



May 21st, 10:00 AM

## Correlating Orphaned Windows Registry Data Structures


Damir Kahvedžić

*Centre for Cyber Crime Investigation, University College Dublin, Ireland, damir.kahvedzic@ucd.ie*

Tahar Kechadi

*Centre for Cyber Crime Investigation, University College Dublin, Ireland, tahar.kechadi@ucd.ie*

Follow this and additional works at: <https://commons.erau.edu/adfsl>

 Part of the [Computer Engineering Commons](#), [Computer Law Commons](#), [Electrical and Computer Engineering Commons](#), [Forensic Science and Technology Commons](#), and the [Information Security Commons](#)

---

### Scholarly Commons Citation

Kahvedžić, Damir and Kechadi, Tahar, "Correlating Orphaned Windows Registry Data Structures" (2009). *Annual ADFSL Conference on Digital Forensics, Security and Law*. 10.  
<https://commons.erau.edu/adfsl/2009/thursday/10>

This Peer Reviewed Paper is brought to you for free and open access by the Conferences at Scholarly Commons. It has been accepted for inclusion in Annual ADFSL Conference on Digital Forensics, Security and Law by an authorized administrator of Scholarly Commons. For more information, please contact [commons@erau.edu](mailto:commons@erau.edu).

**EMBRY-RIDDLE**  
Aeronautical University™  
SCHOLARLY COMMONS

(c)ADFSL



# **CORRELATING ORPHANED WINDOWS REGISTRY DATA STRUCTURES**

**Damir Kahvedžić**

Centre for Cyber Crime Investigation,  
University College Dublin, Ireland,  
Tel: +353 1 716 2485  
Email: [damir.kahvedzic@ucd.ie](mailto:damir.kahvedzic@ucd.ie)

**Tahar Kechadi**

Centre for Cyber Crime Investigation,  
University College Dublin, Ireland,  
Tel: +353 1 716 2478  
Email: [tahar.kechadi@ucd.ie](mailto:tahar.kechadi@ucd.ie)

## **ABSTRACT**

Recently, it has been shown that deleted entries of the Microsoft Windows registry (keys) may still reside in the system files once the entries have been deleted from the active database. Investigating the complete keys in context may be extremely important from both a Forensic Investigation point of view and a legal point of view where a lack of context can bring doubt to an argument. In this paper we formalise the registry behaviour and show how a retrieved value may not maintain a relation to the part of the registry it belonged to and hence lose that context. We define registry orphans and elaborate on how they can be created inadvertently during software uninstallation and other system processes. We analyse the orphans and attempt to reconstruct them automatically. We adopt a data mining approach and introduce a set of attributes that can be applied by the forensic investigator to match values to their parents. The heuristics are encoded in a Decision Tree that can discriminate between keys and select those which most likely owned a particular orphan value.

**Keywords:** Windows Registry, Data Structures, Retrieval, Orphans, Correlation

## **1.INTRODUCTION**

The Windows Registry is a hierarchical database that stores information about the system software, hardware, its users and their preferences. Investigators tend to concentrate on the active data already present in the hives (Carvey, H. (2005), Farmer, D.J. and Burlington V. (2009), Registry Hives. (2008), Kahvedžić, D. and Kechadi, T. (2008), Kahvedžić, D. and Kechadi, T. (2008)ii, Wong, L. W. (2009)). After it has been deleted however, this information may still reside in the system files of the registry (B. D. (2009), Morgan, T. D. (2008)). The space of the deleted keys is marked as free and can be reallocated for new keys. If the space is not yet overwritten, the deleted keys can be retrieved. The keys are found by simply parsing deallocated space and following any links to their values and data. The links between the data structures therefore are used to correlate one data structure to another.

However, the structures found in a deleted state may be in a corrupt state and may not preserve the full information that it contains while it was active (Kahvedžić, D. and Kechadi, T. (2009)). The links in particular may not be preserved. Some data structures cannot be reattached to the registry tree and cannot be viewed in context. Registry key values are particularly important, since they store the actual key data and do not store links to their parents. We call all data structures that cannot be reattached to the registry hierarchy orphans.

Software uninstallers and registry cleaners (JavaCoolSoftware (2009)) usually delete many values and keys from the registry. Many links between the deleted values and the points in the registry that they were deleted from can be easily retrieved. However some links may be lost and require specific

processing to retrieve them. In this paper we will illustrate how many orphans are created during an uninstallation. We will then formalise the problem of reassembling these keys using a finite state machine model and describe a methodology for reattaching them by exploiting a number of observations on how the registry keys are managed. We consider these observations as attributes and take a data mining approach to reattach the orphaned values to the most likely owner keys.

Section 2 defines the formal model. Section 3 discusses the data mining approach undertaken to solve this problem and the experimental setup that is used to test the technique and validate the results. Section 4 discusses the various patterns identified in the way that the registry stores its data structures. Section 5 combines the various attributes to construct the Decision Tree classifier. Section 6 and 7 summarise the paper and describe future work.

## 2. FORMAL MODEL

We concentrate on the three major registry data structures; the key, the value and the security key (B. D. (2009)). In this section, we formalise the operation of the registry with respect to these structures. We define and describe how orphans are created and formalise the problem of connecting them to the most likely parent keys.

### 2.1 Concepts

Let  $\mathbb{R}$ ,  $\mathbb{V}$  and  $\mathbb{S}$  be the set of all keys, values and security keys respectively in the registry. Let  $\mathbb{U}$  be the union of all of them. Any registry entry (key)  $r$  can therefore be defined as a triplet consisting of a set of subkeys, values and a single security key:

$$r_i(k_i, v_i, s_i), \text{ where } k_i \in \mathbb{R}, v_i \in \mathbb{V}, s_i \in \mathbb{S}$$

$r(0, 0, 0)$  is a key that does not have any values, subkeys or security keys associated to it. The maximum number of structures a key can have is  $|\mathbb{R}| - 1$  subkeys,  $|\mathbb{V}|$  values and a single security key, where  $|\mathbb{R}|$  and  $|\mathbb{V}|$  denote the cardinality of  $\mathbb{R}$  and  $\mathbb{V}$ , respectively.

In addition, the keys can be in one of the two states; Active ( $\mathcal{A}$ ) and Deleted ( $\mathcal{D}$ ). All keys must have been active at some point in the registry. At certain time  $t$ , we can have some active keys and deleted keys. So we can write:

$$\begin{aligned} \mathbb{R} &= \{\mathbb{R}_a \in \mathcal{A} \cup \mathbb{R}_d \in \mathcal{D}\} \\ \mathbb{V} &= \{\mathbb{V}_a \in \mathcal{A} \cup \mathbb{V}_d \in \mathcal{D}\} \\ \mathbb{S} &= \{\mathbb{S}_a \in \mathcal{A} \cup \mathbb{S}_d \in \mathcal{D}\} \end{aligned}$$

Therefore the set of all deleted structures is  $\mathbb{U}_d = \mathbb{R}_d \cup \mathbb{V}_d \cup \mathbb{S}_d$ . In normal operation of the registry, the user cannot access the deleted structures or retrieve deleted data. All of the keys available to the user are in the Active state and are denoted by  $\mathbb{U}_a$ .

We can extend the definition of a key  $r$  by adding a number of constraints based on its state:

$$\begin{aligned} r_d &\equiv r(k_d, v_d) \text{ or} \\ r_d &= r(k_d, v_d, s_d) \end{aligned}$$

Therefore, all deleted keys must reference deleted structures. A deleted key is not restricted to reference a deleted security key. Security keys can be used by multiple registry entries, ensuring a similar permissions policy (B. D. (2009)).

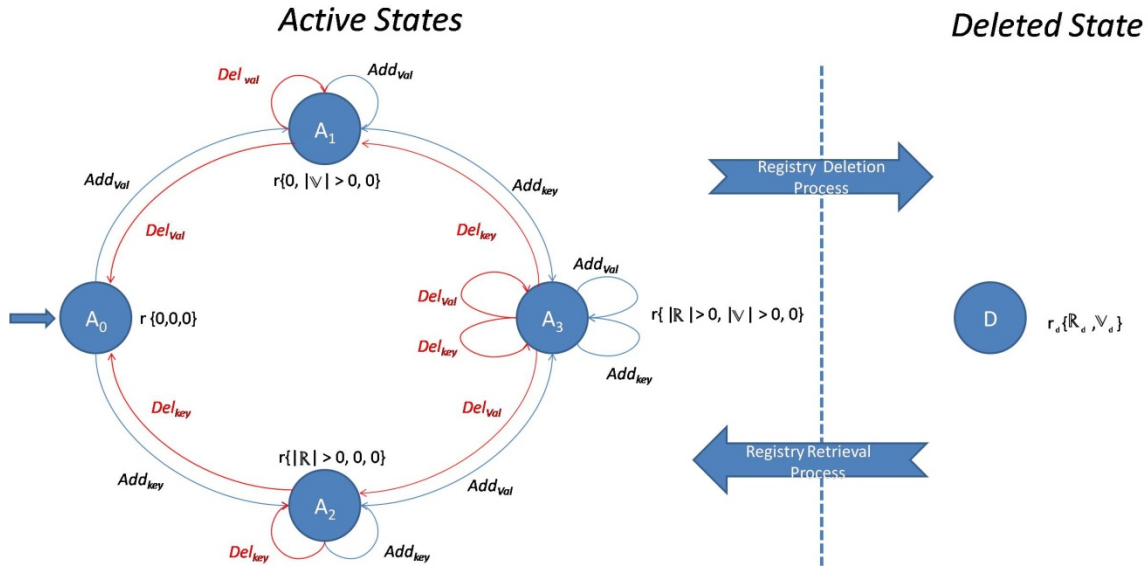


Figure 6: Deletion and Retrieval of Keys

### Finite State Machine

A finite state machine model is adopted to illustrate the different actions that are possible on the registry key during its lifetime. Formally, a finite state model (FSM) for this problem can be defined as a tuple of three elements  $T = (Q, I, \varphi)$  where:

- $Q$ : A finite set of all possible states.
- $I$ : A finite set of all possible events.
- $\varphi : I \times Q \rightarrow Q$ : A *transition* function that maps input symbols of the current state to the next state.

The resulting FSM can be illustrated with a transition graph. Figure 1 illustrates the states, transitions and events between them. A single key can contain many peripheral data structures to store values and keys more efficiently. The starting state of the key is an empty state where it contains no subkeys or values. During its lifetime a key can have multiple values or keys and conversely can have these subkeys or values removed. The events are termed **Add\_val**, **Add\_key**, **Del\_val**, **Del\_key** and have important distinctions.

The transitions **Add\_val** and **Add\_key** associate a new value or new subkey to a key. The space for the structures is allocated and a link from the key to the structure is created. In the case of a new subkey, a link from the subkey to its parent is also created.

The transition, **Del\_key**, converts  $r_a \in R$  to  $r_d$ . All of the structures referenced by the key are deleted,  $r(k_a, v_a, s_a)$  to  $r(k_d, v_d)$ . The links between them are preserved and the associations can be retrieved. The key data structure itself contains a link from the structure to the parent key in the hierarchy. These are not modified when a key is deleted and can be retrieved and used to reattach the deleted keys to the tree hierarchy.

The transition, **Del\_val**, deletes a single value and converts it from  $v_a \in V$  to  $v_d$ . A value data structure does not retain a link to its parent key. Therefore the value, once deleted, is disconnected from the tree hierarchy. They cannot be easily reattached to the registry hierarchy.

The series of transitions from one state to another are termed *computations*. A finite computation is a sequence of steps  $c_j = (l_j, q_j)$  where each step is made up of an event,  $l_j \in I$ , and a state  $q_j \in Q$ .

The set of computations is denoted by  $\mathcal{C}_T$ . The computation,  $\mathcal{C}_T$ , is defined by a finite number of steps,  $\mathcal{C}_T(t)$ , ranging from key creation to key deletion.

Orphans are defined as any data structure that has lost its link to the registry tree and are represented in the set  $\mathcal{O}$ , where  $\mathcal{O} \subset \mathcal{D}$ . Orphans are denoted  $\mathcal{U}_o$  and represent both orphan keys  $\mathbb{R}_o \in \mathcal{U}_o$  and values  $\mathbb{V}_o \in \mathcal{U}_o$ .



Figure 7: Sample Computation

### 2.3 Problem Statement

Figure 2 illustrates one possible computation in the model  $T$  for a registry key  $r$ . There are eight states starting with the key creation and ending with its deletion. During the process three values were added (at  $t_1, t_2$  and  $t_4$ ) and two were deleted (at  $t_3$  and  $t_5$ ). Current registry recovery programs can retrieve the key  $r_r$  and its values, but cannot associate the deleted values to the key. These orphans seem unrelated to the key. Therefore the number of data structures associated with a deleted key depends on its state when it was deleted.

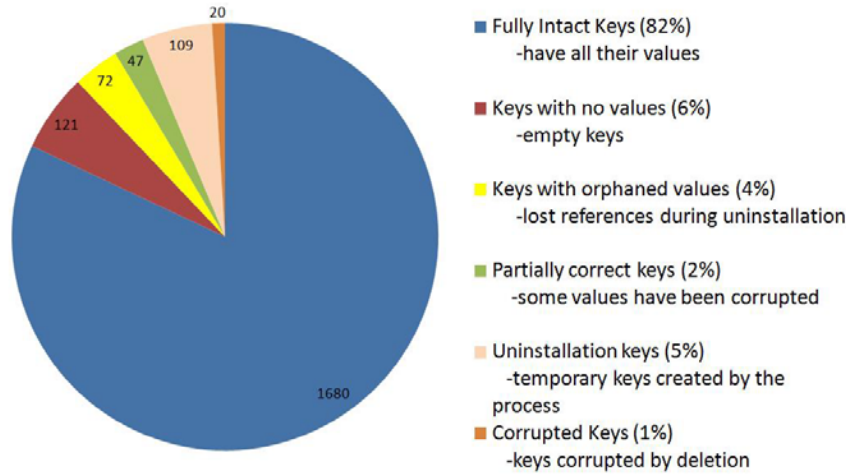


Figure 8: Different states of the retrieved keys (2049 total)

Let  $P(v, r)$ , where  $v \in \mathcal{O}$  and  $r \in \mathbb{R}$ , be the probability of an orphan value  $v$  belonging to the registry key  $r$ . Our aim is to maximise  $P$  with respect to all orphans  $v \in \mathcal{O}$ .

The further in the past a value is deleted the less likely it can be retrieved (Kahvedžić, D. and Kechadi, T. (2009)). This is due to the reallocation of deallocated space by the registry manager (Russinovich, M. (2009)). However a large number of orphans can be created during the uninstallation of software or if a large number of values were recently deleted.

As an illustration, registry keys belonging to the media platform Real Player (RealNetworks. (2009)) were retrieved after the software's uninstaller was performed. To minimise possible orphan creation as a normal operation of the software, Real Player was uninstalled directly after it was installed. Even in this best case scenario, where the time range  $t$  of the computation is kept deliberately very low, 20%

of the keys were modified and 15% of the values-key relations were lost in the process. The values could still be retrieved but the link to the keys that they belonged to was removed.

Figure 3 illustrates the different states of the retrieved keys for the Real Player uninstallation process. Of particular interest are the 72 keys that have had their values deleted prior to the key itself. The values of these keys can be found in the deallocated space but could not be associated to the keys. The rest of this paper will detail our methodology in extracting these associations and linking the orphans to the most likely key that may have contained them it.

### 3. APPROACH AND VALIDATION TEST SETUP

We use a data mining approach to classify the orphan values to their most likely keys. We are able to process the values and other structures that are retrieved from the unallocated spaces with high accuracy. The stages of the mining process are described in Han, J. and Kamber, M. (2006). From this point of view, the registry retrieval processes is the first step; Data Acquisition. The next sections describe subsequent stages, namely Attribute Selection and Key Associations.

The attributes are evaluated and validated using sample test systems. A number of different registries were extracted from a variety of systems to test out the attributes. All registries were from Windows XP SP2 Operating System computers. They ranged from registries of three months usage to registries taken from systems that were in use for a number of years. The Software and the 'ntuser.dat' hives are the most active hives and contain the most useful information for a forensic investigation. As such we will concentrate on them in our validation tests.

The summary of the various registries and the number of keys and values that they contain are summarized in Table 1. Security keys, although important for forensics, account for less than 0.05% of the total data structures. As such we will concentrate on the key and value structures only for the rest of the paper.

<b>Details of Hives Used</b>		
<b>Hives</b>	<b># of Keys</b>	<b># of Values</b>
<b>Software Hives</b>	285378	426929
<b>System Hives</b>	33226	95832
<b>Security Hives</b>	625	622
<b>SAM Hives</b>	234	260
<b>ntuser.dat Hives (38 users)</b>	53723	200130

Table 3: Summary of the Hives and their Combined Keys and Values

### 4. ATTRIBUTE SELECTION

In this section we describe some of the features of the way keys and values are stored in the registry. These features will define attributes describing the value and key data structures. We use classifiers to predict which orphan values belong to which key. The attributes are based on a number of observations in the functioning of the registry. The observations are formalised as attributes and validated through experimentation in the following sections.

#### 4.1 Value-Key Position Relation

The position of the value data structure with respect to its key depends on the allocation strategy of the registry management system; the Configuration Manager (CM). Microsoft Windows documentation describes the allocation strategy of the CM when new keys are created (Russovich. M. (2009)). The CM would first find the parent key data structure and then search for a block that is big enough to store the new key. If there is no block that satisfies the request then a new space is allocated at the end

of the hive file. It has been found that the hive files contain much more smaller free fragments than large ones (Kahvedžić, D. and Kechadi, T. (2009)) and that some key values are created as soon as the key is created (Morgan, T. D. (2008)). Therefore it is likely that when a new key is created the CM would not find a big enough fragment and allocate new space. At least some values therefore are found close to, if not directly after, its key data structure.

Formally, we consider a registry hive as a list of data structures with the physical position of the data structure within the hive denoted as  $Pos(u)$ , where  $u \in \mathbb{U}$ . We also define a binary operator,  $Pos(u) \rightarrow Pos(u_1)$ , which states that the data structure  $Pos(u_1)$  is stored later in the hive file than  $Pos(u)$ . The operator  $\leftarrow$  is the opposite. This attribute states that  $P(r, v) \succ P(r, v_1)$  where  $Pos(r) \rightarrow Pos(v) \rightarrow Pos(v_1)$ . Namely, the closer in the hive file a value is found after a key, the more likely that the value belongs to that key. In this case the value  $v$  is more likely to belong to  $r$  than  $v_1$ .

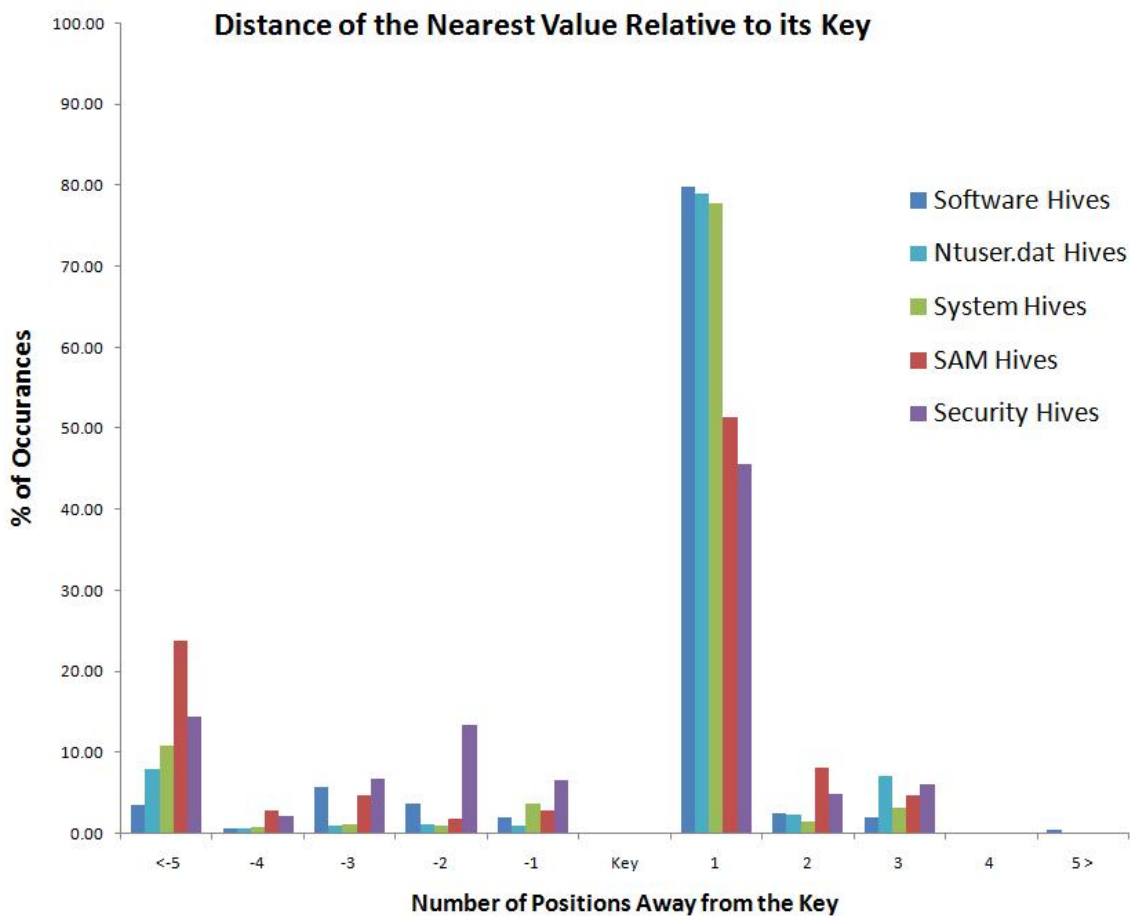


Figure 9: Relative Positions of Values to the Keys they Belong to

Figure 4 shows the respective distances of a key's closest value data structure relative to the position of the key in the hive in our test corpus. It is clear that in the majority of cases, the first value encountered after the key in the hive belongs to that key. Hives with a high amount of activity, such as the user hive ('ntuser.dat') or the 'SOFTWARE' hive, tend to have more keys that follow the above heuristic. While hives with a lower activity, such as 'SAM' and 'SECURITY', tend not to follow the

heuristic as much as the active ones.

**Value-Value Position Relation**

Similar to the key-value position relation, if one key has a large number of values, the CM, if possible, would allocate space for all of them in the same place in the hive. MRU lists for example often have a large number of values with their data structures found one after another in the hive. Similarly, if an application creates a number of values at installation, the space is allocated as contiguously as possible and as such the values can be found grouped together. In conjunction with the first attribute, this attribute states that if the first key is found then the orphans found in its group are likely to belong to that key as well. Figure 5 illustrates this case.

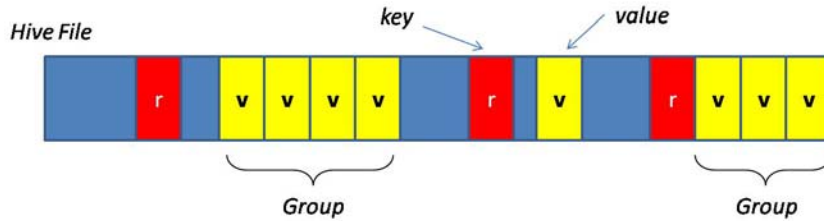


Figure 10: Group Allocation Strategy

**Relation of Subkeys and Values**

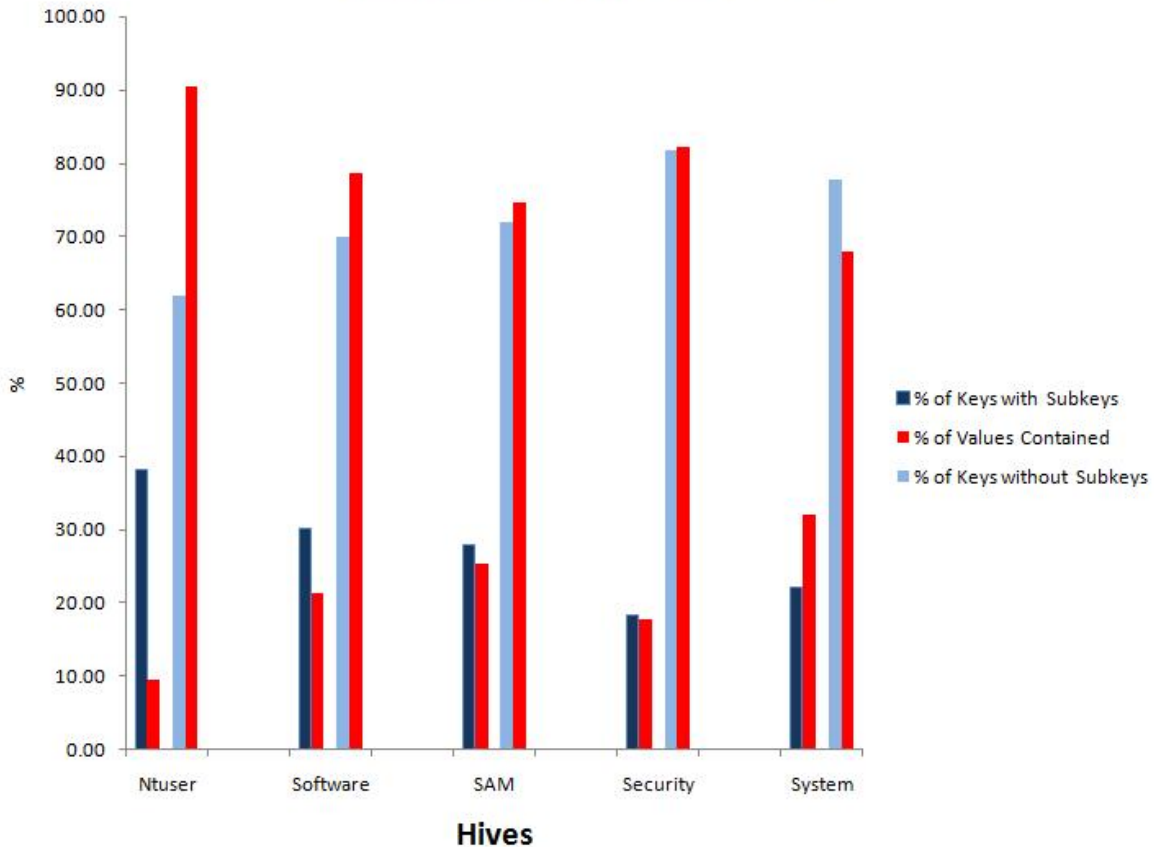


Figure 11: Internal/Leaf Key to Value Relations

Formally, we define a group  $g \in G$  to be a set of orphans  $g = \{v_0, \dots, v_n\}$  where  $v_i \in \mathbb{O}$  and there does not exist a situation where  $Pos(v_0) \leftarrow Pos(r) \leftarrow Pos(v_n)$ . Expansion of the group is also



stopped if a value with an identical name is found.

Similarly,  $P(r, g)$  is the probability of all orphans in  $g$  belonging to the key  $r$ . Through experimentation, we have found that a group with two members is not likely to belong to the same key than any group with a higher number.

### 4.3 Subkeys Number Relation

In the tree like organisation of the registry, we use the term *leaf* to define those keys that do not have any subkeys, ie.  $r(|R| = (0, v, s))$ . These keys, found at the bottom of the registry hierarchy tend to store more values than those keys found higher up in the registry. There is not an enforced policy on how programs using the registry are meant to store its registry values but Microsoft recommends software vendors to follow the “*HKCU\Software\Vendor\Program\Version*” organisation for storing keys in the registry (Honeycutt, J. (2002)). This results in the leaf keys storing meaningful information such as values while the internal keys are used for organisational purposes.

In the formal model, the pattern can be described as,  $P(r, v) > P(r_1, v)$  where  $r(|k|) < r_1(|k|)$ . Namely, the probability of an orphan value belonging to key  $r$  is increased if the key contains a low number of subkeys. For simplicity, in our classifier we let  $r(|k|) = 0$  and assume that if a key has any subkeys then they do not have any values.

Figure 6 illustrates the relationship between the number of values and the position of keys in the registry hierarchy of our test corpus. In the majority of cases, a key that does not have values would have subkeys. The user hives in particular contain the majority of values in the leaf keys rather than the internal keys.

### 4.4 Value List Presence Relation

Once a single value is identified, the value list of that key can also be found. A value list is a peripheral data structure and stores references to all the values of that key. The structure does not have an identifier and cannot be easily recognised when retrieved. However all the unidentified structures can be searched to see if any contain the address for the orphan value. Any unidentified structure that contains this address is likely to be a value list. The value list can be as small as 8 bytes. Therefore, the list, in contrast to values, can be found far away from its key.

Amount of Value List Slack						
Hive	# of Keys	With Slack	Without Slack	Slack (bytes)	# of Pointers	Pointers to Hidden Keys
Software	43,212	7,818	30,529	33,060	304	7
System	9,140	5,572	2,617	24,156	255	0
Security	208	1	206	8	1	1
SAM	75	8	66	32	0	0
ntuser.dat	533	148	179	592	7	0

Table 4: Amount of Value List slack

In addition, the Value List continuously adds and deletes offsets of created and deleted values (Morgan, T. D. (2008)). Under certain conditions, offsets to deleted values are still present at the end of the list data structure in an area called Value List slack. Table 2 shows the amount of slack in a sample of hives in the test corpus. Note that some keys do not contain any value lists at all. As seen in the table, many keys do have value list slack and do contain many offsets that appear to be valid. Most

of the offsets, however, are copies of active keys and do not point to hidden keys. Although the presence of any useful pointers is not common, the slack should not be disregarded in the process for correlating orphans.

#### 4.5 Value size

Although the values in the registry have a variable structure, some valuable forensic keys have a predictable size that can be used to identify them if they are orphans. In particular, features of the Windows Operating System registry keys are known and can be used to identify them if they are found as orphans (Rubenking, N. J. (2009)). The “HKLM\Software\Microsoft\Windows\CurrentVersion\App Management\ARPCache\AppName” key, for example, stores information of each application displayed by the control panel of Windows. Each application has a ‘SlowInfoCache’ value which includes important information on the application’s usage frequency, install size and last used time. The values have a constant size of 552 bytes, the specific structure of which is known (Rubenking, N. J. (2009)). As well as the aforementioned data, the value also contains the name of the program it references and can be used to relate the value to the relevant key.

#### 4.6 Specific Value Content

Similar to using the value size as an attribute, some important keys contain values that store a predictable type of content. The most important of these keys are the MRU (Most Recently Used) lists which store records of most recently opened files. MRUs are widespread in the registry and can provide vital clues to user activity (Farmer, D.J. and Burlington V. (2009)). Windows does not enforce a standard way of storing values in all MRU lists. Therefore it is possible to extract and identify the relation between an orphan MRU entry and its parent key. The identifying features of these values include the information type (binary or ASCII), the naming conventions, the type of data and the recording policy. A few important MRUs and the identifying features of their values are documented in Table 3.

Identifying Features of MRUs				
1:	\Windows\CurrentVersion\Explorer\ComDlg32\OpenSaveMRU	Type: Last Saved File	Name: Letters (a-j)	Scheme: ASCII Data: Filepath
2:	\Windows\CurrentVersion\Explorer\ComDlg32\OpenSaveMRU\‘extension’	Type: Last Saved File by Extension	Name: Letters (a-j)	Scheme: ASCII Data: Filepath with specific extension
3:	\Windows\CurrentVersion\Explorer\ComDlg32\LastVisitedMRU	Type: Last Opened File	Name: Letters (a-y)	Scheme: Binary Data: Filepath
4:	\Windows\CurrentVersion\Applets\Paint\Recent	Type: MSPaint Recent Files	Name: ‘File(n)’ where n>0	Scheme: ASCII Data: Filepath of ‘Image’ file
5:	\MediaPlayer\Player\RecentFileList	Type: Media Player Recent Files List	Name: ‘File(n)’ where n>0	Scheme: ASCII Data: Filepath of ‘Multimedia’ file
6:	\SearchAssistant\ACMru\5603	Type: XP Searches	Name: 3 Digit Number (001)	Scheme: ASCII Data: User search query strings
7:	\Windows\CurrentVersion\Explorer\RecentDocs	XP Recently Opened Docs	Name: 1 Digit Number (0-9)	Scheme: Binary Data: Filepath of opened docs
8:	HKCU\Software\Windows\Microsoft\Office\12.0\Word\File MRU	Word Recently Opened Docs	Name: ‘Item (n)’ where n>0	Scheme: ASCII Data: ‘[F0000000][17characters]Filepath’
9:	HKCU\Software\Windows\Microsoft\Office\11.0\PowerPoint\Recent File List	Powerpoint Recently Opened Docs	Name: ‘File(n)’ where n>0	Scheme: ASCII Data: Filepath
10:	HKCU\Software\Windows\Microsoft\Office\12.0\PowerPoint\File MRU	Powerpoint Recently Opened Docs	Name: ‘Item (n)’ where n>0	Scheme: ASCII Data: ‘[F0000000][17characters]Filepath’

Table 5: Identifying features of MRU Values

### 5. DATA MINING STAGE

In this section we will use the attributes described above to create a decision tree classifier for deleted values. We use Decision Trees as an initial mining technique. Other algorithms such as association rules, clustering algorithms and neural networks will be applied in future work. The situation

described in Section 2.3 revealed that 72 deleted keys had all their values orphaned prior to being deleted. In this section we will use a decision tree classifier, illustrated in Figure 7, to re-attach the orphans to their keys.

The decision tree classifier described here represents our initial work in associating orphan values to keys. The classifier is based on structural attributes described above and exploits the manner in which the Configuration Manager allocates new structures. It does not take into account any of the content that the various data structures may hold and does not attempt to do any content analysis or similarity matching as described in the latter parts of Section 4. A clustering approach would be more suited for that stage of mining and is left for future work.

### **5.1 Results**

As previously stated, 72 keys in the Real Player uninstallation had their values orphaned, totalling 146 values. 143 (98%) of these values were retrieved from deallocated space while the remaining 3 values were overwritten and could not be retrieved. The aim of the decision tree is to associate the 143 values back to their keys. The total number of orphans retrieved was 360. The decision tree should avoid these orphans and minimise false positives.

The classifier first attempts to find the first value data structure after the key's data structure in the hive. The classifier associated 68 orphans to 68 keys in this way. However, it couldn't associate values to 4 keys. Of the 68 values, 63 of them were accurate and did indeed belong to the key prior to the uninstallation. 5 of the orphans were incorrectly associated. The precision rate is therefore 93%.

The classifier extends the association criteria by finding any groups that the first orphan belongs to. The groups are filtered out if they only contain two elements as we have found that  $P(r, g)$  is maximised in larger groups. The classifier correctly associated 77 values to their corresponding keys and misclassified 11. The precision rate therefore is reduced to 86%.

Before the orphans are associated to the keys, the classifier attempts to find the value list storing the offset to the value. If the value list is found, all of the values that it references will be associated to the key. As described in Section 4.4, the value list may not be found even if it did at one point list the orphan. Of the 68 values found, 32 of them found value lists, 36 of them did not. 8 of the found value lists were incorrect. The precision rate is therefore 78% for value list searches. The found value lists classified 2 new orphans correctly and 2 incorrectly. As described above, the vast majority of value lists 97% only contained a single offset.

In total 79 values were correctly associated by using the closest value and value list search heuristics, 13 were misclassified. Therefore, 55% of the orphans were associated correctly with a reliability of 84%.

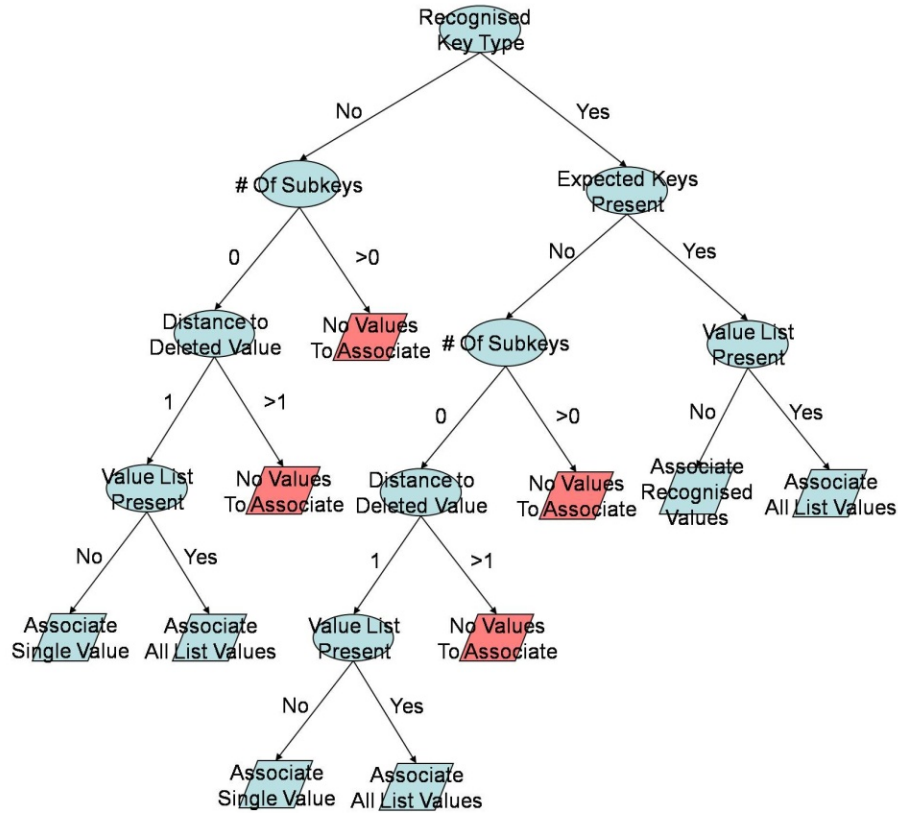


Figure 12: Decision Tree Classifier

## 6. SUMMARY AND CONCLUSION

In this paper we have illustrated how values, once deleted from the registry, can be orphaned from their parent keys and cannot be easily re-associated to them. We have explored the effects of the Real Player uninstallation process on the registry and demonstrated how the uninstallation created orphans. The resulting orphans can make it difficult for the investigator to know the context of the retrieved registry entry and lay doubt in legal arguments. We have laid the formal theory and the ground work for the application of data mining techniques in retrieving and correlating these deleted data structures to the points in the registry that they were found.

The formal theory is presented in the form of Finite State Machines. Using this model we have demonstrated that the amount of information retrieved from a deleted key depends on the state when it was deleted. However, values that once belonged to the key but that were deleted prior to the key can also be retrieved. The link between the key and its former value is not preserved and the association cannot be made trivially. The unlinked data structures are termed orphans and require special processing to reattach them to their parent keys.

Presented in this paper are a preliminary set of patterns in the organisation of the registry data structures. The patterns are encoded as attributes that can be used by classifiers to relate orphaned values to the most likely owner keys. Tests were carried out on sets of real world registries to analyse the validity of the attributes presented. A decision tree classifier was described and used as a preliminary Data Mining technique to associate orphan values to the most likely parent keys.

## 7. FUTURE WORK

The features presented here constitute the preliminary attributes discovered while manually reconstructing the registry keys. A more formal and in-depth feature extraction methodology will be developed to extract ever more accurate features to discriminate between keys. Future work will entail

expanding on the data mining stages to further refine the accuracy of the classifier. Clustering of the values, based on the value's name and content, will be used to further associate the orphans with more accuracy. Data pre-processing stages will be expanded to clean and filter out any outliers and data structures that have been corrupted, for example if they are partially overwritten.

The features presented in the classifier in Figure 7 are binary. A more accurate classifier would discriminate between continuous attributes. For example, in the subkey-number relation, in Section 4.3, if a key is found to contain subkeys it is assumed not to have any values. This feature does not take into account the position of the key in the global tree hierarchy. A key may be deep in the tree hierarchy and may contain both subkeys and values. The position of the key may dictate the probability of the key holding values even if it has subkeys.

Content features are not used in the above decision classifier and is also left for future work. The content of the deleted keys can be used to find similar active keys and predict which values are missing from the deleted key based on the properties of the similar active one. This type of mining process is more suited to clustering. Outlier analysis in particular can be used to determine not which values belong to keys, but which values do not belong to them. The removal of these outliers can eliminate clutter to the investigator.

Following hidden pointers to deleted data structure is prone to error. It is possible that the structure is a newer data structure belonging to some other key which was subsequently deleted. Although unlikely it cannot be disregarded, extra processing and checks are required to disprove this possibility.

The Windows Vista operating system subtly changes the manner in which registry keys are stored (SWGDE (2008), Hargreaves, C. et al. (2008)). The MRU keys in particular are store in binary more frequently and stored more information than the corresponding keys in the XP operating system. Future work on the classifier will include a method for identifying these MRU keys in addition to their XP counterparts.

## REFERENCES

Carvey, H. (2005) "The Windows Registry as a forensic resource", *Digital Investigation*, Vol 2 (Issue 3) p201–205, 2005.

B. D. (2009) 'Registry File Format', <http://home.eunet.no/pnordahl/ntpsswd/WinReg.txt>. visited Feb 2009.

Farmer, D.J. and Burlington V. (2009) 'A Forensic Analysis of the Windows Registry', [http://eptuners.com/forensics/Registry\\_Forensics.pdf](http://eptuners.com/forensics/Registry_Forensics.pdf), visited Jan 2009.

Han, J. and Kamber, M. (2006) 'Data Mining: Concepts and Techniques', Morgan Kaufmann, 2nd edition, 2006.

Hargreaves, C. et al. (2008), "Windows Vista and Digital Investigations", *Digital Investigation*, Vol 5 (Issue 1), p34 – 48, 2008.

Honeycutt, J. (2002) 'Microsoft Windows XP Registry Guide', Microsoft Press, 2002.

JavaCoolSoftware. (2009) 'MRU-Blaster', <http://www.javacoolsoftware.com/mrudownload.html>, visited Jan 2009.

Kahvedžić, D. and Kechadi, T. (2008). 'Extraction of User Activity through Comparison of Windows

Restore Points’, SECAU08, 6th Australian Digital Forensics Conference, Dec 2008, Perth, Australia.

Kahvedžić, D. and Kechadi, T. (2008)ii. “Extraction and Catagorisation of User Activity from Windows Restore Points”, JDFSL: Journal of Digital Forensics, Security and Law, Vol4 (Issue4) (to be published).

Kahvedžić, D. and Kechadi, T. (2009). ‘On the Persistence of Deleted Windows Registry Data Structures’, 24th Annual ACM Symposium on Applied Computing, March 2009, Hawaii, USA.

Morgan, T. D. (2008) ‘Recovering Deleted Data From the Windows Registry’, Digital Forensic Research Workshop, Aug 2008, Baltimore, USA.

RealNetworks. (2009). ‘Real Player 11 Basic’, <http://europe.real.com/player/win/>. visited Jan 2009.

Registry Hives. (2008) ‘Forensicmatter.com: Registry Hives’, [http://www.forensicmatter.com/registry\\_hives.php](http://www.forensicmatter.com/registry_hives.php), visited Feb 2009.

Rubenking. N. J. (2009) ‘Unclean 2’, <http://www.pcmag.com/article2/0,1759,1159867,00.asp>. visited Jan 2009.

Russinovich. M. (2009) ‘Inside the registry’, <http://technet.microsoft.com/en-gb/library/cc750583.aspx>. visited Jan 2009.

SWGDE (2009), ‘Technical Notes on Microsoft Vista (submitted for review)’, Scientific Working Group on Digital Evidence, <http://www.swgde.org/documents.html>, visited Feb 2009.

Wong, L. W. (2009) ‘Forensic Analysis of the Windows Registry’, <http://www.forensicfocus.com/forensic-analysis-windows-registry>, visited Jan 2009.

#### **ACKNOWLEDGEMENTS**

The authors would like to acknowledge “Higher Education Authority” (HEA) and “Irish Research Council for Science, Engineering and Technology” for providing funding that made this research possible.

