Volume 4, Issue 3, March 2014 ISSN: 2277 128X
International Journal of Advanced Research in Computer Science and Software Engineering
Research Paper
Available online at: www.ijarcsse.com

Technical Prestudy of Computer Investigation and Analysis Techniques for Presentation of Gathering Evidence in Legal Constitution

Mr. D. S. Jadhav
I/C Director,
Ideal Institute of Management (IIMK), Kondigre, MS, India

Mrs. S. S. Bardiya
Assistant Professor,
Ideal Institute of Management (IIMK), Kondigre, MS, India

Mr. S. M. Khandke
Assistant Professor,
Ideal Institute of Management (IIMK), Kondigre, MS, India

Dr. S. K. Patil
I/c Principal,
Associate Professor
BPSCC, Barshi, India

Abstract - Digital forensics is a branch of forensic science concerned with the use of digital information (produced, stored and transmitted by computers) as source of evidence in investigations and legal proceedings. Digital Forensic Research Workshop has defined digital forensics as - "The use of scientifically derived and proven methods toward the preservation, validation, identification, analysis, interpretation, documentation and presentation of digital evidence derived from digital sources for the purpose of facilitating or furthering the reconstruction of events found to be criminal, or helping to anticipate unauthorized actions shown to be disruptive to planned operations"[1]. The highly technical nature of computer crimes facilitated a wholly new branch of forensic science called digital forensics. Instead of dead bodies, digital forensic scientists collect and analyze data produced, transmitted, and stored by digital devices. The aim of digital forensic analysis remains the same - to clarify events of the incident and, ultimately, identify its perpetrators.

Keywords: Digital Forensics, digital information, digital source, analysis, digital devices.

I. INTRODUCTION

Before starting the actual implementation of the prototype software we have to make an in-depth investigation of which timestamps there are to collect, how we can find them and how we can decode them. This is a detailed technical pre-study, which will serve as a base for designing and implementing the actual evidence extraction part of the prototype. We have focused the prototype on systems running Windows but log files gathered from other systems, mainly server systems, have also be used. Our goal has also been to make the design of the prototype so scalable so that it would be easy to implement support for other file systems and data formats in the future[2]. To prepare for this and to give a broad overview we have also included detailed information about other file systems in the pre-study. Furthermore to some extent evidence from different files from other client and server systems are covered in the report although we have only implemented support for a small subset of them in the prototype. First of all we have to identify the sources, where we can find evidence of which we can determine its time. Then we have to list all the kinds of evidence we can gather from the sources. We will put an emphasis on where the timestamp information can be found and in which way it is coded, because that is the core of what needs to be handled in the scanner prototype.

II. FILES IN DIFFERENT FILE SYSTEMS

The majority of all operating systems use files in a file system to store data. There are many different brands of file systems available on most operating systems, some systems use unique, custom-built file systems while other operating systems use common and widely used file systems. The main cause of a file system is to organize data in a files and folders hierarchy. Additional to giving files paths in the hierarchy and a size, most file systems allow meta-data about each file to be read and written (meta-data is extra information, for example author, image size and sound file length). It is common that simple properties like time for last-changed, created and last-accessed, as well as if it is possible to read, write or execute different files is specified by the file system. It is also common for file systems to hold information about which access different users and groups should have on different files. Some file systems even allow extra data streams, additional to the content, to be contained within a file. In Windows environments a file system called FAT (File Allocation Table) and another called NTFS (Originated from Windows NT) is mainly used for hard drives. Apart from in Windows FAT is also commonly used in embedded devices for example digital cameras, mobile phones and mp3 players[3]. It is also a very common format for memory cards. FAT comes in different flavors: FAT12, FAT16 and FAT32. The number after FAT specifies how many bits are used to store unit addresses (we call the smallest amount of space possible to allocate a unit), hence a higher number will support larger storage volumes and smaller unit sizes. FAT12 is mainly used for floppy disks today whereas the others are used for hard drives and other larger storage devices. All files in a FAT file system contain the time when it was created, last modified and last read. The timestamps are stored as actual year (seven bits), month (four bits), day (five bits), hour (five bits), minute (six bits) and second (five bits). To

store seconds at one second accuracy six bits would have been required so the five bits are only enough for saving the seconds with 2s accuracy. The year field starts counting at 1980. The accuracy is different for each time stamp. The last access timestamp only contains the date. The last modified timestamp contain both date and time and the creation time even include an extra byte giving an accuracy down to 1/100 second[4]. An overview of this can be seen in the following Figure 1.

```
//This is a conceptual overview, actual positioning vary

struct Date { //Used for create, modify and access
      unsigned int year       : 7; //Starts counting on 1980
      unsigned int month      : 4;
      unsigned int day        : 5;
};
struct Time { //Used for create and modify
      unsigned int hour : 5;
      unsigned int minute     : 6;
      unsigned int second     : 5;
};
struct TimeFiner { //Used for create only
      unsigned int hundredth  : 8;
};
```

Figure 1 : Timestamp format in FAT16 and FAT32

NTFS is an alternative to FAT and is mainly used in internal hard drives. NTFS supports very large drives and exist in a number of versions, where the latter also natively support features like encryption end compression. It also contains support for disk quota (limiting users disk space) and enables an administrator to specify which directories different users and groups should have access to and what kind of access[4,5]. NTFS like FAT stores information about when a file was created as well as last changed and read but it also stores an additional timestamp. This additional timestamp specifies when the metadata for the file was last changed. NTFS seem to store timestamps using a standard Windows structure called FILETIME. FILETIME is a 64 bit unsigned integer counting the number of 1/10000000 seconds from 1601-01-01. We could not find any reference for this but according to our tests by comparing the time extracted via a disk image with the representation in Windows we are very certain that it is correct. A representation of this timestamp format can be seen in this Figure 2.

```
struct WindowsTimestamp {
      unsigned int timestamp  : 64; //Number of seconds * 10000000
                                    //since 1601-01-01 00:00:00
};
```

Figure 2: Timestamp format in NTFS

On Unix system UFS1 and UFS2 can be found. These file systems exist in different modified and improved versions in many operating systems. Some examples of operating systems supporting this file system are: Apple OSX, Sun Solaris, OpenBSD, NetBSD, FreeBSD and HP-UX. Timestamps in UFS are composed of a 32 bit unsigned integers, counting the number of seconds since 1970-01-01 00:00:00 GMT. For completeness we illustrate this timestamp in the following Figure. On Linux systems today you can find EXT2 and EXT3 file systems. We refer to both of them as just Ext. EXT3 is just EXT2 with a journal added. EXT3 is perfectly compatible with EXT2 however if an EXT3 volume is mounted with EXT2 drivers, the advanced journaling features will not be in effect. Ext contains 4 timestamps, last access, last modification, last change and deletion time[6,7,8]. Timestamps in Ext is like in UFS composed of a 32 bit unsigned integer, counting the number of seconds since 1970-01-01 00:00:00 GMT.

```
struct UnixTimestamp {
      unsigned int timestamp  : 32; //Number of seconds since
                                    //1970-01-01 00:00:00 GMT
};
```

Figure 3 : Timestamp format in Unix systems

On the Macintosh a file system called HFS (Hierarchical File System) and its sequel, HFS+ (Mac OS Extended) is used, the later introduced in OS 8.1. Since HFS is legacy we will not go into detail on what timestamps there are to find however we will look into HFS+. In HFS+ each timestamp is stored as a 32bit unsigned integer holding the number of seconds since 1904-01-01 00:00:00 GMT, The time is specified in GMT by contrast to original HFS who would specify the time in the computers local time zone. HFS+ contain timestamps for the creation, last content modification, last attribute modification, last accessed and when the file was last backed up[7,8]. See Figure 4 for an illustration.


```
struct HFSPlusTimestamp {
      unsigned int timestamp  : 32; //Number of seconds since
                                    //1904-01-01 00:00:00 GMT
};
```

Figure 4 : Timestamp format in HFS+

On the CD-ROM you usually find a file system called ISO9660. ISO9660 is a standard file system for CR-ROMs which implements the least common denominator for most file systems, leaving it to system developers to add their own extensions to the format to make it comply with the systems native file system (like special attributes and meta-data). In the ISO9660 format four different timestamps exist for each file. It contains timestamps for the creation and last modification of a file as well as the uncommon timestamps effective and expiration time. The effective and expiration timestamps specifies when a certain file can be first used and when it is obsolete and should not be used any more[9,10,11,12]. The timestamps in ISO9660 is composed of seventeen bytes. The bytes contain information as following: number of years after 1900, month, day, hour, minute, second and finally offset from GMT in fifteen second intervals. By contrast to other timestamp format, this format saves all values except the time zone information in ASCII whereas the other formats use binary representation. A detailed description of the format can be seen in the following Figure 4.10. On the DVD a file system called UDF (Universal Disk Format) is used. This file system format can be seen as an extension of ISO9660, improving the performance and removing some of its limitations. It also enables the use of much larger media. The same format is used for expressing timestamps as in ISO9660 and the different timestamps stored in UDF are also the same except that the last modification timestamp does not exist in UDF[12].

```
struct ISO9660AndUDFTimestamp {
      char[4] year;
      char[2] month;
      char[2] day;
      char[2] hour;
      char[2] minute;
      char[2] second;
      char[2] hundredthSecond;
      char    gmtOffset;
};
```

Figure 5 : Timestamp format in ISO9660 and UDF

III. WINDOWS REGISTRY KEYS

On early windows systems all application settings were stored in a central text-file called win.ini located in the Windows folder. Later the application settings were moved to individual text files ending with .ini because of a 64K file limit of the win.ini file. The text-file system had many drawbacks where some examples are slow performance and administration issues. In Windows 3.1 the Windows Registry was introduced. The registry is a so-called hierarchical database created to organize all system and application settings into a structured layout. The registry was build as a tree, structure almost like the directories in a file system. By contrast to a common file system, instead of containing files, each directory contains settings. At the time the complete registry was stored in a signle file called Reg.dat located in the Windows folder. In Windows 95, 98 and ME the registry was moved to two files, User.dat and System.dat located in the Windows folder. If user profiles were enabled every user also got their own User.dat file, containing personal settings, located in their profile folder. In Windows ME an additional Classes.dat file was added. In Windows NT, 2000, XP and Vista the registry was split into even more files, most stored in Windows\System32\Config. In the Config folder you can find a file named “Default”, which stores the default settings, “SAM” which stores accounts securely, “SECURITY” which stores security related keys[13,14]. Finally there are the “SOFTWARE” and “SYSTEM” files, which stores settings for all installed programs and the settings of the Windows system respectively. User specific settings are stored in a file called Ntuser.dat, located in the users profile directory. The registry contains almost all settings in windows, from the control panel and information about connected hardware. It also contains application specific settings and data, for example recently open files, recently visited URLs. The windows registry is like a goldmine for the forensic examiner[15].

The logical structure of the registry is not mapped directly to the hive files, each hive file represent one or more sub paths in the registry hierarchy. In the root of the registry there are 5 main categories. HKEY_CLASSES_ROOT,HKEY_CURRENT_USER,HKEY_LOCAL_MACHINE,HKEY_USERS and HKEY_CURRENT_CONFIG. Below each of these there are a tree of so-called Keys. Keys could somewhat be considered equivalent with folders in a file system, each having a name in the hierarchy. Each Key can contain sub keys,

but keys can also contain value elements. A value element is a named object containing a value of a specific type. There are many value types in the registry. Some examples of value types are binary, word and string. The different registry files are generally called hives. These hive files are stored in a special binary format and held in memory although parts not recently used are paged out to disk to save memory. The binary file format is not easily readable by humans without a special reader or editor and by using the Windows Registry Editor not all meta-data are accessible. Something that makes the registry incredibly important in computer forensics is that each key has a last-write timestamp, which specifies when the key was last written. This is something that is not showed by the original registry editor provided by Windows. This timestamp specifies the number of nanoseconds since 1601-01-01 00:00:00 and is in the same format as the FILETIME structure described in above Figure 5. There is no official detailed information about the internal structures of the registry available[15,16,17,18]. Although there are individuals and organizations that have reverse engineered the file format of the registry hives and sorted out most of how the majority of the data structures within the registry works. The Samba project and the Wine project are two large contributors to this success. Since the registry in Windows NT/XP is what we have added support for in the prototype we have chosen to go a little bit deeper into this version of the Windows registry. In the rest of this chapter we will refer to the registry version used in Windows NT/XP as just "the registry". The registry is divided into blocks of 4096 bytes each and there are two kinds of blocks. There are the "regf" block and "hbin" blocks. The regf block is placed at position zero of the registry file and seems to be a header block specifying generic information about the registry hive. The regf block contains some important information. For example it contains an offset to the root entry and the size of the binary data. A detailed overview of the regf block can be seen in the following Figure 6.

```
struct regfHeader {
      char[4]     signature;        //always "regf"
      int         unknown1 : 32;
      int         unknown2 : 32;    //Seem to be equal to unknown1
      WindowsTimestamp lastWriteTime;
      int         unknown3 : 32;    //Seem to be always 1?
      int         unknown4 : 32;    //Seem to be always 3?
      int         unknown5 : 32;    //Seem to be always 0?
      int         unknown6 : 32;    //Seem to be always 1?
      int         offsetRoot : 32;  //First key record
      int         dataBlockSize : 32;
      int         unknown7 : 32;    //Seem to be always 1?
      int         checksum : 32;    //Sim of all dwords from
                                    //0x0 to 0x1BF
};
```

Figure 6 : Header of the regf block

The binary data is located in another kind of blocks called hbin blocks. These blocks are always a multiple of 4096 bytes and each has a header containing among other things a pointer to the first and the next hbin block in the chain. Apart from the header each hbin block contain a list of "records". Each record begins with a size defined as a 16bit signed integer and right after the size follows data of a length specified by the size. If the size is negative the block should be considered as free space of "–record size" bytes

```
struct hbinHeader {
      char[4]     signature;        //Always "hbin"
      int         offsetToFistBlock;
      int         offsetToNextBlock;
      char[16]    unknown;
      int         blockSize;
};
```

Figure 7 : Header of the hbin block

There are 3 record types that are important to us. There is the nk-record, which is actually representing a registry key. The structure contains the key name and a timestamp specifying when the key was last written to disk. It also contains a pointer to a list of subkeys and a pointer to a value-list. The values are stored in another record called a vkrecord. The vk-record contains a name and type of the value. It also contains a pointer and a length, which together describes where the data of the value is located.


```
struct nkRecord {
      char[2]     signature;  //Always "nk"
      int         keyType:16; //0x2C = rootkey, 0x20 = other key
      WindowsTimestamp lastWriteTime;
      int         unknown1          : 32;
      int         parentKeyOffset   : 32;
      int         numberOfSubkeys   : 32;
      int         subkeysLfRecordOffset : 32;
      int         numberOfValues    : 32;
      int         valueListOffset   : 32;
      int         skRecordOffset    : 32;
      int         classNameOffset   : 32;
      int         unknown2          : 32;
      int         nameLength        : 16;
      int         classNameLength   : 16;
      char[nameLength] keyName;
};
```

Figure 8 : An nk-record

```
struct valueList {
      int:32      value[];    //A list of value offsets
};
```

Figure 9 : A value-list

```
struct vkRecord {
      char[2]     signature;        //Always "vk"
      int         nameLength  : 16;
      int         dataLength  : 32;
      int         dataOffset  : 32;
      int         dataType    : 32;
      int         flags       : 16;
      int         unknown     : 16;
      char[nameLength] name;
};
```

Figure 10 : A vk-record

IV. CONCLUSION

We have created a prototype tool. The tool is divided into two parts: A scanner that scans hard drives for evidence and generates index files. These index files can be read by the other part, the viewer. The scanner can handle can handle file systems like NTFS, FAT. It can then find e-mail, instant messaging and a number of other file formats, which it will add to the index file. The viewer will read index files and display a timeline where all evidences are displayed. In the viewer the end user can zoom, browse evidence by timestamp and look into evidence properties and hexadecimal data. The tool is evaluated by a set of test subjects that compared the solution with a well-known software forensic tool available in the market. The results show that it could be very advantageous to implement the ideas in Cyber Forensics Time Lab and that it at least would make it a competitor when the case is appropriate. The user testing shows a number of places where improvements should be made. All the test subjects seemed to be positive about the tool in general.

REFERENCES

1. Computer Forensics World. "Computer Forensics Basics: Frequently Asked Questions". *(*Online) Retrieved April 3rd, 2009
2. *Dixon*, Phillip D. *"*An overview of computer forensics". 2005 Potentials, IEEE. Volume 24 Issue 5. IEEE International.
3. United States Computer Emergency Readiness Team. "Computer Forensics". *(*Online) Retrieved April 3rd, 2009
4. Marcella Jr., Albert J. *"Cyber Forensics: A Field Manual for Collecting, Examining and Preserving Evidence of Computer Crimes"*. 2008. Taylor & Francis Group, LLC. Auerbach Publications.
5. How Stuff Works. "How Computer Forensics Work". *(*Online) Retrieved April 5th, 2009

6. Vacca, John. *"Computer Crime Scene Investigation: Second Edition".* 2005. Cengage Learning. Charles River Media, Inc.
7. Hayes, Darren R and Qureshi, Shareq. "A Framework for Computer Forensics Investigations Involving Microsoft Vista". 2008. Systems, Applications and Technology Conference, 2008 IEEE Long Island.
8. Allen, William H. "Computer Forensics". 2005. Security & Privacy, IEEE. Volume 3, Issue 4.
9. Access Data. "Forensic Toolkit". (Online) Retrieved April 13th, 2009
10. Sleuthkit.orf. "The Sleuth Kit". (Online) Retrieved April 13th, 2009
11. Get Data Software. "Recover My Files". (Online) Retrieved April 13th, 2009
12. New Technologies, Inc. "Computer Evidence Processing Steps". (Online) Retrieved April 13th, 2009
13. Computer Forensics Online www.shk-dplc.com/cfo
14. Asian School of Cyber Laws Available at: http://www.asianlaws.org, May 2004
15. Information Security and Forensics Society (ISFS), *Computer Forensics, Part 2: Best Practices*, May 2004, http://www.isfs.org.hk/publications/ComputerForensics/ComputerForensics_part2.pdf
16. Regulation of Investigatory Powers Act 2000, Anti-terrorism, Crime and Security Act 2001
17. Guide to Cyber Laws by Rodney D. Ryder Hand book of Cyber & E-commerce Laws by P.M. Bakshi & R.K.Suri
18. DIGITAL EVIDENCE Emerging Problems in Forensic Computing Peter Sommer
19. Federoc Ga, egps (2005) "Computer Forensic : An Overview" Information Sciences Control Jr. Vol. VI.
20. A case for forensic tools in Cross Domain data transfers of SANS institutes InfoSec reading room.

Authors Biography

D. S. Jadhav was born in India, Maharashtra, in 1979. He received the BCA, MCA, MBA degrees from Shivaji University, Kolhapur (MS), University of Pune (MS) and Sikkim Manipal University respectively. He is registered student for PhD (2010) as a Research Scholar, from Solapur University, Solapur. From 2008-2010 he was worked as lecturer Smt. K. W. College, Sangli. From 2010 to 2012 he was worked as Asst. Professor at Bharat Ratna Indira Gandhi College of Engineering, Solapur (MS) and Sinhgad Institute of Computer Sciences, Pandharpur(MS) respectively. From September 2013 to till date he is working as I/C Director at Ideal Institute of Management(IIMK), Kondigre – Ichalkaranji (MS). He has published more than 30 papers in International and National journals and conference Proceedings. He is member of various National & International Professional Bodies and member of Editorial / Reviewer of various International Journals. His research interest includes Cyber Crime, Cyber / Computer Forensic, Information Security.

S. S. Bardiya was born in India, Maharashtra, in 1982. She received the BSc(CS), MSc (CS), MCA degrees from Swami Ramand Tirth Marathwada University, Nanded (MS), Manav Bharati University (HP) respectively. She had grabbed Gold Medal in MSc (CS) in 2004. From 2004-2006 she was worked as lecturer Dr. J. J. Magdum College of Engineering (Polytechnic Wing), Jaysingpur . From 2008-2010 she was worked as lecturer & HOD at G. A. College, Sangli. From 2011-to till date she is working as Asst. Professor at Ideal Institute of Management(IIMK), Kondigre – Ichalkaranji (MS). Her research interest includes Cyber Crime, Information Security, and ICT.

S. M. Khandke was born in India, Maharashtra, in 1982. He received the BSc, MCA degrees from Shivaji University, Kolhapur (MS). From 2008-2009 he was worked as lecturer at KES KRP ACS College, Islampur. From 2009-2013 he was worked as lecturer at D. R. Mane College, Kagal. From 2013 to till date he is working as Asst. Professor at Ideal Institute of Management(IIMK), Kondigre – Ichalkaranji (MS). He has published more than 2 papers in International and National journals and conference Proceedings. His research interest includes Cyber Crime, Information Security, and ICT.

Dr. S. K. Patil was born in India, Maharashtra, in 1962. He received PhD from, Swami Ramanand Tirth Marathwada University, Nanded. He worked as Lecturer, Assistant Professor & now he is working as Associate Professor & Head of Department at BPSCC, Barshi, (MS). He has published more than 45 papers in International and National journals and conference Proceedings. He is member of various National & International Professional Bodies and member of Editorial / Reviewer of various International Journals. His research interest includes Traditional Law, Cyber Crime, Cyber / Computer Forensic, Information Security, Management.

