Basics

MBR

The Master Boot Record (MBR) is a table located on the first addressable logical cylinder (address 0 on the disk, if you will).
It is a 512-byte structure that describes the partition layout (how many partitions, which one holds the primary operating system, etc.) but also contains executable code that passes control to the Boot sector of the primary partition, which in turn loads the operating system.

MFT

An NTFS partition starts with the boot sector table that loads the Windows operating system75. It is then followed by the Master File Table, a global index holding metadata of all files and directories in the partition.Each file or directory is assigned a single entry in the MFT table holding its attributes (creation, modification and access time, block addresses, access permission, files in the folder, etc.).

Investigation Methodology

When performing an investigation on a disk, all we need is to parse the MFT to understand what exactly happened on the disk at the time of the attack: which files were modified, created, hidden, etc. The main advantage of directly parsing the MFT over simply mounting the partition using regular tools (mount on Linux) is to be able to inspect every corner of the sectors allocated to the system. We can thus retrieve deleted files, detect hidden data (Alternate Data Streams), check the MFT's integrity, inspect bad sectors, get slack space, etc.

Analyzing Disks with SeluthKit

Windows

Parsing the disk image

```
mmls disk.image
```

In case you are analyzing a disk image dump for a virtual machine then you need to specify the options [-i afflib]

```
mmls -i afflib disk.vmdk
```

Below is a sample output that highlights the partition that holds the operating system simply because the first one is too small and the last one is too big.

```
      Slot      Start        End          Length       Description
000:  Meta      0000000000   0000000000   0000000001   Primary Table (#0)
001:  -------   0000000000   0000002047   0000002048   Unallocated
002:  000:000   0000002048   0000718847   0000716800   NTFS / exFAT (0x07)
003:  000:001   0000718848   0083884031   0083165184   NTFS / exFAT (0x07)
004:  000:002   0083884032   1060446534   0976562502   NTFS / exFAT (0x07)
```

Next we analyze the MFT of the previously highlighted partition

```
fls -i afflib -o 718848 disk.vmdk
```

[718848] is the start of the disk based on the previous screenshot. A sample output is below

```
r/r    1-128-1 :      $MFTMirr
[…]
d/d   58-144-1 :      PerfLogs
d/d   59-144-6 :      Program Files
d/d   78-144-6 :      Program Files (x86)
d/d   94-144-6 :      ProgramData
d/d  137-144-5 :      Users
r/r  182-144-5 :      Windows
[…]
```

Value / means the entry was deleted
The second column holds unique identifiers called

inodes.
To walk down the directory or browser files, we need to supply the inode number for each directory.
Let's say we want to browser [users] directory, we would take its indoe from above screenshot and issue the below command

```
fls -i afflib -o 718848 disk.vmdk 137-144-5
```

Once we narrow down the choice on a file, we can retrieve its content with [icat] command providing is inode number

```
icat -i afflib -o 718848 sv0088_disk.vmdk 181620-144-1 > temp.pdt
```

See below

```
d/d    68742-144-1 :   Application Data
d/d    68744-144-1 :   History
d/d    68680-144-1 :   Microsoft
d/d    68679-144-1 :   Temp
r/r   181620-144-1 :   temp.pdt
```

We can also retrieve the contents of the MFT table to inspect last access time, last creation time and

last modification time of every file. This useful if you are investigating if an attacker managed to exfilterate corporate data.
First we dump its contents into a file

```
icat -i afflib -o [disk-inode] disk.vmdk
[MFT-Inode] > mft_table.mft
```

Then we parse the time data along with full file paths using the below command. This creates an csv output file.

```
analyzeMFT.py -f mft_table.mft -o
output_sv0088.csv -progress --bodyfull
```

Record Number	Parent File Rec. #	Filename #1	Std Info Access date
1060	1060	/Board/Business/Shareholders/list_significant_sharehold	2017-01-31 22:46:12.260544
1061	1061	/Board/Business/Shareholders/new_offering.pdf	2017-01-31 22:46:12.360544
1062	1062	/Board/Business/Shareholders/new_offering_v1.1.pdf	2017-01-31 22:46:12.560545
1063	1063	/Board/Business/Shareholders/zz_draft_revenues2017.p	2017-01-31 22:46:12.760546
1064	1064	/Board/Business/Takeover_2018/companies.xlsx	2017-01-31 15:46:13.260547
1065	1065	/Board/Business/Takeover_2018/companies_revenues	2017-01-31 22:46:12.360548
1066	1066	/Board/Business/Takeover_2018/Directors_approval.pdf	2017-01-31 22:46:12.560549
1067	1067	/Board/Business/Takeover_2018/HR_validation	2017-01-31 22:46:12.760550
1068	1068	/Board/Business/Takeover_2018/risks_A1	2017-01-31 22:46:13.260551

Linux

Parsing the disk image

```
mmls disk.image
```

In case you are analyzing a disk image dump for a virtual machine then you need to specify the options [-i afflib]

```
mmls -i afflib disk.vmdk
```

Like in Windows, we get a list of partitions along with their offsets (In Windows we called them Inodes).
Simply we can dump the partition in question using below commands

```
mkdir /mnt/disk/
mount -o ro,loop,offset=1048576 disk.vmdk /mnt/disk/
```

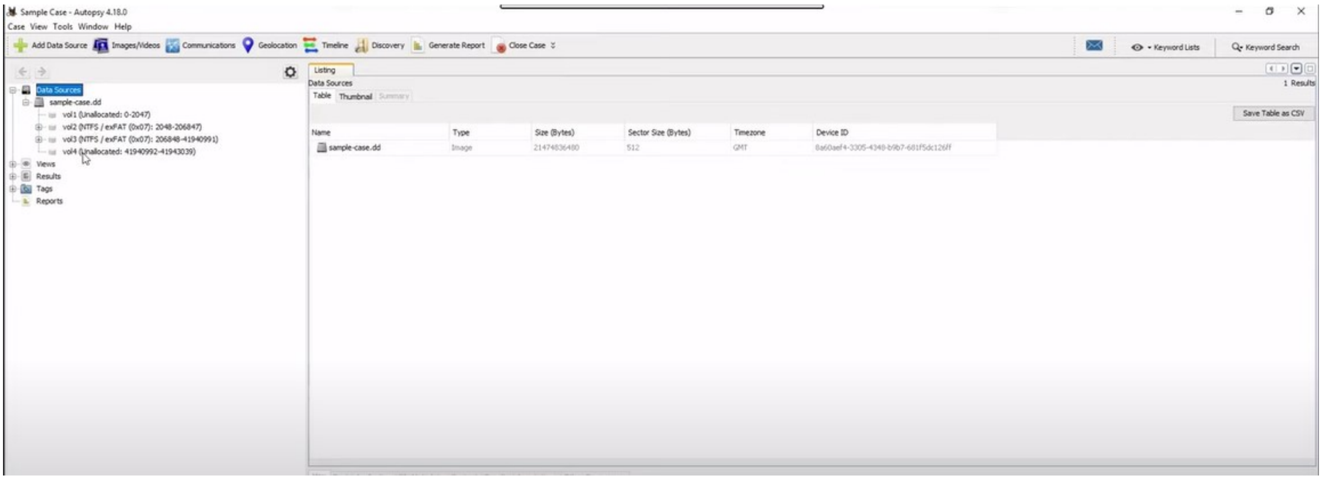
[ro] read only

Analyzing Disks with Autopsy

Getting started

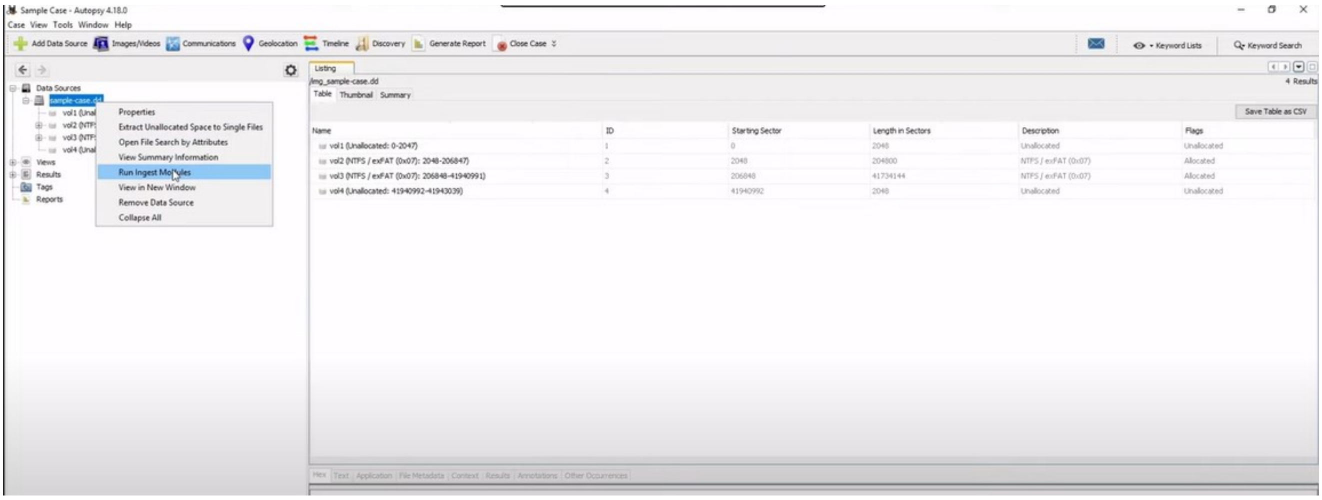
In Autopsy, we start by creating a new case or opening an already saved case. You can do that easily by following the wizard that pops-up once you open the program.

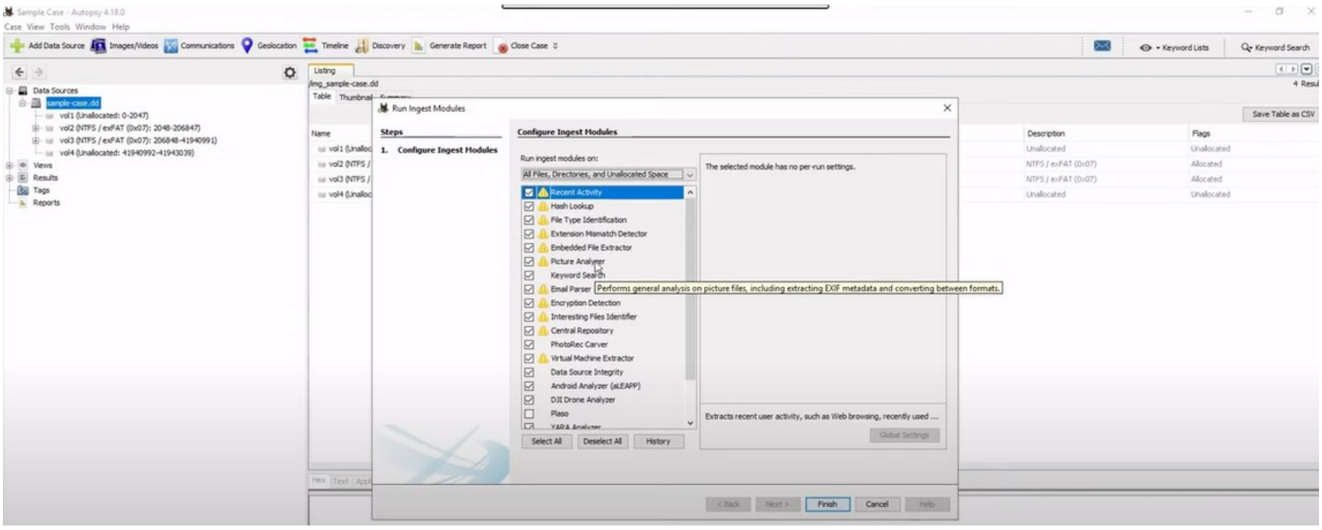
Below you see the disk that is attached to the case along with its partitions



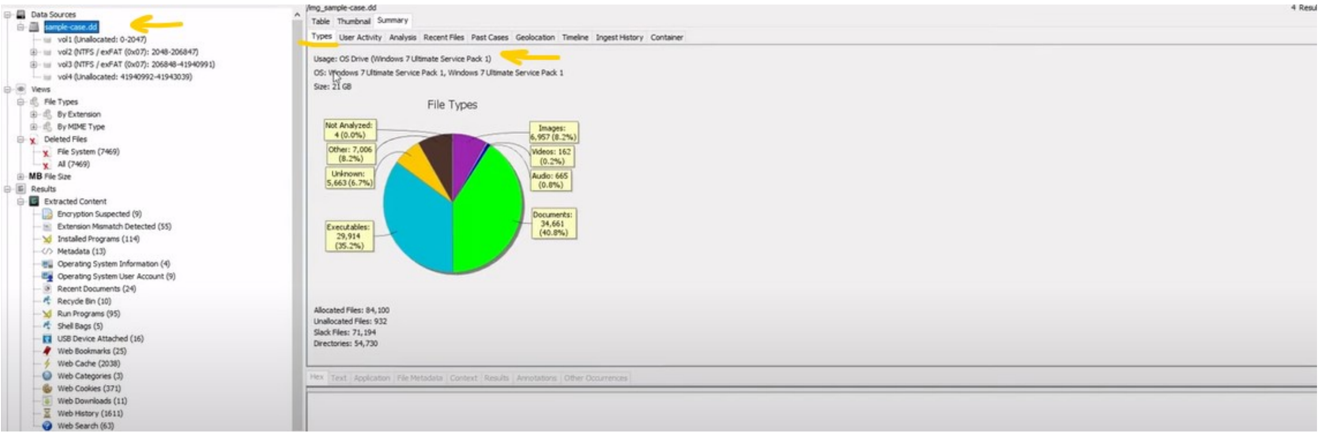
Analyzing disk artifacts

Analyzing disk artifacts can be accomplished by running ingest modules as you see below. Ingest modules appear under the [results] section in the left tree pane.

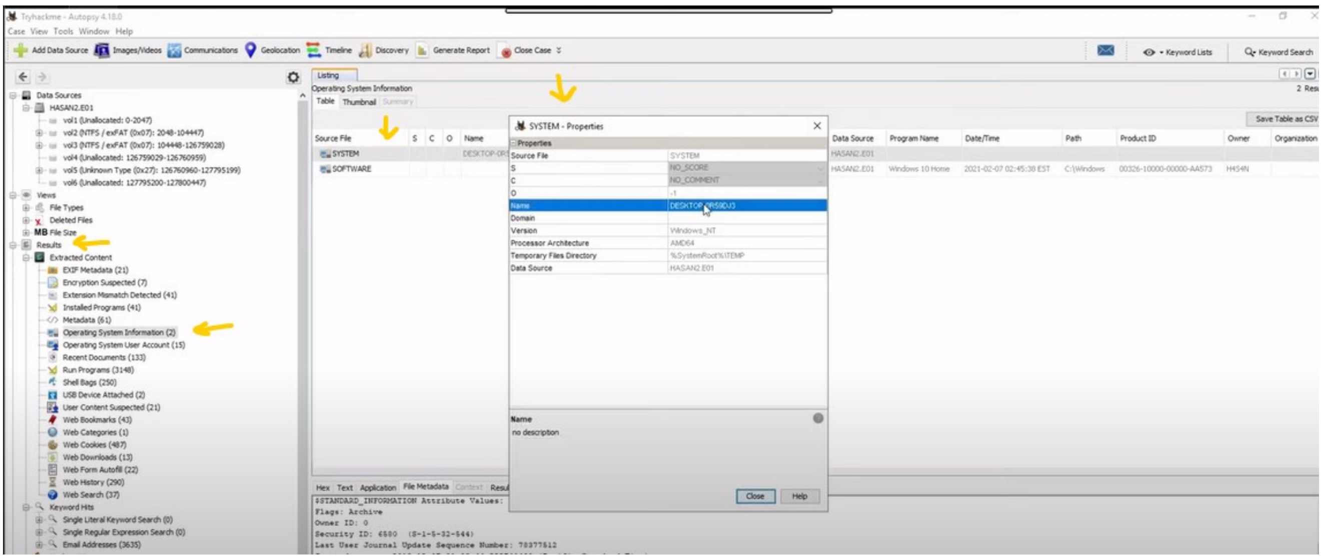




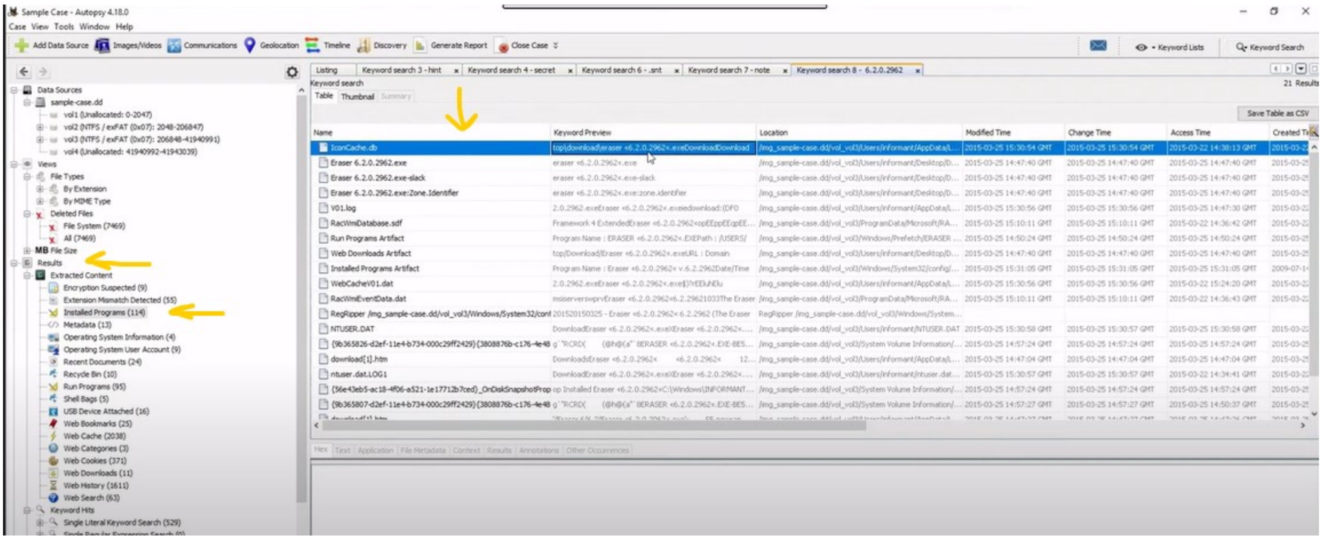
Extracting OS Data



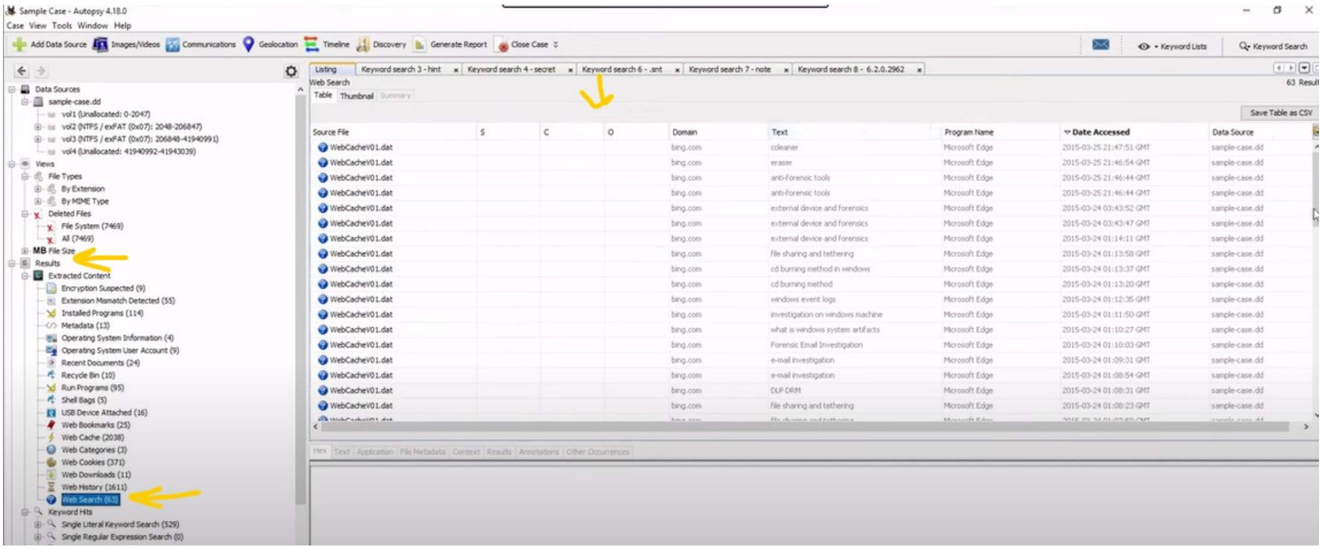
Or



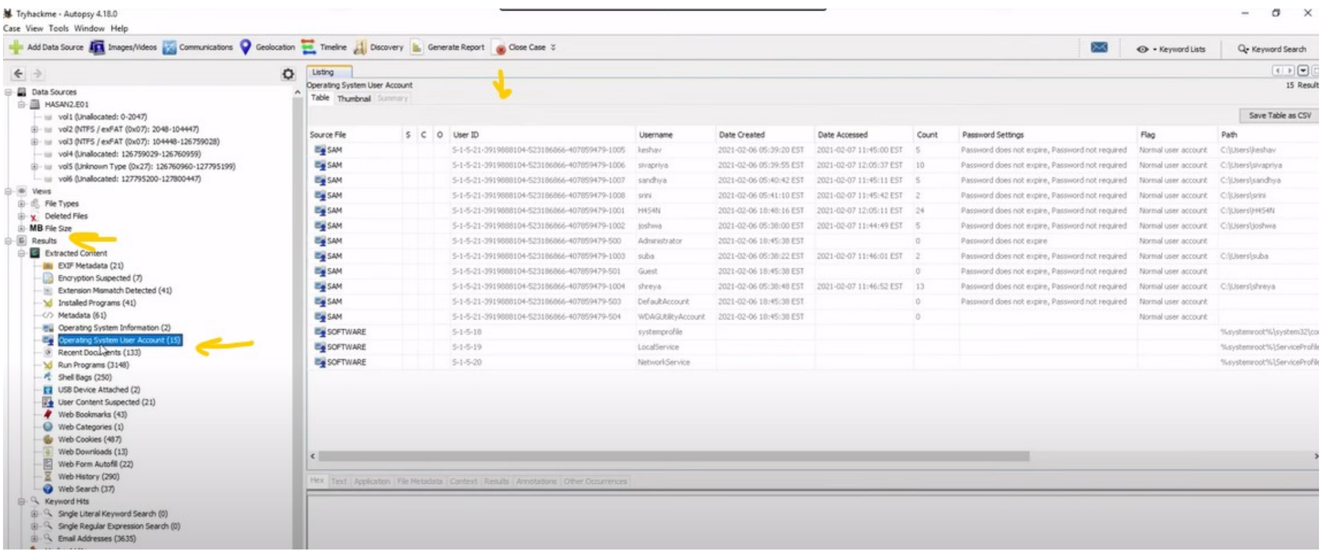
Viewing Installed Programs



Extracting web activity



Extrating user accounts



Extracting Images