

PRACTICAL 1

AIM: Static analysis of malware samples using disassemblers such as IDA Pro or Ghidra.

Tools Required:

1. IDA Pro (Free Version)
2. A sample malware binary (e.g., DarkComet)
3. A safe analysis environment (Virtual Machine)

Procedure:

Step 1: Setting Up a Safe Environment.

1. Use a virtual machine with no network access to analyze malware safely.
2. Install IDA Free Version on the analysis system.
3. Obtain a known malware sample for analysis.(e.g., DarkComet)

Step 2: Loading the Malware in IDA Free Version.

1. Open IDA Free Version and load the malware executable.
2. Let the disassembler analyze the binary and generate assembly code.

Step 3: Identifying Key Components.

1. Look for the main function or entry point of the malware.
2. Identify API calls related to file access, network comm's, or registry changes.
3. Check for suspicious strings, obfuscation techniques, and encryption routines.

Step 4: Understanding the Malware's Behavior.

1. Analyze function calls and code flow to determine the malware's intent.
2. Identify hardcoded IP addresses, domains, or commands that indicate communication with a C2 server.
3. Check for anti-debugging or anti-analysis techniques used by the malware.

Step 5: Documenting Findings.

1. Summarize the malware's capabilities (e.g., keylogging, file encryption, data exfiltration).
2. Report the indicators of compromise (IOCs) such as file hashes, API calls, and network connections.
3. Suggest mitigation techniques to prevent infection and spread.

Practicla Demo:

Step 1: Download and Install IDA Free Version Into Kali Linux and It's License File and Put That File Into That IDA Installed Folder To Detect and Run IDA Free Version.

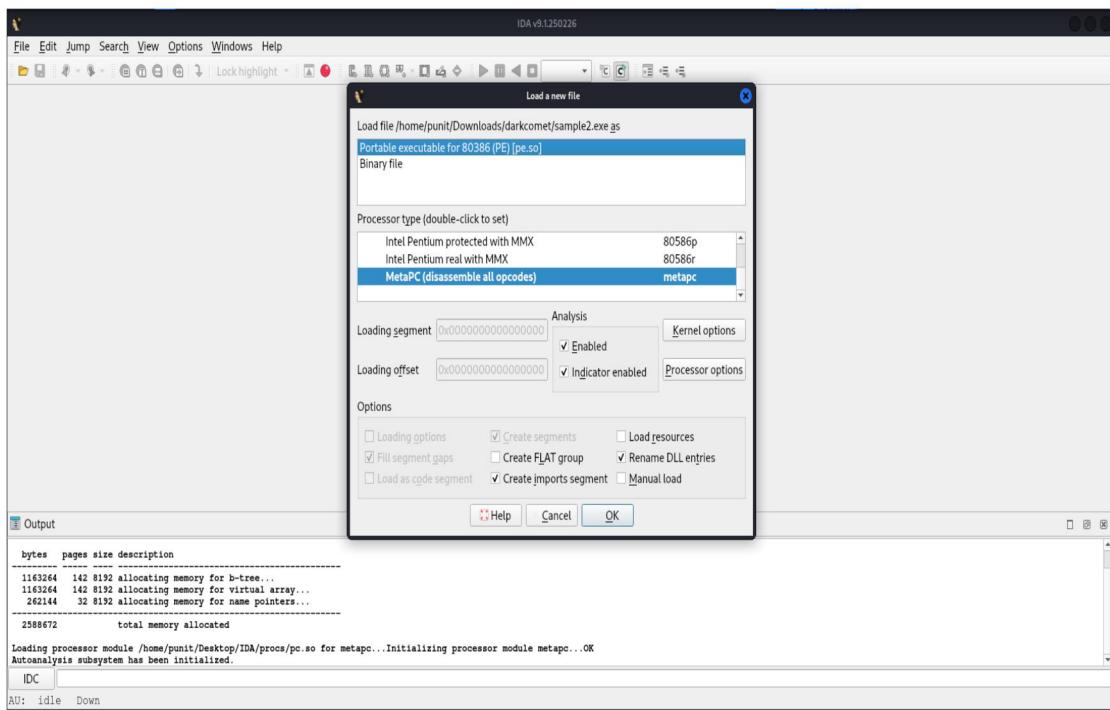
Command To Run The IDA Software:

./ida



```
punit@kali:~/Downloads
File Actions Edit View Help
(punit@kali) [~/Downloads]
$ ls
ida-free-pc_91_x64linux.run idafree_96-183A-2ED6-C9.hexlic
(punit@kali) [~/Downloads]
$ chmod +x ida-free-pc_91_x64linux.run
(punit@kali) [~/Downloads]
$ ./ida-free-pc_91_x64linux.run
(punit@kali) [~/Downloads]
$ ./ida-free-pc_91_x64linux.run
(punit@kali) [~/Downloads]
$ mv idafree_96-183A-2ED6-C9.hexlic /home/punit/Desktop/IDA/
(punit@kali) [~/Downloads]
```

Step 2: After Running The IDA Tool Choose New File and Select The Malware Sample For Static Analysis. (**Darkcomet -> Sample2.exe**)



OutPut:

```

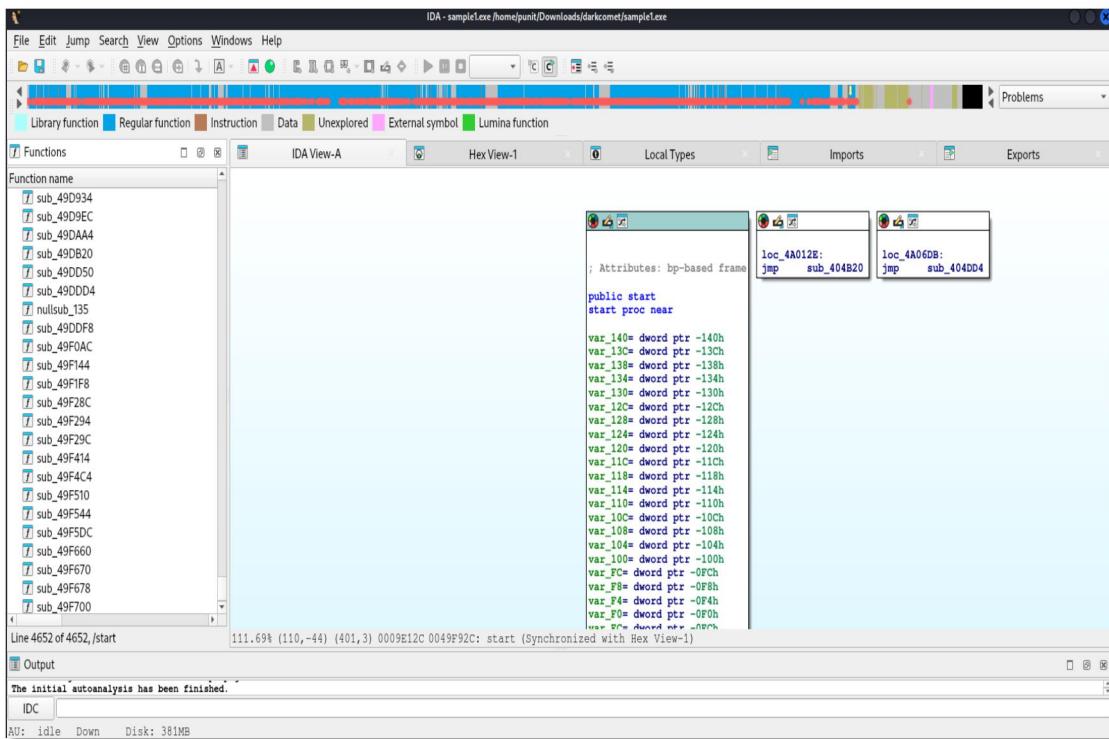
Directory /tmp/ida does not exist, creating...
Possible file format: Portable executable for 80386 (PE) (/home/punit/Desktop/IDA/loaders/pe.so)

bytes    pages   size  description
-----
1163264   142 8192 allocating memory for b-tree...
1163264   142 8192 allocating memory for virtual array...
262144     32 8192 allocating memory for name pointers...
-----
2588672          total memory allocated

Loading processor module /home/punit/Desktop/IDA/procs/pc.so for metapc...Initializing processor module metapc...OK
Autoanalysis subsystem has been initialized.
Loading file '/home/punit/Downloads/darkcomet/sample2.exe' into database...
Detected file format: Portable executable for 80386 (PE)
Assuming __cdecl calling convention by default
 0. Creating a new segment (00401000-00481000) ... ... OK
 1. Creating a new segment (00481000-004C7000) ... ... OK
Reading imports directory...
Type library 'mssdk_win7' loaded. Applying types...
Types applied to 0 names.
Plan FLIRT signature: SEH for vc7-14
Marking typical code sequences...
Flushing buffers, please wait...ok
File '/home/punit/Downloads/darkcomet/sample2.exe' has been successfully loaded into the database.
Flushing buffers, please wait...ok
Hex-Rays Cloud Decompiler plugin has been loaded (v9.1.0.250226)
The decompilation hotkey is F5.
Please check the Edit/Plugins menu for more information.
Using FLIRT signature: SEH for vc7-14
Propagating type information...
Function argument information has been propagated
The initial autoanalysis has been finished.

```

Step 3: After Scan The Malware Sample Into IDA Free Version You Get The Function's and The IDA View For Analysis Into Assembly Language and Hexa-View and You Can See The Graph and Connected Node's That Show's The Working of The Executable File and The Code Structure and Functionality of The Executable File.



Conclusion:

Disassemblers like IDA Free Version play a vital role in malware analysis, offering deep insights into the code structure and functionality. While static analysis is effective in identifying malicious behavior, combining it with other techniques, including dynamic analysis, provides a more complete picture of the threat landscape.

PRACTICAL 2

AIM: Dynamic analysis of malware samples using a virtual machine and tools such as Process Monitor, Wireshark, and Sandboxie.

Dynamic Analysis in AnyRun Sandbox:

Sample 1:

General Info

File name: sample1.exe
Full analysis: <https://app.any.run/tasks/d50c6c7c-ed46-4d5b-a09d-e1ba327fb077>
Verdict: Malicious activity / DarkComet
Threats: DarkComet RAT is a malicious program designed to remotely control or administer a victim's computer, steal private data and spy on the victim.
Remote Access Trojan
Remote access trojans (RATs) are a type of malware that enables attackers to establish complete to partial control over infected computers. Such malicious programs often have a modular design, offering a wide range of functionalities for conducting illicit activities on compromised systems. Some of the most common features of RATs include access to the users' data, webcam and keystrokes. This malware is often distributed through phishing emails and links.

Analysis date: March 07, 2025 at 01:07:56
OS: Windows 10 Professional (build: 19045, 64 bit)
Tags: darkcomet, rat
Indicators:
MIME: application/vnd.microsoft.portable-executable
File info: PE32 executable (GUI) Intel 80386, for MS Windows, 9 sections
MD5: C1F9C6B996C64CEFD3AE5B04396BE7
SHA1: 6C981DC9F9CCA212895A747A3ED4FECB8B681DF880
SHA256: 988B710B0D6547494DA52784E3BDC1605AF499194C181312996D31CF063D93612
SSDeep: 12288:5zF5pMx2ueAY91N7qcE5nfmbjyaMLVeb59y40xStejNb2iZ+Bi4bYnNvXZt:5zfMx2ug91N7LE2Fm8jaZVtqZZt8rn9

Software environment set and analysis options

Launch configuration

Task duration: 60 seconds	Heavy Evasion option: off	Network geolocation: off
Additional time used: none	MITM proxy: off	Privacy: Public submission
Fakemnet option: off	Route via Tor: off	Autoconfirmation of UAC: on
Network: on		

MALICIOUS	SUSPICIOUS	INFO
Changes the autorun value in the registry • sample1.exe (PID: 7464) • msdcsc.exe (PID: 7588)	Starts CMD.EXE for commands execution • sample1.exe (PID: 7464)	Checks supported languages • sample1.exe (PID: 7464) • msdcsc.exe (PID: 7588)
Changes the login/logout helper path in the registry • sample1.exe (PID: 7464)	Executable content was dropped or overwritten • sample1.exe (PID: 7464)	Reads the computer name • sample1.exe (PID: 7464) • msdcsc.exe (PID: 7588)
DARKCOMET mutex has been found • msdcsc.exe (PID: 7588) • iexplore.exe (PID: 7664)	Reads security settings of Internet Explorer • sample1.exe (PID: 7464)	The sample compiled with english language support • sample1.exe (PID: 7464)
	Starts itself from another location • sample1.exe (PID: 7464)	Process checks computer location settings • sample1.exe (PID: 7464)
	Executing commands from a *.bat* file • sample1.exe (PID: 7464)	Create files in a temporary directory • sample1.exe (PID: 7464)
	Uses ATTRIB.EXE to modify file attributes • cmd.exe (PID: 7552) • cmd.exe (PID: 7512)	Creates files or folders in the user directory • BackgroundTransferHost.exe (PID: 5452)
		Reads security settings of Internet Explorer • BackgroundTransferHost.exe (PID: 7172) • BackgroundTransferHost.exe (PID: 7496) • BackgroundTransferHost.exe (PID: 4400) • BackgroundTransferHost.exe (PID: 5452) • BackgroundTransferHost.exe (PID: 7300)
		Reads the software policy settings • BackgroundTransferHost.exe (PID: 5452)
		Checks proxy server information • BackgroundTransferHost.exe (PID: 5452)
		Connects to unusual port • iexplore.exe (PID: 7664)

Malware configuration

No Malware configuration.

Sample 2:

ANYRUN
INTERACTIVE MALWARE ANALYSIS

General Info

URL: [smartonestartpdf.com](https://app.any.run/tasks/a969af83-94b6-4f6e-bd0f-3339e4bd452a)

Full analysis: <https://app.any.run/tasks/a969af83-94b6-4f6e-bd0f-3339e4bd452a>

Verdict: Malicious activity

Threats: Adware

Adware is a form of malware that targets users with unwanted advertisements, often disrupting their browsing experience. It typically infiltrates systems through software bundling, malicious websites, or deceptive downloads. Once installed, it may track user activity, collect sensitive data, and display intrusive ads, including pop-ups or banners. Some advanced adware variants can bypass security measures and establish persistence on devices, making removal challenging. Additionally, adware can create vulnerabilities that other malware can exploit, posing a significant risk to user privacy and system security.

Loader

A loader is malicious software that infiltrates devices to deliver malicious payloads. This malware is capable of infecting victims' computers, analyzing their system information, and installing other types of threats, such as trojans or stealers. Criminals usually deliver loaders through phishing emails and links by relying on social engineering to trick users into downloading and running their executables. Loaders employ advanced evasion and persistence tactics to avoid detection.

Stealer

Stealers are a group of malicious software that are intended for gaining unauthorized access to users' information and transferring it to the attacker. The stealer malware category includes various types of programs that focus on their particular kind of data, including files, passwords, and cryptocurrency. Stealers are capable of spying on their targets by recording their keystrokes and taking screenshots. This type of malware is primarily distributed as part of phishing campaigns.

Analysis date: March 07, 2025 at 01:07:09

OS: Windows 10 Professional (build: 19045, 64 bit)

Tags: [Adware](#), [AdvancedInstaller](#), [loader](#), [stealer](#)

Indicators:

MD5: 5C5D156304ECD13621ECC05ADE24D

SHA1: A3A0BA2C7B4029EFBF9B2166E95F2C2022E1C6F9

SHA256: E309D5127E148557327CB823455CF2787D08AB1B616FBE92F37FF904DBE65D3B

SSDeep: 3.GRAaBdi:GAJ

Software environment set and analysis options

Launch configuration

Task duration: 300 seconds	Heavy Evasion option: off	Network geolocation: off
Additional time used: 240 seconds	MITM proxy: off	Privacy: Public submission
Fakemnet option: off	Route via Tor: off	Autoconfirmation of UAC: on
Network: on		

Version: 122.0.2365.59

2908 'C:\Program Files (x86)\Microsoft\Edge\Application\msedge.exe' --type=crashpad-handler --user-data-dir=C:\Users\admin\AppData\Local\Microsoft\Edge\User Data\prefetch\4 -monitor-self-annotation-type=crashpad-handler -database=C:\Users\admin\AppData\Local\Microsoft\Edge\User Data\Crahpad\annotation\IsOfficialBuild\1 -annotation=prod-Microsoft Edge version=122.0.2365.70 -annotation=seed-version=C:\Program Files (x86)\Microsoft\Edge\Application\msedge.exe" -annotation=plat=Win64" -annotation=prod=Microsoft Edge" -annotation=ver=122.0.2365.59 -initial-client-data=0x314,0x318,0x31c,0x30c,0x324,0x7ffc887e5fd8,0x7ffc887e5fd0,0x7ffc887e5fd0 e5fe4,0x7ffc887e5fd0

Information

User: admin	Company: Microsoft Corporation
Integrity Level: MEDIUM	Description: Microsoft Edge
Version: 122.0.2365.59	

7312 'C:\Program Files (x86)\Microsoft\Edge\Application\msedge.exe' --gpu-process=no-appcompat-clear -gpu-preferences=WAAAAAAAADAAAMAAMAAAAAAAAMAAAAAAAAB gAAAAAAAAMAAAAAAAAGAAAAAAAEEAAAAAAAAMAAAAAAAAB gAAAAAAAAMAAAAAAAAGAAAAAAAAYAAAAAAAAGAAAA AAAACAAAAAAAIAAAAAAAAIA== -mojo-platform-channel-handle=230 -field-trial-handle=3234,150276873632257605,5236874692018770281,26 2144 -variations-seed-version /prefetch:2

Information

User: admin	Company: Microsoft Corporation
Integrity Level: LOW	Description: Microsoft Edge
Version: 122.0.2365.59	

7320 'C:\Program Files (x86)\Microsoft\Edge\Application\msedge.exe' --type=utility-sub-type=network,mojom NetworkService lang=en-US -service-sandbox:type=none -no-appcompat-clear -mojo-platform-channel-handle=2420 -field-trial-handle=3234,150276873632257605,5236874692018770281,26 2144 -variations-seed-version /prefetch:3

Information

User: admin	Company: Microsoft Corporation
Integrity Level: MEDIUM	Description: Microsoft Edge
Version: 122.0.2365.59	

7344 'C:\Program Files (x86)\Microsoft\Edge\Application\msedge.exe' --type=utility-sub-type=storage,mojom StorageService lang=en-US -service-sandbox:type=service -no-appcompat-clear -mojo-platform-channel-handle=2640 -field-trial-

Conclusion:

Dynamic analysis of malware using a virtual machine (VM) environment, combined with tools like Process Monitor, Wireshark, and Sandboxie, offers a powerful and safe method for studying malicious software behavior in real time. By isolating the malware within a virtual environment, researchers can observe its interactions with the operating system and network without risking the host system. Process Monitor provides insight into file system and registry activity, Wireshark captures and analyzes network traffic, and Sandboxie allows for contained execution and rollback capabilities.

PRACTICAL 3

AIM: Memory analysis of malware samples using tools such as Volatility.

Memory analysis is a crucial technique in malware forensics that involves examining a system's volatile memory (RAM) to detect and analyze malicious activity. Tools like Volatility, an open-source memory forensics framework, are widely used for this purpose. Volatility can extract detailed information such as running processes, loaded DLLs, network connections, open files, and evidence of code injection or rootkits, even if malware attempts to hide its presence or remove traces from disk.

In the context of malware analysis, memory forensics allows analysts to capture the actual behavior of malicious samples during execution. By loading a memory dump into Volatility and applying plugins (like pslist, malfind, dlllist, netscan, cmdline, etc.), investigators can trace how the malware operates, what processes it spawns, and how it manipulates the system. This analysis provides deep insights into advanced persistent threats (APTs), fileless malware, and zero-day attacks.

Step 1:

Run chmod +x and .sh volatility file

```
(root㉿kali)-[~/home/kali/Downloads/volatility-installation]
# chmod +x volatility-installation.sh
# ./volatility-installation.sh
Found vol.py at: /opt/volatility/setup.py
Volatility Foundation Volatility Framework 2.6.1
** Failed to import volatility.plugins.registry.shutdown (ImportError: No module named Crypto.Hash)
** Failed to import volatility.plugins.getservicesids (ImportError: No module named Crypto.Hash)
** Failed to import volatility.plugins.timeliner (ImportError: No module named Crypto.Hash)
** Failed to import volatility.plugins.malware.apiphooks (NameError: name 'distorm3' is not defined)
** Failed to import volatility.plugins.servicendiff (ImportError: No module named Crypto.Hash)
** Failed to import volatility.plugins.registry.userassist (ImportError: No module named Crypto.Hash)
** Failed to import volatility.plugins.getsids (ImportError: No module named Crypto.Hash)
** Failed to import volatility.plugins.teaudit (ImportError: No module named Crypto.Hash)
** Failed to import volatility.plugins.registry.shellbags (ImportError: No module named Crypto.Hash)
** Failed to import volatility.plugins.evlogs (ImportError: No module named Crypto.Hash)
** Failed to import volatility.plugins.shimcache (ImportError: No module named Crypto.Hash)
** Failed to import volatility.plugins.registry.dumpregistry (ImportError: No module named Crypto.Hash)
** Failed to import volatility.plugins.registry.lsadump (ImportError: No module named Crypto.Hash)
** Failed to import volatility.plugins.malware.threads (NameError: name 'distorm3' is not defined)
** Failed to import volatility.plugins.malware.apiphooks_kernel (ImportError: No module named distorm3)
** Failed to import volatility.plugins.registry.lsadump (ImportError: No module named Crypto.Hash)
** Failed to import volatility.plugins.malware.svscan (ImportError: No module named distorm3)
** Failed to import volatility.plugins.malware.svscan (ImportError: No module named Crypto.Hash)
** Failed to import volatility.plugins.registry.auditpol (ImportError: No module named Crypto.Hash)
** Failed to import volatility.plugins.sddt (NameError: name 'distorm3' is not defined)
** Failed to import volatility.plugins.registry.registryapi (ImportError: No module named Crypto.Hash)
** Failed to import volatility.plugins.mac.apiphooks (ImportError: No module named distorm3)
** Failed to import volatility.plugins.malware.getenvs (ImportError: No module named Crypto.Hash)
ERROR : volatility.debug : You must specify something to do (try -h)
python2-dev is available. Installing...
Reading package lists... Done
Building dependency tree... Done
Reading state information... Done
The following additional packages will be installed:
libpython2.7-dev libpython2.7-dev linux-image-6.12.20-amd64 linux-image-amd64 python2.7-dev
Suggested packages:
linux-doc-6.12 debian-kernel-handbook
The following NEW packages will be installed:
libpython2.7-dev libpython2.7-dev linux-image-6.12.20-amd64 python2-dev python2.7-dev
The following packages will be upgraded:
linux-image-amd64
1 upgraded, 6 newly installed, 0 to remove and 459 not upgraded.
Need to get 109 MB/112 MB of archives.
After this operation, 128 MB of additional disk space will be used.
Get:1 http://Kali.download/kali kali-rolling/main amd64 linux-image-6.12.20-amd64 amd64 6.12.20-1kalil1 [109 MB]
```

Step 2:

Move sample.vmem file in opt/volatility directory and get the image info of the sample file using

Python2 vol.py -f sample.vmem imageinfo:

```
(root㉿kali)-[~/home/kali/Downloads]
# mv sample.vmem /opt/volatility
(root㉿kali)-[~/home/kali/Downloads]
# cd ..
[root@kali ~]# ./vol.py -f sample.vmem imageinfo
Volatility Foundation Volatility Framework 2.6.1
INFO    : volatility.debug : Determining profile based on KDBG search ...
INFO    : volatility.debug : Suggested Profile(s) : WinXPSP2x86, WinXPSP3x86 (Instantiated with WinXPSP2x86)
INFO    : volatility.debug :   AS Layer 1 : FileAddressSpaceMemoryPae (Kernel AS)
INFO    : volatility.debug :   AS Layer 2 : FileAddressSpace (/opt/volatility/sample.vmem)
INFO    : volatility.debug :   PAE type : PAE
INFO    : volatility.debug :   DTB : 0x319000L
INFO    : volatility.debug :   KDBG : 0x80545aa0L
INFO    : volatility.debug :   Number of Processors : 1
INFO    : volatility.debug :   Image Type (Service Pack) : 3
INFO    : volatility.debug :   Image CPU : 0xFdf0000L
INFO    : volatility.debug :   KUSER_SHARED_DATA : 0xFdf00000L
INFO    : volatility.debug :   Image date and time : 2011-06-03 04:31:36 UTC+0000
INFO    : volatility.debug :   Image local date and time : 2011-06-03 04:31:36 -0400
[root@kali ~]#
```

Step 3:

Scan pslist using profile of sample file:

Offset(V)	Name	PID	PPID	Thds	Hnds	Sess	Wow64	Start	Exit
0x823c880	System	4	0	59	403	—	0	0 2010-10-29 17:08:53 UTC+0000	
0x820f020	smss.exe	376	0	3	19	—	0	0 2010-10-29 17:08:54 UTC+0000	
0x820f020	csrss.exe	600	376	11	305	0	0	0 2010-10-29 17:08:54 UTC+0000	
0x81d5650	vmwaretray.exe	624	376	19	570	0	0	0 2010-10-29 17:08:54 UTC+0000	
0x82073020	services.exe	668	624	21	431	0	0	0 2010-10-29 17:08:54 UTC+0000	
0x81e70020	lsass.exe	680	624	19	342	0	0	0 2010-10-29 17:08:54 UTC+0000	
0x823315d8	vmacthlp.exe	844	668	1	25	0	0	0 2010-10-29 17:08:55 UTC+0000	
0x823315d8	svchost.exe	850	668	17	103	0	0	0 2010-10-29 17:08:55 UTC+0000	
0x81e1d1a0	svchost.exe	948	668	13	312	0	0	0 2010-10-29 17:08:55 UTC+0000	
0x822843e8	svchost.exe	1032	668	61	1169	0	0	0 2010-10-29 17:08:55 UTC+0000	
0x81e1b8e8	svchost.exe	1088	668	5	80	0	0	0 2010-10-29 17:08:55 UTC+0000	
0x81ff7f020	svchost.exe	1200	668	14	197	0	0	0 2010-10-29 17:08:55 UTC+0000	
0x81e8b800	spoolsv.exe	1412	668	10	118	0	0	0 2010-10-29 17:08:56 UTC+0000	
0x81e1d1a0	svchost.exe	1508	668	5	108	0	0	0 2010-10-29 17:08:56 UTC+0000	
0x81f5e52d0	vmtoolsd.exe	1664	668	0	284	0	0	0 2010-10-29 17:09:05 UTC+0000	
0x821a0568	VMUpgradeHelper	1816	668	3	96	0	0	0 2010-10-29 17:09:08 UTC+0000	
0x8205ad0	alg.exe	188	668	6	107	0	0	0 2010-10-29 17:09:09 UTC+0000	
0x820c7e8	explorer.exe	1196	1728	16	582	0	0	0 2010-10-29 17:11:49 UTC+0000	
0x820c7e8	wsmfilter.exe	2064	1728	1	28	0	0	0 2010-10-29 17:11:49 UTC+0000	
0x81e68f78	VMNC.exe	324	1196	7	54	0	0	0 2010-10-29 17:11:49 UTC+0000	
0x81f5d0	VMwareTray.exe	1912	1196	1	50	0	0	0 2010-10-29 17:11:50 UTC+0000	
0x81e6b660	VMwareUser.exe	1356	1196	9	251	0	0	0 2010-10-29 17:11:50 UTC+0000	
0x82279900	lmhosts.exe	756	1196	4	110	0	0	0 2010-10-29 17:11:50 UTC+0000	
0x81e1d1a0	svchost.exe	976	1832	3	133	0	0	0 2010-10-29 17:12:03 UTC+0000	
0x81c543a0	Procmon.exe	668	1196	13	189	0	0	0 2011-06-03 04:25:56 UTC+0000	
0x81f5390	wmiptvse.exe	1872	856	5	134	0	0	0 2011-06-03 04:25:58 UTC+0000	
0x81c498c0	lsass.exe	868	668	2	23	0	0	0 2011-06-03 04:26:55 UTC+0000	
0x81c47c00	lsass.exe	1928	668	4	65	0	0	0 2011-06-03 04:26:55 UTC+0000	
0x81c0cd0	cmd.exe	968	1664	0	—	0	0	0 2011-06-03 04:31:35 UTC+0000	2011-06-03 04:31:36 UTC+0000
0x81f14938	ipconfig.exe	384	968	0	—	0	0	0 2011-06-03 04:31:35 UTC+0000	2011-06-03 04:31:36 UTC+0000

Step 4:

Run psscan using the profile of sample file:

Conclusion:

Memory analysis using tools like Volatility is a powerful method for uncovering hidden or sophisticated malware. Unlike traditional disk-based forensics, it allows real-time behavioral insights and detection of in-memory artifacts that may not leave permanent traces. By leveraging Volatility's extensive plugin system, analysts can perform a detailed post-compromise investigation, extract indicators of compromise (IOCs), and better understand the techniques employed by threat actors. In modern cybersecurity workflows, memory forensics has become an indispensable skill for incident response, reverse engineering, and threat hunting.

PRACTICAL 4

AIM: Analyzing malicious documents such as PDF files and Office documents using tools such as PDFStreamDumper, oledump, and OfficeMalScanner.

Analyzing malicious documents such as PDFs and Office files is a critical task in cybersecurity, helping identify embedded threats like macros, JavaScript, or shellcode without executing the file. Tools like PDFStreamDumper, oledump.py, and OfficeMalScanner enable static analysis by inspecting the internal structure of documents, detecting suspicious elements, and extracting hidden content. While PDFStreamDumper and OfficeMalScanner are Windows-based tools that can run on Kali Linux using Wine, oledump.py is a Python-based tool that runs natively and is ideal for examining OLE files with embedded macros.

Step 1: Analyzing the PDF using PDFStreamDumper.

```
(kali㉿kali)-[~/Downloads/DidierStevensSuite]
$ python3 pdfid.py AWA_53.pdf

PDFid 0.2.10 AWA_53.pdf
PDF Header: %PDF-1.7
obj 110
endobj 110
stream 58
endstream 58
xref 1
trailer 1
startxref 1
/JS 23
/Encrypt 0
/OjStm 0
/J/S 0
/JavaScript 0
/OpenAction 0
/AcroForm 0
/JBIG2Decode 0
/RichMedia 0
/Launch 0
/EmbeddedFile 0
/XFA 0
/URI 0
/Colors > 2^24 0
```

Step 2: Analyzing The PDF File With pdf-parser.py

Kali - VMware Workstation

File Edit View VM Tabs Help || Type here to search

Library My Computer Kali MST2

Home KALI

(kali㉿kali) [~/Downloads/DidierStevensSuite]\$ python3 pdf-parser.py AMA_53.pdf

PDF Comment 'XPDF-1.7\r\n'

PDF Comment '%<+c2>\b3\x7d\x08\r\n'

obj 3 0

Type: Referencing;

<<

/Author (Patel Meet)

/Comments ()

/Company ()

/CreateDate "(D:20241022210836+15'38")"

/Creator (wPS Writer)

/Keywords ()

/ModDate "(D:20241022210836+15'38")"

/Producer ()

/SourceModified "(D:20241022210836+15'38")"

/Subject ()

/Title ()

/Trapped /False

>>

obj 9 0

Type: /XObject

Referencing:

Contains stream

<<

/BitsPerComponent 8

/ColorSpace /DeviceGray

/Filter /FlateDecode

/Height 9

/Length 391

/Matte [0 0]

/Subtype /Image

/Type /XObject

/Width 318

>>

obj 8 0

Type: /XObject

Conclusion:

Tools like PDFStreamDumper, oledump.py, and OfficeMalScanner are essential for safely analyzing potentially malicious documents without execution. They allow cybersecurity professionals to identify hidden macros, scripts, and exploits, helping to uncover the techniques used by attackers. When used together within a secure analysis environment, these tools provide valuable insights into document-based malware, aiding in threat detection, response, and the development of stronger security defenses.

PRACTICAL 5

AIM: Reverse engineering malware using reverse engineering frameworks such as Radare2.

Step 1:

Install Radare2 on your system using the following command:

- sudo apt install radare2

Step 2:

Set up a secure analysis environment (VM or sandbox) to avoid executing malicious code.

- Loading the Malware Sample in Radare2
 1. Open the terminal and navigate to the folder containing the malware binary.
 2. Run the following command to load the binary into Radare2:
 - r2 -AA malware_sample.exe
 -

The screenshot shows a terminal window titled 'root@kali: /home/punit/Downloads/darkcomet'. The user has run the command '# ls' which shows two files: 'sample1.exe' and 'sample2.exe'. The user then runs the command '# r2 -AA sample2.exe'. The output of this command is displayed in the terminal, showing various analysis steps and messages. These include:

- WARN: Relocs has not been applied. Please use `e bin.relocs.apply=true` or `e bin.cache=true` next time
- INFO: Analyze all flags starting with sym. and entry0 (aa)
- INFO: Analyze imports (af@@i)
- INFO: Analyze entrypoint (af@ entry0)
- INFO: Analyze symbols (af@@s)
- INFO: Analyze all functions arguments/locals (afva@@F)
- INFO: Analyze function calls (aac)
- INFO: Analyze len bytes of instructions for references (aar)
- INFO: Finding and parsing C++ vtables (avrr)
- INFO: Analyzing methods (af @@ method.*)
- INFO: Recovering local variables (afva@@F)
- INFO: Type matching analysis for all functions (aift)
- INFO: Propagate noreturn information (aanr)
- INFO: Scanning for strings constructed in code (/azs)
- INFO: Finding function preludes (aap)
- INFO: Enable anal.types.constraint for experimental type propagation

The terminal prompt '[0x004c69c0]>' is visible at the bottom.

Step 3:

Analyzing the Binary Structure

- Use the `i` command to display basic binary information:



```
File Actions Edit View Help
INFO: Enable anal.types.constraint for experimental type propagation
[0x004c69c0]> i
fd      3
file   sample2.exe
size    0x47400
humansz 285K
mode    r-x
format  pe
iornw   false
block   0x100
type    EXEC (Executable file)
arch    x86
badr   0x400000
binsz  291840
bintype pe
bits   32
canary  true
injprot false
retguard false
class   PE32
cmp.csum 0x0004cd01
compiled Mon Oct 31 01:52:14 2011
crypto  false
 endian little
havecode true
hdr.csum 0x00000000
laddr   0x0
lang    c
linenum true
lisms   true
machine i386
nx     false
os     windows
overlay false
cc     cdecl
pic    false
relocs  true
signed  false
sanitize false
static  false
```

- List the functions detected in the binary using: **afl**



```
Sanitize false
static false
stripped false
subsys Windows GUI
va    true
[0x004c69c0]> afl
0x004c69c0 53 443 entry0
0x004837a0 3 19 fcn.004837a0
0x00487a5e 1 36 fcn.00487a5e
0x004c3695 1 87 int.004c3695
0x00481c2b 3 33 fcn.00481c2b
0x0048379c 1 4 fcn.0048379c
[0x004c69c0]>
```

Step 4:

Disassembling and Identifying Code Flow

- Run the following command to extract readable strings: **iz**

```
[root@kali /home/punit/Downloads/darkcomet]
[Strings]
nth paddr      vaddr      len size section type      string
0  0x000469e8  0x004c79e8  11   24   .rsrc    utf16le \nCHANGEDATE
1  0x00046a00  0x004c7a00  6    14   .rsrc    utf16le CHIDED
2  0x00046a0e  0x004c7a0e  26   54   .rsrc    utf16le CHIDED\tCOMBOPATH\tDIRATTRIB
3  0x00046a44  0x004c7a44  33   68   .rsrc    utf16le DVCLAL\EDTDATE\EDTPATH\FILEATTRIB
4  0x00046a88  0x004c7a88  27   56   .rsrc    utf16le FWB\ENCODE\INSTALL\KEYNAME
5  0x00046ac0  0x004c7ac0  4    10   .rsrc    utf16le MELT
6  0x00046aca  0x004c7aca  43   88   .rsrc    utf16le MUTEX\NETDATA\OFFLINEK\PACKAGEINFO\PRSINST
7  0x00046b3a  0x004c7b3a  15   32   .rsrc    utf16le VS_VERSION_INFO
8  0x00046b96  0x004c7b96  14   30   .rsrc    utf16le StringFileInfo
9  0x00046bba  0x004c7bba  8    18   .rsrc    utf16le 040904b0
10 0x00046bd2  0x004c7bd2  8    18   .rsrc    utf16le Comments
11 0x00046b64  0x004c7b64  26   54   .rsrc    utf16le Remote Service Application
12 0x00046c22  0x004c7c22  11   24   .rsrc    utf16le CompanyName
13 0x00046c3c  0x004c7c3c  15   32   .rsrc    utf16le Microsoft Corp.
14 0x00046c62  0x004c7c62  15   32   .rsrc    utf16le FileDescription
15 0x00046c84  0x004c7c84  26   54   .rsrc    utf16le Remote Service Application
16 0x00046cc2  0x004c7cc2  11   24   .rsrc    utf16le FileVersion
17 0x00046cdc  0x004c7cdc  10   22   .rsrc    utf16le 1, 0, 0, 1
18 0x00046dfa  0x004c7dfa  12   26   .rsrc    utf16le InternalName
19 0x00046d14  0x004c7d14  8    18   .rsrc    utf16le MSRSAAPP
20 0x00046d2e  0x004c7d2e  14   30   .rsrc    utf16le LegalCopyright
21 0x00046d4c  0x004c7d4c  18   38   .rsrc    utf16le Copyright (C) 1999
22 0x00046d7a  0x004c7d7a  16   34   .rsrc    utf16le OriginalFilename
23 0x00046d9c  0x004c7d9c  11   24   .rsrc    utf16le MSRSAAP.EXE
24 0x00046dba  0x004c7dba  11   24   .rsrc    utf16le ProductName
25 0x00046dd4  0x004c7dd4  26   54   .rsrc    utf16le Remote Service Application
26 0x00046e12  0x004c7e12  14   30   .rsrc    utf16le ProductVersion
27 0x00046e30  0x004c7e30  10   22   .rsrc    utf16le 4, 0, 0, 0
28 0x00046e4e  0x004c7e4e  11   24   .rsrc    utf16le VarFileInfo
29 0x00046e6e  0x004c7e6e  11   24   .rsrc    utf16le Translation
30 0x00047100  0x004c8100  12   13   .rsrc    ascii  KERNEL32.DLL
31 0x0004710d  0x004c810d  12   13   .rsrc    ascii  advapi32.dll
32 0x0004711a  0x004c811a  12   13   .rsrc    ascii  AVICAP32.DLL
33 0x00047127  0x004c8127  12   13   .rsrc    ascii  comctl32.dll
34 0x00047134  0x004c8134  9    10   .rsrc    ascii  gdi32.dll
35 0x0004713e  0x004c813e  11   12   .rsrc    ascii  gdiplus.dll
```

Step 5:

Disassembling and Identifying Code Flow

- Open the disassembler view using: **pdr @entry**

```
[root@kali /home/punit/Downloads/darkcomet]
[File Actions Edit View Help]
[0x004c69c0]> pdr @0x004c69c0
; CODE XREF from entry0 @ 0x4c6b73(x)
0x0049f92c  c49efe4785f6  les ebx, [esi - 0x97ab802]
0x0049f932  cc              tnt

;-- elip: crackme
443: entry0 ();
0x004c69c0  60              pushal
0x004c69c1  b00104800        mov esi, section.UPX1           ; 0x481000
0x004c69c6  8d8e00008ff     lea edi, [esi - 0x80000]          ; -1
0x004c69cc  c787b0070a.    mov dword [edi + 0xa07b0], 0x7919e424 ; [0xa07b0:4]=-1
0x004c69d6  57              push edi
0x004c69d7  83cdf1          or ebp, 0xffffffff
; CODE XREF from int.004c3695 @ +0x3300(x)
0x004c69da  eb0e            jmp 0x4c69ea
// true: 0x004c69ea
; CODE XREF from entry0 @ 0x4c69f1(x)
0x004c69e0  8d96              mov al, byte [esi]
0x004c69e2  46              inc esi
0x004c69e3  8807            mov byte [edi], al
0x004c69e5  47              inc edi
// true: 0x004c69e6
; CODE XREF from entry0 @ 0x4c6a9fx(x), 0x4c6ab5(x)
0x004c69e6  01db            add ebx, ebx
0x004c69e8  7507            jne 0x4c69f1
// true: 0x004c69f1 false: 0x004c69ea
; CODE XREF from entry0 @ 0x4c69d0(x)
0x004c69ea  8b1e              mov ebx, dword [esi]
0x004c69ec  83eefc          sub esi, 0xffffffff
0x004c69ef  11db            adc ebx, ebx
// true: 0x004c69f1
; CODE XREF from entry0 @ 0x4c69e8(x)
0x004c69f1  72ed            jb 0x4c69e0
// true: 0x004c69e0 false: 0x004c69f3
0x004c69f3  0801000000        mov eax, 1
// true: 0x004c69f8
; CODE XREF from entry0 @ 0x4c6a22(x)
0x004c69f8  01db            add ebx, ebx
0x004c69fa  7507            jne 0x4c6a03
// true: 0x004c6a03 false: 0x004c69f5
```

Conclusion:

By following these steps, you can successfully reverse engineer a malware sample using Radare2. Reverse engineering helps security professionals understand malware behavior, detect threats early, and develop strategies to defend against cyberattacks. Always remember to work in a secure and isolated environment to avoid any risk of accidental execution of the malware.

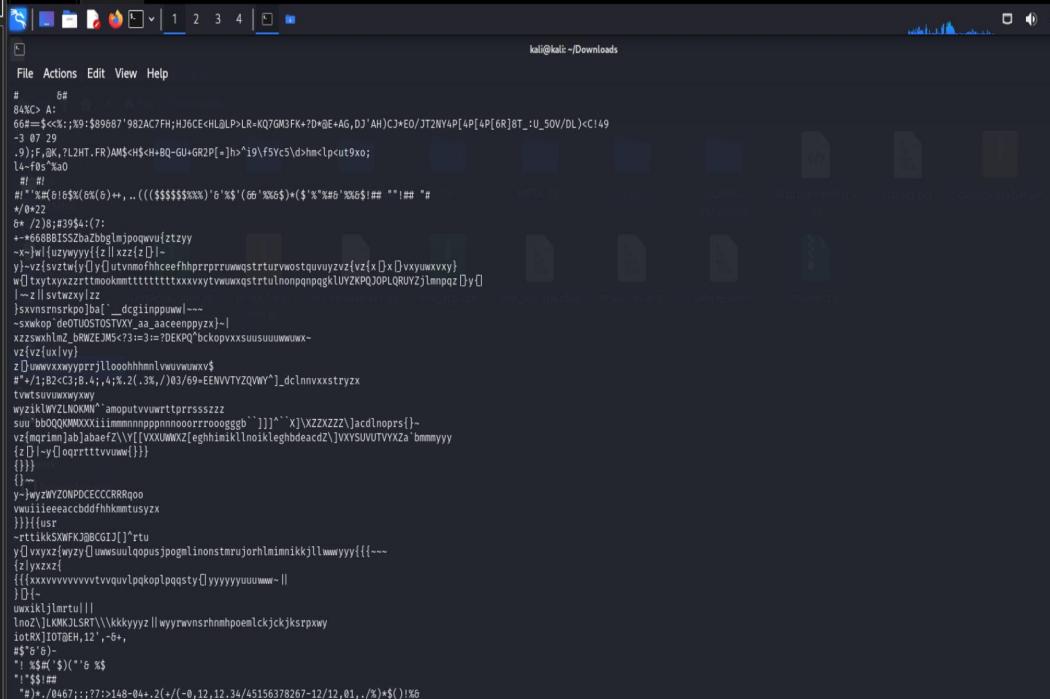
PRACTICAL 6

AIM: Analyzing malware that uses anti-analysis techniques to evade detection.

Malware that uses anti-analysis techniques is specifically designed to detect and avoid security tools, sandboxes, debuggers, and virtualized environments. The primary goal of these techniques is to prevent detection, delay analysis, and hide malicious behavior from researchers and automated tools. Understanding and analyzing such malware requires a careful combination of static, dynamic, and behavioral analysis in a controlled environment.

1. Static Analysis:

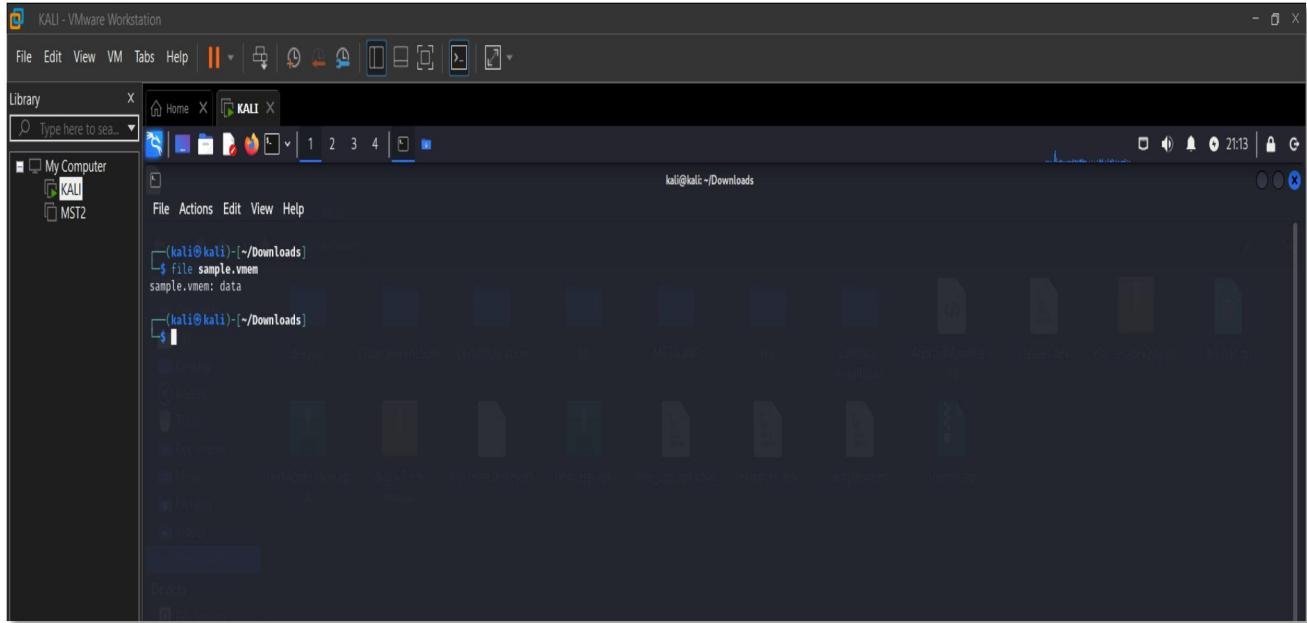
Extract String Using: **string sample..vmem | less**



The screenshot shows a terminal window titled "kali@kali:~\$ Downloads" running on Kali Linux. The terminal displays a large, multi-line exploit payload. The payload is a series of assembly-like instructions, likely generated by a debugger or exploit development tool. It includes various memory addresses, registers, and control flow operations. The code is heavily obfuscated with comments and labels such as "A:", "B:", "C:", and "D:". The payload is designed to exploit a vulnerability in a program, possibly related to the "MS17-010" exploit mentioned in the title. The terminal interface includes a file manager window in the background showing various files and folders.

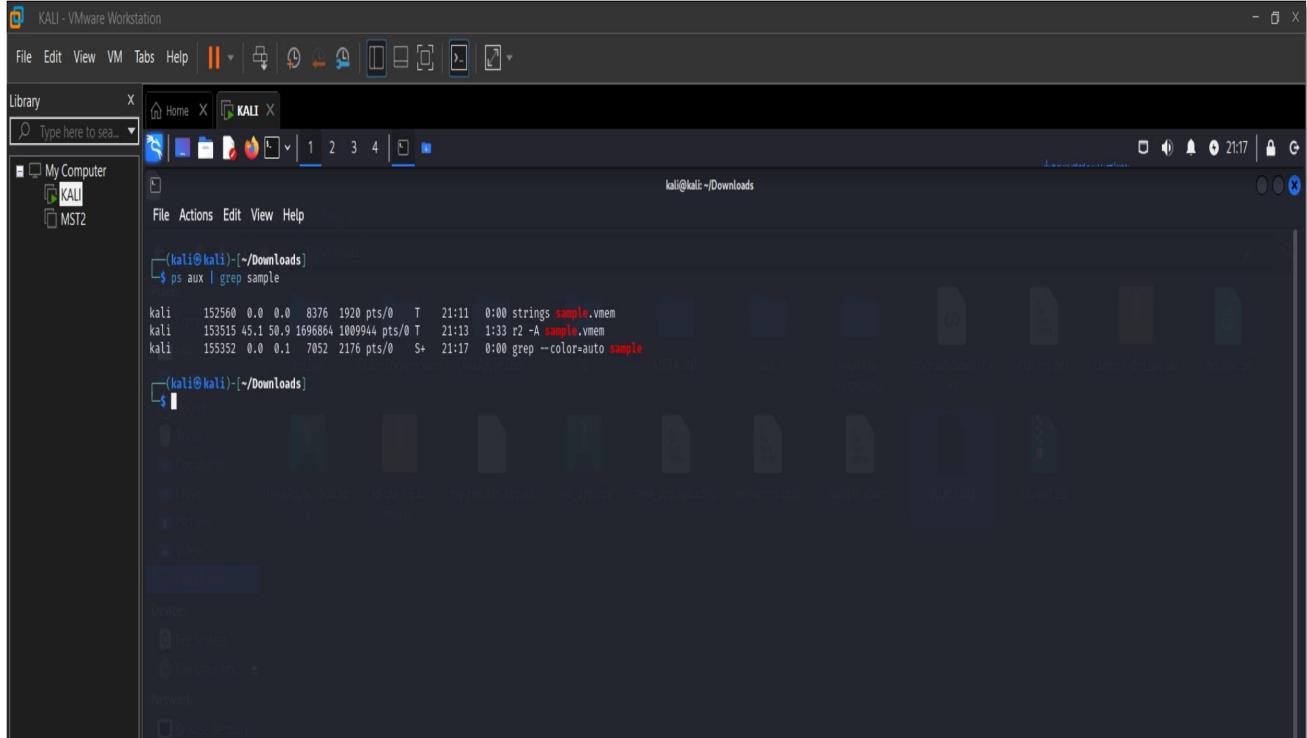
2. Check File Type:

File sample.vmem



3. Memory Analysis:

ps aux | grep sample



4. Disassemble:

r2 -A sample1.exe

Then inside Radare2 Use:

aaa -> Analyze All

Afl -> List Functions

```
(kali㉿kali)-[~/Desktop/darkcomet]
$ r2 -A sample1.exe

WARN: Relocs has not been applied. Please use '-e bin.relocs.apply=true' or '-e bin.cache=true' next time
INFO: Analyze all flags starting with sym. and entry0 (aa)
INFO: Analyze imports (af0@0)
INFO: Analyze exports (af0@0 entry0)
WARN: set your favourite calling convention in 'e anal.cc=?'
INFO: Analyze symbols (af0@0)
INFO: Recovering variables
INFO: Analyze all functions arguments/locals (afva@0@F)
INFO: Analyze function calls (aac)
WARN: Unaligned function 'sub_kernel32.dll_GetWindowThreadProcessId' at 0x00408190 (vs 0x00408188)Try disabling 'e anal.nopskip=false'
WARN: Unaligned function 'sub_user32.dll_GetWindow' at 0x00408148 (vs 0x004080e8)Try disabling 'e anal.nopskip=false'
WARN: Unaligned function 'sub_user32.dll_SendMessageA' at 0x00408318 (vs 0x00408310)Try disabling 'e anal.nopskip=false'
WARN: Unaligned function 'sub_kernel32.dll_FindResourceA' at 0x00407958 (vs 0x00407950)Try disabling 'e anal.nopskip=false'
WARN: Unaligned function 'sub_kernel32.dll_CreateFileA' at 0x00407890 (vs 0x00407888)Try disabling 'e anal.nopskip=false'
WARN: Unaligned function 'sub_shlwapi.dll_ShellExecuteA' at 0x00445970 (vs 0x00445968)Try disabling 'e anal.nopskip=false'
INFO: Analyzing len bytes of instructions for references (aar)
INFO: Finding and parsing C++ vtables (avr)
INFO: Recovering local variables (afva)
INFO: Type matching analysis for all functions (aft)
INFO: Propagate noreturn information (aanr)
INFO: Use -AA or -aaa to perform additional experimental analysis
[0x00009f92c]: afl
Do you want to print 3186 lines? (y/N)
[0x00009f92c]: 
```

To direct input to this VM, move the mouse pointer inside or press Ctrl+G.

Conclusion:

Malware that uses anti-analysis techniques presents significant challenges to security analysts by actively resisting inspection and evading detection tools. Successful analysis requires a deep understanding of both the malware's behavior and the methods used to obscure it. By combining static and dynamic techniques, using modified environments, and carefully stepping through execution, analysts can bypass anti-analysis defenses, uncover the true behavior of the malware, and develop effective detection and mitigation strategies. This kind of analysis is essential in defending against sophisticated, evasive threats in the modern cybersecurity landscape.

PRACTICAL 7

AIM: Analyzing malware that uses cryptographic techniques to hide its functionality.

Analyzing malware that uses cryptographic techniques to hide its functionality involves identifying how the malware encrypts or obfuscates its code, payloads, or communication to evade detection and analysis. These malware types often rely on encryption, encoding, or hashing to prevent researchers from easily reading or understanding the malicious code and activities.

Understanding Cryptographic Techniques in Malware:

Malware using cryptographic techniques typically employs one or more of the following methods to evade detection:

1. **Encryption of Payloads:** The malicious payload is encrypted or obfuscated to hide its content until it is executed or decrypted in memory.
2. **Encrypted Communication:** Malware may use encryption to secure command-and-control (C&C) communications, making it difficult for network monitoring tools to identify the nature of traffic.
3. **Code Obfuscation:** The malware's code may be encrypted or obfuscated to make reverse engineering harder.
4. **Hashing:** Malware may hash file names, strings, or specific data within the malware to hide its true intent.
5. **Key Management:** The malware might generate or use specific keys (either hardcoded or dynamically generated) to decrypt its payload or to interact with external resources.

Steps to Analyze Cryptographically Protected Malware:

1. Initial Static Analysis:

Start with static analysis to understand how the malware works without running it.

Identify and Extract the Encrypted Payload:

Use strings or binwalk to search for patterns like encryption keys or common cryptographic library calls:

Hardcoded keys: Strings that look like encryption keys.

Encryption functions: Common functions like AES_encrypt, RC4, or XOR.

File Integrity Checking:

If the malware is packed or obfuscated:

- Use PEiD or Detect It Easy (DIE) to check for packing:

If the file is packed or obfuscated, look for the unpacking process in the disassembly.

Disassemble the Code:

- Use Ghidra, IDA Pro, or Radare2 to disassemble the binary and look for cryptographic functions.
- Look for calls to cryptographic libraries like OpenSSL, Crypto++, or custom encryption routines.
- Analyze the surrounding code to understand when and how decryption occurs.

2. Dynamic Analysis: Unpacking and Decryption:

If the malware uses runtime decryption or obfuscation, you'll need to observe its behavior during execution.

Monitor System Calls and Memory:

Use x64dbg, OllyDbg, or GDB to debug and inspect the malware's runtime behavior. Set breakpoints at encryption-related functions.

1. Set breakpoints for cryptographic calls (e.g., AES_decrypt or RC4).
2. Step through the code to catch when the encrypted payload is decrypted into memory.
3. Dump memory after the payload is decrypted to extract it.

3. Post-Decryption Analysis:

Once decrypted, you can analyze the payload for malicious behavior. Look for:

- **File manipulation** (e.g., creating or modifying files)
- **Registry changes** (in Windows environments)
- **Malicious network connections** (e.g., backdoors, C&C)
- **Persistence mechanisms** (e.g., startup keys, scheduled tasks)

Conclusion:

Malware that uses cryptographic techniques to hide its functionality is more difficult to analyze due to the added complexity of encryption and obfuscation. However, by combining static analysis (e.g., strings, disassemblers) and dynamic analysis (e.g., debuggers, memory dumping), you can reverse-engineer the cryptographic methods, extract the encrypted payloads, and decrypt them for further inspection. Tools like Wireshark, tcpdump, and Volatility help in uncovering network and memory-based cryptographic activity. Understanding and breaking these encryption layers is crucial for revealing the true intent of the malware, whether it's stealing data, creating backdoors, or evading detection.

PRACTICAL 8

AIM: Analyzing malware that targets specific industries or geographic regions.

Malware that targets specific industries or geographic regions—often referred to as targeted or tailored malware—is designed with precision to infiltrate particular sectors such as finance, healthcare, defense, or energy, or to operate within specific countries or regions. This type of malware is usually associated with Advanced Persistent Threats (APTs), cyber-espionage campaigns, or state-sponsored attacks, and its analysis requires a nuanced understanding of both technical indicators and contextual intelligence.

Characteristics of Targeted Malware:

Customized Payloads

Tailored to exploit vulnerabilities common in specific industry software or infrastructure (e.g., SCADA systems in energy, EMR software in healthcare).

Geolocation Awareness

Malware may activate only if the host system is located in a particular region (e.g., based on IP address, system locale, timezone, or language settings).

Spear Phishing

Delivered through carefully crafted emails to specific individuals in an organization, using contextually relevant lures.

Selective Execution

Designed to avoid detection and analysis by executing only on intended targets or environments, often using checks for hostname, domain name, or user profile.

Use of Native Language

Malware may use strings, filenames, or interfaces in the native language of the target region, further disguising itself as legitimate software.

Steps for Analyzing Targeted Malware:

1. Gather Intelligence:

Understand the target industry or region: what technologies are used, what are common threats, and what vulnerabilities exist.

Use OSINT to collect related Indicators of Compromise (IOCs), attack patterns, and previous campaigns.

2. Static Analysis:

Look for hardcoded values like:

- IP ranges or domain names related to a country

- Language or timezone checks
- Registry keys or file paths specific to targeted industry software

3. Dynamic Behavior Analysis:

Execute the sample in different environments and compare behaviors:

- In a VM configured for the target region (language, keyboard layout, IP geo-location).
- In an industry-specific emulated environment.

Observe which features are enabled or disabled depending on system configuration.

4. Geolocation and Locale Checks:

Look for APIs such as:

- GetSystemDefaultLangID()
- GetTimeZoneInformation()
- GetGeoID()
- Language/environment-specific commands in scripts or batch files

5. Network Analysis:

- Targeted malware often communicates with C2 servers hosted in or near the target region.
- Analyze DNS queries, HTTP requests, and SSL certificates to trace geographic clues.

Case Examples:

Stuxnet: Targeted Iranian nuclear facilities by exploiting industrial control systems.

Industroyer/CrashOverride: Targeted Ukrainian power grid systems.

Emotet: Evolved to send lures tailored to specific countries, with local language templates.

Conclusion:

Analyzing malware that targets specific industries or regions requires more than just technical inspection it demands contextual intelligence and environment simulation. These threats are often stealthy, well-crafted, and activated only under specific conditions. By combining static and dynamic analysis with threat intelligence, analysts can uncover how such malware operates, whom it targets, and how to defend against it. Understanding the geopolitical, linguistic, and technological nuances behind the attack is key to unraveling these sophisticated threats.

PRACTICAL 9

AIM: Analyzing malware that uses rootkits to hide its presence on a system.

Description:

A rootkit is a type of stealthy malware that Operates with root/admin-level privileges Hides files, processes, registry keys, and network connections Often hooks or modifies kernel/system calls Can be user-mode or kernel-mode

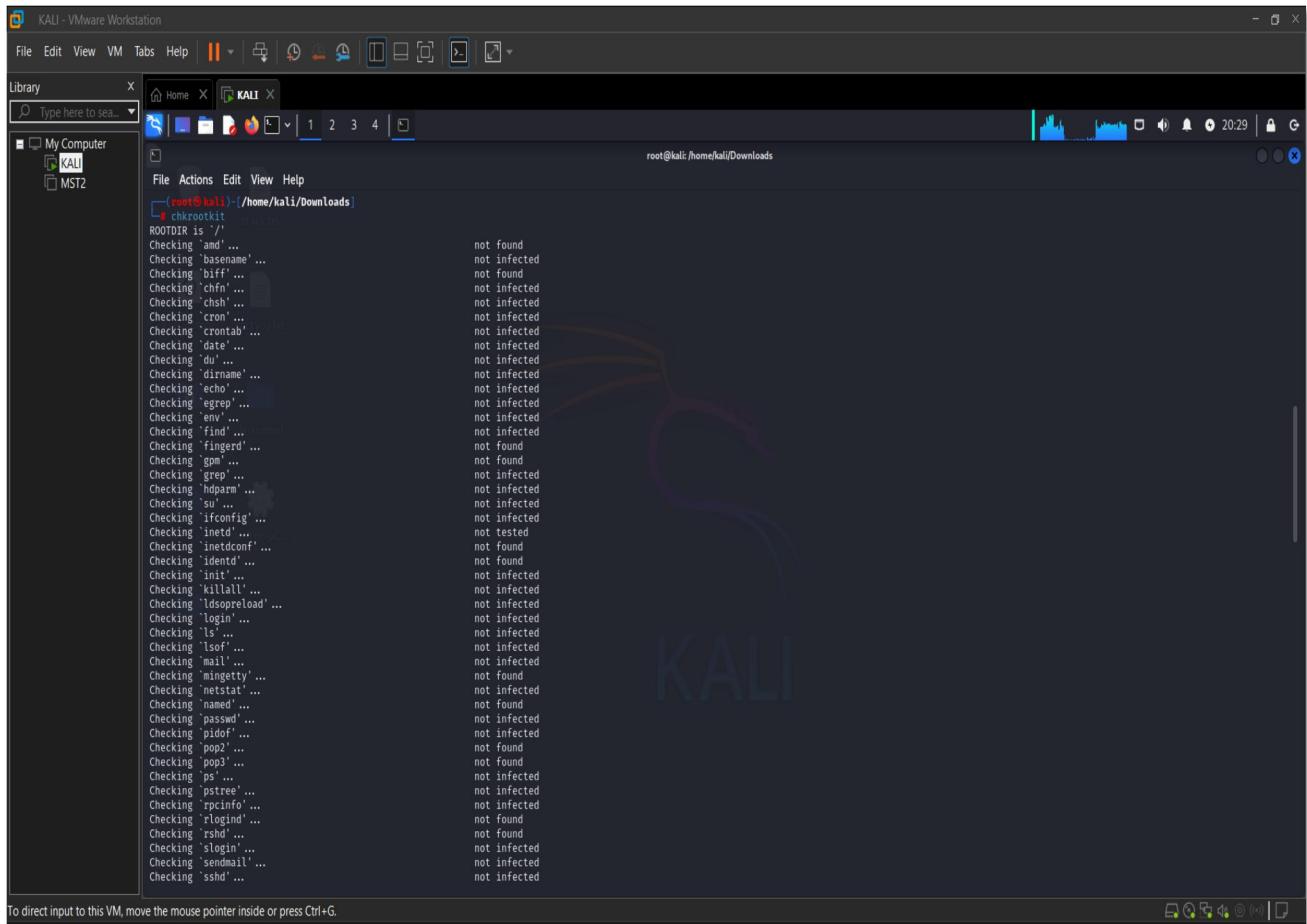
Rootkits are dangerous because they persist across reboots, evade detection, and enable backdoors, data exfiltration, and command execution.

Step 1: Use Anti-Rootkit Scanners.

chkrootkit:

Checks for known rootkits

Detects anomalies in system binaries (e.g., ifconfig, ls, ps)



```
(root@kali:~/home/kali/Downloads]# chkrootkit
ROOTDIR is '/'
Checking 'and' ...
Checking 'basename' ...
Checking 'biff' ...
Checking 'chfn' ...
Checking 'chsh' ...
Checking 'Cron' ...
Checking 'Crontab' ...
Checking 'date' ...
Checking 'du' ...
Checking 'dirname' ...
Checking 'echo' ...
Checking 'egrep' ...
Checking 'env' ...
Checking 'find' ...
Checking 'fingerd' ...
Checking 'gpm' ...
Checking 'grep' ...
Checking 'hdparm' ...
Checking 'su' ...
Checking 'ifconfig' ...
Checking 'inet' ...
Checking 'inetdconf' ...
Checking 'identd' ...
Checking 'init' ...
Checking 'killall' ...
Checking 'ldsopreload' ...
Checking 'login' ...
Checking 'ls' ...
Checking 'lsof' ...
Checking 'mail' ...
Checking 'mingetty' ...
Checking 'netstat' ...
Checking 'named' ...
Checking 'passwd' ...
Checking 'pidof' ...
Checking 'pop2' ...
Checking 'pop3' ...
Checking 'ps' ...
Checking 'pstree' ...
Checking 'rpcinfo' ...
Checking 'rlogind' ...
Checking 'rshd' ...
Checking 'slogin' ...
Checking 'sendmail' ...
Checking 'sshd' ...

To direct input to this VM, move the mouse pointer inside or press Ctrl+G.
```

Step 2: Use rkhunter (Rootkit Hunter):

rkhunter --checkall

```
(root@kali)-[~/home/kali/Downloads]
# sudo rkhunter --checkall

[ Rootkit Hunter version 1.4.6 ]

Checking system commands ...
Performing 'strings' command checks
  Checking 'strings' command [ OK ]
Performing 'shared libraries' checks
  Checking for preloading variables [ None found ]
  Checking for preloaded libraries [ None found ]
  Checking LD_LIBRARY_PATH variable [ Not found ]

Performing file properties checks
  Checking for prerequisites?
    /usr/sbin/adduser [ OK ]
    /usr/sbin/lastlog [ OK ]
    /usr/sbin/cron [ OK ]
    /usr/sbin/demod [ OK ]
    /usr/sbin/fscck [ OK ]
    /usr/sbin/groupadd [ OK ]
    /usr/sbin/groupdel [ OK ]
    /usr/sbin/groupmod [ OK ]
    /usr/sbin/grpck [ OK ]
    /usr/sbin/ifconfig [ OK ]
    /usr/sbin/lsmod [ OK ]
    /usr/sbin/mount [ OK ]
    /usr/sbin/nice [ OK ]
    /usr/sbin/init [ OK ]
    /usr/sbin/insmod [ OK ]
    /usr/sbin/ip [ OK ]
    /usr/sbin/modinfo [ OK ]
    /usr/sbin/modprobe [ OK ]
    /usr/sbin/nologin [ OK ]
    /usr/sbin/pwck [ OK ]
    /usr/sbin/reload [ OK ]
    /usr/sbin/route [ OK ]
    /usr/sbin/runlevel [ OK ]
    /usr/sbin/smod [ OK ]
    /usr/sbin/umount [ OK ]
    /usr/sbin/login [ OK ]
    /usr/sbin/syctl [ OK ]
    /usr/sbin/useradd [ OK ]
    /usr/sbin/userdel [ OK ]
    /usr/sbin/usermod [ OK ]
    /usr/sbin/vipw [ OK ]

To direct input to this VM, move the mouse pointer inside or press Ctrl+G.
```

Step 3: Analyze Kernel Modules

Rootkits often insert malicious kernel modules.

List loaded modules: lsmod

Module	Size	Used by
mptcp_diag	12288	0
xsk_diag	12288	0
vsock_diag	12288	0
tcp_diag	12288	0
udp_diag	12288	0
raw_diag	12288	0
inet_diag	26872	4 tcp_diag,mptcp_diag,raw_diag,udp_diag
unix_diag	12288	0
af_packet_diag	12288	0
netlink_diag	12288	0
tls	15152	0
dm_mod	221184	0
cpid	12288	0
snd_seq_dummy	12288	0
snd_hrtimer	12288	1
snd_seq_midi	29480	0
snd_seq_midi_event	10488	9 snd_seq_midi
snd_seq_midi	114488	9 snd_seq_midi,snd_seq_midi_event,snd_seq_dummy
rfkill	49960	2
qrtr	57344	4
sunrpc	880640	1
bifrost_msc	28672	0
intel_rapl_msr	20480	0
intel_rapl_common	36864	1 intel_rapl_msr
intel_urecore_frequency_common	16384	0
intel_pmc_core	114688	0
intel_vsc	20480	1 intel_pmc_core
pmt_telemetry	10240	1 intel_pmc_core
pmt_class	19228	1 pmt_telemetry
rapl	20480	0
snd_ens1371	36864	1
snd_ac97_codec	196688	1 snd_ens1371
vmw_balloon	28672	0
ac97_bus	12288	1 snd_ac97_codec
gasowl	20872	1 snd_ens1371
snd_rawmidi	53248	2 snd_seq_midi,snd_ens1371
snd_seq_device	16384	3 snd_seq,snd_seq_midi,snd_rawmidi
pcspkr	12288	0
snd_pcm	192512	2 snd_ac97_codec,snd_ens1371
snd_timer	5120	2 snd_seq,snd_hrtimer,snd_pcm
snd	155648	11 snd_seq,snd_seq_device,snd_timer,snd_ac97_codec,snd_pcm,snd_rawmidi,snd_ens1371
soundcore	16384	1 snd
ac	16384	0

To direct input to this VM, move the mouse pointer inside or press Ctrl+G.

Conclusion:

Analyzing malware that uses rootkits requires advanced techniques, because rootkits are designed to hide their presence by manipulating system internals. On Kali Linux, tools like chkrootkit, rkhunter, and file integrity checkers like AIDE help you uncover these hidden threats. Always compare suspected systems to known-good baselines, and perform memory and disk analysis to detect signs of manipulation. Rootkit detection is an essential part of deep forensic investigations and system hardening.

PRACTICAL 10

AIM: Conducting threat intelligence analysis to understand the motivations and goals of malware authors.

Threat intelligence analysis plays a vital role in understanding not only how malware operates but why it was created, who is behind it, and what objectives they aim to achieve. By analyzing malware campaigns from a strategic perspective, security professionals can go beyond technical indicators and uncover the broader intent and context of cyber threats. This process helps organizations improve their defenses, anticipate future attacks, and tailor responses more effectively.

Key Components of Threat Intelligence Analysis:

1. Attribution (Who?):

Identify the possible threat actor(s) behind the malware. This involves studying:

- Coding patterns
- Language or time zone hints
- Infrastructure reuse (e.g., C2 domains or IPs)
- Tactics, techniques, and procedures (TTPs)
- Historical campaigns associated with similar tools

2. Motivations (Why?):

Understanding the purpose of the malware helps define its threat level. Common motivations include:

- **Cybercrime** (financial gain, data theft, ransomware)
- **Cyber espionage** (stealing sensitive or classified information)
- **Hacktivism** (politically or ideologically motivated attacks)
- **Sabotage** (disrupting operations, especially in critical infrastructure)
- **Cyber warfare** (state-sponsored attacks with geopolitical goals)

3. Targets (Who or What?):

Analyzing which industries, regions, or organizations are targeted can reveal the strategic goals of the malware authors:

- Targeted industries (e.g., energy, defense, finance)
- Specific countries or regions
- High-profile individuals or executives (via spear phishing)

4. Tactics, Techniques, and Procedures (How?):

Use frameworks like **MITRE ATT&CK** to map observed behaviors to known techniques, helping to identify common patterns used by known threat actors.

5. Infrastructure Analysis:

Studying the attacker's infrastructure (e.g., domains, IPs, certificates, hosting providers) can help link campaigns, track actor movements, and detect future threats.

6. Malware Capabilities (What?):

Examine the malware's features to infer intent:

- **Keylogging or data exfiltration** → espionage or surveillance
- **Ransomware encryption** → financial gain
- **Destructive payloads** → sabotage or political messaging

Sources for Threat Intelligence:

- **Technical sources:** Malware samples, logs, sandbox reports.
- **Open-source intelligence (OSINT):** Blogs, forums, social media, paste sites.
- **Dark web monitoring:** Forums, marketplaces, communication channels.
- **Threat intelligence feeds:** Commercial or community-based feeds (e.g., VirusTotal, Abuse.ch, AlienVault OTX).
- **Reports from security vendors:** Threat group profiles and campaign summaries.

Tools and Techniques:

- **Maltego** – Link analysis and entity mapping
- **Threat Intelligence Platforms (TIPs)** – e.g., MISP (Malware Information Sharing Platform)
- **YARA** – Signature-based malware hunting
- **MITRE ATT&CK Navigator** – Visualizing attacker TTPs
- **ELK Stack / Splunk** – Analyzing threat data from logs and alerts

Conclusion:

Threat intelligence analysis goes beyond the technical dissection of malware to uncover the who, why, and what behind a cyberattack. By correlating indicators with known behaviors, actors, and motivations, analysts can build a clearer picture of the threat landscape. This holistic view enables organizations to prepare not just for isolated attacks but for ongoing campaigns, targeted threats, and evolving adversary tactics. Understanding the intent and strategy behind malware helps transform reactive security into proactive defense.