Carving the Windows registry files based on the internal structure

Zhenhua Tang[1] Hong Ding[2] Ming Xu[3] Jian Xu[4]
tzhtom@126.com, dinghong@hdu.edu.cn, mxu@hdu.edu.cn jian.xu@hdu.edu.cn
School of Computer, HangZhou DianZi University, HangZhou, ZheJiang, 310018, China

Abstract—The Windows registry stores a lot of system information which can be used as forensic evidence. Numerous researchers have worked to interpret the information stored in the registry, but no definitive resource is yet available which describes how to carve the registry files from the raw disk. In this paper, a carving algorithm for the registry files based on the registry file internal structure is described. The carving method can recover the Windows registry files, and the file directory metadata is not available, even if the registry files are fragmented between two HBIN blocks. The experiments demonstrate that our method is effective for carving the Windows registry files with more accuracy than other file carving techniques.

Keyword-computer forensic; registry; data carving

I. INTRODUCTION

One of the most popular and most often attacked operating system is the Microsoft Windows, so much of the computer forensic research focus on this famous system. The Windows registry is a database that stores configuration settings for System, users, applications, hardware and many other system parameters. Due to a lot of information contained in Windows registry, the registry can be an excellent source for potential evidential data in aiding forensic examiners on other aspects of forensic analysis. Although the Windows registry appears as a single hierarchy in registry editor such as regedit.exe, it is actually made up of a number of different binary files called hives on disk. In this paper, the discussed registry is base on Windows XP SP2.

An important traditional field of computer forensic is data carving. In the data carving, the evidential files are extracted from raw images without using any file system metadata, such as the allocation information. Carving is very important for computer forensic when the file system is crashed or damaged.

This paper seeks to use the data carving technique to recover the Windows hive files. Section II briefly describes previous work on file carving and registry forensic. The internal structure of the hive will be illuminated in section III. In section IV, an algorithm for carving the hive files from the raw disk is presented and some validation is offered to enhance our algorithm. Section V describes some experimental results from several disk images and the analysis of the experimental results. In section VI, we outline future work, and in section VII, we conclude this paper.

II. RELATED WORK

WinHex [1] doesn't support the hive file carving in its file type definition database, but it provides the manual extension to support other files. The file type, header, and the relative offset within the hive file at which the magic number occurs can be added to the database, so that the WinHex can carve out the files that start with "regf" and its length is specified by the user as the maximum. But it has poor efficiency when fragment happen and the manual specified length is not very easy for a freshman.

Wong presents the guideline on how to analysis the registry keys to assist forensic investigation on Windows system [2]. Morgan has provided an algorithm to recover deleted data from the Windows registry [3]. Dolan-Gavitt introduces a way on how to forensic analysis of the Windows registry in memory while the attack modifies the cached version of the registry without altering the on-disk version [4].

The first outlined information about hive internal structure has been written by Mark Russinovich [5]. Jolanta Thomassen has provided extensive information of how registry information is organized into data structures on disk [6]. These resources are very important to the hive file carving.

III. HIVE INTERNAL STRUCTURE OVERVIEW

The internal structure of Windows registry hives is illustrated in Fig. 1. Registry hive files start with a header, or base block, and continue with a series of hive bin blocks. The base block has a stable size of 4096 bytes and contains a magic number of "regf". It is possible to extract the name of the file including its local path except for SYSTEM files, the timestamp and the file name from the base block. The maximum length of the stored name is limited to 32 characters, so the file name is often shortened by the Windows operating system (first characters are cut off). The timestamp is a FILETIME structure which is a 64-bit value representing the number of 100-nanosecond intervals since January 1, 1601 UTC [7].

Each hive bin (HBIN) is typically 4096 bytes, but sometimes may be any larger multiple of that size. HBINs are linked together through the HBINs header. The size of a HBIN header is always 32 bytes, and the HBIN header contains the magic number "hbin", the size of this HBIN, the offset to the first HBIN and the timestamp. The size of a HBIN can also be explained as an offset to the following HBIN. In other words, HBINs are strung together, in that each HBIN points to the HBIN that follows. Each HBIN references

978-0-7695-3887-7/09/$26.00 ©2009 IEEE

the beginning of the next HBIN in addition to indicating its distance from the first HBIN. In each hive file, the timestamp of first HBIN is equal to the base block's timestamp, but unfortunately in other HBINs, the value of the timestamp is always zero.

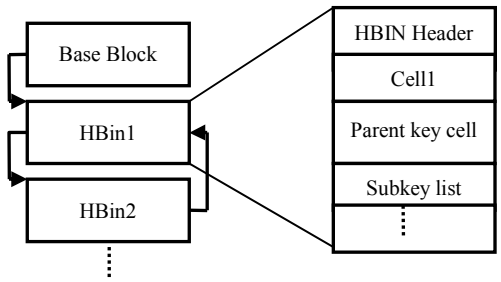

Figure 1. Hive file internal structure

Each HBIN can be found a series of variable-length cells. Each cell's total length is a multiple of 8 bytes, which is specified in the first four bytes of the cell. The registry contains following types of cells: key cell, subkey lists, value lists, value cell, and security cell and class name.

The key cell is the most important structure which contains a number of offset fields to other cells and ties all of these elements together. Key cells contain the signature "nk", the size of the cell, name, timestamp of the key, and offsets to parent key, subkey list, value list, security descriptor and class name. If a key does not have a subkey list, a value list, a security descriptor or a class name, offsets are set to 0xffffffff. Key cells use subkey-lists to reference a set of other key cells. Key cells also store the offset of their parent NK record, except for the root key which don't have a parent key and the value of parent key offset is unavailable. These key-related pointers are illustrated in Fig. 2. Registry key cells form an unbalanced tree. The offsets to parent key and subkey list can be use to the recovery. Subkey lists contain information about the number of subkeys that they refer to, these subkeys are sorted in alphabetical order.

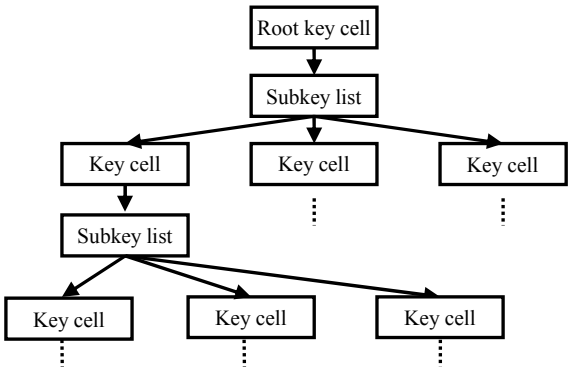

Figure 2. Registry tree

Key cells also contain offsets to value lists which in turn reference value cell. A value list does not have any signature and contains only offsets to key values. It is a little similar to subkey list cells. Value cells stored the signature "vk", value name, type and either value data or an offset to a value data cell. More information of the structure can be found in the paper [8].

IV. CARVING THE HIVE FILES

Hard disks store data in sectors, which are typically 512 bytes long. The operating system then combines several sectors into clusters. Depending on the size of the partition, the hard disk cluster can vary in size, but currently the usual cluster size is 4 KB, which often also is the size of pages in RAM. It means that when the hive files are fragmented, the internal structure of the base blocks are not destroyed, but sometimes the HBINs' may be fragmented when their lengths are larger than the size of cluster. Header carving can be used to extract all the clusters that begin with "regf" header.

In typical Windows XP SP2, all hives file are illuminated in following registry path: HKEY_LOCAL_MACHINE\ SYSTEM\CurrentControlSet\Control\hivelist. 12 hives file can be found. The six user hives locate in user home directory, they are: the ntuser and UsrClass hives for the currently logged on system user, the LocalService user, and the NetworkService user; and the rest (except for HARDWARE) store in %Systemroot%\system32\config path, they are: default; SAM; system; SECURITY; software; and the HARDWARE hive which is generated at boot and provides information on the hardware detected in the system. It means that HARDWARE hive is not stored on disk [9]. Each system user contains two hives: ntuser and UsrClass. The above hive files contain useful and evidential information, but there are some other files that also have the two magic numbers, a summary is included below:

- For each hive file, Windows creates additional supporting files in the related directory. The LOG files store the transaction log of changes to the related hive and contain the structure of the base block. In the offset 0X200, there is an identifier (0X44495254FF) that shows the file is a log file. The identifier can be used to distinguish the hive base block from the log base block. The SAV files are the backup copies of hives created at the end of text-mode phrase during Windows XP setup, and contain both the base blocks and HBINs structures. The base blocks of the SAV files don't include the timestamp and file name.
- There are some hive files with the path of %Systemroot%\repair. These hive files were created when the Windows system was installed in the first time and their base blocks have neither timestamp nor file name. REGLOCS.OLD is a registry rubbish file that contains the base block without the timestamp and file name.
- hiberfil.sys is a file that the system creates when the computer goes into hibernation mode. Windows uses the file when it is turned back on. This file size is also the same as the physical memory size. It always contains the registry data that is loaded into memory. pagefile.sys is the virtual memory file. Typically, on install, Windows sets the size of the file at around 1.5 times your physical memory size. Sometimes it contains the registry data when the memory is not enough to use, and the system may put some idle

registry data into the virtual memory. The structures of the base blocks and HBINs in above two files are intact.

- The unallocated space which is not used on disk sometimes includes several registry data. These unavailable data are produced when the Windows system needs delete a hive; it drops the original space, but leaves the registry data. This situation always occurs when we delete a system user, the ntuser and UsrClass are deleted. There is another probability that these data are created by last Windows system and not covered by the current system. The structures of the base blocks and HBINs in above files are intact.

From the above explanation, we have developed a method for carving the most important hives, except for the HARDWARE hive, because it doesn't store on disk. The technique demands on that each HBIN block structure is intact. There are no checksums or other integrity mechanisms built into HBIN blocks, it is difficult to validate that if a HBIN is intact. The carving process shows in figure 3. They are:

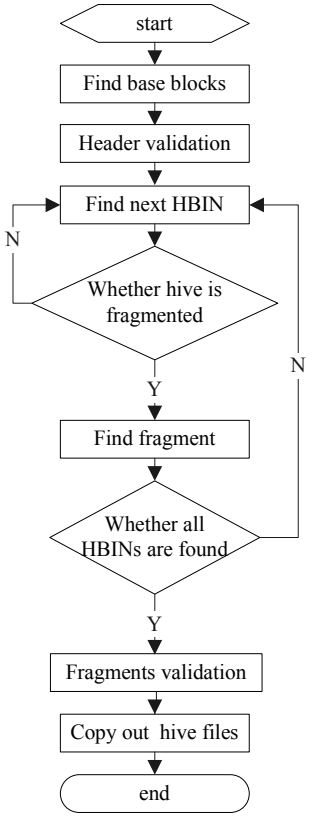

Figure 3. The flowchart of our algorithm

Step 1: Find base blocks. Based on above expression, Header carving technique can be used in this step. If a cluster begins with the magic number "regf" and the file name is included the 11 hives file name, this cluster should be the base block of a hive file.

Step 2: verify whether the hive file is fragmented. The HBINs have the magic number "hbin", the offset to the first HBIN and the length. According to the hive internal structure, all HBINs follow after the base block, and the difference of the offset to the first HBIN between two conjoint HBIN blocks is the length of the previous HBIN. If this rule is broken, the fragment must happen. There is another fragmented case. If a HBIN's next cluster not starts with the magic number "hbin" and the current found HBINs' total size is not greater than the base block's total size of all HBINs. Otherwise, all the HBIN blocks can be found and the found HBIN blocks length is equal to the base block's total size of all HBINs.

Step 3: Find all the fragments, and reassemble them together. If the hive file is fragmented, the next fragment must start with the magic number "hbin", and its offset to the first HBIN can be made sure because the previous HBIN' offset to the first HBIN and its size are known. If a found fragment already belong to another hive, we ignore this fragment and try to next one while accords with our condition. Based on this information, the fragment may be sought, but the candidates may be more than one. Try to find all the fragments until total size of all HBINs is the same as the base block's total size of all HBINs and integrate those HBIN blocks together by the offsets to the first HBIN. There is a high-level validation is mentioned to verify the all found fragments are the real ones. For each new carved hive, we try to build the registry tree from the root key, if the build fails, there must be some fragments don't fit the hive. Try to use some other candidates, until the registry tree build succeeds.

To decrease the false positive, header validation should be used to delete the corrupted base blocks. The available bases block should include the magic number, file name and timestamp. The timestamp has very important role in the carving. If some base blocks have the same file name, the timestamp can be used to select the latest base block. Because other data blocks are produced at the process of running system and are not updated their internal registry data after its creation, but the latest ones are composed of the most valuable registry evidences. There is another way to validate base blocks is checksum that can be stored at offset 0X1FC. As mentioned above, the log file can be excluded easily by the identifier (0X44495254FF) and the first HBINs should contain the timestamp which is the same as the base block timestamp.

Currently, we only care the single system user condition, but Windows system always has two or more users. In order to carve the hive of multi-user, first, we should carve SAM hive accurately and acquire the key with path \SAM\Domains\ Account\Users\Names. The available system users' names are this key's subkeys. So each found base block of system user's ntuser, the filename must be included in the Names key's subkeys; it means that the found ntuser is belonged to an active system user, not a deleted one. There is an issue that the every UsrClass hive has the same file name. As mentioned above, the first characters of the file name are cut off, when its length is longer than 32 characters. Unfortunately, the names of the three UsrClass hives are too long, and the first characters are deleted resulting as the same file name. So for carving UsrClass hives, in the first step, we care all the base blocks that include the file name of UsrClass and check the root key name. The table 1 shows the relationship between UsrClass hives and root key names. The root key names are the same as the security identifiers. Second, we can use root keys to

identify the base blocks of two Service Users and select the latest one for each UsrClass hive by the timestamp. But every system user has the same root key because their SID is the same; it says that we can't identify the UsrClass hives are accurately belonged to which system users; however our method can carve these UsrClass hives successfully. The above shows us how to find the base blocks, and other two steps of our method are unchanged in multi-user situation.

TABLE I. THE RELATIONSHIP BETWEEN USRCLASS HIVES AND ROOT KEY NAMES

UsrClass name	Root key name
LocalService User	S-1-5-19_Classes
NetworkService User	S-1-5-20_Classes
System User	S-1-5-21-domain_Classes

V. EXPERIMENTAL RESULTS

Our method attempts to carve the latest hive files and copy them out. In the paper [4], it says that Windows does a reasonably good job at avoiding registry fragmentation over short periods of time. Based on this, several experiments were performed to evaluate the effectiveness of our technique. First, we extracted the original hive files before took the disk images. Those original hives will be used to compare with the carved hives, which can evaluate the efficiency of our method. Second, the hives were taken from the disk images by WinHex and our tool. Finally, we counted the number of keys and values for the original hives and carved hives. We tested three images with Windows XP SP2:

1. A recently installed system had been run several days; both of default and the ntuser of system user hives are fragmented between two HBIN blocks,

2. The second image had been in use for 5 months. The ntuser of system user is fragmented within a HBIN.

3. For the first system, we added some registry hive data in the unallocated space, after several days running, then took the disk image.

TABLE II. NUMBER OF KEYS/VALUES CARVED (DI MEANS DISK IMAGE)

	DI 1	DI 2	DI 3
Keys(original)	54452	128359	54452
Values(original)	137998	242366	127998
Keys(WinHex)	66863	201043	749561
Values(WinHex)	163656	397023	181943
Keys(our algorithm)	54452	128346	54452
Values(our algorithm)	127998	242302	127998

The result is showed in table 2. The data reveal a few interesting facts: first, WinHex recovered more keys and values than either original copy or our algorithm, because WinHex can't clearly distinguish the real hives from the confused ones that we mentioned above. Second, our technique of carving the hive files is more effective than the WinHex recovery tool; it carved all the useful keys and values from the system when some hives are fragmented between two HBINs. Some registry data are missed with our algorithm in the second experiment, because the ntuser of system user includes a HBIN which is fragmented in two pieces. As mentioned in section IV, currently we can't verify a HBIN block whether is fragment, but our method has carved more keys and values than WinHex. Finally, our technique carves the latest hives, so it can ignore the registry data in the unallocated space. As a result, our technique has high successful rate in this complex situation.

VI. FUTURE WORK

Currently, our tool for carving the registry hives work well when each block is not fragmented. In order to be more useful, support for the fragmented condition should be added. Unfortunately, it is difficult to verify this condition, so we need more practices in this area. In addition, we should do some research to distinguish between different UsrClass hives of system users. Finally, one limitation of the technique is that in order to carve the latest hives, it ignores some import hives, such as the hives of a deleted user which the investigators have interest in this user.

VII. CONCLUSIONS

In this paper, we aim to propose an adaptive method to carve the registry hive files based on file's internal structure. Our approach, which is not based on file system's metadata information, can be applied in different kinds of file system for disk forensic. Recovery of the hives can be performed with more accuracy than other file carving techniques because the exact internal structure is used. The method is successful even if the hives are fragmented between two HBIN blocks.

VIII. ACKNOWLEDGEMENTS

We thank Natural Science Foundation of Zhejiang Province under Grand No.Y106176 and Y106427, and Science and Technology Research Planned Projects of Zhejiang Provincial No.2007C33058 and 2008C21075 for funding this work.

IX. REFERENCE

[1] X-Ways Corporation, "WinHex," http://www.winhex.com/index-c.html.

[2] Wong L.W, "Forensic Analysis of the Windows Registry," http://www.forensicfocus.com/index.php?name=Content&pid=73&page=1.

[3] Morgan T.D, "Recovering deleted data from the Windows registry," Digital Investigation. Vol. 5, Supplement, pp. 33–41,2008.

[4] Dolan-Gavitt B, "Challenges in Carving Registry Hives from Memory". http://moyix.blogspot.com/2007/09/challenges-in-carving-registry-hives.html.

[5] Russinovich Mark, "Inside the Registry," http://technet.microsoft.com/en-us/library/cc750583.aspx.

[6] Jolanta Thomassen, Forensic analysis of unallocated space in Windows registry hive files, University of Livepool, 2008.

[7] Microsoft Corporation, "FILETIME Structure," http://msdn.microsoft.com/en-us/library/ms724284.aspx.

[8] Morgan T.D, "the windows nt registry file format," http://sentinelchicken.com/data/TheWindowsNTRegistryFileFormat.pdf

[9] Dolan-Gavitt Brendan, "Forensic analysis of the Windows registry in memory" Digital Investigation. Vol. 5, Supplement 1, pp. 52–57,2008.