FragFS: An Advanced Data Hiding Technique

Irby Thompson and Mathew Monroe

Defcon 14 - August 2006
Why is Data Hiding important?

- Do rootkits alleviate the need for data hiding?
  - NO!!
  - Rootkits hide data on a live system
  - Hiding from forensic/offline analysis is much harder
- What if the rootkit is entirely memory-based?
  - Offline disk analysis will not find it
  - No reboot persistence
- Some techniques rely on covert, persistent storage
  - “The patient hacker”
History of Data Hiding

- Information Hiding is Old News
  - Writing with invisible ink
- Hiding data on computers is often just a modern day application of existing principles
- Three major categories of data hiding:
  - Out-of-Band
  - In-Band
  - Application Layer
History of Data Hiding
Out-of-Band

Definition:
- The portion of a medium that is outside the normal specifications for that medium
- The “Media Management” Layer

Examples:
- Slack space beyond the end of a partition
- Slack space at the end of files
  - Example: slacker.exe
- Sectors marked as bad
- Host Protected Area
History of Data Hiding
In-Band

- **Definition:**
  - The portion of a medium that is inside the normal specifications for that medium
  - The “File System” Layer
  - Hidden data must not break the format of the specification

- **Examples:**
  - Alternative File Streams
  - File-System Journal Logs
  - Reserved but unallocated sectors
History of Data Hiding

Application Layer

- **Definition:**
  - Hiding in a higher-level format specification
  - Often a subset of In-Band Data Hiding viewed at a different level of granularity

- **Examples:**
  - Steganography (hiding data within data)
  - Hidden text within documents
    - Example: extra white space, tabs, new-line characters
  - Virus hiding within EXE’s code (.text) section
    - Hydran uses redundancies in i386 code to hide data
Analysis of Hiding Methods

- Well known to Forensic Tools
  - Forensic tools will specifically look for known hiding methods
    - Alternative File Streams
    - Slack space at the end of files
  - A *strings* search over a raw disk will find textual results wherever they are located
- Experienced Analysts will detect anomalies not directly identified by Forensic Tools
Out-of-Band Analysis

“Coloring Outside the Lines”

Strengths
- Being outside the boundaries usually results in being overlooked
- There is often a large amount of space available
- Hard to discover without special tools
- Resilient

Weaknesses
- Hard to access without special tools
- Hard to hide from plain-sight analysis of the out-of-band area
In-Band Analysis

“Coloring in the Nooks and Crannies”

Strengths

- Usually easy to access with existing tools
- Follows the specifications
- Less devious?

Weaknesses

- Storage space is often small
- Relies on security through obscurity – easy to detect once method is known
- Specifications may change
Application Layer Analysis

"Splatter-Painting the Canvas"

Strengths
- Hiding in plain sight
- Often hard to detect

Weaknesses
- Storage quantity varies with the size of underlying data, but must be relatively small to remain hidden
- Difficult to access without special tools
- Complex algorithms to hide/retrieve data
- Not resilient
### EnCase - Alternate File Streams

The image shows a screenshot of the EnCase Forensic software interface, specifically focusing on the Alternate File Streams feature. The interface includes a tree view of file structures on the left side and a detailed list of alternate file streams on the right side.

<table>
<thead>
<tr>
<th>Name</th>
<th>Filter</th>
<th>In Report</th>
<th>File Ext</th>
<th>File Type</th>
<th>File Category</th>
<th>Signal</th>
</tr>
</thead>
<tbody>
<tr>
<td>$BadClus:$Bad</td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>$Secure</td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>$Secure:$SOS</td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>$Secure:$SOD</td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>$Secure:$STI</td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>$UpCase</td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>termcap</td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>termcap:hidden</td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>testfile</td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>testfile:hidden</td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>Unallocated Clusters</td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
</tbody>
</table>

The bottom of the screenshot displays a text editor window showing hidden text: "This is test hidden in a ADS."
Finding New Places to Hide

- Determine constraints
  - How much space is needed?
  - What type of access is required?
  - How sensitive is hidden data?
- Decide which hiding category best fits the constraints
- Look for previously-unknown hiding methods in that category
  - Analyze an existing specification
    - May require reverse-engineering
  - Study existing data hiding techniques
  - Find unused reserved or slack space
An NTFS Overview

- Standard file system on Windows NT, Windows 2000, Windows XP, and upcoming Windows Vista
- Master File Table (MFT)
  - Every file or directory is an entry in the table
  - Stores all file system metadata in one place
  - Can grow, but not shrink
  - Not well documented or understood
MFT Entries

- Each entry is of fixed size
  - Defined in the boot sector
- Each file and directory usually requires one entry but can span multiple entries if needed
- Information about an entry is stored as attributes
  - Each entry has multiple attributes
    - Most files have a few common attributes
    - Attributes can be stored in any order
- Has per sector fix up bytes to detect defects
  - Last two bytes of each sector stored in header and fixed up on every read and write
MFT Attributes

- Attributes have different types
- Some attribute types can be repeated
  - Duplicate $DATA attributes commonly called Alternate File Streams
  - Directories entries stored as individual attributes
- Each attribute can be named, compressed, encrypted, etc
- Each attribute is either resident or non-resident
  - Resident attributes stored within MFT entry
  - Non-resident attributes stored as data run (extent)
MFT Attribute Examples

- All entries have
  - $STANDARD_INFORMATION
    - Stores timestamps, owner ID, security ID, etc
  - $FILE_NAME
    - Name by which an entry is known, size, and create/rename timestamp

- All files have $DATA attribute

- Directories use several attributes-
  - Each entry in a directory is stored as a $FILE_NAME attribute
  - DOS 8.3 name stored in a second $FILE_NAME attribute
  - Directories have additional indexing attributes to improve filename lookup performance

- End of attributes in an entry is marked by 0xFFFFFFFF

- Most attribute types are kept for backward compatibility
MFT Entry

MFT Entry Header
- Magic
- US Offset
- US Size
- LSN
- Sequence Number
- Hard Links
- Attribute Offset
- Flags
- Real Size
- Allocated Size
- Base Entry Number
- Next Attribute ID
- Reserved
- Entry Number
- US Data

MFT Entry Header

MFT Attribute

MFT Attribute

MFT Attribute

Attribute End Mark

Slack Space

Resident Attribute
- Type
- Length
- Resident Flag
- Name Length
- Name Offset
- Flags
- ID
- Attribute Length
- Attribute Offset
- Reserved
- Attribute Name
- Attribute Data

Non-Resident Attribute
- Type
- Length
- Resident Flag
- Name Length
- Name Offset
- Flags
- ID
- Data Run Offset
- ...
- Attribute Name
- Data Run
Usable Space in MFT entries

- **Reserved space within entries**
  - Many small unused areas
    - 2 bytes reserved in every entry header
    - 4 byte reserved in resident attributes
    - Up to 14 bytes are reserved in non-resident attributes
    - All attributes are 8 bytes aligned
  - Each file typically has 32 usable bytes
  - Each directory typically has 64 usable bytes

- **Slack space after entry attributes**
  - Files and directories typically have less than 450 bytes of attributes
  - Default NTFS file systems allocate 1024 bytes per MFT entry
  - Almost 600 bytes per entry!
Usage Concerns

- **Common concerns**
  - Entries may be deleted
  - Entries zeroed on allocation

- **Reserved Space**
  - Might change in future versions of NTFS
  - Normally these bytes are zeroed

- **After-attribute slack space**
  - Attributes might expand or be added
  - Commonly zero but not always
    - Attributes shrink due to going from resident to non-resident, but can’t go back to being resident
    - All directories start as resident and go to non-resident, but can’t go back to being resident
    - Attributes can be removed
Avoiding Pitfalls

- How do we find “safe” entries?
- Many files are rarely modified or deleted
  - Operating system files (drivers, .inf, font, and help files)
  - Most installed application files are only read
  - If it has never been modified it most likely never will be
  - Files that have been around for a long time are rarely deleted
- Non-resident attributes can never become resident
- Directories are rarely deleted
  - Non-resident directories in particular
- Summary - Choose entries that are
  - Non-resident
  - Have never been modified
  - Old
Putting It All Together

- How much space is available?
  - Base Windows XP Professional install has over 12,000 MFT entries
  - Typical systems have over 100,000 MFT entries
  - Not all entries are safe to use, but testing has shown ~60% of MFT entries are “safe” to use
  - 100,000 entries x 60% x 600 bytes/entry = 36,000,000 bytes!
Additional Issues

Chunking

- Small scattered chunks are not very useful
- The mapping problem
  - Need an interface that can map large blocks or streams across many chunks
  - No matter what space is being used it should look like one contiguous block to higher-level applications
- Mapping should be dynamic
  - Users will delete old files and directories and add new ones
  - Might lose data or need to use additional entries
Additional Issues

Encryption

- Data can be found by searching the raw device
- Detected data can still be protected
- How good is good enough?
  - XOR
  - Blowfish
  - LRW-AES (Narrow-block Encryption)
- Encryption plus Integrity
  - Self-Authenticating encryption is best
- Key management is hard
  - Static forensic analysis can be made difficult
  - Dynamic forensic analysis can always find the keys
Additional Issues
Change Tracking/Redundancy

- What happens when Windows updates an entry you are using?
  - NTFS only changes what it needs to change
  - Might lose some but not all of your data
- Keep extra copies
  - How much redundancy is enough?
- Do your changes get noticed by NTFS?
- Watch for NTFS changing an entry
Additional Issues
Usability

- How is the data presented to the user?
- How is the data presented to the OS?
- Use standard interfaces
  - Prevent the need to rewrite applications
- Reading and writing data files is easy
- Files execution is hard
  - Windows will only execute files from a file system that it understands
FragFS
On-Disk Implementation

- **Format**
  - Scan MFT Table for suitable entries
    - Non-resident files that have not been modified within the last year
  - Calculate how much space is available in each entry
  - Divide space into 32 byte chunks
    - 4 bytes for Logical Chunk Number
    - 28 bytes for data
  - No book marking or index of chunks on disk
    - Check the last 32 bytes of every entry to see if it is a valid chunk
    - If a chunk is valid then check 32 bytes before it the same entry to see if it is also a valid chunk
FragFS
On-Disk Implementation

- Advantages
  - Unlimited redundancy
  - Modification detection
  - Localization of data corruption
  - Easy to relocate or replicate individual chunks of data

- Disadvantages
  - Must scan entire MFT to make updates
FragFS
In-Memory Implementation

- Stackable block device interface
  - Easy to update and add new features
  - On disk format can easily change
- User-space Application Library
  - Can be linked to and used by any application
  - Built-in mini file system
- Kernel Device Driver
  - Creates a virtual disk
  - Uses the FAT file system
  - Can execute files directly from it!
FragFS Proof of Concept Demonstration
Detecting NTFS Anomalies

• Current forensic tools treat the MFT as a black box
  • There is a need for forensic tools to better understand file system structures
  • Forensic Analysts do not often have the time to comb through hex dumps

• We have developed a detection tool for data hidden in MFT entry slack space
  • Any data beyond the End-of-Attribute marker is considered suspicious
Detection Demonstration
Future Considerations

- “Hiding through Obscurity” only buys you time
- Many other unexplored data storage areas
- Hiding access tools is still a problem
  - Bootstrap out of the hidden space?
- Should file system standards be open?
  - Forensic tools could better detect hidden data
  - File systems will be easier to exploit
Special Thanks To:

- The Grugg
  - For his previous work in the field of file system anti-forensics
- Brian Carrier
  - For making file system forensics available to everyone
- Fred Jacobs
  - For his help with the detection utility
- Sam Stover
  - For help with forensic tools and brainstorming
- Matt Hartley
  - For excellent insight and patience with the project
Contact Information

Irby Thompson
lantholin (at) gmail.com

Mathew Monroe
mathew.monroe (at) gmail.com