Chapter 6

TEMPORAL ANALYSIS OF WINDOWS MRU REGISTRY KEYS

Yuandong Zhu, Pavel Gladyshev and Joshua James

Abstract The Microsoft Windows registry is an important resource in digital forensic investigations. It contains information about operating system configuration, installed software and user activity. Several researchers have focused on the forensic analysis of the Windows registry, but a robust method for associating past events with registry data values extracted from Windows restore points is not yet available. This paper proposes a novel algorithm for analyzing the most recently used (MRU) keys found in consecutive snapshots of the Windows registry. The algorithm compares two snapshots of the same MRU key and identifies data values within the key that have been updated in the period between the two snapshots. User activities associated with the newly updated data values can be assumed to have occurred during the period between the two snapshots.

Keywords: MRU registry keys, restore points, registry snapshots

1. Introduction

The Microsoft Windows registry is "a central hierarchical database" [8] that contains information (stored as keys) related to users, hardware devices and applications installed on the system. As such, it is an important forensic resource that holds a significant amount of information about user activities. This paper focuses on the most recently used (MRU) keys that contain data values (file names, URLs, command line entries, etc.) related to recent user activity [1]. An example is the key `HKCU\Software\Microsoft\Office\12.0\Word\File MRU` that stores the list of recently opened Microsoft Word 2007 documents (Figure 1).

Several MRU keys are used throughout the Windows operating system. Some keys have "MRU" in their names, such as `OpenSaveMRU`,

G. Peterson and S. Shenoi (Eds.): Advances in Digital Forensics V, IFIP AICT 306, pp. 83–93, 2009.

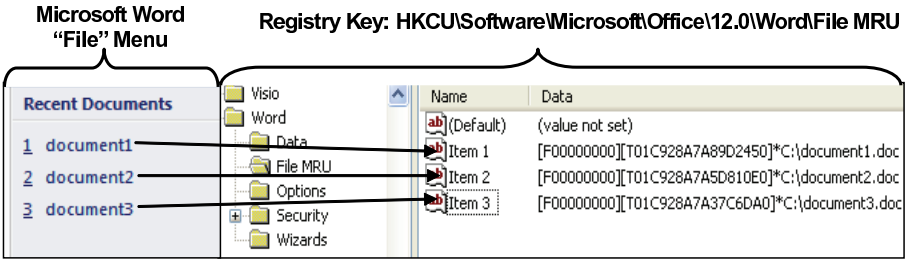

Figure 1. Word file MRU example.

which contains the names of files recently saved by applications that use the standard Microsoft Windows OpenAndSave shell dialog. Others reflect the nature of items in the key such as `TypedURLs`, which contains a list of the URLs typed into the Internet Explorer address bar by the user. The connections between MRU keys and user actions can be found in several publications (see, e.g., [1, 2, 7]). MRU keys are particularly useful in investigations where it is important to determine the actions performed by specific users [9].

This paper describes an algorithm for analyzing MRU keys in consecutive snapshots of the Windows registry. User activities occurring during the period between the snapshots can be identified by analyzing the updated data values corresponding to a specific MRU key.

2. Analysis of Registry Snapshots

The Windows registry is stored in the file system in blocks called "hives." System Restore Point hives are backups of the Windows registry created every 24 hours or upon installation of new software [3]. The hives contain several earlier versions of the registry. These versions, which we call "snapshots," can provide an investigator with a detailed picture of how the registry changed over time [4].

Forensic analysis of the registry rarely focuses on the restore points. At best, registry snapshots within the restore points are examined as separate entities during an investigation. This overlooks the links between registry snapshots, which provide more timestamp information about user activities and system behavior than single instances of the Windows registry.

The forensic value of Windows registry information is limited by the relative scarcity of timestamp information [1]. Only one timestamp, the last modification time, is recorded for each registry key [2]. A registry key usually contains multiple data values. Even if it could be determined which value was the last to be updated, it is not possible to determine

when the other data values were last modified. This is problematic when an investigator attempts to construct a timeline of events by combining information from several registry keys using only a single instance of the registry.

This problem can be addressed by comparing consecutive registry snapshots and determining which data values were updated. Each registry snapshot is usually associated with a creation time that is recorded as part of the snapshot. Any registry data value updated between two registry snapshots must have been updated between the creation time of the preceding registry snapshot and the last modification time of the registry key that contains the newly updated data value. Furthermore, the activity that caused the registry key update would have occurred within the same time interval.

The YD algorithm presented in this paper compares two snapshots of an MRU key and identifies the data values within the key that were updated during the time interval between the two snapshots. The algorithm provides an investigator with a timeline of changes from the restore points in the Windows registry by reversing the MRU key update process.

3. MRU Key Updates

The obvious way to determine the difference between two MRU key snapshots is to compare them value-by-value and identify the registry data values that were updated. However, although this method is valid for some registry keys, it is not applicable to all MRU keys. As we discuss below, the context in which specific registry data values are stored and the way the values are updated provide clues to the events that took place. In particular, it may be possible to deduce from the context of the data that a specific registry data value was updated even if the content of the data value did not change.

Data updates can be categorized based on two types of MRU keys. The first type of MRU key (Figure 2) stores data in several values that are named using numbers or letters and saves the order of the values in a special value called `MRUList` or `MRUListEx`. The leftmost letter of the `MRUList` or `MRUListEx` value corresponds to the most recently updated entry in the list. As shown in Figure 2, the value `c`, which represents the command `c:`, was the most recently typed command in the Run dialog window in the first snapshot. When a user executes the `regedit` command, the `MRUList` value data is updated to the new value sequence and the value `a` that contains `regedit\1` is not affected by this action.

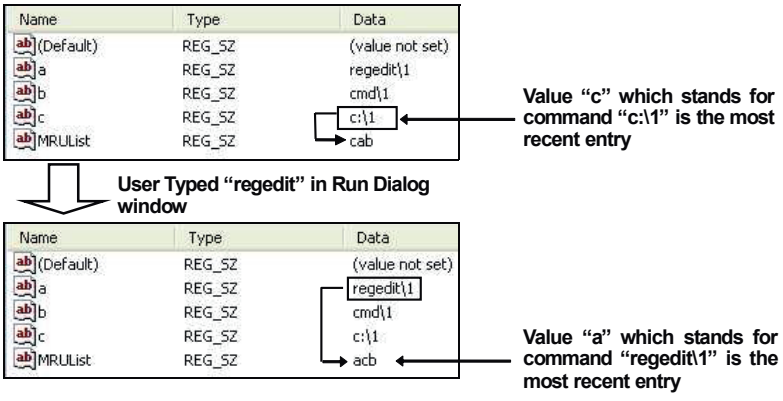

Figure 2. `RunMRU` example.

The second type of MRU key consists of a list of most recently used records. However, instead of using a special value to denote the sequence of values in the key, the name of each value contains a number that indicates its order in the list. If the order of values is changed, the system simply renames the values to maintain the order within the key.

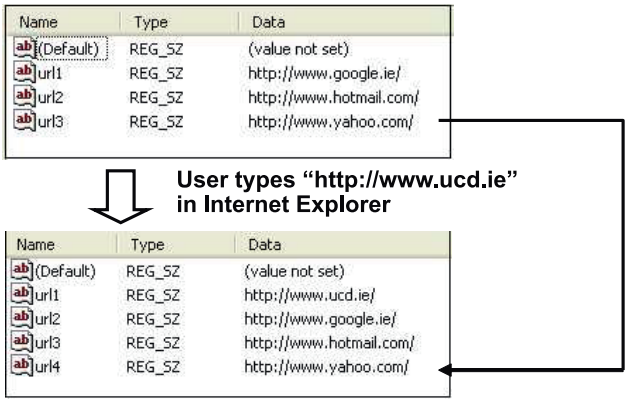

Figure 3. `TypedURLs` example.

Figure 3 presents an example of the second type of MRU key. It shows the consecutive states of the `TypedURLs` key before and after typing the URL `http://www.ucd.ie` in Internet Explorer. The value `url1` contains the most recently typed URL while `url3` contains the least recent entry in the first state. After the new URL `http://www.ucd.ie` is entered, it is added into the `TypedURLs` key as a new `url1` value. The initial data values in the `TypedURLs` key are renamed, increasing their index by one – the value `url1` becomes `url2`, `url2` becomes `url3`, and so on.

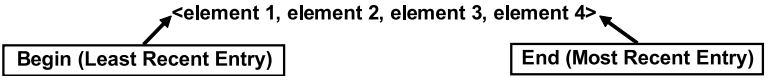

Figure 4. MRU list.

Note that in the example in Figure 2, although the MRU key was updated because `regedit` was typed in the Run dialog window, the corresponding value `a` was not modified. Only the `MRUList` value was updated because the value sequence was changed. On the other hand, all the data values in the second `TypedURLs` key example (Figure 3) were modified even though only one of them (`url1`) was added because of a user action. The `url3` value, for instance, was renamed to `url4` in the new state, but the user action did not change the content. Therefore, when comparing MRU keys in two consecutive snapshots, it cannot be assumed that the content of a changed value within an MRU key was typed, selected or produced by some other user action. The analyst must consider how the particular MRU key was updated in order to decide whether or not a particular value within the key reflects a user action that occurred between two states of the MRU key.

3.1 MRU List

Although different update processes are used for the two types of MRU keys, the keys are still updated in a similar manner. Consequently, a common model – the ordered MRU list – is used to represent the different MRU key types in the registry. As shown in Figure 4, the first element of the MRU list corresponds to the oldest entry of the MRU key; the last element of the list is the most recent entry of the MRU key. The MRU list thus abstracts the details of the MRU update process.

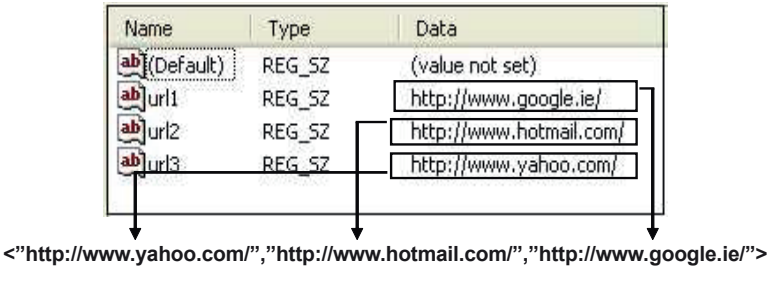

Figure 5. `TypedURLs` MRU list.

Figures 5 and 6 illustrate how the two types of MRU keys are converted into MRU lists.

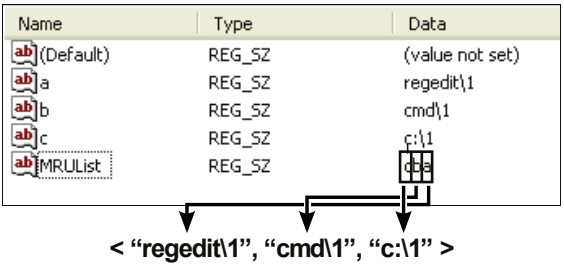

Figure 6. `RunMRU` MRU List.

Since most MRU keys have a limit on the number of elements, their MRU lists inherit this property. The maximum number of elements varies according to the key. For example, `TypedURLs` can have at most 25 elements while `OpenSaveMRU` can store at most ten elements. The least recent entry is removed when the number of elements in the list exceeds the maximum allowed.

3.2 MRU List Updating Algorithm

Updating the MRU list involves two steps [5, 6]. The first is common to every MRU list; it describes how a new element is added to the existing MRU list. The second involves additional list processing that is unique to each MRU key.

The updating process begins when the system receives a request to add a new element to the MRU list. The first step enumerates the existing MRU list and compares each entry with the new element. If no entries match, the new element is appended to the end of the current MRU list. If a match occurs, the old entry is removed and the new element is appended to the end of MRU list.

In the second step, the new list is processed based on additional constraints imposed by the particular MRU list. For most MRU lists, the constraint is the limit on the maximum number of elements (which results in the least recent element being removed when the maximum is exceeded). However, some MRU lists have an additional constraint, which causes elements to be removed from the list even when the maximum number of elements is not exceeded.

Figure 7 presents three examples of the MRU list updating process. The MRU list is assumed to have a maximum size of five. The first example adds a new element `5.txt` to the existing MRU list `<1.txt, 2.txt, 3.txt, 4.txt, 5.txt>`. The system detects that `5.txt` is already in the list; therefore, `5.txt` is removed upon which the new `5.txt` element is added to the list. The same process is followed when a new

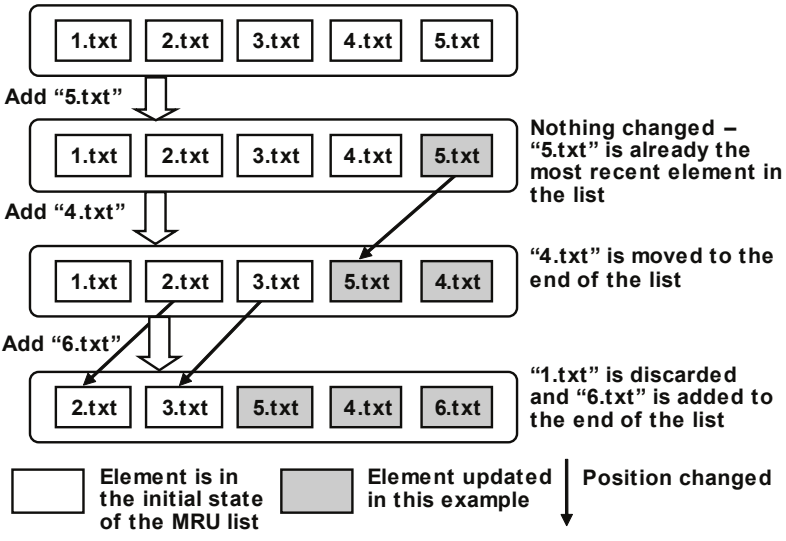

Figure 7. MRU list updating example.

`4.txt` item is added; the old `4.txt` is removed and the new `4.txt` is placed at the end of the list. In the third example, a new element `6.txt` is added to the current MRU list. Because `6.txt` is not already in the list, the system appends `6.txt` to the end of the list. However, the new list exceeds the maximum number of elements, causing the oldest item in the list (`1.txt`) to be removed.

3.3 MRU List Update Rules

Based on the list updating process, we specify two rules for identifying the newly updated elements between two states of the MRU list.

- **MRU List Rule 1**: Element *ele* is a newly updated element if there exists an element before *ele* in the current state of the MRU list that does not appear before *ele* in the previous state of the MRU list. To understand this rule, consider the second step in Figure 7 where `4.txt` is added to the list. In the state that immediately precedes this step, the element `4.txt` is preceded by the elements `1.txt`, `2.txt` and `3.txt` in the list. However, when `4.txt` is "added" to the MRU list, the element `4.txt` is moved to the end of the list, causing it to be preceded by `5.txt` in addition to `1.txt`, `2.txt` and `3.txt`. Note that the "happened before" relationship between `4.txt` and `5.txt` has changed. The rule relies

on detecting this change to identify the newly updated elements between two states of the MRU list.

- **MRU List Rule 2**: If element *ele* in the current list is known to be newly updated, then any elements after *ele* in the current list are also newly updated. This rule follows from the fact that the MRU list update process appends new elements to the end of the existing list.

4. YD Algorithm

The YD algorithm enumerates all the elements in the current state of the MRU list from the first (least recent) to the last (most recent) element looking for the first newly updated element according to MRU List Rule 1. It then returns all the elements starting with the found element to the end of the current state list. All the returned elements are identified as newly updated according to the MRU List Rule 2. Since an MRU list is a model of a particular MRU key, the newly updated elements identidied in the MRU list correspond to the newly updated data values in the MRU key.

Figure 8 presents the flowchart of the YD algorithm. List A denotes the previous state of the MRU list with n elements and List B is the current state of the MRU list with m elements. Note that A_x and B_y denote the x^{th} and y^{th} elements of A and B, respectively.

The YD algorithm first checks if A or B are empty. If B is empty, the algorithm returns NULL. On the other hand, if A is empty, the algorithm returns the elements of B. Next, the algorithm compares the first element B_1 of (current) List B with each element of (previous) List A. The loop ends either when an element A_x is equal to B_1 or the end of List A is encountered. If a match for B_1 is not found, the algorithm terminates and returns the entire List B. If B_1 is found in A, the algorithm continues to compare each consecutive element in B with each element in A following the matched A_x element. The process terminates when element B_y is not found in A (and the algorithm returns the elements from B_y to B_m) or the algorithm reaches the end of List B (and the algorithm returns NULL).

Figure 9 shows an example involving the first and last MRU lists from Figure 7. The algorithm requires three steps to identify the newly updated elements `4.txt` and `6.txt`. These results match the results of the updating process discussed in Section 3.2. Note, however, that one updated element `5.txt` is not identified. MRU List Rule 1 does not detect `5.txt` as newly updated because the addition of `5.txt` to the

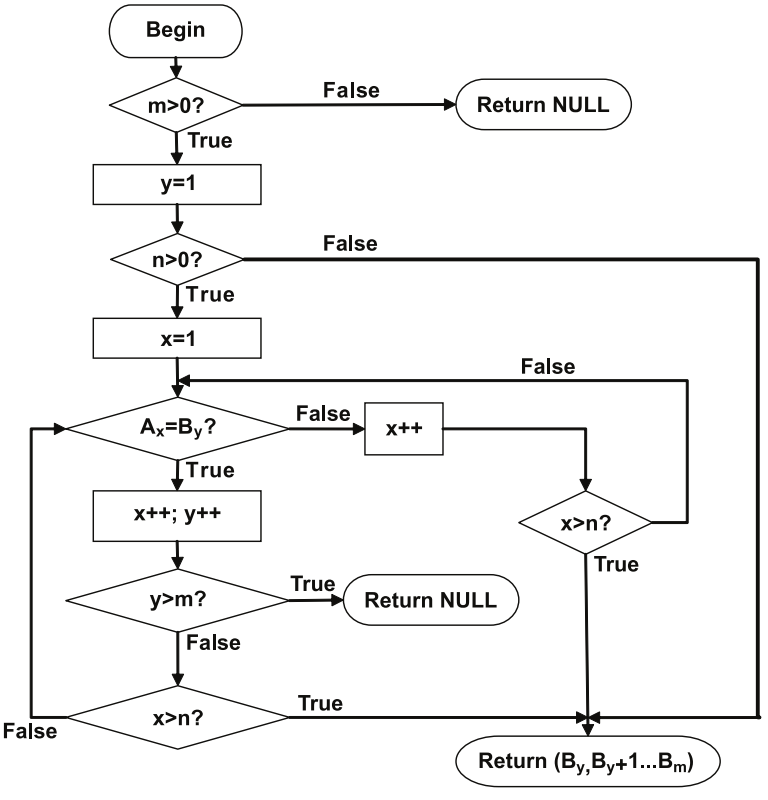

Figure 8. YD algorithm.

MRU list does not change the "happened before" order of the MRU list elements.

5. YD Algorithm Limitations

As discussed above, the YD algorithm may not detect all the elements updated between two MRU lists. This is because the algorithm relies on two (somewhat limited) rules that rely on the fact that elements change their positions in the list when they are updated. However, in some cases (as in the first step of the updating example in Figure 7), the state of the MRU key does not change in response to a user action.

To address this issue, we propose that the elements in the most recent MRU list be divided into a "Definitely Newly Updated Elements" set and a "Possibly Newly Updated Elements" set. "Definitely Newly Updated

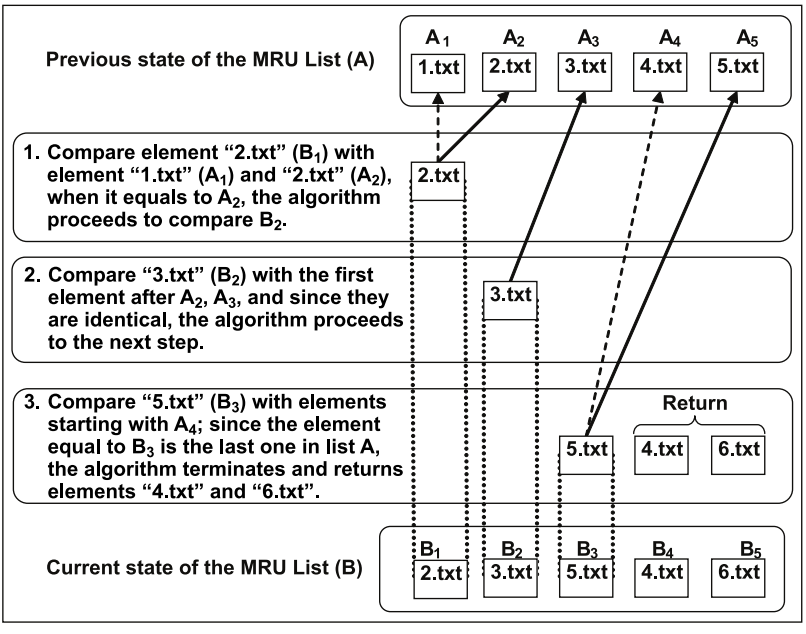

Figure 9. Using the YD algorithm to examine MRU lists.

Elements" are the elements identified by MRU List Rules 1 and 2. On the other hand, "Possibly Newly Updated Elements" are the elements that cannot be determined as newly updated when comparing two MRU lists. In the example discussed above, `2.txt`, `3.txt` and `5.txt` are possibly newly updated elements – it cannot be determined from MRU data alone if these elements were actually updated. However, if some external evidence exists to suggest that one of the three elements (say `3.txt`) was newly updated, then, according to MRU List Rule 2, any possibly newly updated elements that happened after it (i.e., `5.txt`) must also have been updated during the same time period.

6. Conclusions

The YD algorithm for analyzing MRU keys found in different snapshots of the Windows registry relies on two rules derived from a generic MRU key update model. Although the algorithm may not detect all the newly updated data values corresponding to MRU keys, it can identify "definitely newly updated" values. These data values are important because they enable investigators to determine the associated events that occurred between the two snapshots. The algorithm can be used

to examine MRU keys in consecutive registry snapshots extracted from Windows restore points as well as from consecutive backups of a system.

Acknowledgements

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

References

[1] H. Carvey, The Windows registry as a forensic resource, *Digital Investigation*, vol. 2(3), pp. 201–205, 2005.

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

[3] B. Harder, Microsoft Windows XP system restore, Microsoft Corporation, Redmond, Washington (technet.microsoft.com/en-us/library/ms997627.aspx), 2001.

[4] K. Harms, Forensic analysis of system restore points in Microsoft Windows XP, *Digital Investigation*, vol. 3(3), pp. 151–158, 2006.

[5] J. Holderness, MRU lists (Windows 95) (www.geocities.com/SiliconValley/4942/mrulist.html), 1998.

[6] E. Kohl and J. Schmied, `comctl32undoc.c`, Wine Cross Reference (source.winehq.org/source/dlls/comctl32/comctl32undoc.c), 2000.

[7] V. Mee, T. Tryfonas and I. Sutherland, The Windows registry as a forensic artifact: Illustrating evidence collection for Internet usage, *Digital Investigation*, vol. 3(3), pp. 166–173, 2006.

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

[9] B. Sheldon, Forensic analysis of Windows systems, in *Handbook of Computer Crime Investigation: Forensic Tools and Technology*, E. Casey (Ed.), Academic Press, London, United Kingdom, pp. 133–166, 2002.