Analysis of hidden data in NTFS file system

Cheong Kai Wee
Edith Cowan University
ckw214@yahoo.com

Abstract

Criminals with sensitive information such as crime records tend to hide/encrypt this information so that even if their computers are collected by police department, there is no evidence that can be used against them. There are many ways data can be hid. The most famous ways are data encryption and steganography. File system, in addition, can also be used to hide data.

This paper discusses some of the possible ways to hide data in NTFS file system and analysis techniques that can be applied to detect and recover hidden data. This paper focuses on criminals as the users of data hiding techniques and the main targets that they want to hide data from are forensic analysts. Certain data hiding techniques that can only be used to hide data against normal users such as setting the hidden attribute of a file will not be included.

Keyword

Data hiding, analysis technique, NTFS

INTRODUCTION

This paper discusses some of the methods that can be used to hide data in NTFS and analysis techniques that can be used to detect and recover hidden data. Target readers for this paper are forensic analysts and examiners. Throughout this paper, the phrase "suspect" is used to refer to the owner of digital devices, where analysis is performed to retrieve digital evidence.

RunTime's DiskExplorer for NTFS v2.31 is used to create the hidden data manually for testing purpose. The only exception is hidden data for alternate data stream which is created by normal DOS command. Tools that are used to analyse hidden data are Windows XP chkdsk, Sleuth Kit 2.02, Foremost 0.69, comeforth 1.00, dd, hexedit and strings. Test data is created on a machine with Windows XP version 5.1.2600.

BACKGROUND OF NTFS

In NTFS, everything is file. This includes file system metadata about the structure of the file system. MFT (Master File Table) is the heart of NTFS. Every file or directory has at least one entry in MFT (Master File Table). Microsoft calls each entry in MFT as file record and its default size is 1024 bytes (Mikhailov, n.d.). The first 42 bytes is fixed for MFT entry header and the rest of the entry stores attributes, which is small data structure with specific purpose. Example of attributes are $STANDARD_INFORMATION, $FILE_NAME and $DATA (Microsoft, n.d.). The content of an attribute can be either resident or non resident. A resident attribute stores its content in the MFT entry. A non resident attribute stores its content at external clusters. The list of clusters used is stored as cluster run in the run list of an attribute.

Data unit in NTFS is called cluster, which is the smallest disk space allocation unit. Every cluster in NTFS has a LCN (Logical Cluster Number). The cluster number starts with 0 at the first cluster of the file system (Svensson, 2005). Clusters belong to a file are also assigned a VCN (Virtual Cluster Number). For example, a file with 6 clusters will have cluster 1 of the file with VCN 0 and last cluster with VCN 5.

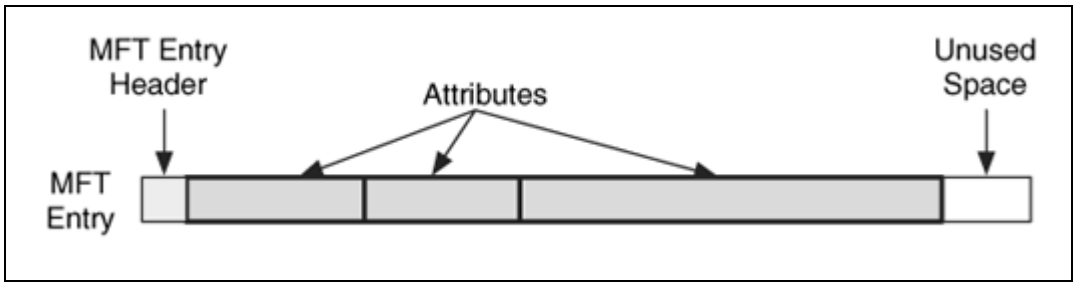

Figure 1: Structure of MFT entry (Carrier, 2005)

Metadata files are files that describe the file system. Example of metadata files are $MFT, $MFTMirr, $LogFile, $Volume, $AttrDef, ., $Bitmap, $Boot, $BadClus, $Secure, $Upcase and $Extend. Table 1 show the description of some of the metadata files.

Table 1: Description of metadata files in NTFS (Solomon & Russinovich, 2000)

Metadata file	Description
$MFT	Store MFT record
$MFTMirr	Contain partial backup of MFT
$LogFile	Transaction logging file
$Volume	Contain volume information such as label, identifier and version
$AttrDef	Attribute definition
.	Root directory of file system
$Bitmap	Contain the allocation status of all clusters
$Boot	Contain the boot record
$BadClus	Mark clusters as bad clusters
$Secure	Contain information about the security and access control information

FAKED BAD CLUSTERS

For old hard disks that do not have the capability to handle errors, operating systems detect and mark sectors/ clusters as damaged. Nowadays, modern hard disks handle bad sectors themselves by remapping bad sectors to spare sectors (storagereview, n.d.). It is unlikely that an operating system would detect bad sectors before hard disk does. Clusters marked as bad may be used to hide data.

In NFTS, bad clusters are marked in metadata file called $BadClus, which is in MFT entry 8. Originally, $BadClus is a sparse file which file size is set to the size of entire file system. When bad clusters are detected, they will be allocated to this file.

The size of data that can be hid with this technique is unlimited. Suspects can simply allocate more clusters to $BadClus and use it to hide data.

Procedure to create test data

1) Clusters are added to the run list of $Bad attribute of $BadClus file
2) The size of $Bad attribute and the size of this MFT file record are modified if necessary
3) Allocation status of clusters used to hide data is set to 1
4) Hidden data is pasted to the clusters

Process of how hidden data is created manually in faked bad clusters is shown in Appendix A.

Analysis techniques

Sleuth Kit is used to analyse the file system. Throughout this paper, /case1/image1 will be used in examples as the acquired image of NTFS that need to be analysed. Figure 2 shows the flow to analyse hidden data in faked bad sectors.

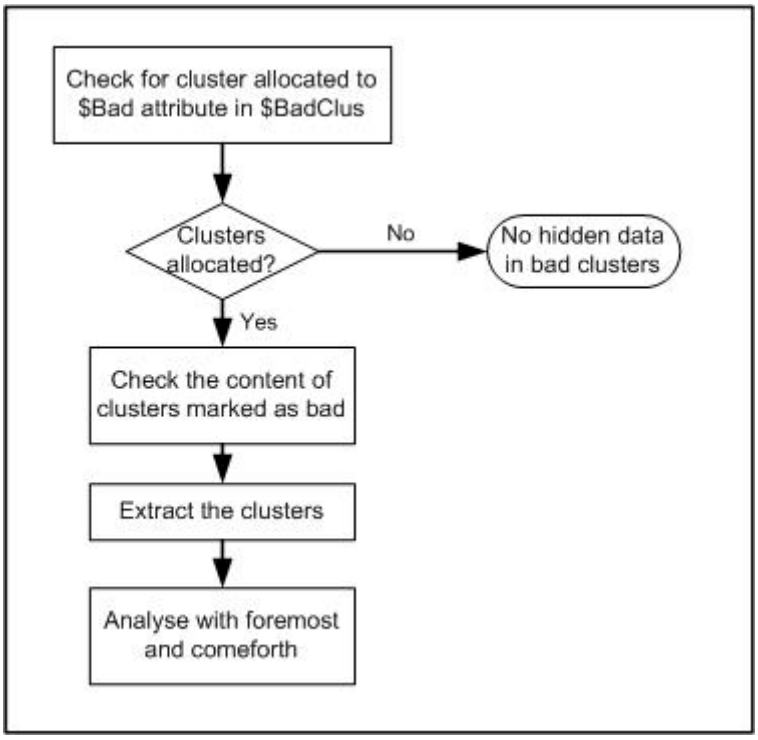

Figure 2: Flow to analyse hidden data in faked bad clusters

Check for clusters allocated to $Bad attribute of $BadClus.

```
istat /case1/image1 –f ntfs 8
```

If there is clusters allocated to $Bad, it must be inspected as modern hard disks usually handle bad sectors themselves. If you have the physical hard disk with you, you can also perform a surface scan to verify whether there are bad sectors on the disk.

Check the content of the clusters with dcat in Sleuth Kit. However, this only reveal hidden data if the data is stored in ASCII encoding. In this example, let's assume cluster 383624-383635 are marked as bad cluster and suspected to contain hidden data.

```
dcat /case1/image1 –f ntfs 383624 12
```

For further analysis, extract the clusters and use data craving tools such as foremost and comeforth to recover data.

```
dd if=/case1/image1 bs=4096 skip=383624 count=12 of=/case1/badclusters
foremost –c /etc/foremost.conf –v –o /forensic/recover /case1/badclusters
```

This analysis technique works if a suspect stores the files sequentially in the clusters. However, suspects can segment a file and store it in non sequential way or even randomly in clusters (Carvey, 2004b). For example, if a suspect stores a Microsoft Word file with 383629 as starting cluster, moving backward and store the last portion of file in cluster 383624, foremost is unable to recover the file correctly. This technique does not prevent suspects from retrieving the files as they can record the order of clusters when hiding them.

In addition, a suspect may also remove the signature of a file to avoid detection of data craving tools (Carvey, 2004a). Data carving tools such as foremost recover files based on their data structure such as header and footer (sourceforge, n.d.).Without these structures, it is impossible to recover the file. During this research, testing has been carried out to recover files hidden using these techniques in faked bad clusters and results are shown in table 2.

Table 2: Result of hidden data detection/recovery attempts for different hiding techniques

Hidden file	Technique	Result of foremost
A Microsoft document file	Normal, follow the sequence	Success in detect and recover
A Microsoft document file and a html file	Normal, follow the sequence	Success in detect and recover
A Microsoft document file and a html file	Reversed order of clusters	Success in detecting the files but unable to open them or files opened with meaningless data displayed
A Microsoft document file and a jpg file	Header removed	Fail to detect the files

Comeforth, an add-on of Sleuth Kit is more useful in recovering data if a file is not stored in sequence. Comeforth is similar to lazarus, where it divides file into block and run file command on every block (sleuth kit.org, n.d.). Users can then view each block and select blocks to be recovered as a file.

Keyword search can also be performed with hexedit, strings or other tools if part of the content of the hidden file is known.

```
strings /case1/image1 | grep keyword
```

VOLUME SLACK AND FILE SYSTEM SLACK

Volume slack is the unused space between the end of file system and end of the partition where the file system resides. File system slack is the unused space in the end of a file system that is not allocated to any cluster. This happens due to the partition size may not be the multiple of the cluster size (Carrier, 2005). For example, there is 10001 sectors in the partition, there first 10000 sectors are allocated to 2500 clusters with the cluster size of 4 sectors and the last sector left becomes file system slack.

The size of hidden data in volume slack is unlimited as suspects can simply change the size of volume slack to hide more data. The data that can be hid in file system slack, however, is depends on size of cluster. For example, for a file system with cluster size of 8 sectors, the maximum size of file system slack is 7 sectors.

Procedure to create test data

1) Sectors allocated to file system is modified in the $Boot file
2) Bits used for setting the allocation status of clusters in $Bitmap is reduced
3) Data is pasted to the volume slack and file system slack

Analysis techniques

Figure 3 shows the flow to analyse hidden data in volume slack.

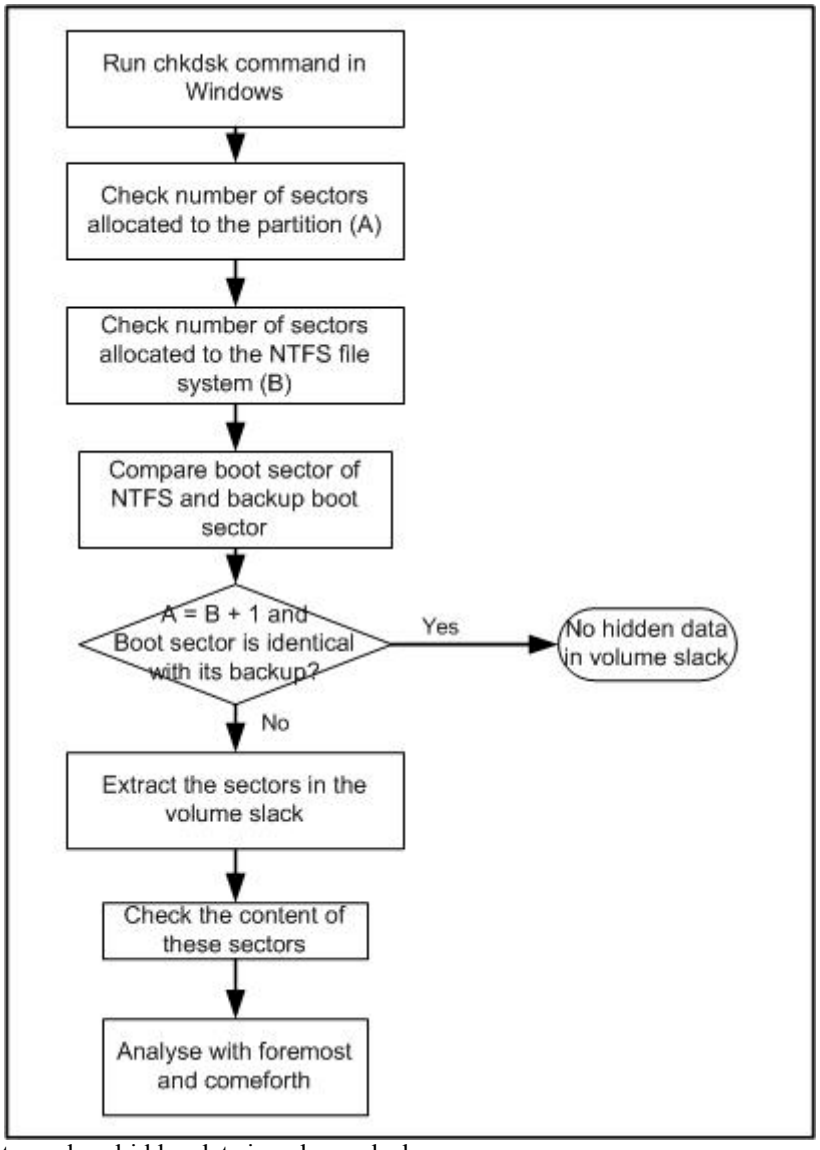

Figure 3: Flow to analyse hidden data in volume slack

Analysis should start with chkdsk command in Windows to check the file system. If suspects manipulate the file system manually and forget any of the needed steps, errors might be generated by chkdsk and give some indication about hidden data. For example, it is discovered that if suspects simply change the number of sectors allocated to the file system in the $Boot file without appropriate change in the $Bitmap file, message "Correcting errors in the Volume Bitmap" would appear by chkdsk command.

Check the number of sectors allocated to the partition with mmls command. In this example, mmls command returns 6136830 as total sectors in /case1/wholeimage.

```
./mmls /case1/wholeimage –t dos
```

Check the number of sectors allocated to NTFS file system in that partition with fsstat command. In this example, fsstat shows 6136782 sectors are allocated to the file system.

```
./fsstat /case1/image1 –f ntfs
```

Create md5 checksum of boot sector and backup boot sector

```
dd if=/case1/image1 bs=512 count=1 skip=6136829 of=/case1/backupbootsector
dd if=/case1/image1 bs=512 count=1 of=/case1/bootsector
```

```
md5sum /case1/backupbootsector
md5sum /case1/bootsector
```

For Windows NT 4.0, 2000 and XP, if there are A sectors in the partition, A-1 sectors is allocated to NTFS and the last sector is used to store the backup boot sector (Carrier, 2005). As a result, it is uncommon to have more than 1 sector of volume slack or the boot sector is not identical with the backup boot sector. If any of these situations happen, further analysis must be carried out on the volume slack.

Due to volume slack has no cluster number, dcat cannot be used to view its content. Dd can be used to extract volume slack and hex editor used to view it content. The volume slack is then analysed with foremost and comeforth similar to analysis of faked bad sector. Keyword search can also be performed.

```
dd if=/case1/image1 bs=512 count=48 skip=6136782 of=/case1/volumeslack
hexedit /case1/volumeslack
foremost –c /etc/foremost.conf –v –o /forensic/recover2 /case1/volumeslack
```

Figure 4 shows the flow to analyse hidden data in file system slack

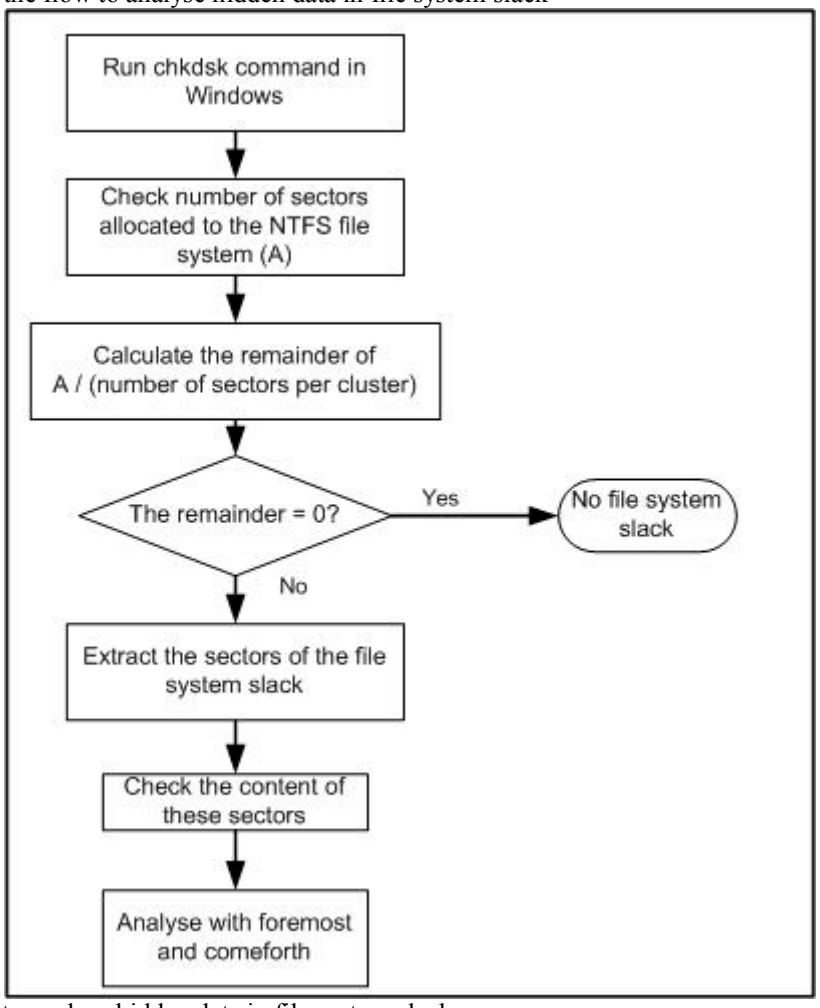

Figure 4: Flow to analyse hidden data in file system slack

Check the number of sectors allocated to the NTFS file system (A) and number of sectors per cluster (B)

```
./fsstat /case1/image1 –f ntfs
```

Calculate A/B. if the remainder is 0, there is no file system slack. Else, analyse the file system slack in the similar way to analysis of volume slack.

ADDITIONAL CLUSTERS ALLOCATED TO A FILE

This hiding technique hides data in the additional clusters allocated to a file. For example, there is a file with file size 10752 bytes, which would be allocated 3 clusters in a NTFS with cluster size of 8 sectors. Suspects can allocate extra clusters to this file and hide data in the additional clusters allocated.

With this technique, the size of hidden data is unlimited as suspects are free to allocate as many additional clusters as they wish. At the point of view of suspects, one disadvantage of this hiding technique is that whenever the file increases in size, the hidden data would be replaced and lost. As a result, stable files are preferable targets of this technique.

Procedure to create test data

1) Run list information of the file is modified to allocate more clusters to the file
2) Last VCN of the file is modified to appropriate value
3) Allocated size of the file is modified to appropriate value
4) Allocation status of the additional clusters is set to 1

Analysis techniques

Figure 5 shows the flow to analyse hidden data in additional clusters allocated to a file.

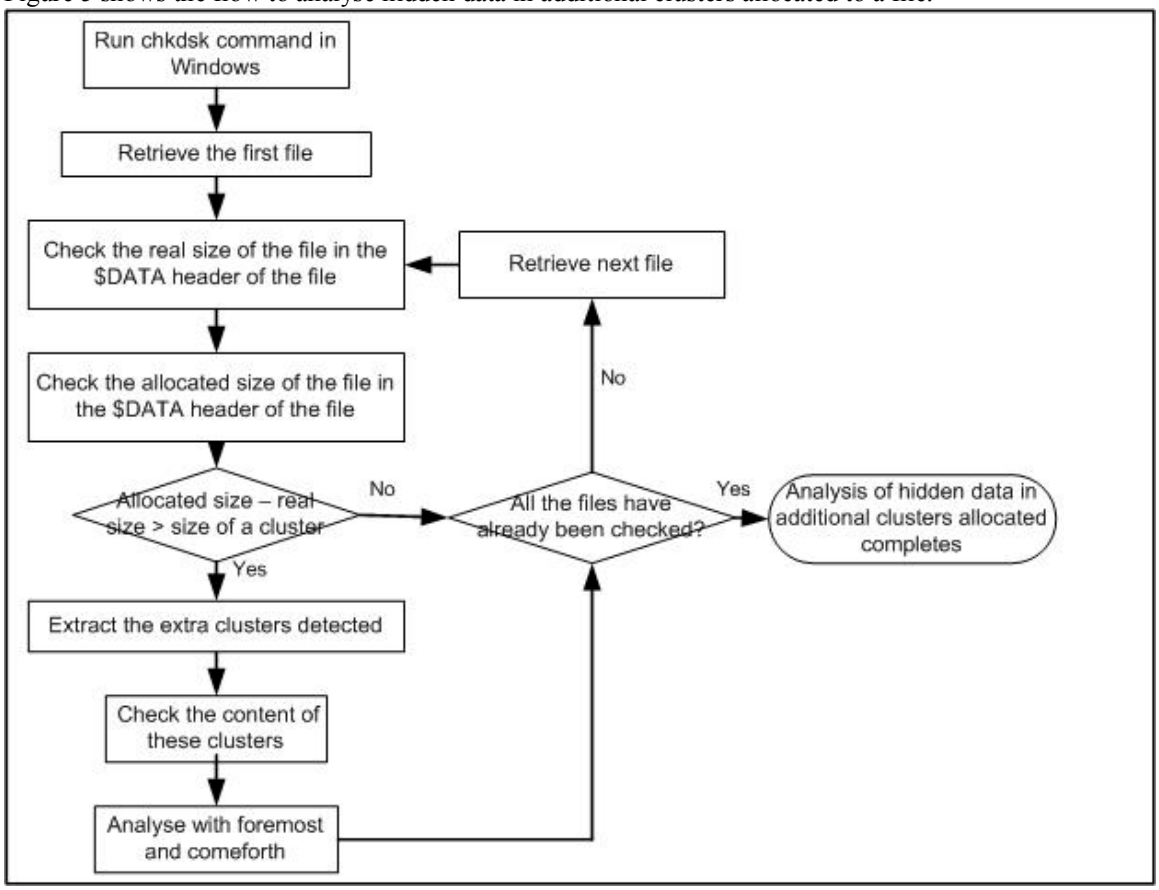

Figure 5: Flow to analyse hidden data in additional clusters allocated to a file

Run chkdsk to check the file system. During the testing, it is discovered that if a suspect forgets to complete any of the first 3 steps in the procedure, error will occur as shown in figure 6 and if he forgets to perform step 4 after first 3 steps, error message will be displayed as figure 7. This would be an indication of file system manipulation.

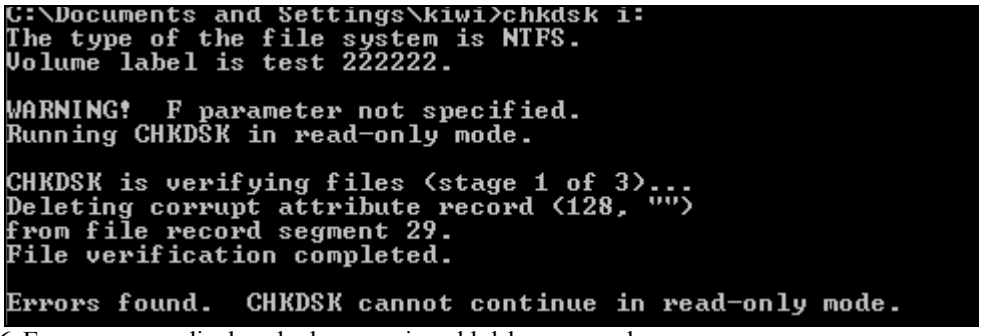

Figure 6: Error message displayed when running chkdsk command

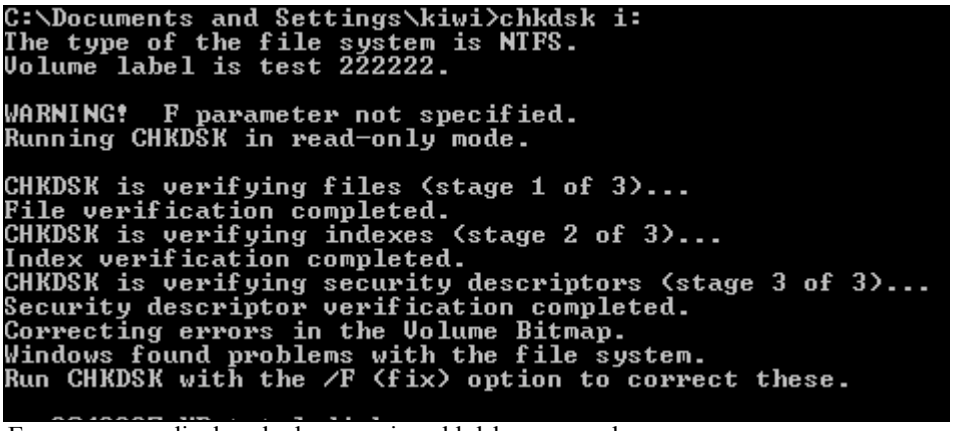

Figure 7: Error message displayed when running chkdsk command

Perform a recursive directory listing of the file system to identify all files

./fls –rFf ntfs /case1/image1

Analysis begins by retrieving the first file and comparing the allocated size and real size of the file. The fastest way to obtain these values is from the header of $DATA attribute of the file. However, only the real size is shown with istat command of Sleuth Kit but not the allocated size. As a result, you can either processing the MFT file record manually to get the value (which is time consuming), use other tools (DiskExplorer) or calculate the allocated size by multiplying cluster size and total clusters allocated to the file.

Number of allocated clusters and real size of the file can be retrieved with istat command:

./istat /case1/image1 29

Get the cluster size of the file system with fsstat

./fsstat –f ntfs /case1/image1

Calculate the additional space of the file

Allocated size = number of clusters * cluster size

Additional space = allocated size – real size

If additional space is larger than cluster size, additional/unnecessary clusters have been allocated to that file. This is uncommon and might contain hidden data. The additional clusters should be extracted and analyse with hex editor, foremost and comeforth similar to the analysis of faked bad clusters. Keyword

search can also be performed. The analysis is then continues with other files until all files have been analysed. This process is time consuming if performed manually. However, there is no specific tool that automates this process at the moment.

FILE SLACK

File slack is the unused space between the end of file and the end of cluster. File slack appears because cluster is the smallest unit of disk space allocation in NTFS and whole cluster is used even the file does not fill the whole cluster (Mallery, 2001). This empty space can be used to hide data (Chuvakin, 2002)

There are 2 types of file slack, which are RAM slack and drive slack. RAM slack spans from end of a file to the end of sector while drive slack spans from the start of next sector to the end of cluster (NTI, 2004). For example, a 600 bytes file is stored in a NTFS with 2048-bytes cluster and 512-bytes sector as shown in figure 8. RAM slack is from the end of file to the end of sector 2 and drive slack is composed of sector 3 and 4.

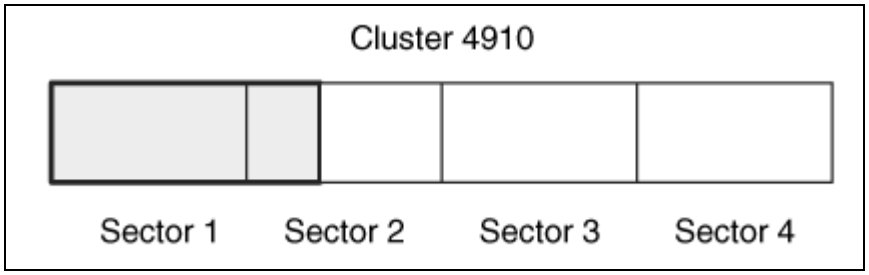

Figure 8: Slack space of a 600 bytes file in 4096 bytes cluster (Carrer, 2005)

Analysis of hidden data in slack space is depending on operating system as it is the operating system that decides how to handle file slack and not the file system. For example, Microsoft Windows pads RAM slack with 0 and ignores drive slack when storing a file (Carrier, 2005). So any non 0 bit in RAM slack of a file is suspicious and worth further analysis.

The size of data that can be hid in slack space of single file is based on the file size and cluster size (Paladion Network, 2004). The smaller the file size and the larger the cluster size, the more data can be hid. The size of hidden data in file slack is actually huge as suspects can hide data in slack space of multiple files in stead of just a single file. Hidden data in file slack have the danger of being erased/ replaced when the file size increases. As a result, stable files are preferable for this hiding technique.

Procedure to create test data

Test data is created by hiding data in RAM slack, drive slack and both.

1) Locate the suitable file/files with sufficient slack to hide data
2) Paste the data to slack space of the files

Analysis techniques

Figure 9 shows the flow to analyse hidden data in file slack. Analysis begins by checking the RAM slack of all files. If there is non 0 bit in the RAM slack, both RAM slack and drive slack of files are extracted for further analysis. Else, only drive slack is extracted.

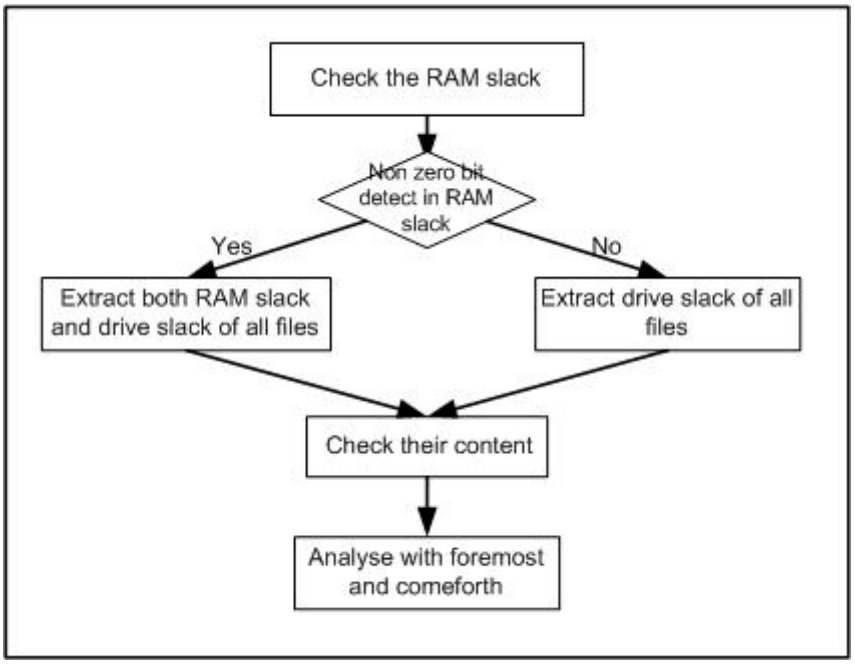

Figure 9: Flow to analyse hidden data in file slack

This example extracts the slack space of file with MFT number 28. To get the file size

./istat –f ntfs /case1/image1 28

Now calculate the RAM slack and drive slack. Int is function to get the integer value of a number. For example, both int(4.9) and int(4.1) return 4 as result.

file slack = allocated size – real size
drive slack = int(file slack / 512) * 512
RAM slack = file slack – drive slack

To extract the entire file with MFT number 28 including its file slack

./icat –sf ntfs /case1/image1 28 > /case1/file28

Let's assume the file size is 119875 and RAM slack is 445 bytes. To extract RAM slack

dd if=/case1/file28 of=/case1/file28RAMslack bs=1 skip=119875 count=445

To extract drive slack

dd if=/case1/file28 of=/case1/file28driveslack bs=512 skip=235 count=1

This process should be repeated to extract slack space of all files for analysis. Similar to the analysis of faked bad clusters, the extracted data is then analysed by using hex editor, foremost and comeforth. Keyword search can also be performed if you know the content of the hidden files. However, if a word is separated in 2 clusters and are not extracted correctly, the keyword search would fail. For example, a suspect hide data only in the first sector in the drive slack but extraction is done on the entire file slack, a word might be separated as shown in figure 10 and keyword search fails.

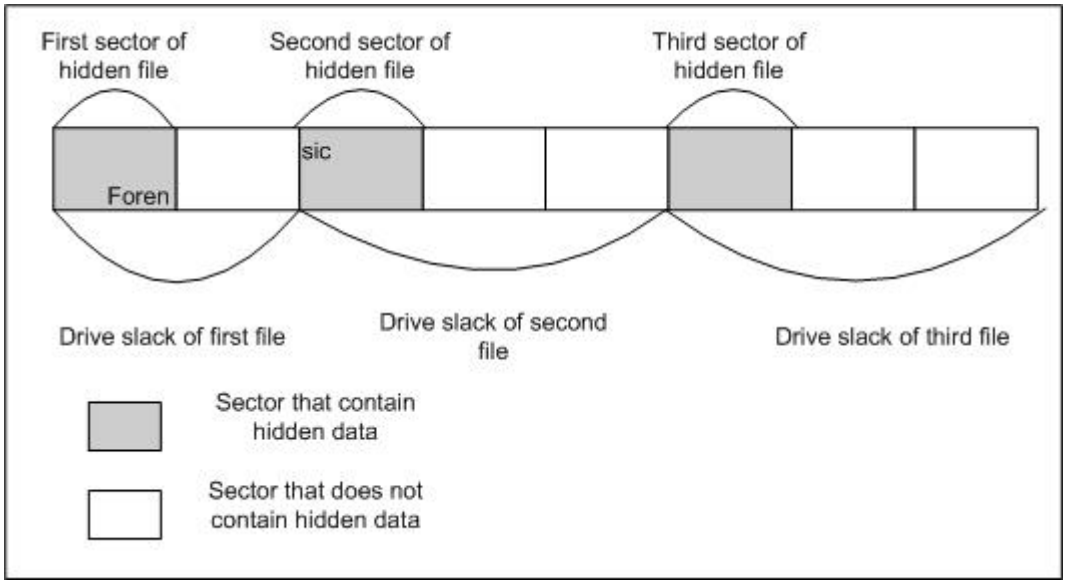

Figure 10: separation of a word in different sector

The most challenging step in detecting and recovering hidden data in file slack is to guess how data is hid. For example, suspects can hide data only in the first sector of the drive slack. A Suspect can also hide data only in the drive slack of the first file in each of the 5 directories created by him. Detection and recovery is hard, if not impossible without knowing this. It is recommended to search for data hiding tools in the system (Kruse & Heiser, 2001). The algorithm used to hid data is then analysed to decide on appropriate ways to extract file slack.

ADS (ALTERNATE DATA STREAM)

Whenever a MFT file record has more than 1 $DATA attribute, additional $DATA attribute is called ADS (Alternate Data Stream). ADS can be used to hide data in NTFS file system as ADS does not show up in directory listing and the file size of original file does not change (Cook, 2005). There are also legitimate uses of ADS. For example, ADS is used to store summary data and volume change tracking (Means, 2003)

The size of data that can be hidden in ADS is unlimited. One major difference between this data hiding technique and others is that ADS is relatively easy to create (Zadjmool, 2004). Other data hiding techniques discussed in this paper requires specific program or low level file system manipulation tools such as hex editor. ADS can be created easily with DOS command as shown in example below.

To hide slacker.exe in an ADS called hahaha of abcd.txt

```
type slacker.exe > h:\abcd.txt:hahaha
```

Procedure to create test data

ADS are created by issuing DOS command as shown above

Analysis techniques

Figure 11 shows the flow to analyse hidden data with ADS. Due to the popularity of using ADS to hide data, there are many well developed tools can be used to scan a drive for ADS such as lads and streams.

```
lads /sv h:
streams –s h:
```

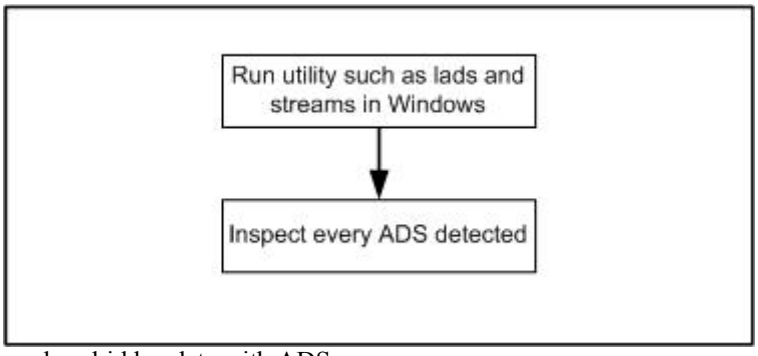

Figure 11: flow to analyse hidden data with ADS

Since there are legitimate uses of ADS, it cannot be assumed that every single ADS encountered is used to hide data. Each of these ADS should be examined to verify whether it is used for legitimate purpose or used to hide data. On the other hand, a suspect may also use the common ADS name used by legitimate program to avoid detection.

To get the MFT address and attribute identifier of ADS called "h:\abcd.txt:hahaha" shown in the result of lads or streams:

```
./fls –rf ntfs /case1/image1 | grep abcd.txt:hahaha
```

From the result, you get the MFT address, attribute type and attribute identifier. Let's say 59-128-4 is returned. Inspect the content of ADS as following:

```
./icat –f ntfs /case1/image1 59-128-4 > /case1/ADS1
file /case1/ADS1
hexedit /case1/ADS1
```

$DATA ATTRIBUTE IN DIRECTORY

$DATA attribute is usually used to store the content of a file or other specific information such as allocation status. It is a common attribute on normal file and some metadata file but not directory. Although $DATA attribute is unnecessary for a directory, validation checking with chkdsk does not return error when a directory contains a $DATA attribute. As a result, $DATA attribute in directory can be used to hide data (Carrier, 2005). In addition, alternate data streams can also be created on directories in stead of files to hide data. The size of data that can be hid with this technique is unlimited.

Procedure to create test data

1) A directory is created
2) $DATA attribute is inserted in the directory. The $DATA attribute is inserted before $INDEX_ROOT, $INDEX_ALLOCATION and $BITMAP because the type identifier of $DATA is smaller than these attributes.
3) Allocated size of MFT entry is modified to appropriate size
4) Attribute identifier of other attributes might need to be changed to avoid the new created $DATA attribute has the same identifier with other existing attributes.
5) Allocation status of clusters for this $DATA attribute is set to 1.
6) The file content is pasted to the clusters.

Analysis techniques

Figure 12 shows the flow to analyse hidden data in $DATA attribute of directory.

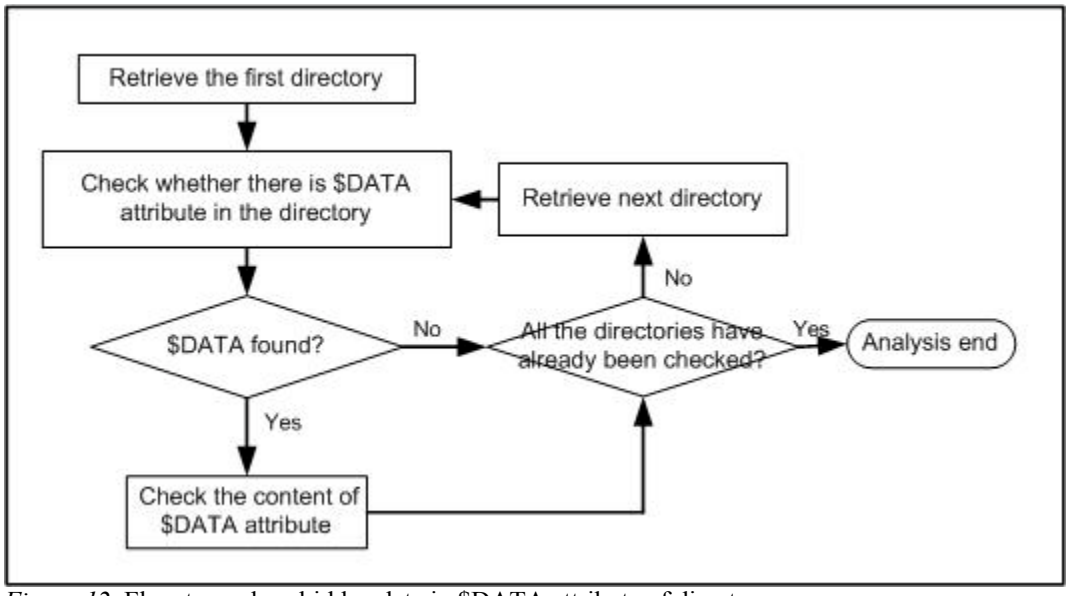

Figure 12: Flow to analyse hidden data in $DATA attribute of directory

List directory entries information of all directories

```
./fls –rDf ntfs /case1/image1 | grep 128
```

Examine the result for $DATA attributes. For example, the "r/r 40-128-3" is a $DATA attribute which might contain hidden data.

```
root@linux:~/sleuthkit-2.02/bin# ./fls -rDf ntfs /dev/sda2 | grep 128
r/r 40-128-3:   finally
+ r/r 40-128-3: .
root@linux:~/sleuthkit-2.02/bin#
```

Figure 13: Result of running fls command to check for $Data attributes in directories

Check the content of the $DATA attribute

```
./icat –f ntfs /case1/image1 40-128-3 > /case1/dataindirectory1
file /case1/dataindirectory1
hexedit /case1/dataindirectory1
```

Analysis techniques used to detect ADS in files also detect ADS in directory at the same time. The process has been explained in the previous section and is not repeated here.

HIDDEN DATA IN $BOOT FILE OR BOOT RECORD

Everything in NTFS is a file includes boot record, which is stored in a metadata file called $Boot. In NTFS, this is the only file with fix location, which always starts at the first cluster of the file system. Windows allocates 16 sectors to this file but in usual only half of these sectors contain non zero bit (Carrier, 2005).

According to the documentation about NTFS at Linux NTFS project (Sourceforge, n.d.), there are certain unused bytes in the boot sector. However, Windows would not mount the file system if there is any none 0 values in these unused bytes (Carrier, 2005). As a result, this cannot be used to hide data.

In $Boot file of NTFS file system, data usually is hid at the bytes allocated for boot code. Boot code is essential for bootable file system to locate needed files to boot up Windows (Carrier, 2005). For non bootable file system, it usually used to store the error message that is displayed if users try to boot from this partition.

The size of data that can be hidden in the boot code is unlimited as suspects can simply allocated more clusters to $DATA attribute of $Boot file to hide data.

Procedure to create test data

1) Run list of $Data attribute of $Boot is modified
2) Last VCN value of $Data attribute of $Boot is modified
3) Real size, allocated size and compressed size of $Data attribute of $Boot is modified
4) Allocation status of the additional clusters allocated to $Boot is set to 1
5) Hidden data is pasted to the selected clusters.

Analysis techniques

Figure 14 show the flow to analyse hidden data in $Boot file and backup boot sector.

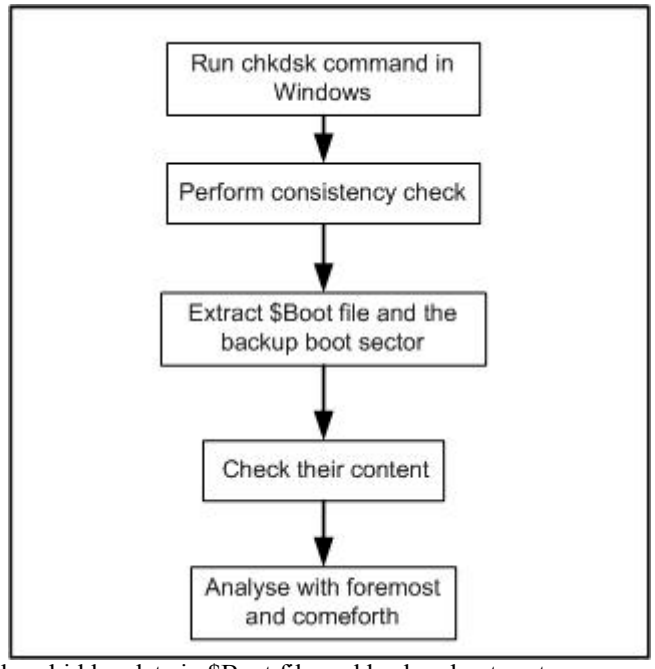

Figure 14: Flow to analyse hidden data in $Boot file and backup boot sector

Chkdsk might return error if suspects forget to complete one of the needed step. For example, if any of the first 3 steps is not completed, error message as show in figure 15 will occur. If suspects forget to perform step 4, error message as shown in figure 16 will occur.

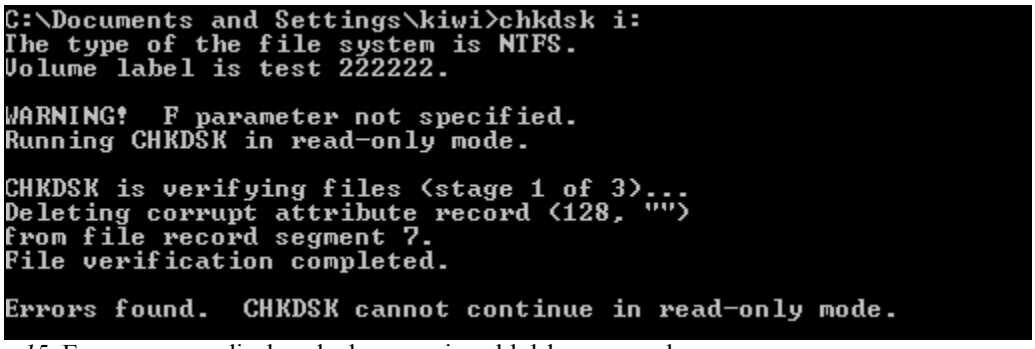

Figure 15: Error message displayed when running chkdsk command

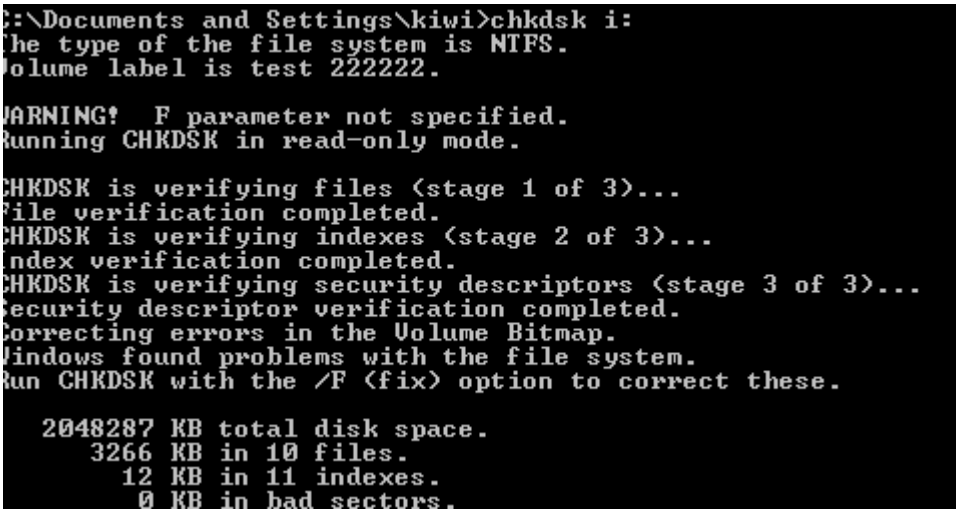

Figure 16: Error message displayed when running chkdsk command

Analysis of hidden data in $Boot file should start by comparing boot sector and backup boot sector. Let's assume there are 6136830 sectors in the partition.

```
dd if=/case1/image1 bs=512 count=1 skip=6136829 of=/case1/backupbootsector
dd if=/case1/image1 bs=512 count=1 of=/case1/bootsector
md5sum /case1/backupbootsector
md5sum /case1/bootsector
```

However, this consistency check would only give indication of file system manipulation if the checksums differ from each other. If they are the same, the possibilities are that suspects copy the modified boot sector to its backup to avoid detection or the file system does not contain hidden data. The extracted content is then analysed with hex editor, foremost and comeforth similar to analysis of hidden data in faked bad clusters. Keyword search can also be performed.

UNCOMMON/ INEFFECTIVE DATA HIDING TECHNIQUES

From the perspective of suspects, a good data hiding techniques in NTFS file system should meet the goals of security and capacity as following (Provos & Honeyman, n.d.).

1) Normal system check with utility such as chkdsk does not return any error
2) Hidden data would not be overwritten or the possibility of data being overwritten is low
3) Normal user would not notice the hidden data
4) The technique can store reasonable amount of hidden data

Table 3 shows some uncommon and ineffective techniques which does not satisfy characteristic discussed above.

Table 3: Description of certain uncommon and ineffective data hiding techniques

Data hiding techniques	Weaknesses
Set the allocation bit of certain unallocated clusters to 1 and hide data in these clusters	Simple chkdsk validation would detect this
Hide data in unallocated clusters	High possibility of hidden data being erased unless certain countermeasures are implemented such as set the hard disk to read only access
Hide data in unallocated MFT entries	High possibility of hidden data being erased unless certain countermeasures are implemented such as set the hard disk to read only access
Unused space in a MFT file record	Hidden data is erased if more attributes is added to the MFT file record or the size of an attribute increases.

However, there are certain environments where these hiding techniques might be effective. Let's take data hiding in unallocated clusters as example. If there is hard disk write protector that protects the hard disk or rootkit installed in the system that will never write to these unallocated location, the hidden data would not be erased. As a result, a through check on the system and the physical environment are important and might give some clue about the data hiding techniques used.

CONSISTENCY CHECK ON NTFS

Certain consistency check should be performed on NTFS before any analysis starts. Chkdsk should be performed and if it returns any error, there might be file system manipulation on it that causes the file system in an unstable state.

Default cluster size of NTFS is depends on the size of file system (Svensson, 2005). Usually a normal user would not modify the default cluster size. However, suspects might modify it so that more hidden data can be stored. For example, the larger the cluster size, the more hidden data can be stored in the file slack of a file. A NTFS with cluster size that does not match its file system size is suspicious.

System should be searched for data hiding tools. Attention should also be paid on both digital and physical files that contain some weird numbers that might be the clusters address where hidden data is stored.

CONCLUSION

In general, analysis of hidden data in NTFS file system is divided into 2 phases. The first phase is to identify whether there is hidden data by searching for anomaly. For example, in the analysis of hidden data in faked bad clusters, it is abnormal to have an operating system to detect bad sectors before a hard disk does. This is suspicious and worth further analysis.

The second phase is to recover the hidden file. Due to hidden data is usually stored in the file system without any structure or metadata, it is hard to recover them. Recovery is particularly challenging if suspects store data in random order and remove the file signature.

There is no specific forensic analysis tool that checks for hidden data in NTFS file system except tools that check for alternate data stream. While the analysis techniques that already discussed might be able to detect/recover the hidden data, it is time consuming without automated tools.

The data hiding techniques that have been discussed in this paper are just a fraction of possible way to hide data. There are always new techniques to hide data and the art of data hiding is depends on suspects' creativity. One of the difficulties of analysing NTFS file system for hidden data is that it is so flexible and can support many options (Carrier, 2005). As a result, there are many possible ways to hide data. In addition, without a published specification, which value combinations are valid and which are not is unknown (Carrier, 2005).

REFERENCE

Carrier B. (2005, March 17). *File System Forensic Analysis.* Addison Wesley Professional.

Carvey B. (2004a, July 21). *Windows Forensics and Incident Recovery*. Addison Wesley.

Carvey B. (2004b). *Data Hiding on a Live System*. Retrieved September 23, 2005 from http://www.blackhat.com/presentations/win-usa-04/bh-win-04-carvey.ppt

Cook R. (2005, September 16). *Alternate Data Streams: Threat or Menace?* Retrieved September 20, 2005 from http://www.informit.com/articles/printerfriendly.asp?p=413685

Chuvakin A. (2002, March 10). *Linux Data Hiding and Recovery.* Retrieved September 23, 2005 from http://www.linuxsecurity.com/content/view/117638/49/

Microsoft (n.d.). *The NTFS File System*. Retrieved September 13, 2005 from Microsoft.com: http://www.microsoft.com/resources/documentation/Windows/2000/server/reskit/en-us/Default.asp?url=/resources/documentation/Windows/2000/server/reskit/en-us/core/fncc_fil_khzt.asp

Kruse W. G. & Heiser J. G. (2001, September 26). *Computer Forensics : Incident Response Essentials*. Addison-Wesley Professional

Li K. (2003). *A Guide to Programming Intel IA32 PC Architecture*. Retrieved September 16, 2005 from http://www.cs.princeton.edu/courses/archive/fall04/cos318/docs/pc-arch.html

Mallery J. R. (2001, July 16). *Secure File Deletion, Fact or Fiction*? Retrieved September 20, 2004 from http://www.sans.org/rr/papers/27/631.pdf

Means R. L. (2003). *Alternate Data Streams: Out of the Shadows and into the Light*. Retrieved September 20, 2005 from http://www.giac.com/practical/GCWN/Ryan_Means_GCWN.pdf

Mikhailov D. (n.d.). *NTFS file system.* Retrieved September 13, 2005 from http://www.digit-life.com/articles/ntfs/

NTI (2004, January 6). *File Slack Defined*. Retrieved September 20, 2005 from http://www.forensics-intl.com/def6.html

Paladion Network (2004). *Incident handling/ Forensic FAQs.* Retrieved September 23, 2005 from http://www.paladion.net/papers/ihfaq.htm#hidden_data

Provos N. & Honeyman P. (n.d.). *Detecting Steganographic Content on the Internet.* Retrieved October 4, 2005 from http://niels.xtdnet.nl/papers/detecting.pdf

sleuth kit.org (n.d.). *README*. Retrieved September 23, 2005 from http://www.sleuthkit.org/sleuthkit/download.php

Solomon D. A. & Russinovich M. E. (2000, September 1). *Inside Microsoft Windows 2000, Third Edition.* Microsoft Press

Sourceforge (n.d.). *Foremost.* Retrieved September 16, 2005 from http://foremost.sourceforge.net/

Sourceforge (n.d.). *File - $Boot (7)*. Retrieved September 22, 2005 from http://linux-ntfs.sourceforge.net/ntfs/files/boot.html

Storagereview (n.d.). *Reference Guide - Hard Disk Drives: Reference Guide - Hard Disk Drives*. Retrieved September 13, 2005 from http://www.storagereview.com/guide2000/ref/hdd/geom/formatDefect.html

Svensson A (2005, April). *Computer Forensics Applied to Windows NTFS Computers.* Retrieved September 23, 2005 from http://www.dsv.su.se/research/seclab/pages/pdf-files/2005-x-268.pdf

Zadjmool R. (2004, March 24). *Hidden Threat: Alternate Data Streams.* Retrieved September 20, 2005 from http://www.windowsecurity.com/articles/Alternate_Data_Streams.html

APPENDIX A: DETAIL PROCESS OF HOW DATA IS HID IN FAKED BAD CLUSTERS

Tool used in this example is Runtime's DiskExplorer for NTFS v2.31. While using this tool to edit the NTFS might seem awkward and too complicated for script kiddies level suspects, it is just a matter of time for a tool that can automate this task to occur. This tool is used here to just to show how NTFS file system can be manipulated to hide data and prepare a test file system for analysis

1) Locate where $DATA of $Bitmap is stored. As shown in Figure 17, it is stored from x0005DA54C and it is allocated 24 clusters.
2) Inspect the clusters that are allocated to $Bitmap.

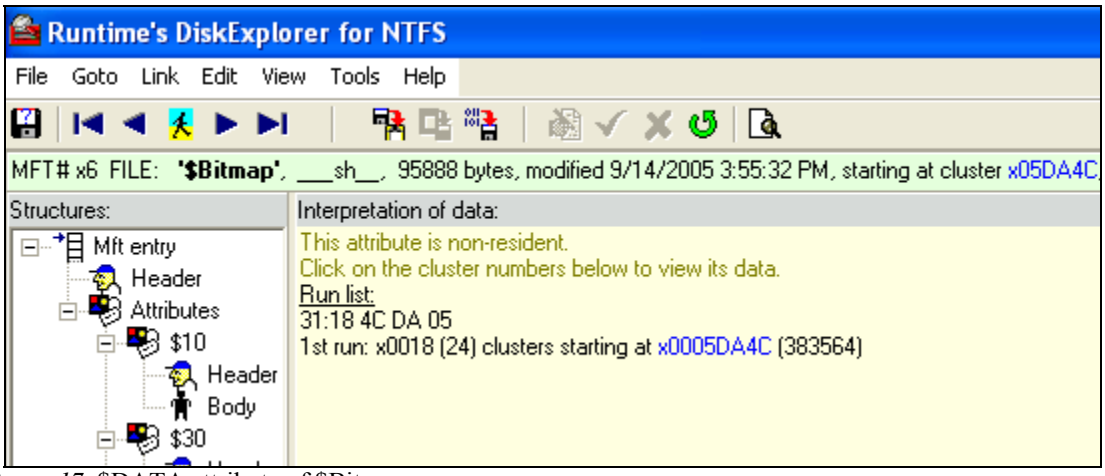

Figure 17: $DATA attribute of $Bitmap

3) Any cluster that is unallocated can be used. In this example, 12 clusters (from 383624 to 383635) would be marked as bad clusters to store hidden data as shown in figure 18. Calculation of the cluster number represented by a bit in $Bitmap is shown below:

   Cluster number = ((A – B) * C * D) + (E * F * D) + (G * D) + H – 1

   A = sector address of current sector
   B = starting sector for $Data attribute of $Bitmap
   C = number of bytes in a sector
   D = number of bits per byte
   E = number of rows before the targeted bit
   F = number of bytes per row
   G = number of bytes before the byte for that bit
   H = location of the bit in that particular byte. For example, H=1 for 01 and H=4 for 08

   For example, calculation of the bit that represents cluster address 383524 in $Bitmap is shown below:

   Starting sector for $DATA attribute of $Bitmap: 029FE8D6
   This sector: 029FE933
   Cluster number = ((029FE933 – 029FE8D6) * 512 * 8) + (21 * 16 * 8) + (1 * 8) + 1 - 1
   = 380928 + 2688 + 8
   = 383624

4) Set cluster 383624-383635 as bad clusters in the run list of $BAD attribute as shown in figure 19. Note that this is an Intel PC and it uses little-endian-ordering, which put the least significant byte of a number in the first storage byte (Li, 2003). For example, 383624 (05dA88) is stored as 88DA05
5) Modify the size of the $Bad attribute to 0x58
6) Modify the size of the MFT to 0x180

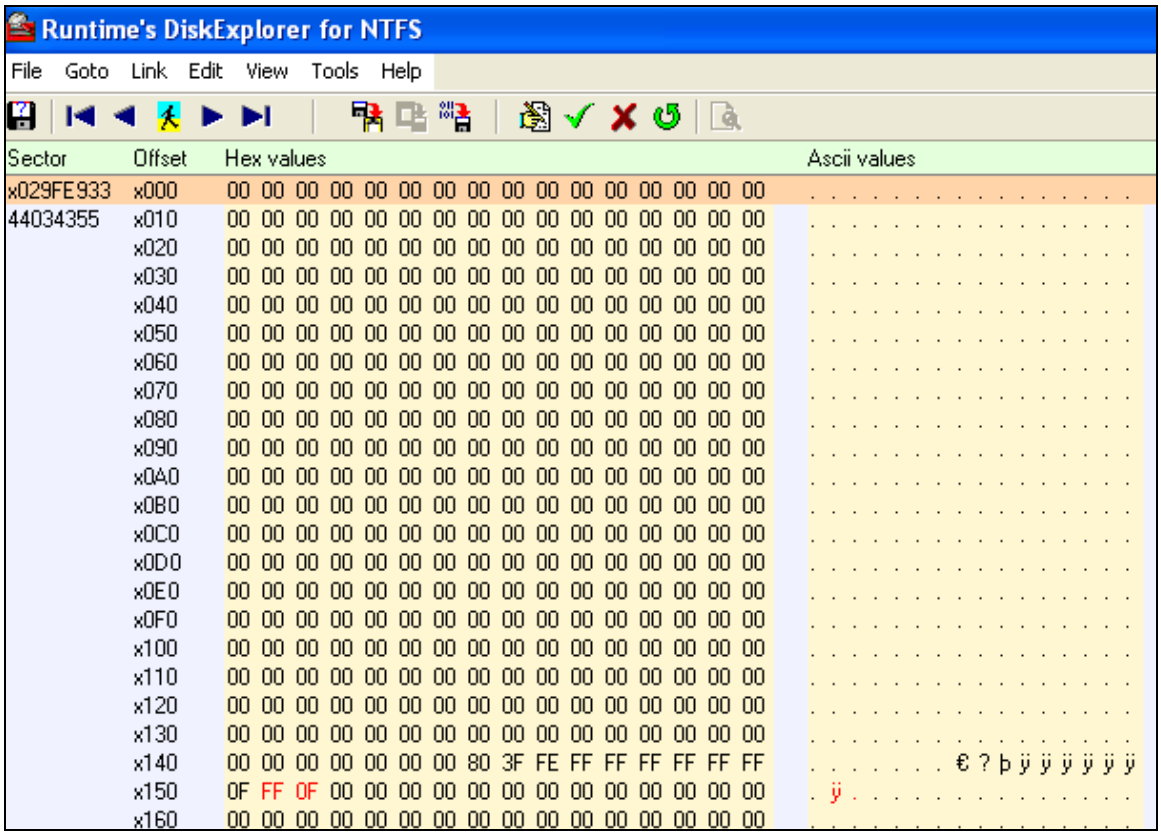

Figure 18: Allocation status of faked bad clusters is set to 1

7) Hide data into that cluster 383624 using the “paste from file command” in DiskExplorer
8) Run chkdsk in Windows XP to ensure there is no structure error happen after modification.
9) The result is tested using Sleuth Kit before and after data is hidden. Before data is hide in the faked bad cluster, run dstat to ensure the target clusters are unallocated and istat to ensure no cluster is assigned to $BadClus. Example:

    dstat /dev/sda2 –f ntfs 383624, repeat this for all targeted clusters
    istat /dev/sda2 –f ntfs 8

10) After the data is hidden, perform the same commands again to ensure the clusters are allocated to $BadClus properly. Besides dstat and istat command, dcat command is run to check to content of hidden data. It is also recommended to try to extract the hidden clusters with dd command in Linux and try to open the file. Example:

    dstat /dev/sda2 –f ntfs 383624, repeat this for all targeted clusters
    istat /dev/sda2 –f ntfs 8

    dcat /dev/sda2 –af ntfs 383624, repeat this for other clusters
    dd if=/dev/sda2 bs=4096 skip=383624 count=12 of=/mnt/usb/recoverfile

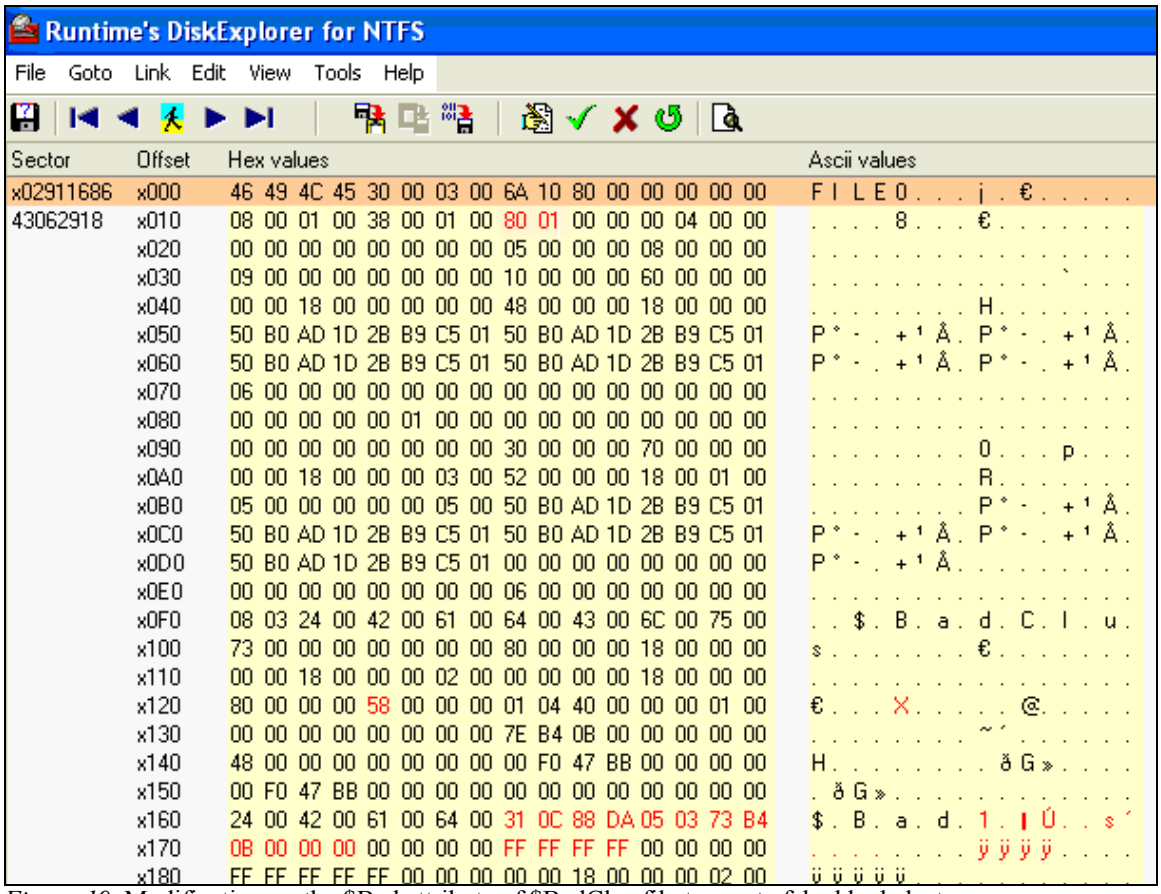

Figure 19: Modification on the $Bad attribute of $BadClus file to create faked bad clusters