most versions of Microsoft Windows, three timestamps are associated with a file, corresponding to when the file was last accessed, last modified and created. Each extracted event obtained from the registry indicates an update of the registry file, so the extracted event should cause the modification timestamp of the registry file to be updated if the file handle is properly closed after the update. The consistency check for this situation is:

  $$Max(T^{Ext\ Event\ 1}, \ldots, T^{Ext\ Event\ n}) \leq T^{Reg\ Mod\ Time}$$

  A better method to implement this check is to compare the Last-Write time of each extracted event with the last modification time of each file because Max($T^{ExtEvents}$) is equal to $T^{LastWriteKey}$. The consistency check for this situation is:

  $$Max(T^{LastWrite\ 1}, \ldots, T^{LastWrite\ n}) \leq T^{Reg\ Mod\ Time}$$

  Note that this inequality does not apply when the registry file handle is not closed properly due to abnormal termination.

- **Registry Snapshots:** Each registry snapshot is created by updating the contents of the preceding registry snapshot. Therefore, any events that have occurred between two snapshots will appear after the creation time of the preceding snapshot:

  $$T^{Prec\ Snapshot} < Min(T^{Ext\ Event\ 1}, \ldots, T^{Ext\ Event\ n})$$

  $$T^{Prec\ Snapshot} < Min(T^{Inf\ Event\ 1}, \ldots, T^{Inf\ Event\ n})$$

  Whether an extracted event has occurred or not is determined by comparing the timestamps of a registry key in the snapshot and in the next snapshot. If the timestamp is updated in the newest snapshot, then it implies that the corresponding extracted event did occur. According to Equation 3, the timestamp of the extracted event cannot be after the updated LastWrite timestamp. Therefore, the consistency check for this situation is:

  $$T^{Prec\ Snapshot} \leq Min(T^{Updated\ LastWrite\ 1}, \ldots, T^{Updated\ LastWrite\ n})$$

3.3 Other Consistency Checking Considerations

After identifying inconsistent information, there may be a need to understand how the inconsistency was created. As mentioned above, if the system runs without any intentional changes, it will be consistent all the time. Some of the reasons for inconsistent information are:

- **System Clock Adjustment:** Many timestamps are based on the current system clock, especially the LastWrite timestamp associated with each key. If the system clock is temporarily adjusted, it may leave detectible inconsistencies in the timestamps recorded during the altered time period. A consistency check may identify these inconsistencies if they are not overwritten by new information.

- **Registry Information Modification:** It is often the case that the registry API is used to modify registry information. While any part of the registry can be modified, the LastWrite timestamp of the key is also updated because the system considers the invocation of the API as a normal registry operation. Therefore, it is possible to identify this trace by examining the consistency of extracted events. An inconsistent timestamp may imply that tampering has occurred. For example, in Table 2, if RegEdit was used to modify the contents of the `jpg` subkey at `22:12:28 21/07/2009`, the inconsistency shown in Table 2 would be produced.

- **Registry Hive Modification:** A file editor tool can be used to modify the registry hive directly. This is hard to implement in practice because it requires the user to understand the unique structure of the particular registry hive. Also, it is difficult to detect if the registry has been modified in this manner because the registry is not automatically updated when the modification is made unless the timestamp is also edited at the same time.

4. Conclusions

The method for checking the consistency of registry information involves extracting and inferring events and their corresponding timestamps from the registry database. Appropriate consistency checking models are then used to verify the information that is collected and help detect counter-forensic activity. Our future research will examine the potential of using other registry information as well as data from other sources in sophisticated consistency models.

Acknowledgements

This research was funded by the Science Foundation of Ireland under the Research Frontiers Program 2007 Grant No. CMSF575.

References

[1] H. Carvey, *Windows Forensic Analysis*, Syngress, Burlington, Massachusetts, 2007.

[2] M. Geiger and F. Cranor, Counter-Forensic Privacy Tools: A Forensic Evaluation, Technical Report CMU-ISRI-05-119, Institute for Software Research International, Carnegie-Mellon University, Pittsburgh, Pennsylvania (reports-archive.adm.cs.cmu.edu/anon/isri2005/CMU-ISRI-05-119.pdf), 2005.

[3] P. Gladyshev and A. Enbacka, Rigorous development of automated inconsistency checks for digital evidence using the B method, *International Journal of Digital Evidence*, vol. 6(2), pp. 1–21, 2007.

[4] P. Gladyshev and A. Patel, Formalizing event time bounding in digital investigations, *International Journal of Digital Evidence*, vol. 4(2), pp. 1–14, 2005.

[5] S. Hilley, Anti-forensics with a small army of exploits, *Digital Investigation*, vol. 4(1), pp. 13–15, 2007.

[6] Microsoft Corporation, Windows registry information for advanced users, Redmond, Washington (support.microsoft.com/kb/256986), 2008.

[7] S. Willassen, Hypothesis-based investigation of digital timestamps, in *Advances in Digital Forensics IV*, I. Ray and S. Shenoi (Eds.), Springer, Boston, Massachusetts, pp. 75–86, 2008.

[8] Y. Zhu, P. Gladyshev and J. James, Using ShellBag information to reconstruct user activities, *Digital Investigation*, vol. 6(S1), pp. S69–S77, 2009.

[9] Y. Zhu, J. James and P. Gladyshev, A comparative methodology for the reconstruction of digital events using Windows restore points, *Digital Investigation*, vol. 6(1-2), pp. 8–15, 2009.