FORENSICS

Analyzing an Unknown Image

Submitted by
PRAVEEN DARSHANAM

praveen_recker@sify.com
http://www.darshanams.blogspot.com/

KNOWLEDGE IS NOT AN OBJECT. IT'S A FLOW.

This is not a highly technical document but wrote with a notion that this document might help someone somewhere gain some knowledge and pave path to delve deep into forensics depending on the interest.

Thanks to str0ke, All Andhra Hackers and All Indian Hackers !!!

Grabbing the image for analyzing

I extracted this image under analysis from a Virtual Machine with Windows 2000 Server Running.

Many open source and commercial tools are available to take the image of a drive, hard disk, partition etc. Few tools which can be used are dd, windd etc.
I used dd command for taking the image of the running virtual machine.

First, lets list all the available drives/partitions on the VM.

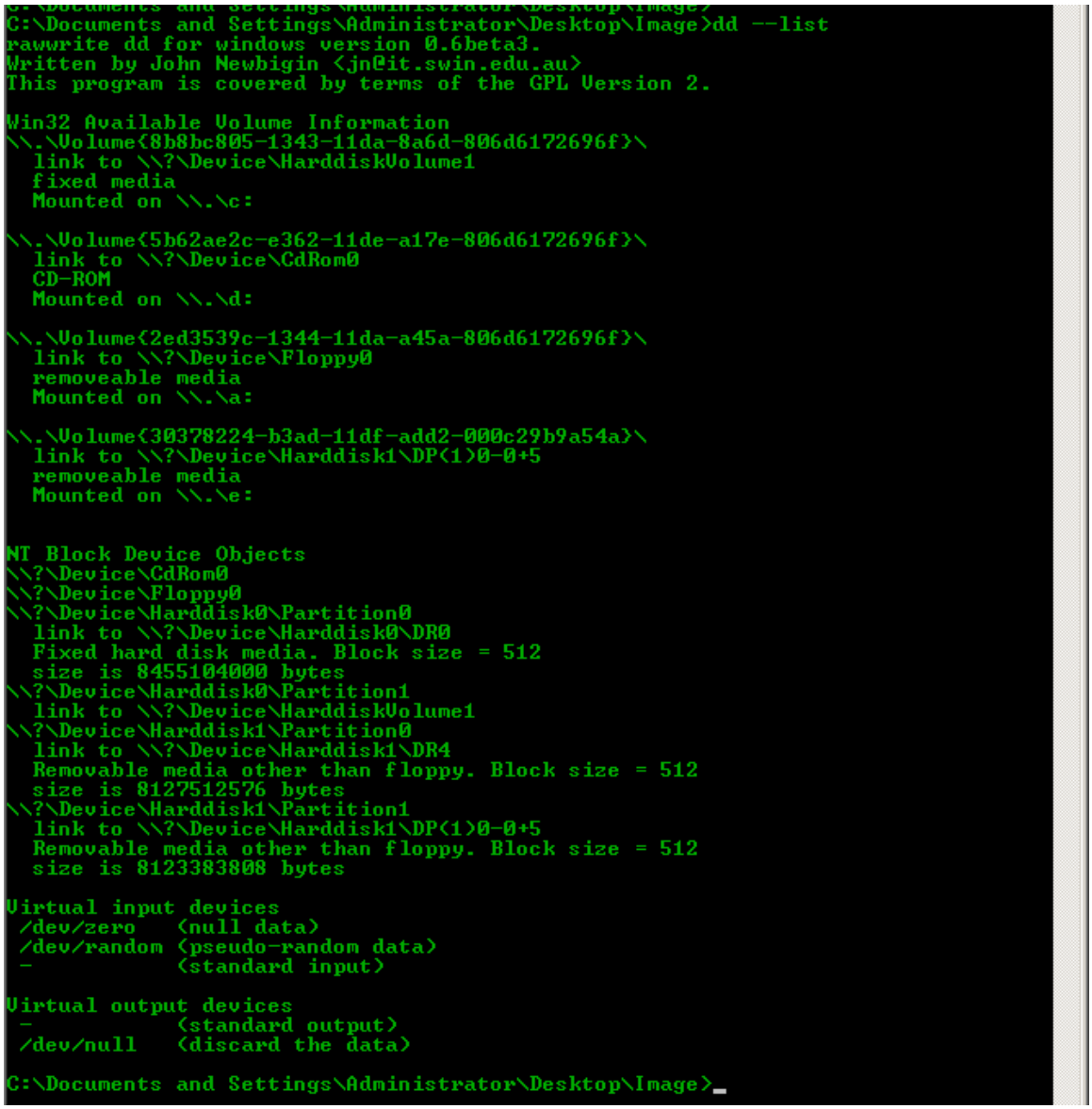


Figure: Available drives/partitions on the VM

\\.\a: A Drive, Floppy Drive
\\.\c: C Drive
\\.\d: C Drive, CD ROM Drive
\\.\e: C Drive, USB Drive

Extracted the image using dd.exe.

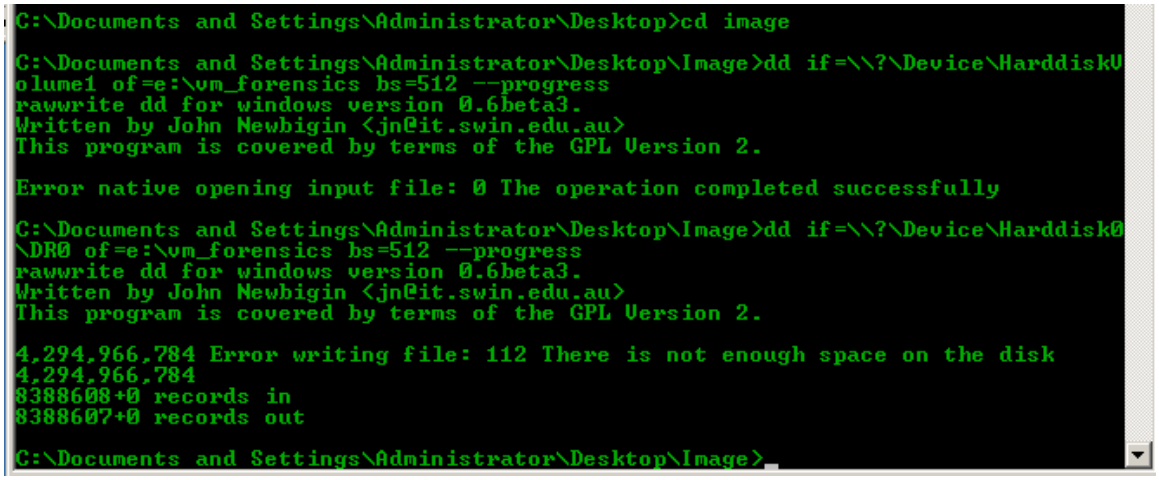


Figure. Snapshot of the dd command usage for extracting the VM image

if	input drive name which is used for analysis
of	output path to save the image
bs	block size
--progress	shows the progress of the image

P.S. Couldn't get the full image as there is no sufficient space on my machine.

By conducting investigations on the disk image, we could unearth any hidden intrusions since the image captures the invisible information as well. The advantages of analyzing disk images are that the investigators can:

a) preserve the digital crime-scene
b) obtain the information in slack space
c) access unallocated space, free space, and used space
d) recover file fragments, hidden or deleted files and directories
e) view the partition structure and
f) get date-stamp and ownership of files and folders.

Lets check the md5 hash of the image under analysis for integrity purposes. The md5 hash algorithm produces a 128 bit "fingerprint" of a file, also known as a message digest. This ensures non-repudiation integrity of the file. To view the md5 hash value assigned to a given file, the md5sum utility can be used.

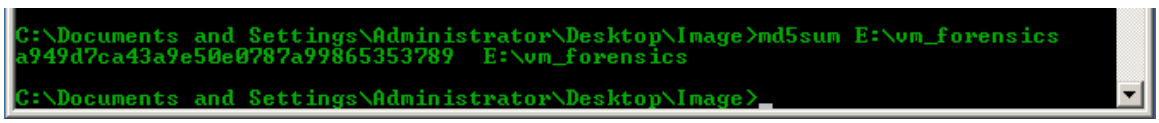


Figure. md5sum of the image

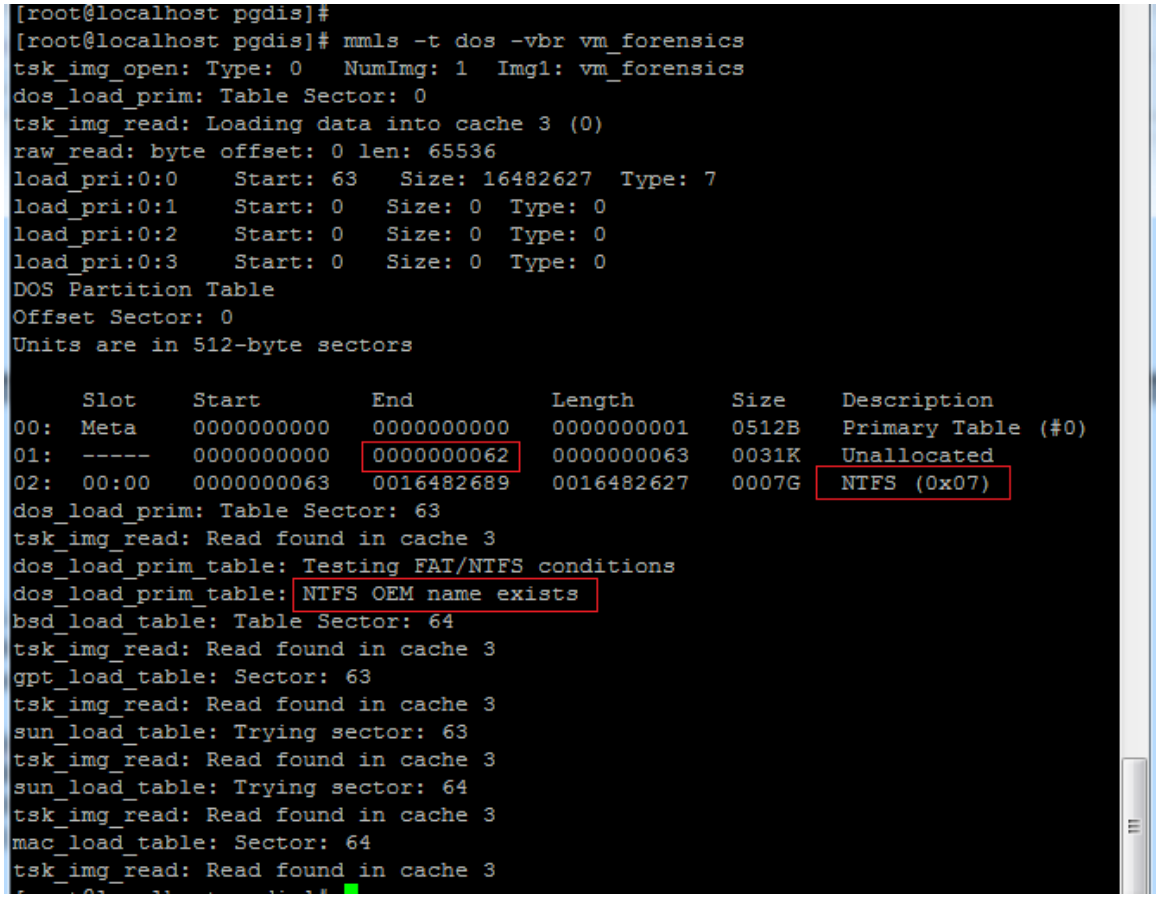
Lets check the **file type** of the image under analysis by using **file** command. The file command works by testing "arguments" within a file, and will then classify the file as whichever file type the file command sees fit. We see from the output of the file command that the image file contains an x86 boot sector. The boot sector of a computer is a primary starting point for an OS. The operating system will start at the boot loader, and the machine will read the first 512 bytes of the disk, which is known as the boot sector. The first 512 Bytes (boot sector) will be loaded into memory and will then be executed. This will initiate the boot process.

The x86 boot sector type message was obtained because the magic number 0xAA55 value is located at the 0x1FE offset within the image; defined in the file "/usr/share/file/magic" which is used by file command.

```
[root@localhost pgdis]#
[root@localhost pgdis]# file vm_forensics
vm_forensics: x86 boot sector, Microsoft Windows XP MBR, Serial 0x9e43652a; part
ition 1: ID=0x7, active, starthead 1, startsector 63, 16482627 sectors, code off
set 0xc0
[root@localhost pgdis]#
[root@localhost pgdis]#
```

The x86 boot sector type message was obtained because the magic number 0xAA55 value is located at the 0x1FE offset within the image; defined in the file "/usr/share/file/magic" which is used by file command.

Lets run mmls utility to determine the File system type of the given image extracted by using dd command.



-t Specify the media management type (dos, mac, bsd etc)
-b partition sizes in bytes
-r Recurse into DOS partitions and look for other partition tables.
-v verbose

We see above that the NTFS (*New Technology File System*) partition begins at sector 63 (to see this look at the last column in the row where it says NTFS (0x07). Now look to the left in the start column of the row NTFS and we can see the value 0000000063). So for all the Sleuth Kit commands we need to specify an **offset of 63** if the file used is raw image.

MMLS is a forensics utility that query's an image file, and prints the partition tables and disk labels. This command is very useful when attempting to determine at which sector a partition begins and ends. We see that there is a NTFS file system on this image. We use the –t dos switch to tell mmls to utilize a dos based architecture for the file system.

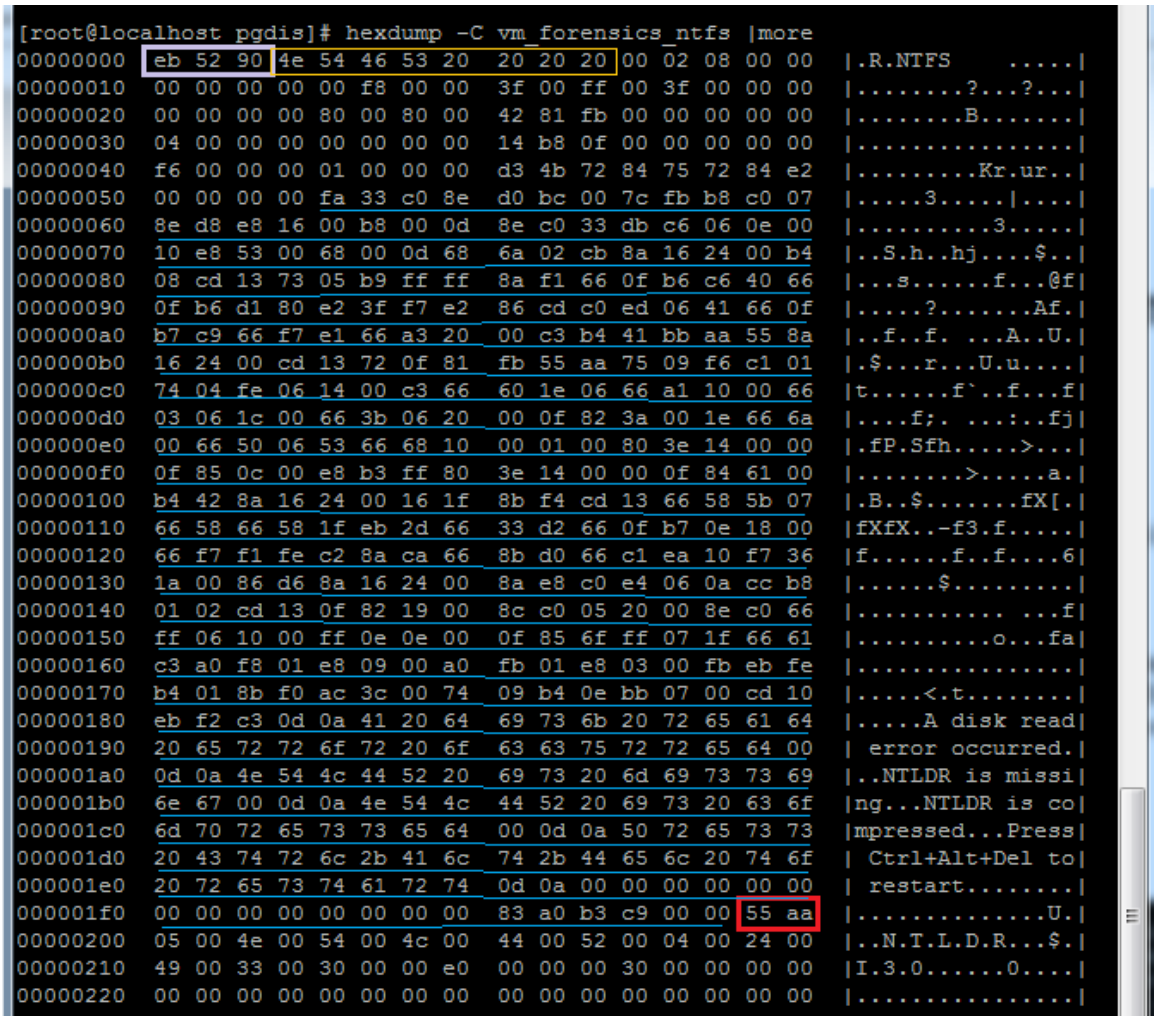
NTFS Structure

This structure is before separating Physical File System from the Logical File system or Raw Image.

[root@localhost pgdis]# hexdump -C vm_forensics |more

```
00007e00  eb 52 90 4e 54 46 53 20  20 20 20 00 02 08 00 00  |.R.NTFS    .....|
00007e10  00 00 00 00 00 f8 00 00  3f 00 ff 00 3f 00 00 00  |........?...?...|
00007e20  00 00 00 00 80 00 80 00  42 81 fb 00 00 00 00 00  |........B.......|
00007e30  04 00 00 00 00 00 00 00  14 b8 0f 00 00 00 00 00  |................|
00007e40  f6 00 00 00 01 00 00 00  d3 4b 72 84 75 72 84 e2  |.........Kr.ur..|
00007e50  00 00 00 00 fa 33 c0 8e  d0 bc 00 7c fb b8 c0 07  |.....3.....|....|
00007e60  8e d8 e8 16 00 b8 00 0d  8e c0 33 db c6 06 0e 00  |..........3.....|
00007e70  10 e8 53 00 68 00 0d 68  6a 02 cb 8a 16 24 00 b4  |..S.h..hj....$..|
00007e80  08 cd 13 73 05 b9 ff ff  8a f1 66 0f b6 c6 40 66  |...s......f...@f|
00007e90  0f b6 d1 80 e2 3f f7 e2  86 cd c0 ed 06 41 66 0f  |.....?.......Af.|
00007ea0  b7 c9 66 f7 e1 66 a3 20  00 c3 b4 41 bb aa 55 8a  |..f..f. ...A..U.|
00007eb0  16 24 00 cd 13 72 0f 81  fb 55 aa 75 09 f6 c1 01  |.$...r...U.u....|
00007ec0  74 04 fe 06 14 00 c3 66  60 1e 06 66 a1 10 00 66  |t......f`..f...f|
00007ed0  03 06 1c 00 66 3b 06 20  00 0f 82 3a 00 1e 66 6a  |....f;. ...:..fj|
00007ee0  00 66 50 06 53 66 68 10  00 01 00 80 3e 14 00 00  |.fP.Sfh.....>...|
00007ef0  0f 85 0c 00 e8 b3 ff 80  3e 14 00 00 0f 84 61 00  |........>.....a.|
00007f00  b4 42 8a 16 24 00 16 1f  8b f4 cd 13 66 58 5b 07  |.B..$.......fX[.|
00007f10  66 58 66 58 1f eb 2d 66  33 d2 66 0f b7 0e 18 00  |fXfX..-f3.f.....|
00007f20  66 f7 f1 fe c2 8a ca 66  8b d0 66 c1 ea 10 f7 36  |f......f..f....6|
00007f30  1a 00 86 d6 8a 16 24 00  8a e8 c0 e4 06 0a cc b8  |......$.........|
00007f40  01 02 cd 13 0f 82 19 00  8c c0 05 20 00 8e c0 66  |........... ...f|
00007f50  ff 06 10 00 ff 0e 0e 00  0f 85 6f ff 07 1f 66 61  |..........o...fa|
00007f60  c3 a0 f8 01 e8 09 00 a0  fb 01 e8 03 00 fb eb fe  |................|
00007f70  b4 01 8b f0 ac 3c 00 74  09 b4 0e bb 07 00 cd 10  |.....<.t........|
00007f80  eb f2 c3 0d 0a 41 20 64  69 73 6b 20 72 65 61 64  |.....A disk read|
00007f90  20 65 72 72 6f 72 20 6f  63 63 75 72 72 65 64 00  | error occurred.|
00007fa0  0d 0a 4e 54 4c 44 52 20  69 73 20 6d 69 73 73 69  |..NTLDR is missi|
00007fb0  6e 67 00 0d 0a 4e 54 4c  44 52 20 69 73 20 63 6f  |ng...NTLDR is co|
00007fc0  6d 70 72 65 73 73 65 64  00 0d 0a 50 72 65 73 73  |mpressed...Press|
00007fd0  20 43 74 72 6c 2b 41 6c  74 2b 44 65 6c 20 74 6f  | Ctrl+Alt+Del to|
00007fe0  20 72 65 73 74 61 72 74  0d 0a 00 00 00 00 00 00  | restart........|
00007ff0  00 00 00 00 00 00 00 00  83 a0 b3 c9 00 00 55 aa  |..............U.|
00008000  05 00 4e 00 54 00 4c 00  44 00 52 00 04 00 24 00  |..N.T.L.D.R...$.|
00008010  49 00 33 00 30 00 00 e0  00 00 00 30 00 00 00 00  |I.3.0......0....|
00008020  00 00 00 00 00 00 00 00  00 00 00 00 00 00 00 00  |................|
```

Structure of Sector 1 (MBR)



```
[root@localhost pgdis]# hexdump -C vm_forensics_ntfs |more
00000000  eb 52 90 4e 54 46 53 20  20 20 20 00 02 08 00 00  |.R.NTFS    .....|
00000010  00 00 00 00 00 f8 00 00  3f 00 ff 00 3f 00 00 00  |........?...?...|
00000020  00 00 00 00 80 00 80 00  42 81 fb 00 00 00 00 00  |........B.......|
00000030  04 00 00 00 00 00 00 00  14 b8 0f 00 00 00 00 00  |................|
00000040  f6 00 00 00 01 00 00 00  d3 4b 72 84 75 72 84 e2  |.........Kr.ur..|
00000050  00 00 00 00 fa 33 c0 8e  d0 bc 00 7c fb b8 c0 07  |.....3.....|....|
00000060  8e d8 e8 16 00 b8 00 0d  8e c0 33 db c6 06 0e 00  |..........3.....|
00000070  10 e8 53 00 68 00 0d 68  6a 02 cb 8a 16 24 00 b4  |..S.h..hj....$..|
00000080  08 cd 13 73 05 b9 ff ff  8a f1 66 0f b6 c6 40 66  |...s......f...@f|
00000090  0f b6 d1 80 e2 3f f7 e2  86 cd c0 ed 06 41 66 0f  |.....?.......Af.|
000000a0  b7 c9 66 f7 e1 66 a3 20  00 c3 b4 41 bb aa 55 8a  |..f..f. ...A..U.|
000000b0  16 24 00 cd 13 72 0f 81  fb 55 aa 75 09 f6 c1 01  |.$...r...U.u....|
000000c0  74 04 fe 06 14 00 c3 66  60 1e 06 66 a1 10 00 66  |t......f`..f...f|
000000d0  03 06 1c 00 66 3b 06 20  00 0f 82 3a 00 1e 66 6a  |....f;. ...:..fj|
000000e0  00 66 50 06 53 66 68 10  00 01 00 80 3e 14 00 00  |.fP.Sfh.....>...|
000000f0  0f 85 0c 00 e8 b3 ff 80  3e 14 00 00 0f 84 61 00  |........>.....a.|
00000100  b4 42 8a 16 24 00 16 1f  8b f4 cd 13 66 58 5b 07  |.B..$.......fX[.|
00000110  66 58 66 58 1f eb 2d 66  33 d2 66 0f b7 0e 18 00  |fXfX..-f3.f.....|
00000120  66 f7 f1 fe c2 8a ca 66  8b d0 66 c1 ea 10 f7 36  |f......f..f....6|
00000130  1a 00 86 d6 8a 16 24 00  8a e8 c0 e4 06 0a cc b8  |......$.........|
00000140  01 02 cd 13 0f 82 19 00  8c c0 05 20 00 8e c0 66  |........... ...f|
00000150  ff 06 10 00 ff 0e 0e 00  0f 85 6f ff 07 1f 66 61  |..........o...fa|
00000160  c3 a0 f8 01 e8 09 00 a0  fb 01 e8 03 00 fb eb fe  |................|
00000170  b4 01 8b f0 ac 3c 00 74  09 b4 0e bb 07 00 cd 10  |.....<.t........|
00000180  eb f2 c3 0d 0a 41 20 64  69 73 6b 20 72 65 61 64  |.....A disk read|
00000190  20 65 72 72 6f 72 20 6f  63 63 75 72 72 65 64 00  | error occurred.|
000001a0  0d 0a 4e 54 4c 44 52 20  69 73 20 6d 69 73 73 69  |..NTLDR is missi|
000001b0  6e 67 00 0d 0a 4e 54 4c  44 52 20 69 73 20 63 6f  |ng...NTLDR is co|
000001c0  6d 70 72 65 73 73 65 64  00 0d 0a 50 72 65 73 73  |mpressed...Press|
000001d0  20 43 74 72 6c 2b 41 6c  74 2b 44 65 6c 20 74 6f  | Ctrl+Alt+Del to|
000001e0  20 72 65 73 74 61 72 74  0d 0a 00 00 00 00 00 00  | restart........|
000001f0  00 00 00 00 00 00 00 00  83 a0 b3 c9 00 00 55 aa  |..............U.|
00000200  05 00 4e 00 54 00 4c 00  44 00 52 00 04 00 24 00  |..N.T.L.D.R...$.|
00000210  49 00 33 00 30 00 00 e0  00 00 00 30 00 00 00 00  |I.3.0......0....|
00000220  00 00 00 00 00 00 00 00  00 00 00 00 00 00 00 00  |................|
```

Byte Offset	Field Length	Field Name
0x00	3 bytes	Jump Instruction (0xeb5290)
0x03	LONGLONG	OEM ID (4e 54 46 53 20 20 20 20)
0x0B	25 bytes	BPB
0x24	48 bytes	Extended BPB
0x54	426 bytes	Bootstrap Code
0x01FE	WORD	End of Sector Marker (55 aa)

eb 52 JMP 82 //Jump 82 (dec) bytes

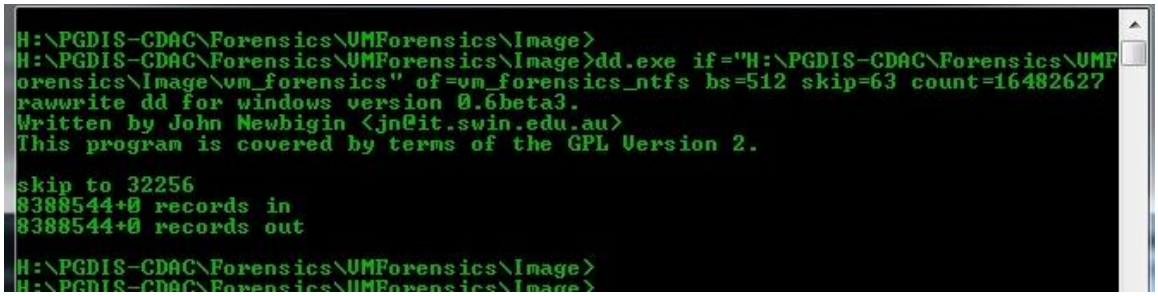
90 NOP

On NTFS volumes, the data fields that follow the BPB form an extended BPB. The data in these fields enables Ntldr (NT loader program) to find the master file table (MFT) during startup. On NTFS volumes, the MFT is not located in a predefined sector, as on FAT16 and FAT32 volumes. For this reason, the MFT can be moved if there is a bad sector in its normal location. However, if the data is corrupted, the MFT cannot be located, and Windows NT/2000 assumes that the volume has not been formatted.

Bytes 0x00- 0x0A are the jump instruction and the OEM ID
Bytes 0x0B-0x53 are the BPB and the extended BPB.
The remaining code is the bootstrap code and the end of sector.

Extracting the File System from the image

File system is extracted using dd.exe command. Input file is the raw image collected from the machine which is under forensic investigation. Block size used to extract File system is 512 bytes and skipped 62 sectors because our NTFS File System is starting after those sectors.

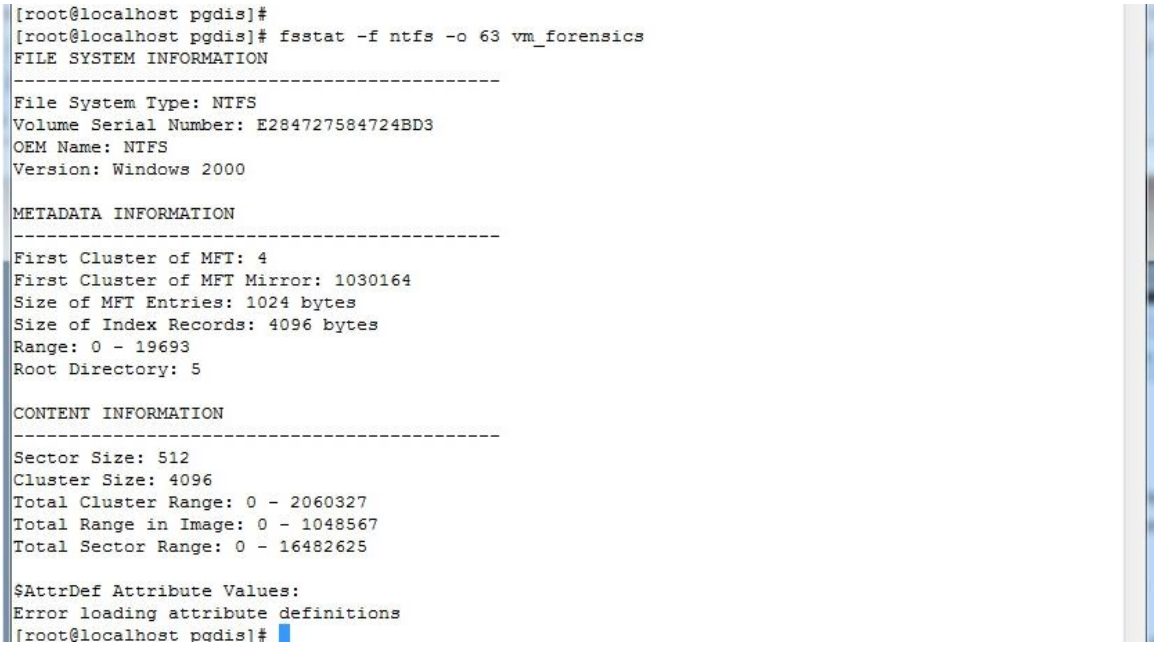


Calculating md5 of the extracted NTFS file system image

Analysis of different Layers

Layers	Sleuth Kit Commands
Physical Layer	m*-commands: mmls
File System Layer	fs*-commands: fsstat
File Name Layer	f*-commands: fls
Metadata Layer	i*-commands: icat
Data Layer	d*-commands: dcat

fsstat command output of the image is



-f type of file system (fat12, ext2, ntfs, mac etc)
-o sector offset where the file system starts in the image

[root@localhost pgdis]# fsstat vm_forensics_ntfs
FILE SYSTEM INFORMATION

File System Type: NTFS
Volume Serial Number: E284727584724BD3
OEM Name: NTFS
Version: Windows 2000

METADATA INFORMATION

First Cluster of MFT: 4
First Cluster of MFT Mirror: 1030164
Size of MFT Entries: 1024 bytes
Size of Index Records: 4096 bytes
Range: 0 - 19693
Root Directory: 5

CONTENT INFORMATION

Sector Size: 512
Cluster Size: 4096
Total Cluster Range: 0 - 2060327

Total Range in Image: 0 - 1048567
Total Sector Range: 0 - 16482625

$AttrDef Attribute Values:
Error loading attribute definitions
[root@localhost pgdis]#

To investigate how intrusions result in data hiding, data deletion and other obfuscations, it is essential to understand the physical characteristics of the Microsoft NTFS file system. Master File Table (MFT) is the core of NTFS since it contains details of every file and folder on the volume and allocates two sectors for every MFT entry. Hence, a good knowledge of the MFT layout structure also facilitates the disk recovery process.

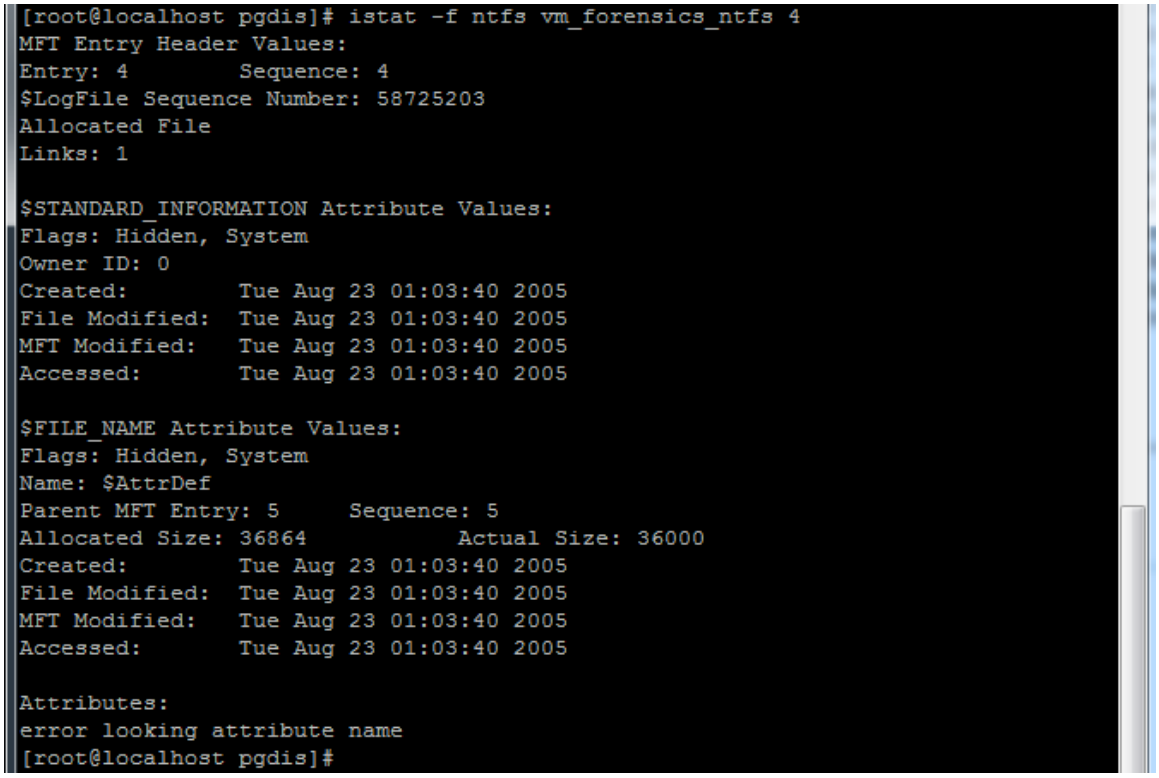
In NTFS, everything on disk is a file. Even the metadata is stored as a set of files. The Master File Table (MFT) is an index of every file on the volume. For each file, the MFT keeps a set of records called attributes and each attribute stores a different type of information. Each MFT entry has a fixed size which is 1 KB (at byte offset 64 in the boot sector one could identify the MFT record size).

```
[root@localhost pgdis]# istat -f ntfs vm_forensics_ntfs 7
MFT Entry Header Values:
Entry: 7        Sequence: 7
$LogFile Sequence Number: 0
Allocated File
Links: 1

$STANDARD_INFORMATION Attribute Values:
Flags: Hidden, System
Owner ID: 0
Created:        Tue Aug 23 01:03:40 2005
File Modified:  Tue Aug 23 01:03:40 2005
MFT Modified:   Tue Aug 23 01:03:40 2005
Accessed:       Tue Aug 23 01:03:40 2005

$FILE_NAME Attribute Values:
Flags: Hidden, System
Name: $Boot
Parent MFT Entry: 5     Sequence: 5
Allocated Size: 8192    Actual Size: 8192
Created:        Tue Aug 23 01:03:40 2005
File Modified:  Tue Aug 23 01:03:40 2005
MFT Modified:   Tue Aug 23 01:03:40 2005
Accessed:       Tue Aug 23 01:03:40 2005

Attributes:
error looking attribute name
[root@localhost pgdis]#
```

The $Boot metadata file structure is located in MFT entry 7 (inode) and contains the boot sector of the file system. It contains information about the size of the volume, clusters and the MFT. The $Boot metadata file structure has four attributes, namely, $STANDARD_INFORMATION, $FILE_NAME, $SECURITY_DESCRIPTION and $DATA. The $STANDARD_INFORMATION attribute contains temporal information such as flags, owner, security ID and the last accessed, written, and created times.

The $FILE_NAME attribute contains the file name in UNICODE, the size and temporal information as well. The $SECURITY_DESCRIPTION attribute contains information about the access control and security properties. Finally, the $DATA attribute contains the file contents. These attributes values for the test sample are shown in above snapshot as an illustration. To achieve this, we used the following TSK command tools:

[root@localhost pgdis]# istat -f ntfs vm_forensics_ntfs 7



istat utility displays details of a meta-data structure i.e. inode. -f ntfs says File System type of the image is NTFS, inode number 4 gives the information of $AttrDef file.

NTFS includes several system files, all of which are hidden from view on the NTFS volume. A *system file* is one used by the file system to store its metadata and to implement the file system. System files are placed on the volume by the Format utility.

Table explaining Metadata Stored in the Master File Table

SYSTEM FILE	FILE NAME	MFT RECORD	PURPOSE OF THE FILE
Master file table	**$Mft**	0	Contains one base file record for each file and folder on an NTFS volume. If the allocation information for a file or folder is too large to fit within a single record, other file records are allocated as well.
Master file table 2	**$MftMirr**	1	A duplicate image of the first four records of the MFT. This file guarantees access to the MFT in case of a single-sector failure.
Log file	**$LogFile**	2	Contains a list of transaction steps used for NTFS recoverability. Log file size depends on the volume size and can be as large as 4 MB. It is used by Windows NT/2000 to restore consistency to NTFS after a system failure.
Volume	**$Volume**	3	Contains information about the volume, such as the volume label and the volume version.
Attribute definitions	**$AttrDef**	4	A table of attribute names, numbers, and descriptions.
Root file name index	**$**	5	The root folder.
Cluster bitmap	**$Bitmap**	6	A representation of the volume showing which clusters are in use.
Boot sector	**$Boot**	7	Includes the BPB used to mount the volume and additional bootstrap loader code used if the volume is bootable.
Bad cluster file	**$BadClus**	8	Contains bad clusters for the volume.
Security file	**$Secure**	9	Contains unique security descriptors for all files within a volume.
Upcase table	**$Upcase**	10	Converts lowercase characters to matching Unicode uppercase characters.
NTFS extension file	**$Extend**	11	Used for various optional extensions such as quotas, reparse point data, and object identifiers.
		12-15	Reserved for future use.
Quota management file	**$Quota**	24	Contains user assigned quota limits on the volume space.
Object Id file	**$ObjId**	25	Contains file object IDs.
Reparse point file	**$Reparse**	26	This file contains information about files and folders on the volume include reparse point data

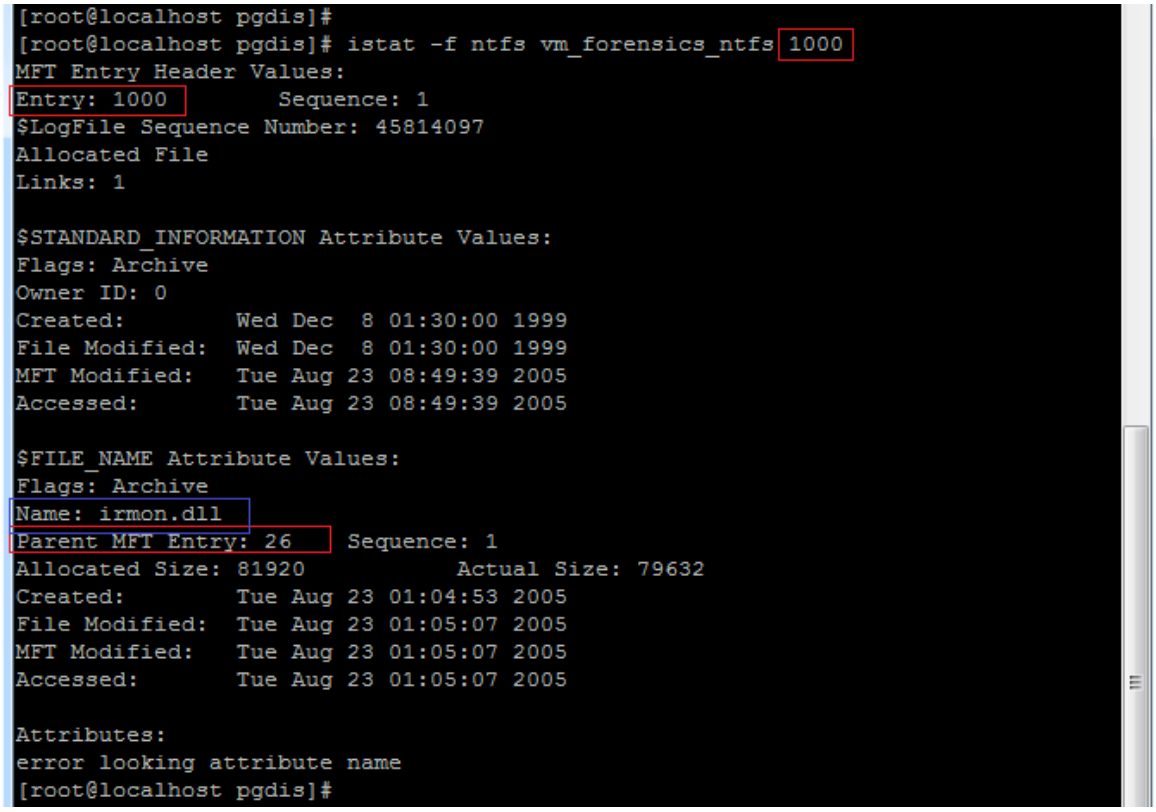


Figure. Showing contents at inode 1000 (picked randomly)

Following commands are not giving output when ran on vm_forensics image
[root@localhost pgdis]# fls -f ntfs -o 63 -aD vm_forensics
[root@localhost pgdis]# fls -f ntfs -o 63 -a vm_forensics
[root@localhost pgdis]# fls -f ntfs -o 63 vm_forensics

[root@localhost pgdis]# blkstat -vvf ntfs -o 63 vm_forensics 20
tsk_parse_offset: Offset set to 32256
tsk_img_open: Type: 0 NumImg: 1 Img1: vm_forensics
tsk_img_read: Loading data into cache 3 (32256)
raw_read: byte offset: 32256 len: 65536
ntfs_dinode_lookup: Processing MFT 0
tsk_img_read: Read found in cache 3
ntfs_dinode_lookup: upd_seq 1 Replacing: 0068 With: 0000
ntfs_dinode_lookup: upd_seq 2 Replacing: 0068 With: 0000
ntfs_proc_attrseq: Processing entry 0
ntfs_proc_attrseq: Resident Attribute in 0 Type: 16 Id: 0 Name: N/A
ntfs_proc_attrseq: Resident Attribute in 0 Type: 48 Id: 3 Name: N/A
ntfs_proc_attrseq: Non-Resident Attribute in 0 Type: 128 Id: 1 Name: $Data Start VCN: 0

```
ntfs_make_data_run: Len idx: 0 cur: 60 (3c) tot: 60 (3c)
ntfs_make_data_run: Len idx: 1 cur: 19 (13) tot: 4924 (133c)
ntfs_make_data_run: Off idx: 0 cur: 4 (4) tot: 4 (4)
ntfs_make_data_run: Signed addr_offset: 4 Previous address: 0
ntfs_proc_attrseq: Non-Resident Attribute in 0 Type: 176 Id: 5 Name: N/A  Start VCN: 0
ntfs_make_data_run: Len idx: 0 cur: 1 (1) tot: 1 (1)
ntfs_make_data_run: Off idx: 0 cur: 2 (2) tot: 2 (2)
ntfs_make_data_run: Signed addr_offset: 2 Previous address: 0
ntfs_dinode_lookup: Processing MFT 3
ntfs_dinode_lookup: Found in offset: 4  size: 4924 at offset: 3072
ntfs_dinode_lookup: Entry address at: 19456
tsk_img_read: Read found in cache 3
ntfs_dinode_lookup: upd_seq 1   Replacing: 0068   With: 0000
ntfs_dinode_lookup: upd_seq 2   Replacing: 0068   With: 0000
ntfs_proc_attrseq: Processing entry 3
ntfs_proc_attrseq: Resident Attribute in 3 Type: 16 Id: 0 Name: N/A
ntfs_proc_attrseq: Resident Attribute in 3 Type: 48 Id: 1 Name: N/A
ntfs_proc_attrseq: Resident Attribute in 3 Type: 64 Id: 6 Name: N/A
ntfs_proc_attrseq: Resident Attribute in 3 Type: 80 Id: 2 Name: N/A
ntfs_proc_attrseq: Resident Attribute in 3 Type: 96 Id: 4 Name: N/A
ntfs_proc_attrseq: Resident Attribute in 3 Type: 112 Id: 5 Name: N/A
ntfs_proc_attrseq: Resident Attribute in 3 Type: 128 Id: 3 Name: $Data
ntfs_dinode_lookup: Processing MFT 6
ntfs_dinode_lookup: Found in offset: 4  size: 4924 at offset: 6144
ntfs_dinode_lookup: Entry address at: 22528
tsk_img_read: Read found in cache 3
ntfs_dinode_lookup: upd_seq 1   Replacing: 003b   With: 0000
ntfs_dinode_lookup: upd_seq 2   Replacing: 003b   With: 0000
ntfs_make_data_run: Len idx: 0 cur: 63 (3f) tot: 63 (3f)
ntfs_make_data_run: Off idx: 0 cur: 95 (5f) tot: 95 (5f)
ntfs_make_data_run: Off idx: 1 cur: 226 (e2) tot: 57951 (e25f)
ntfs_make_data_run: Off idx: 2 cur: 15 (f) tot: 1040991 (fe25f)
ntfs_make_data_run: Signed addr_offset: 1040991 Previous address: 0
tsk_img_read: Loading data into cache 2 (4263931392)
raw_read: byte offset: 4263931392 len: 65536
ssize: 512 csize: 8 serial: e284727584724bd3
mft_rsize: 1024 idx_rsize: 4096 vol: 2060328 mft: 4 mft_mir: 1030164
tsk_img_read: Loading data into cache 1 (114176)
raw_read: byte offset: 114176 len: 65536
Cluster: 20
Allocated
[root@localhost pgdis]#
```

[root@localhost pgdis]# ils -rf ntfs vm_forensics_ntfs
class|host|device|start_time
ils|localhost.localdomain||1274610996
st_ino|st_alloc|st_uid|st_gid|st_mtime|st_atime|st_ctime|st_crtime|st_mode|st_nlink|st_size
16|f|0|0|0|0|0|0|0|0
17|f|0|0|0|0|0|0|0|0
18|f|0|0|0|0|0|0|0|0
19|f|0|0|0|0|0|0|0|0
20|f|0|0|0|0|0|0|0|0
21|f|0|0|0|0|0|0|0|0
22|f|0|0|0|0|0|0|0|0
23|f|0|0|0|0|0|0|0|0
[root@localhost pgdis]#

string Analysis

Extracted all the strings from image, vm_forensics_ntfs using strings utility. Was able to extract different Username/ Password combinations shown below.

#define HOST_USER_DEFAULT "tornado"
#define HOST_PASSWORD_DEFAULT "tornado+"

#define LOGIN_USER_NAME "target"
#define LOGIN_PASSWORD "bReb99RRed"

/* $Id: bedrock.h,v 1.1 2003/03/14 20:12:20 keith Exp $
 * This file is subject to the terms and conditions of the GNU General Public
 * License. See the file "COPYING" in the main directory of this archive
 * for more details.
 * Copyright (C) 1992 - 1997, 2000-2002 Silicon Graphics, Inc. All rights reserved.
#ifndef _ASM_IA64_SN_SN1_BEDROCK_H
#define _ASM_IA64_SN_SN1_BEDROCK_H
/* The secret password; used to release protection */
#define HUB_PASSWORD 0x53474972756c6573ull

Lets analyze using MountImagePro v4.12

Downloaded Mount Image Pro v4.12 (Trial) and tried to mount vm_forensics_ntfs image but vm_forensics_ntfs was not mounting properly so renamed to vm_forensics_ntfs.dd

Mounting procedure:
1. "Add Image" to add a forensic image file
2. Select the device or image that you wish to mount and then press the "Mount Filesystem" button
3. The device or image will then mount and display
4. If the drive is mounted with a drive letter, you should then be able to browse to the drive using Windows. Double click on the drive letter to open Windows Explorer.

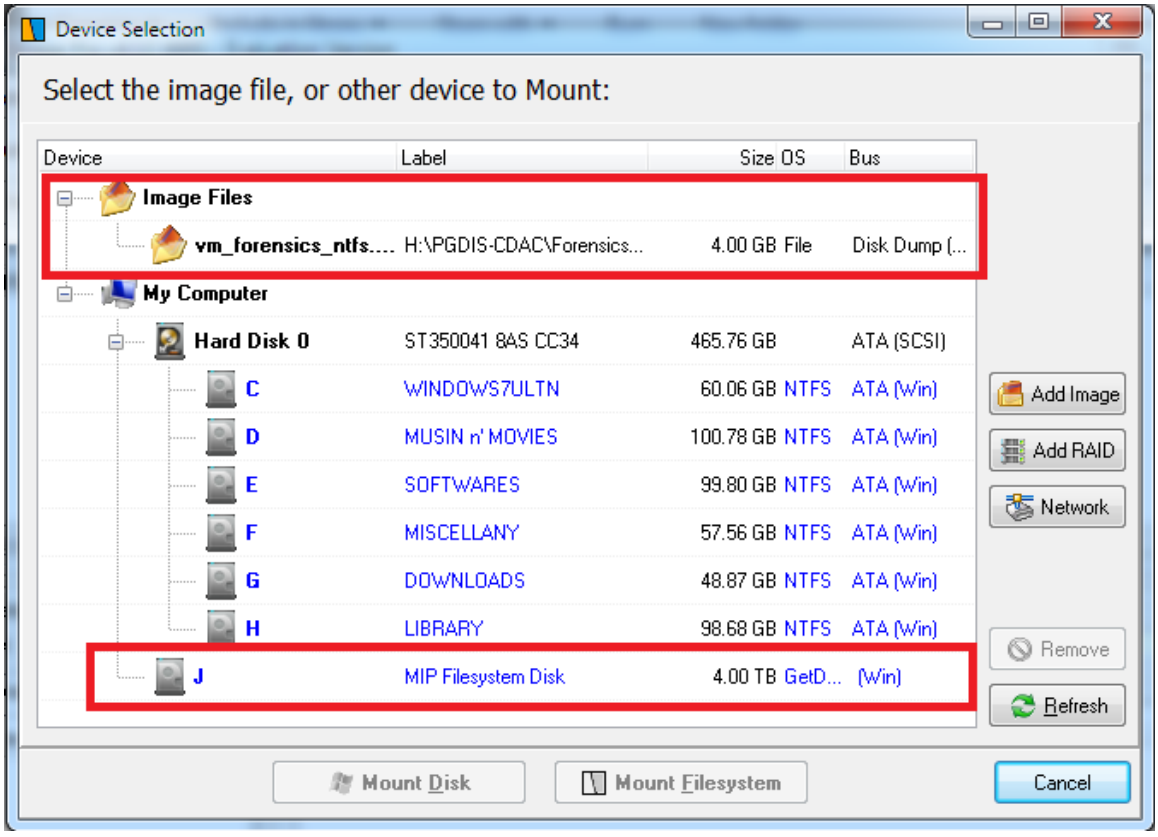


Figure: Mount snapshot on MountImagePro v4.12

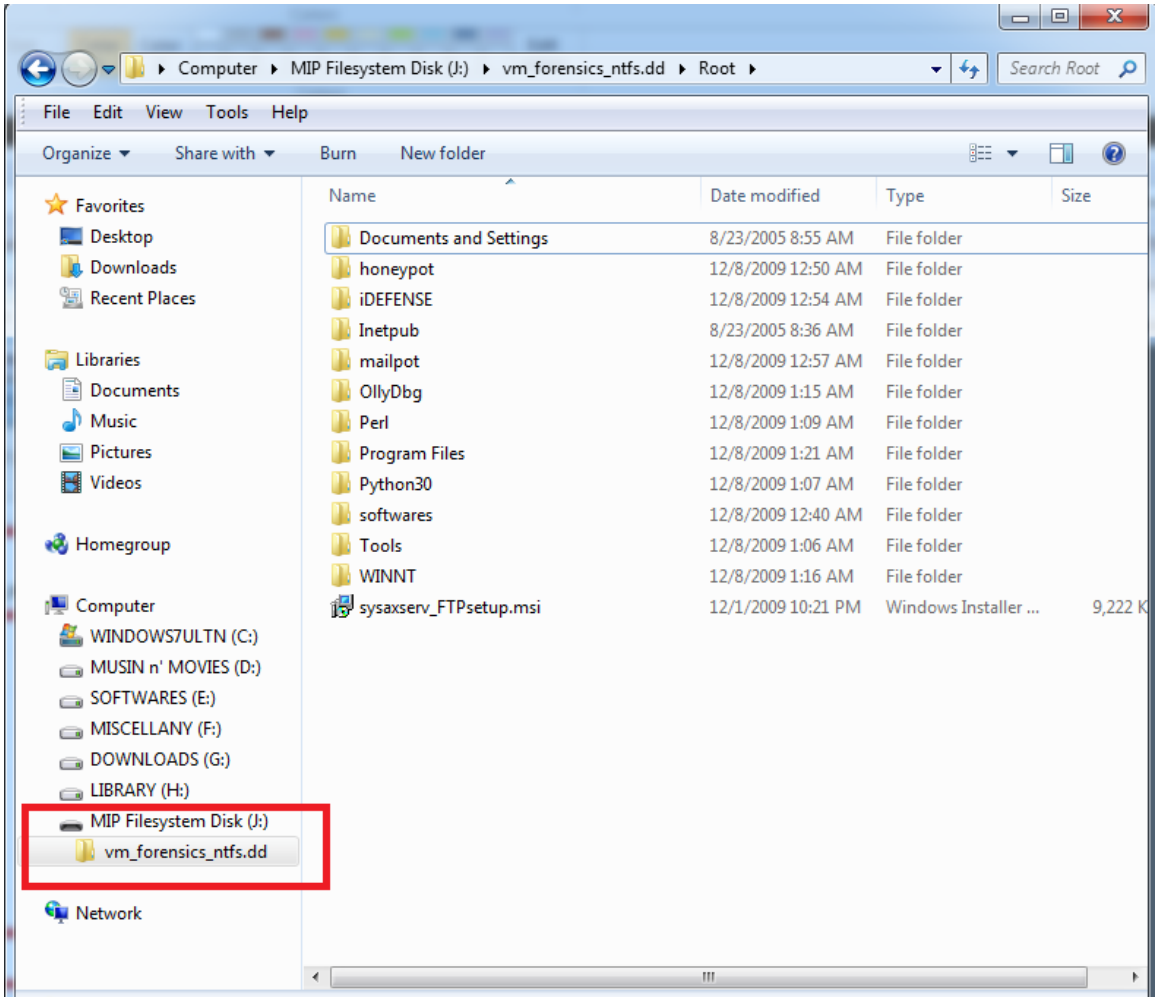


Figure: Mounted drive J and it's Directories as seen in Windows Explorer

References:

http://en.wikipedia.org/wiki/Computer_forensics
http://www.sleuthkit.org/
http://www.ntfs.com/
http://www.volatilesystems.com/
http://www.mountimage.com/encase-image-mount.php
http://www.forensicfocus.com/dissecting-ntfs-hidden-streams
http://blogs.sans.org/computer-forensics/2009/12/18/ntfs-an-introduction/
http://www.darshanams.blogspot.com/