



University of  
Western Sydney

Bringing knowledge to life

# Data Hiding in the NTFS File System

**Ewa Huebner**

**University of Western Sydney**

**Australia**

**[e.huebner@scm.uws.edu.au](mailto:e.huebner@scm.uws.edu.au)**

# Introduction

→ **This presentation is based on the joint work with Derek Bem and Cheong Kai Wee**

→ **Published in**

*Digital Investigation The International Journal of Digital Forensics & Incident Response, December 2006*

# Outline

- **Examination of the methods of hiding data in the NTFS file system**
- **the analysis of techniques**
  - to detect and recover data hidden using each of these methods.
- **focus on sophisticated data hiding**
  - the goal is to prevent detection by forensic analysis.
  - obvious data hiding techniques, for example setting the hidden attribute of a file, will not be included.
- **Hidden data can be further obfuscated**
  - by file system independent approaches like data encryption and steganography.
  - This work is only concerned with the methods which are made possible by the structure of the NTFS file system.

# Criteria

## → **Methods analysed fulfil the following criteria:**

- Standard file system check with a utility such as chkdsk should not return any errors.
- Hidden data will not be overwritten or the possibility of data being overwritten will be low.
- Normal user will not notice the hidden data.
- A reasonable amount of hidden data can be stored.

# Categories of methods

- **Specific methods based on unique NTFS data structures**
  - The focus of this work
  
- **Generic slack space based methods in NTFS**
  - Discussed for completeness
  - applicable to other file systems

# Methodology And Tools

## → Tools for hiding data

- Runtime's DiskExplorer for NTFS v2.31
- except for Alternate Data Streams ADS

## → Tools for restoring data

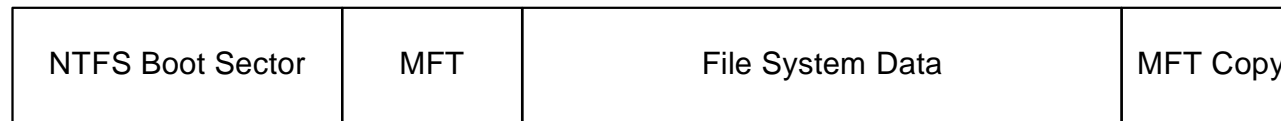
- Windows XP chkdsk and Fsutil
- Windows OEM Support Tools Nfi
- Sleuth Kit 2.02
- Foremost 0.69
- comeforth 1.00
- Also: dd, hexedit and strings.

# NTFS Background

## → In the NTFS file system (and Windows OS)

- every object is a file
- and as such it inherits all the characteristics of a file.
- This includes file system metadata which defines the structure of the file system itself.

## → NTFS structure – 4 logical blocks



- NTFS Boot Sector – layout of the volume, the file system structures and the boot code,
- MFT – Master File Table,
- File System Data – storage for data that is not contained within the Master File Table,
- MFT Copy – Master File Table copy.

# Master File Table (MFT)

## → an array of records

- Every file and every directory has at least one entry in MFT, including the MFT itself.
- Each entry in MFT is called a file record and has a default fixed size of 1024 bytes
- The first 42 bytes in a file record are reserved for the MFT entry header,
- and the remaining bytes are used to store the so called attributes

## → MFT entry with a resident record

Standard Information	File or Directory Name	Data or Index	Unused Space
----------------------	------------------------	---------------	--------------

# Attributes

- **An attribute is a small data structure with a specific purpose,**
  - for example \$STANDARD\_INFORMATION, \$FILE\_NAME or \$DATA
  - Two attributes, \$STANDARD\_INFORMATION and \$FILE\_NAME are present in every file record, independently of the file type.
  - Some attributes are specific to file types, for example \$DATA for data files and \$INDEX\_ROOT for directory files.
  
- **IMPORTANT: apart from the attributes required by the system, a file may have any other attribute, even if it is not used by the system.**

# Clusters

## → A data unit in NTFS is called a cluster

- It is the smallest disk space allocation unit.
- Every cluster in NTFS has a Logical Cluster Number (LCN), starting with 0 for the first cluster of the file system
- Clusters which belong to a file are also assigned a Virtual Cluster Number (VCN).
- For example,
  - if a file consists of six clusters, the first cluster of the file will have VCN 0 and the last cluster – VCN 5.
- The cluster runs for non-resident attributes provide mapping between VCNs and LCNs.

# Metadata files

## → Metadata files

- describe the structure of the file system itself.
- Microsoft defined 16 standard MFT records (file numbers 0 – 15), (some currently unused - reserved for future use).
- Additional metadata files are stored in the extensions metadata directory
- File numbers for extensions metadata files start with 24, and all following numbers are available for user files.
- The range of file numbers from 16 to 23 is not allocated,
  - and it is reasonable to assume that Microsoft reserves these numbers for some future use.

# Metadata files in NTFS

File no	Metadata file	Description
0	\$MFT	MFT record
1	\$MFTMirr	Partial backup of MFT
2	\$LogFile	Transaction logging file
3	\$Volume	Volume information such as label, identifier and version
4	\$AttrDef	Attribute definition
5	.(dot)	Root directory of file system
6	\$Bitmap	Allocation status of all clusters
7	\$Boot	Boot record
8	\$BadClus	List of bad clusters
9	\$Secure	Security and access control information
10	\$Upcase	Converts lowercase characters to matching Unicode uppercase characters
11	\$Extend	Optional extensions directory
12-15	unnamed	Reserved for future use
24 -	\$Extend\Quota	Quota file
	\$Extend\ObjId	Object identifier file
	\$Extend\Reparse	Reparse point file
	\$Extend\UsnJrnl	Change journal file

# NTFS Specific Data Hiding Methods

- **any object in NTFS is a file, and as such can have any file attribute.**
  - Some attributes, for example \$DATA, \$INDEX\_ROOT or \$INDEX\_ALLOCATION may legitimately appear multiple times in a single file, and in such case they are given unique names.
  - NTFS is **not sensitive** to a file including attributes which are unnecessary.
  - This feature can be exploited for hiding data without affecting correct functioning of the system
- **Except for \$DATA attribute, most of the other attributes have specific formats so that they can serve their intended purpose.**
  - Any manipulation of the content of these attributes may affect the integrity of the system, so they are not suitable for hiding data.

# Exceptions

- **When an attribute is provided purely for backward compatibility and no longer used,**
  - for example \$SECURITY\_DESCRIPTOR or extended attributes \$EA and \$EA\_INFORMATION.
- **In theory**
  - it could be possible to use such attributes for hiding data, assuming that a software tool is available for adding these attributes to files.
- **In practice**
  - such approach does not appear effective, especially that there is no certainty that the backward compatibility will be maintained in future versions of NTFS.

## \$DATA attribute

→ **The most effective hiding methods are based on the \$DATA attribute of files.**

- This attribute is the only one without an implied format, so its content can be arbitrary,
- it can have any size without raising suspicion,
- and it is reasonable to assume that it will always remain a feature of NTFS.

## Metadata files based methods

- **All attributes of these files are well defined,**
  - even the \$DATA attribute, where it is present, has a defined internal format to support the NTFS operations.
  - Exception: \$BadClus file
- **potential for hiding some data in the metadata files without affecting the functionality of the system**
  - The difficulty: no guarantee that NTFS will remain insensitive to the additions in its future versions
  - Specialised software tools are required to manipulate the metadata files.
  - The very presence of such tools in the file system is suspicious

# Hiding data in \$BadClus file

## → **Bad clusters in modern systems**

→ hard disk controllers handle bad sectors themselves without any involvement of the operating system.

- slipping - modifying Logical Block Number (LBN) to physical mapping to skip the defective sector
- remapping - reallocating LBN from defective area to a spare sector
- Also the volume managers included with Windows are capable of remapping bad sectors

## → **Older hard disks that do not have this capability,**

→ operating systems and file systems have to retain the ability to detect and mark defective sectors and clusters as damaged.

→ This ability may be used to exclude undamaged clusters from the normal file system activities, and use them to hide data in any system.

## Hiding data in \$BadClus file

### → In NTFS,

- bad clusters are marked in the metadata file \$BadClus, which is the standard MFT entry 8.
- \$BadClus is a sparse file with the size set to the size of the entire file system.
- detected bad clusters are allocated to this file
- The size of hidden data = the capacity of the entire file system.
- Any number of clusters can be allocated to \$BadClus and used to store data.

# Hiding Data in \$DATA Attribute

- **Most of the metadata files contain the \$DATA attribute**
  - except the directories and the extension metadata files
  - Except for \$BadClus, the content used in NTFS operation (format depends on the role the metadata file fulfils in the file system)
  - possible to append some hidden data to these \$DATA attributes without disturbing the data already present.
  - \$DATA attribute is both read and written by NTFS, so the appended data could be overwritten
- **some of the metadata files are static**
  - for example in \$AttrDef, \$Bitmap, \$Boot, and \$UpCase.
  - more clusters can be allocated to the \$DATA attribute of these metadata files to hide data.

## NTFS \$Boot file

→ **In NTFS the boot record is stored in a metadata file called \$Boot.**

→ This is the only file with a fixed location

→ it always starts at the first cluster of the file system.

→ Windows allocates 16 sectors to this file but typically only half of these sectors contain non-zero bits

→ **unused bytes in the boot sector**

→ Windows will not mount the file system if there are any non-zero values in these unused bytes

→ As a result, this space cannot be used to hide data.

## Hiding Data in \$Boot file

- **the bytes allocated to boot code in the \$Boot file of NTFS file system can be used to hide data.**
  - Boot code is only essential for a bootable file system to locate files required to boot up Windows
  - For a non-bootable file system, it is used to store the error message that is displayed if an attempt is made to boot from this partition.
  - The size of data that can be hidden in this manner is limited by the number of non-zero bytes in the \$Boot file.

## Data files based methods

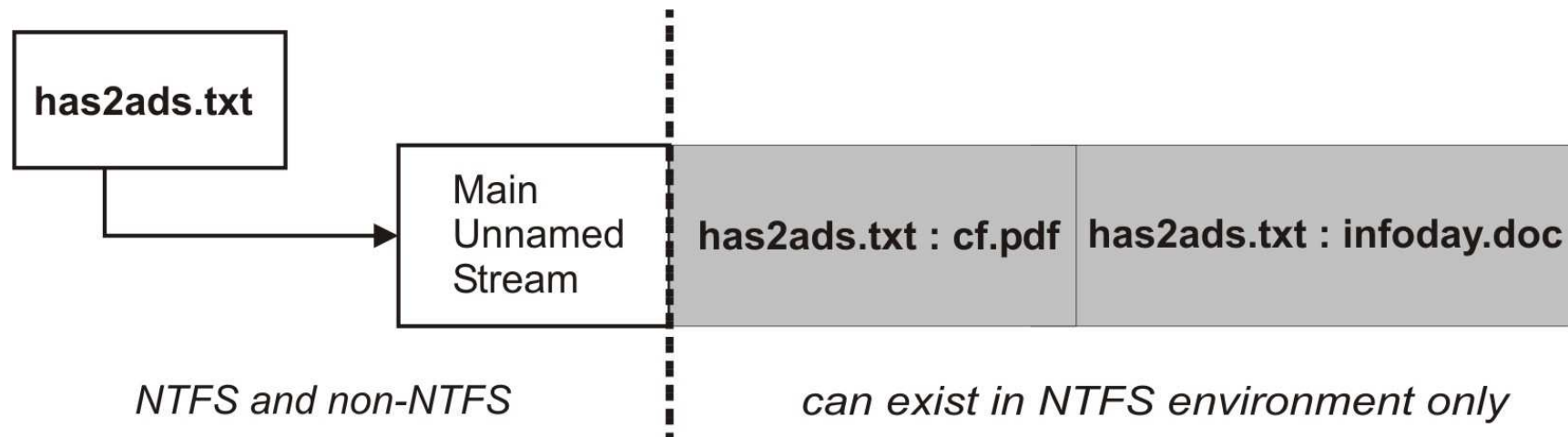
→ **each file and each directory in NTFS has at least one entry (a file record) in Master File Table.**

- A typical GUI interface for the file system assumes that each data file has a single unnamed \$DATA attribute and each directory - a single \$INDEX\_ROOT attribute.
- It is the \$DATA attribute which provides most scope for hiding data, both in terms of capacity and longevity of hidden data.
- We will examine three methods of data hiding in \$DATA attribute:
  - in alternate data streams in data files,
  - in alternate data streams in directories
  - and in extension of the existing unnamed \$DATA attribute

# Alternate Data Streams (ADSs)

- **Alternate Data Streams (ADSs) are a unique feature of NTFS file systems introduced with Windows NT 3.1 (early 1990s)**
  - compatibility between Windows NT servers and Macintosh clients which use Hierarchical File System (HFS).
- **In NTFS an MFT file record may have more than one \$DATA attribute**
  - the additional **named** \$DATA attributes are alternate data streams.
  - ADSs can be used to hide data in NTFS file system as most of the system utilities only examine the first unnamed \$DATA attribute.

# ADS visibility in NTFS and non-NTFS environments



ADSs can be created easily with the DOS command “type”:

```
type cf.pdf > h:\has2ads.txt:cf.pdf
```

```
type infoday.pdf > h:\has2ads.txt:infoday.pdf
```

## \$DATA Attribute in a Directory

- **The \$DATA is a common attribute for all data files and some metadata files but not directories.**
  - \$DATA attribute is unnecessary for a directory
  - validation checking with chkdsk or any other standard utility does **not return an error** when a directory contains a \$DATA attribute.
  - As a result, the \$DATA attribute in a directory can be used to hide data
  - Alternate data streams (named \$DATA attributes) can also be created with directories, not just data files, to hide data.
  - The size of data that can be hidden with this technique is only limited by the capacity of the file system.

## Data hiding in added clusters

### → **Technique is based on the ability to allocate additional clusters to an NTFS file.**

- If there is an existing file with some clusters already allocated by NTFS, more clusters can be allocated manually, and data can be hidden in these clusters.
- the size of hidden data is only limited by the capacity of the file system as owners are free to allocate as many additional clusters as they wish.
- **Disadvantage:** whenever the original file increases in size, the hidden data could be overwritten and lost.
- Stable files are preferable targets
  - This is not a real hindrance, as the original file is owned by the hiding party, and other users can be denied write access to this file.

# Slack Space Based Hiding Methods

## → Slack space

→ all areas on disk surface which cannot be utilised by the file system because of discrete nature of space allocation.

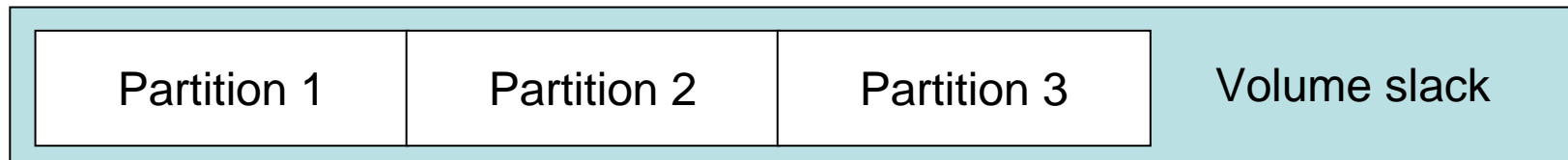
## → Existence of slack space is a characteristic of all file systems, not just NTFS.

→ The discussion of data hiding methods in NTFS would not be complete without discussing slack space

→ We will highlight those aspects of the methods which are specific to NTFS.

# Volume Slack

- **The unused space between the end of the volume and the end of the partition**
  - The size of the hidden data in volume slack is only limited by the space on the hard disk available for a partition
  - the size of partition can be changed in relation to the size of volume to hide more data



# File System Slack

- **File system slack is the unused space at the end of the file system that is not allocated to any cluster.**
  - This happens because the partition size may not be the multiple of the cluster size
  - For example, if there are 10001 sectors in the partition, the first 10000 sectors are allocated to 2500 clusters with the cluster size of 4 sectors and the last sector becomes file system slack.
  - The size of data that can be hidden in file system slack depends on the size of a cluster.
  - For example, for a standard NTFS cluster size of 8 sectors, the maximum size of the file system slack is 7 sectors.

# File Slack

→ **File slack is the unused space between the end of a file and the end of the last allocated cluster.**

→ The file slack appears because a cluster is the smallest unit of disk space allocation in NTFS

→ a whole cluster has to be used even if the file does not fill the whole cluster

→ **There are two types of file slack,**

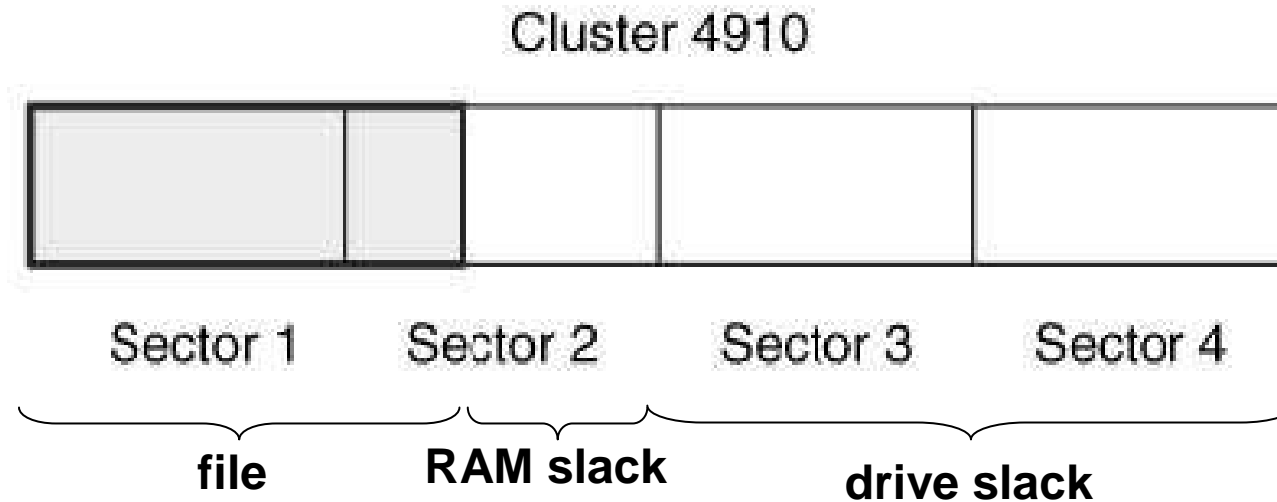
→ the RAM slack

- the space from the end of a file to the end of the last partially used sector in the last allocated cluster

→ and the drive slack.

- the space from the start of the next sector to the end of the last allocated cluster

# RAM and drive slack



- For example, let's consider a 600 byte file is stored in NTFS with a 2048-byte cluster and a 512-byte sector
  - RAM slack stretches from the end of file to the end of sector 2
  - and drive slack is composed of sectors 3 and 4.

# Testing NTFS Integrity (1)

→ **A general integrity check should be performed on NTFS before any analysis**

→ The chkdsk command

- any error indicates that the file system could have been manipulated and left in an unstable state.

→ standard Windows utility Fsutil or the Sleuth Kit command fsstat

- to obtain general information about NTFS system under investigation.

## Testing NTFS Integrity (2)

- **The cluster size of the system checked against the default size matching the file system size.**
  - The default cluster size in NTFS depends on the size of file system
  - It is uncommon for a normal user to modify the default cluster size. However, it may be modified in such a way that more hidden data can be stored.
  - For example,
    - with a larger cluster size, more hidden data can be stored in the file slack of a file.
  - An NTFS system with a cluster size that does not match its file system size is suspicious.

## Testing NTFS Integrity (3)

- **Examine metadata files with Microsoft OEM tool Nfi**
  - detect any abnormalities in terms of the length of cluster runs in \$DATA attributes, presence of named \$DATA attributes, or any other sign that the metadata files are different than expected.
- **Search for the presence of data hiding tools.**
  - Also attention should also be paid to both digital and physical files that contain a record of numbers that might be the cluster addresses where hidden data is stored. All such files could themselves be hidden using manual methods
- **Depending on the result of the steps above**
  - any combination of places for hidden data should be thoroughly examined
  - potential hidden data extracted and investigated further.

# Conclusion

## → the analysis of hidden data in NTFS file system is divided into three phases

→ determine whether any data is hidden

- either systematically check for all hiding methods, or search the system for anomalies.
- time consuming without automated tools, and requires in depth knowledge of NTFS structures

→ extract the hidden data

- hidden information may be stored in the file system without any structure or metadata

→ recover the hidden files

- Cryptographic/steganographic techniques may prevent complete data recovery

## Conclusions cont.

- **Difficulty of analysing NTFS file system for hidden data**
  - very flexible and supports many options, some not commonly used, some only partially documented.
  - As a result, there are many possible ways to hide data, especially when NTFS specific data structures are used for this purpose.
- **The additional difficulty is that the complete documentation for the NTFS system is not published**
  - it is not always possible to determine which value combinations in system data structures are valid and which are not
  - A good example is the ability to add a \$DATA attribute to a directory, which is ignored by system utilities although it does not fit the logical structure of the file system.
  - Other similar options of adding hidden data may already exist, or may yet be created by new versions of NTFS.

## Conclusions cont.

→ **Probably the best and most comprehensive documentation of NTFS**

- not from Microsoft, but from the Linux community projects which work on providing NTFS drivers for Linux operating systems.
- This documentation is the result of years of back-engineering and many experiments

→ **All this means that**

- the number of methods for hiding data will inevitably grow in the future, and to match this growth new forensic analysis tools and methodology will be created.



**Thank you.**  
**Any questions?**