Malware Analysis and Detection

Assignment-1

Submitted by:

Muhammad Osama Khalid

20I-1955 (MSCNS)

Section: MSCNS

Table of Contents

Malware-1 GenHeur.PonyStealer.4 1

Task-1 What is complete information malware have 1

Task-1.1 Magic Byte 1

Tool-1 Notepad++ 1

Tool-2 HxD 2

Tool-3 CFF Explorer 2

Task-1.2 File Signature 3

Tool-1 PE Studio 3

Tool-2 CFF Explorer 3

Task-1.3 Machine Information 4

Tool-1 PE Studio 4

Tool-2 CFF Explorer 4

Task-1.4 Exe Type 5

Tool-1 PE Studio 5

Tool-2 CFF Explorer 5

Task-1.5 File Type 6

Tool-1 Python Magic Library 6

Tool-2 TrID Net 7

Task-2 What is the fingerprinting information of the malware file? 8

Tool-1 HashCalc 8

Tool-2 Python Hashlib Library 8

Task-3 What type of malicious strings are in the Malware file? 10

Tool-1 Strings 10

Tool-2 YARA Rule 11

Task-4 Number, type, and offset of different sections of Malware? 12

Tool-1 PE Studio 12

Task-5 Which DLLs and their number of functions are used by Malware? 13

Tool-1 PE Studio 13

Task-6 What is packing information if Malware is packed? 14

Tool-1 Exeinfo 14

Task-7 VirusTotal analysis of the malware file 14

Task-8 Analysis of the malware file through Hybrid Analysis 15
Malware-2 Dropped:Trojan.Dropper.Agent.VOE 17
Task-1 What is complete information malware have 17
Task-1.1 Magic Byte 17
Tool-1 Notepad++ 17
Tool-2 HxD 17
Tool-3 CFF Explorer 18
Task-1.2 File Signature 18
Tool-1 PE Studio 18
Tool-2 CFF Explorer 19
Task-1.3 Machine Information 20
Tool-1 PE Studio 20
Tool-2 CFF Explorer 20
Task-1.4 Exe Type 21
Tool-1 PE Studio 21
Tool-2 CFF Explorer 21
Task-1.5 File Type 22
Tool-1 Python Magic Library 22
Tool-2 TrID Net 23
Task-2 What is the fingerprinting information of the malware file? 24
Tool-1 HashCalc 24
Tool-2 Python Hashlib Library 24
Task-3 What type of malicious strings are in the Malware file? 26
Tool-1 Strings 26
Tool-2 YARA Rule 27
Task-4 Number, type, and offset of different sections of Malware? 28
Tool-1 PE Studio 28
Task-5 Which DLLs and their number of functions are used by Malware? 29
Tool-1 PE Studio 29
Task-6 What is packing information if Malware is packed? 30
Tool-1 Exeinfo 30
Task-7 VirusTotal analysis of the malware file 30
Task-8 Analysis of the malware file through Hybrid Analysis 31

Malware-3 Trojan.GenericKD.3652107 33
Task-1 What is complete information malware have 33
Task-1.1 Magic Byte 33
Tool-1 Notepad++ 33
Tool-2 HxD 33
Tool-3 CFF Explorer 34
Task-1.2 File Signature 34
Tool-1 PE Studio 34
Tool-2 CFF Explorer 35
Task-1.3 Machine Information 36
Tool-1 PE Studio 36
Tool-2 CFF Explorer 36
Task-1.4 Exe Type 37
Tool-1 PE Studio 37
Tool-2 CFF Explorer 37
Task-1.5 File Type 38
Tool-1 Python Magic Library 38
Tool-2 TrID Net 39
Task-2 What is the fingerprinting information of the malware file? 40
Tool-1 HashCalc 40
Tool-2 Python Hashlib Library 41
Task-3 What type of malicious strings are in the Malware file? 43
Tool-1 Strings 43
Tool-2 YARA Rule 44
Task-4 Number, type, and offset of different sections of Malware? 45
Tool-1 PE Studio 45
Task-5 Which DLLs and their number of functions are used by Malware? 46
Tool-1 PE Studio 46
Task-6 What is packing information if Malware is packed? 47
Tool-1 Exeinfo 47
Task-7 VirusTotal analysis of the malware file 47
Task-8 Analysis of the malware file through Hybrid Analysis 48
Malware-4 Password-Stealer (003bbfec1) 50

Task-1 What is complete information malware have 50
Task-1.1 Magic Byte 50
Tool-1 Notepad++ 50
Tool-2 HxD 50
Tool-3 CFF Explorer 51
Task-1.2 File Signature 51
Tool-1 PE Studio 51
Tool-2 CFF Explorer 52
Task-1.3 Machine Information 52
Tool-1 PE Studio 52
Tool-2 CFF Explorer 53
Task-1.4 Exe Type 53
Tool-1 PE Studio 53
Tool-2 CFF Explorer 54
Task-1.5 File Type 54
Tool-1 Python Magic Library 54
Tool-2 TrID Net 55
Task-2 What is the fingerprinting information of the malware file? 56
Tool-1 HashCalc 56
Tool-2 Python Hashlib Library 56
Task-3 What type of malicious strings are in the Malware file? 58
Tool-1 Strings 58
Tool-2 YARA Rule 59
Task-4 Number, type, and offset of different sections of Malware? 60
Tool-1 PE Studio 60
Task-5 Which DLLs and their number of functions are used by Malware? 61
Tool-1 PE Studio 61
Task-6 What is packing information if Malware is packed? 62
Tool-1 Exeinfo 62
Task-7 VirusTotal analysis of the malware file 62
Task-8 Analysis of the malware file through Hybrid Analysis 63
Malware-5 GenVariant.Ransom.Cerber.171 65
Task-1 What is complete information malware have 65

Task-1.1 Magic Byte 65
Tool-1 Notepad++ 65
Tool-2 HxD 65
Tool-3 CFF Explorer 66
Task-1.2 File Signature 66
Tool-1 PE Studio 66
Tool-2 CFF Explorer 67
Task-1.3 Machine Information 67
Tool-1 PE Studio 67
Tool-2 CFF Explorer 68
Task-1.4 Exe Type 68
Tool-1 PE Studio 68
Tool-2 CFF Explorer 69
Task-1.5 File Type 69
Tool-1 Python Magic Library 69
Tool-2 TrID Net 70
Task-2 What is the fingerprinting information of the malware file? 71
Tool-1 HashCalc 71
Tool-2 Python Hashlib Library 71
Task-3 What type of malicious strings are in the Malware file? 73
Tool-1 Strings 73
Tool-2 YARA Rule 74
Task-4 Number, type, and offset of different sections of Malware? 75
Tool-1 PE Studio 75
Task-5 Which DLLs and their number of functions are used by Malware? 76
Tool-1 PE Studio 76
Task-6 What is packing information if Malware is packed? 77
Tool-1 Exeinfo 77
Task-7 VirusTotal analysis of the malware file 77
Task-8 Analysis of the malware file through Hybrid Analysis 78
Malware-6 in32Tnega.bXRKZUB 80
Task-1 What is complete information malware have 80
Task-1.1 Magic Byte 80

Tool-1 Notepad++ 80
Tool-2 HxD 80
Tool-3 CFF Explorer 81
Task-1.2 File Signature 81
Tool-1 PE Studio 81
Tool-2 CFF Explorer 82
Task-1.3 Machine Information 82
Tool-1 PE Studio 82
Tool-2 CFF Explorer 83
Task-1.4 Exe Type 83
Tool-1 PE Studio 83
Tool-2 CFF Explorer 84
Task-1.5 File Type 84
Tool-1 Python Magic Library 84
Tool-2 TrID Net 85
Task-2 What is the fingerprinting information of the malware file? 86
Tool-1 HashCalc 86
Tool-2 Python Hashlib Library 86
Task-3 What type of malicious strings are in the Malware file? 88
Tool-1 Strings 88
Tool-2 YARA Rule 89
Task-4 Number, type, and offset of different sections of Malware? 90
Tool-1 PE Studio 90
Task-5 Which DLLs and their number of functions are used by Malware? 91
Tool-1 PE Studio 91
Task-6 What is packing information if Malware is packed? 92
Tool-1 Exeinfo 92
Task-7 VirusTotal analysis of the malware file 92
Task-8 Analysis of the malware file through Hybrid Analysis 94
Malware-7 W32.SecretKAN.Trojan 95
Task-1 What is complete information malware have 95
Task-1.1 Magic Byte 95
Tool-1 Notepad++ 95

Tool-2 HxD 95
Tool-3 CFF Explorer 96
Task-1.2 File Signature 96
Tool-1 PE Studio 96
Tool-2 CFF Explorer 97
Task-1.3 Machine Information 97
Tool-1 PE Studio 97
Tool-2 CFF Explorer 98
Task-1.4 Exe Type 98
Tool-1 PE Studio 98
Tool-2 CFF Explorer 99
Task-1.5 File Type 99
Tool-1 Python Magic Library 99
Tool-2 TrID Net 100
Task-2 What is the fingerprinting information of the malware file? 101
Tool-1 HashCalc 101
Tool-2 Python Hashlib Library 101
Task-3 What type of malicious strings are in the Malware file? 103
Tool-1 Strings 103
Tool-2 YARA Rule 104
Task-4 Number, type, and offset of different sections of Malware? 105
Tool-1 PE Studio 105
Task-5 Which DLLs and their number of functions are used by Malware? 106
Tool-1 PE Studio 106
Task-6 What is packing information if Malware is packed? 107
Tool-1 Exeinfo 107
Task-7 VirusTotal analysis of the malware file 107
Task-8 Analysis of the malware file through Hybrid Analysis 108
Malware-8 Backdoor.TXLK-8101 110
Task-1 What is complete information malware have 110
Task-1.1 Magic Byte 110
Tool-1 Notepad++ 110
Tool-2 HxD 110

Tool-3 CFF Explorer 111
Task-1.2 File Signature 111
Tool-1 PE Studio 111
Tool-2 CFF Explorer 112
Task-1.3 Machine Information 112
Tool-1 PE Studio 112
Tool-2 CFF Explorer 113
Task-1.4 Exe Type 113
Tool-1 PE Studio 113
Tool-2 CFF Explorer 114
Task-1.5 File Type 114
Tool-1 Python Magic Library 114
Tool-2 TrID Net 115
Task-2 What is the fingerprinting information of the malware file? 116
Tool-1 HashCalc 116
Tool-2 Python Hashlib Library 116
Task-3 What type of malicious strings are in the Malware file? 118
Tool-1 Strings 118
Tool-2 YARA Rule 119
Task-4 Number, type, and offset of different sections of Malware? 120
Tool-1 PE Studio 120
Task-5 Which DLLs and their number of functions are used by Malware? 121
Tool-1 PE Studio 121
Task-6 What is packing information if Malware is packed? 122
Tool-1 Exeinfo 122
Task-7 VirusTotal analysis of the malware file 122
Task-8 Analysis of the malware file through Hybrid Analysis 124

List of Figures

Figure 1: Malware-1 Magic Byte through Notepad++ 1
Figure 2: Malware-1 Magic Byte through HxD 2
Figure 3: Malware-1 Magic Byte through CFF Explorer 2
Figure 4: Malware-1 File Signature through PE Studio 3
Figure 5: Malware-1 File Signature through CFF Explorer 3
Figure 6: Malware-1 Machine Information through PE Studio 4
Figure 7: Malware-1 Machine Information through CFF Explorer 4
Figure 8: Malware-1 Exe type through PE Studio 5
Figure 9: Malware-1 Exe type through CFF Explorer 5
Figure 10: Malware-1 File Type through Python Magic Library 6
Figure 11: Malware-1 File Type through TrID Net 7
Figure 12: Malware-1 Fingerprinting through HashCalc 8
Figure 13: Malware-1 Fingerprinting through Python Hashlib Library 9
Figure 14: Malware-1 Strings Extraction through Strings Tool 10
Figure 15: Malware-1 Strings Extracted through Strings Tool 10
Figure 16: Malware-1 Malware detection through YARA Rule 11
Figure 17: Malware-1 Information of Sections through PE Studio 12
Figure 18: Malware-1 Information of Sections through PE Studio 12
Figure 19: Malware-1 Information of DLLs and functions through PE Studio 13
Figure 20: Malware-1 Packed or unpacked malware information through Exeinfo 14
Figure 21: Malware-1 VirusTotal Analysis 14
Figure 22: Malware-1 VirusTotal Analysis 15
Figure 23: Malware-1 Analysis through Hybrid Analysis 15
Figure 24: Malware-1 Analysis through Hybrid Analysis 16
Figure 25: Malware-2 Magic Byte through Notepad++ 17
Figure 26: Malware-2 Magic Byte through HxD 17
Figure 27: Malware-2 Magic Byte through CFF Explorer 18
Figure 28: Malware-2 File Signature through PE Studio 18
Figure 29: Malware-2 File Signature through CFF Explorer 19
Figure 30: Malware-2 Machine Information through PE Studio 20
Figure 31: Malware-2 Machine Information through CFF Explorer 20
Figure 32: Malware-2 Exe type through PE Studio 21
Figure 33: Malware-2 Exe type through CFF Explorer 21
Figure 34: Malware-2 File Type through Python Magic Library 22
Figure 35: Malware-2 File Type through TrID Net 23
Figure 36: Malware-2 Fingerprinting through HashCalc 24
Figure 37: Malware-2 Fingerprinting through Python Hashlib Library 25
Figure 38: Malware-2 Strings Extraction through Strings Tool 26
Figure 39: Malware-2 Strings Extracted through Strings Tool 26
Figure 40: Malware-2 Malware detection through YARA Rule 27
Figure 41: Malware-2 Information of Sections through PE Studio 28
Figure 42: Malware-2 Information of Sections through PE Studio 28
Figure 43: Malware-2 Information of DLLs and functions through PE Studio 29

Figure 44: Malware-2 Packed or unpacked malware information through Exeinfo 30
Figure 45: Malware-2 VirusTotal Analysis..... 30
Figure 46: Malware-2 VirusTotal Analysis..... 31
Figure 47: Malware-2 Analysis through Hybrid Analysis 31
Figure 48: Malware-2 Analysis through Hybrid Analysis 32
Figure 49: Malware-3 Magic Byte through Notepad++ 33
Figure 50: Malware-3 Magic Byte through HxD 33
Figure 51: Malware-3 Magic Byte through CFF Explorer 34
Figure 52: Malware-3 File Signature through PE Studio 34
Figure 53: Malware-3 File Signature through CFF Explorer 35
Figure 54: Malware-3 Machine Information through PE Studio..... 36
Figure 55: Malware-3 Machine Information through CFF Explorer..... 36
Figure 56: Malware-3 Exe type through PE Studio 37
Figure 57: Malware-3 Exe type through CFF Explorer 37
Figure 58: Malware-3 File Type through Python Magic Library..... 38
Figure 59: Malware-3 File Type through TrID Net 39
Figure 60: Malware-3 Fingerprinting through HashCalc..... 40
Figure 61: Malware-3 Fingerprinting through Python Hashlib Library 42
Figure 62: Malware-3 Strings Extraction through Strings Tool 43
Figure 63: Malware-3 Strings Extracted through Strings Tool 43
Figure 64: Malware-3 Malware detection through YARA Rule..... 44
Figure 65: Malware-3 Information of Sections through PE Studio 45
Figure 66: Malware-3 Information of Sections through PE Studio 45
Figure 67: Malware-3 Information of DLLs and functions through PE Studio 46
Figure 68: Malware-3 Packed or unpacked malware information through Exeinfo 47
Figure 69: Malware-3 VirusTotal Analysis..... 47
Figure 70: Malware-3 VirusTotal Analysis..... 48
Figure 71: Malware-3 Analysis through Hybrid Analysis 48
Figure 72: Malware-3 Analysis through Hybrid Analysis 49
Figure 73: Malware-4 Magic Byte through Notepad++ 50
Figure 74: Malware-4 Magic Byte through HxD 50
Figure 75: Malware-4 Magic Byte through CFF Explorer 51
Figure 76: Malware-4 File Signature through PE Studio 51
Figure 77: Malware-4 File Signature through CFF Explorer 52
Figure 78: Malware-4 Machine Information through PE Studio..... 52
Figure 79: Malware-4 Machine Information through CFF Explorer..... 53
Figure 80: Malware-4 Exe type through PE Studio 53
Figure 81: Malware-4 Exe type through CFF Explorer 54
Figure 82: Malware-4 File Type through Python Magic Library..... 55
Figure 83: Malware-4 File Type through TrID Net 55
Figure 84: Malware-4 Fingerprinting through HashCalc..... 56
Figure 85: Malware-4 Fingerprinting through Python Hashlib Library 57
Figure 86: Malware-4 Strings Extraction through Strings Tool 58
Figure 87: Malware-4 Strings Extracted through Strings Tool 58

Figure 88: Malware-4 Malware detection through YARA Rule 59
Figure 89: Malware-4 Information of Sections through PE Studio 60
Figure 90: Malware-4 Information of Sections through PE Studio 60
Figure 91: Malware-4 Information of DLLs and functions through PE Studio 61
Figure 92: Malware-4 Packed or unpacked malware information through Exeinfo 62
Figure 93: Malware-4 VirusTotal Analysis 62
Figure 94: Malware-4 VirusTotal Analysis 63
Figure 95: Malware-4 Analysis through Hybrid Analysis 63
Figure 96: Malware-4 Analysis through Hybrid Analysis 64
Figure 97: Malware-5 Magic Byte through Notepad++ 65
Figure 98: Malware-5 Magic Byte through HxD 65
Figure 99: Malware-5 Magic Byte through CFF Explorer 66
Figure 100: Malware-5 File Signature through PE Studio 66
Figure 101: Malware-5 File Signature through CFF Explorer 67
Figure 102: Malware-5 Machine Information through PE Studio 67
Figure 103: Malware-5 Machine Information through CFF Explorer 68
Figure 104: Malware-5 Exe type through PE Studio 68
Figure 105: Malware-5 Exe type through CFF Explorer 69
Figure 106: Malware-5 File Type through Python Magic Library 70
Figure 107: Malware-5 File Type through TrID Net 70
Figure 108: Malware-5 Fingerprinting through HashCalc 71
Figure 109: Malware-5 Fingerprinting through Python Hashlib Library 72
Figure 110: Malware-5 Strings Extraction through Strings Tool 73
Figure 111: Malware-5 Strings Extracted through Strings Tool 73
Figure 112: Malware-5 Malware detection through YARA Rule 74
Figure 113: Malware-5 Information of Sections through PE Studio 75
Figure 114: Malware-5 Information of Sections through PE Studio 75
Figure 115: Malware-5 Information of DLLs and functions through PE Studio 76
Figure 116: Malware-5 Packed or unpacked malware information through Exeinfo 77
Figure 117: Malware-5 VirusTotal Analysis 77
Figure 118: Malware-5 VirusTotal Analysis 78
Figure 119: Malware-5 Analysis through Hybrid Analysis 78
Figure 120: Malware-5 Analysis through Hybrid Analysis 79
Figure 121: Malware-6 Magic Byte through Notepad++ 80
Figure 122: Malware-6 Magic Byte through HxD 80
Figure 123: Malware-6 Magic Byte through CFF Explorer 81
Figure 124: Malware-6 File Signature through PE Studio 81
Figure 125: Malware-6 File Signature through CFF Explorer 82
Figure 126: Malware-6 Machine Information through PE Studio 82
Figure 127: Malware-6 Machine Information through CFF Explorer 83
Figure 128: Malware-6 Exe type through PE Studio 83
Figure 129: Malware-6 Exe type through CFF Explorer 84
Figure 130: Malware-6 File Type through Python Magic Library 85
Figure 131: Malware-6 File Type through TrID Net 85

Figure 132: Malware-6 Fingerprinting through HashCalc 86
Figure 133: Malware-6 Fingerprinting through Python Hashlib Library 87
Figure 134: Malware-6 Strings Extraction through Strings Tool 88
Figure 135: Malware-6 Strings Extracted through Strings Tool 88
Figure 136: Malware-6 Malware detection through YARA Rule 89
Figure 137: Malware-6 Information of Sections through PE Studio 90
Figure 138: Malware-6 Information of Sections through PE Studio 90
Figure 139: Malware-6 Information of DLLs and functions through PE Studio 91
Figure 140: Malware-6 Packed or unpacked malware information through Exeinfo 92
Figure 141: Malware-6 VirusTotal Analysis 92
Figure 142: Malware-6 VirusTotal Analysis 93
Figure 143: Malware-6 VirusTotal Analysis 93
Figure 144: Malware-6 Analysis through Hybrid Analysis 94
Figure 145: Malware-6 Analysis through Hybrid Analysis 94
Figure 146: Malware-7 Magic Byte through Notepad++ 95
Figure 147: Malware-7 Magic Byte through HxD 95
Figure 148: Malware-7 Magic Byte through CFF Explorer 96
Figure 149: Malware-7 File Signature through PE Studio 96
Figure 150: Malware-7 File Signature through CFF Explorer 97
Figure 151: Malware-7 Machine Information through PE Studio 97
Figure 152: Malware-7 Machine Information through CFF Explorer 98
Figure 153: Malware-7 Exe type through PE Studio 98
Figure 154: Malware-7 Exe type through CFF Explorer 99
Figure 155: Malware-7 File Type through Python Magic Library 100
Figure 156: Malware-7 File Type through TrID Net 100
Figure 157: Malware-7 Fingerprinting through HashCalc 101
Figure 158: Malware-7 Fingerprinting through Python Hashlib Library 102
Figure 159: Malware-7 Strings Extraction through Strings Tool 103
Figure 160: Malware-7 Strings Extracted through Strings Tool 103
Figure 161: Malware-7 Malware detection through YARA Rule 104
Figure 162: Malware-7 Information of Sections through PE Studio 105
Figure 163: Malware-7 Information of Sections through PE Studio 105
Figure 164: Malware-7 Information of DLLs and functions through PE Studio 106
Figure 165: Malware-7 Packed or unpacked malware information through Exeinfo 107
Figure 166: Malware-7 VirusTotal Analysis 107
Figure 167: Malware-7 VirusTotal Analysis 108
Figure 168: Malware-7 Analysis through Hybrid Analysis 108
Figure 169: Malware-7 Analysis through Hybrid Analysis 109
Figure 170: Malware-8 Magic Byte through Notepad++ 110
Figure 171: Malware-8 Magic Byte through HxD 110
Figure 172: Malware-8 Magic Byte through CFF Explorer 111
Figure 173: Malware-8 File Signature through PE Studio 111
Figure 174: Malware-8 File Signature through CFF Explorer 112
Figure 175: Malware-8 Machine Information through PE Studio 112

Figure 176: Malware-8 Machine Information through CFF Explorer 113
Figure 177: Malware-8 Exe type through PE Studio 113
Figure 178: Malware-8 Exe type through CFF Explorer 114
Figure 179: Malware-8 File Type through Python Magic Library 115
Figure 180: Malware-8 File Type through TrID Net 115
Figure 181: Malware-8 Fingerprinting through HashCalc 116
Figure 182: Malware-8 Fingerprinting through Python Hashlib Library 117
Figure 183: Malware-8 Strings Extraction through Strings Tool 118
Figure 184: Malware-8 Strings Extracted through Strings Tool 118
Figure 185: Malware-8 Malware detection through YARA Rule 119
Figure 186: Malware-8 Information of Sections through PE Studio 120
Figure 187: Malware-8 Information of Sections through PE Studio 120
Figure 188: Malware-8 Information of Sections through PE Studio 121
Figure 189: Malware-8 Information of DLLs and functions through PE Studio 121
Figure 190: Malware-8 Packed or unpacked malware information through Exeinfo 122
Figure 191: Malware-8 VirusTotal Analysis 122
Figure 192: Malware-8 VirusTotal Analysis 123
Figure 193: Malware-8 VirusTotal Analysis 123
Figure 194: Malware-8 Analysis through Hybrid Analysis 124
Figure 195: Malware-8 Analysis through Hybrid Analysis 124

Static analysis is a type of malware analysis in which we examine the malware without running it. In the static analysis, we extract the information of the malware through its binary to analyze its type, signature, machine information, DLLs and functions it will use, etc. In this assignment, I have to perform static analysis on the given malware samples with the help of different tools.

Malware-1 GenHeur.PonyStealer.4

This is a password stealer malware that steals passwords from the victim's machine and has the ability to decrypt them.

Task-1 What is complete information malware have

In this task, I have to find the basic information of the malware that includes identifying magic byte, signatures, machine info, exe type, and file type.

Task-1.1 Magic Byte

A magic byte is are the bytes that are used to recognize the file type e.g., exe, pdf, png, etc. Every file type has its unique magic byte. An example of some of the magic bytes of most commonly used files is as follows.

File Type	Magic Byte	Hex
EXE	MZ	4D 5A
PNG	PNG	50 4E 47
ZIP	PK	50 4B

To find the magic byte of the malware there are a bunch of tools available and for this task, I have used three different tools Notepad++, HxD, and CFF Explorer.

Tool-1 Notepad++

To find the magic byte of the malware file I open the file in Notepad++, then from the plugin tab click on "HEX-Editor" and then click on "View in HEX". On viewing the file in HEX-Editor I found that the magic byte of the malware file is "**MZ**". This means that the malware file is executable. Moreover, I also find that this file can't be able to run in DOS mode.

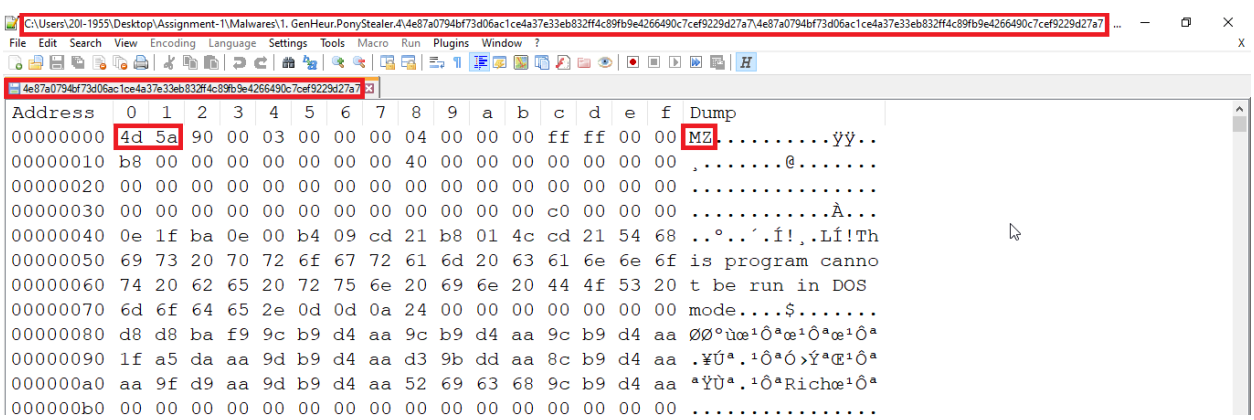

Figure 1: Malware-1 Magic Byte through Notepad++

Tool-2 HxD

To find the magic byte of the malware file I open the file in the HxD and find that the magic byte of the file is "**MZ**" which means that the malware file is executable. Moreover, I also find that this file can't be able to run in DOS mode.

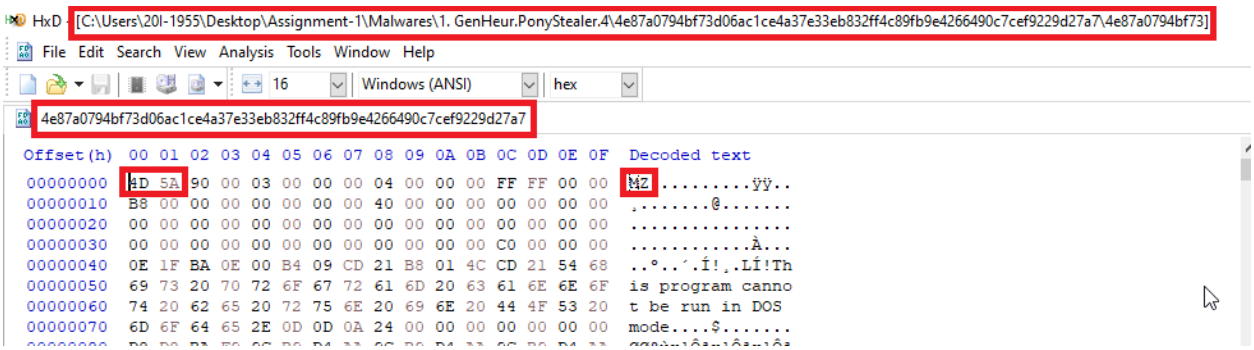

Figure 2: Malware-1 Magic Byte through HxD

Tool-3 CFF Explorer

To find the magic byte of the malware file I open the file in the CFF Explorer, then from the left panel I click on "**DOS Header**". In the "**DOS Header**" I find the magic byte in the "e_magic" column that is "**5A 4D**" which is equivalent to "**4D 5A**" because in CFF Explorer hexadecimal are represented in right to left order. So "**4D = M**" and "**5A = Z**" mean that the malware file is executable.

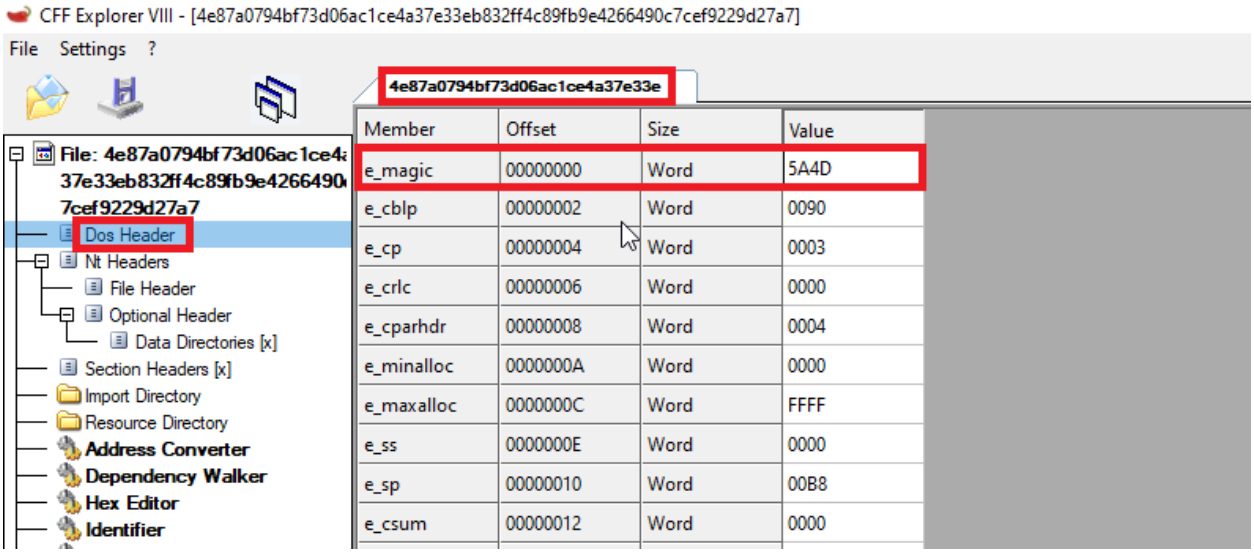

Figure 3: Malware-1 Magic Byte through CFF Explorer

Task-1.2 File Signature

A file signature is used to identify the content of the file. This information is stored in the file's header. To find the file signature of the malware I have used two different tools PE Studio and CFF Explorer.

Tool-1 PE Studio

To find the file signature of the malware file I open the file in the PE Studio, then from the left panel I click on "**file-header**" and then from the right panel I found the column named "**signature**". From there I found that the signature of the file is "**PE**" which means that this file is portable executable.

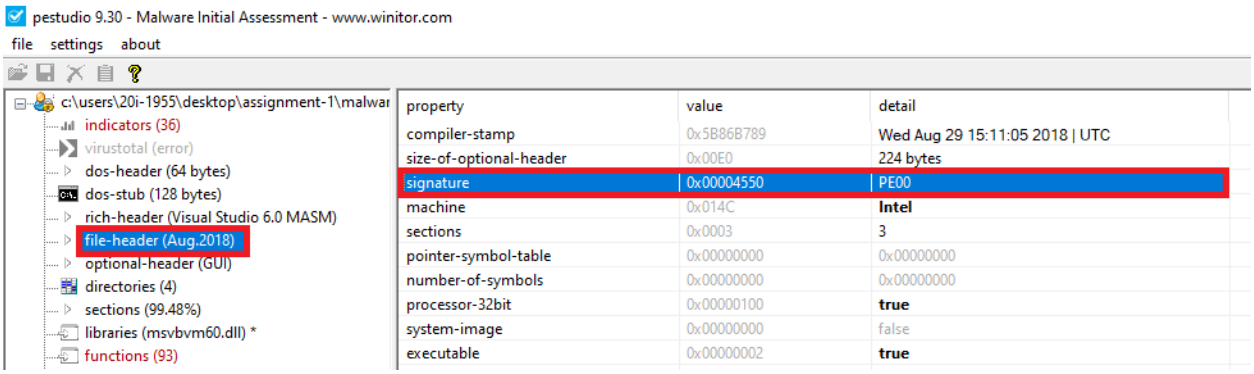

Figure 4: Malware-1 File Signature through PE Studio

Tool-2 CFF Explorer

To find the file signature of the malware file I open the file in the CFF Explorer, then from the left panel I click on "**Nt Headers**" and then from the right panel I found the column named "**signature**". From there I found that the signature of the file is "**45 50**" which is equivalent to "**50 45**" because in CFF Explorer hexadecimal are represented in right to left order. So "**50 = P**" and "**45 = E**" which means that this file is portable executable.

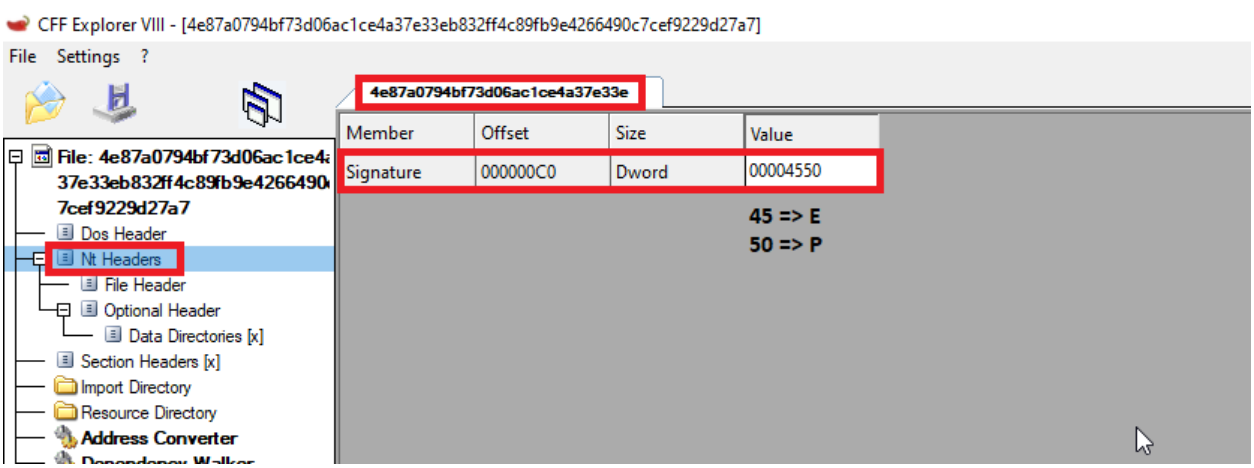

Figure 5: Malware-1 File Signature through CFF Explorer

Task-1.3 Machine Information

Machine information is used to identify that which machine or architecture this malware is designed in other words, we can say that on which type of machines this malware will run. To find the machine information of the malware I have used two different tools PE Studio and CFF Explorer.

Tool-1 PE Studio

To find the file signature of the malware file I open the file in the PE Studio, then from the left panel I click on "**file-header**" and then from the right panel I found the column named "**machine**". From there I found that this malware is designed for Intel-based machines. Other than machine information I also find that when the malware is created from the "**compiler-stamp**" column. This malware is created on Aug 29, 2018, on Wednesday at 03:11 PM. Also find that this malware file has 3 sections and the size of the optional header is 224 bytes.

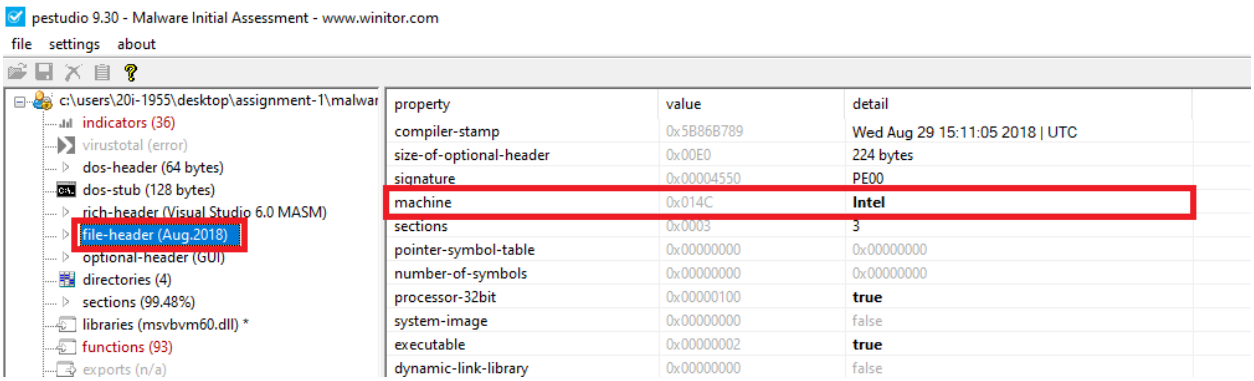

Figure 6: Malware-1 Machine Information through PE Studio

Tool-2 CFF Explorer

To find the file signature of the malware file I open the file in the CFF Explorer, then from the left panel I click on "**file header**" and then from the right panel I found the column named "**Machine**". From there I found that this malware is designed for Intel 386 based machines.

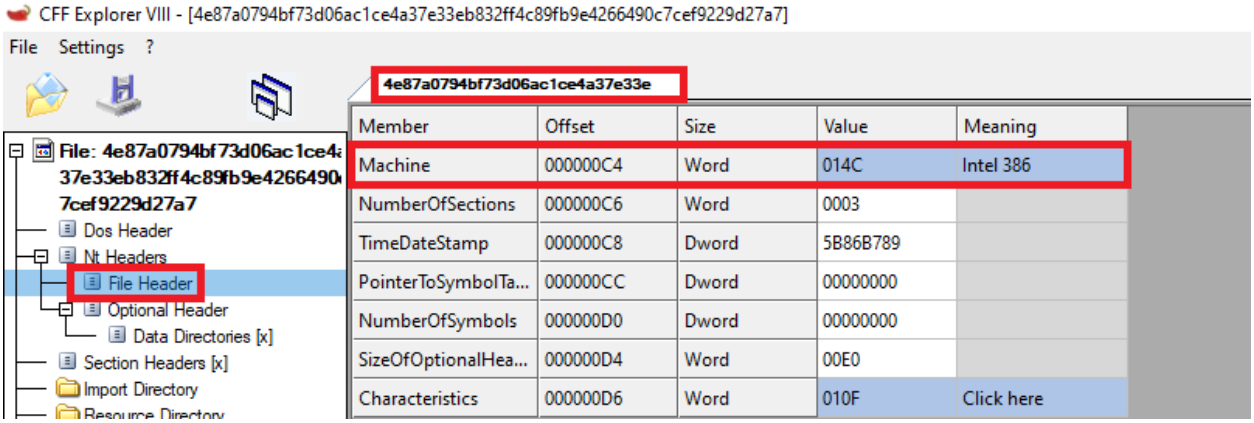

Figure 7: Malware-1 Machine Information through CFF Explorer

Task-1.4 Exe Type

Exe type is used to identify which type of executable it is either 32-bit or 64-bit and to find that I used two different tools PE Studio and CFF Explorer.

Tool-1 PE Studio

To find the exe type of the malware file I open the file in the PE Studio, then from the left panel I click on "**optional-header**" and then from the right panel I found the column named "**magic**". From there I found that the exe type of this malware file is "0x010B" which means that this malware file is a 32-bit portable executable. Other than exe type I also find the entry point of the malware which is ".text".

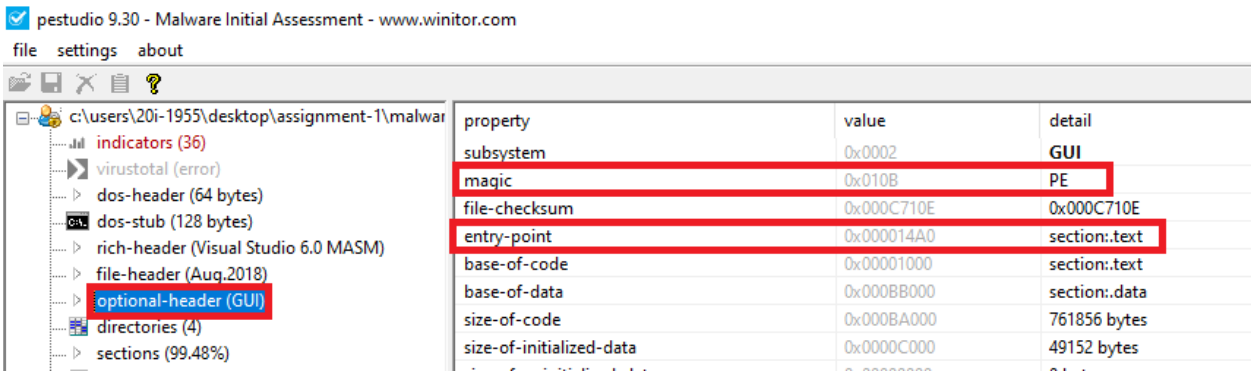

Figure 8: Malware-1 Exe type through PE Studio

Tool-2 CFF Explorer

To find the file signature of the malware file I open the file in the CFF Explorer, then from the left panel I click on "**optional header**" and then from the right panel I found the column named "**Magic**". From there I found that the exe type of this malware file is "010B" which means that this malware file is a 32-bit portable executable. Other than exe type I also find the entry point of the malware which is ".text".

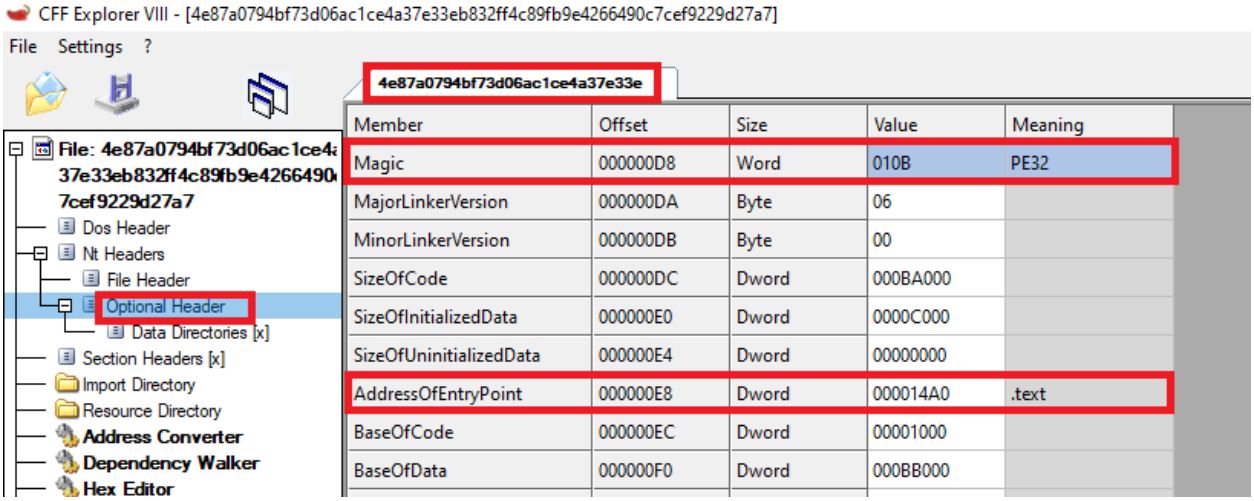

Figure 9: Malware-1 Exe type through CFF Explorer

Task-1.5 File Type

File Type is used to identify which type of file it is. I used two different tools Python Magic Library and TrID Net.

Tool-1 Python Magic Library

To find the file type of the malware using the python magic library I write the following code.

```
import magic

mgicByte = magic.from_file('C:\\Users\\20I-1955\\Desktop\\Assignment-
1\\Malwares\\1.
GenHeur.PonyStealer.4\\4e87a0794bf73d06ac1ce4a37e33eb832ff4c89fb9e4266490c7cef9
229d27a7\\4e87a0794bf73d06ac1ce4a37e33eb832ff4c89fb9e4266490c7cef9229d27a7')

print(mgicByte)
```

After running the code, I found that the file type of malware is “PE32 executable (GUI) Intel 80386, for MS Windows”.

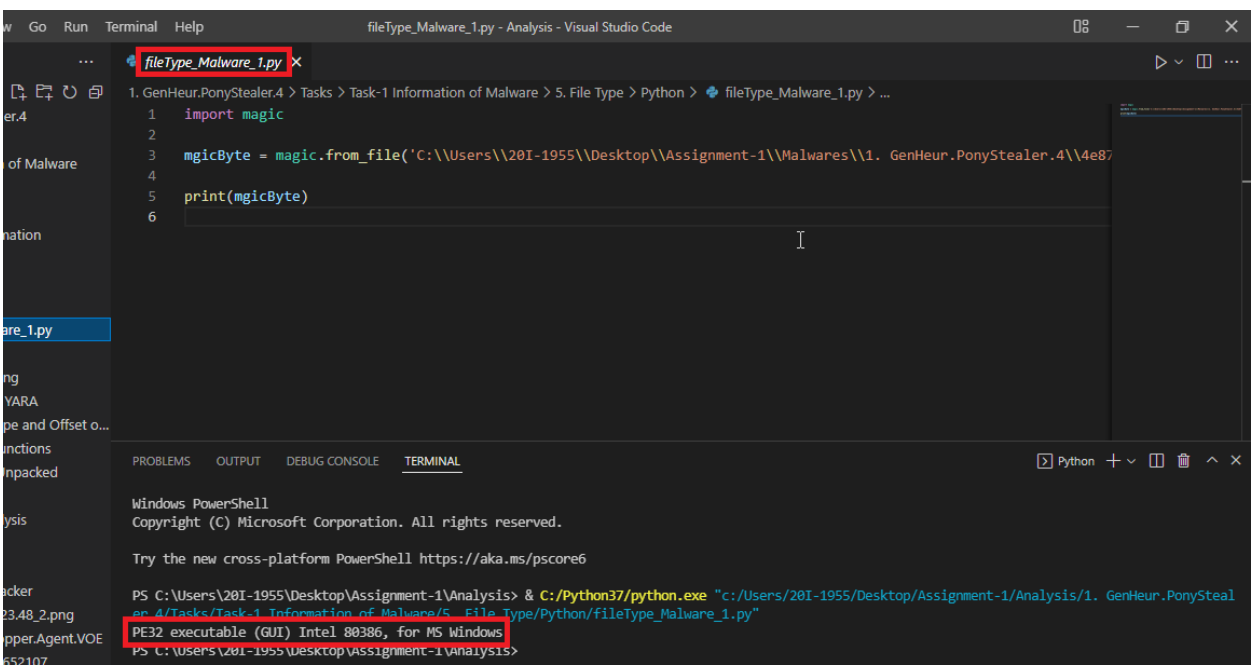

Figure 10: Malware-1 File Type through Python Magic Library

Tool-2 TrID Net

To find the file type of the malware using TrID Net I open the file in TrID Net and find that the malware file is “Win32 Executable Microsoft Visual Basic 6” as it has the “78.7%” match.

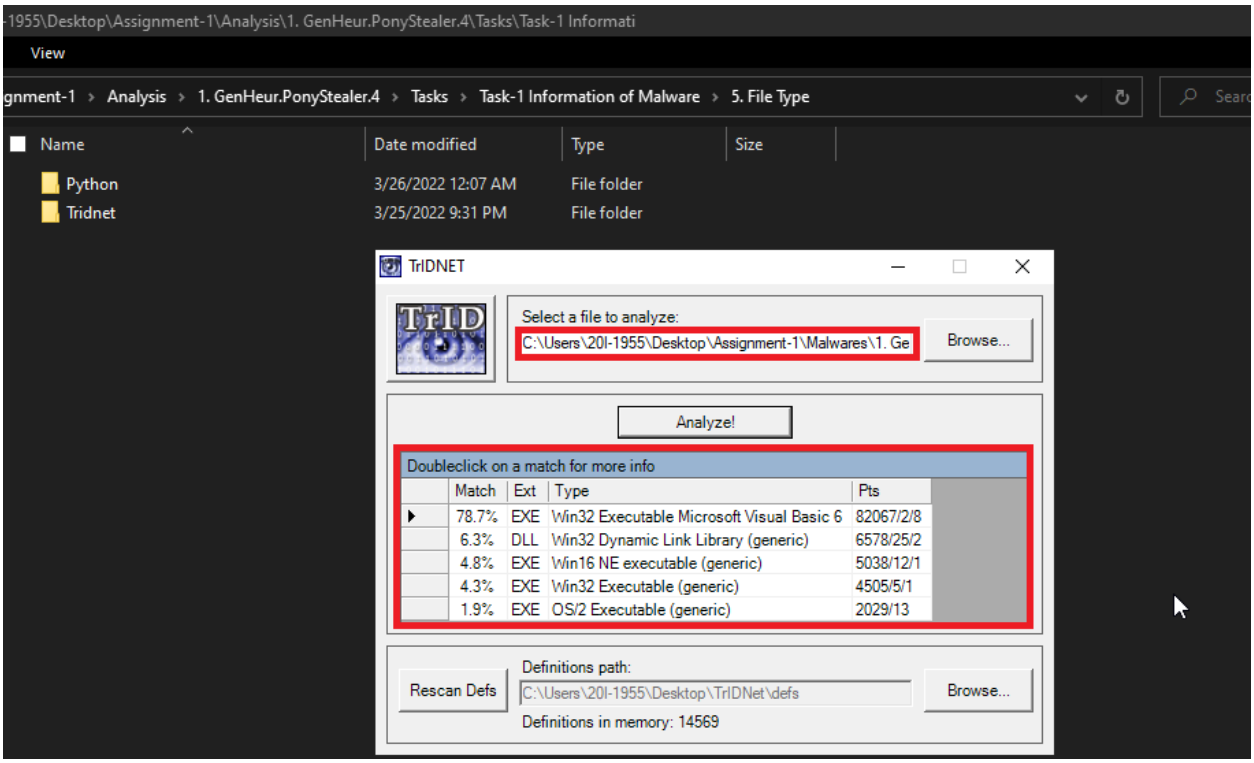

Figure 11: Malware-1 File Type through TrID Net

Task-2 What is the fingerprinting information of the malware file?

Fingerprinting is the method of generating the hashes of the file based on its content. In malware analysis fingerprinting is used to uniquely identify the malware. To find the hashes of the malware file I have used two different tools HashCalc and Python hashlib library.

Tool-1 HashCalc

To find the hash function of the file in HashCalc I open the file in HashCalc and find the hash of the file.

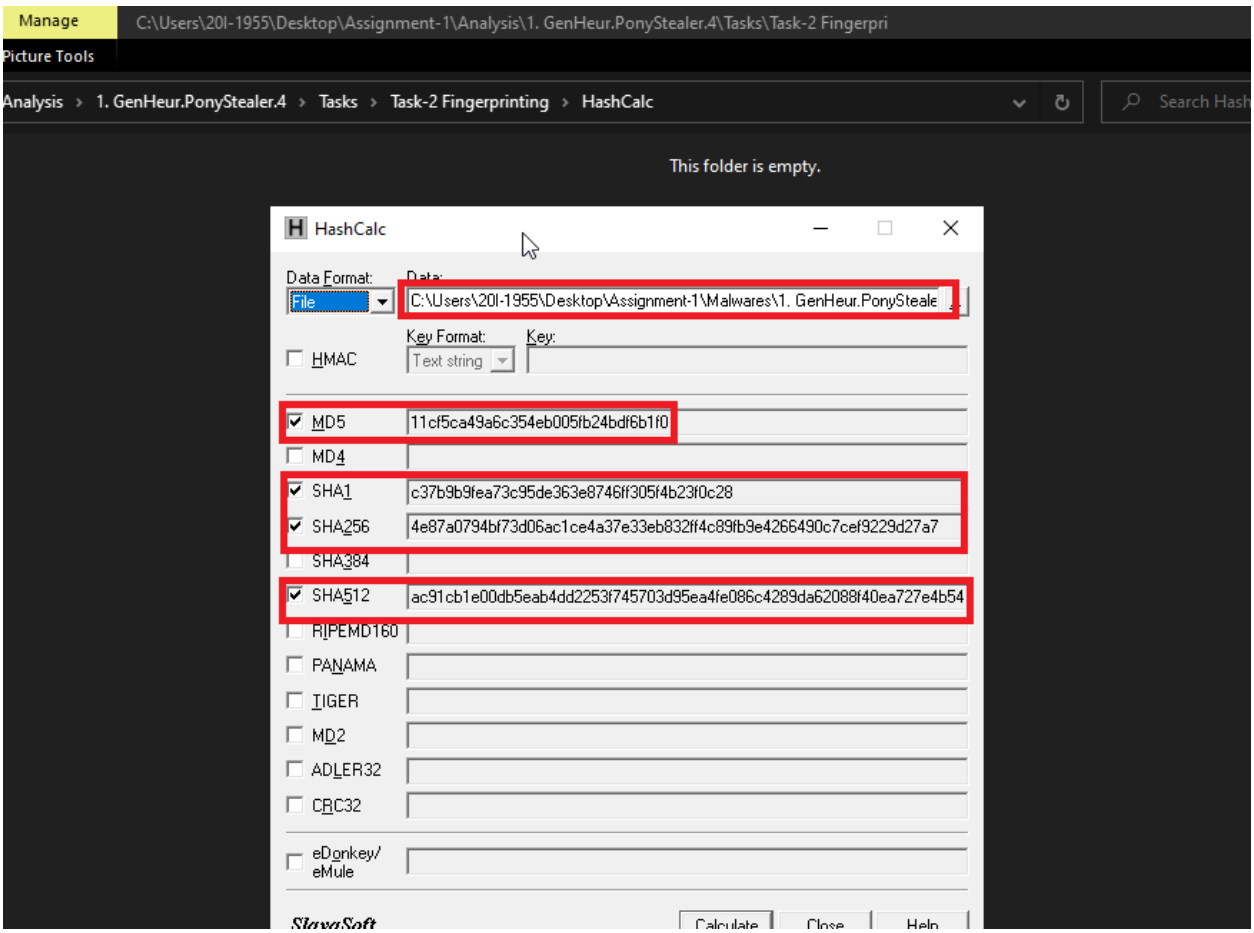

Figure 12: Malware-1 Fingerprinting through HashCalc

Tool-2 Python Hashlib Library

To find the hash of the malware using the python Hashlib library I write the following code.

```
import hashlib

hashMd5 = hashlib.md5(open("C:\\Users\\20I-1955\\Desktop\\Assignment-
1\\Malwares\\1.
GenHeur.PonyStealer.4\\4e87a0794bf73d06ac1ce4a37e33eb832ff4c89fb9e4266490c7cef9
229d27a7\\4e87a0794bf73d06ac1ce4a37e33eb832ff4c89fb9e4266490c7cef9229d27a7",
"rb").read()).hexdigest()
```

```
hashSha1 = hashlib.sha1(open("C:\\Users\\20I-1955\\Desktop\\Assignment-
1\\Malwares\\1.
GenHeur.PonyStealer.4\\4e87a0794bf73d06ac1ce4a37e33eb832ff4c89fb9e4266490c7cef9
229d27a7\\4e87a0794bf73d06ac1ce4a37e33eb832ff4c89fb9e4266490c7cef9229d27a7",
"rb").read()).hexdigest()
hashSha256 = hashlib.sha256(open("C:\\Users\\20I-1955\\Desktop\\Assignment-
1\\Malwares\\1.
GenHeur.PonyStealer.4\\4e87a0794bf73d06ac1ce4a37e33eb832ff4c89fb9e4266490c7cef9
229d27a7\\4e87a0794bf73d06ac1ce4a37e33eb832ff4c89fb9e4266490c7cef9229d27a7",
"rb").read()).hexdigest()
hashSha512 = hashlib.sha512(open("C:\\Users\\20I-1955\\Desktop\\Assignment-
1\\Malwares\\1.
GenHeur.PonyStealer.4\\4e87a0794bf73d06ac1ce4a37e33eb832ff4c89fb9e4266490c7cef9
229d27a7\\4e87a0794bf73d06ac1ce4a37e33eb832ff4c89fb9e4266490c7cef9229d27a7",
"rb").read()).hexdigest()

print("MD5:    ",hashMd5)
print("SHA1:   ",hashSha1)
print("SHA256: ",hashSha256)
print("SHA512: ",hashSha512)
```

After running the code, I found that the hashes of the malware file

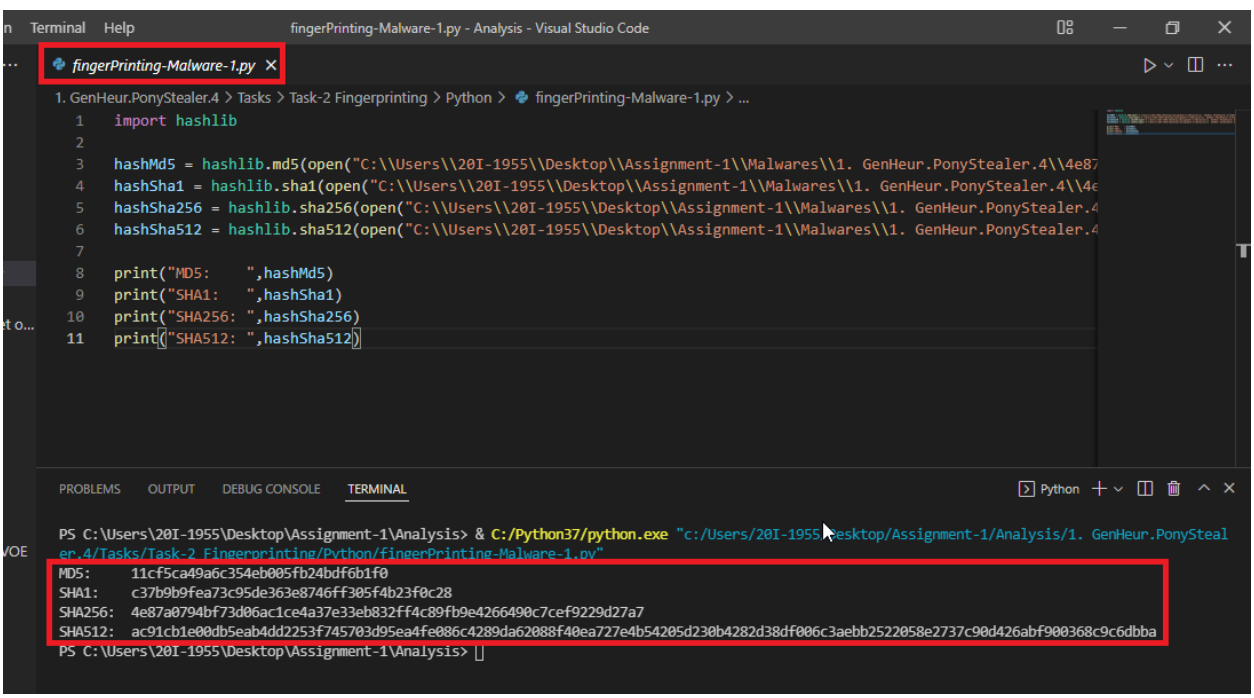

Figure 13: Malware-1 Fingerprinting through Python Hashlib Library

Task-3 What type of malicious strings are in the Malware file?

Strings are the readable characters extracted from the malware file and are used to identify the functionality of the malware. To extract the strings from the malware file I use the tool called "**Strings**" and then to scan the malware according to the extracted strings I write the YARA rule.

Tool-1 Strings

To extract the strings from the malware file and save them in the text file I run the following command

```
strings malware file path > text file name
```

After running the command all the strings are extracted from the malware file and stored in the text file.

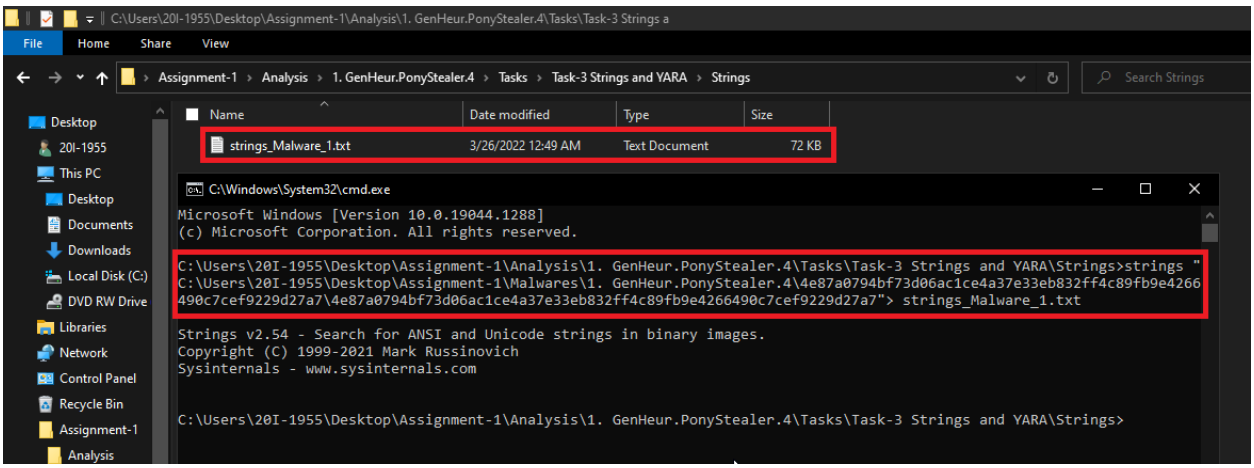

Figure 14: Malware-1 Strings Extraction through Strings Tool

After that, I open the text file to check the strings that were extracted from the malware and found some malicious strings.

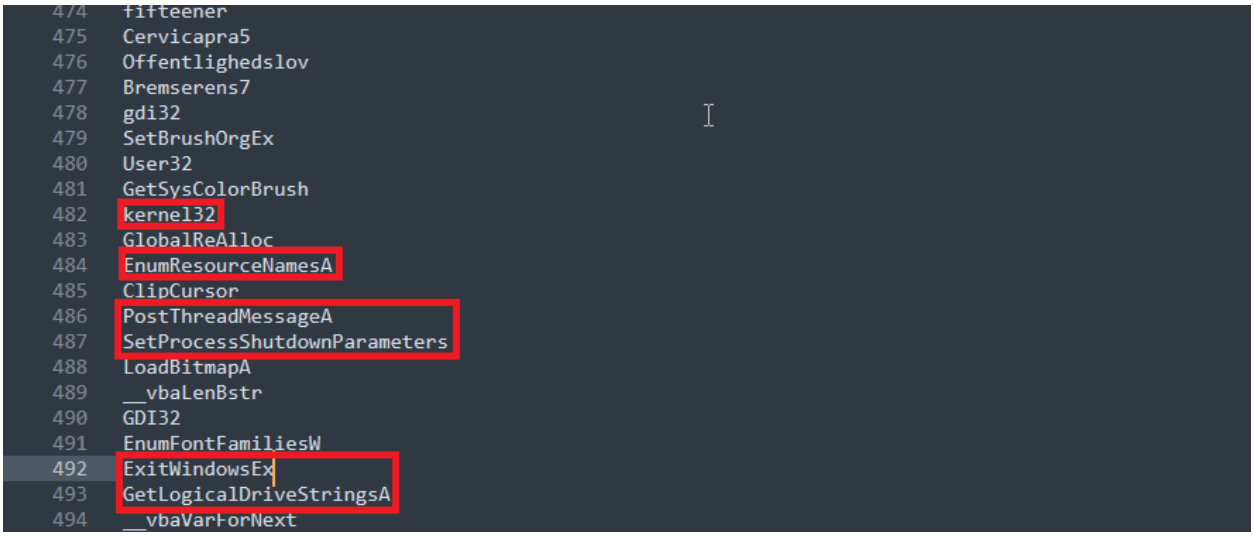

Figure 15: Malware-1 Strings Extracted through Strings Tool

Tool-2 YARA Rule

After extracting the string my next task is to write the YARA rule that scans the file and triggers the warning if it finds the malicious string in the malware file. For that, I write the following rule

```
rule malware_GenHeur_PonyStealer {
    meta:
        malwareName = "GenHeur.PonyStealer.4"
        ruleAuthor = "20I-1955_Muhammad Osama Khalid"
    strings:
        $str1 = "RegOpenKey" nocase
        $str2 = "RegQueryValue" nocase
        $str3 = "RegCloseKey" nocase
        $str4 = "kernel32" nocase
        $str5 = "shell32" nocase
    condition:
        any of them
}
```

Note: These strings are extracted from analyzing all the malware samples that are given for this assignment.

After that, I run the rule and find that the rule that I write works successfully as it detects the malicious string extracted above in the malware file.

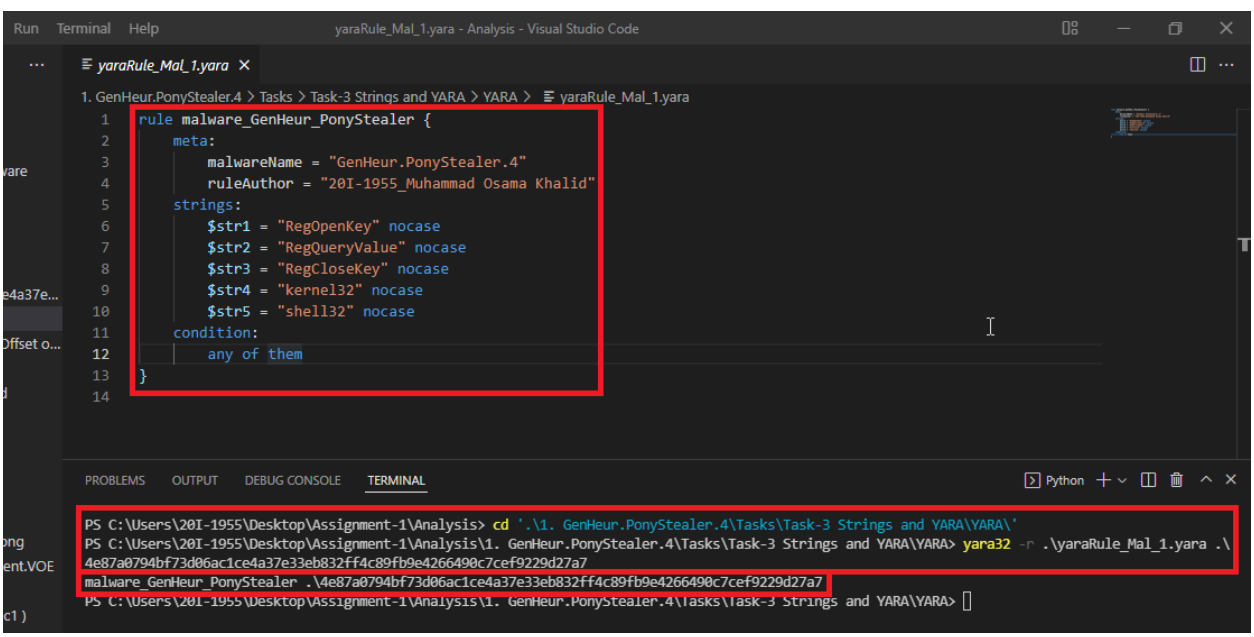

Figure 16: Malware-1 Malware detection through YARA Rule

Task-4 Number, type, and offset of different sections of Malware?

Sections contain the important information about the file as it describes how the executable is made. To find the number, type, and offset of different sections of malware files I use three different tools PE Studio and Exeinfo.

Tool-1 PE Studio

To find the number, type, and offset of different sections of the malware file I open the file in the PE Studio, then from the left panel first I click on "**file-header**" and then from the right panel I found the column named "**sections**". From there I found that this malware file has three sections and to find then what they are I click on "**sections**" in the left panel. After that from the right panel, I found that this malware file has three sections "**.text**", "**.data**" and "**.rsrc**". The information in each section is as follows

No. of Section	Type	Offset
1	.text	0x00401000
2	.data	0x004BB000
3	.rsrc	0x004C3000

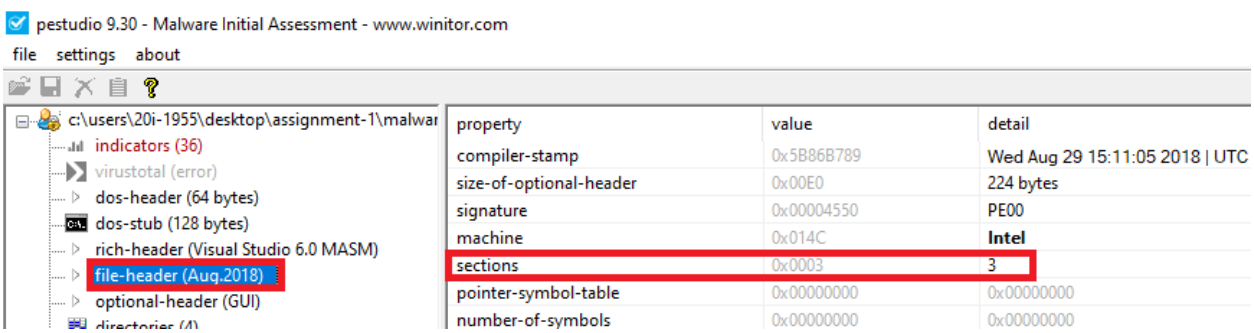

Figure 17: Malware-1 Information of Sections through PE Studio

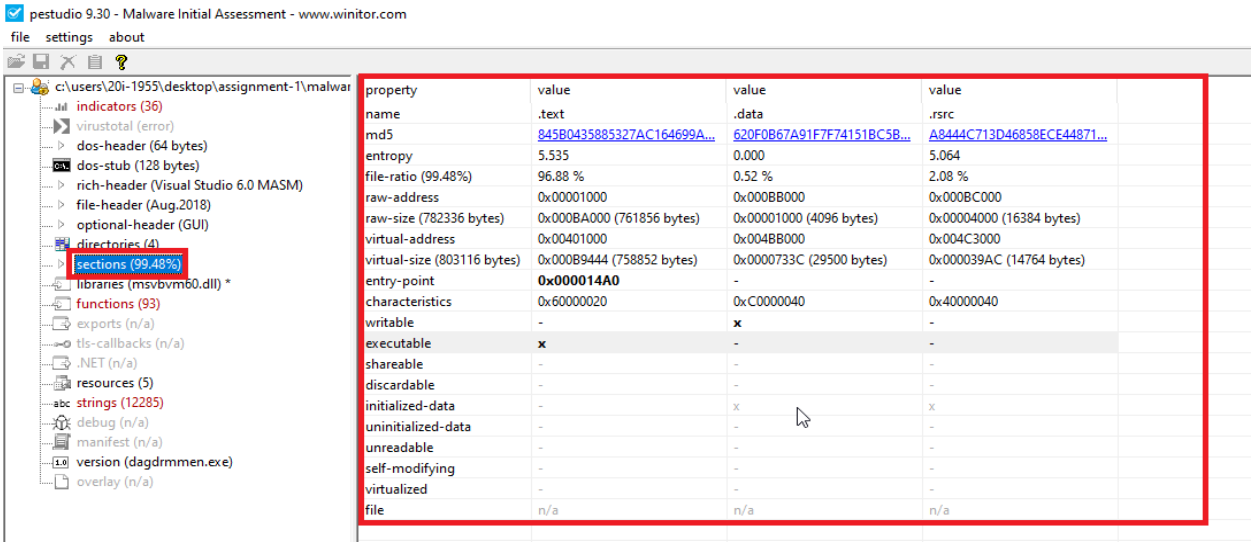

Figure 18: Malware-1 Information of Sections through PE Studio

Task-5 Which DLLs and their number of functions are used by Malware?

DLLs and functions are used to identify what actions will the file perform when we execute it. To find what DLLs and functions this malware used I used the tool PE Studio.

Tool-1 PE Studio

To find the DLLs and their number of functions used by the malware file I open the file in the PE Studio, then from the left panel first I click on "**libraries**" and then from the right pane I found that this malware will only use one DLL "msvbvm600.dll" and with this DLL it will use 93 functions.

No. of DLL	Name	No. of functions
1	msvbvm600.dll	93

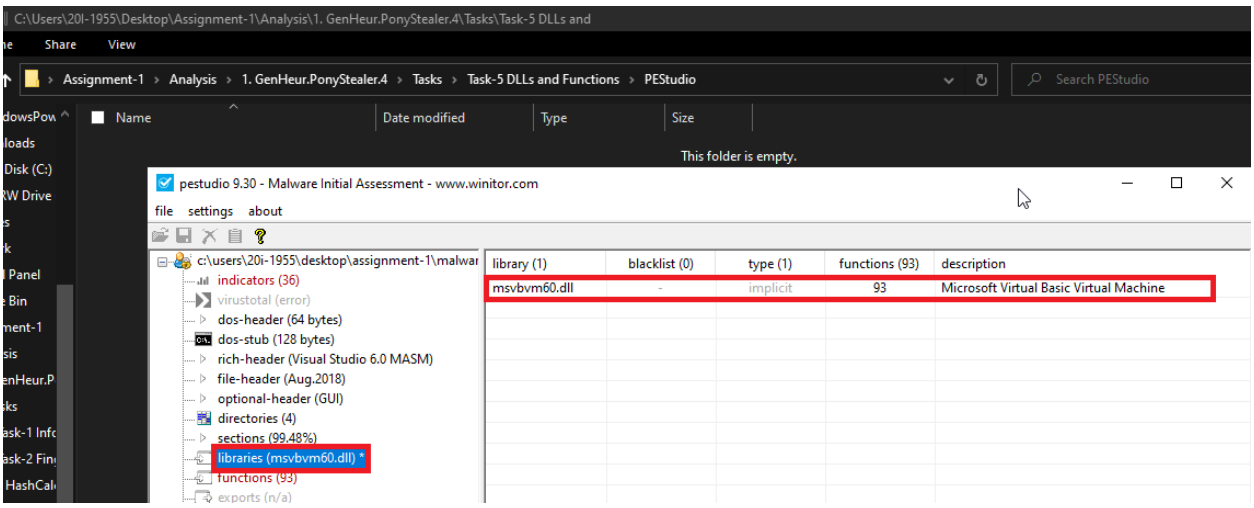

Figure 19: Malware-1 Information of DLLs and functions through PE Studio

Task-6 What is packing information if Malware is packed?

Packing is used by malware developers to hide the malicious code of the malware. If the malware is packed then when it executes it first unpack itself and then perform the action which it is designed for. The information about the packing is stored in the malware file so that it helps the malware to unpack itself during the execution process. To check whether the malware is packed or not I use the tool Exeinfo.

Tool-1 Exeinfo

To find whether the malware file is packed or not I open the file in Exeinfo and at the bottom I found that the malware is not packed.

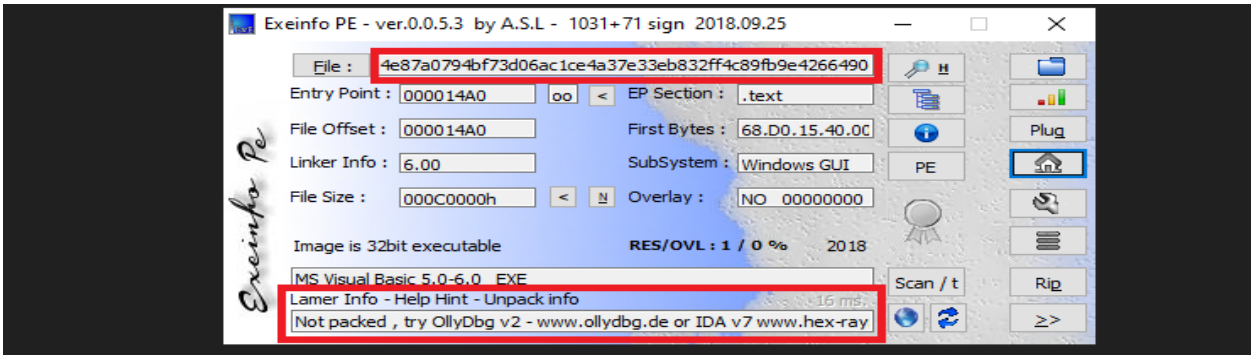

Figure 20: Malware-1 Packed or unpacked malware information through Exeinfo

Task-7 VirusTotal analysis of the malware file

Virus total is the online malware analysis tool that is used to scan the malicious file through different antivirus search engines in one go that then defines whether the file is malicious or not. To scan the malware file from the virus total I visit the website and upload the malware file. After some time, I found that the file is triggered as malicious by 60 antivirus search engines.

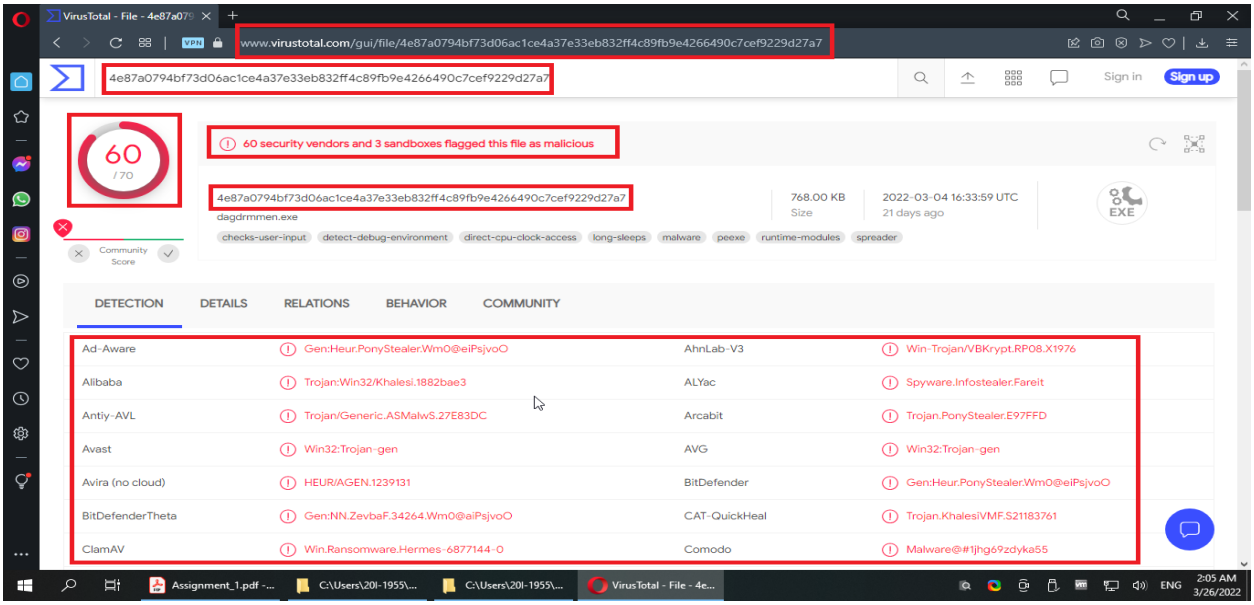

Figure 21: Malware-1 VirusTotal Analysis

Moreover, in the details tab, I find more information about the malware that I already explained in the previous tasks.

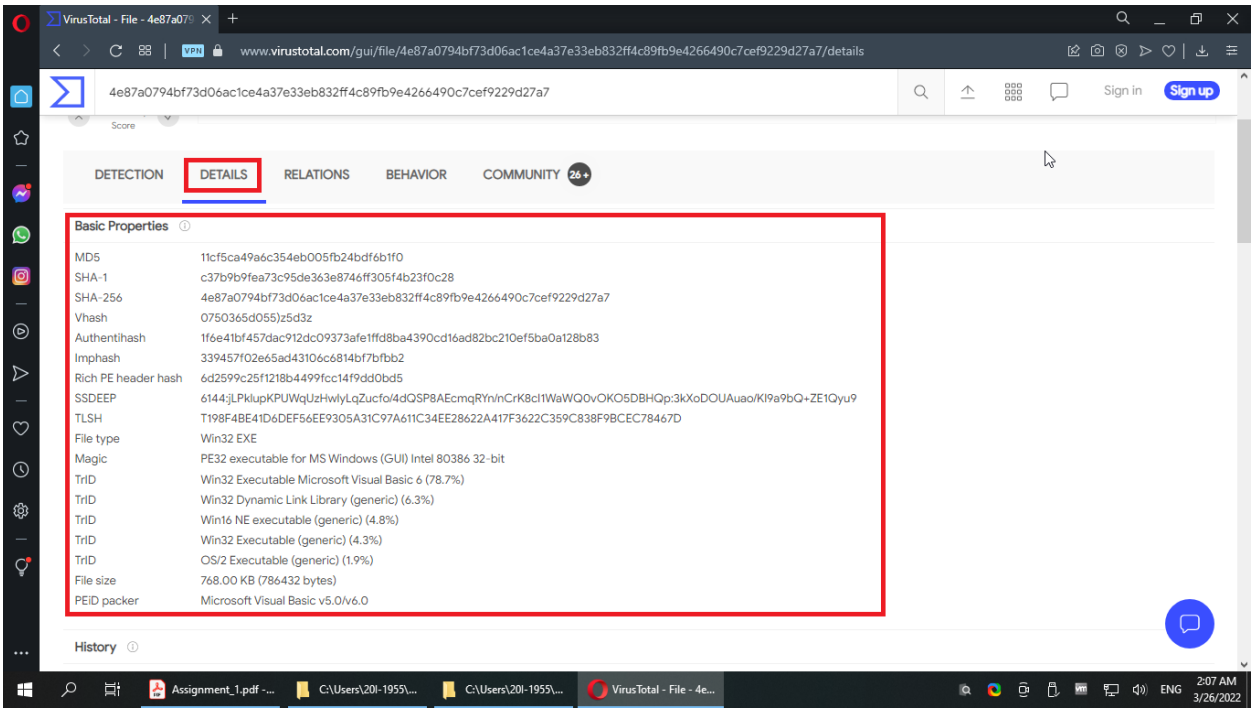

Figure 22: Malware-1 VirusTotal Analysis

Task-8 Analysis of the malware file through Hybrid Analysis

Like VirusTotal it is also the online malware analysis tool that is used to scan the malicious file. To scan the malware file from Hybrid Analysis I visit the website and upload the malware file. After some time, I found that the file is triggered as malicious by antivirus search engines.

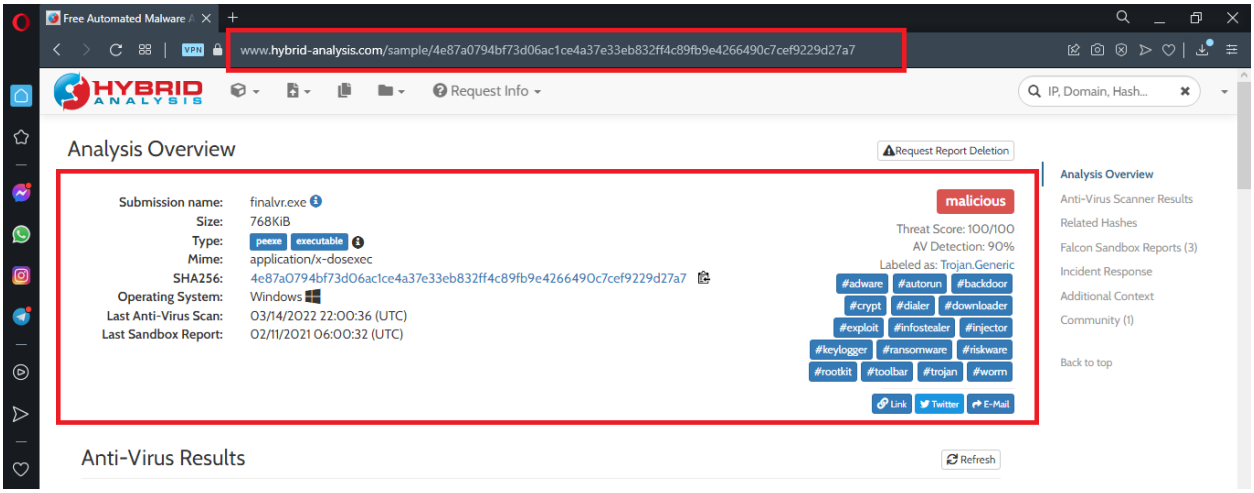

Figure 23: Malware-1 Analysis through Hybrid Analysis

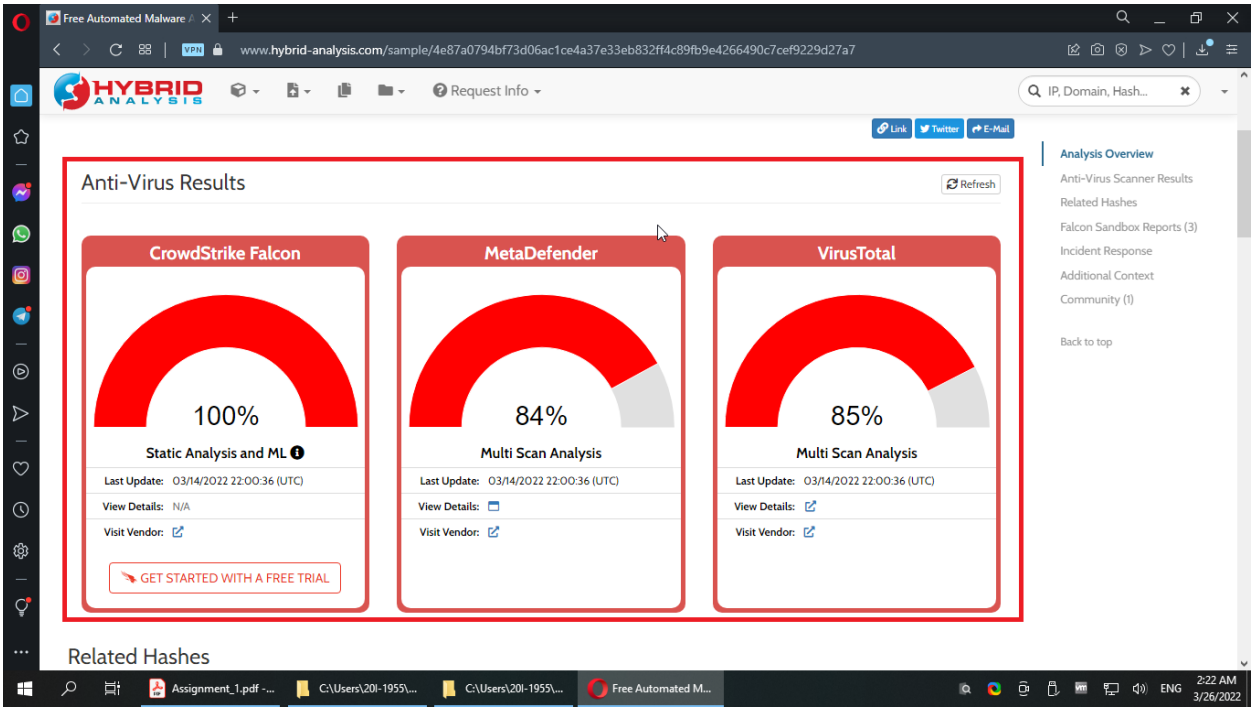

Figure 24: Malware-1 Analysis through Hybrid Analysis

Malware-2 Dropped:Trojan.Dropper.Agent.VOE

This malware belongs to the family of Trojan that installs and executes other malicious programs on the victim machine.

Task-1 What is complete information malware have

In this task, I have to find the basic information of the malware that includes identifying magic byte, signatures, machine info, exe type, and file type.

Task-1.1 Magic Byte

To find the magic byte of the malware there are a bunch of tools available and for this task, I have used three different tools Notepad++, HxD, and CFF Explorer.

Tool-1 Notepad++

To find the magic byte of the malware file I open the file in Notepad++, then from the plugin tab click on "HEX-Editor" and then click on "View in HEX". On viewing the file in HEX-Editor I found that the magic byte of the malware file is "**MZ**". This means that the malware file is executable. Moreover, I also find that this file can't be able to run in DOS mode.

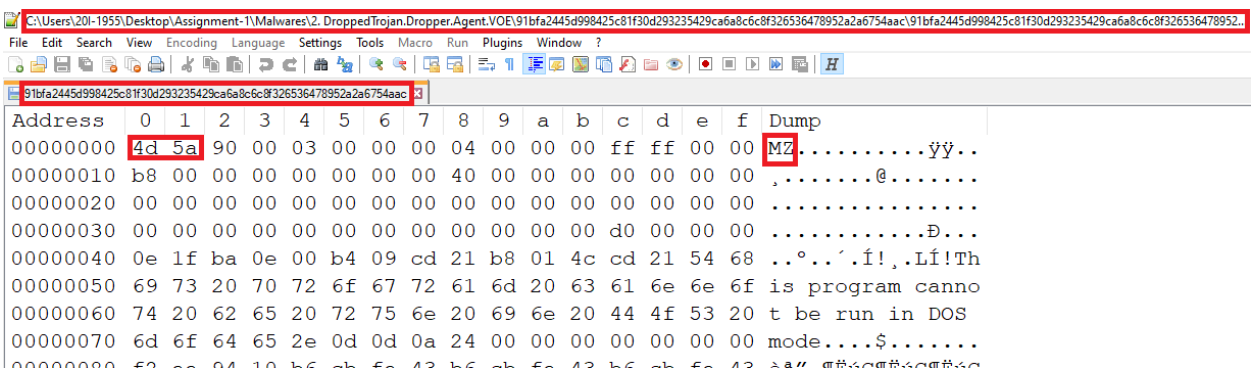

Figure 25: Malware-2 Magic Byte through Notepad++

Tool-2 HxD

To find the magic byte of the malware file I open the file in the HxD and find that the magic byte of the file is "**MZ**" which means that the malware file is executable. Moreover, I also find that this file can't be able to run in DOS mode.

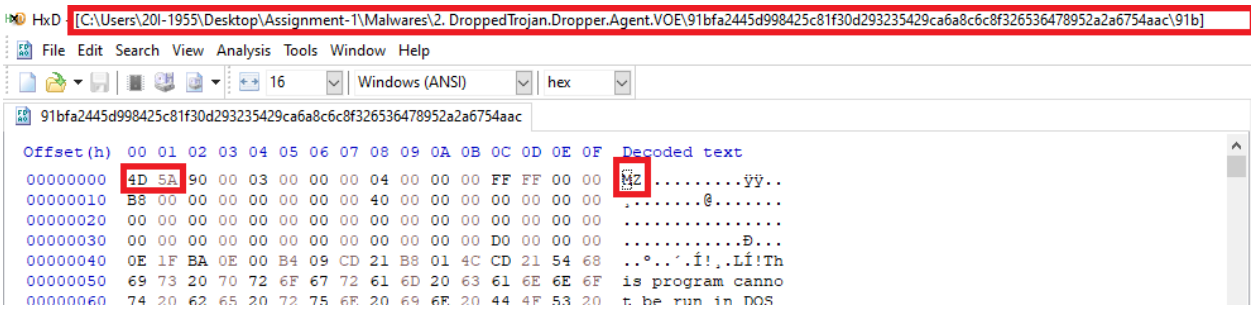

Figure 26: Malware-2 Magic Byte through HxD

Tool-3 CFF Explorer

To find the magic byte of the malware file I open the file in the CFF Explorer, then from the left panel I click on "**DOS Header**". In the "**DOS Header**" I find the magic byte in the "e_magic" column that is "**5A 4D**" which is equivalent to "**4D 5A**" because in CFF Explorer hexadecimal are represented in right to left order. So "**4D = M**" and "**5A = Z**" mean that the malware file is executable.

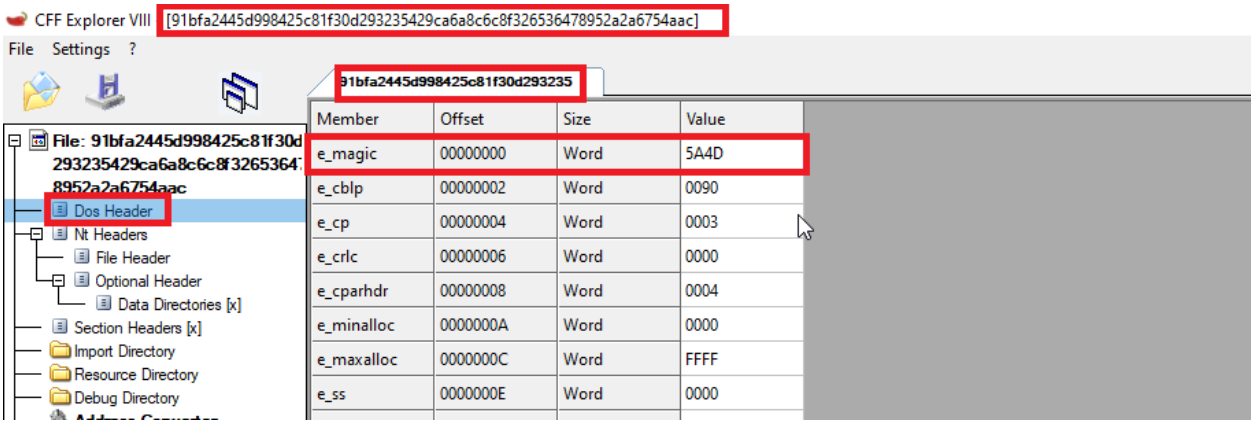

Figure 27: Malware-2 Magic Byte through CFF Explorer

Task-1.2 File Signature

A file signature is used to identify the content of the file. This information is stored in the file's header. To find the file signature of the malware I have used two different tools PE Studio and CFF Explorer.

Tool-1 PE Studio

To find the file signature of the malware file I open the file in the PE Studio, then from the left panel I click on "**file-header**" and then from the right panel I found the column named "**signature**". From there I found that the signature of the file is "**PE**" which means that this file is portable executable.

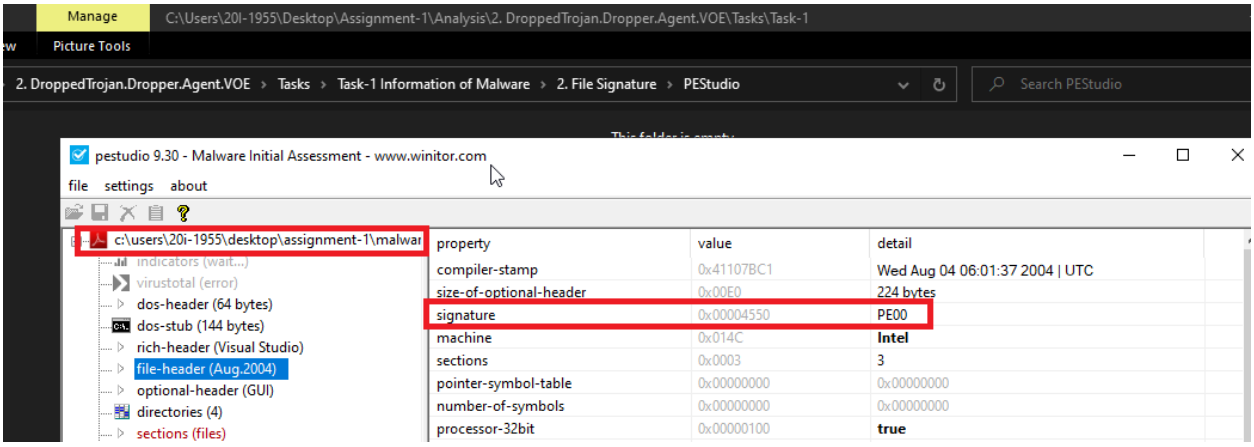

Figure 28: Malware-2 File Signature through PE Studio

Tool-2 CFF Explorer

To find the file signature of the malware file I open the file in the CFF Explorer, then from the left panel I click on "**Nt Headers**" and then from the right panel I found the column named "**signature**". From there I found that the signature of the file is "**45 50**" which is equivalent to "**50 45**" because in CFF Explorer hexadecimal are represented in right to left order. So "**50 = P**" and "**45 = E**" which means that this file is portable executable.

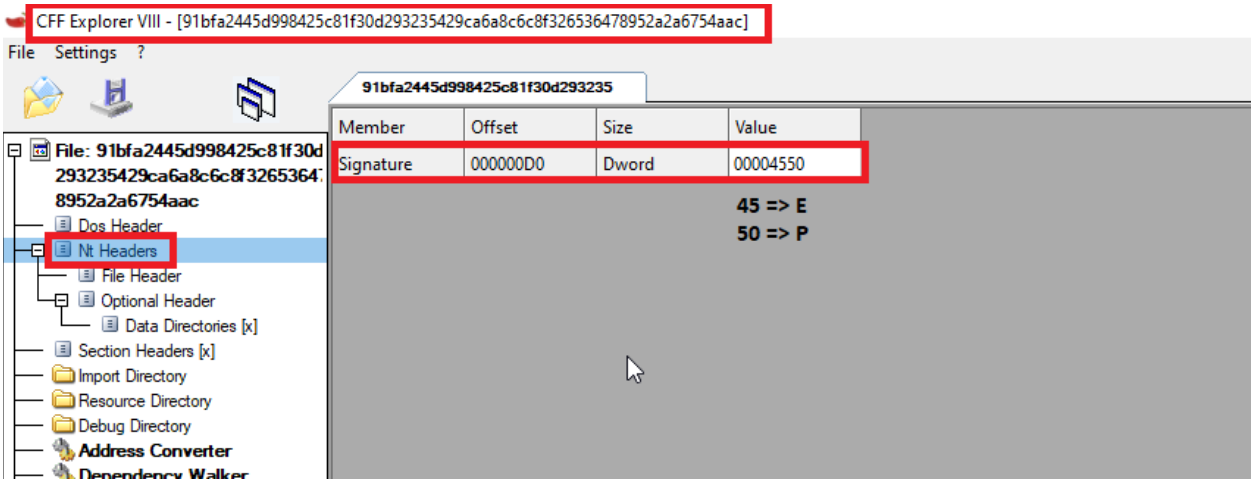

Figure 29: Malware-2 File Signature through CFF Explorer

Task-1.3 Machine Information

To find the machine information of the malware I have used two different tools PE Studio and CFF Explorer.

Tool-1 PE Studio

To find the file signature of the malware file I open the file in the PE Studio, then from the left panel I click on "**file-header**" and then from the right panel I found the column named "**machine**". From there I found that this malware is designed for Intel-based machines. Other than machine information I also find that when the malware is created from the "**compiler-stamp**" column. This malware is created on Aug 04, 2004, on Wednesday at 04:06 AM. Also find that this malware file has 3 sections and the size of the optional header is 224 bytes.

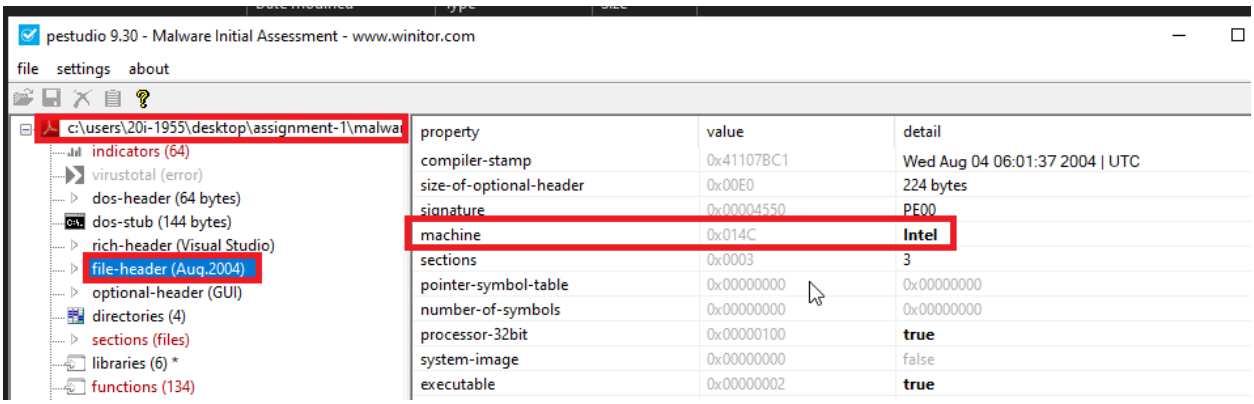

Figure 30: Malware-2 Machine Information through PE Studio

Tool-2 CFF Explorer

To find the file signature of the malware file I open the file in the CFF Explorer, then from the left panel I click on "**file header**" and then from the right panel I found the column named "**Machine**". From there I found that this malware is designed for Intel 386 based machines.

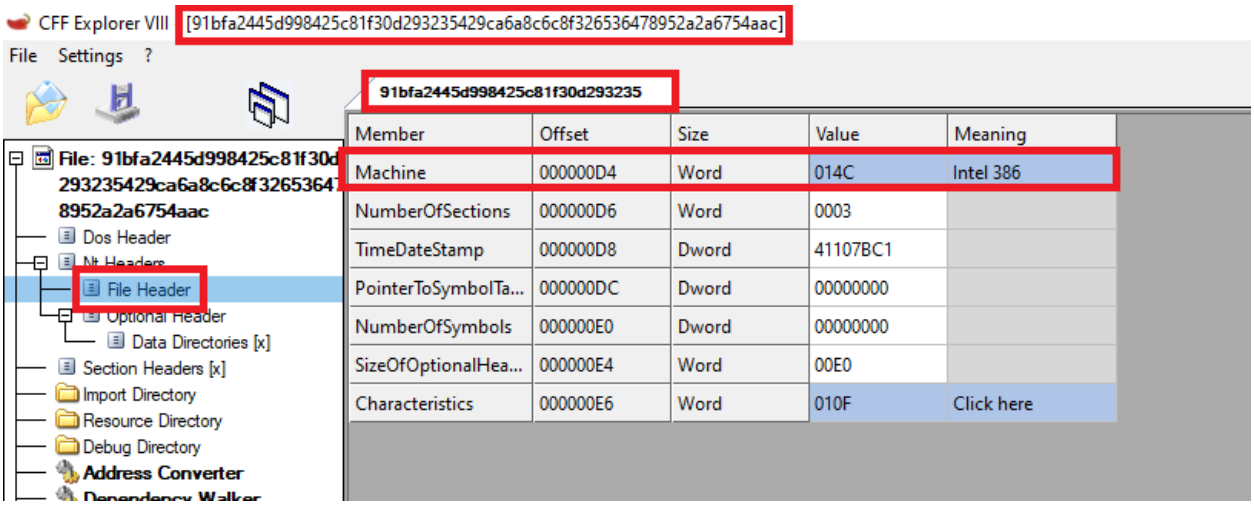

Figure 31: Malware-2 Machine Information through CFF Explorer

Task-1.4 Exe Type

Exe type is used to identify which type of executable it is either 32-bit or 64-bit and to find that I used two different tools PE Studio and CFF Explorer.

Tool-1 PE Studio

To find the exe type of the malware file I open the file in the PE Studio, then from the left panel I click on "**optional-header**" and then from the right panel I found the column named "**magic**". From there I found that the exe type of this malware file is "0x010B" which means that this malware file is a 32-bit portable executable. Other than exe type I also find the entry point of the malware which is ".text".

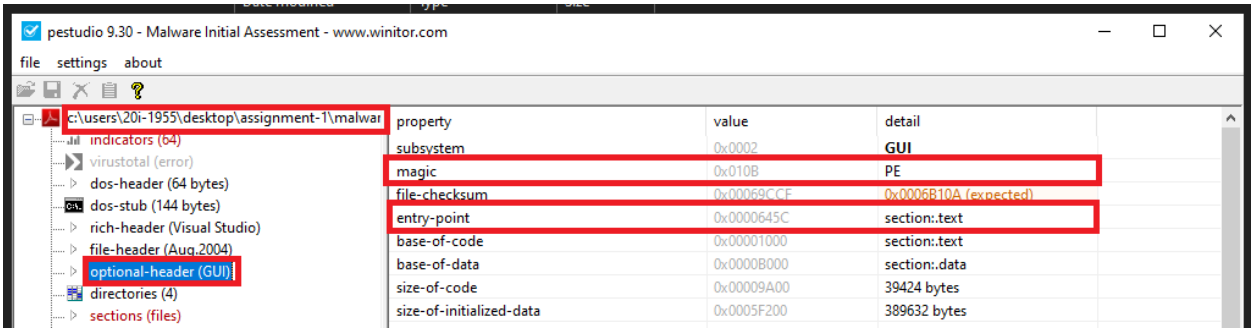

Figure 32: Malware-2 Exe type through PE Studio

Tool-2 CFF Explorer

To find the file signature of the malware file I open the file in the CFF Explorer, then from the left panel I click on "**optional header**" and then from the right panel I found the column named "**Magic**". From there I found that the exe type of this malware file is "010B" which means that this malware file is a 32-bit portable executable. Other than exe type I also find the entry point of the malware which is ".text".

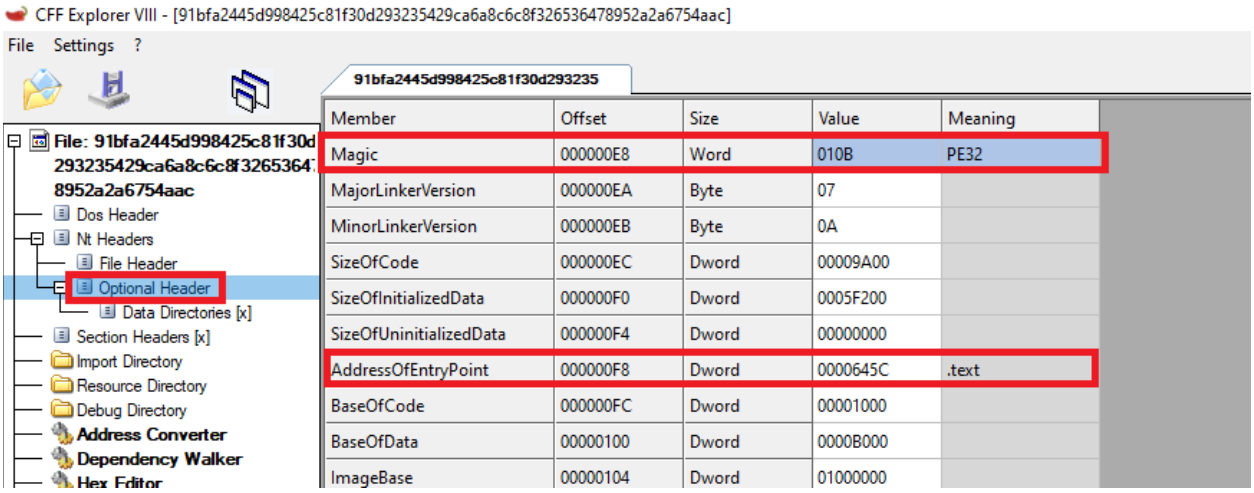

Figure 33: Malware-2 Exe type through CFF Explorer

Task-1.5 File Type

File Type is used to identify which type of file it is. I used two different tools Python Magic Library and TrID Net.

Tool-1 Python Magic Library

To find the file type of the malware using the python magic library I write the following code.

```
import magic

mgicByte = magic.from_file('C:\\Users\\20I-1955\Desktop\\Assignment-
1\Malwares\\2.
DroppedTrojan.Dropper.Agent.VOE\\91bfa2445d998425c81f30d293235429ca6a8c6c8f3265
36478952a2a6754aac\\91bfa2445d998425c81f30d293235429ca6a8c6c8f326536478952a2a67
54aac')

print(mgicByte)
```

After running the code, I found that the file type of malware is “PE32 executable (GUI) Intel 80386, for MS Windows”.

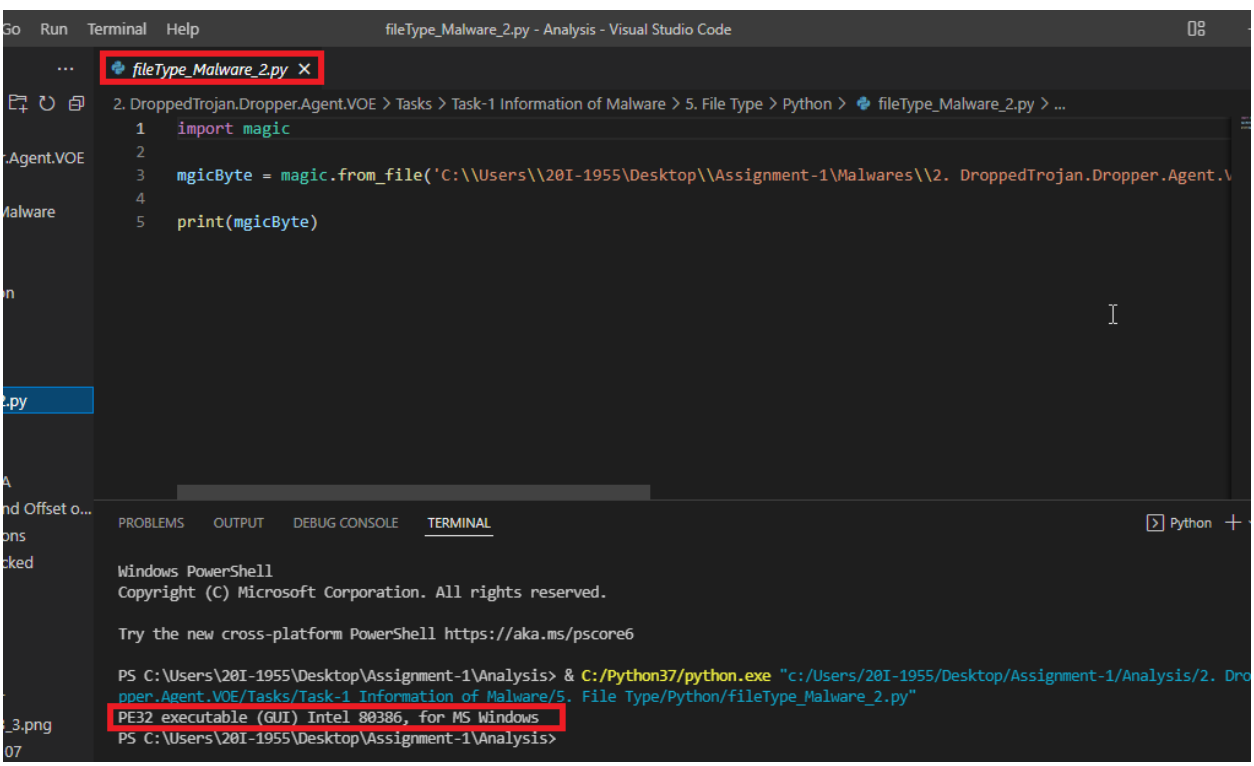

Figure 34: Malware-2 File Type through Python Magic Library

Tool-2 TrID Net

To find the file type of the malware using TrID Net I open the file in TrID Net and find that the malware file is “Win32 MS Cabinet Self Extractor” as it has the “82.6%” match.

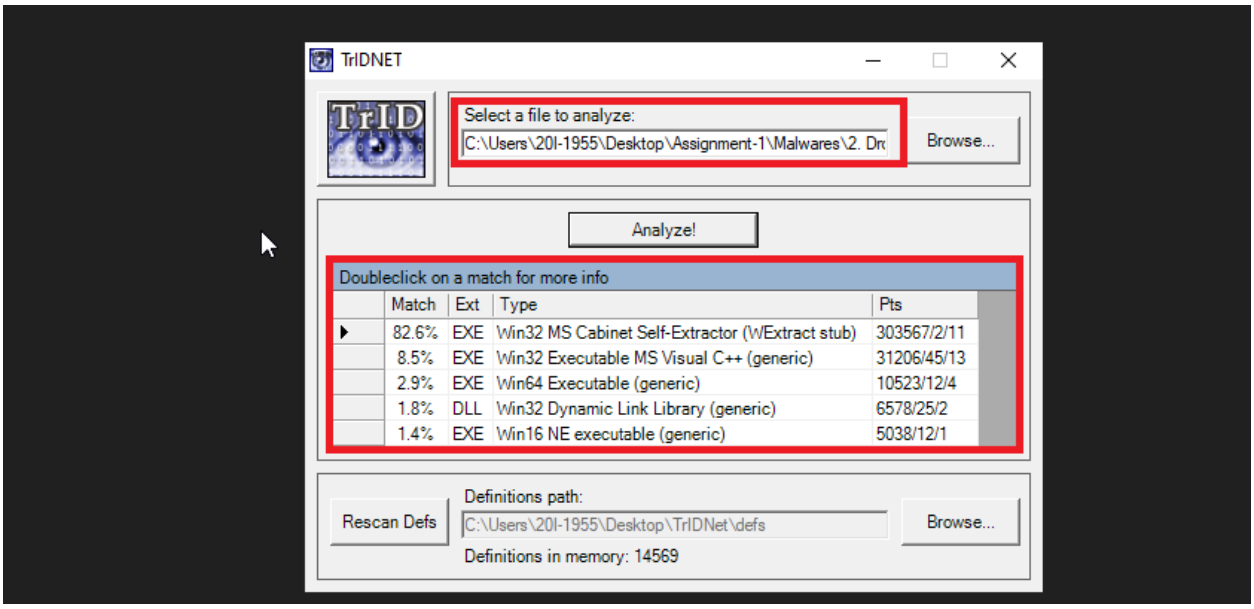

Figure 35: Malware-2 File Type through TrID Net

Task-2 What is the fingerprinting information of the malware file?

To find the hashes of the malware file I have used two different tools HashCalc and Python hashlib library.

Tool-1 HashCalc

To find the hash function of the file in HashCalc I open the file in HashCalc and find the hash of the file.

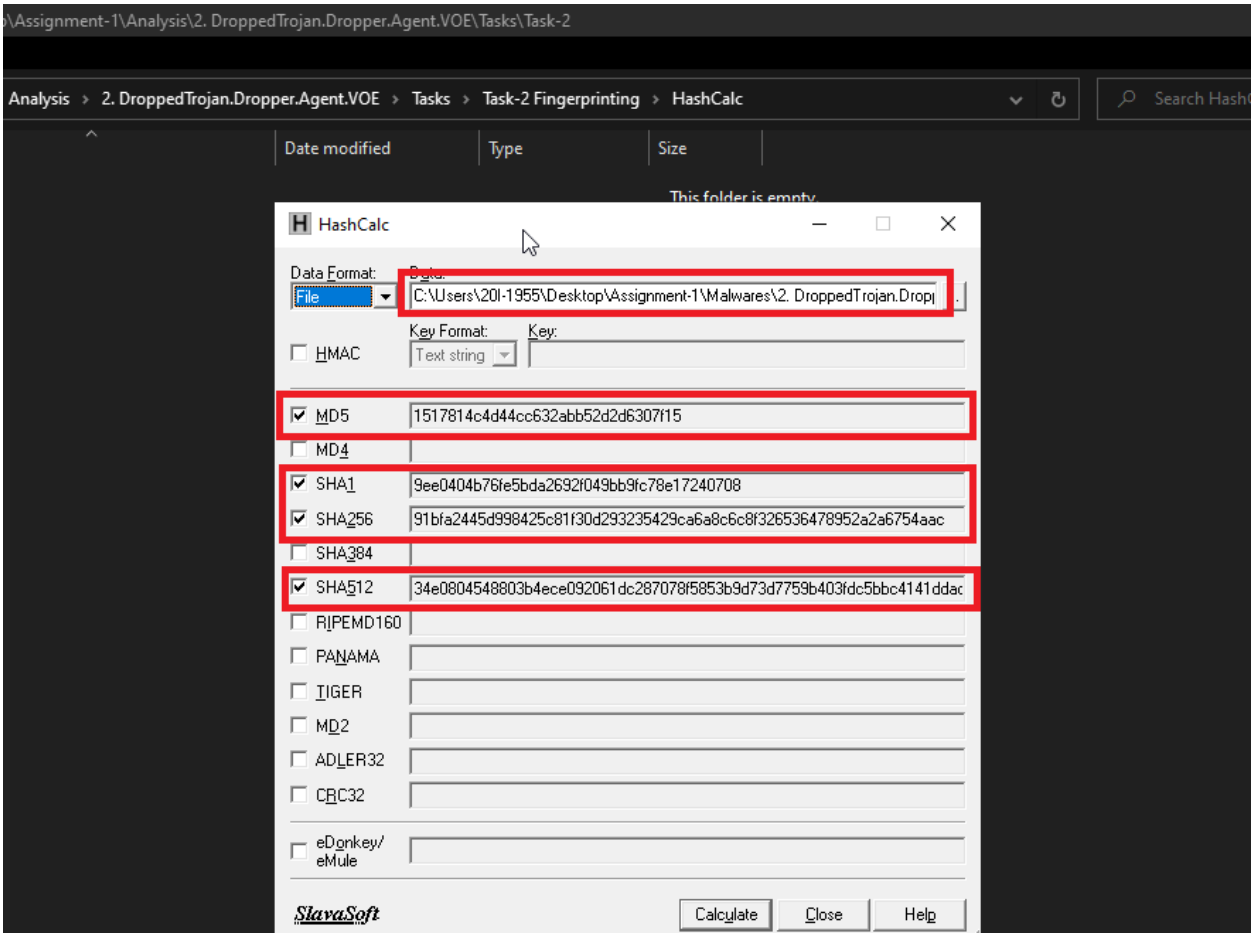

Figure 36: Malware-2 Fingerprinting through HashCalc

Tool-2 Python Hashlib Library

To find the hash of the malware using the python Hashlib library I write the following code.

```
import hashlib

hashMd5 = hashlib.md5(open("C:\\Users\\20I-1955\Desktop\\Assignment-
1\Malwares\\2.
DroppedTrojan.Dropper.Agent.VOE\\91bfa2445d998425c81f30d293235429ca6a8c6c8f3265
36478952a2a6754aac\\91bfa2445d998425c81f30d293235429ca6a8c6c8f326536478952a2a67
54aac", "rb").read()).hexdigest()
```

```
hashSha1 = hashlib.sha1(open("C:\\Users\\20I-1955\Desktop\\Assignment-
1\Malwares\\2.
DroppedTrojan.Dropper.Agent.VOE\\91bfa2445d998425c81f30d293235429ca6a8c6c8f3265
36478952a2a6754aac\\91bfa2445d998425c81f30d293235429ca6a8c6c8f326536478952a2a67
54aac", "rb").read()).hexdigest()
hashSha256 = hashlib.sha256(open("C:\\Users\\20I-1955\Desktop\\Assignment-
1\Malwares\\2.
DroppedTrojan.Dropper.Agent.VOE\\91bfa2445d998425c81f30d293235429ca6a8c6c8f3265
36478952a2a6754aac\\91bfa2445d998425c81f30d293235429ca6a8c6c8f326536478952a2a67
54aac", "rb").read()).hexdigest()
hashSha512 = hashlib.sha512(open("C:\\Users\\20I-1955\Desktop\\Assignment-
1\Malwares\\2.
DroppedTrojan.Dropper.Agent.VOE\\91bfa2445d998425c81f30d293235429ca6a8c6c8f3265
36478952a2a6754aac\\91bfa2445d998425c81f30d293235429ca6a8c6c8f326536478952a2a67
54aac", "rb").read()).hexdigest()

print("MD5:    ",hashMd5)
print("SHA1:   ",hashSha1)
print("SHA256: ",hashSha256)
print("SHA512: ",hashSha512)
```

After running the code, I found that the hashes of the malware file

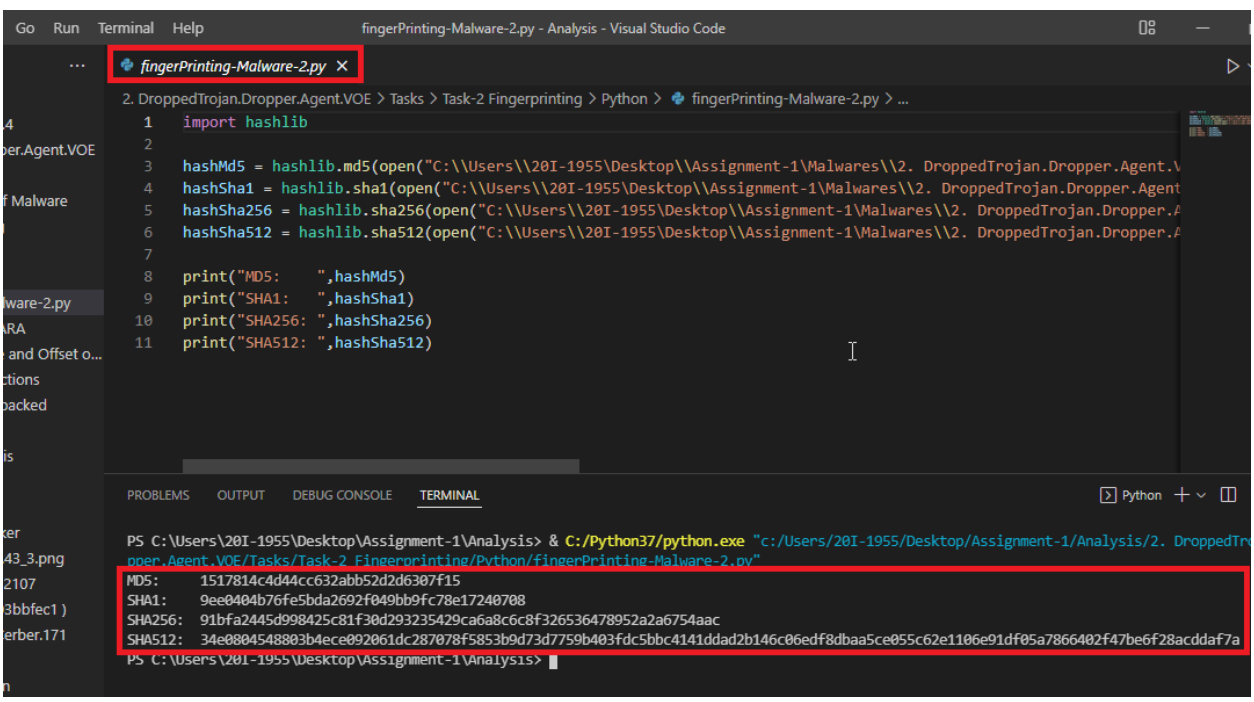

Figure 37: Malware-2 Fingerprinting through Python Hashlib Library

Task-3 What type of malicious strings are in the Malware file?

To extract the strings from the malware file I use the tool called "**Strings**" and then to scan the malware according to the extracted strings I write the YARA rule.

Tool-1 Strings

To extract the strings from the malware file and save them in the text file I run the following command

```
strings malware file path > text file name
```

After running the command all the strings are extracted from the malware file and stored in the text file.

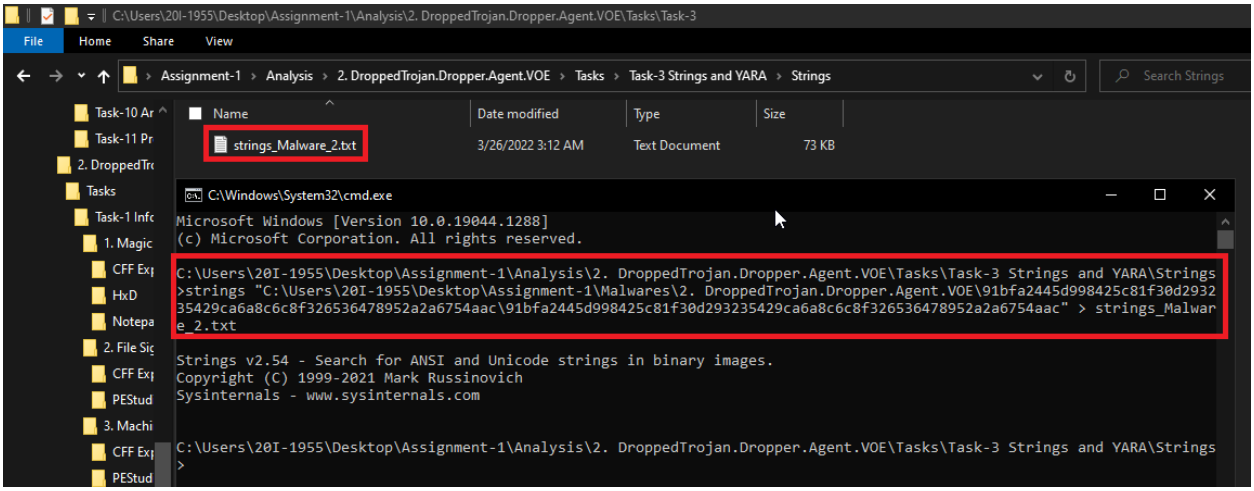

Figure 38: Malware-2 Strings Extraction through Strings Tool

After that, I open the text file to check the strings that were extracted from the malware and found some malicious strings.

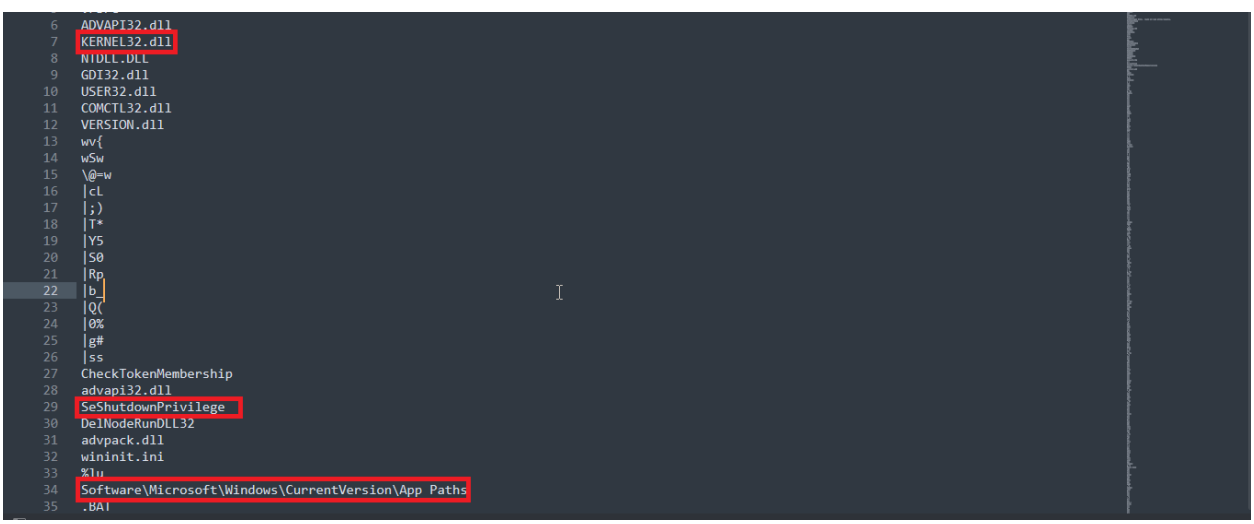

Figure 39: Malware-2 Strings Extracted through Strings Tool

Tool-2 YARA Rule

After extracting the string my next task is to write the YARA rule that scans the file and triggers the warning if it finds the malicious string in the malware file. For that, I write the following rule

```
rule malware_DroppedTrojan_Dropper_Agent_VOE {
    meta:
        malwareName = "DroppedTrojan.Dropper.Agent.VOE"
        ruleAuthor = "20I-1955_Muhammad Osama Khalid"
    strings:
        $str1 = "RegOpenKey" nocase
        $str2 = "RegQueryValue" nocase
        $str3 = "RegCloseKey" nocase
        $str4 = "kernel32" nocase
        $str5 = "shell32" nocase
    condition:
        any of them
}
```

Note: These strings are extracted from analyzing all the malware samples that are given for this assignment.

After that, I run the rule and find that the rule that I write works successfully as it detects the malicious string extracted above in the malware file.

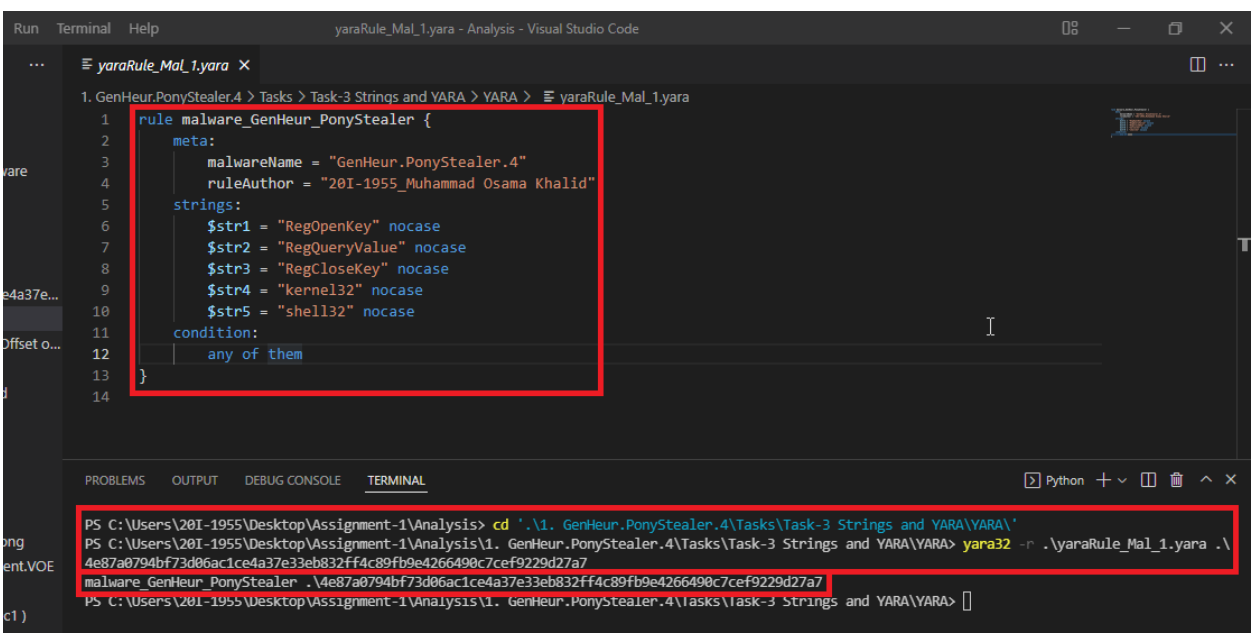

Figure 40: Malware-2 Malware detection through YARA Rule

Task-4 Number, type, and offset of different sections of Malware?

Sections contain the important information about the file as it describes how the executable is made. To find the number, type, and offset of different sections of malware files I use three different tools PE Studio and Exeinfo.

Tool-1 PE Studio

To find the number, type, and offset of different sections of the malware file I open the file in the PE Studio, then from the left panel first I click on "**file-header**" and then from the right panel I found the column named "**sections**". From there I found that this malware file has three sections and to find then what they are I click on "**sections**" in the left panel. After that from the right panel, I found the sections that this malware file has.

No. of Section	Type	Offset
1	.text	0x01001000
2	.data	0x0100B000
3	.rsrc	0x0100D000

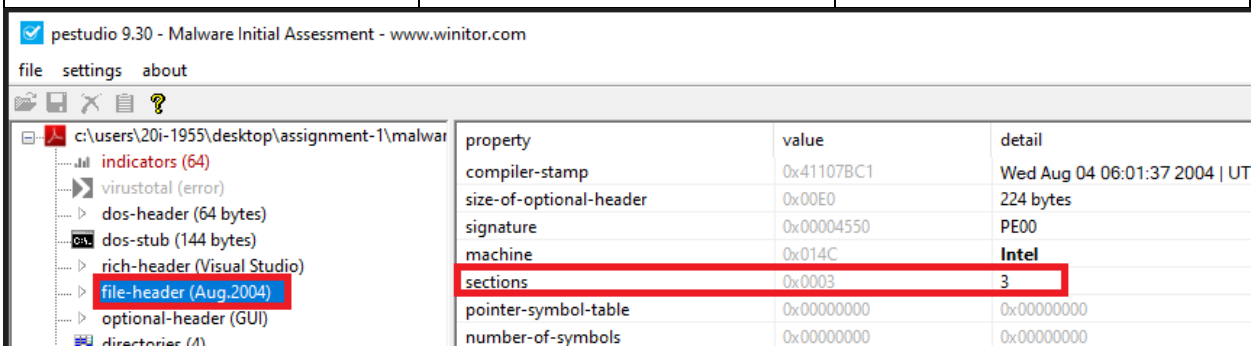

Figure 41: Malware-2 Information of Sections through PE Studio

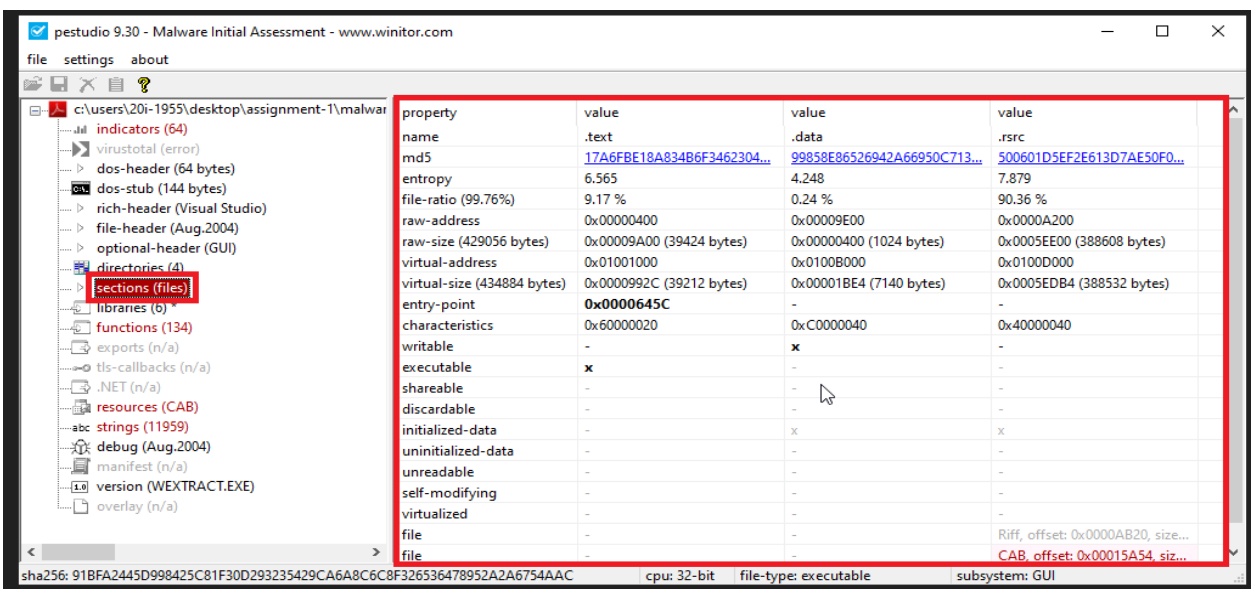

Figure 42: Malware-2 Information of Sections through PE Studio

Task-5 Which DLLs and their number of functions are used by Malware?

To find what DLLs and functions this malware used I used the tool PE Studio.

Tool-1 PE Studio

To find the DLLs and their number of functions used by malware files I open the file in the PE Studio, then from the left panel first I click on "**libraries**" and then from the right pane I found the list of DLLs and their functions used by this malware.

No. of DLL	Name	No. of functions
1	advapi32.dll	14
2	kernel32.dll	84
3	gdi32.dll	1
4	user32.dll	31
5	comctl32.dll	1
6	version.dll	3

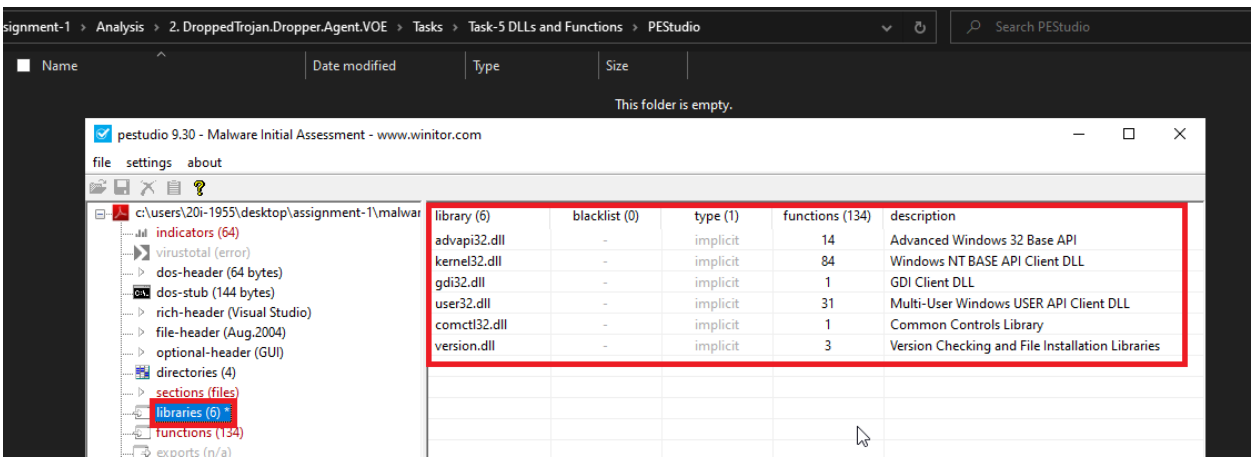

Figure 43: Malware-2 Information of DLLs and functions through PE Studio

Task-6 What is packing information if Malware is packed?

To check whether the malware is packed or not I use the tool Exeinfo.

Tool-1 Exeinfo

To find whether the malware file is packed or not I open the file in Exeinfo and at the bottom I found that the malware is packed with “Generic: MS IExpress” and we can unpack it using “Win32 Cabinet Self-Extractor”.

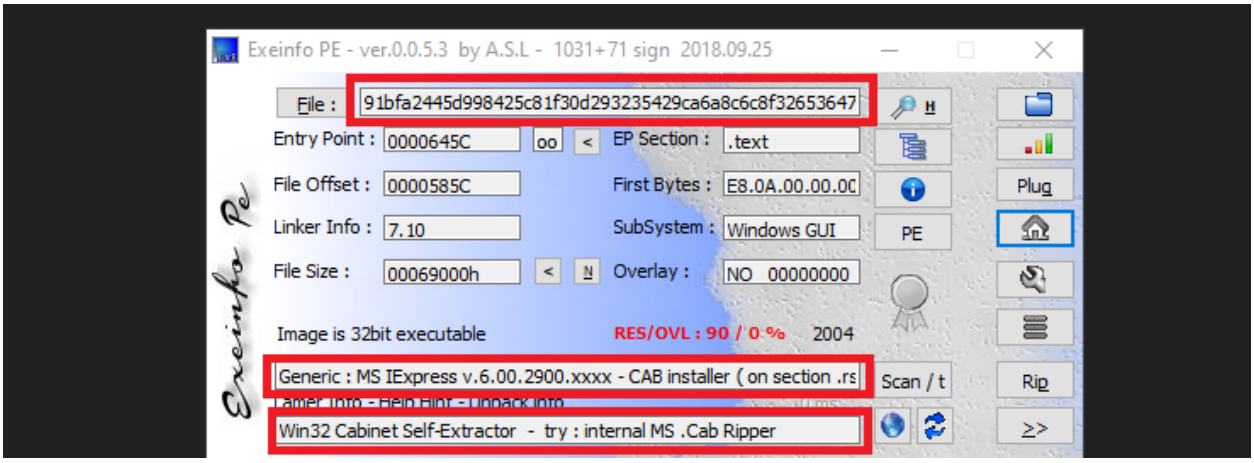

Figure 44: Malware-2 Packed or unpacked malware information through Exeinfo

Task-7 VirusTotal analysis of the malware file

To scan the malware file from the virus total I visit the website and upload the malware file. After some time, I found that the file is triggered as malicious by 56 antivirus search engines.

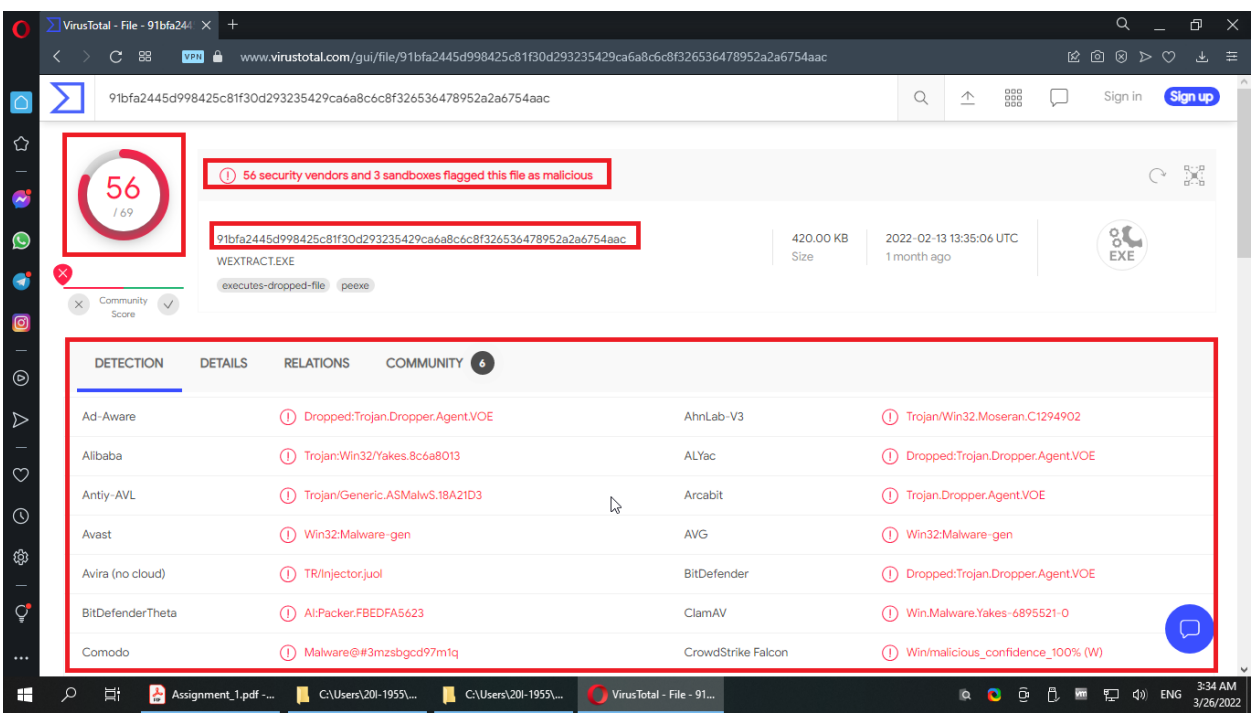

Figure 45: Malware-2 VirusTotal Analysis

Moreover, in the details tab, I find more information about the malware that I already explained in the previous tasks.

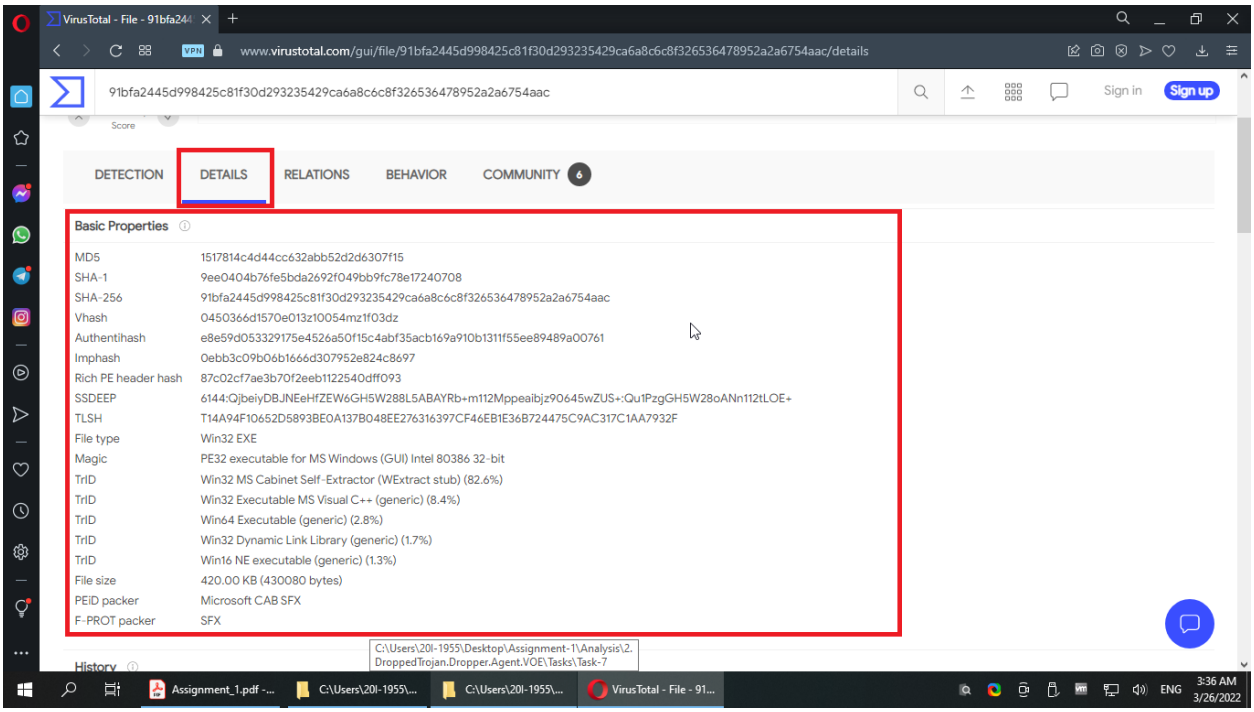

Figure 46: Malware-2 VirusTotal Analysis

Task-8 Analysis of the malware file through Hybrid Analysis

To scan the malware file from Hybrid Analysis I visit the website and upload the malware file. After some time, I found that the file is triggered as malicious by antivirus search engines.

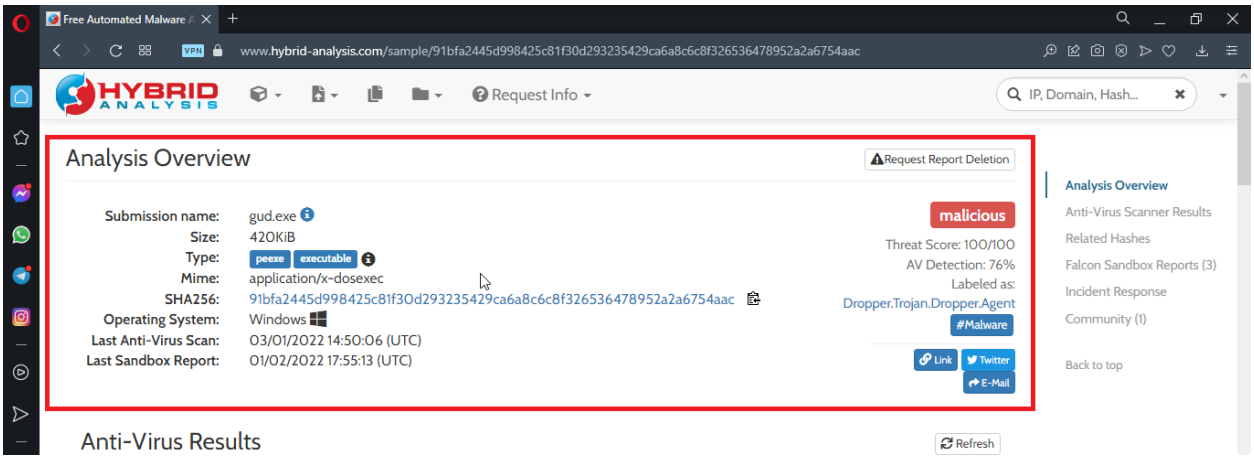

Figure 47: Malware-2 Analysis through Hybrid Analysis

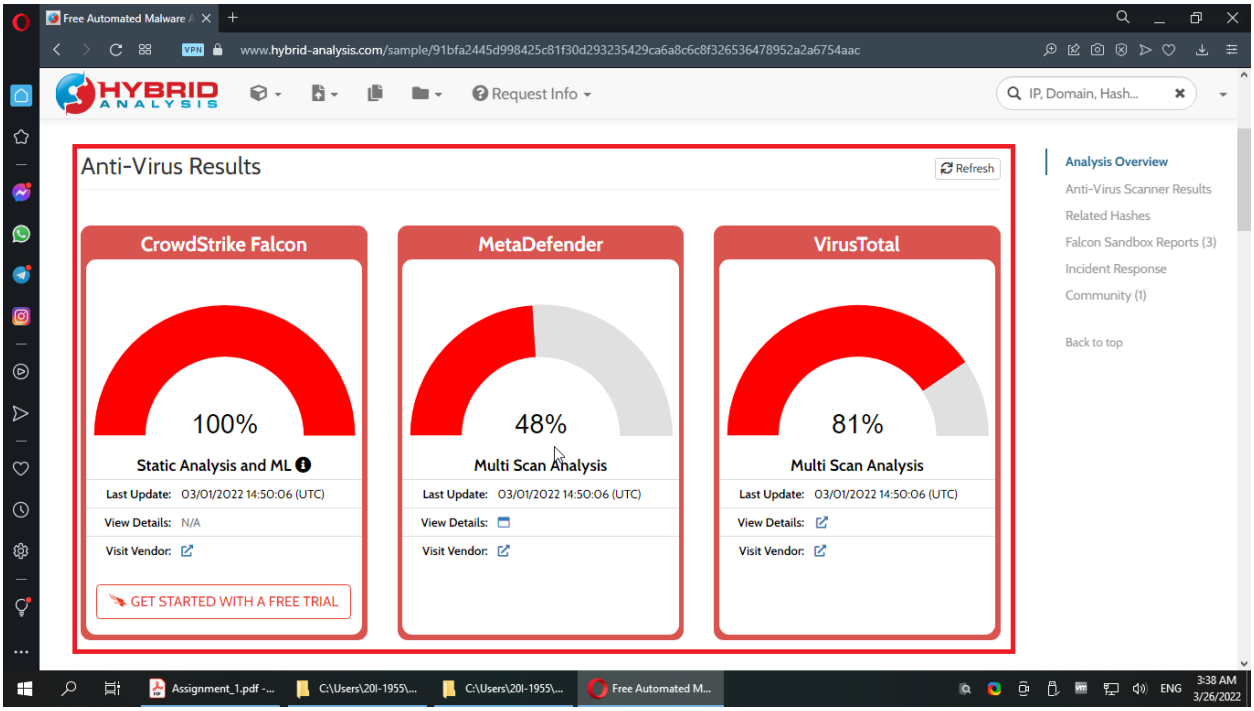

Figure 48: Malware-2 Analysis through Hybrid Analysis

Malware-3 Trojan.GenericKD.3652107

This malware belongs to the family of Trojan that when executed connects to the remote server to perform its malicious activity.

Task-1 What is complete information malware have

In this task, I have to find the basic information of the malware that includes identifying magic byte, signatures, machine info, exe type, and file type.

Task-1.1 Magic Byte

To find the magic byte of the malware there are a bunch of tools available and for this task, I have used three different tools Notepad++, HxD, and CFF Explorer.

Tool-1 Notepad++

To find the magic byte of the malware file I open the file in Notepad++, then from the plugin tab click on “HEX-Editor” and then click on “View in HEX”. On viewing the file in HEX-Editor I found that the magic byte of the malware file is “**MZ**”. This means that the malware file is executable. Moreover, I also find that this file can’t be able to run in DOS mode.

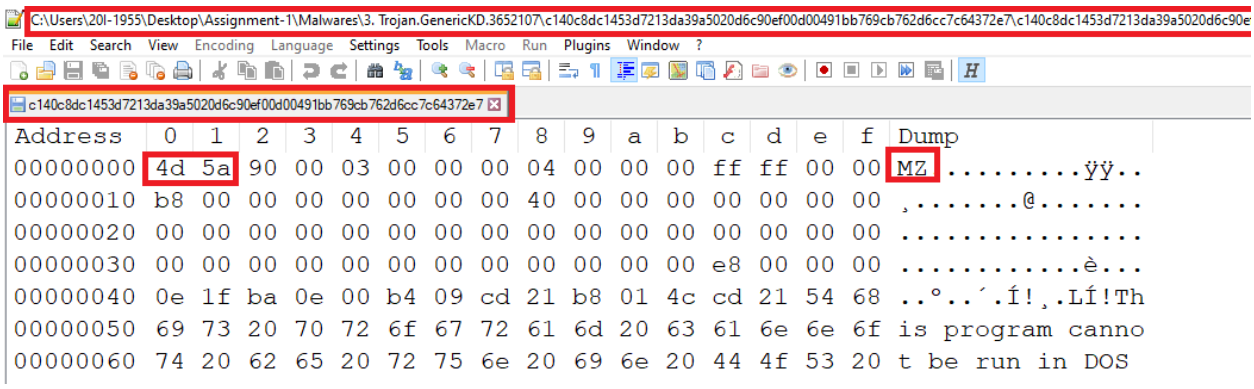

Figure 49: Malware-3 Magic Byte through Notepad++

Tool-2 HxD

To find the magic byte of the malware file I open the file in the HxD and find that the magic byte of the file is “**MZ**” which means that the malware file is executable. Moreover, I also find that this file can’t be able to run in DOS mode.

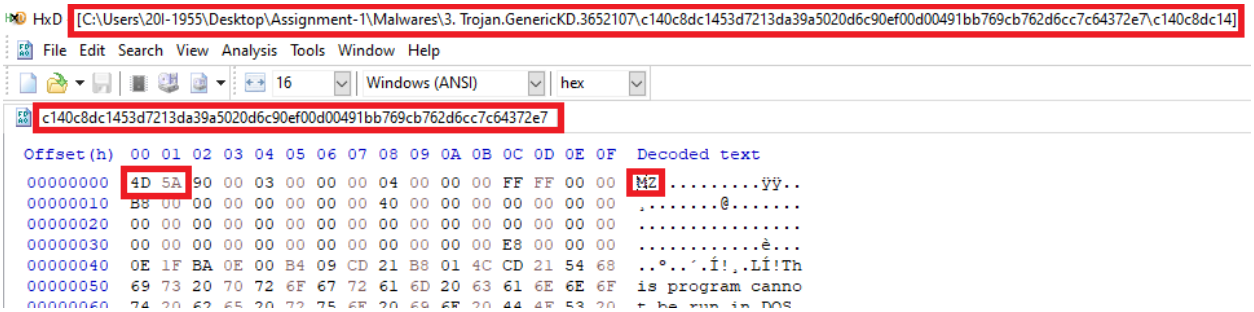

Figure 50: Malware-3 Magic Byte through HxD

Tool-3 CFF Explorer

To find the magic byte of the malware file I open the file in the CFF Explorer, then from the left panel I click on "**DOS Header**". In the "**DOS Header**" I find the magic byte in the "e_magic" column that is "**5A 4D**" which is equivalent to "**4D 5A**" because in CFF Explorer hexadecimal are represented in right to left order. So "**4D = M**" and "**5A = Z**" mean that the malware file is executable.

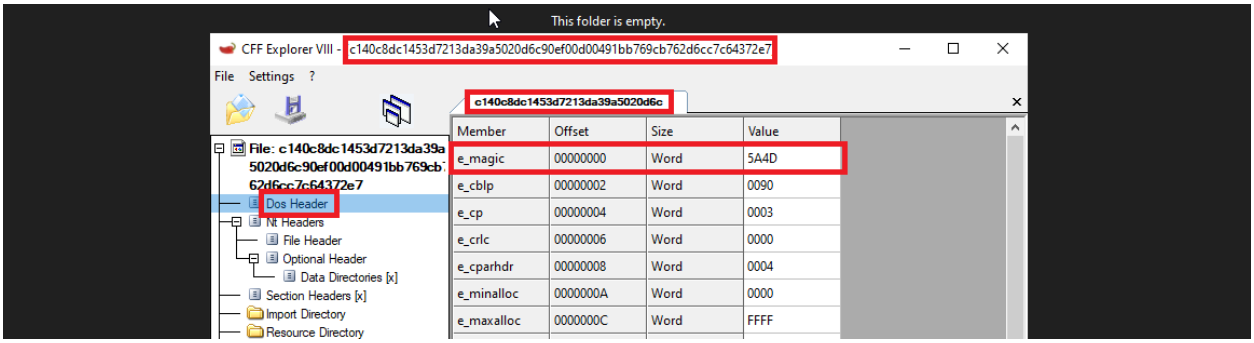

Figure 51: Malware-3 Magic Byte through CFF Explorer

Task-1.2 File Signature

A file signature is used to identify the content of the file. This information is stored in the file's header. To find the file signature of the malware I have used two different tools PE Studio and CFF Explorer.

Tool-1 PE Studio

To find the file signature of the malware file I open the file in the PE Studio, then from the left panel I click on "**file-header**" and then from the right panel I found the column named "**signature**". From there I found that the signature of the file is "**PE**" which means that this file is portable executable.

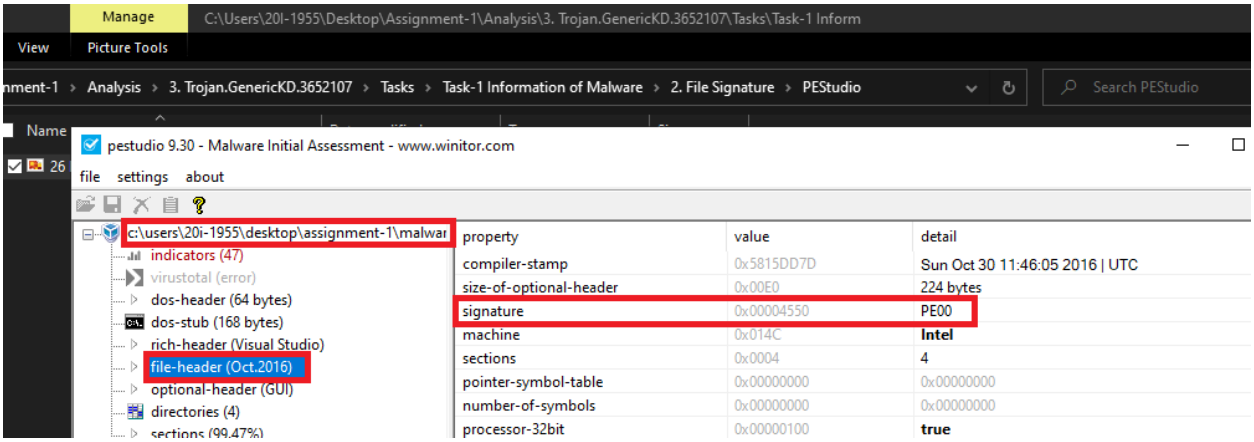

Figure 52: Malware-3 File Signature through PE Studio

Tool-2 CFF Explorer

To find the file signature of the malware file I open the file in the CFF Explorer, then from the left panel I click on "**Nt Headers**" and then from the right panel I found the column named "**signature**". From there I found that the signature of the file is "**45 50**" which is equivalent to "**50 45**" because in CFF Explorer hexadecimal are represented in right to left order. So "**50 = P**" and "**45 = E**" which means that this file is portable executable.

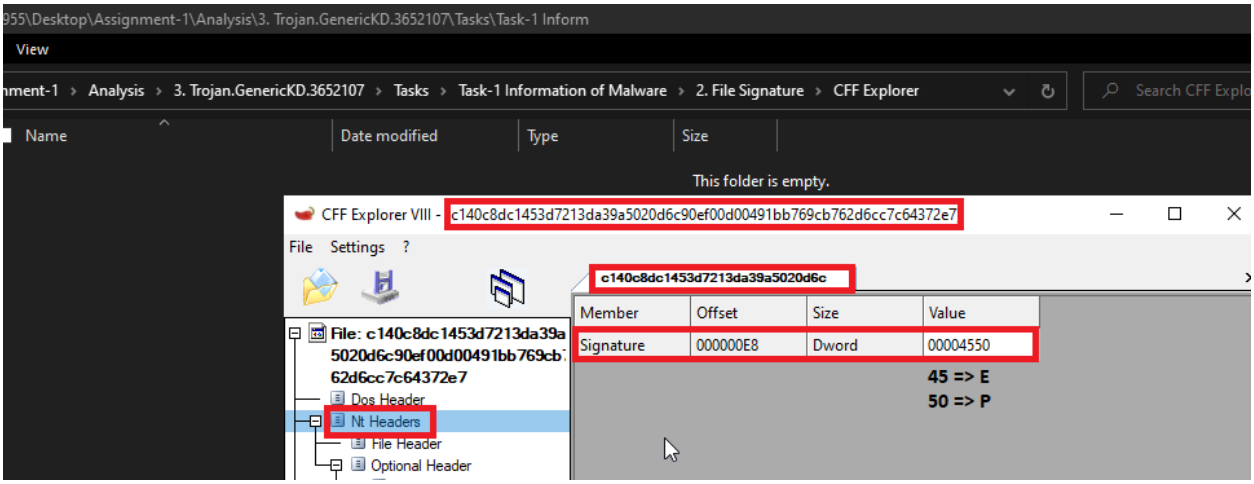

Figure 53: Malware-3 File Signature through CFF Explorer

Task-1.3 Machine Information

To find the machine information of the malware I have used two different tools PE Studio and CFF Explorer.

Tool-1 PE Studio

To find the file signature of the malware file I open the file in the PE Studio, then from the left panel I click on "**file-header**" and then from the right panel I found the column named "**machine**". From there I found that this malware is designed for Intel-based machines. Other than machine information I also find that when the malware is created from the "**compiler-stamp**" column. This malware is created on Oct 30, 2016, on Sunday at 11:46 AM. Also find that this malware file has 4 sections and the size of the optional header is 224 bytes.

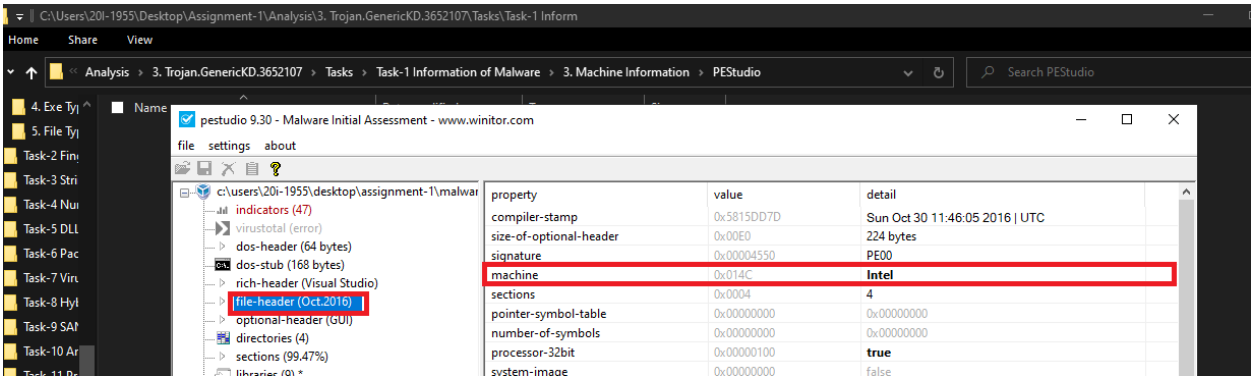

Figure 54: Malware-3 Machine Information through PE Studio

Tool-2 CFF Explorer

To find the file signature of the malware file I open the file in the CFF Explorer, then from the left panel I click on "**file header**" and then from the right panel I found the column named "**Machine**". From there I found that this malware is designed for Intel 386 based machines.

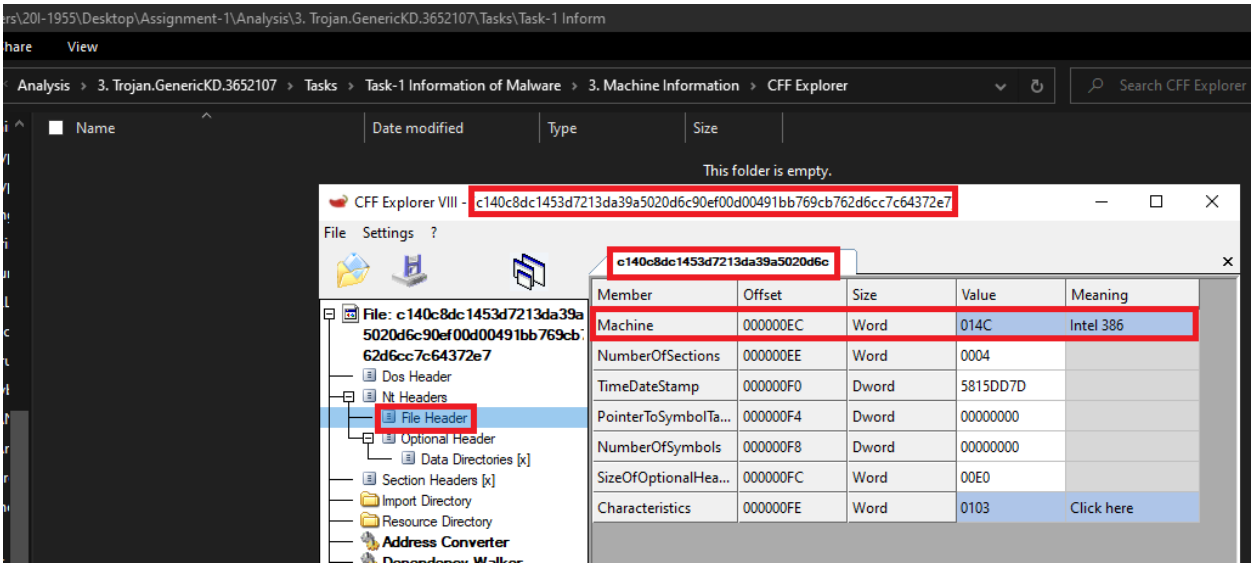

Figure 55: Malware-3 Machine Information through CFF Explorer

Task-1.4 Exe Type

Exe type is used to identify which type of executable it is either 32-bit or 64-bit and to find that I used two different tools PE Studio and CFF Explorer.

Tool-1 PE Studio

To find the exe type of the malware file I open the file in the PE Studio, then from the left panel I click on "**optional-header**" and then from the right panel I found the column named "**magic**". From there I found that the exe type of this malware file is "0x010B" which means that this malware file is a 32-bit portable executable. Other than exe type I also find the entry point of the malware which is ".text".

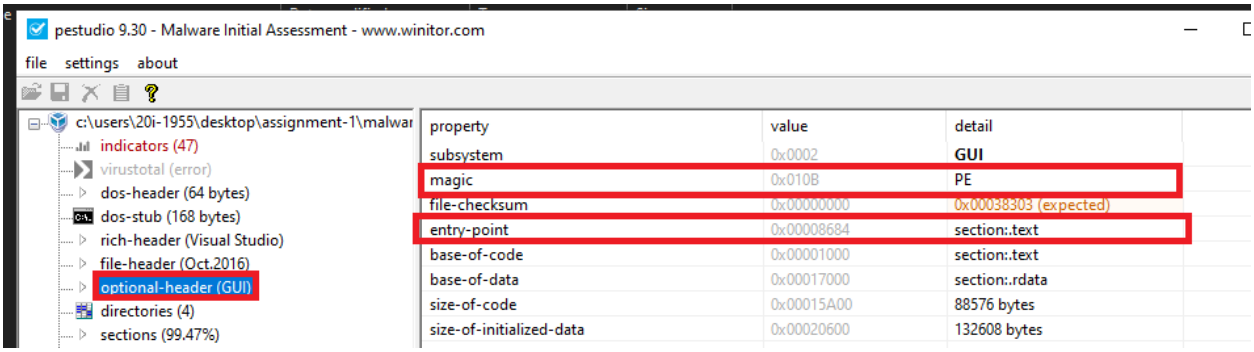

Figure 56: Malware-3 Exe type through PE Studio

Tool-2 CFF Explorer

To find the file signature of the malware file I open the file in the CFF Explorer, then from the left panel I click on "**optional header**" and then from the right panel I found the column named "**Magic**". From there I found that the exe type of this malware file is "010B" which means that this malware file is a 32-bit portable executable. Other than exe type I also find the entry point of the malware which is ".text".

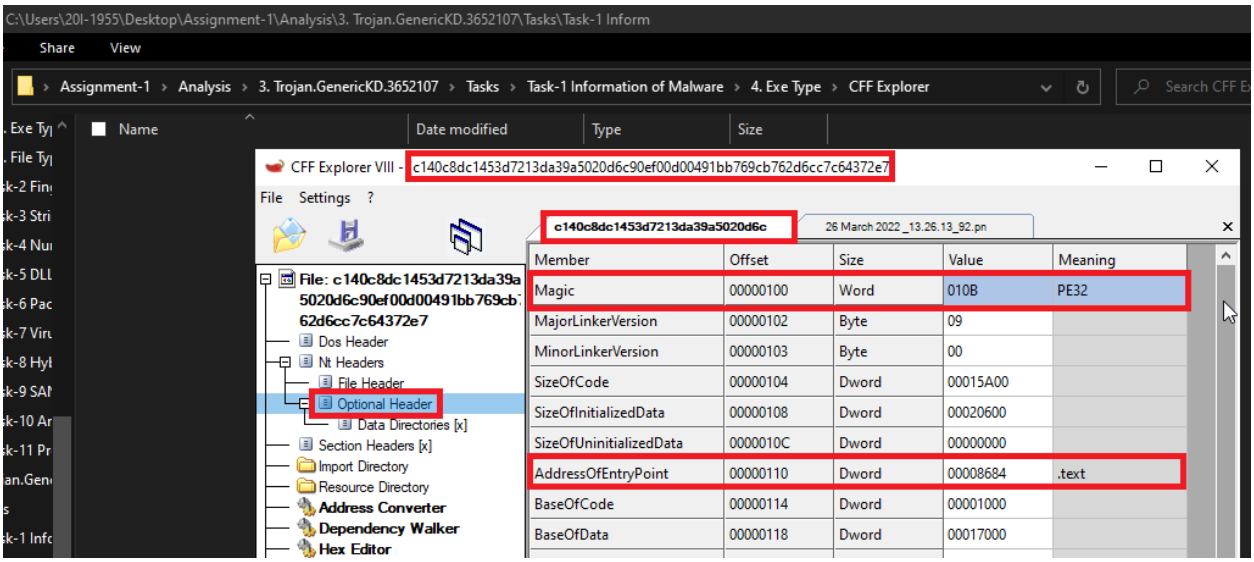

Figure 57: Malware-3 Exe type through CFF Explorer

Task-1.5 File Type

File Type is used to identify which type of file it is. I used two different tools Python Magic Library and TrID Net.

Tool-1 Python Magic Library

To find the file type of the malware using the python magic library I write the following code.

```
import magic

mgicByte = magic.from_file('C:\\Users\\20I-1955\\Desktop\\Assignment-1\\Malwares\\3. Trojan.GenericKD.3652107\\c140c8dc1453d7213da39a5020d6c90ef00d00491bb769cb762d6cc7c64372e7\\c140c8dc1453d7213da39a5020d6c90ef00d00491bb769cb762d6cc7c64372e7')

print(mgicByte)
```

After running the code, I found that the file type of malware is “PE32 executable (GUI) Intel 80386, for MS Windows”.

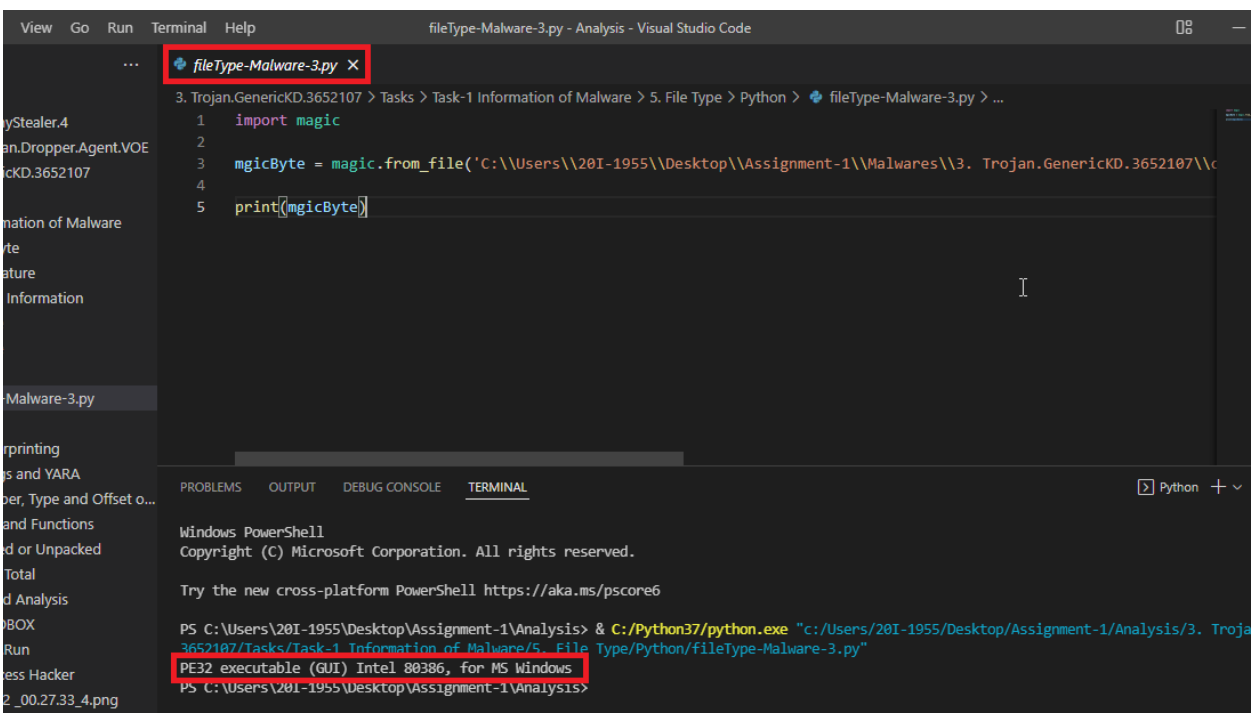

Figure 58: Malware-3 File Type through Python Magic Library

Tool-2 TrID Net

To find the file type of the malware using TrID Net I open the file in TrID Net and find that the malware file is “Win32 Executable MS Visual C++” as it has the “48.9%” match.

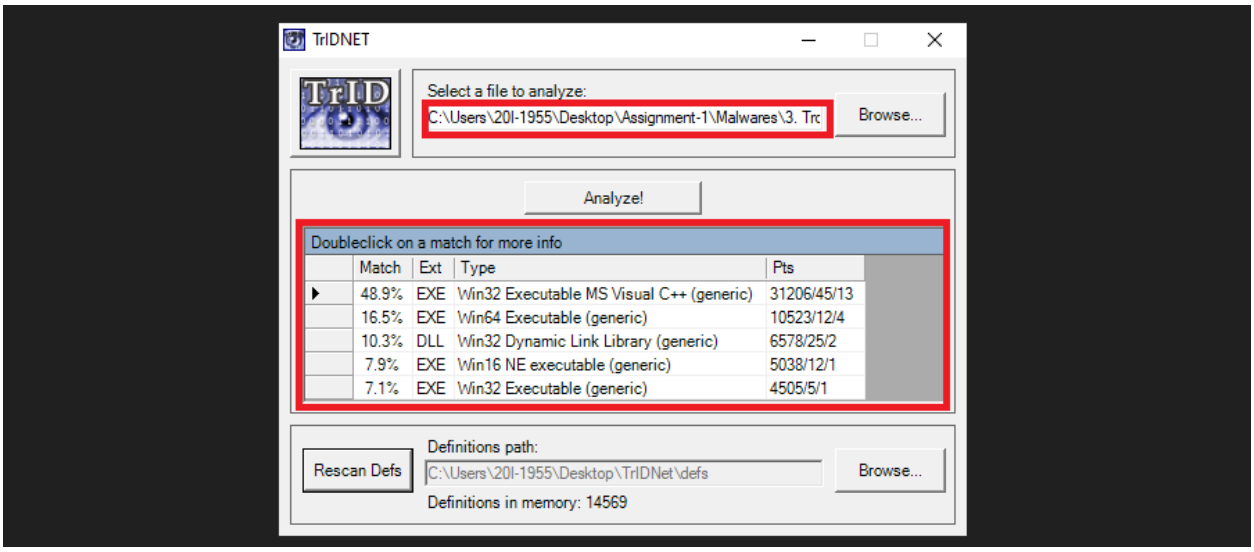

Figure 59: Malware-3 File Type through TrID Net

Task-2 What is the fingerprinting information of the malware file?

To find the hashes of the malware file I have used two different tools HashCalc and Python hashlib library.

Tool-1 HashCalc

To find the hash function of the file in HashCalc I open the file in HashCalc and find the hash of the file.

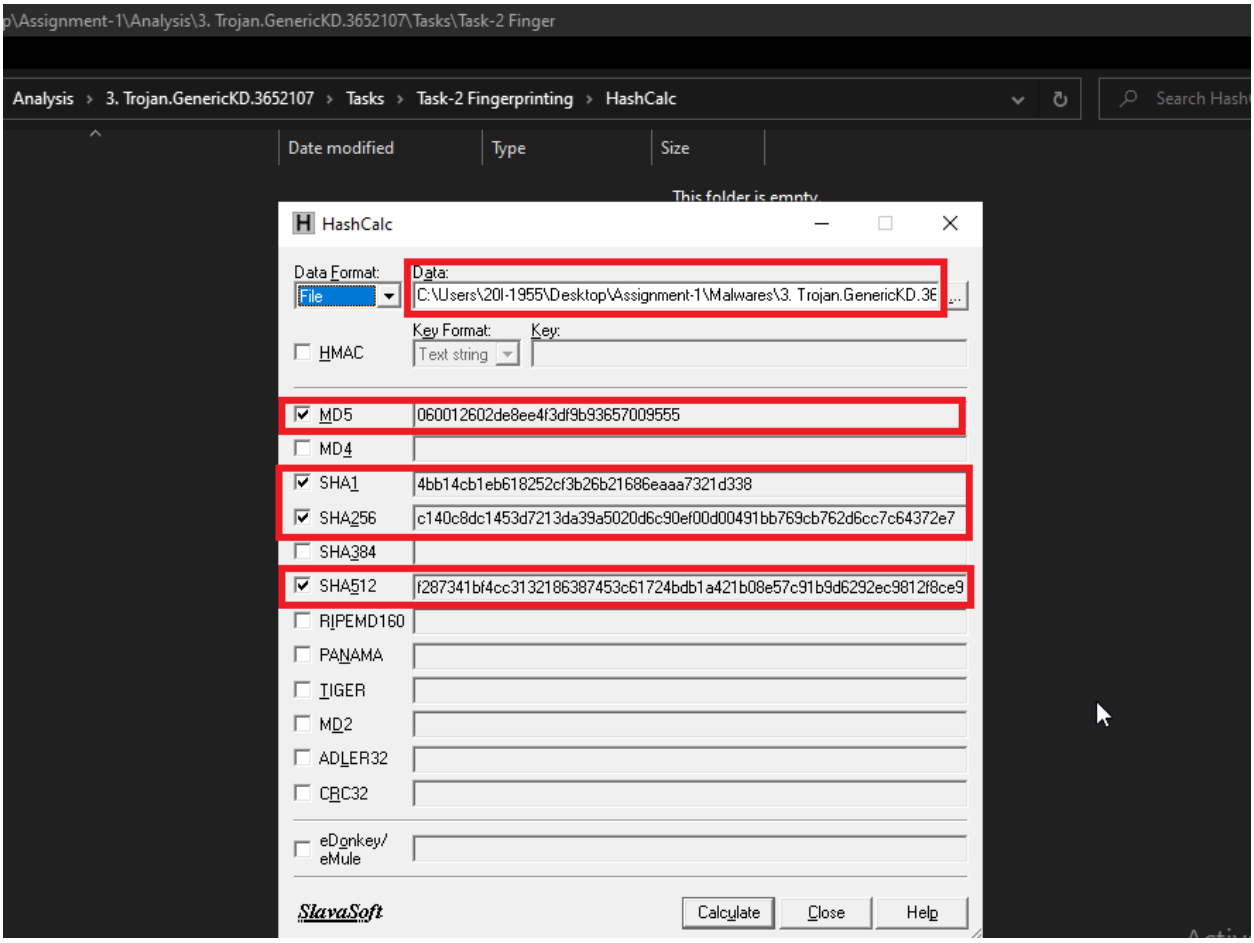

Figure 60: Malware-3 Fingerprinting through HashCalc

Tool-2 Python Hashlib Library

To find the hash of the malware using the python Hashlib library I write the following code.

```
import hashlib

hashMd5 = hashlib.md5(open("C:\\Users\\20I-1955\\Desktop\\Assignment-
1\\Malwares\\3.
Trojan.GenericKD.3652107\\c140c8dc1453d7213da39a5020d6c90ef00d00491bb769cb762d6
cc7c64372e7\\c140c8dc1453d7213da39a5020d6c90ef00d00491bb769cb762d6cc7c64372e7",
"rb").read()).hexdigest()
hashSha1 = hashlib.sha1(open("C:\\Users\\20I-1955\\Desktop\\Assignment-
1\\Malwares\\3.
Trojan.GenericKD.3652107\\c140c8dc1453d7213da39a5020d6c90ef00d00491bb769cb762d6
cc7c64372e7\\c140c8dc1453d7213da39a5020d6c90ef00d00491bb769cb762d6cc7c64372e7",
"rb").read()).hexdigest()
hashSha256 = hashlib.sha256(open("C:\\Users\\20I-1955\\Desktop\\Assignment-
1\\Malwares\\3.
Trojan.GenericKD.3652107\\c140c8dc1453d7213da39a5020d6c90ef00d00491bb769cb762d6
cc7c64372e7\\c140c8dc1453d7213da39a5020d6c90ef00d00491bb769cb762d6cc7c64372e7",
"rb").read()).hexdigest()
hashSha512 = hashlib.sha512(open("C:\\Users\\20I-1955\\Desktop\\Assignment-
1\\Malwares\\3.
Trojan.GenericKD.3652107\\c140c8dc1453d7213da39a5020d6c90ef00d00491bb769cb762d6
cc7c64372e7\\c140c8dc1453d7213da39a5020d6c90ef00d00491bb769cb762d6cc7c64372e7",
"rb").read()).hexdigest()

print("MD5:    ",hashMd5)
print("SHA1:   ",hashSha1)
print("SHA256: ",hashSha256)
print("SHA512: ",hashSha512)
hashSha1 = hashlib.sha1(open("C:\\Users\\20I-1955\Desktop\\Assignment-
1\Malwares\\2.
DroppedTrojan.Dropper.Agent.VOE\\91bfa2445d998425c81f30d293235429ca6a8c6c8f3265
36478952a2a6754aac\\91bfa2445d998425c81f30d293235429ca6a8c6c8f326536478952a2a67
54aac", "rb").read()).hexdigest()
hashSha256 = hashlib.sha256(open("C:\\Users\\20I-1955\Desktop\\Assignment-
1\Malwares\\2.
DroppedTrojan.Dropper.Agent.VOE\\91bfa2445d998425c81f30d293235429ca6a8c6c8f3265
36478952a2a6754aac\\91bfa2445d998425c81f30d293235429ca6a8c6c8f326536478952a2a67
54aac", "rb").read()).hexdigest()
hashSha512 = hashlib.sha512(open("C:\\Users\\20I-1955\Desktop\\Assignment-
1\Malwares\\2.
DroppedTrojan.Dropper.Agent.VOE\\91bfa2445d998425c81f30d293235429ca6a8c6c8f3265
36478952a2a6754aac\\91bfa2445d998425c81f30d293235429ca6a8c6c8f326536478952a2a67
54aac", "rb").read()).hexdigest()
```

```
print("MD5:    ",hashMd5)
print("SHA1:   ",hashSha1)
print("SHA256: ",hashSha256)
print("SHA512: ",hashSha512)
```

After running the code, I found that the hashes of the malware file

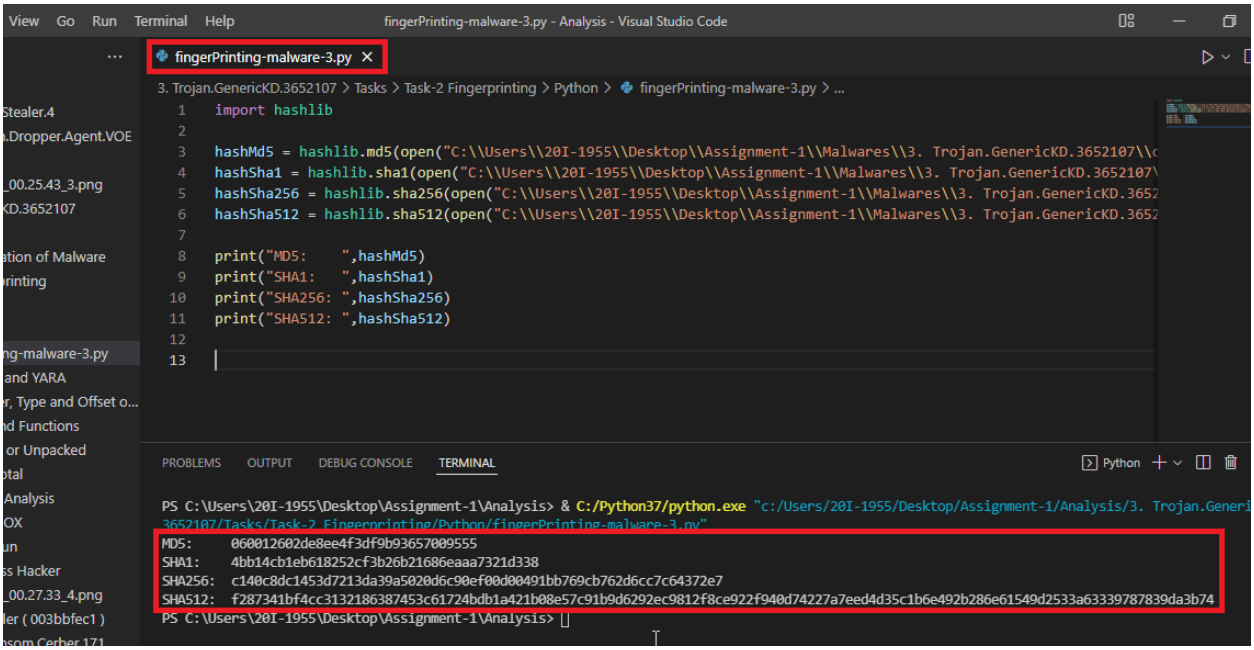

Figure 61: Malware-3 Fingerprinting through Python Hashlib Library

Task-3 What type of malicious strings are in the Malware file?

To extract the strings from the malware file I use the tool called "**Strings**" and then to scan the malware according to the extracted strings I write the YARA rule.

Tool-1 Strings

To extract the strings from the malware file and save them in the text file I run the following command

```
strings malware file path > text file name
```

After running the command all the strings are extracted from the malware file and stored in the text file.

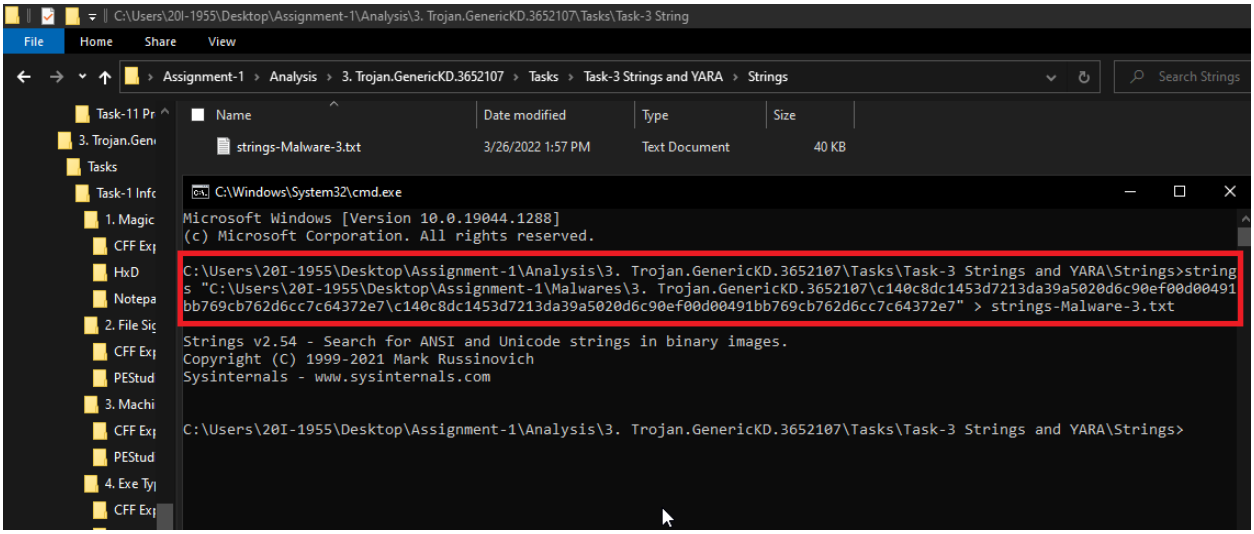

Figure 62: Malware-3 Strings Extraction through Strings Tool

After that, I open the text file to check the strings that were extracted from the malware and found some malicious strings.

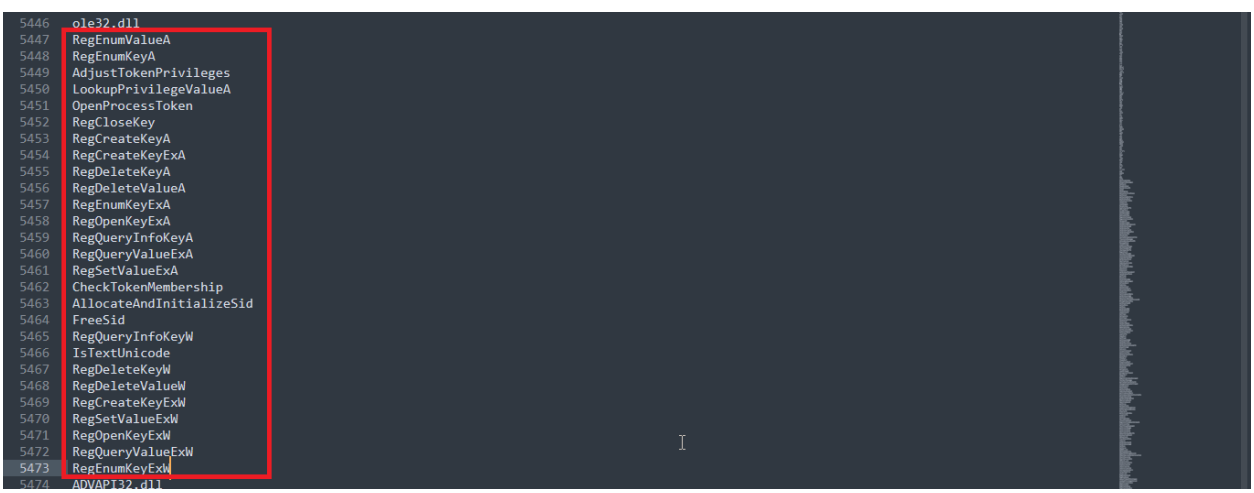

Figure 63: Malware-3 Strings Extracted through Strings Tool

Tool-2 YARA Rule

After extracting the string my next task is to write the YARA rule that scans the file and triggers the warning if it finds the malicious string in the malware file. For that, I write the following rule

```
rule malware_Trojan_GenericKD_3652107{
    meta:
        malwareName = "Trojan.GenericKD.3652107"
        ruleAuthor = "20I-1955_Muhammad Osama Khalid"
    strings:
        $str1 = "RegOpenKey" nocase
        $str2 = "RegQueryValue" nocase
        $str3 = "RegCloseKey" nocase
        $str4 = "kernel32" nocase
        $str5 = "shell32" nocase
    condition:
        any of them
}
```

Note: These strings are extracted from analyzing all the malware samples that are given for this assignment.

After that, I run the rule and find that the rule that I write works successfully as it detects the malicious string extracted above in the malware file.

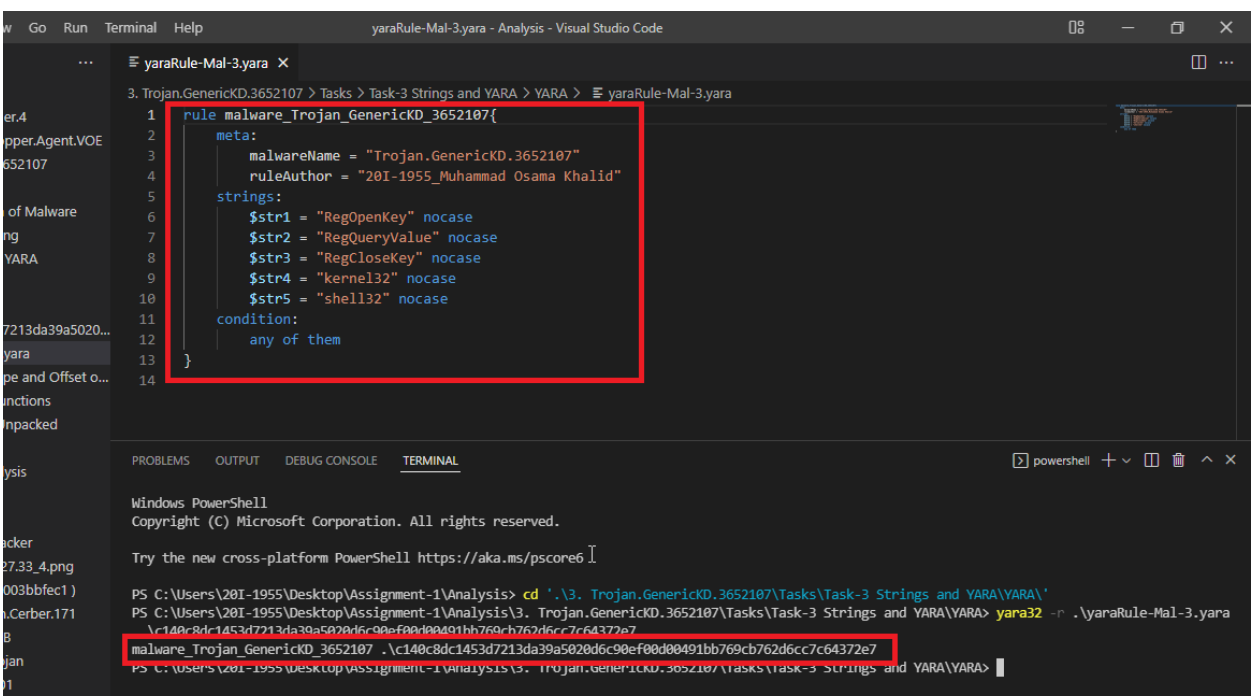

Figure 64: Malware-3 Malware detection through YARA Rule

Task-4 Number, type, and offset of different sections of Malware?

Sections contain the important information about the file as it describes how the executable is made. To find the number, type, and offset of different sections of malware files I use three different tools PE Studio and Exeinfo.

Tool-1 PE Studio

To find the number, type, and offset of different sections of the malware file I open the file in the PE Studio, then from the left panel first I click on "**file-header**" and then from the right panel I found the column named "**sections**". From there I found that this malware file has three sections and to find then what they are I click on "**sections**" in the left panel. After that from the right panel, I found the sections that this malware file has.

No. of Section	Type	Offset
1	.text	0x00401000
2	.rdata	0x00417000
3	.data	0x0042E000
4	.rsrc	0x00437000

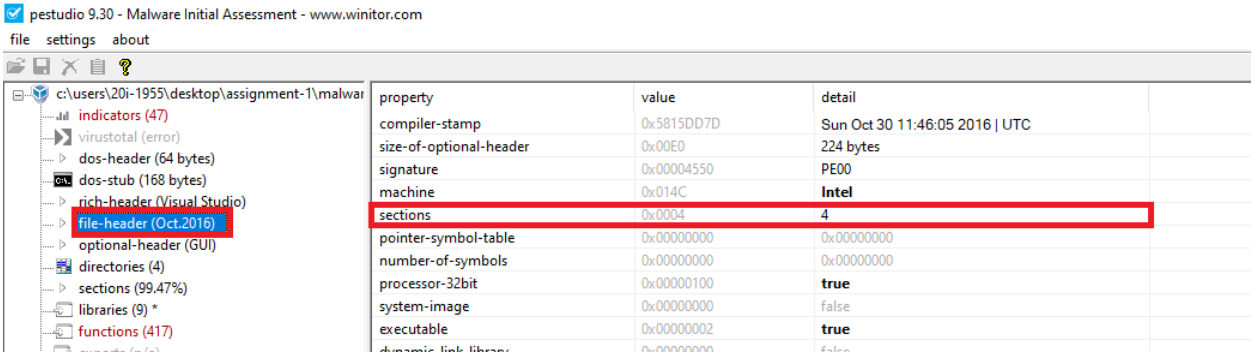

Figure 65: Malware-3 Information of Sections through PE Studio

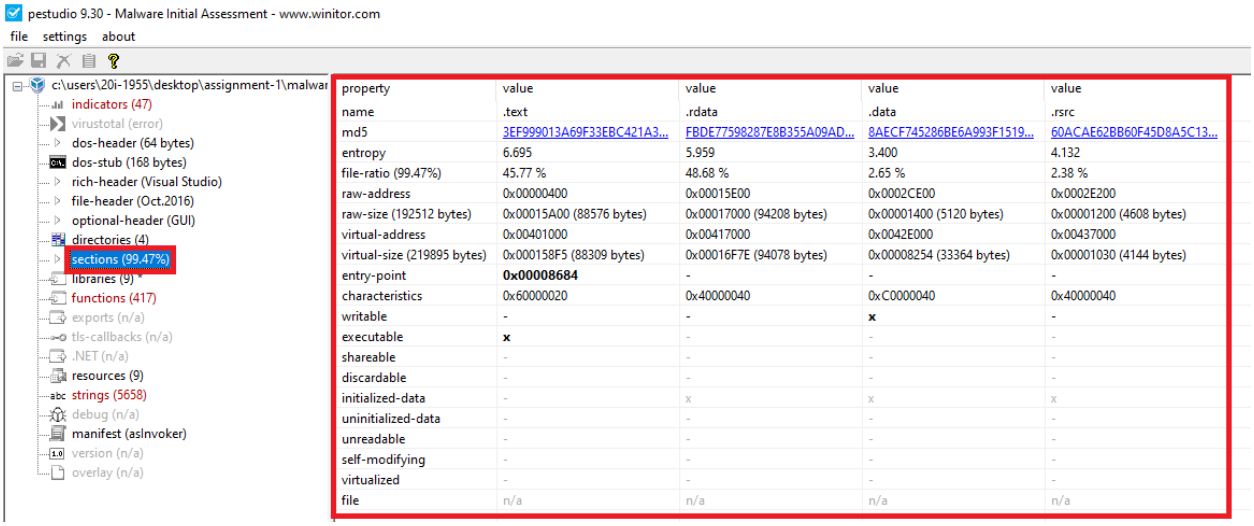

Figure 66: Malware-3 Information of Sections through PE Studio

Task-5 Which DLLs and their number of functions are used by Malware?

To find what DLLs and functions this malware used I used the tool PE Studio.

Tool-1 PE Studio

To find the DLLs and their number of functions used by malware files I open the file in the PE Studio, then from the left panel first I click on "**libraries**" and then from the right pane I found the list of DLLs and their functions used by this malware.

No. of DLL	Name	No. of functions
1	user32.dll	147
2	shell32.dll	5
3	ole32.dll	6
4	advapi32.dll	27
5	gdi32.dll	23
6	comctl32.dll	1
7	version.dll	3
8	shlwapi.dll	12
9	kernel32.dll	192

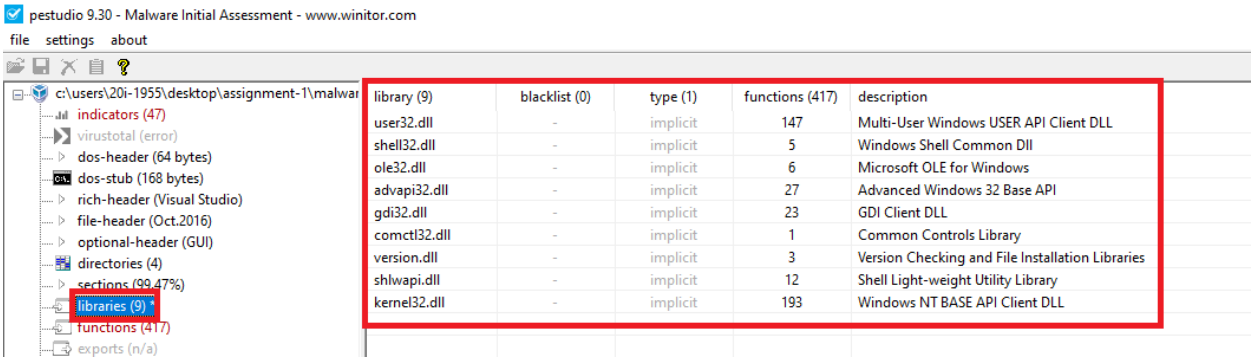

Figure 67: Malware-3 Information of DLLs and functions through PE Studio

Task-6 What is packing information if Malware is packed?

To check whether the malware is packed or not I use the tool Exeinfo.

Tool-1 Exeinfo

To find whether the malware file is packed or not I open the file in Exeinfo and at the bottom I found that the malware is not packed.

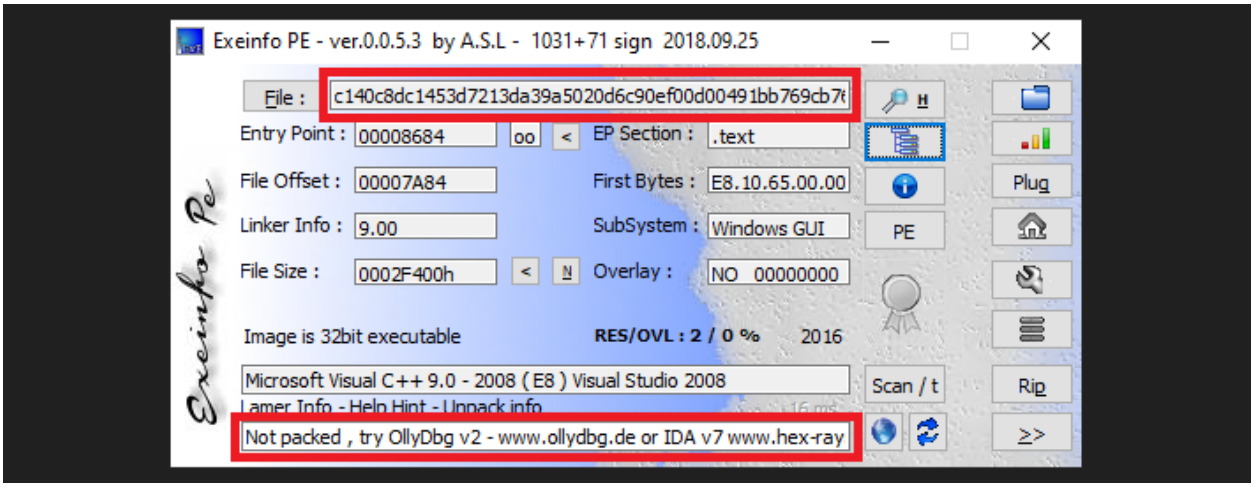

Figure 68: Malware-3 Packed or unpacked malware information through Exeinfo

Task-7 VirusTotal analysis of the malware file

To scan the malware file from the virus total I visit the website and upload the malware file. After some time, I found that the file is triggered as malicious by 61 antivirus search engines.

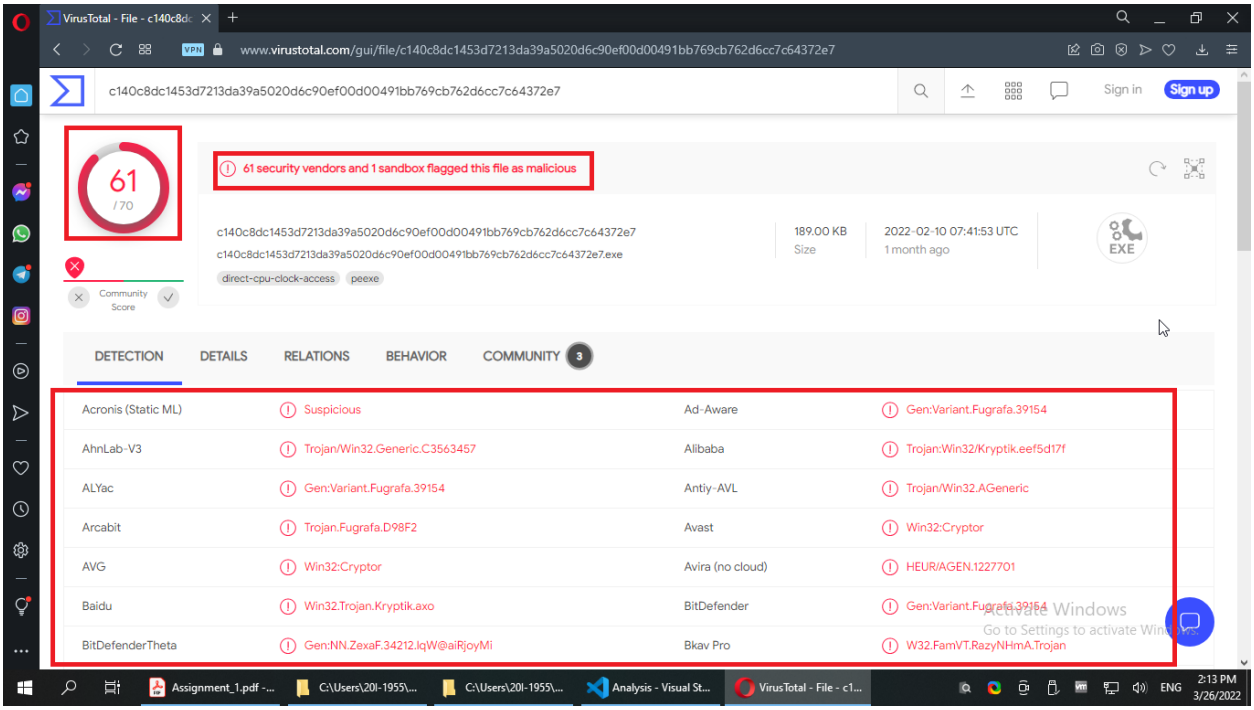

Figure 69: Malware-3 VirusTotal Analysis

Moreover, in the details tab, I find more information about the malware that I already explained in the previous tasks.

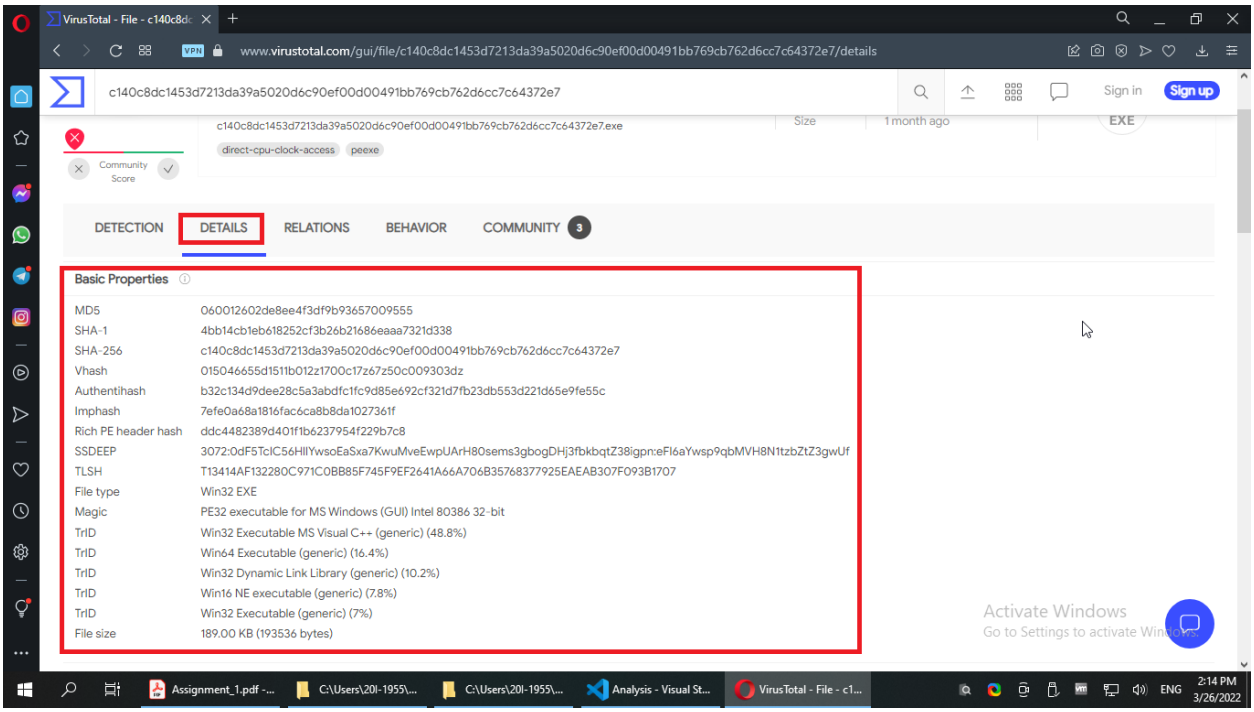

Figure 70: Malware-3 VirusTotal Analysis

Task-8 Analysis of the malware file through Hybrid Analysis

To scan the malware file from Hybrid Analysis I visit the website and upload the malware file. After some time, I found that the file is triggered as malicious by antivirus search engines.

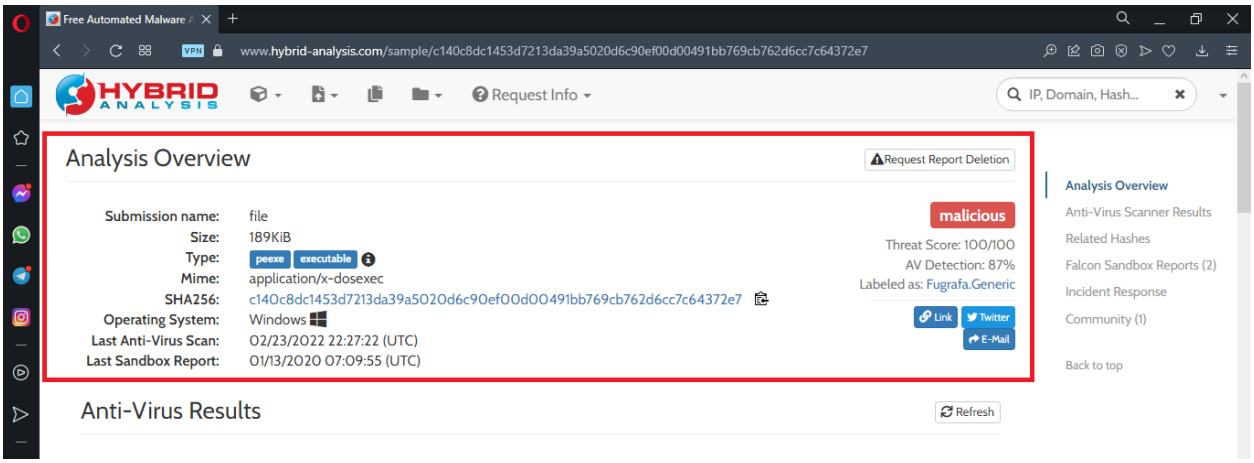

Figure 71: Malware-3 Analysis through Hybrid Analysis

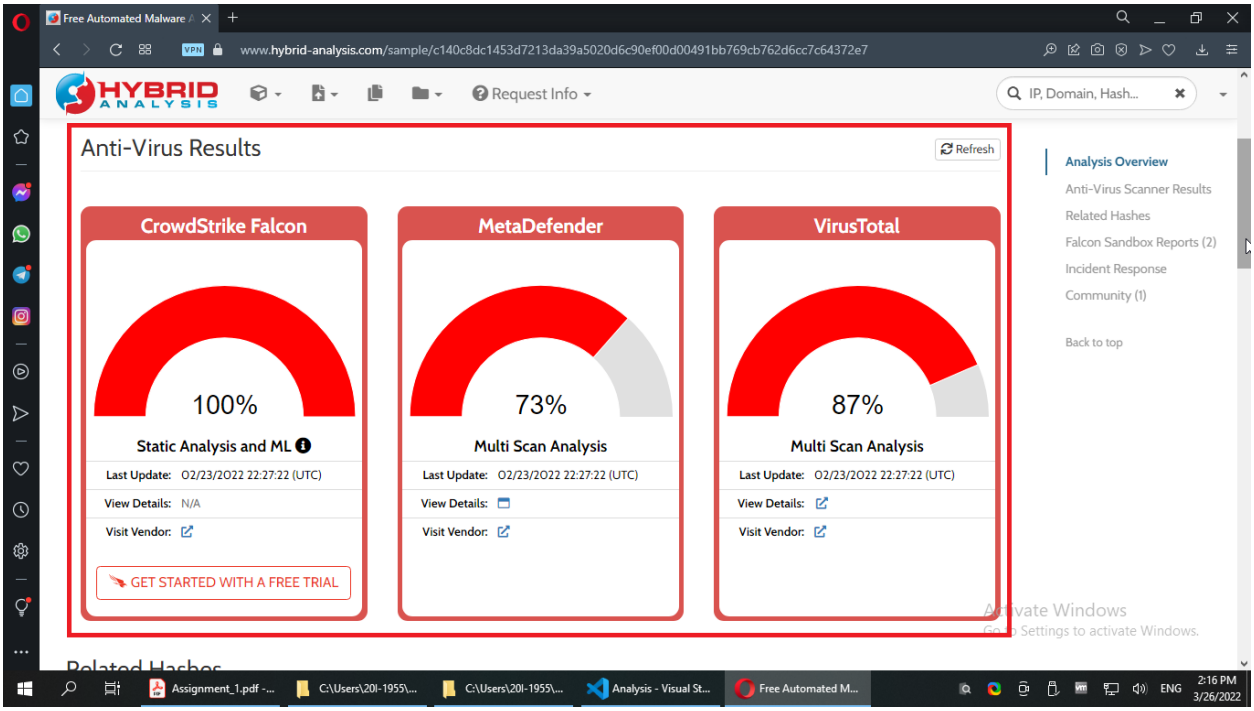

Figure 72: Malware-3 Analysis through Hybrid Analysis

Malware-4 Password-Stealer (003bbfec1)

This is a type of malware that steals the account information of the victim machine.

Task-1 What is complete information malware have

In this task, I have to find the basic information of the malware that includes identifying magic byte, signatures, machine info, exe type, and file type.

Task-1.1 Magic Byte

To find the magic byte of the malware there are a bunch of tools available and for this task, I have used three different tools Notepad++, HxD, and CFF Explorer.

Tool-1 Notepad++

To find the magic byte of the malware file I open the file in Notepad++, then from the plugin tab click on "HEX-Editor" and then click on "View in HEX". On viewing the file in HEX-Editor I found that the magic byte of the malware file is "**MZ**". This means that the malware file is executable. Moreover, I also find that this file can't be able to run in DOS mode.

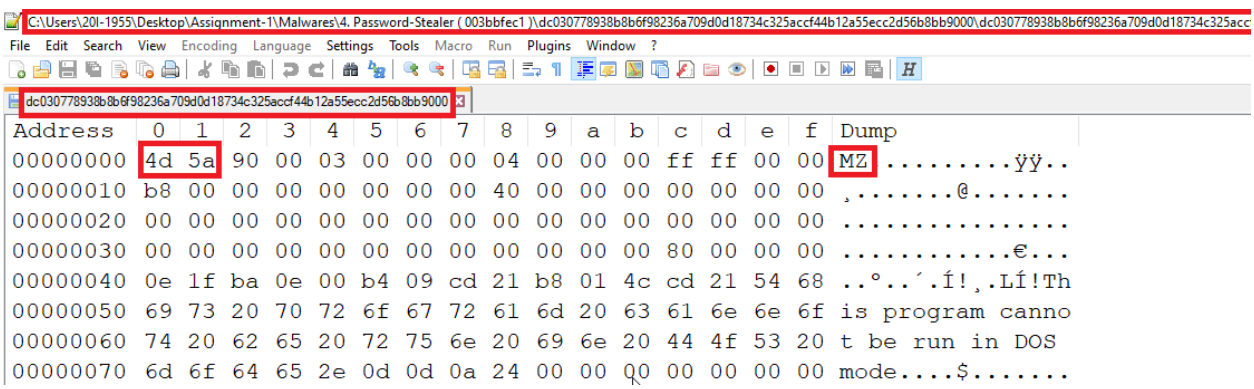

Figure 73: Malware-4 Magic Byte through Notepad++

Tool-2 HxD

To find the magic byte of the malware file I open the file in the HxD and find that the magic byte of the file is "**MZ**" which means that the malware file is executable. Moreover, I also find that this file can't be able to run in DOS mode.

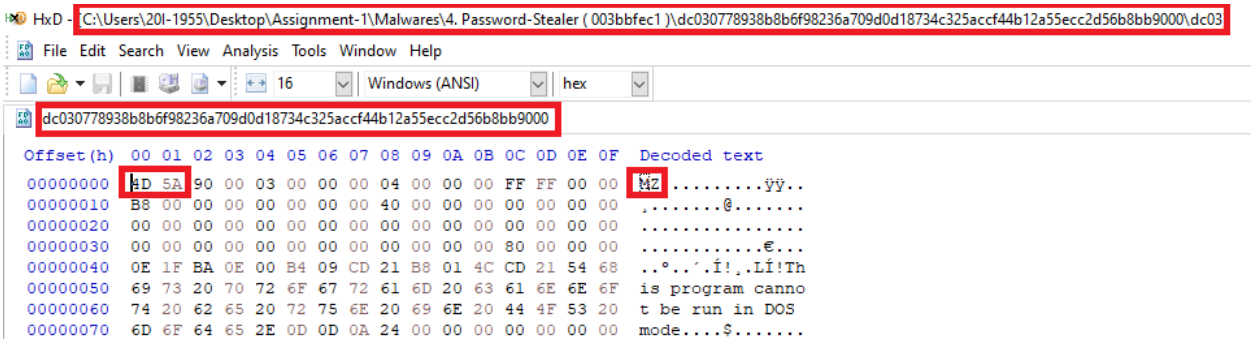

Figure 74: Malware-4 Magic Byte through HxD

Tool-3 CFF Explorer

To find the magic byte of the malware file I open the file in the CFF Explorer, then from the left panel I click on "**DOS Header**". In the "**DOS Header**" I find the magic byte in the "e_magic" column that is "**5A 4D**" which is equivalent to "**4D 5A**" because in CFF Explorer hexadecimal are represented in right to left order. So "**4D = M**" and "**5A = Z**" mean that the malware file is executable.

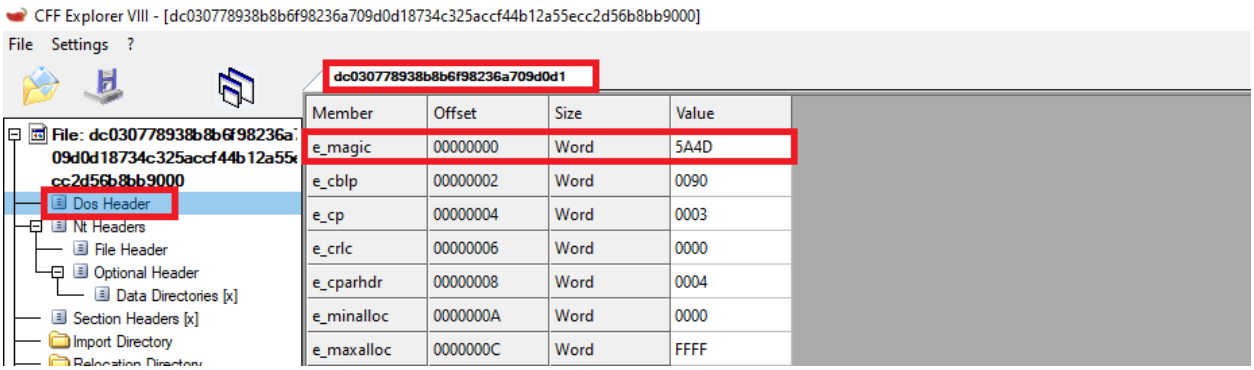

Figure 75: Malware-4 Magic Byte through CFF Explorer

Task-1.2 File Signature

A file signature is used to identify the content of the file. This information is stored in the file's header. To find the file signature of the malware I have used two different tools PE Studio and CFF Explorer.

Tool-1 PE Studio

To find the file signature of the malware file I open the file in the PE Studio, then from the left panel I click on "**file-header**" and then from the right panel I found the column named "**signature**". From there I found that the signature of the file is "**PE**" which means that this file is portable executable.

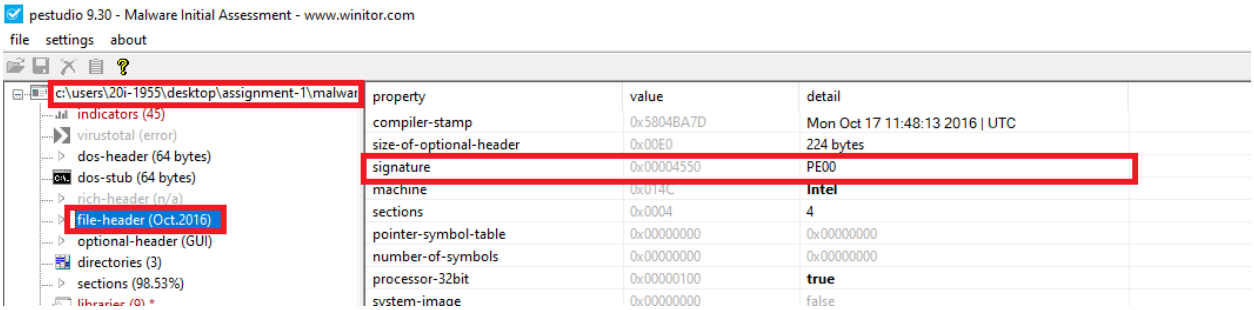

Figure 76: Malware-4 File Signature through PE Studio

Tool-2 CFF Explorer

To find the file signature of the malware file I open the file in the CFF Explorer, then from the left panel I click on "**Nt Headers**" and then from the right panel I found the column named "**signature**". From there I found that the signature of the file is "**45 50**" which is equivalent to "**50 45**" because in CFF Explorer hexadecimal are represented in right to left order. So "**50 = P**" and "**45 = E**" which means that this file is portable executable.

Figure 77: Malware-4 File Signature through CFF Explorer

Task-1.3 Machine Information

To find the machine information of the malware I have used two different tools PE Studio and CFF Explorer.

Tool-1 PE Studio

To find the file signature of the malware file I open the file in the PE Studio, then from the left panel I click on "**file-header**" and then from the right panel I found the column named "**machine**". From there I found that this malware is designed for Intel-based machines. Other than machine information I also find that when the malware is created from the "**compiler-stamp**" column. This malware is created on Oct 17, 2016, on Sunday at 11:48 AM. Also find that this malware file has 4 sections and the size of the optional header is 224 bytes.

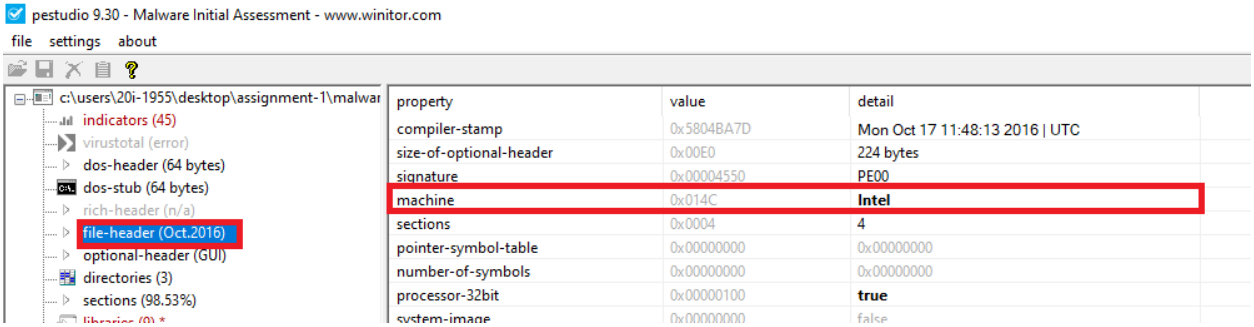

Figure 78: Malware-4 Machine Information through PE Studio

Tool-2 CFF Explorer

To find the file signature of the malware file I open the file in the CFF Explorer, then from the left panel I click on "**file header**" and then from the right panel I found the column named "**Machine**". From there I found that this malware is designed for Intel 386 based machines.

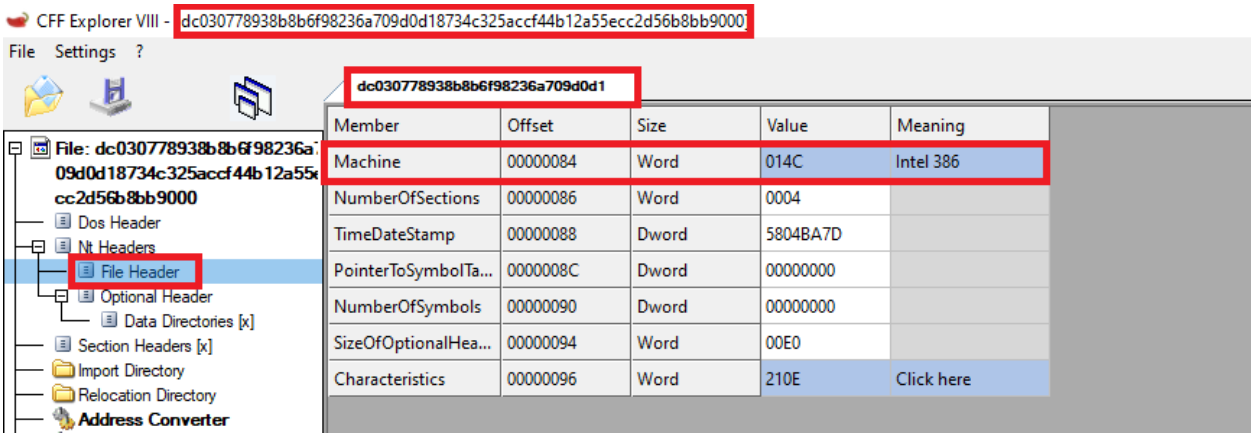

Figure 79: Malware-4 Machine Information through CFF Explorer

Task-1.4 Exe Type

Exe type is used to identify which type of executable it is either 32-bit or 64-bit and to find that I used two different tools PE Studio and CFF Explorer.

Tool-1 PE Studio

To find the exe type of the malware file I open the file in the PE Studio, then from the left panel I click on "**optional-header**" and then from the right panel I found the column named "**magic**". From there I found that the exe type of this malware file is "0x010B" which means that this malware file is a 32-bit portable executable. Other than exe type I also find the entry point of the malware which is ".text".

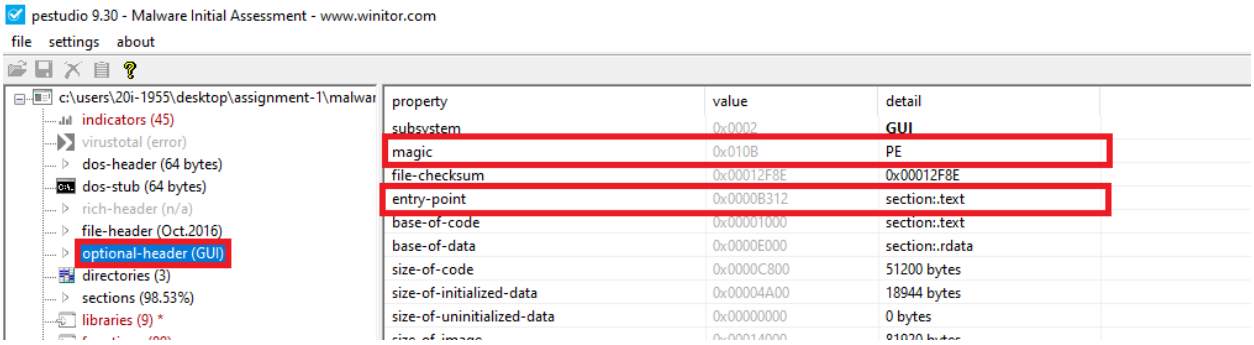

Figure 80: Malware-4 Exe type through PE Studio

Tool-2 CFF Explorer

To find the file signature of the malware file I open the file in the CFF Explorer, then from the left panel I click on "**optional header**" and then from the right panel I found the column named "**Magic**". From there I found that the exe type of this malware file is "010B" which means that this malware file is a 32-bit portable executable. Other than exe type I also find the entry point of the malware which is ".text".

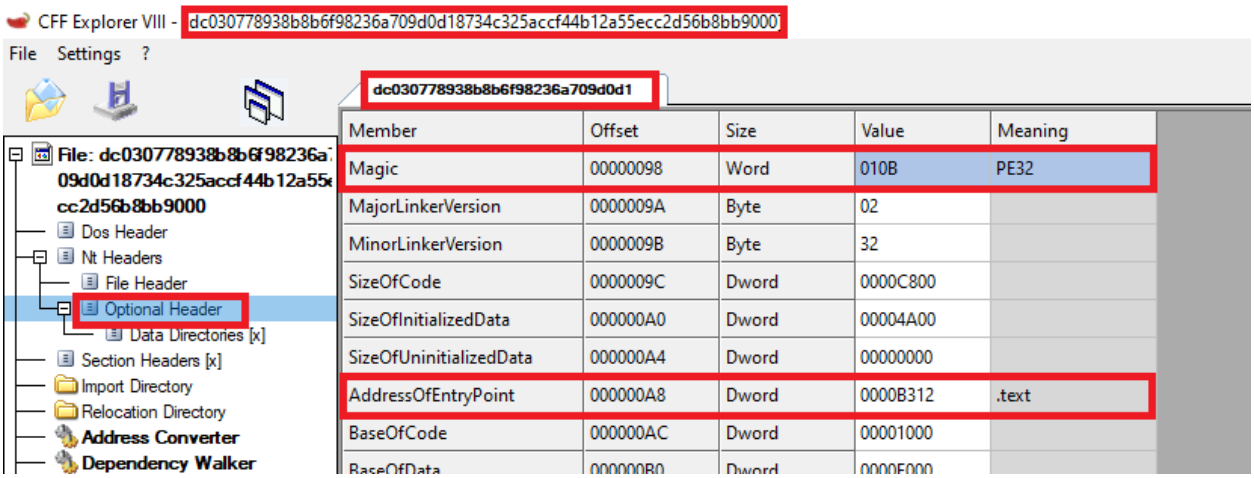

Figure 81: Malware-4 Exe type through CFF Explorer

Task-1.5 File Type

File Type is used to identify which type of file it is. I used two different tools Python Magic Library and TrID Net.

Tool-1 Python Magic Library

To find the file type of the malware using the python magic library I write the following code.

```
import magic

mgicByte = magic.from_file('C:\\Users\\20I-1955\\Desktop\\Assignment-
1\\Malwares\\4. Password-Stealer ( 003bbfec1
)\\dc030778938b8b6f98236a709d0d18734c325accf44b12a55ecc2d56b8bb9000\\dc03077893
8b8b6f98236a709d0d18734c325accf44b12a55ecc2d56b8bb9000')

print(mgicByte)
```

After running the code, I found that the file type of malware is "PE32 executable (GUI) Intel 80386, for MS Windows".

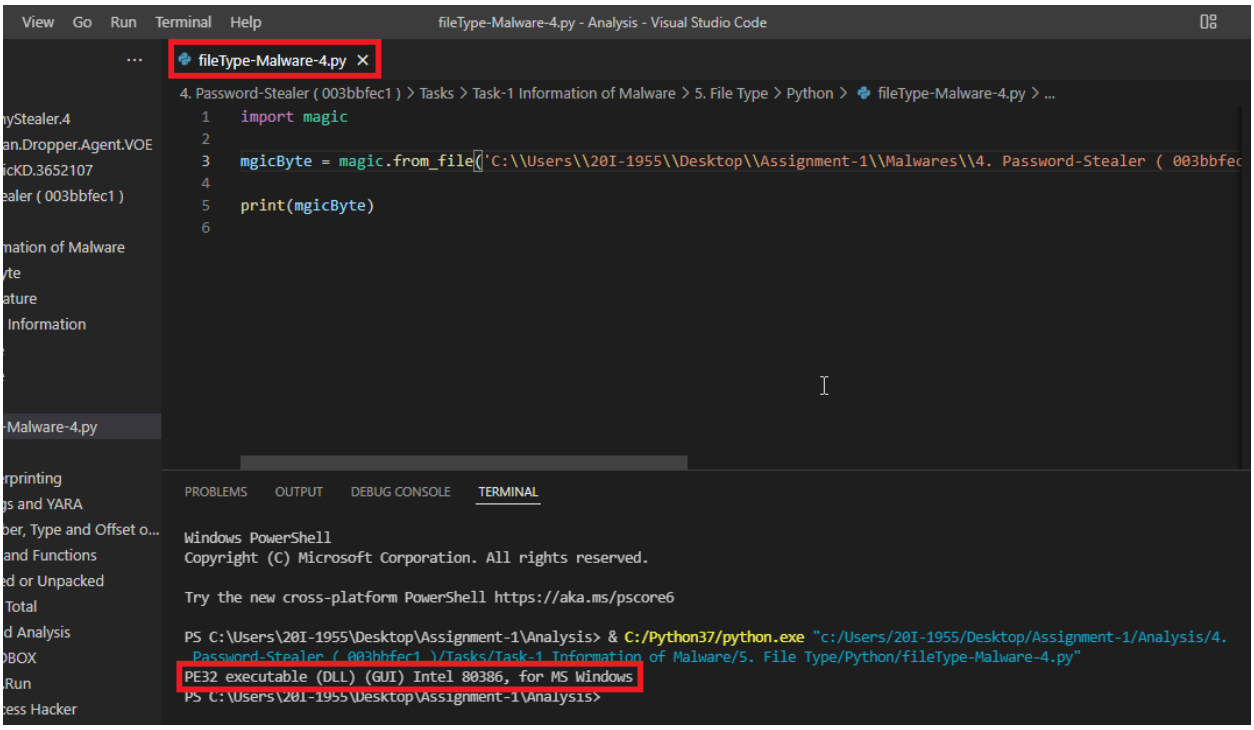

Figure 82: Malware-4 File Type through Python Magic Library

Tool-2 TrID Net

To find the file type of the malware using TrID Net I open the file in TrID Net and find that the malware file is “Win32 Executable MS Visual C++” as it has the “66.3%” match.

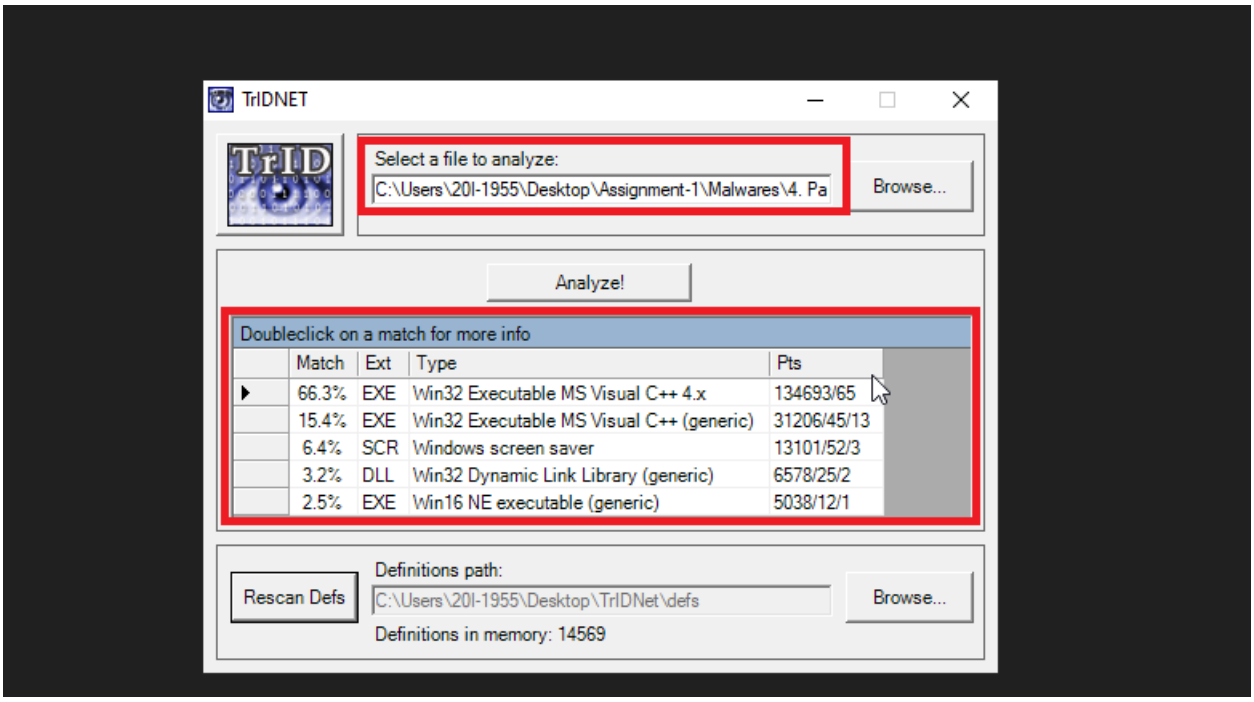

Figure 83: Malware-4 File Type through TrID Net

Task-2 What is the fingerprinting information of the malware file?

To find the hashes of the malware file I have used two different tools HashCalc and Python hashlib library.

Tool-1 HashCalc

To find the hash function of the file in HashCalc I open the file in HashCalc and find the hash of the file.

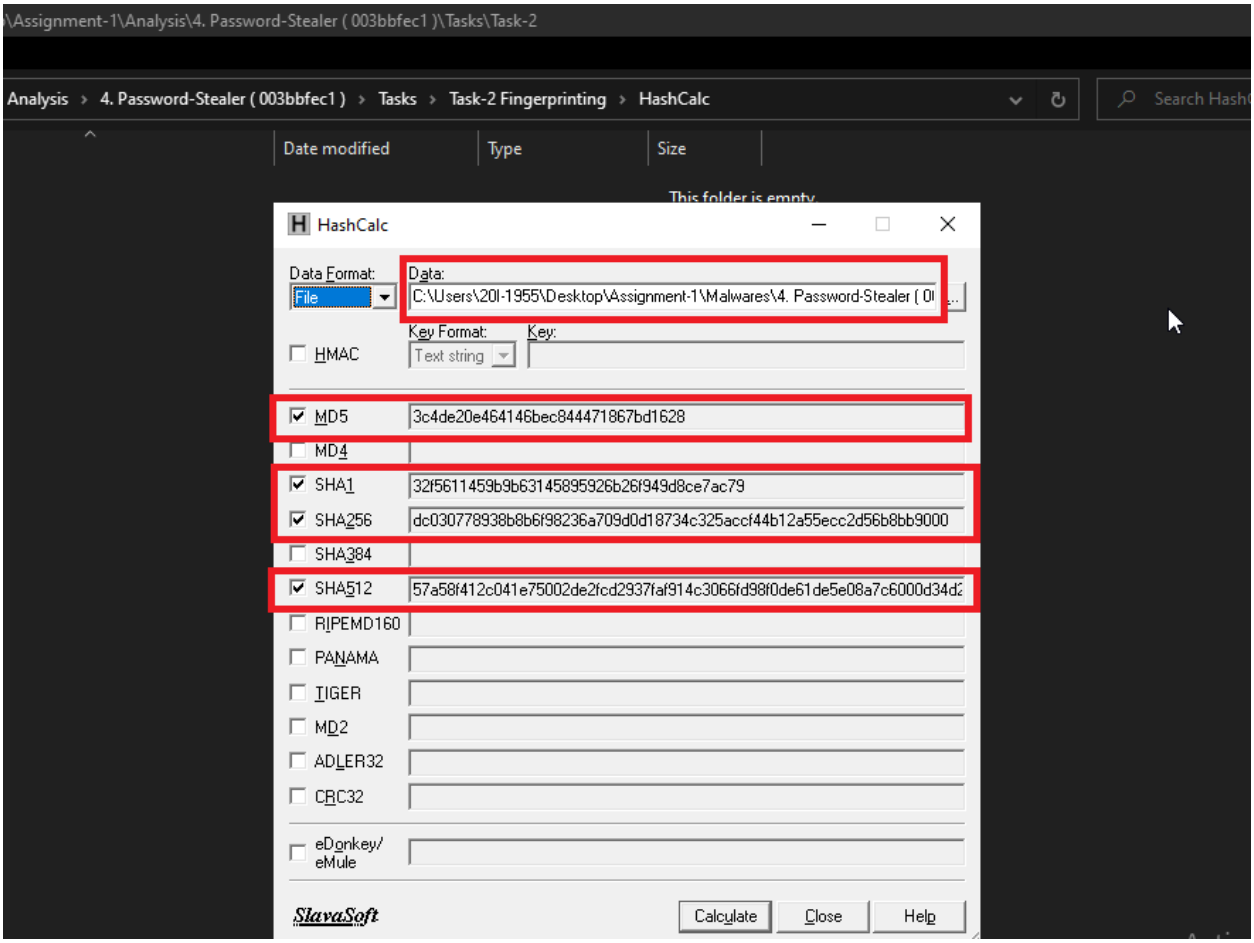

Figure 84: Malware-4 Fingerprinting through HashCalc

Tool-2 Python Hashlib Library

To find the hash of the malware using the python Hashlib library I write the following code.

```
import hashlib

hashMd5 = hashlib.md5(open("C:\\Users\\20I-1955\\Desktop\\Assignment-
1\\Malwares\\4. Password-Stealer ( 003bbfec1
)\\dc030778938b8b6f98236a709d0d18734c325accf44b12a55ecc2d56b8bb9000\\dc03077893
8b8b6f98236a709d0d18734c325accf44b12a55ecc2d56b8bb9000",
"rb").read()).hexdigest()
```

```
hashSha1 = hashlib.sha1(open("C:\\Users\\20I-1955\\Desktop\\Assignment-
1\\Malwares\\4. Password-Stealer ( 003bbfec1
)\\dc030778938b8b6f98236a709d0d18734c325accf44b12a55ecc2d56b8bb9000\\dc03077893
8b8b6f98236a709d0d18734c325accf44b12a55ecc2d56b8bb9000",
"rb").read()).hexdigest()
hashSha256 = hashlib.sha256(open("C:\\Users\\20I-1955\\Desktop\\Assignment-
1\\Malwares\\4. Password-Stealer ( 003bbfec1
)\\dc030778938b8b6f98236a709d0d18734c325accf44b12a55ecc2d56b8bb9000\\dc03077893
8b8b6f98236a709d0d18734c325accf44b12a55ecc2d56b8bb9000",
"rb").read()).hexdigest()
hashSha512 = hashlib.sha512(open("C:\\Users\\20I-1955\\Desktop\\Assignment-
1\\Malwares\\4. Password-Stealer ( 003bbfec1
)\\dc030778938b8b6f98236a709d0d18734c325accf44b12a55ecc2d56b8bb9000\\dc03077893
8b8b6f98236a709d0d18734c325accf44b12a55ecc2d56b8bb9000",
"rb").read()).hexdigest()

print("MD5:    ",hashMd5)
print("SHA1:   ",hashSha1)
print("SHA256: ",hashSha256)
print("SHA512: ",hashSha512)
```

After running the code, I found that the hashes of the malware file

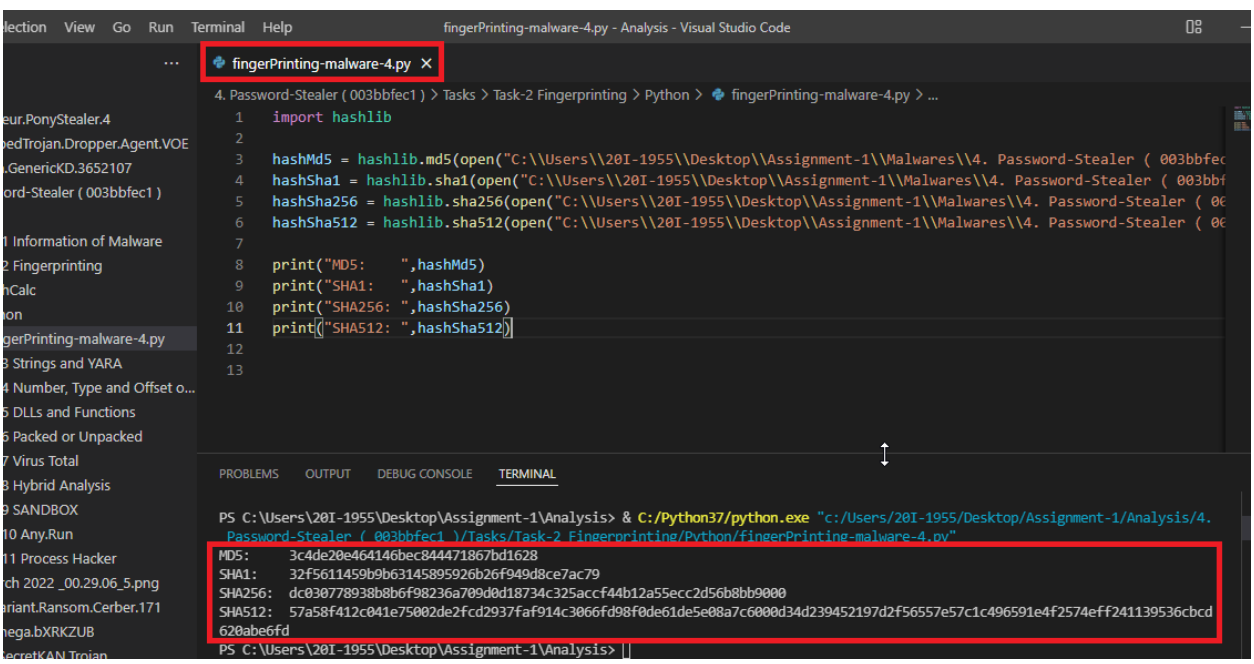

Figure 85: Malware-4 Fingerprinting through Python Hashlib Library

Task-3 What type of malicious strings are in the Malware file?

To extract the strings from the malware file I use the tool called "**Strings**" and then to scan the malware according to the extracted strings I write the YARA rule.

Tool-1 Strings

To extract the strings from the malware file and save them in the text file I run the following command

```
strings malware file path > text file name
```

After running the command all the strings are extracted from the malware file and stored in the text file.

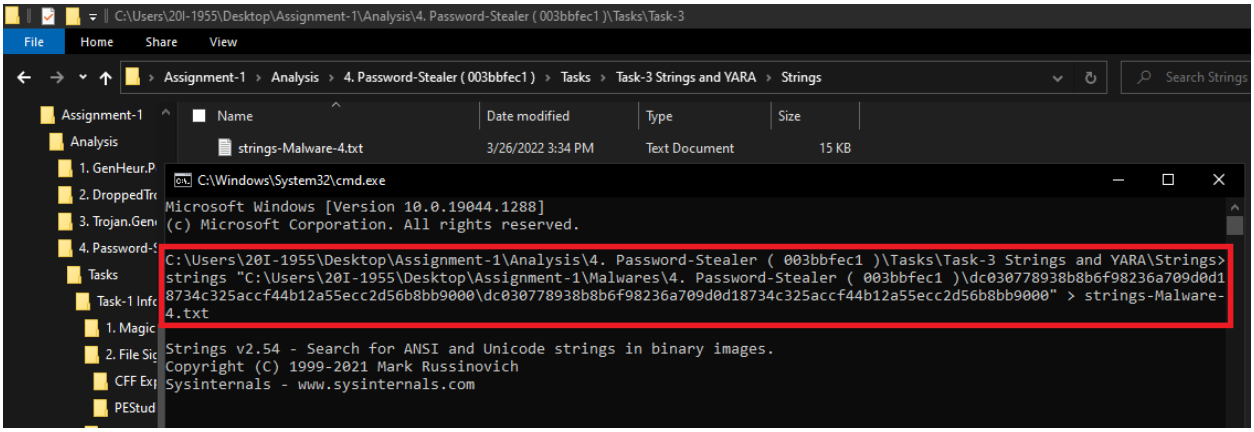

Figure 86: Malware-4 Strings Extraction through Strings Tool

After that, I open the text file to check the strings that were extracted from the malware and found some malicious strings.

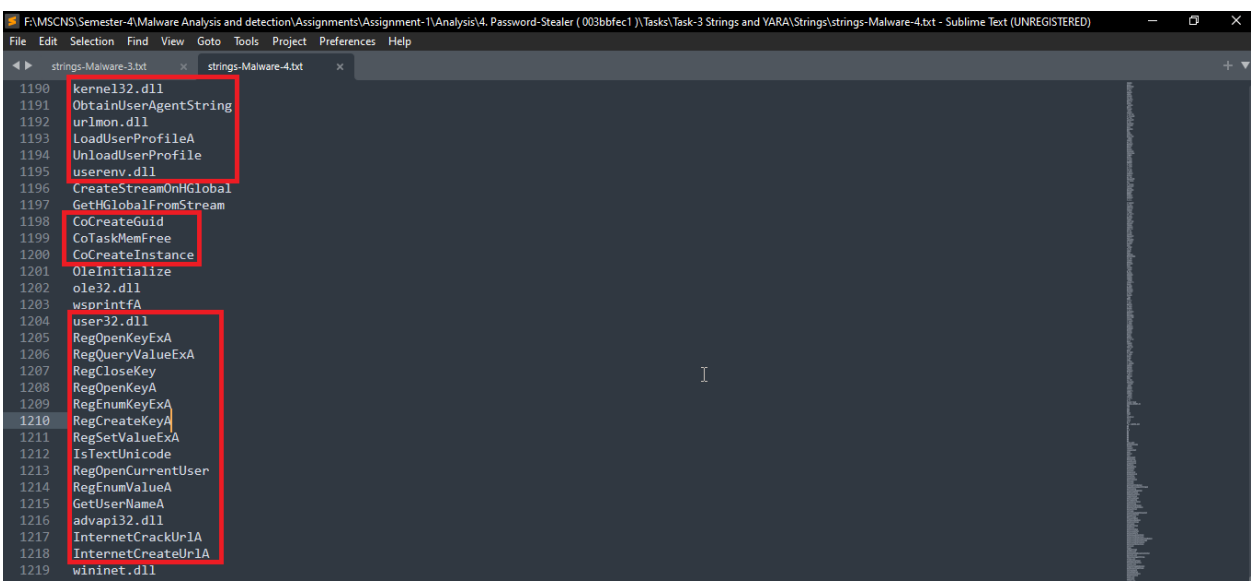

Figure 87: Malware-4 Strings Extracted through Strings Tool

Tool-2 YARA Rule

After extracting the string my next task is to write the YARA rule that scans the file and triggers the warning if it finds the malicious string in the malware file. For that, I write the following rule

```
rule malware_Password_Stealer_003bbfec1{
    meta:
        malwareName = "Password-Stealer ( 003bbfec1 )"
        ruleAuthor = "20I-1955_Muhammad Osama Khalid"
    strings:
        $str1 = "RegOpenKey" nocase
        $str2 = "RegQueryValue" nocase
        $str3 = "RegCloseKey" nocase
        $str4 = "kernel32" nocase
        $str5 = "shell32" nocase
    condition:
        any of them
}
```

Note: These strings are extracted from analyzing all the malware samples that are given for this assignment.

After that, I run the rule and find that the rule that I write works successfully as it detects the malicious string extracted above in the malware file.

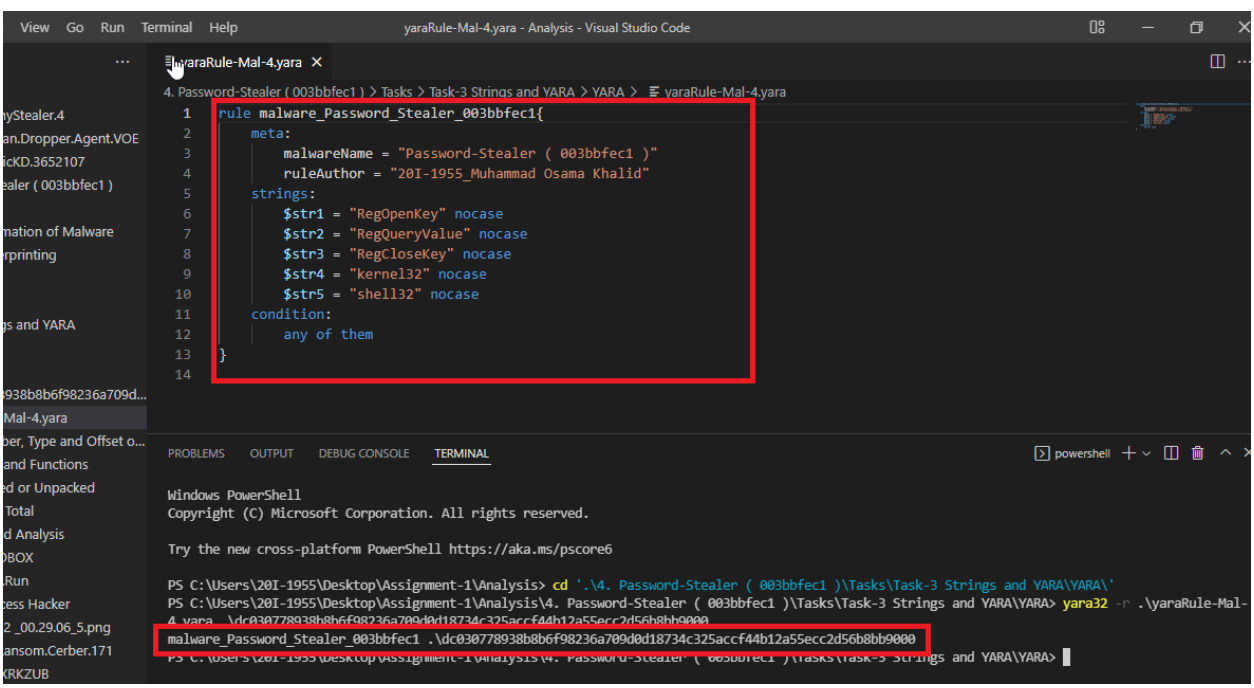

Figure 88: Malware-4 Malware detection through YARA Rule

Task-4 Number, type, and offset of different sections of Malware?

Sections contain the important information about the file as it describes how the executable is made. To find the number, type, and offset of different sections of malware files I use three different tools PE Studio and Exeinfo.

Tool-1 PE Studio

To find the number, type, and offset of different sections of the malware file I open the file in the PE Studio, then from the left panel first I click on "**file-header**" and then from the right panel I found the column named "**sections**". From there I found that this malware file has three sections and to find then what they are I click on "**sections**" in the left panel. After that from the right panel, I found the sections that this malware file has.

No. of Section	Type	Offset
1	.text	0x10001000
2	.rdata	0x1000E000
3	.data	0x1000F000
4	.reloc	0x10013000

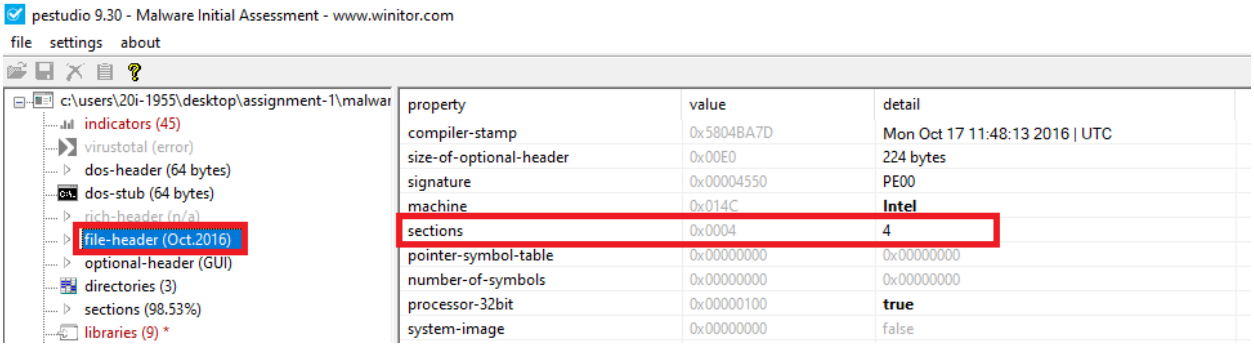

Figure 89: Malware-4 Information of Sections through PE Studio

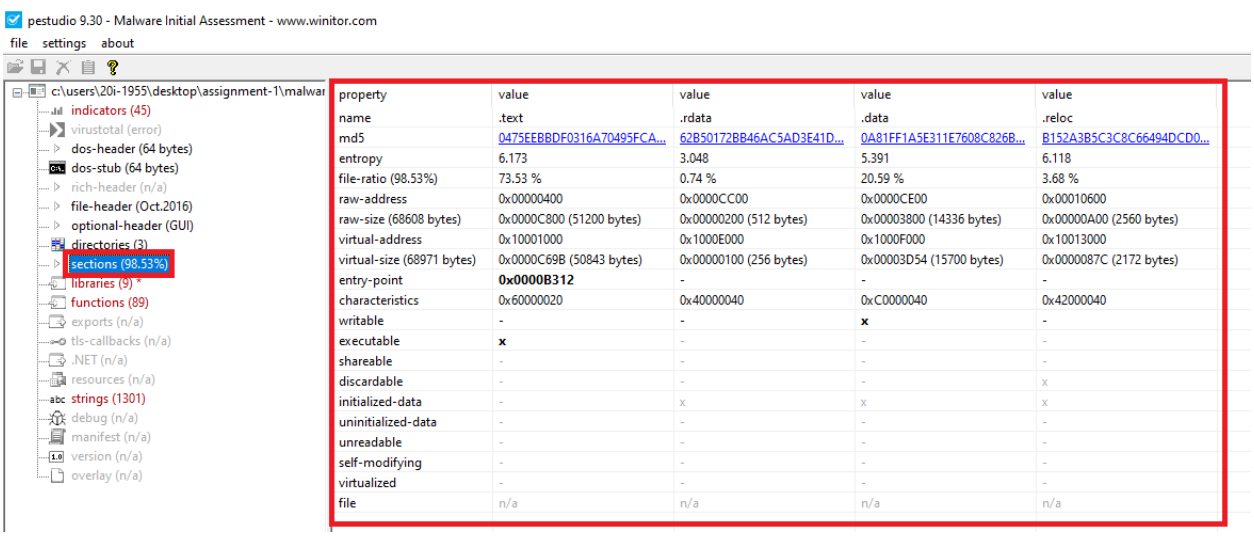

Figure 90: Malware-4 Information of Sections through PE Studio

Task-5 Which DLLs and their number of functions are used by Malware?

To find what DLLs and functions this malware used I used the tool PE Studio.

Tool-1 PE Studio

To find the DLLs and their number of functions used by malware files I open the file in the PE Studio, then from the left panel first I click on "**libraries**" and then from the right pane I found the list of DLLs and their functions used by this malware.

No. of DLL	Name	No. of functions
1	wsock32.dll	10
2	kernel32.dll	50
3	urlmon.dll	1
4	userenv.dll	2
5	ole32.dll	6
6	user32.dll	1
7	advapi32.dll	11
8	wininet.dll	2
9	shlwapi.dll	6

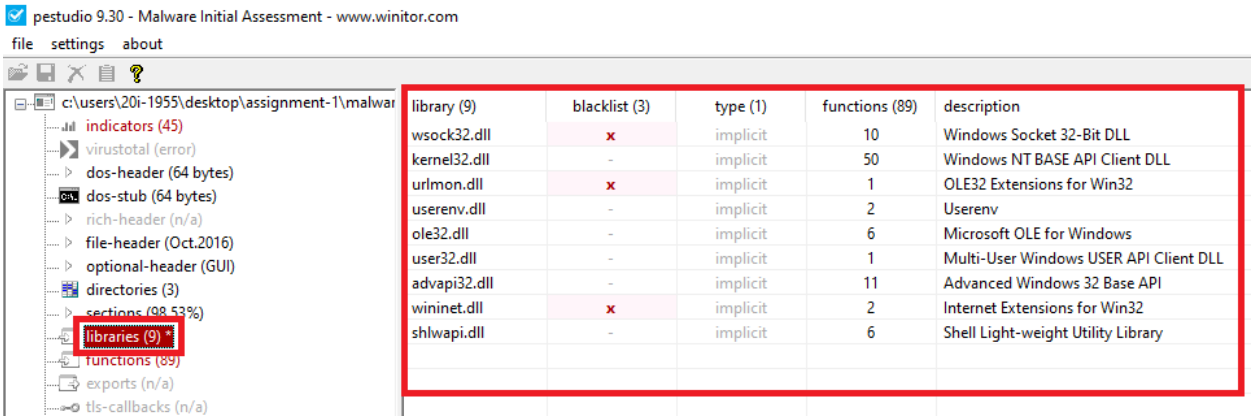

Figure 91: Malware-4 Information of DLLs and functions through PE Studio

Task-6 What is packing information if Malware is packed?

To check whether the malware is packed or not I use the tool Exeinfo.

Tool-1 Exeinfo

To find whether the malware file is packed or not I open the file in Exeinfo and at the bottom I found that the malware is not packed.

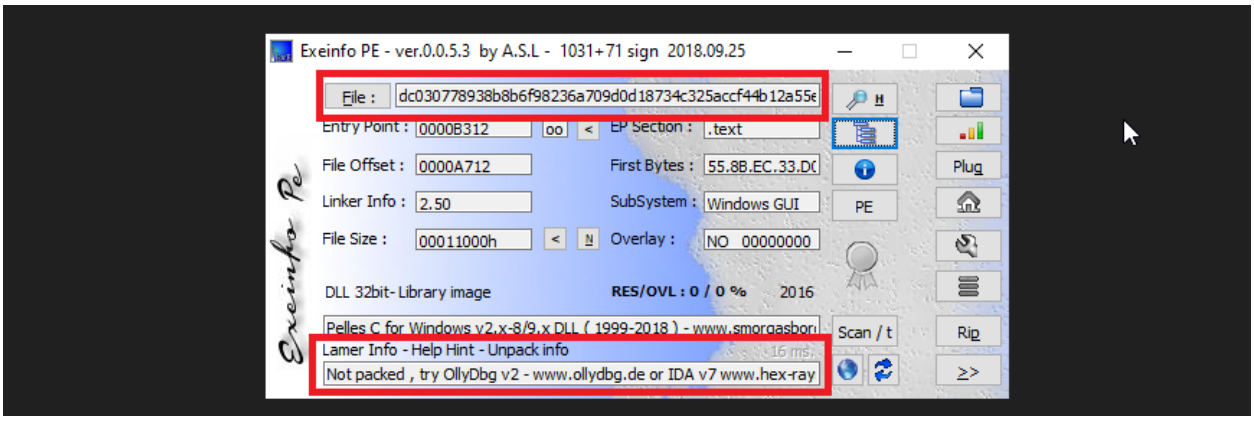

Figure 92: Malware-4 Packed or unpacked malware information through Exeinfo

Task-7 VirusTotal analysis of the malware file

To scan the malware file from the virus total I visit the website and upload the malware file. After some time, I found that the file is triggered as malicious by 60 antivirus search engines.

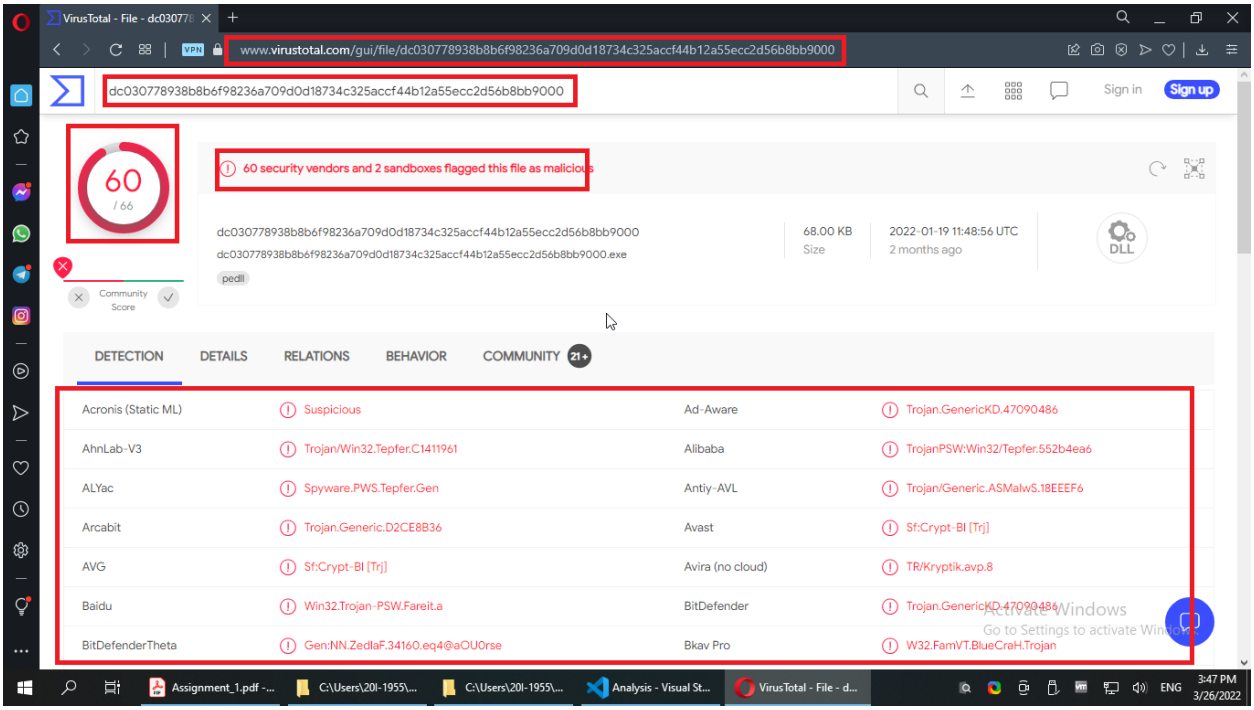

Figure 93: Malware-4 VirusTotal Analysis

Moreover, in the details tab, I find more information about the malware that I already explained in the previous tasks.

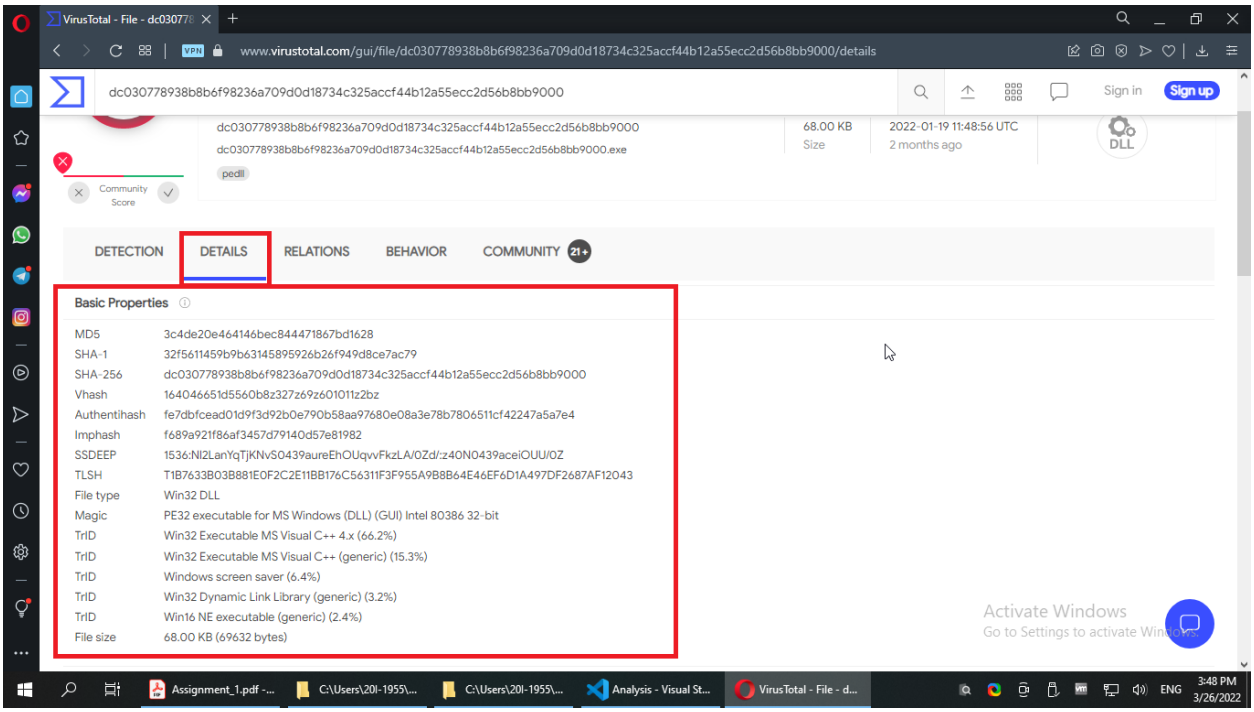

Figure 94: Malware-4 VirusTotal Analysis

Task-8 Analysis of the malware file through Hybrid Analysis

To scan the malware file from Hybrid Analysis I visit the website and upload the malware file. After some time, I found that the file is triggered as malicious by antivirus search engines.

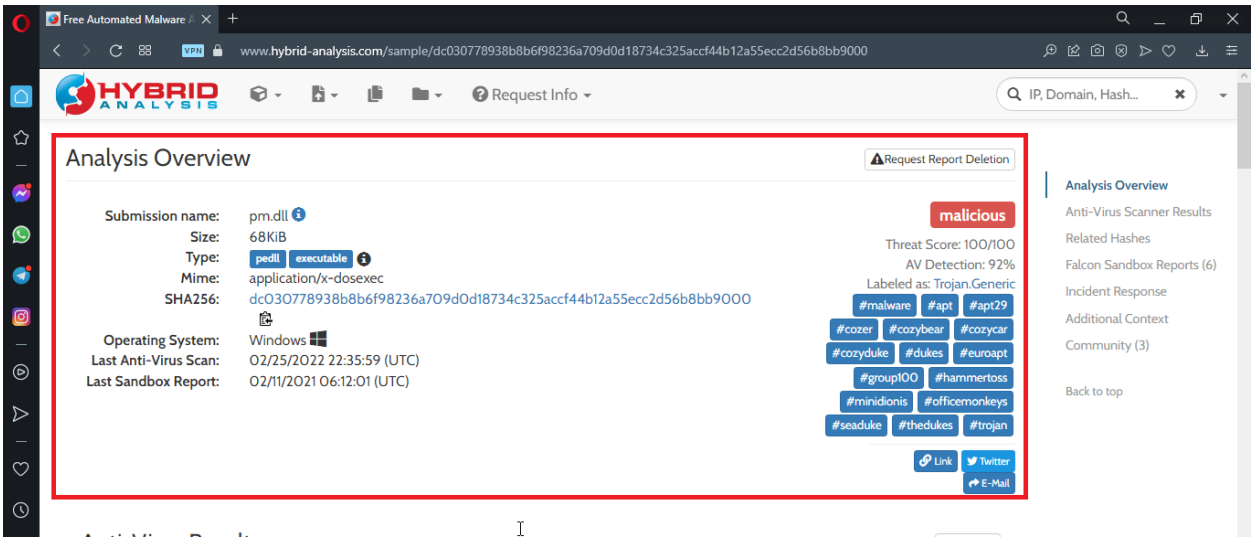

Figure 95: Malware-4 Analysis through Hybrid Analysis

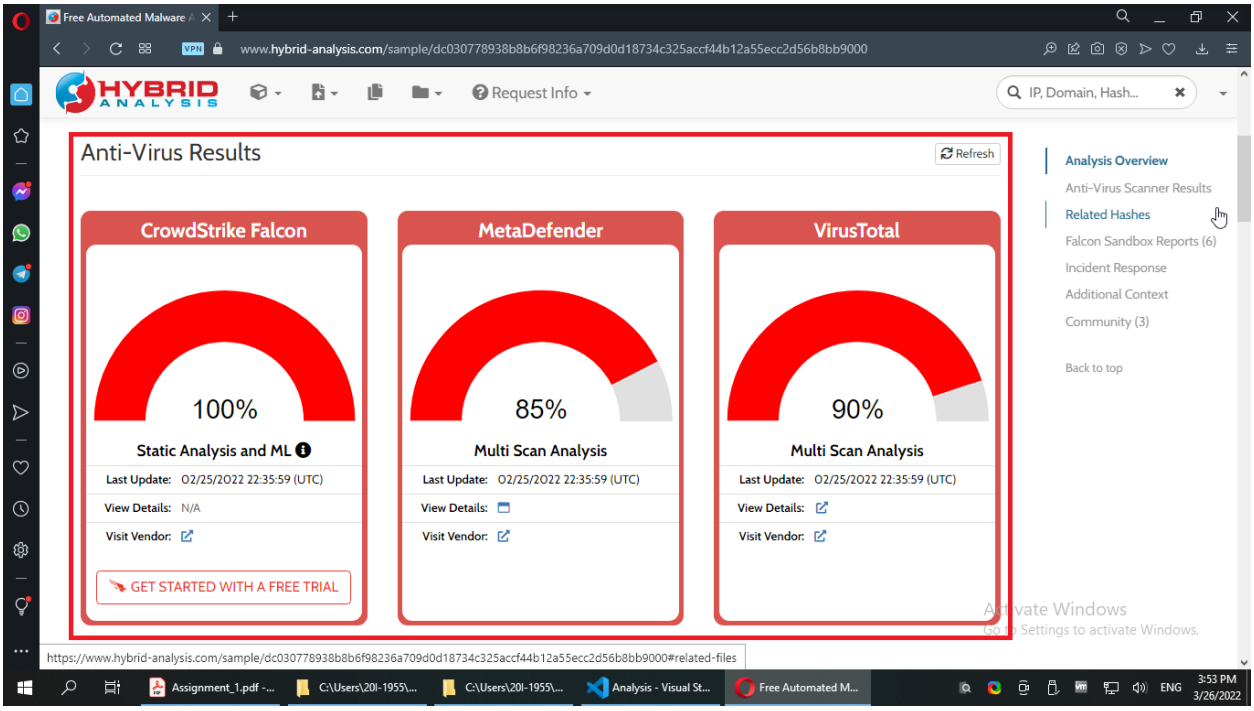

Figure 96: Malware-4 Analysis through Hybrid Analysis

Malware-5 GenVariant.Ransom.Cerber.171

This malware belongs to the family of ransomware that encrypts the victim's data and demands ransom the decrypt it.

Task-1 What is complete information malware have

In this task, I have to find the basic information of the malware that includes identifying magic byte, signatures, machine info, exe type, and file type.

Task-1.1 Magic Byte

To find the magic byte of the malware there are a bunch of tools available and for this task, I have used three different tools Notepad++, HxD, and CFF Explorer.

Tool-1 Notepad++

To find the magic byte of the malware file I open the file in Notepad++, then from the plugin tab click on “HEX-Editor” and then click on “View in HEX”. On viewing the file in HEX-Editor I found that the magic byte of the malware file is “**MZ**”. This means that the malware file is executable. Moreover, I also find that this file can’t be able to run in DOS mode.

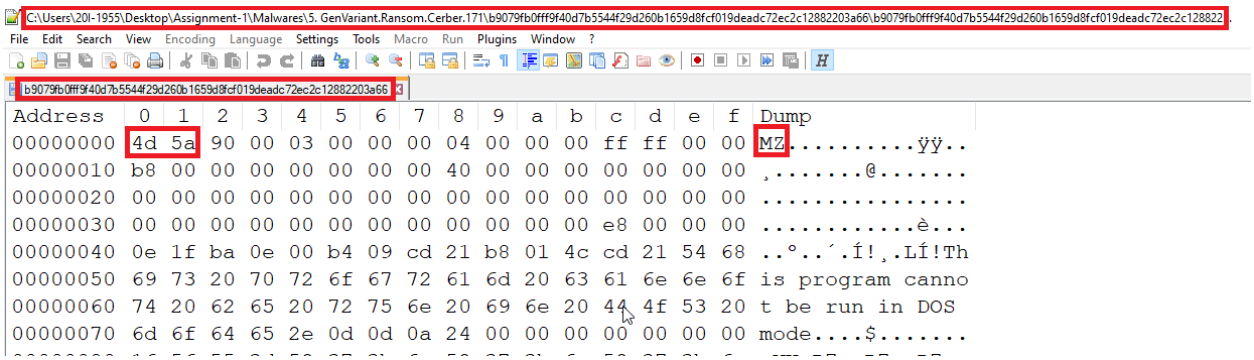

Figure 97: Malware-5 Magic Byte through Notepad++

Tool-2 HxD

To find the magic byte of the malware file I open the file in the HxD and find that the magic byte of the file is “**MZ**” which means that the malware file is executable. Moreover, I also find that this file can’t be able to run in DOS mode.

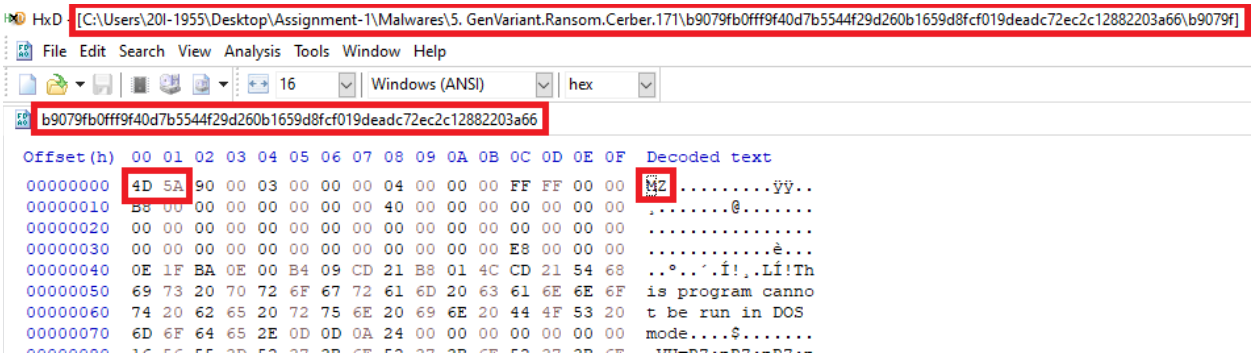

Figure 98: Malware-5 Magic Byte through HxD

Tool-3 CFF Explorer

To find the magic byte of the malware file I open the file in the CFF Explorer, then from the left panel I click on "**DOS Header**". In the "**DOS Header**" I find the magic byte in the "e_magic" column that is "**5A 4D**" which is equivalent to "**4D 5A**" because in CFF Explorer hexadecimal are represented in right to left order. So "**4D = M**" and "**5A = Z**" mean that the malware file is executable.

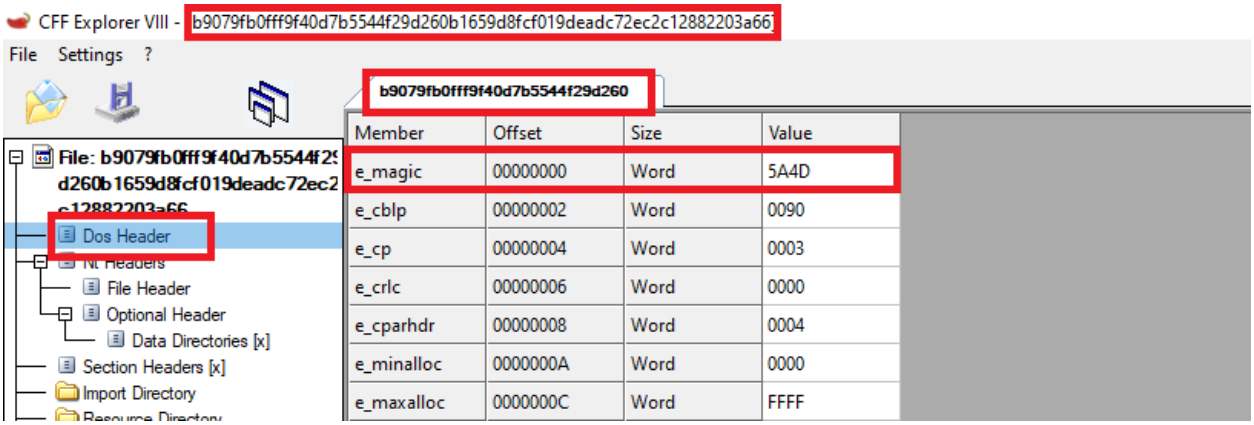

Figure 99: Malware-5 Magic Byte through CFF Explorer

Task-1.2 File Signature

A file signature is used to identify the content of the file. This information is stored in the file's header. To find the file signature of the malware I have used two different tools PE Studio and CFF Explorer.

Tool-1 PE Studio

To find the file signature of the malware file I open the file in the PE Studio, then from the left panel I click on "**file-header**" and then from the right panel I found the column named "**signature**". From there I found that the signature of the file is "**PE**" which means that this file is portable executable.

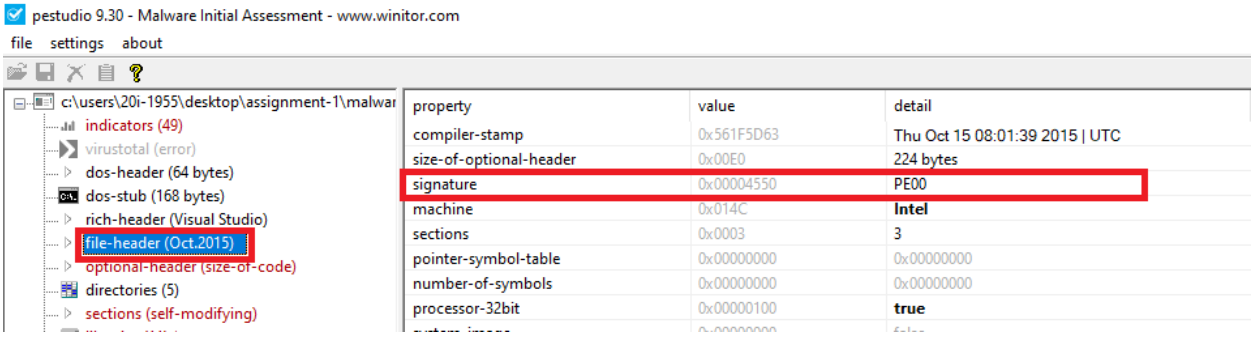

Figure 100: Malware-5 File Signature through PE Studio

Tool-2 CFF Explorer

To find the file signature of the malware file I open the file in the CFF Explorer, then from the left panel I click on "**Nt Headers**" and then from the right panel I found the column named "**signature**". From there I found that the signature of the file is "**45 50**" which is equivalent to "**50 45**" because in CFF Explorer hexadecimal are represented in right to left order. So "**50 = P**" and "**45 = E**" which means that this file is portable executable.

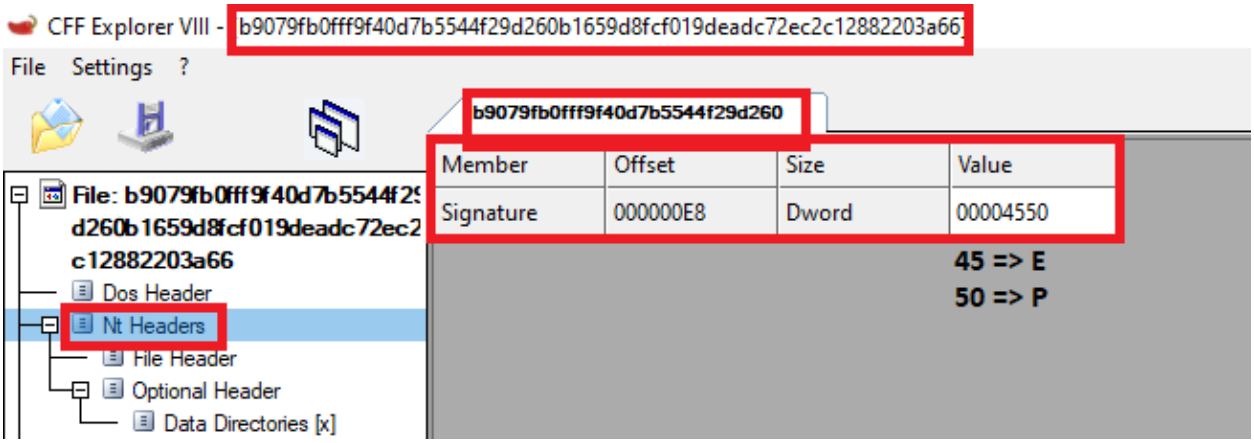

Figure 101: Malware-5 File Signature through CFF Explorer

Task-1.3 Machine Information

To find the machine information of the malware I have used two different tools PE Studio and CFF Explorer.

Tool-1 PE Studio

To find the file signature of the malware file I open the file in the PE Studio, then from the left panel I click on "**file-header**" and then from the right panel I found the column named "**machine**". From there I found that this malware is designed for Intel-based machines. Other than machine information I also find that when the malware is created from the "**compiler-stamp**" column. This malware is created on Oct 15, 2015, on Sunday at 08:01 AM. Also find that this malware file has 3 sections and the size of the optional header is 224 bytes.

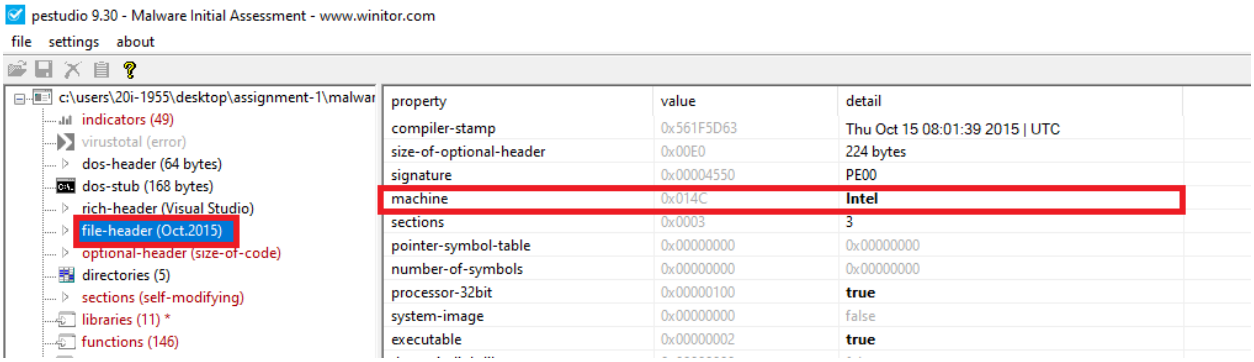

Figure 102: Malware-5 Machine Information through PE Studio

Tool-2 CFF Explorer

To find the file signature of the malware file I open the file in the CFF Explorer, then from the left panel I click on "**file header**" and then from the right panel I found the column named "**Machine**". From there I found that this malware is designed for Intel 386 based machines.

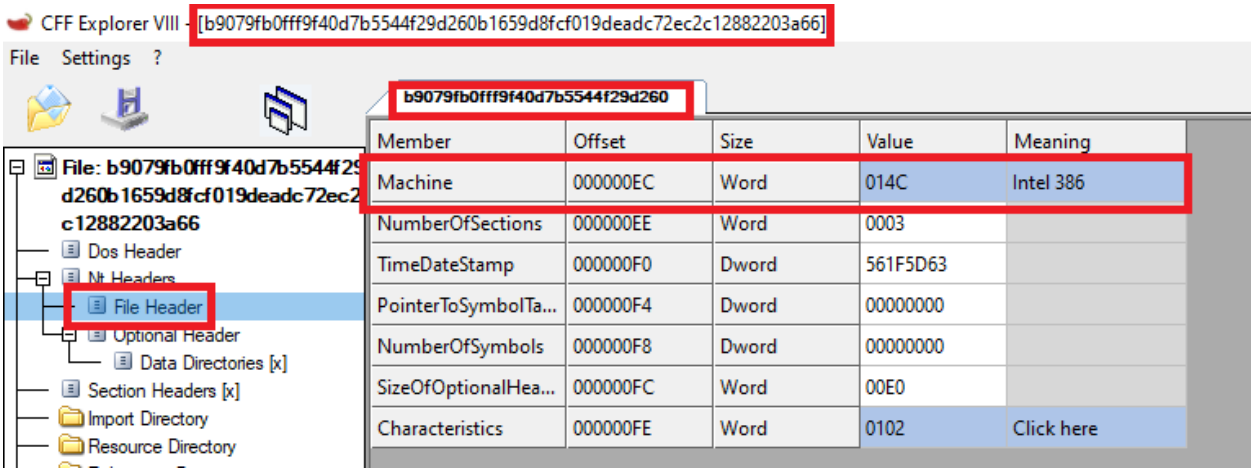

Figure 103: Malware-5 Machine Information through CFF Explorer

Task-1.4 Exe Type

Exe type is used to identify which type of executable it is either 32-bit or 64-bit and to find that I used two different tools PE Studio and CFF Explorer.

Tool-1 PE Studio

To find the exe type of the malware file I open the file in the PE Studio, then from the left panel I click on "**optional-header**" and then from the right panel I found the column named "**magic**". From there I found that the exe type of this malware file is "0x010B" which means that this malware file is a 32-bit portable executable. Other than exe type I also find the entry point of the malware which is ".?865771".

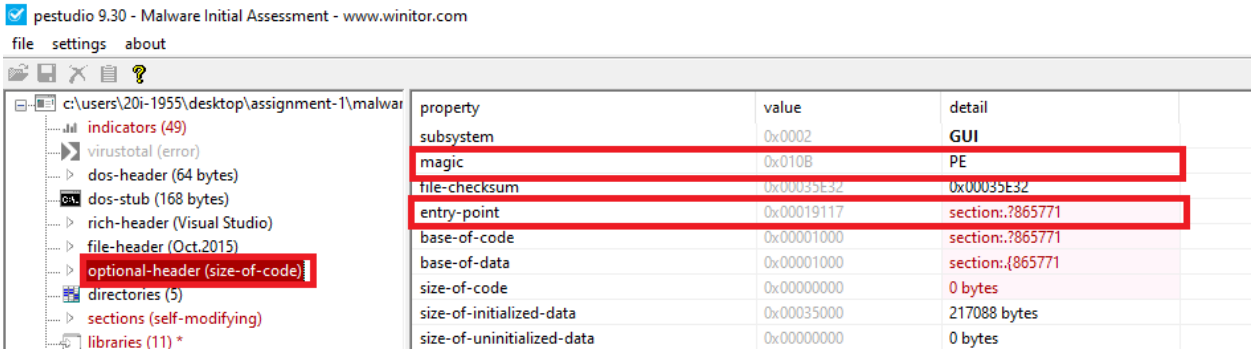

Figure 104: Malware-5 Exe type through PE Studio

Tool-2 CFF Explorer

To find the file signature of the malware file I open the file in the CFF Explorer, then from the left panel I click on "**optional header**" and then from the right panel I found the column named "**Magic**". From there I found that the exe type of this malware file is "010B" which means that this malware file is a 32-bit portable executable. Other than exe type I also find the entry point of the malware which is ".{865771".

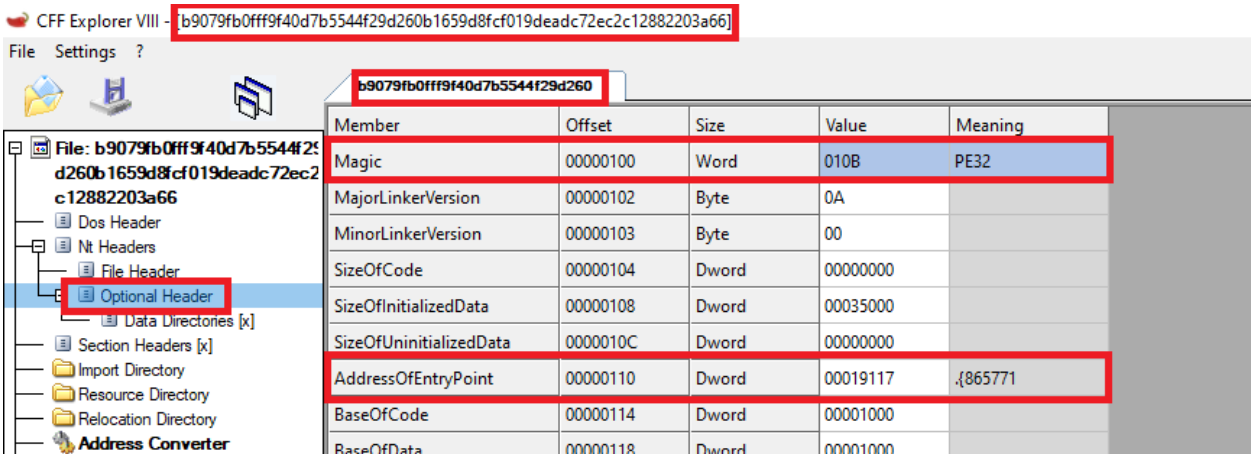

Figure 105: Malware-5 Exe type through CFF Explorer

Task-1.5 File Type

File Type is used to identify which type of file it is. I used two different tools Python Magic Library and TrID Net.

Tool-1 Python Magic Library

To find the file type of the malware using the python magic library I write the following code.

```
import magic

mgicByte = magic.from_file('C:\\Users\\20I-1955\\Desktop\\Assignment-
1\\Malwares\\5.
GenVariant.Ransom.Cerber.171\\b9079fb0fff9f40d7b5544f29d260b1659d8fcf019deadc72
ec2c12882203a66\\b9079fb0fff9f40d7b5544f29d260b1659d8fcf019deadc72ec2c12882203a
66')

print(mgicByte)
```

After running the code, I found that the file type of malware is "PE32 executable (GUI) Intel 80386, for MS Windows".

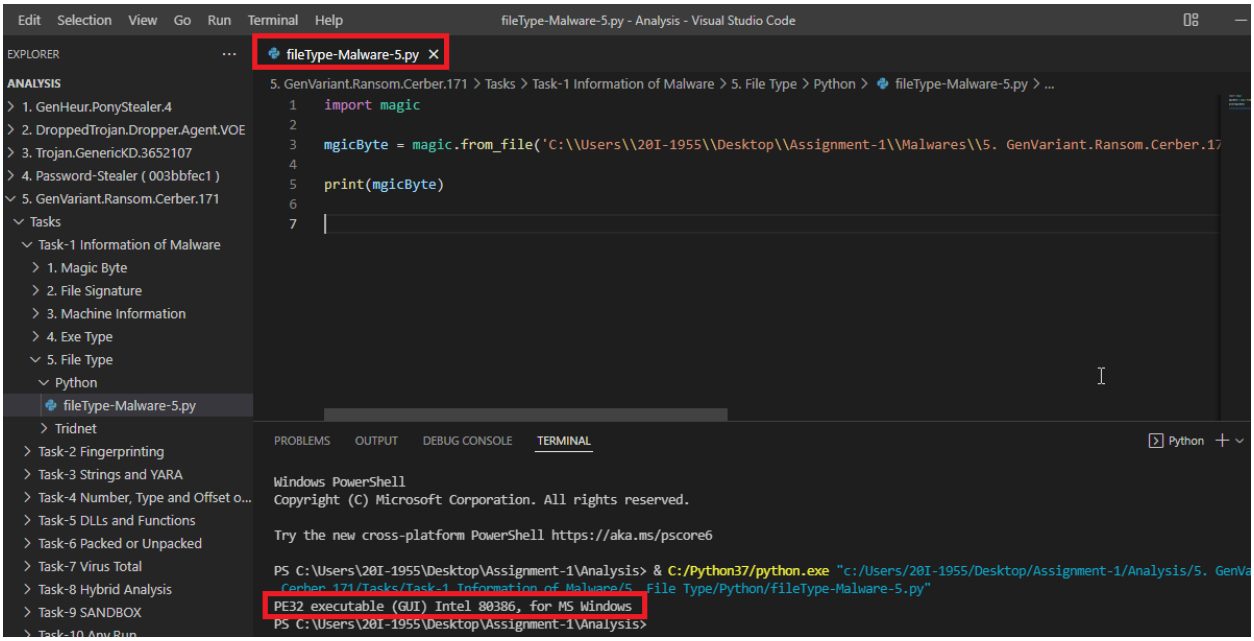

Figure 106: Malware-5 File Type through Python Magic Library

Tool-2 TrID Net

To find the file type of the malware using TrID Net I open the file in TrID Net and find that the malware file is “Win32 Dynamic-Link Library” with a “29.7%” match.

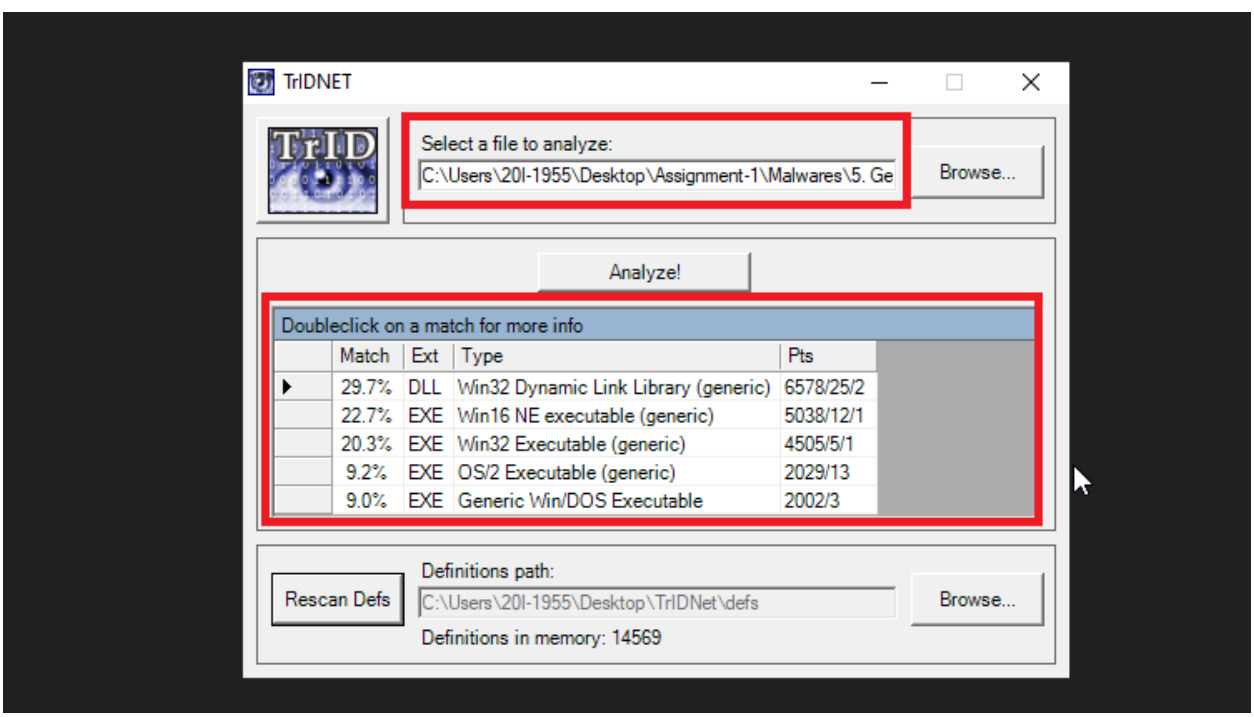

Figure 107: Malware-5 File Type through TrID Net

Task-2 What is the fingerprinting information of the malware file?

To find the hashes of the malware file I have used two different tools HashCalc and Python hashlib library.

Tool-1 HashCalc

To find the hash function of the file in HashCalc I open the file in HashCalc and find the hash of the file.

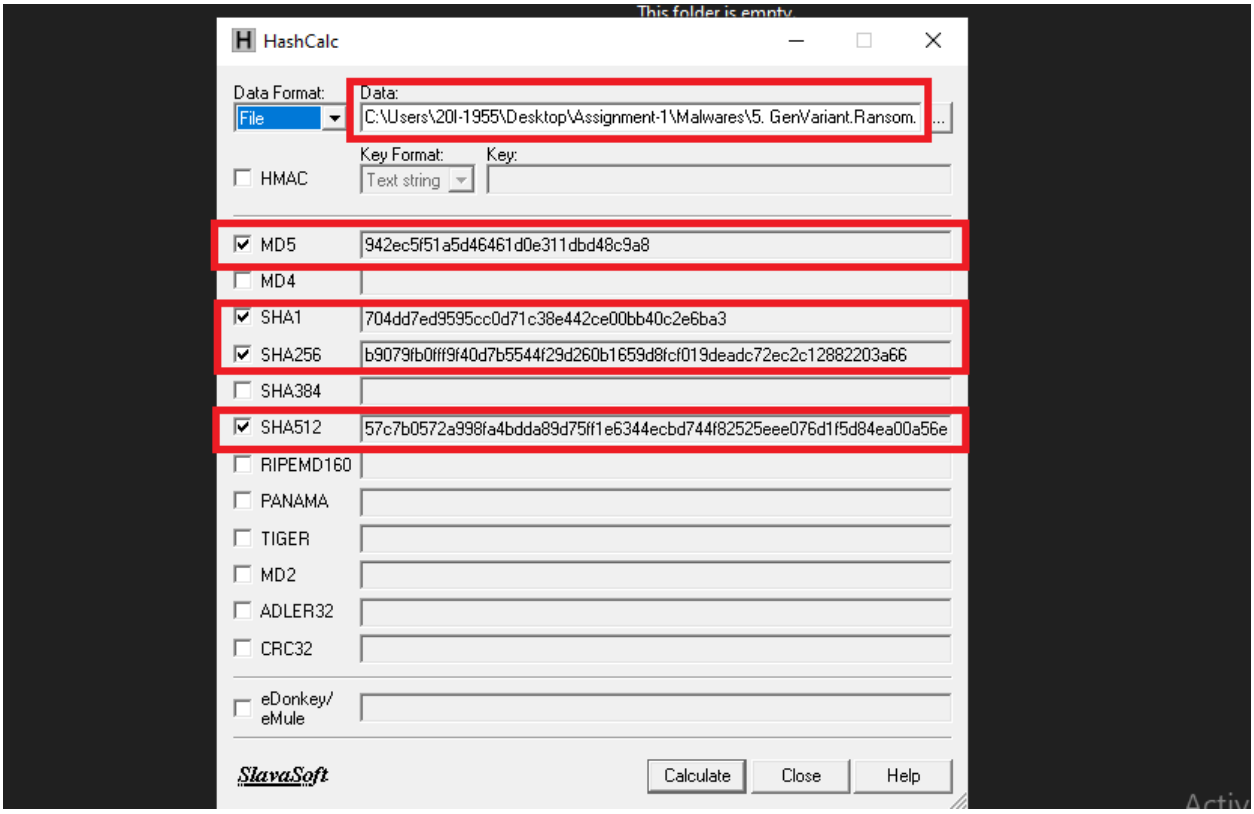

Figure 108: Malware-5 Fingerprinting through HashCalc

Tool-2 Python Hashlib Library

To find the hash of the malware using the python Hashlib library I write the following code.

```
import hashlib

hashMd5 = hashlib.md5(open("C:\\Users\\20I-1955\\Desktop\\Assignment-
1\\Malwares\\5.
GenVariant.Ransom.Cerber.171\\b9079fb0fff9f40d7b5544f29d260b1659d8fcf019deadc72
ec2c12882203a66\\b9079fb0fff9f40d7b5544f29d260b1659d8fcf019deadc72ec2c12882203a
66", "rb").read()).hexdigest()
hashSha1 = hashlib.sha1(open("C:\\Users\\20I-1955\\Desktop\\Assignment-
1\\Malwares\\5.
GenVariant.Ransom.Cerber.171\\b9079fb0fff9f40d7b5544f29d260b1659d8fcf019deadc72
```

```
ec2c12882203a66\\b9079fb0fff9f40d7b5544f29d260b1659d8fcf019deadc72ec2c12882203a
66", "rb").read()).hexdigest()
hashSha256 = hashlib.sha256(open("C:\\Users\\20I-1955\\Desktop\\Assignment-
1\\Malwares\\5.
GenVariant.Ransom.Cerber.171\\b9079fb0fff9f40d7b5544f29d260b1659d8fcf019deadc72
ec2c12882203a66\\b9079fb0fff9f40d7b5544f29d260b1659d8fcf019deadc72ec2c12882203a
66", "rb").read()).hexdigest()
hashSha512 = hashlib.sha512(open("C:\\Users\\20I-1955\\Desktop\\Assignment-
1\\Malwares\\5.
GenVariant.Ransom.Cerber.171\\b9079fb0fff9f40d7b5544f29d260b1659d8fcf019deadc72
ec2c12882203a66\\b9079fb0fff9f40d7b5544f29d260b1659d8fcf019deadc72ec2c12882203a
66", "rb").read()).hexdigest()

print("MD5:    ",hashMd5)
print("SHA1:   ",hashSha1)
print("SHA256: ",hashSha256)
print("SHA512: ",hashSha512)
```

After running the code, I found that the hashes of the malware file

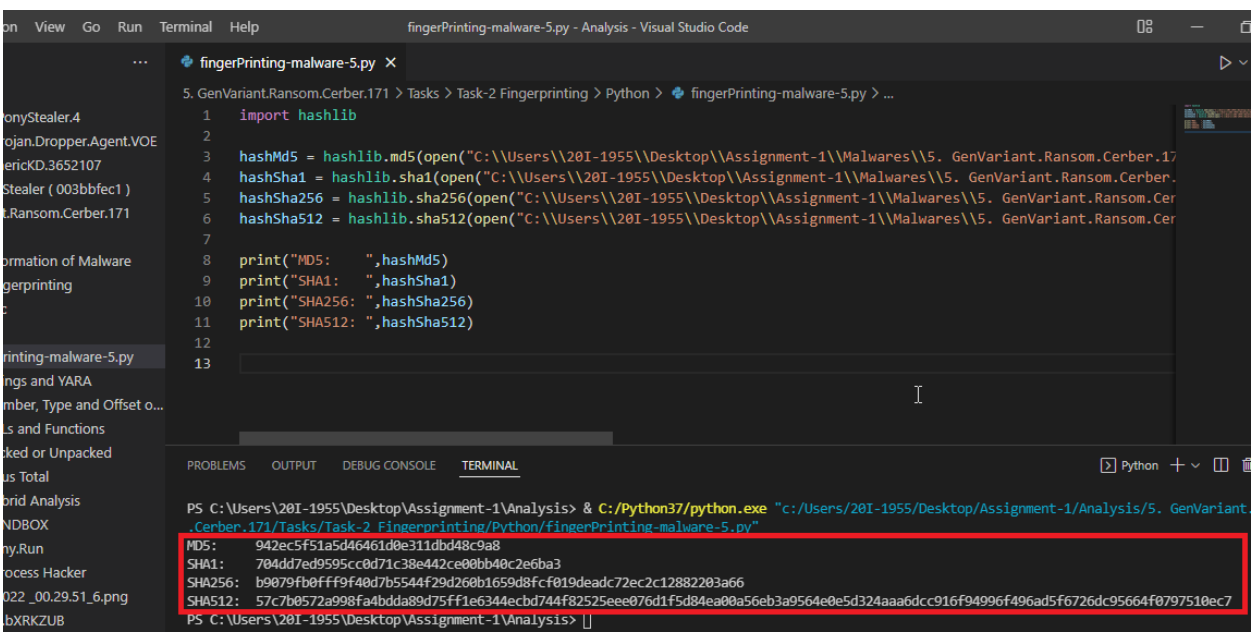

Figure 109: Malware-5 Fingerprinting through Python Hashlib Library

Task-3 What type of malicious strings are in the Malware file?

To extract the strings from the malware file I use the tool called "**Strings**" and then to scan the malware according to the extracted strings I write the YARA rule.

Tool-1 Strings

To extract the strings from the malware file and save them in the text file I run the following command

```
strings malware file path > text file name
```

After running the command all the strings are extracted from the malware file and stored in the text file.

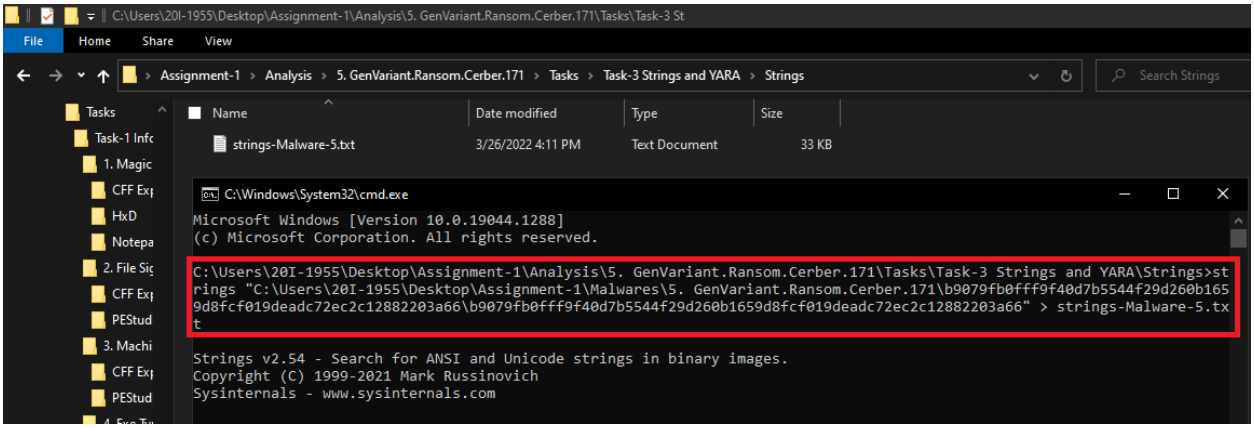

Figure 110: Malware-5 Strings Extraction through Strings Tool

After that, I open the text file to check the strings that were extracted from the malware and found some malicious strings.

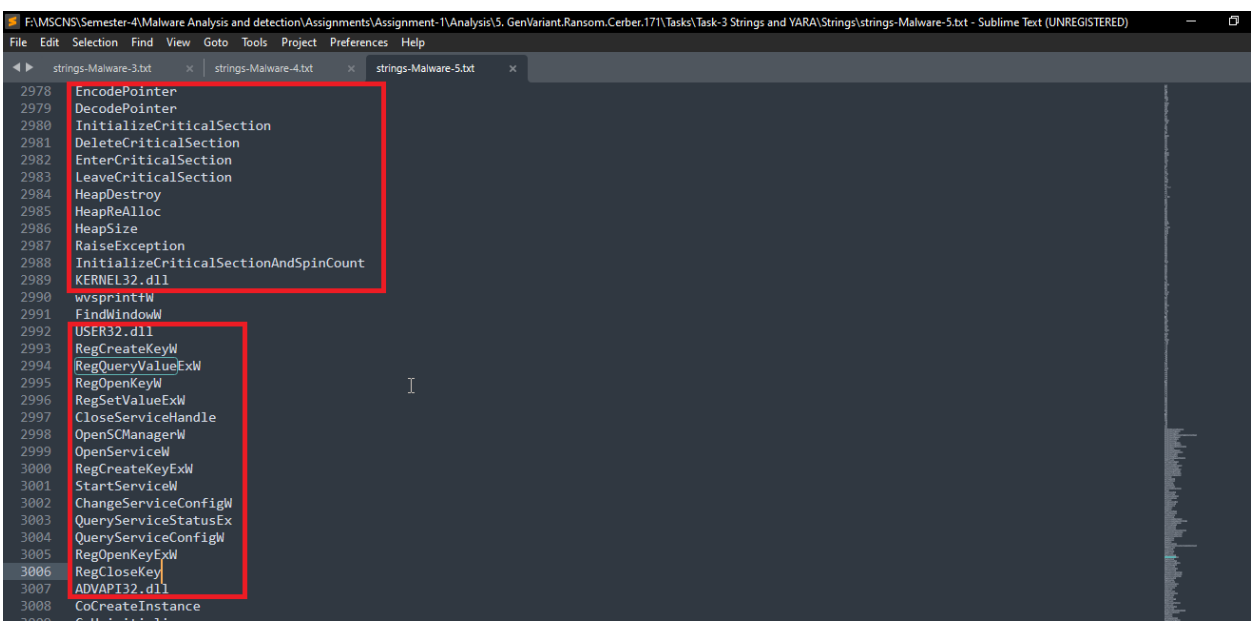

Figure 111: Malware-5 Strings Extracted through Strings Tool

Tool-2 YARA Rule

After extracting the string my next task is to write the YARA rule that scans the file and triggers the warning if it finds the malicious string in the malware file. For that, I write the following rule

```
rule malware_GenVariant_Ransom_Cerber_171{
    meta:
        malwareName = "GenVariant.Ransom.Cerber.171"
        ruleAuthor = "20I-1955_Muhammad Osama Khalid"
    strings:
        $str1 = "RegOpenKey" nocase
        $str2 = "RegQueryValue" nocase
        $str3 = "RegCloseKey" nocase
        $str4 = "kernel32" nocase
        $str5 = "shell32" nocase
    condition:
        any of them
}
```

Note: These strings are extracted from analyzing all the malware samples that are given for this assignment.

After that, I run the rule and find that the rule that I write works successfully as it detects the malicious string extracted above in the malware file.

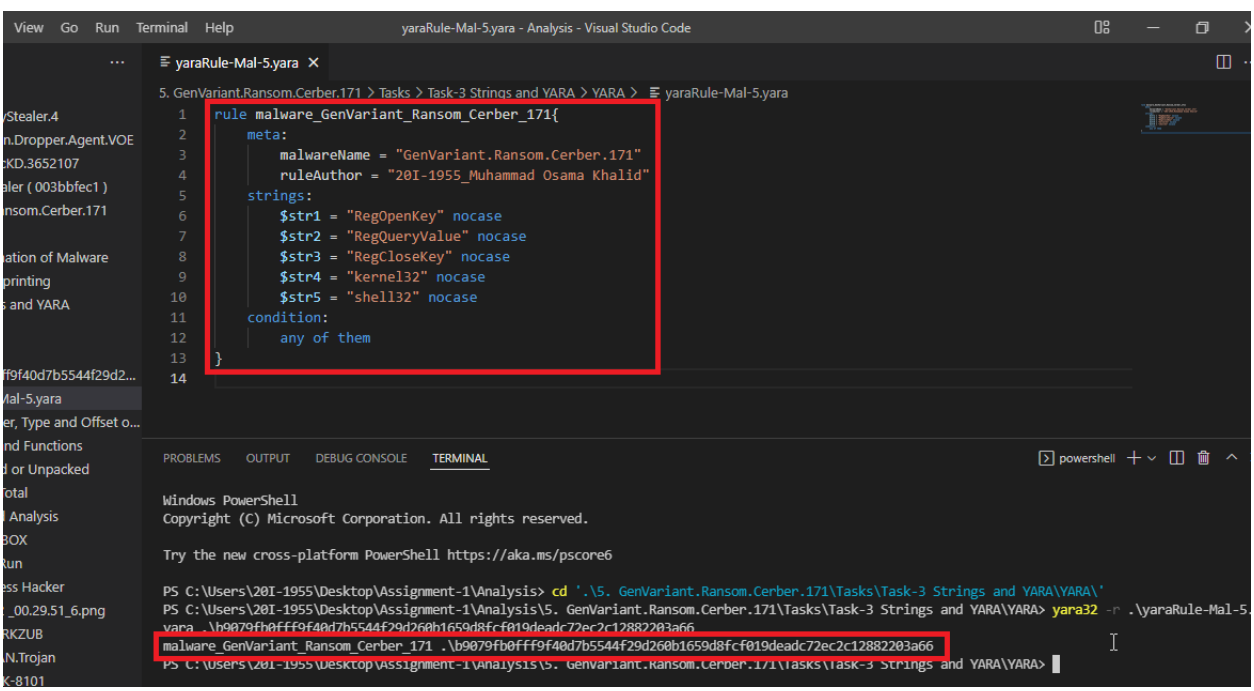

Figure 112: Malware-5 Malware detection through YARA Rule

Task-4 Number, type, and offset of different sections of Malware?

Sections contain the important information about the file as it describes how the executable is made. To find the number, type, and offset of different sections of malware files I use three different tools PE Studio and Exeinfo.

Tool-1 PE Studio

To find the number, type, and offset of different sections of the malware file I open the file in the PE Studio, then from the left panel first I click on "**file-header**" and then from the right panel I found the column named "**sections**". From there I found that this malware file has three sections and to find then what they are I click on "**sections**" in the left panel. After that from the right panel, I found the sections that this malware file has.

No. of Section	Type	Offset
1	.{865771	0x00401000
2	.rsrc	0x00433000
3	.reloc	0x00434000

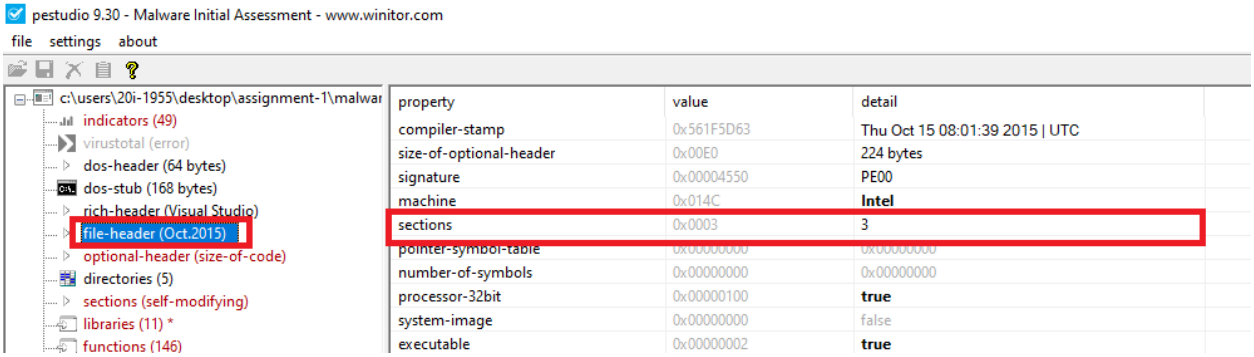

Figure 113: Malware-5 Information of Sections through PE Studio

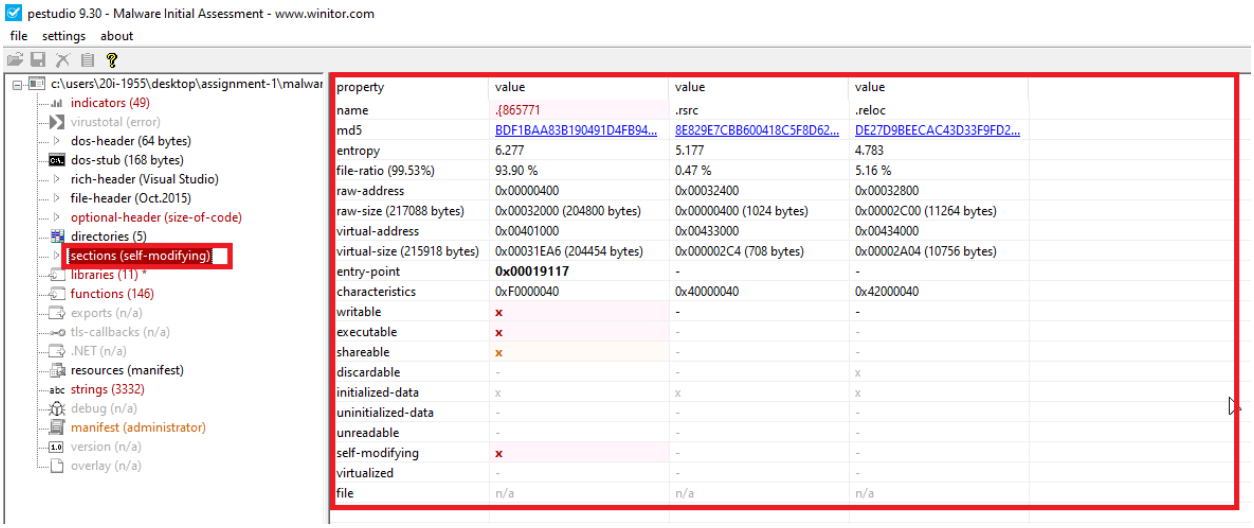

Figure 114: Malware-5 Information of Sections through PE Studio

Task-5 Which DLLs and their number of functions are used by Malware?

To find what DLLs and functions this malware used I used the tool PE Studio.

Tool-1 PE Studio

To find the DLLs and their number of functions used by malware files I open the file in the PE Studio, then from the left panel first I click on "**libraries**" and then from the right pane I found the list of DLLs and their functions used by this malware.

No. of DLL	Name	No. of functions
1	winhttp.dll	16
2	kernel32.dll	98
3	user32.dll	2
4	advapi32.dll	14
5	ole32.dll	3
6	oleaut32.dll	6
7	shlwapi.dll	1
8	rpcrt4.dll	2
9	urlmon.dll	1
10	secur32.dll	1
11	psapi.dll	2

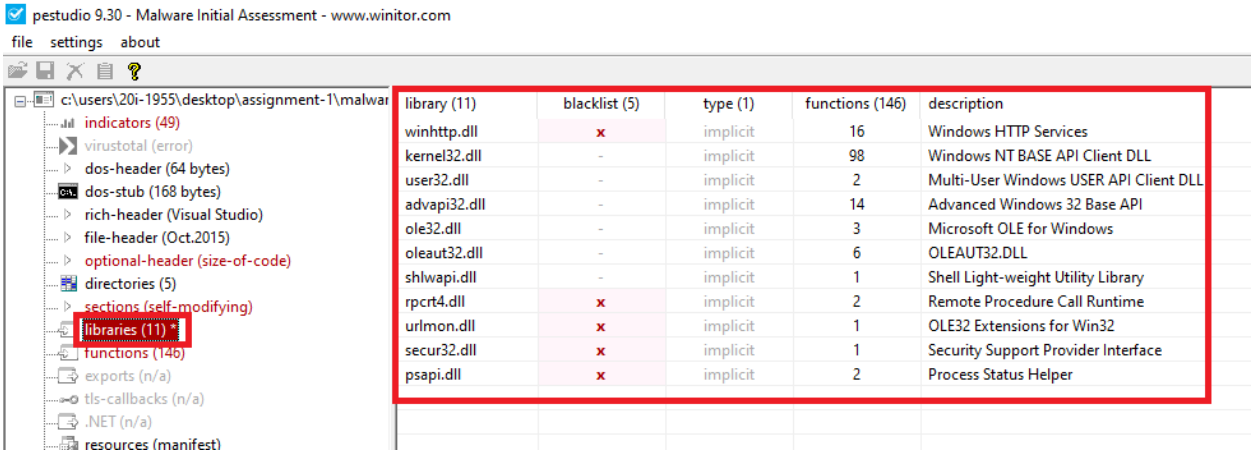

Figure 115: Malware-5 Information of DLLs and functions through PE Studio

Task-6 What is packing information if Malware is packed?

To check whether the malware is packed or not I use the tool Exeinfo.

Tool-1 Exeinfo

To find whether the malware file is packed or not I open the file in Exeinfo and at the bottom I found that the malware is packed with the unknown packer.

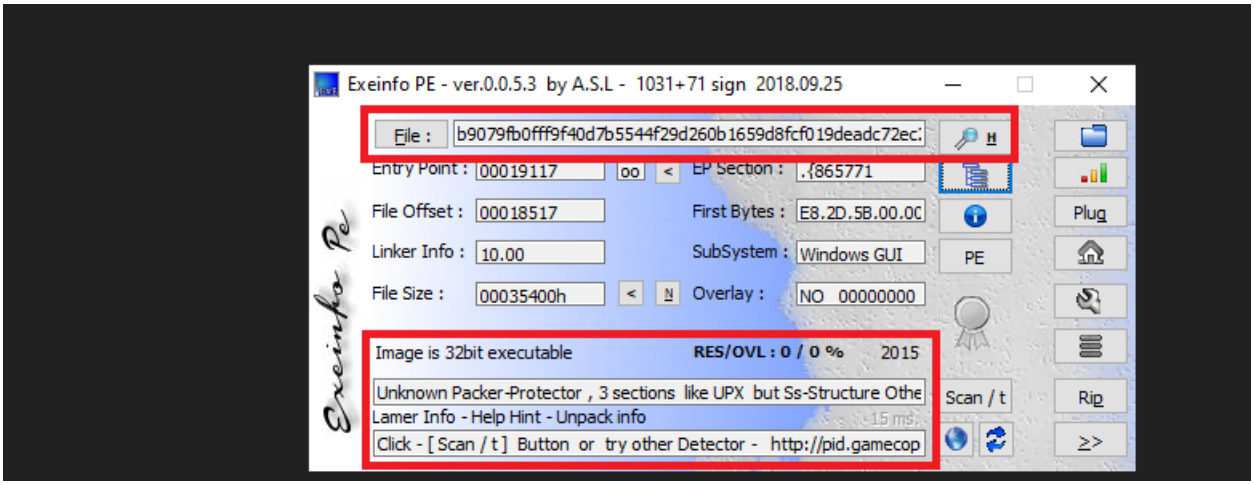

Figure 116: Malware-5 Packed or unpacked malware information through Exeinfo

Task-7 VirusTotal analysis of the malware file

To scan the malware file from the virus total I visit the website and upload the malware file. After some time, I found that the file is triggered as malicious by 58 antivirus search engines.

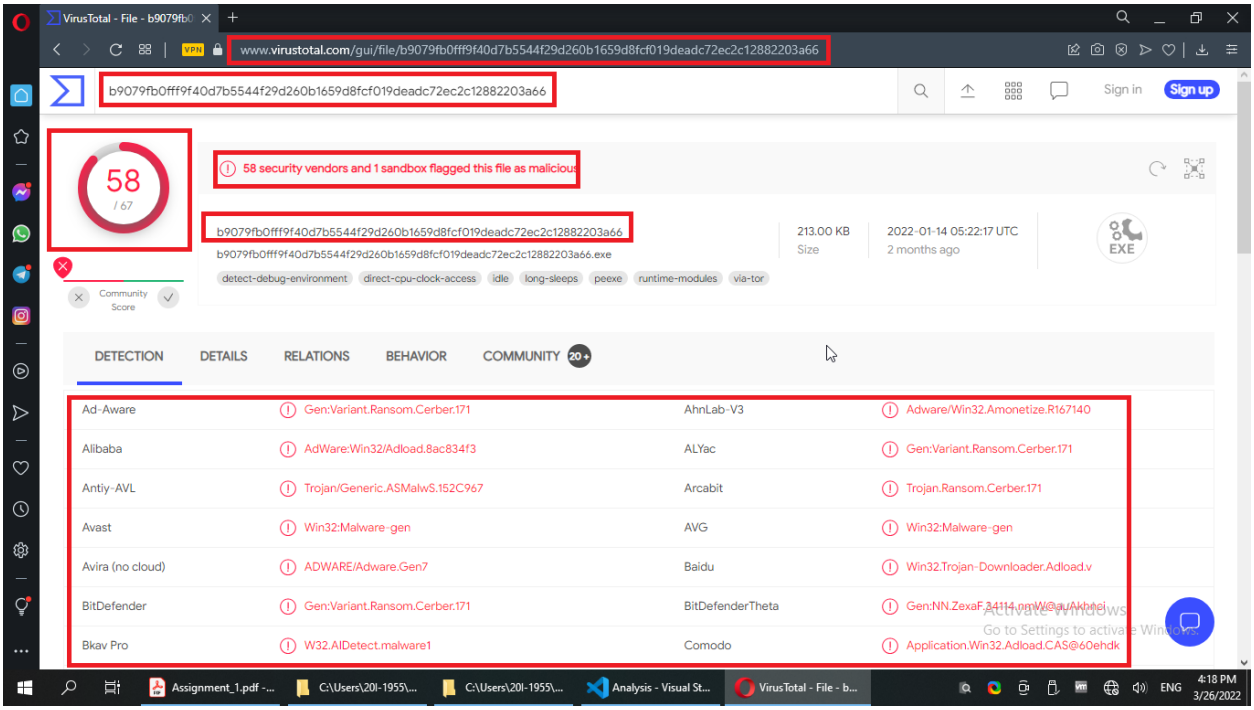

Figure 117: Malware-5 VirusTotal Analysis

Moreover, in the details tab, I find more information about the malware that I already explained in the previous tasks.

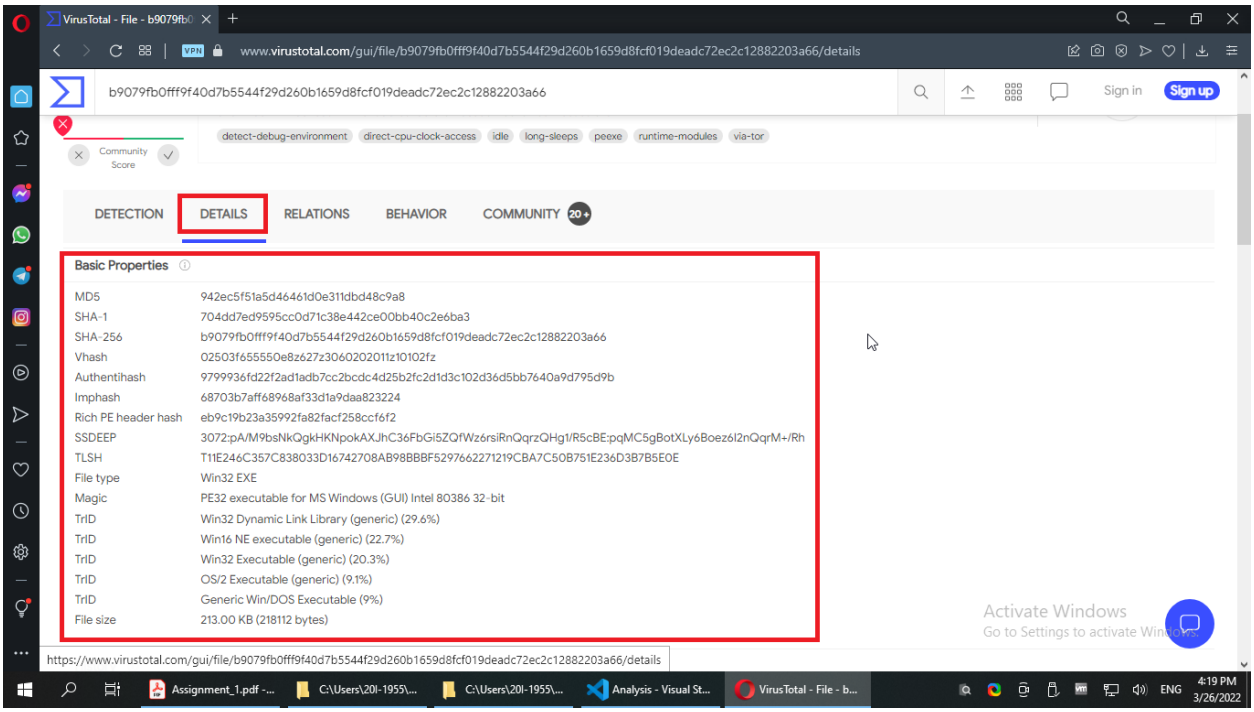

Figure 118: Malware-5 VirusTotal Analysis

Task-8 Analysis of the malware file through Hybrid Analysis

To scan the malware file from Hybrid Analysis I visit the website and upload the malware file. After some time, I found that the file is triggered as malicious by antivirus search engines.

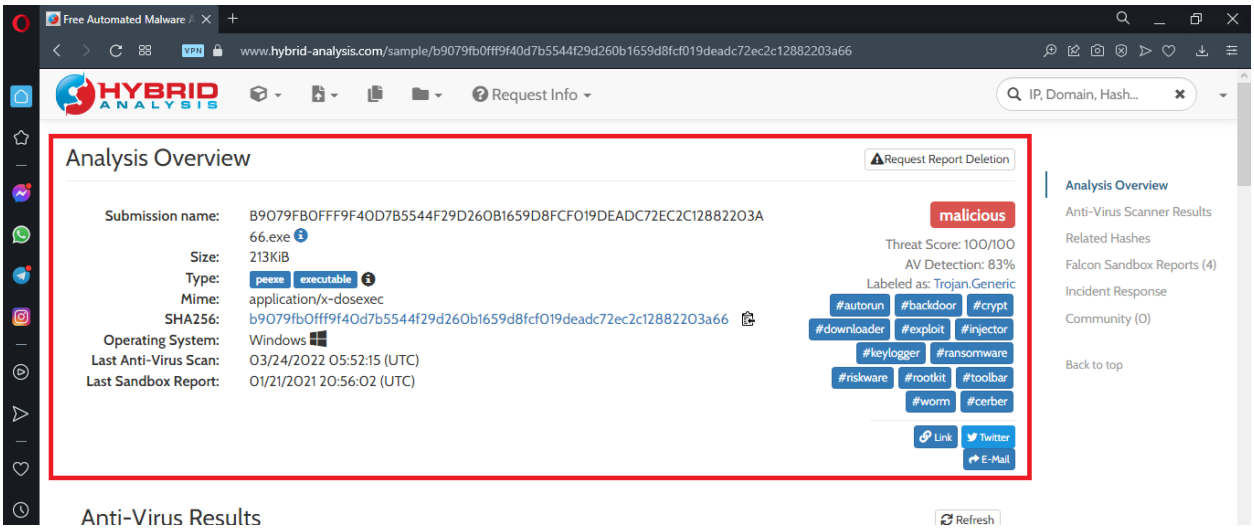

Figure 119: Malware-5 Analysis through Hybrid Analysis

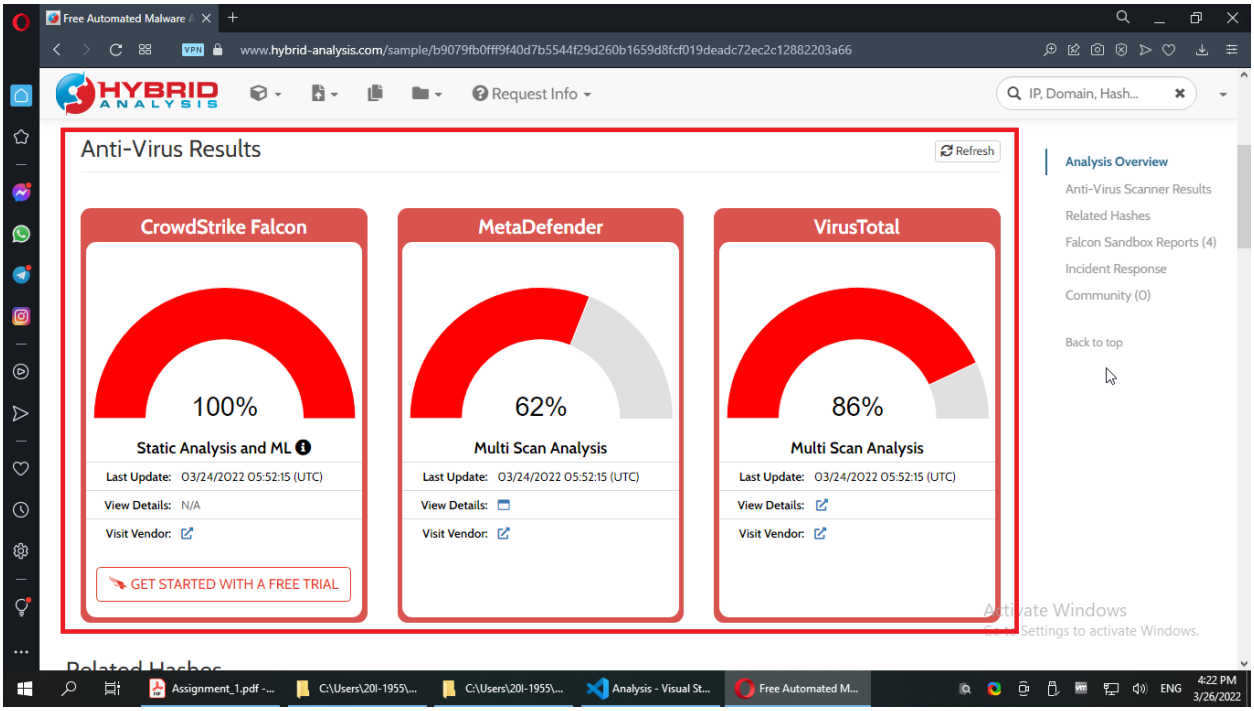

Figure 120: Malware-5 Analysis through Hybrid Analysis

Malware-6 in32Tnega.bXRKZUB

This malware belongs to the family of trojans.

Task-1 What is complete information malware have

In this task, I have to find the basic information of the malware that includes identifying magic byte, signatures, machine info, exe type, and file type.

Task-1.1 Magic Byte

To find the magic byte of the malware there are a bunch of tools available and for this task, I have used three different tools Notepad++, HxD, and CFF Explorer.

Tool-1 Notepad++

To find the magic byte of the malware file I open the file in Notepad++, then from the plugin tab click on "HEX-Editor" and then click on "View in HEX". On viewing the file in HEX-Editor I found that the magic byte of the malware file is "**MZ**". This means that the malware file is executable. Moreover, I also find that this file can't be able to run in DOS mode.

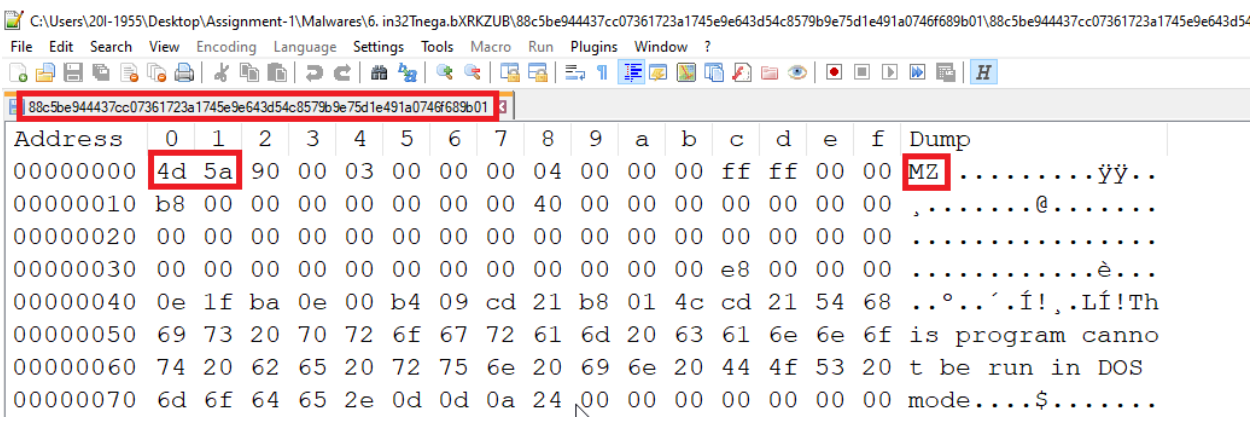

Figure 121: Malware-6 Magic Byte through Notepad++

Tool-2 HxD

To find the magic byte of the malware file I open the file in the HxD and find that the magic byte of the file is "**MZ**" this means that the malware file is executable. Moreover, I also find that this file can't be able to run in DOS mode.

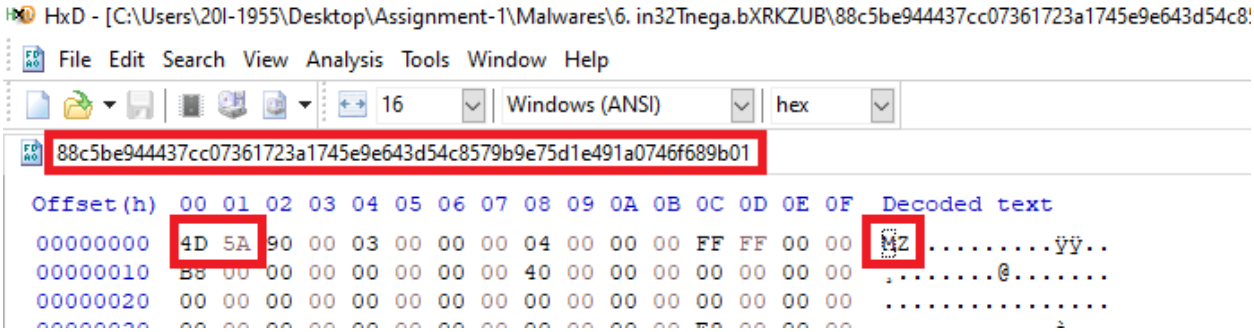

Figure 122: Malware-6 Magic Byte through HxD

Tool-3 CFF Explorer

To find the magic byte of the malware file I open the file in the CFF Explorer, then from the left panel I click on "**DOS Header**". In the "**DOS Header**" I find the magic byte in the "e_magic" column that is "**5A 4D**" which is equivalent to "**4D 5A**" because in CFF Explorer hexadecimal are represented in right to left order. So "**4D = M**" and "**5A = Z**" mean that the malware file is executable.

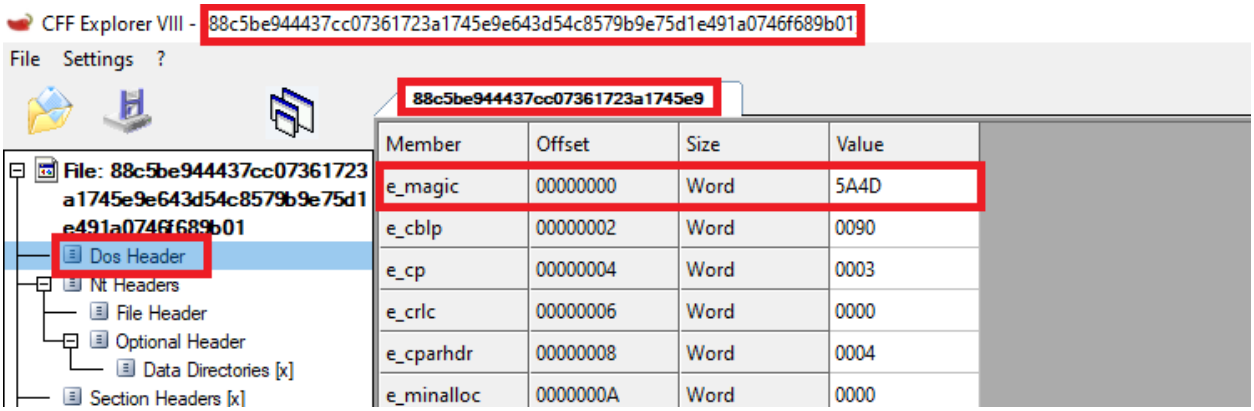

Figure 123: Malware-6 Magic Byte through CFF Explorer

Task-1.2 File Signature

A file signature is used to identify the content of the file. This information is stored in the file's header. To find the file signature of the malware I have used two different tools PE Studio and CFF Explorer.

Tool-1 PE Studio

To find the file signature of the malware file I open the file in the PE Studio, then from the left panel I click on "**file-header**" and then from the right panel I found the column named "**signature**". From there I found that the signature of the file is "**PE**" which means that this file is portable executable.

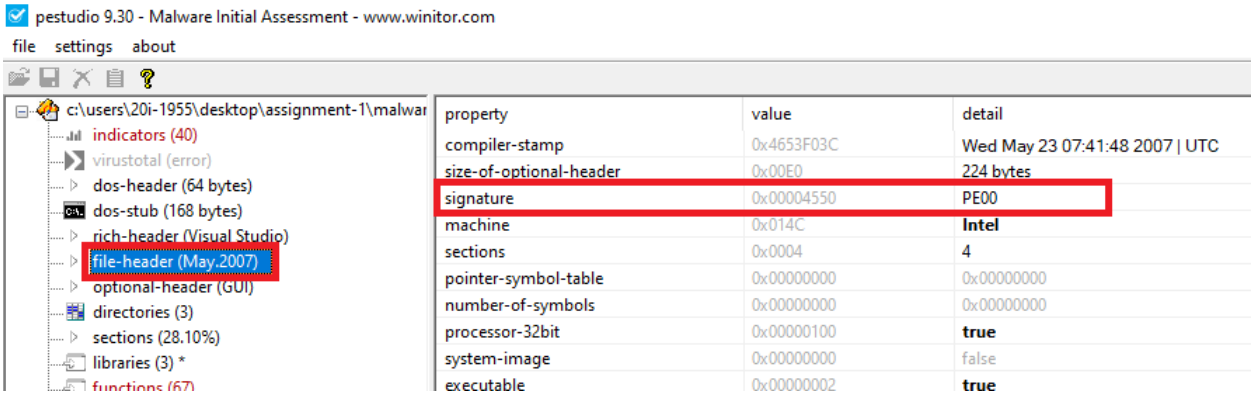

Figure 124: Malware-6 File Signature through PE Studio

Tool-2 CFF Explorer

To find the file signature of the malware file I open the file in the CFF Explorer, then from the left panel I click on "**Nt Headers**" and then from the right panel I found the column named "**signature**". From there I found that the signature of the file is "**45 50**" which is equivalent to "**50 45**" because in CFF Explorer hexadecimal are represented in right to left order. So "**50 = P**" and "**45 = E**" which means that this file is portable executable.

Figure 125: Malware-6 File Signature through CFF Explorer

Task-1.3 Machine Information

To find the machine information of the malware I have used two different tools PE Studio and CFF Explorer.

Tool-1 PE Studio

To find the file signature of the malware file I open the file in the PE Studio, then from the left panel I click on "**file-header**" and then from the right panel I found the column named "**machine**". From there I found that this malware is designed for Intel-based machines. Other than machine information I also find that when the malware is created from the "**compiler-stamp**" column. This malware is created on May 23, 2007, on Sunday at 07:41 AM. Also find that this malware file has 4 sections and the size of the optional header is 224 bytes.

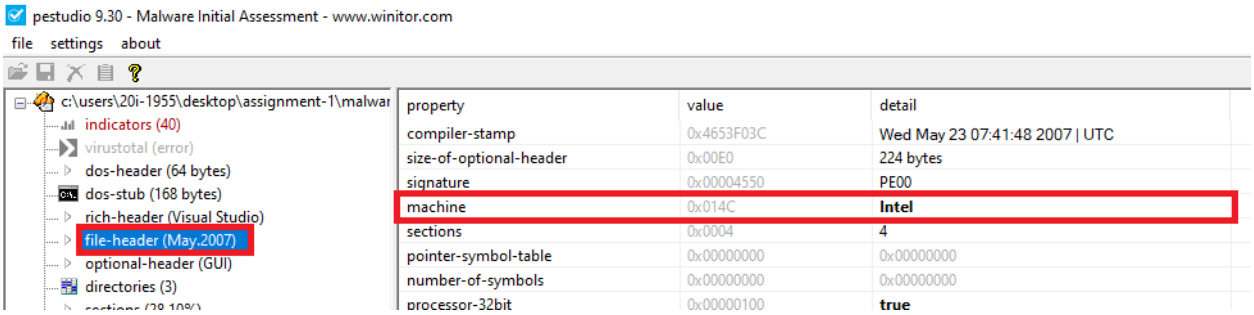

Figure 126: Malware-6 Machine Information through PE Studio

Tool-2 CFF Explorer

To find the file signature of the malware file I open the file in the CFF Explorer, then from the left panel I click on "**file header**" and then from the right panel I found the column named "**Machine**". From there I found that this malware is designed for Intel 386 based machines.

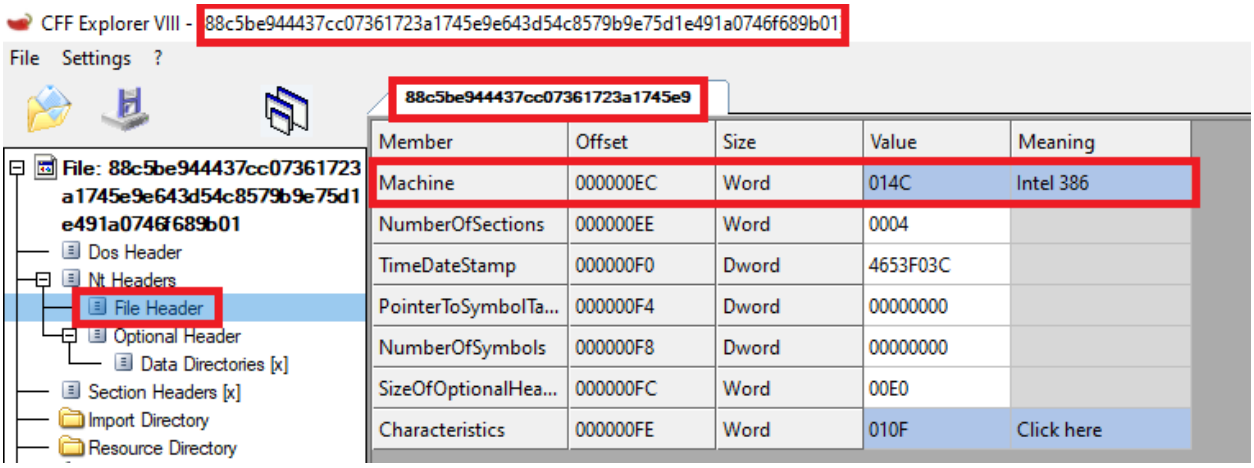

Figure 127: Malware-6 Machine Information through CFF Explorer

Task-1.4 Exe Type

Exe type is used to identify which type of executable it is either 32-bit or 64-bit and to find that I used two different tools PE Studio and CFF Explorer.

Tool-1 PE Studio

To find the exe type of the malware file I open the file in the PE Studio, then from the left panel I click on "**optional-header**" and then from the right panel I found the column named "**magic**". From there I found that the exe type of this malware file is "0x010B" which means that this malware file is a 32-bit portable executable. Other than exe type I also find the entry point of the malware which is ".text".

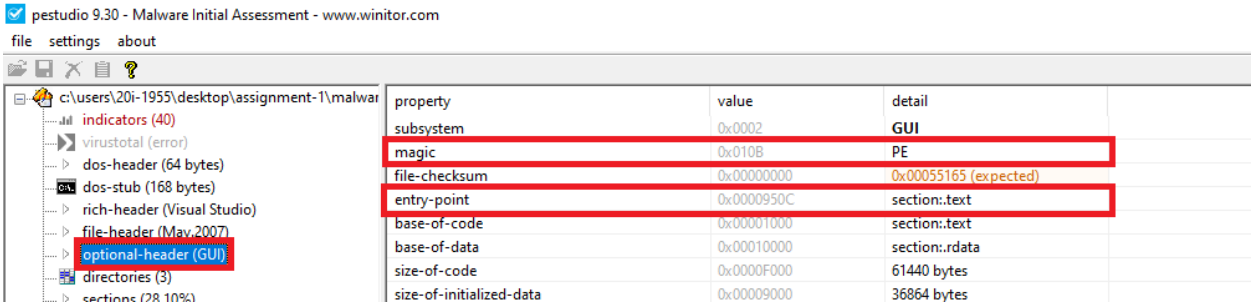

Figure 128: Malware-6 Exe type through PE Studio

Tool-2 CFF Explorer

To find the file signature of the malware file I open the file in the CFF Explorer, then from the left panel I click on "**optional header**" and then from the right panel I found the column named "**Magic**". From there I found that the exe type of this malware file is "010B" which means that this malware file is a 32-bit portable executable. Other than exe type I also find the entry point of the malware which is ".text".

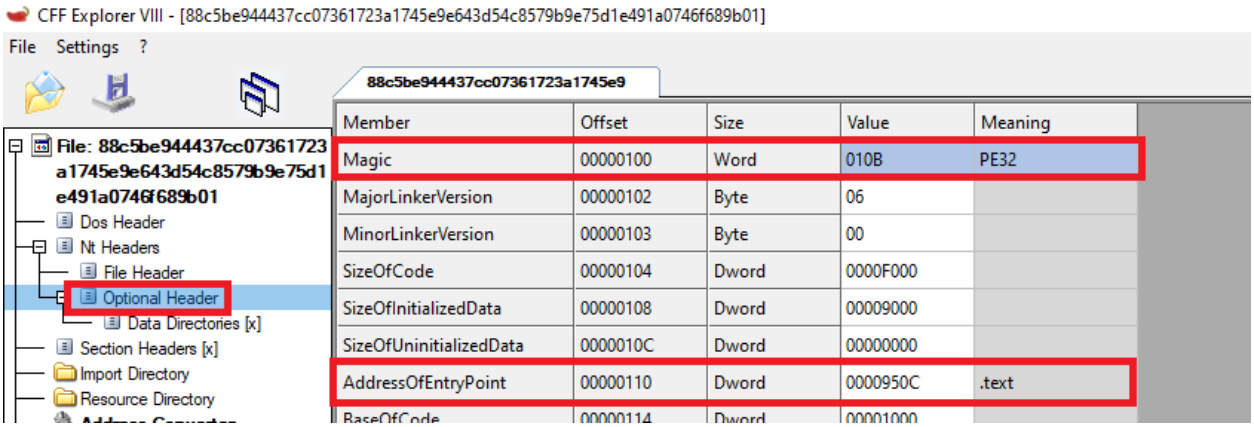

Figure 129: Malware-6 Exe type through CFF Explorer

Task-1.5 File Type

File Type is used to identify which type of file it is. I used two different tools Python Magic Library and TrID Net.

Tool-1 Python Magic Library

To find the file type of the malware using the python magic library I write the following code.

```
import magic

mgicByte = magic.from_file('C:\\Users\\20I-1955\\Desktop\\Assignment-1\\Malwares\\6.in32Tnega.bXRKZUB\\88c5be944437cc07361723a1745e9e643d54c8579b9e75d1e491a0746f689b01\\88c5be944437cc07361723a1745e9e643d54c8579b9e75d1e491a0746f689b01')

print(mgicByte)
```

After running the code, I found that the file type of malware is "PE32 executable (GUI) Intel 80386, for MS Windows".

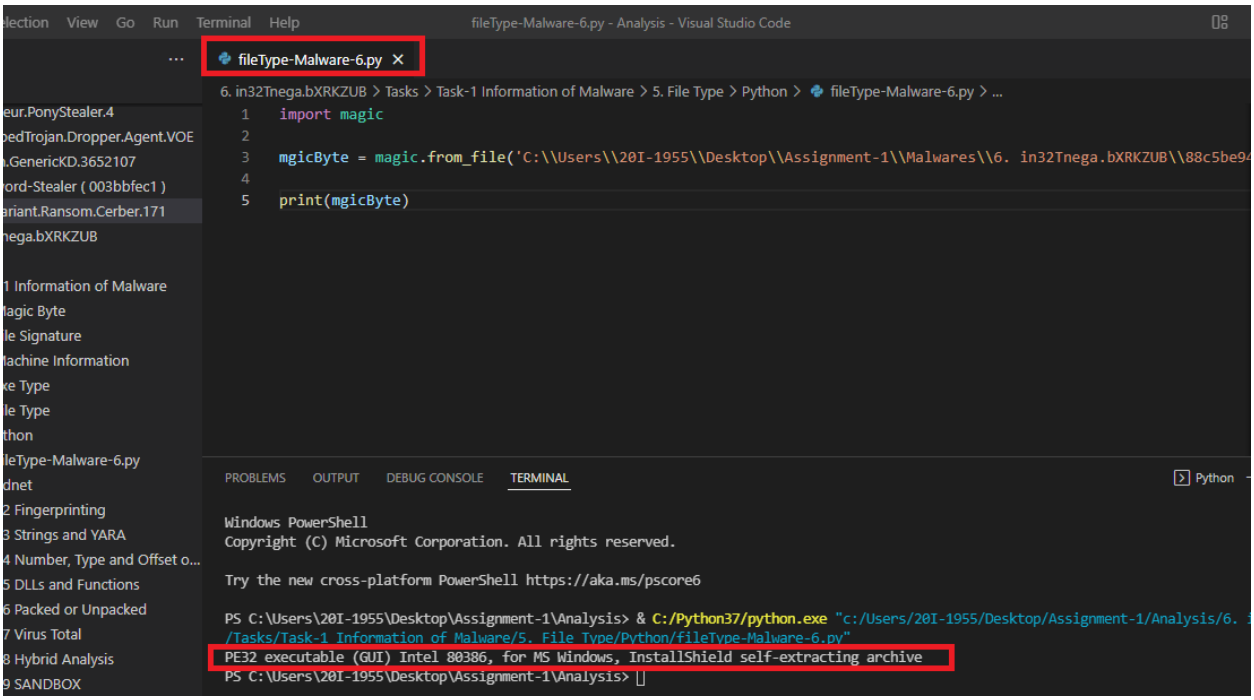

Figure 130: Malware-6 File Type through Python Magic Library

Tool-2 TrID Net

To find the file type of the malware using TrID Net I open the file in TrID Net and find that the malware file is “Win32 Executable MS Visual C++” as it has the “38.8%” match.

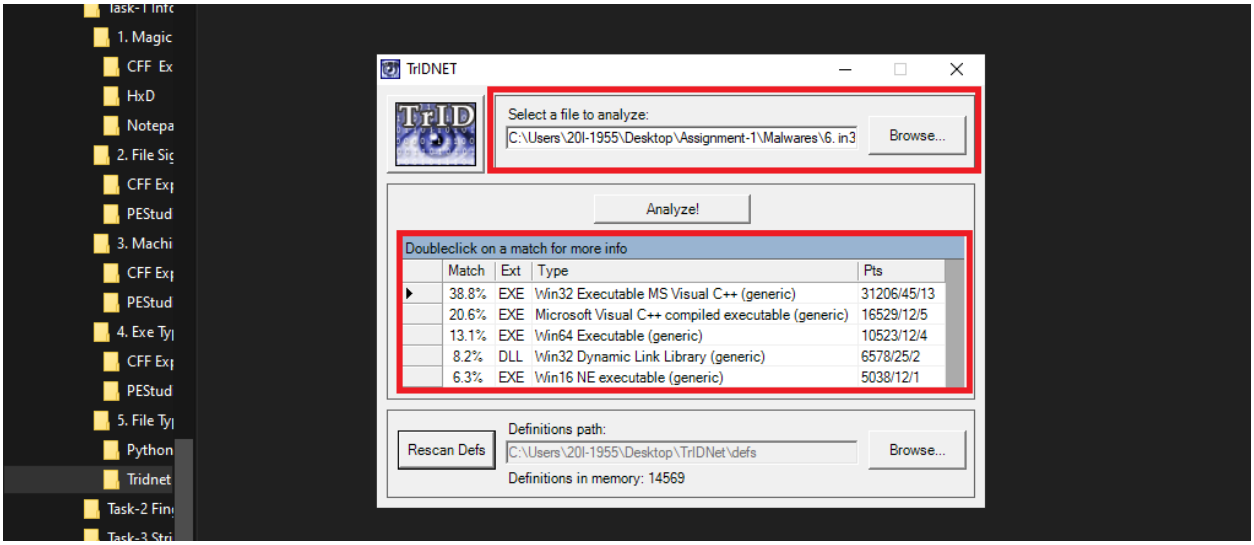

Figure 131: Malware-6 File Type through TrID Net

Task-2 What is the fingerprinting information of the malware file?

To find the hashes of the malware file I have used two different tools HashCalc and Python hashlib library.

Tool-1 HashCalc

To find the hash function of the file in HashCalc I open the file in HashCalc and find the hash of the file.

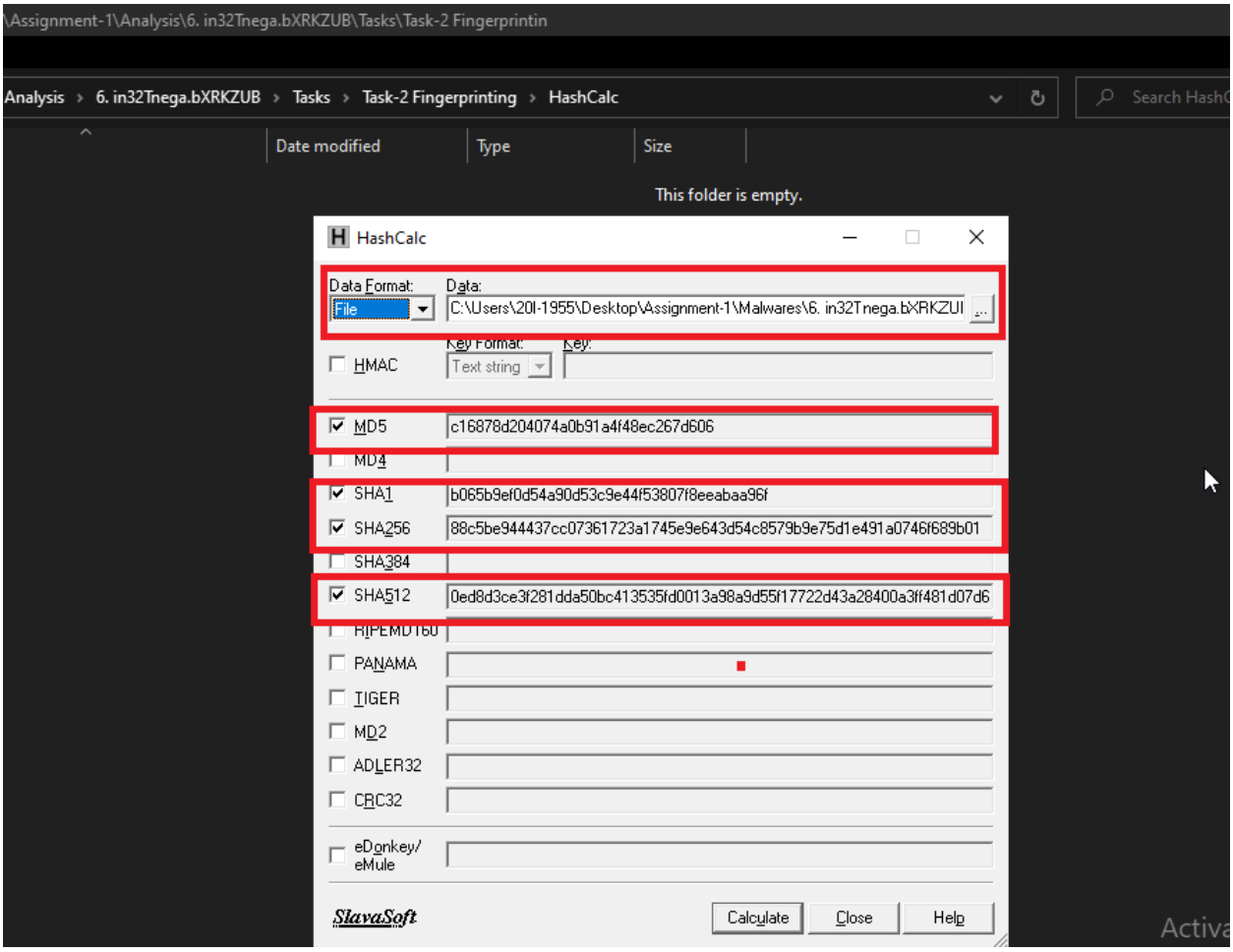

Figure 132: Malware-6 Fingerprinting through HashCalc

Tool-2 Python Hashlib Library

To find the hash of the malware using the python Hashlib library I write the following code.

```
import hashlib

hashMd5 = hashlib.md5(open("C:\\Users\\20I-1955\\Desktop\\Assignment-
1\\Malwares\\6.
in32Tnega.bXRKZUB\\88c5be944437cc07361723a1745e9e643d54c8579b9e75d1e491a0746f68
9b01\\88c5be944437cc07361723a1745e9e643d54c8579b9e75d1e491a0746f689b01",
"rb").read()).hexdigest()
```

```
hashSha1 = hashlib.sha1(open("C:\\Users\\20I-1955\\Desktop\\Assignment-
1\\Malwares\\6.
in32Tnega.bXRKZUB\\88c5be944437cc07361723a1745e9e643d54c8579b9e75d1e491a0746f68
9b01\\88c5be944437cc07361723a1745e9e643d54c8579b9e75d1e491a0746f689b01",
"rb").read()).hexdigest()
hashSha256 = hashlib.sha256(open("C:\\Users\\20I-1955\\Desktop\\Assignment-
1\\Malwares\\6.
in32Tnega.bXRKZUB\\88c5be944437cc07361723a1745e9e643d54c8579b9e75d1e491a0746f68
9b01\\88c5be944437cc07361723a1745e9e643d54c8579b9e75d1e491a0746f689b01",
"rb").read()).hexdigest()
hashSha512 = hashlib.sha512(open("C:\\Users\\20I-1955\\Desktop\\Assignment-
1\\Malwares\\6.
in32Tnega.bXRKZUB\\88c5be944437cc07361723a1745e9e643d54c8579b9e75d1e491a0746f68
9b01\\88c5be944437cc07361723a1745e9e643d54c8579b9e75d1e491a0746f689b01",
"rb").read()).hexdigest()

print("MD5:    ",hashMd5)
print("SHA1:   ",hashSha1)
print("SHA256: ",hashSha256)
print("SHA512: ",hashSha512)
```

After running the code, I found that the hashes of the malware file

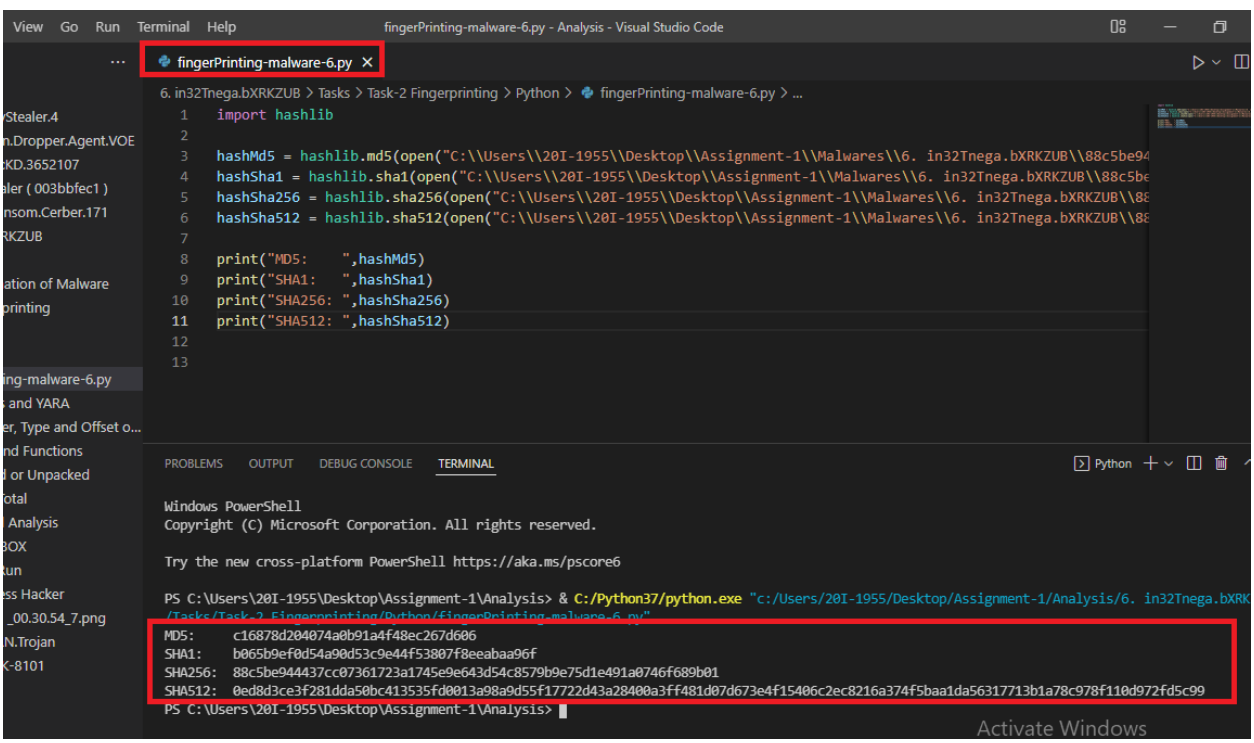

Figure 133: Malware-6 Fingerprinting through Python Hashlib Library

Task-3 What type of malicious strings are in the Malware file?

To extract the strings from the malware file I use the tool called "**Strings**" and then to scan the malware according to the extracted strings I write the YARA rule.

Tool-1 Strings

To extract the strings from the malware file and save them in the text file I run the following command

```
strings malware file path > text file name
```

After running the command all the strings are extracted from the malware file and stored in the text file.

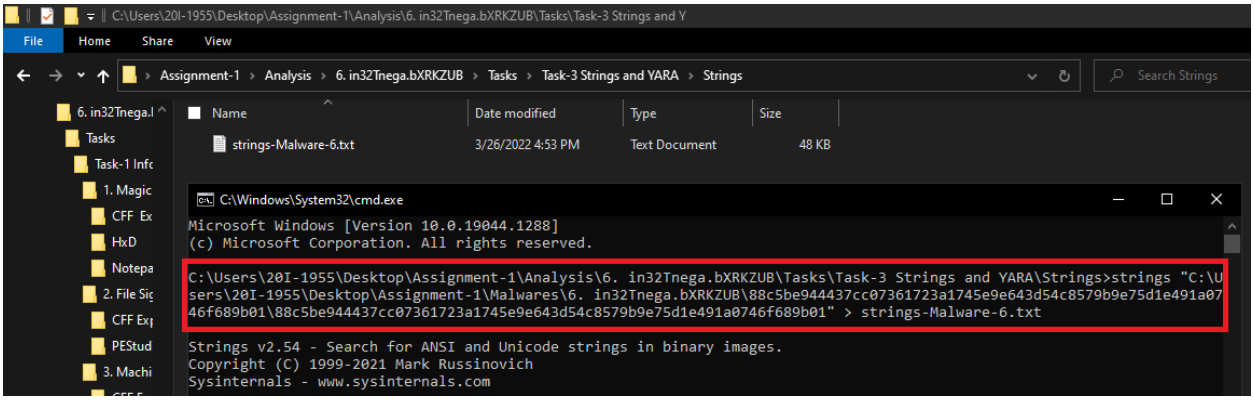

Figure 134: Malware-6 Strings Extraction through Strings Tool

After that, I open the text file to check the strings that were extracted from the malware and found some malicious strings.

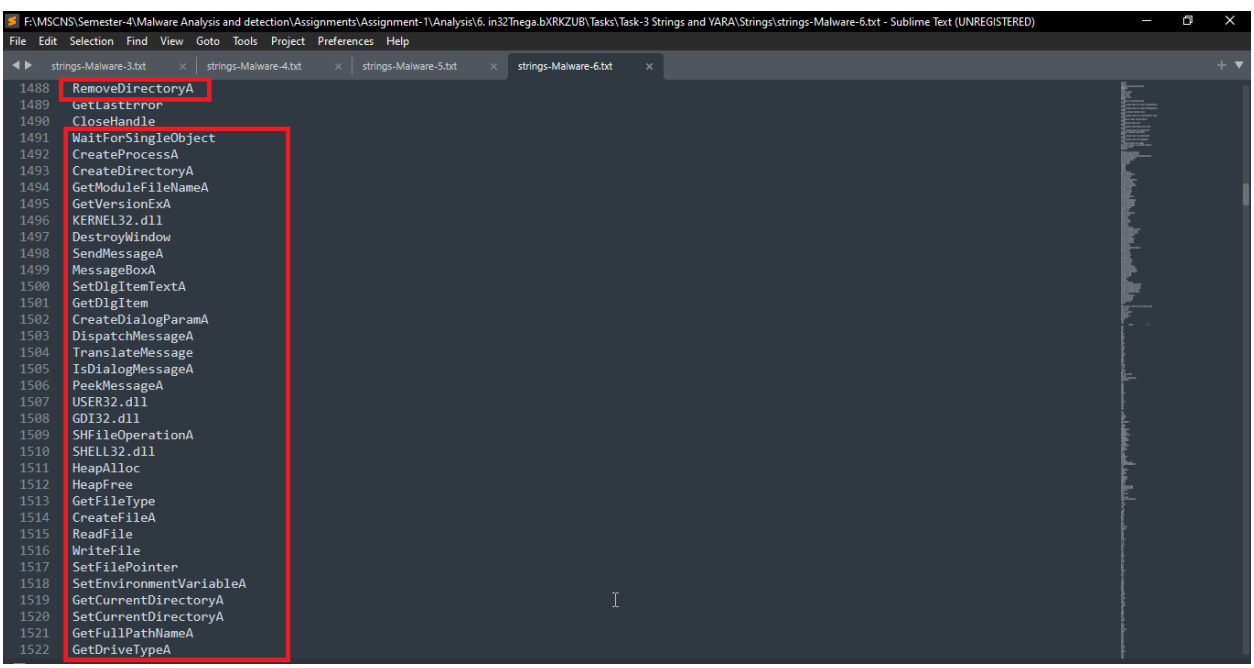

Figure 135: Malware-6 Strings Extracted through Strings Tool

Tool-2 YARA Rule

After extracting the string my next task is to write the YARA rule that scans the file and triggers the warning if it finds the malicious string in the malware file. For that, I write the following rule

```
rule malware_in32Tnega_bXRKZUB{
    meta:
        malwareName = "in32Tnega.bXRKZUB"
        ruleAuthor = "20I-1955_Muhammad Osama Khalid"
    strings:
        $str1 = "RegOpenKey" nocase
        $str2 = "RegQueryValue" nocase
        $str3 = "RegCloseKey" nocase
        $str4 = "kernel32" nocase
        $str5 = "shell32" nocase
    condition:
        any of them
}
```

Note: These strings are extracted from analyzing all the malware samples that are given for this assignment.

After that, I run the rule and find that the rule that I write works successfully as it detects the malicious string extracted above in the malware file.

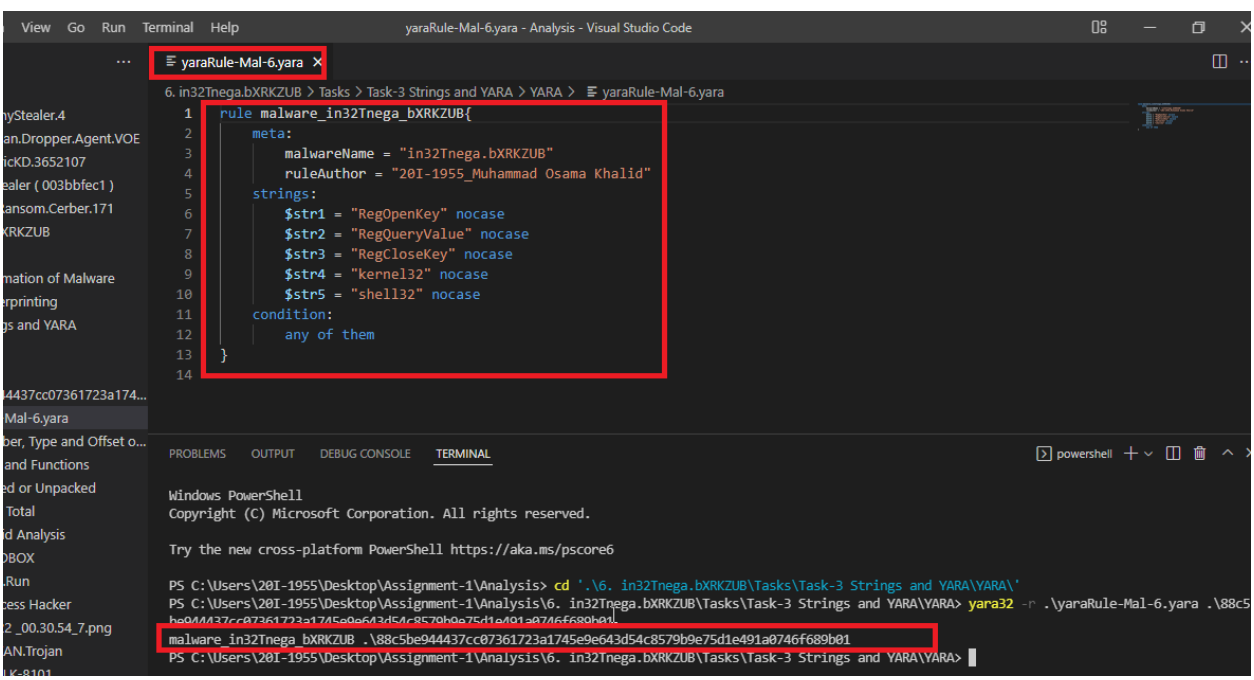

Figure 136: Malware-6 Malware detection through YARA Rule

Task-4 Number, type, and offset of different sections of Malware?

Sections contain the important information about the file as it describes how the executable is made. To find the number, type, and offset of different sections of malware files I use three different tools PE Studio and Exeinfo.

Tool-1 PE Studio

To find the number, type, and offset of different sections of the malware file I open the file in the PE Studio, then from the left panel first I click on "**file-header**" and then from the right panel I found the column named "**sections**". From there I found that this malware file has three sections and to find then what they are I click on "**sections**" in the left panel. After that from the right panel, I found the sections that this malware file has.

No. of Section	Type	Offset
1	.text	0x00401000
2	.rdata	0x00410000
3	.data	0x00411000
4	.rsrc	0x00418000

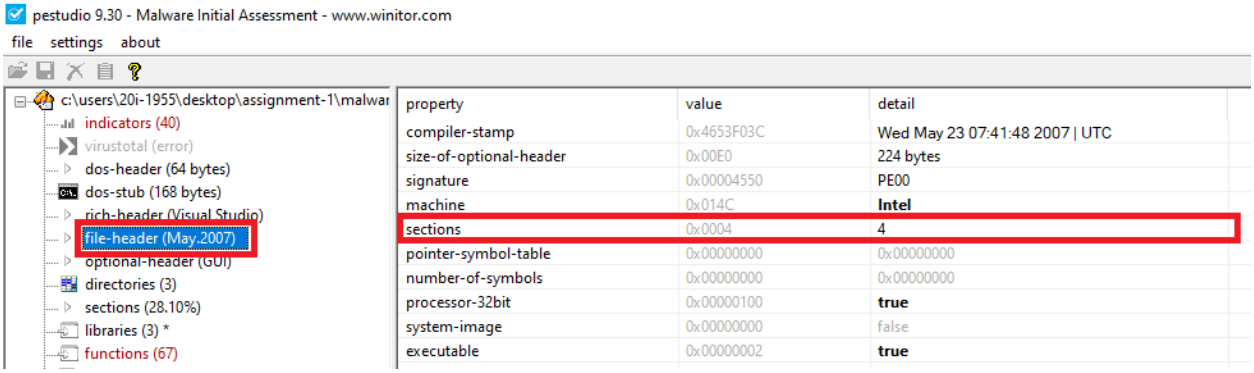

Figure 137: Malware-6 Information of Sections through PE Studio

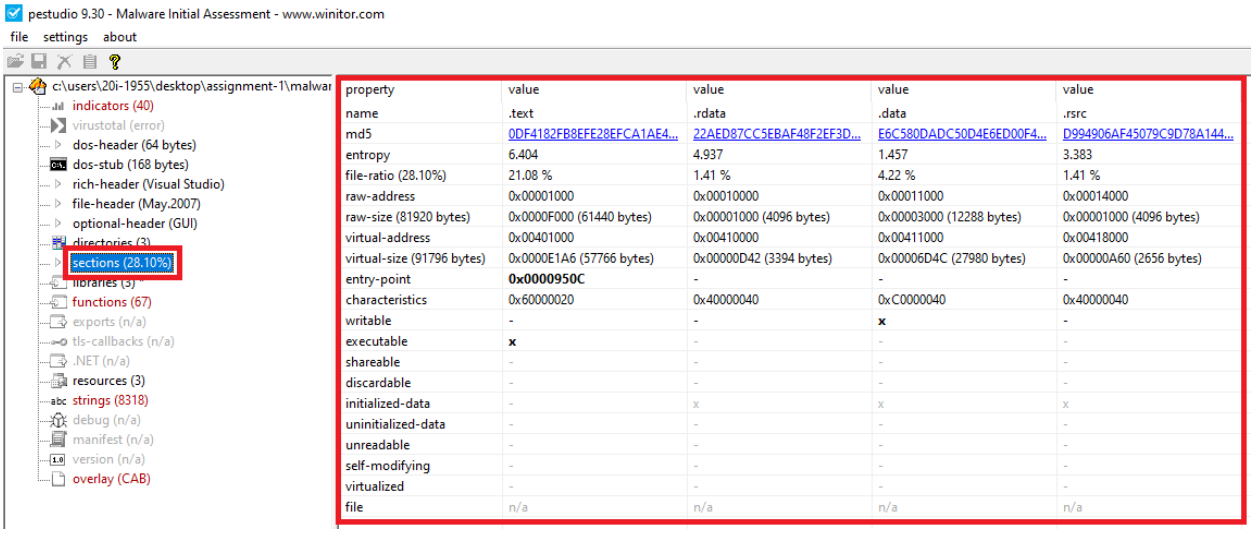

Figure 138: Malware-6 Information of Sections through PE Studio

Task-5 Which DLLs and their number of functions are used by Malware?

To find what DLLs and functions this malware used I used the tool PE Studio.

Tool-1 PE Studio

To find the DLLs and their number of functions used by malware files I open the file in the PE Studio, then from the left panel first I click on "**libraries**" and then from the right pane I found the list of DLLs and their functions used by this malware.

No. of DLL	Name	No. of functions
1	kernel32.dll	56
2	user32.dll	10
3	shell32.dll	1

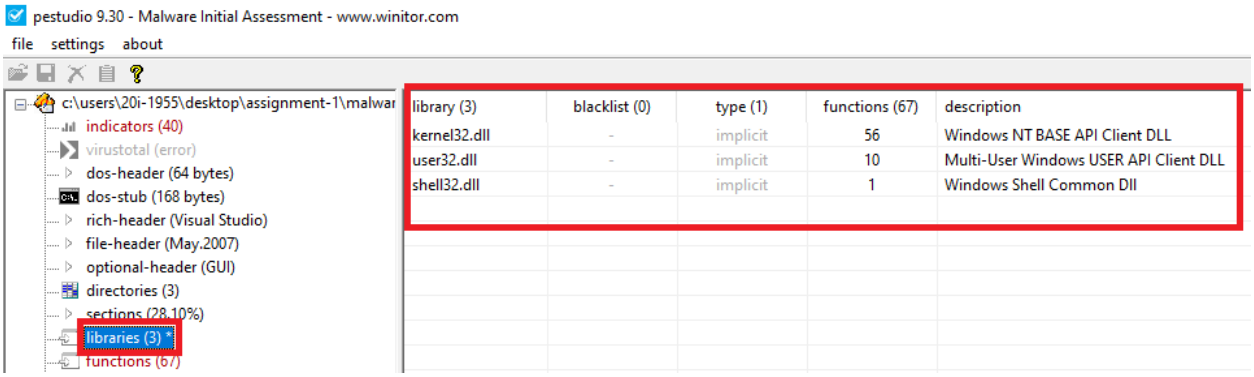

Figure 139: Malware-6 Information of DLLs and functions through PE Studio

Task-6 What is packing information if Malware is packed?

To check whether the malware is packed or not I use the tool Exeinfo.

Tool-1 Exeinfo

To find whether the malware file is packed or not I open the file in Exeinfo and at the bottom I found that the malware is not packed.

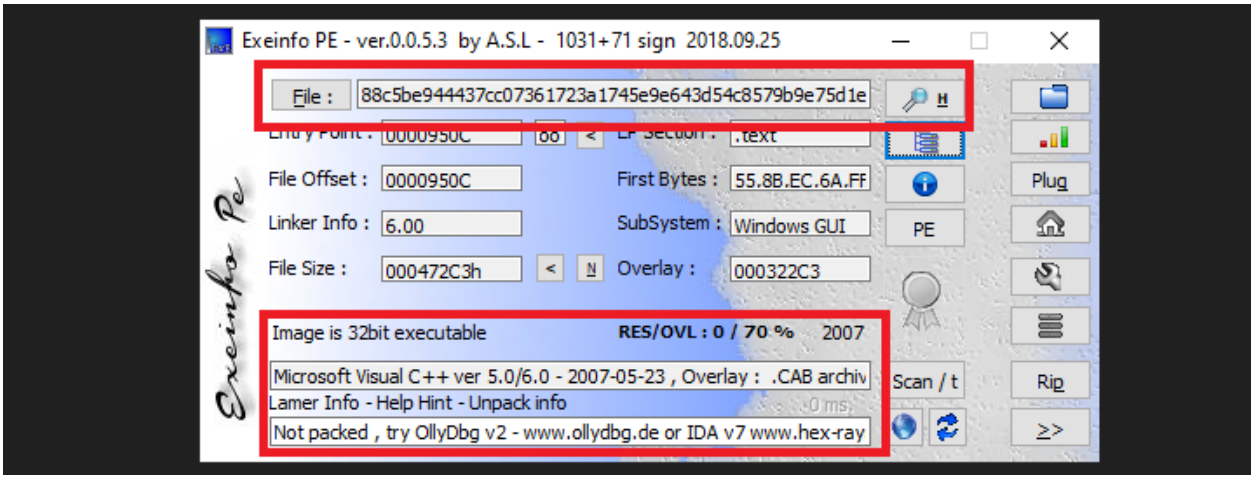

Figure 140: Malware-6 Packed or unpacked malware information through Exeinfo

Task-7 VirusTotal analysis of the malware file

To scan the malware file from the virus total I visit the website and upload the malware file. After some time, I found that surprisingly only 6 antivirus engines marked this malware file as malicious. To my surprise, I find that famous antivirus search engines failed to detect this malware.

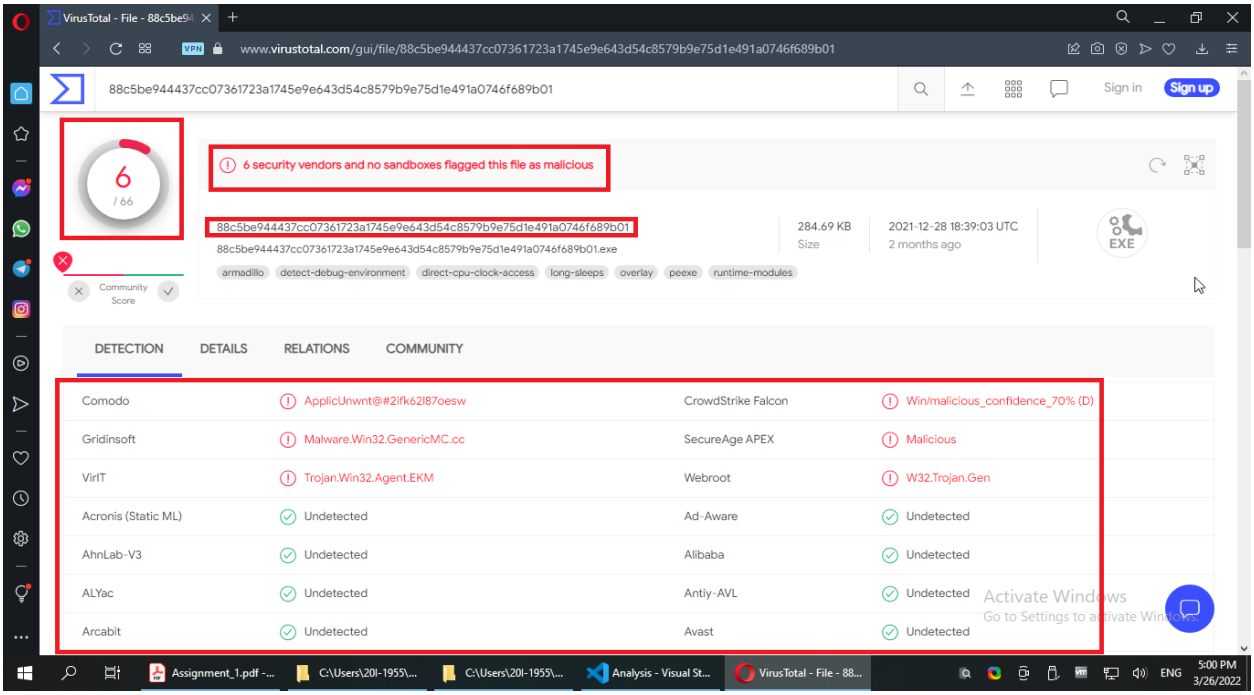

Figure 141: Malware-6 VirusTotal Analysis

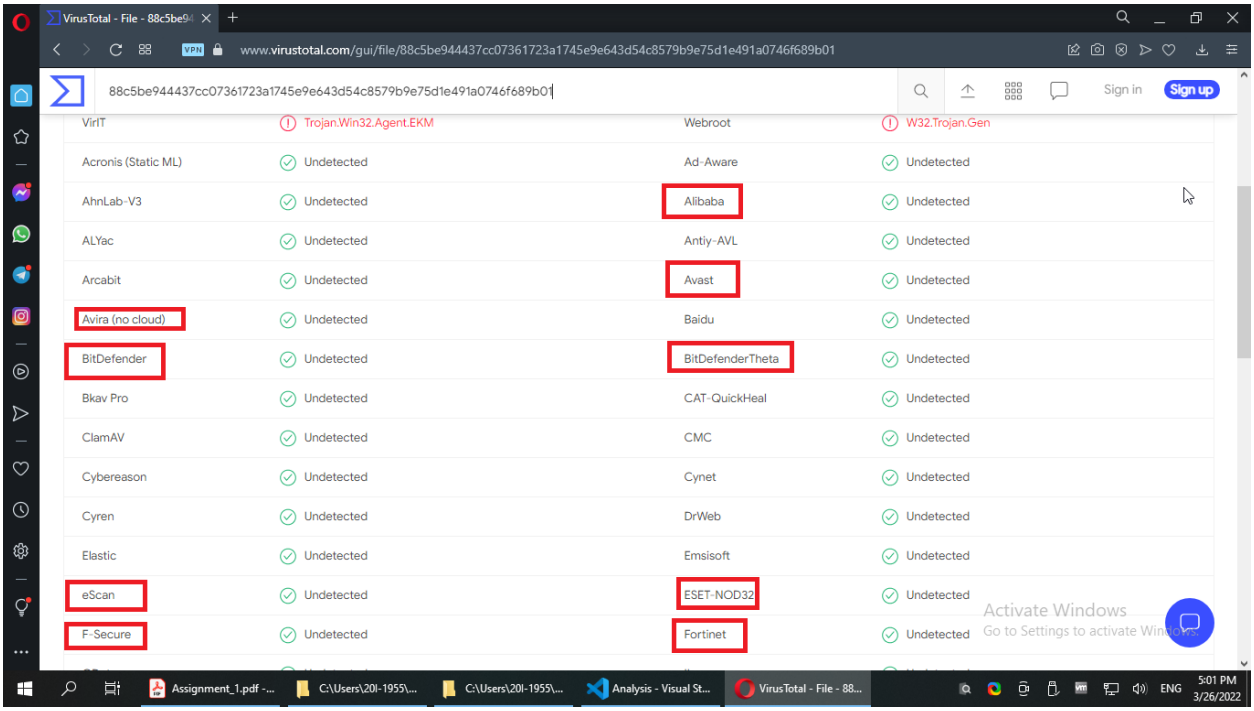

Figure 142: Malware-6 VirusTotal Analysis

Moreover, in the details tab, I find more information about the malware that I already explained in the previous tasks.

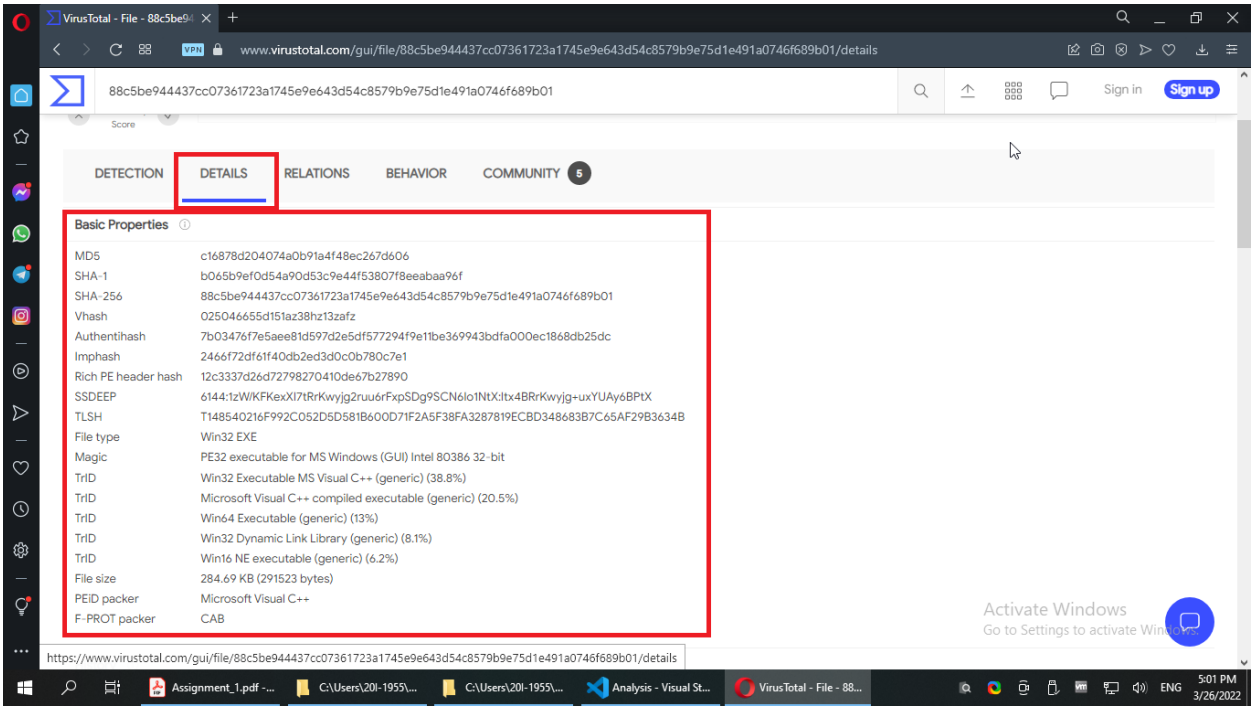

Figure 143: Malware-6 VirusTotal Analysis

Task-8 Analysis of the malware file through Hybrid Analysis

To scan the malware file from Hybrid Analysis I visit the website and upload the malware file. After some time, I found that the file is triggered as malicious by two antivirus search engines, and the VirusTotal search engine marked this malware file as clean.

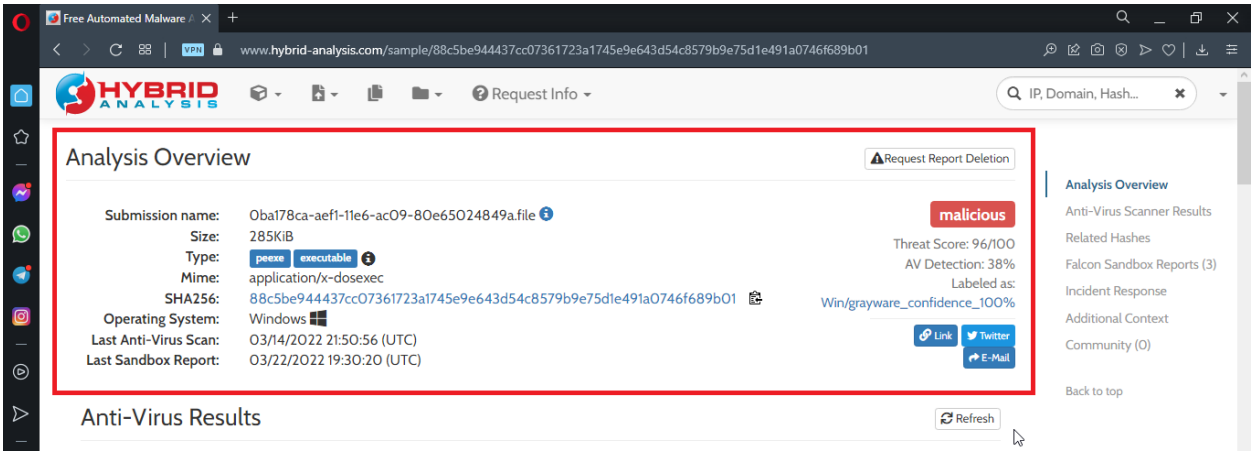

Figure 144: Malware-6 Analysis through Hybrid Analysis

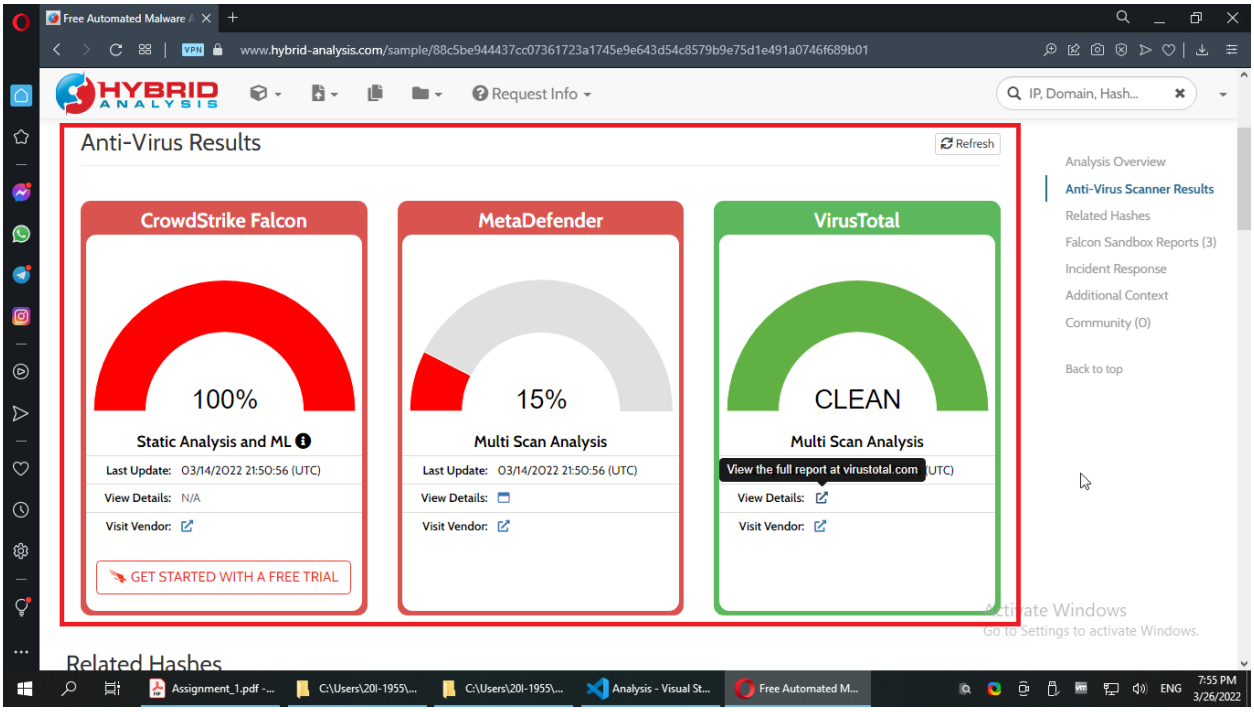

Figure 145: Malware-6 Analysis through Hybrid Analysis

Malware-7 W32.SecretKAN.Trojan

This malware belongs to the family of trojans.

Task-1 What is complete information malware have

In this task, I have to find the basic information of the malware that includes identifying magic byte, signatures, machine info, exe type, and file type.

Task-1.1 Magic Byte

To find the magic byte of the malware there are a bunch of tools available and for this task, I have used three different tools Notepad++, HxD, and CFF Explorer.

Tool-1 Notepad++

To find the magic byte of the malware file I open the file in Notepad++, then from the plugin tab click on "HEX-Editor" and then click on "View in HEX". On viewing the file in HEX-Editor I found that the magic byte of the malware file is "**MZ**". This means that the malware file is executable. Moreover, I also find that this file can't be able to run in DOS mode.

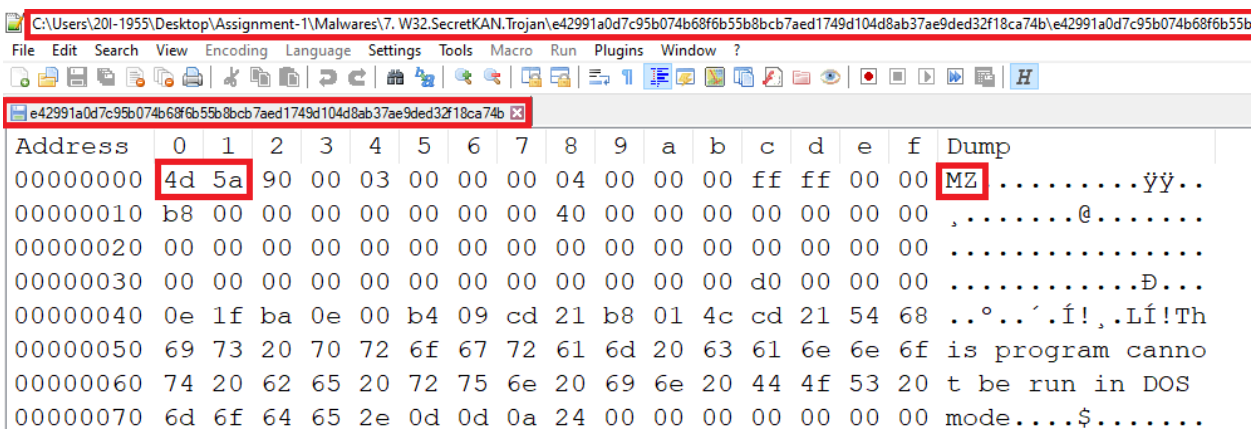

Figure 146: Malware-7 Magic Byte through Notepad++

Tool-2 HxD

To find the magic byte of the malware file I open the file in the HxD and find that the magic byte of the file is "**MZ**" which means that the malware file is executable. Moreover, I also find that this file can't be able to run in DOS mode.

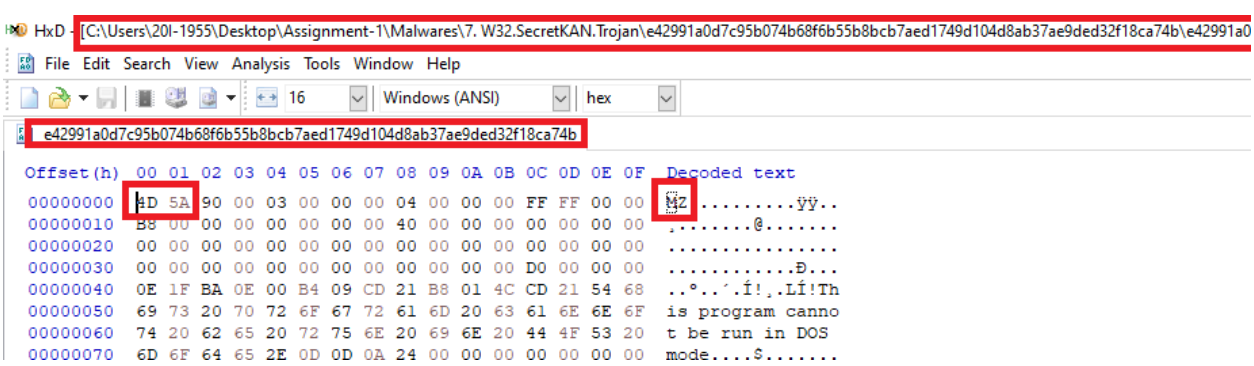

Figure 147: Malware-7 Magic Byte through HxD

Tool-3 CFF Explorer

To find the magic byte of the malware file I open the file in the CFF Explorer, then from the left panel I click on "**DOS Header**". In the "**DOS Header**" I find the magic byte in the "e_magic" column that is "**5A 4D**" which is equivalent to "**4D 5A**" because in CFF Explorer hexadecimal are represented in right to left order. So "**4D = M**" and "**5A = Z**" mean that the malware file is executable.

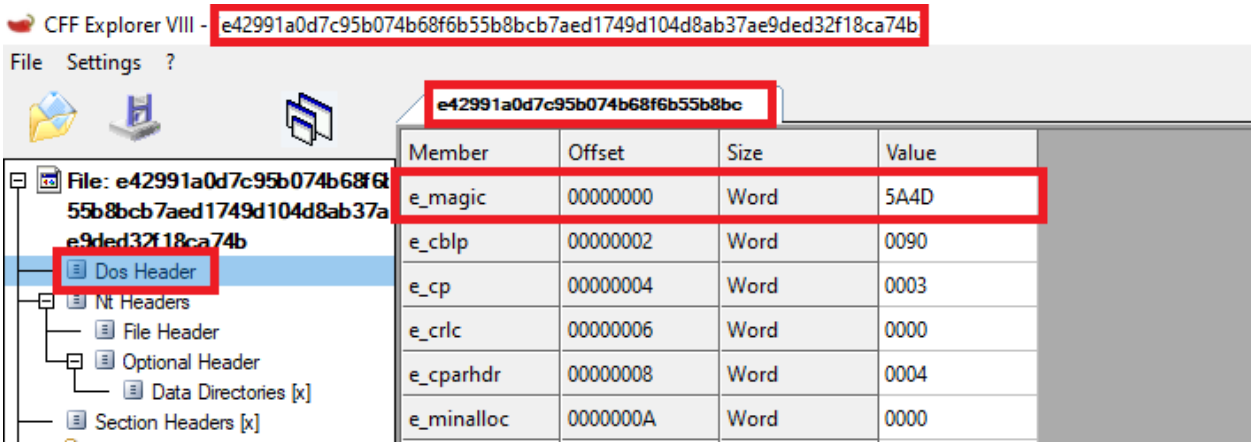

Figure 148: Malware-7 Magic Byte through CFF Explorer

Task-1.2 File Signature

A file signature is used to identify the content of the file. This information is stored in the file's header. To find the file signature of the malware I have used two different tools PE Studio and CFF Explorer.

Tool-1 PE Studio

To find the file signature of the malware file I open the file in the PE Studio, then from the left panel I click on "**file-header**" and then from the right panel I found the column named "**signature**". From there I found that the signature of the file is "**PE**" which means that this file is portable executable.

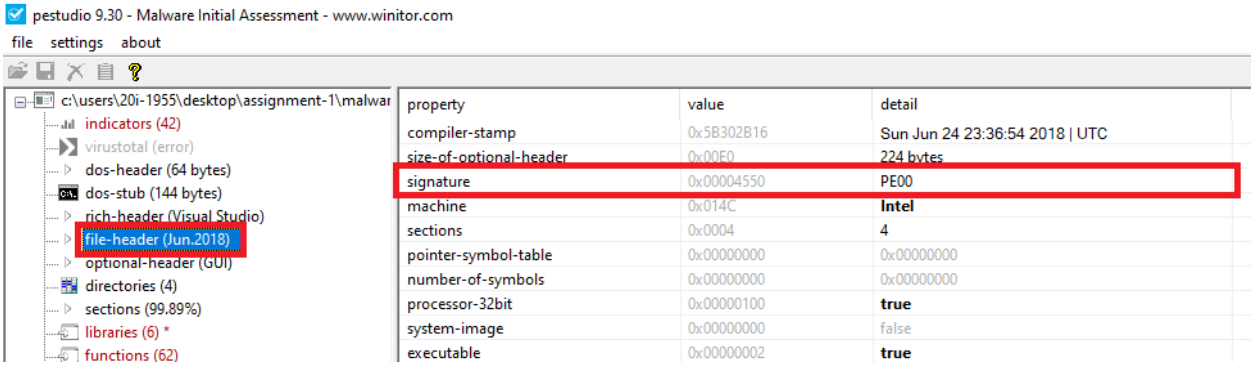

Figure 149: Malware-7 File Signature through PE Studio

Tool-2 CFF Explorer

To find the file signature of the malware file I open the file in the CFF Explorer, then from the left panel I click on "**Nt Headers**" and then from the right panel I found the column named "**signature**". From there I found that the signature of the file is "**45 50**" which is equivalent to "**50 45**" because in CFF Explorer hexadecimal are represented in right to left order. So "**50 = P**" and "**45 = E**" which means that this file is portable executable.

Figure 150: Malware-7 File Signature through CFF Explorer

Task-1.3 Machine Information

To find the machine information of the malware I have used two different tools PE Studio and CFF Explorer.

Tool-1 PE Studio

To find the file signature of the malware file I open the file in the PE Studio, then from the left panel I click on "**file-header**" and then from the right panel I found the column named "**machine**". From there I found that this malware is designed for Intel-based machines. Other than machine information I also find that when the malware is created from the "**compiler-stamp**" column. This malware is created on Jun 24, 2018, on Sunday at 11:36 PM. Also find that this malware file has 4 sections and the size of the optional header is 224 bytes.

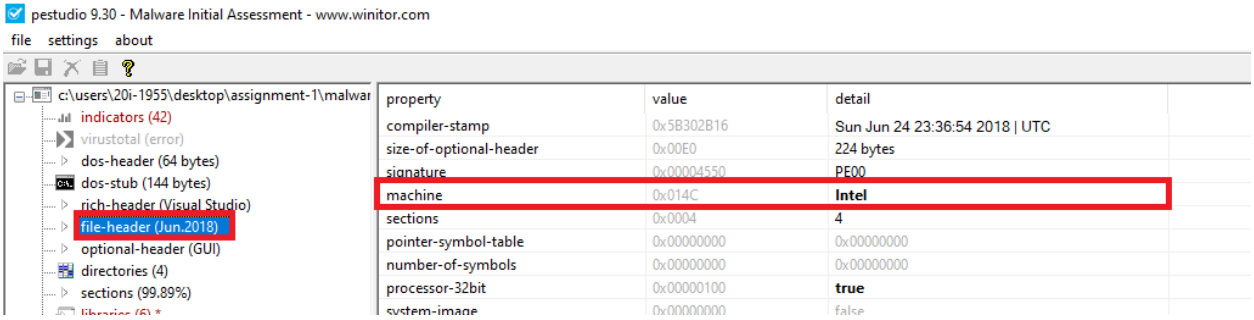

Figure 151: Malware-7 Machine Information through PE Studio

Tool-2 CFF Explorer

To find the file signature of the malware file I open the file in the CFF Explorer, then from the left panel I click on "**file header**" and then from the right panel I found the column named "**Machine**". From there I found that this malware is designed for Intel 386 based machines.

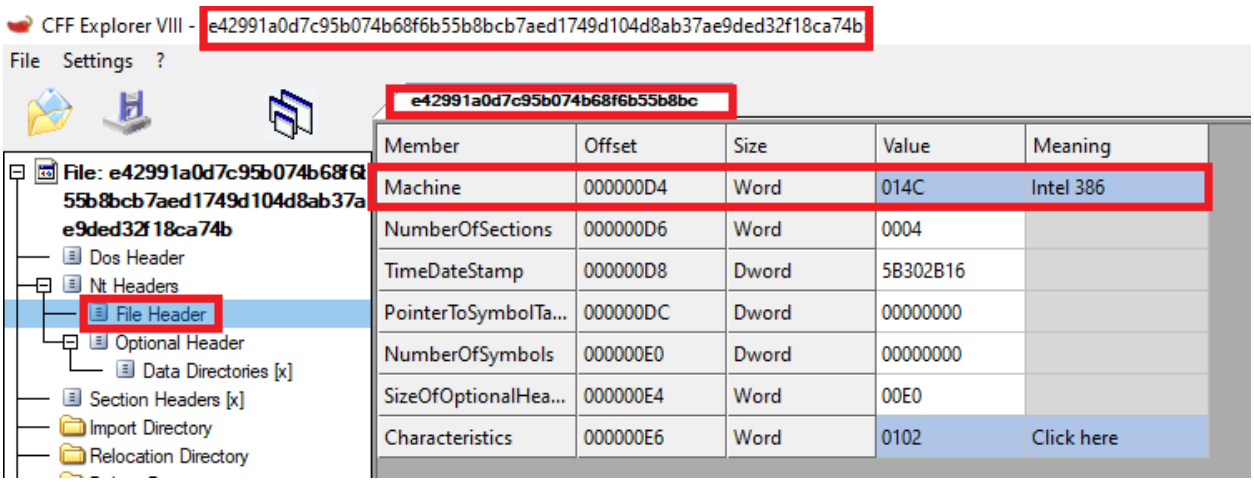

Figure 152: Malware-7 Machine Information through CFF Explorer

Task-1.4 Exe Type

Exe type is used to identify which type of executable it is either 32-bit or 64-bit and to find that I used two different tools PE Studio and CFF Explorer.

Tool-1 PE Studio

To find the exe type of the malware file I open the file in the PE Studio, then from the left panel I click on "**optional-header**" and then from the right panel I found the column named "**magic**". From there I found that the exe type of this malware file is "0x010B" which means that this malware file is a 32-bit portable executable. Other than exe type I also find the entry point of the malware which is ".text".

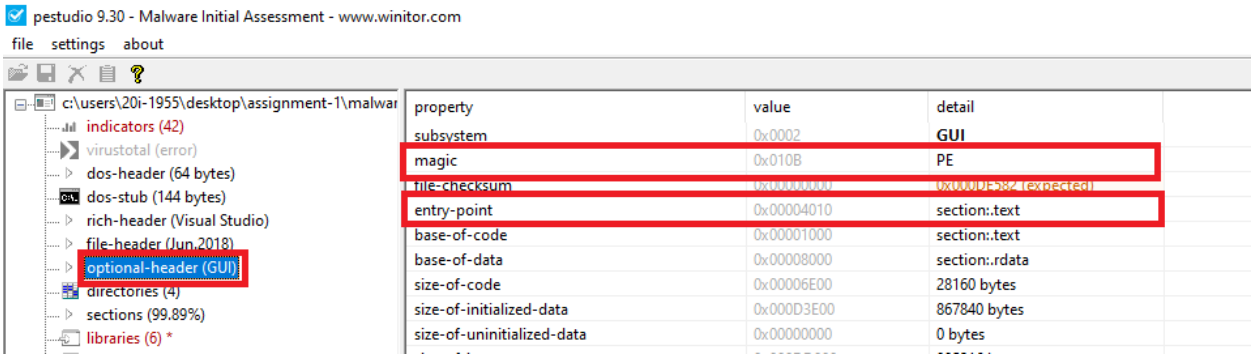

Figure 153: Malware-7 Exe type through PE Studio

Tool-2 CFF Explorer

To find the file signature of the malware file I open the file in the CFF Explorer, then from the left panel I click on "**optional header**" and then from the right panel I found the column named "**Magic**". From there I found that the exe type of this malware file is "010B" which means that this malware file is a 32-bit portable executable. Other than exe type I also find the entry point of the malware which is ".text".

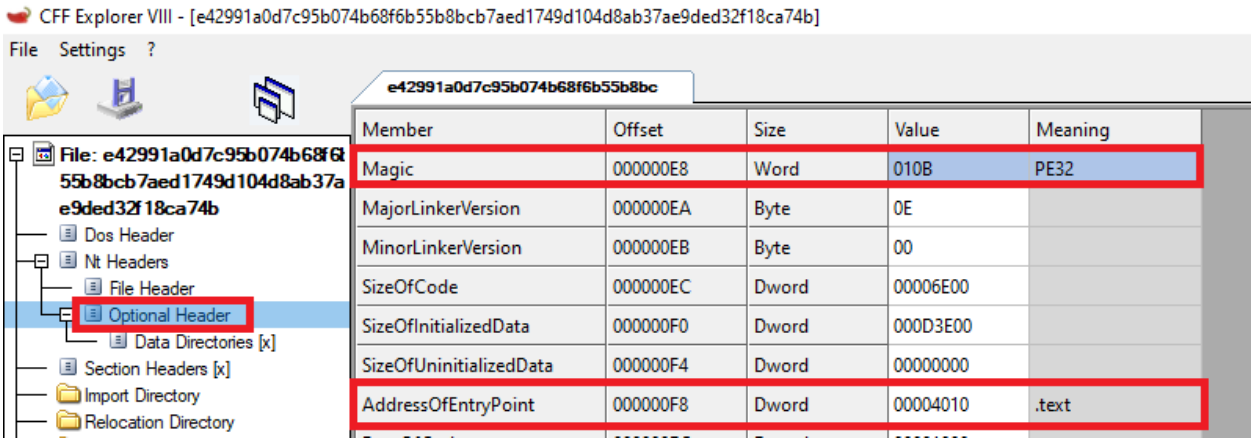

Figure 154: Malware-7 Exe type through CFF Explorer

Task-1.5 File Type

File Type is used to identify which type of file it is. I used two different tools Python Magic Library and TrID Net.

Tool-1 Python Magic Library

To find the file type of the malware using the python magic library I write the following code.

```
import magic

mgicByte = magic.from_file('C:\\Users\\20I-1955\\Desktop\\Assignment-
1\\Malwares\\7.
W32.SecretKAN.Trojan\\e42991a0d7c95b074b68f6b55b8bcb7aed1749d104d8ab37ae9ded32f
18ca74b\\e42991a0d7c95b074b68f6b55b8bcb7aed1749d104d8ab37ae9ded32f18ca74b')

print(mgicByte)
```

After running the code, I found that the file type of malware is "PE32 executable (GUI) Intel 80386, for MS Windows".

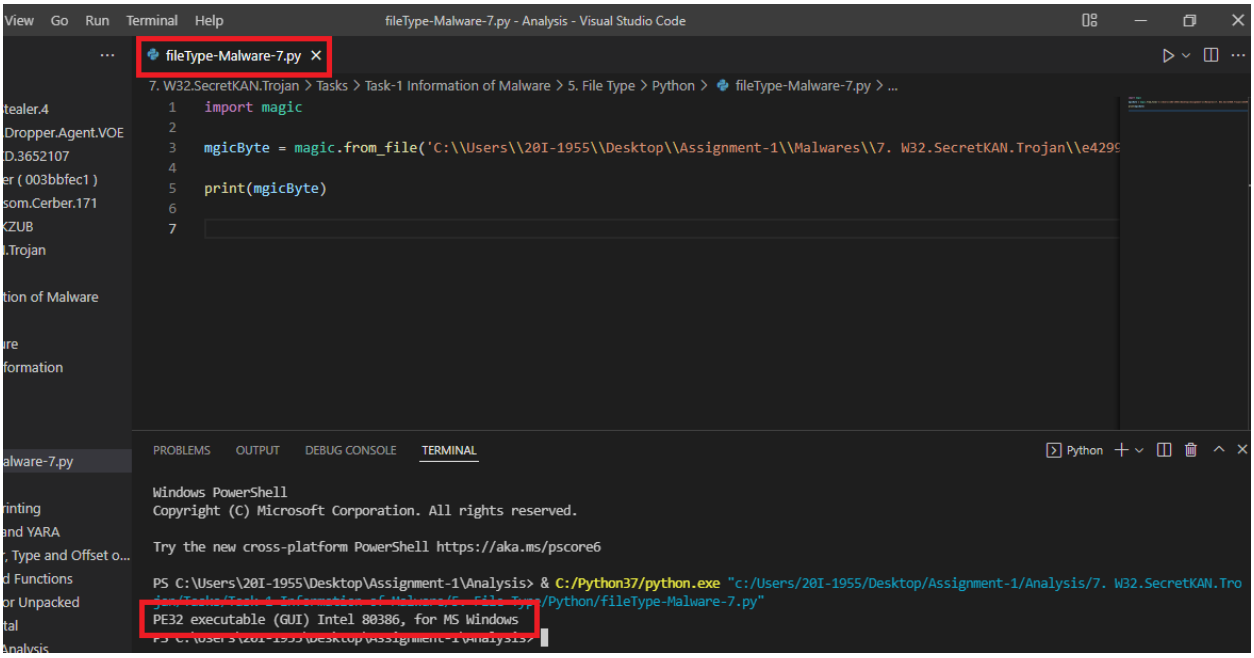

Figure 155: Malware-7 File Type through Python Magic Library

Tool-2 TrID Net

To find the file type of the malware using TrID Net I open the file in TrID Net and find that the malware file is “Win32 Executable MS Visual C++” as it has the “48.9%” match.

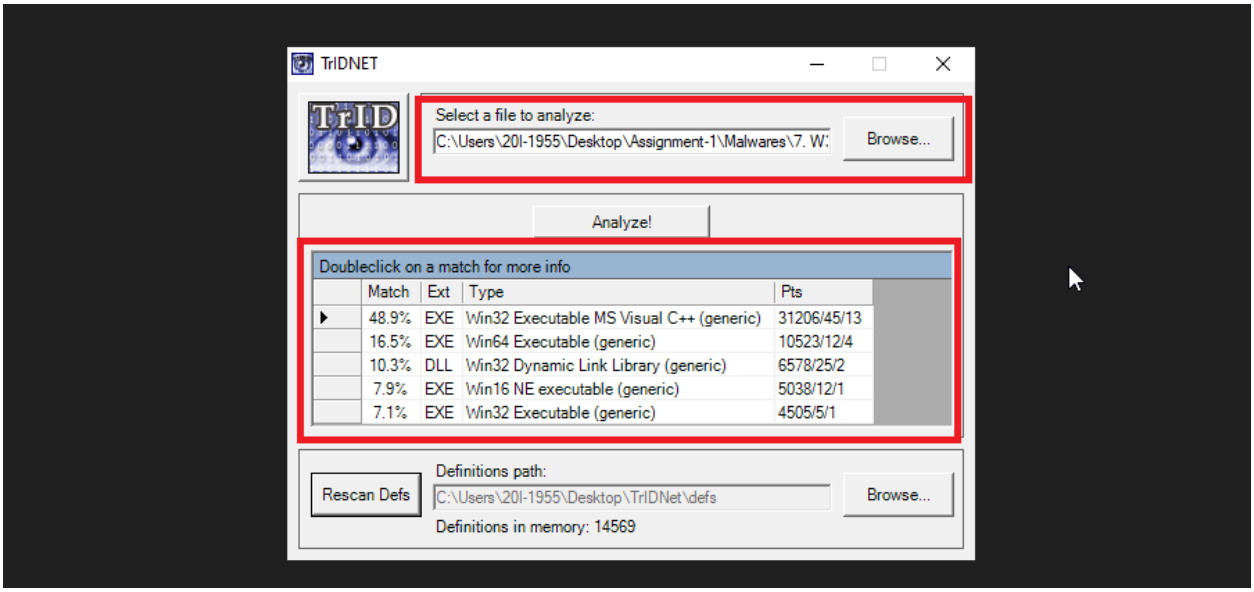

Figure 156: Malware-7 File Type through TrID Net

Task-2 What is the fingerprinting information of the malware file?

To find the hashes of the malware file I have used two different tools HashCalc and Python hashlib library.

Tool-1 HashCalc

To find the hash function of the file in HashCalc I open the file in HashCalc and find the hash of the file.

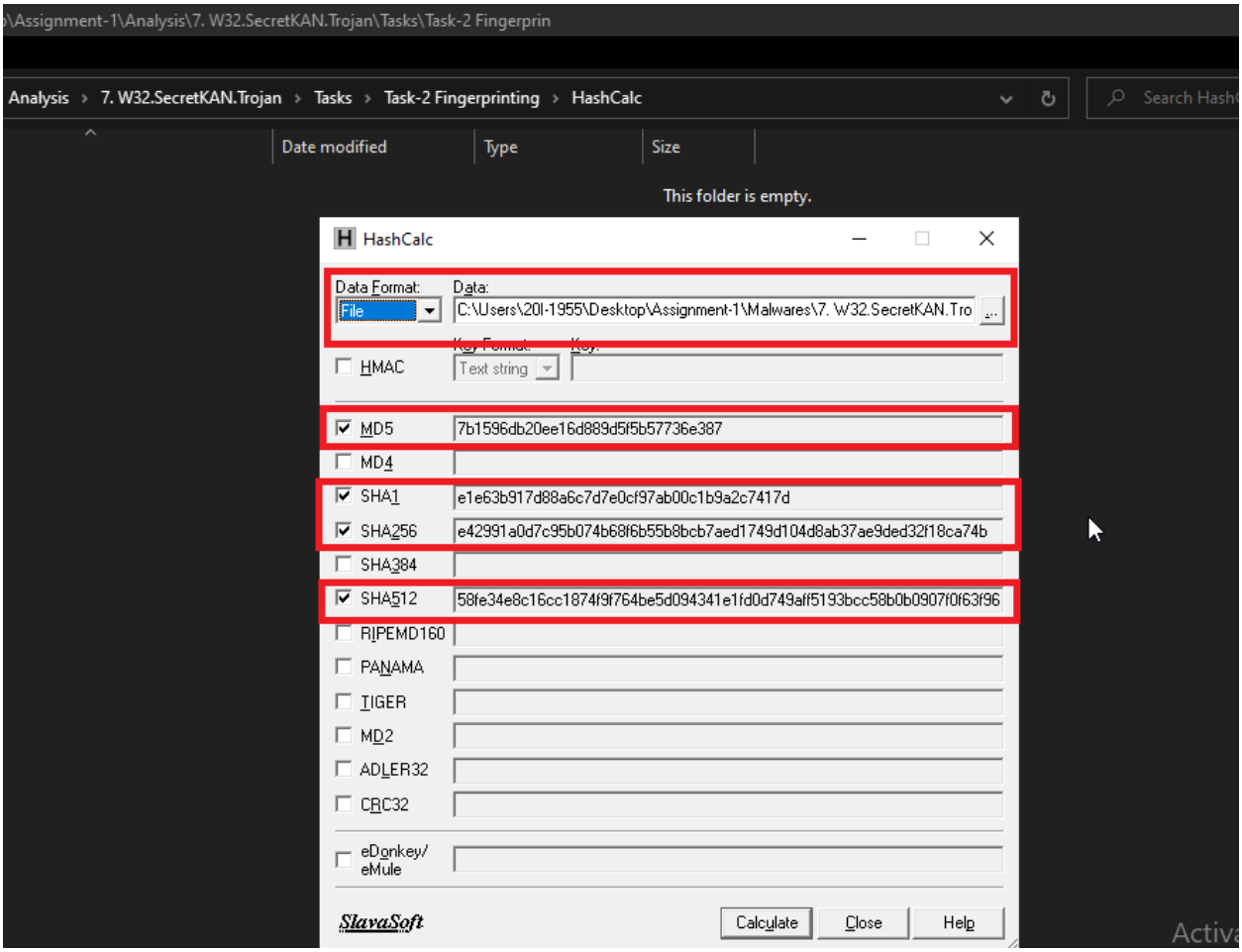

Figure 157: Malware-7 Fingerprinting through HashCalc

Tool-2 Python Hashlib Library

To find the hash of the malware using the python Hashlib library I write the following code.

```
import hashlib

hashMd5 = hashlib.md5(open("C:\\Users\\20I-1955\\Desktop\\Assignment-
1\\Malwares\\7.
W32.SecretKAN.Trojan\\e42991a0d7c95b074b68f6b55b8bcb7aed1749d104d8ab37ae9ded32f
18ca74b\\e42991a0d7c95b074b68f6b55b8bcb7aed1749d104d8ab37ae9ded32f18ca74b",
"rb").read()).hexdigest()
```

```
hashSha1 = hashlib.sha1(open("C:\\Users\\20I-1955\\Desktop\\Assignment-
1\\Malwares\\7.
W32.SecretKAN.Trojan\\e42991a0d7c95b074b68f6b55b8bcb7aed1749d104d8ab37ae9ded32f
18ca74b\\e42991a0d7c95b074b68f6b55b8bcb7aed1749d104d8ab37ae9ded32f18ca74b",
"rb").read()).hexdigest()
hashSha256 = hashlib.sha256(open("C:\\Users\\20I-1955\\Desktop\\Assignment-
1\\Malwares\\7.
W32.SecretKAN.Trojan\\e42991a0d7c95b074b68f6b55b8bcb7aed1749d104d8ab37ae9ded32f
18ca74b\\e42991a0d7c95b074b68f6b55b8bcb7aed1749d104d8ab37ae9ded32f18ca74b",
"rb").read()).hexdigest()
hashSha512 = hashlib.sha512(open("C:\\Users\\20I-1955\\Desktop\\Assignment-
1\\Malwares\\7.
W32.SecretKAN.Trojan\\e42991a0d7c95b074b68f6b55b8bcb7aed1749d104d8ab37ae9ded32f
18ca74b\\e42991a0d7c95b074b68f6b55b8bcb7aed1749d104d8ab37ae9ded32f18ca74b",
"rb").read()).hexdigest()

print("MD5:    ",hashMd5)
print("SHA1:   ",hashSha1)
print("SHA256: ",hashSha256)
print("SHA512: ",hashSha512)
```

After running the code, I found that the hashes of the malware file

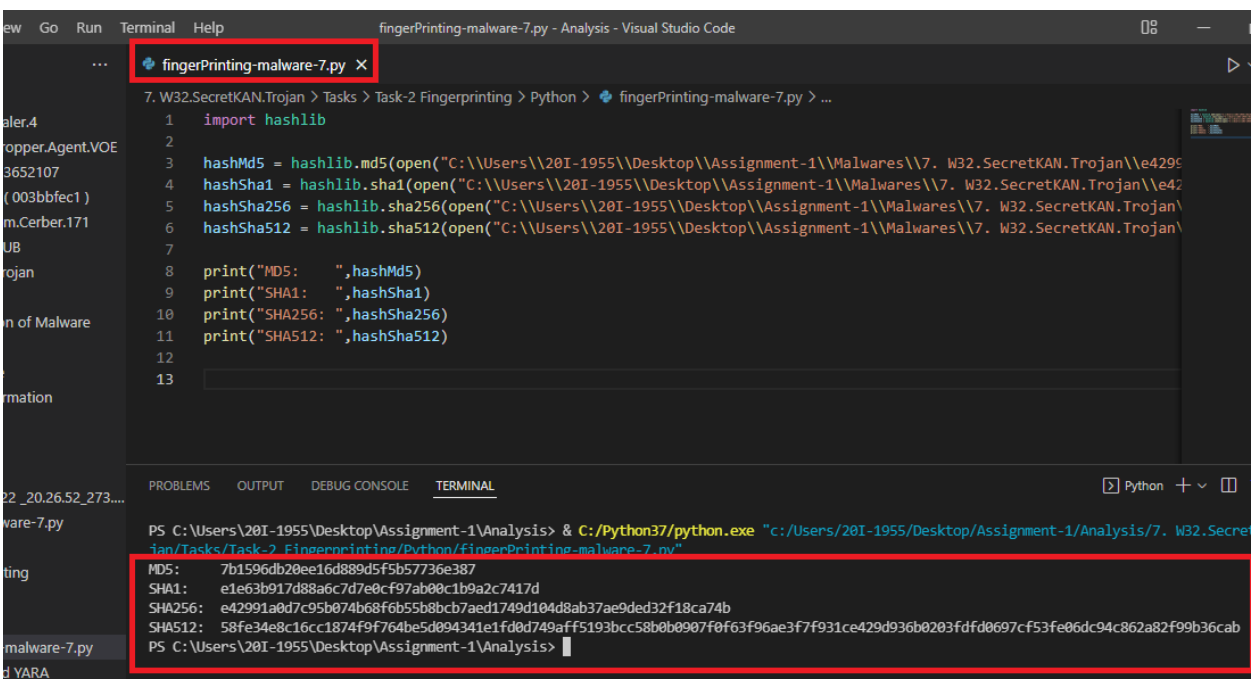

Figure 158: Malware-7 Fingerprinting through Python Hashlib Library

Task-3 What type of malicious strings are in the Malware file?

To extract the strings from the malware file I use the tool called "**Strings**" and then to scan the malware according to the extracted strings I write the YARA rule.

Tool-1 Strings

To extract the strings from the malware file and save them in the text file I run the following command

```
strings malware file path > text file name
```

After running the command all the strings are extracted from the malware file and stored in the text file.

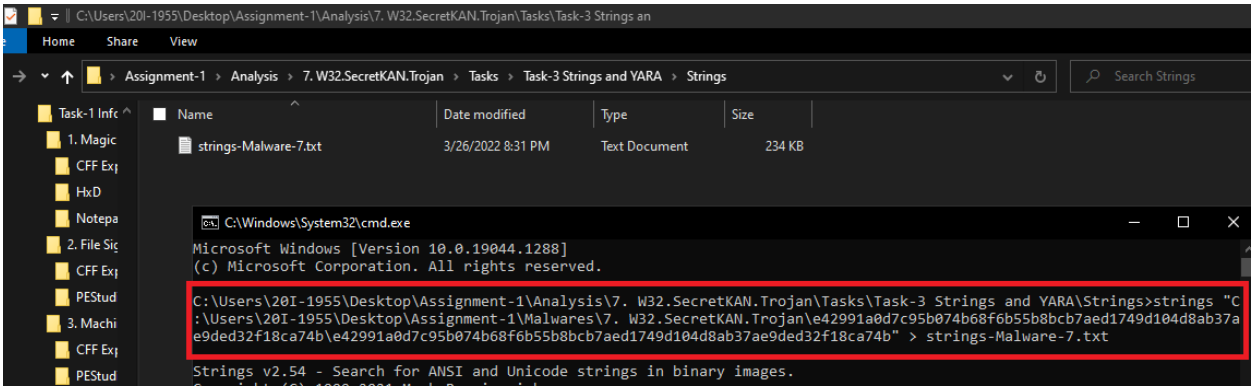

Figure 159: Malware-7 Strings Extraction through Strings Tool

After that, I open the text file to check the strings that were extracted from the malware and found some malicious strings.

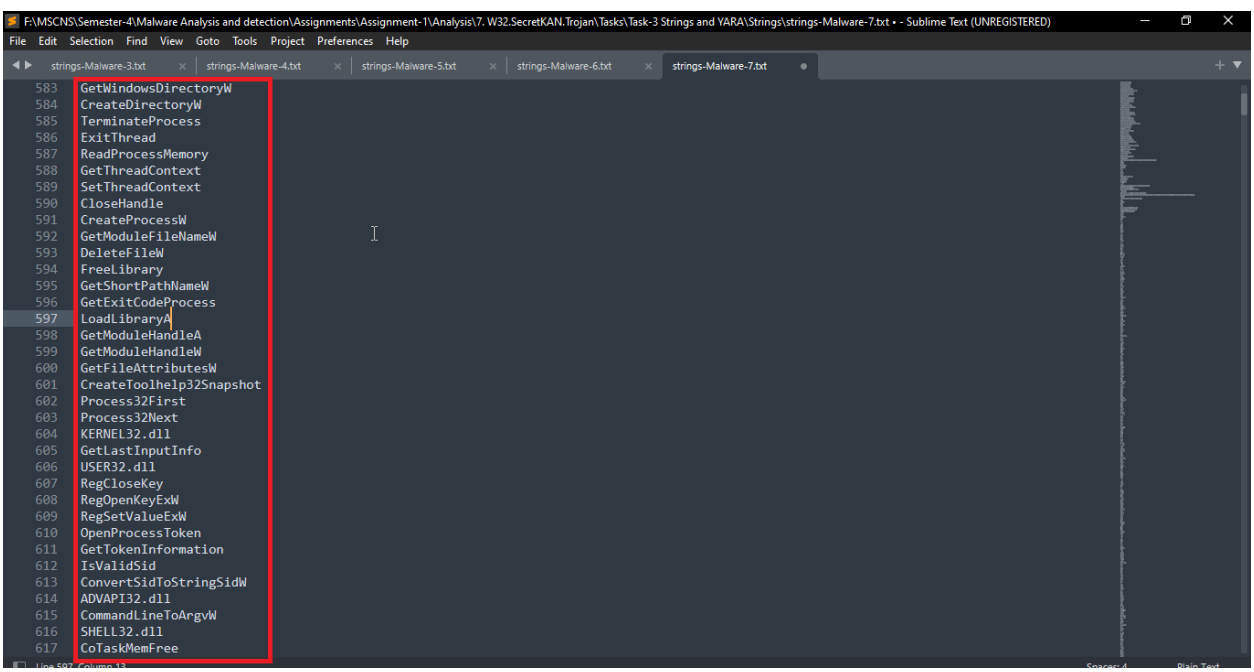

Figure 160: Malware-7 Strings Extracted through Strings Tool

Tool-2 YARA Rule

After extracting the string my next task is to write the YARA rule that scans the file and triggers the warning if it finds the malicious string in the malware file. For that, I write the following rule

```
rule malware_W32_SecretKAN_Trojan{
    meta:
        malwareName = "W32.SecretKAN.Trojan"
        ruleAuthor = "20I-1955_Muhammad Osama Khalid"
    strings:
        $str1 = "RegOpenKey" nocase
        $str2 = "RegQueryValue" nocase
        $str3 = "RegCloseKey" nocase
        $str4 = "kernel32" nocase
        $str5 = "shell32" nocase
    condition:
        any of them
}
```

Note: These strings are extracted from analyzing all the malware samples that are given for this assignment.

After that, I run the rule and find that the rule that I write works successfully as it detects the malicious string extracted above in the malware file.

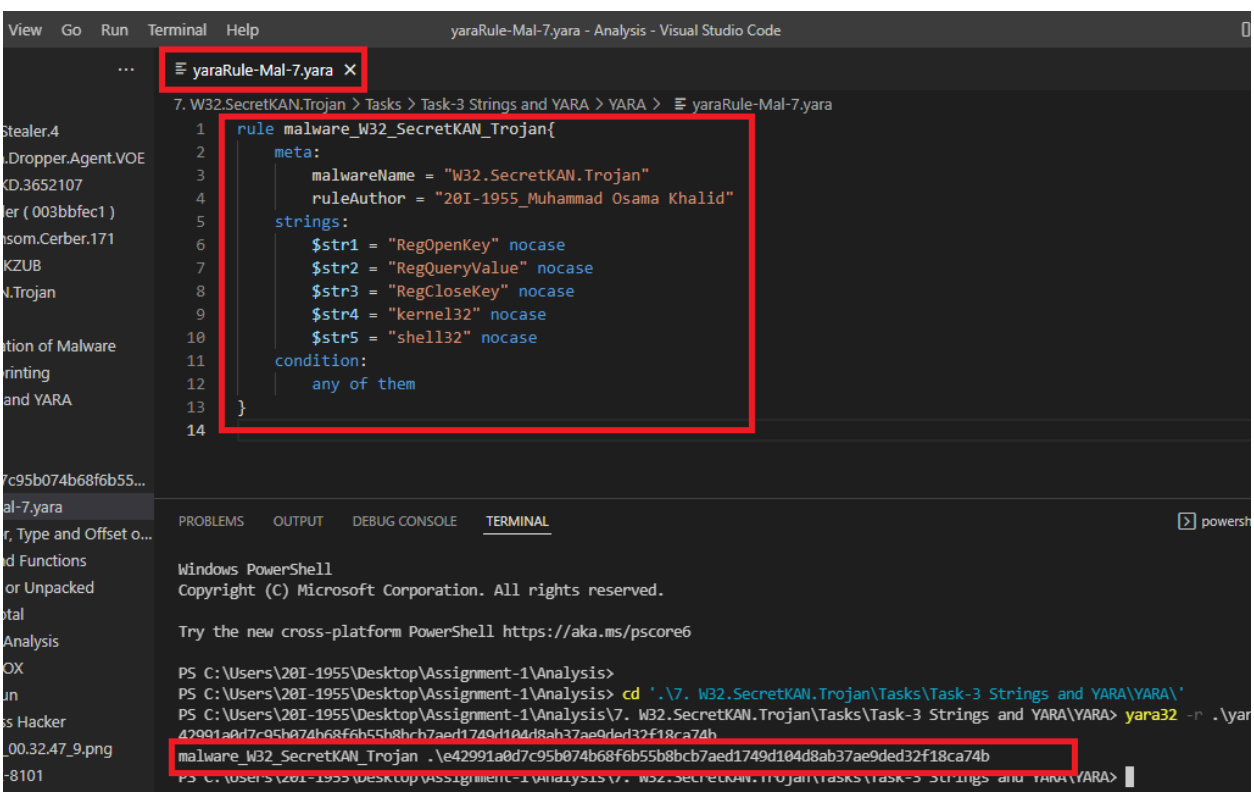

Figure 161: Malware-7 Malware detection through YARA Rule

Task-4 Number, type, and offset of different sections of Malware?

Sections contain the important information about the file as it describes how the executable is made. To find the number, type, and offset of different sections of malware files I use three different tools PE Studio and Exeinfo.

Tool-1 PE Studio

To find the number, type, and offset of different sections of the malware file I open the file in the PE Studio, then from the left panel first I click on "**file-header**" and then from the right panel I found the column named "**sections**". From there I found that this malware file has three sections and to find then what they are I click on "**sections**" in the left panel. After that from the right panel, I found the sections that this malware file has.

No. of Section	Type	Offset
1	.text	0x00401000
2	.rdata	0x00408000
3	.data	0x0040A000
4	.reloc	0x004DC000

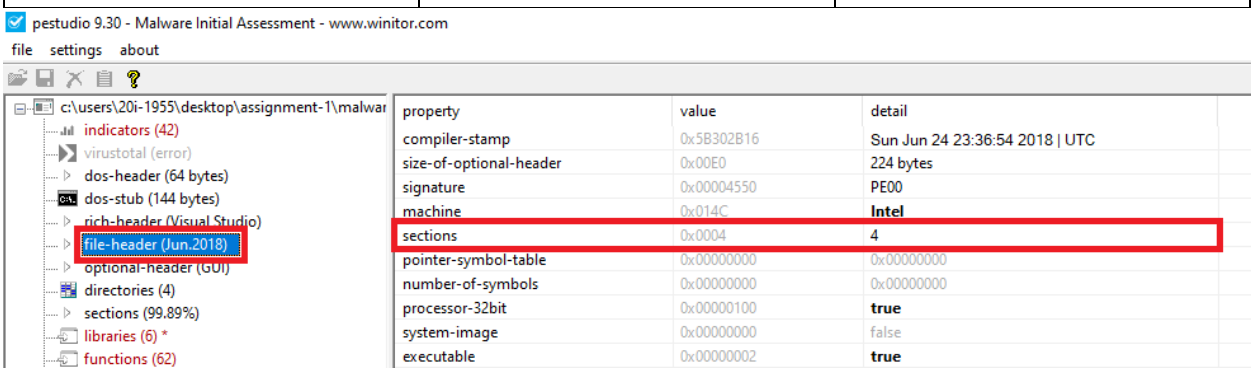

Figure 162: Malware-7 Information of Sections through PE Studio

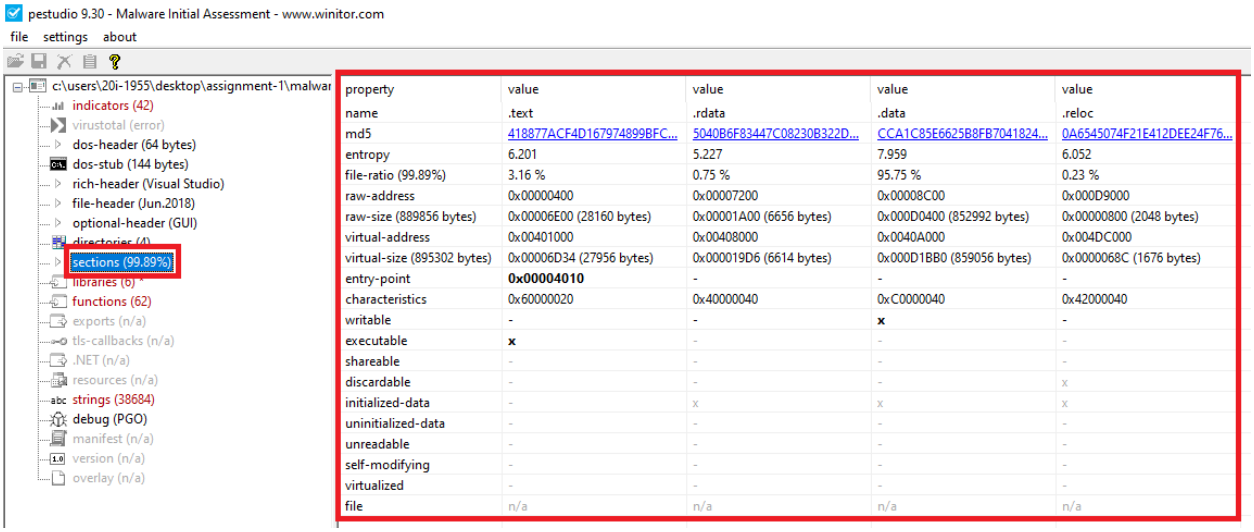

Figure 163: Malware-7 Information of Sections through PE Studio

Task-5 Which DLLs and their number of functions are used by Malware?

To find what DLLs and functions this malware used I used the tool PE Studio.

Tool-1 PE Studio

To find the DLLs and their number of functions used by malware files I open the file in the PE Studio, then from the left panel first I click on "**libraries**" and then from the right pane I found the list of DLLs and their functions used by this malware.

No. of DLL	Name	No. of functions
1	wininet.dll	9
2	kernel32.dll	43
3	user32.dll	1
4	advapi32.dll	7
5	shell32.dll	1
6	ole32.dll	1

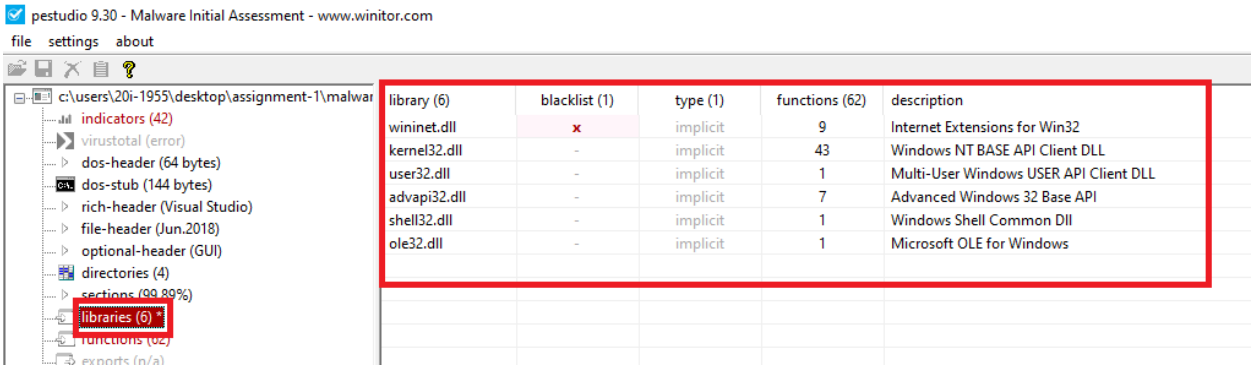

Figure 164: Malware-7 Information of DLLs and functions through PE Studio

Task-6 What is packing information if Malware is packed?

To check whether the malware is packed or not I use the tool Exeinfo.

Tool-1 Exeinfo

To find whether the malware file is packed or not I open the file in Exeinfo and at the bottom I found that the malware is not packed.

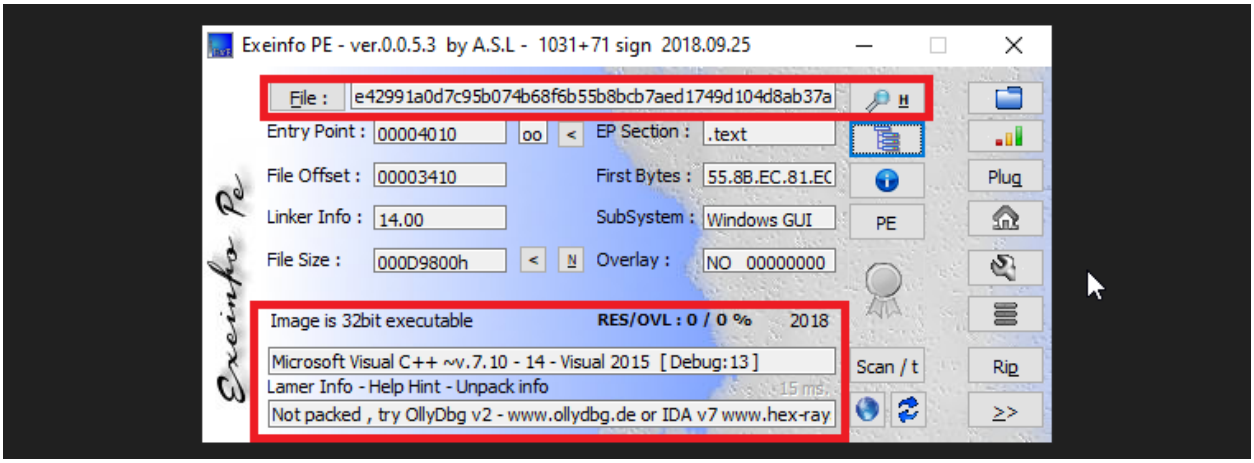

Figure 165: Malware-7 Packed or unpacked malware information through Exeinfo

Task-7 VirusTotal analysis of the malware file

To scan the malware file from the virus total I visit the website and upload the malware file. After some time, I found that 67 antivirus engines marked this malware file as malicious.

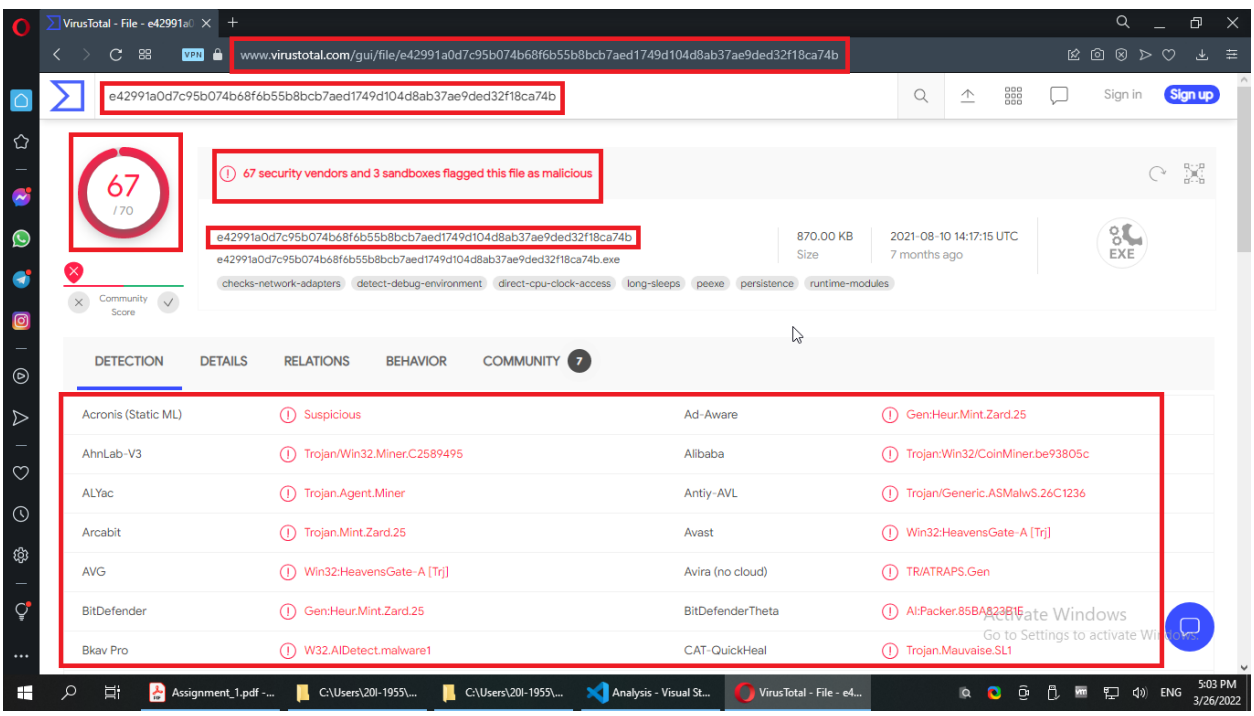

Figure 166: Malware-7 VirusTotal Analysis

Moreover, in the details tab, I find more information about the malware that I already explained in the previous tasks.

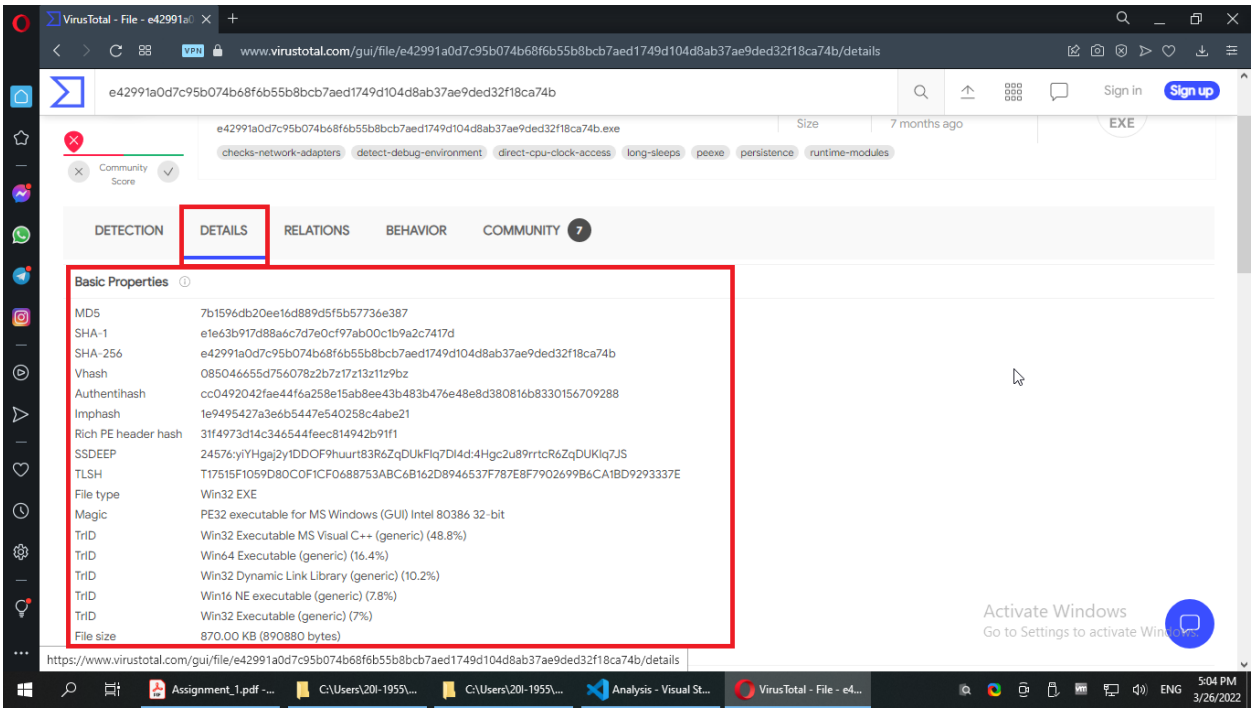

Figure 167: Malware-7 VirusTotal Analysis

Task-8 Analysis of the malware file through Hybrid Analysis

To scan the malware file from Hybrid Analysis I visit the website and upload the malware file. After some time, I found that the file is triggered by malicious antivirus search engines.

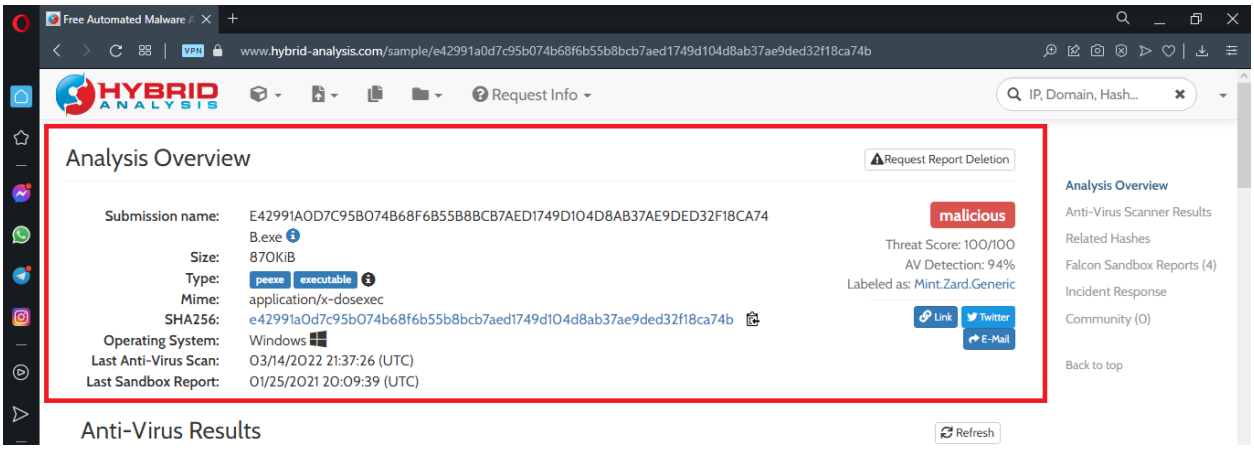

Figure 168: Malware-7 Analysis through Hybrid Analysis

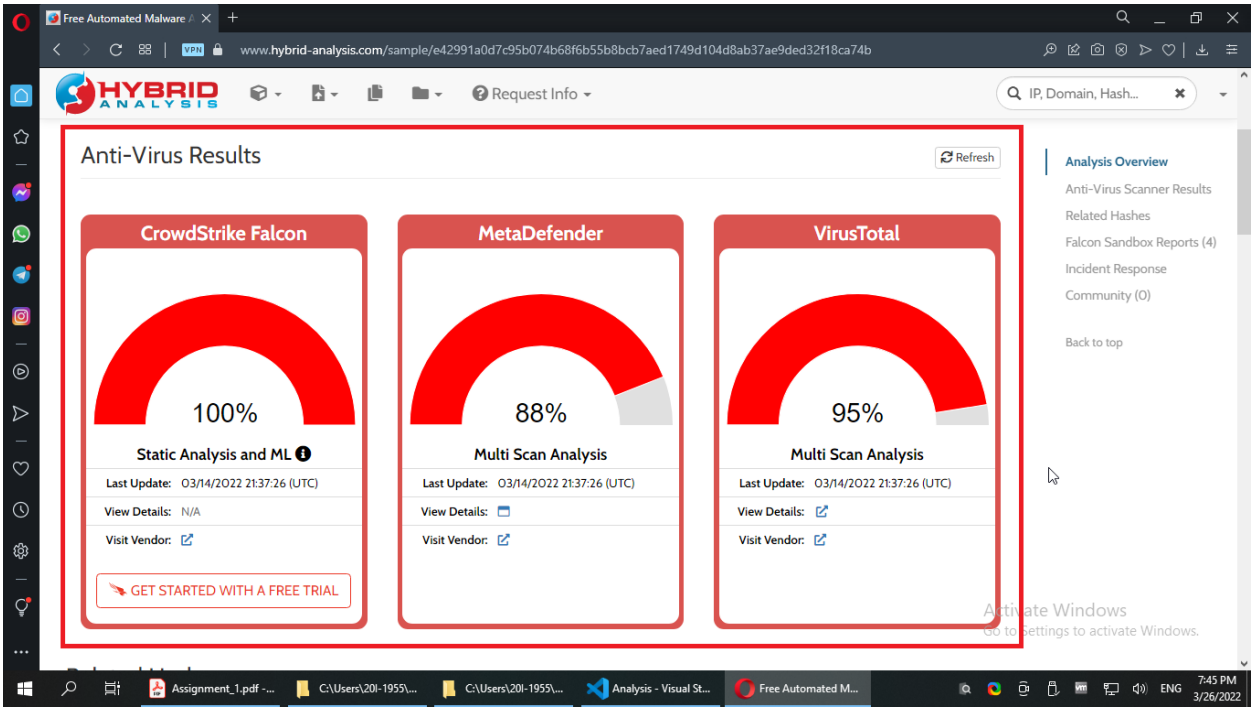

Figure 169: Malware-7 Analysis through Hybrid Analysis

Malware-8 Backdoor.TXLK-8101

This malware belongs to the family of backdoors.

Task-1 What is complete information malware have

In this task, I have to find the basic information of the malware that includes identifying magic byte, signatures, machine info, exe type, and file type.

Task-1.1 Magic Byte

To find the magic byte of the malware there are a bunch of tools available and for this task, I have used three different tools Notepad++, HxD, and CFF Explorer.

Tool-1 Notepad++

To find the magic byte of the malware file I open the file in Notepad++, then from the plugin tab click on “HEX-Editor” and then click on “View in HEX”. On viewing the file in HEX-Editor I found that the magic byte of the malware file is “**MZ**”. This means that the malware file is executable. Moreover, I also find that this file can’t be able to run in DOS mode.

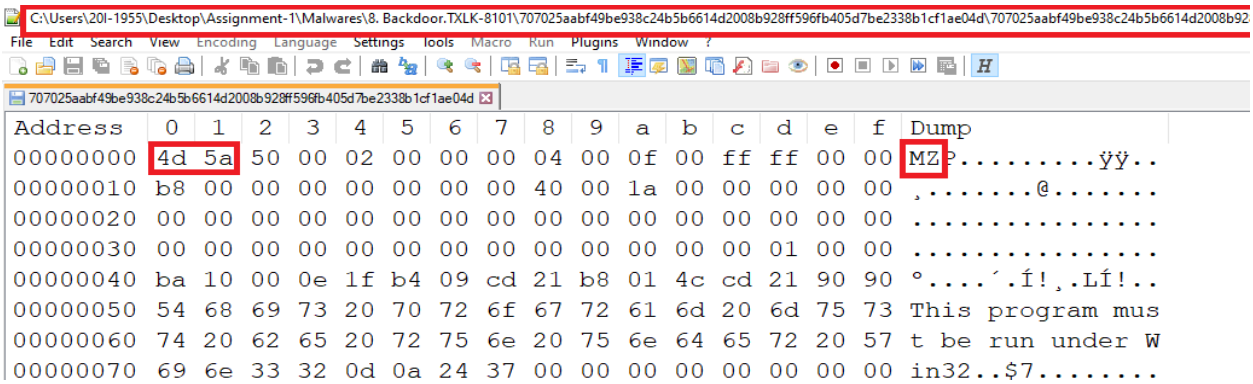

Figure 170: Malware-8 Magic Byte through Notepad++

Tool-2 HxD

To find the magic byte of the malware file I open the file in the HxD and find that the magic byte of the file is “**MZ**” this means that the malware file is executable. Moreover, I also find that can only run on the Win32 operating system.

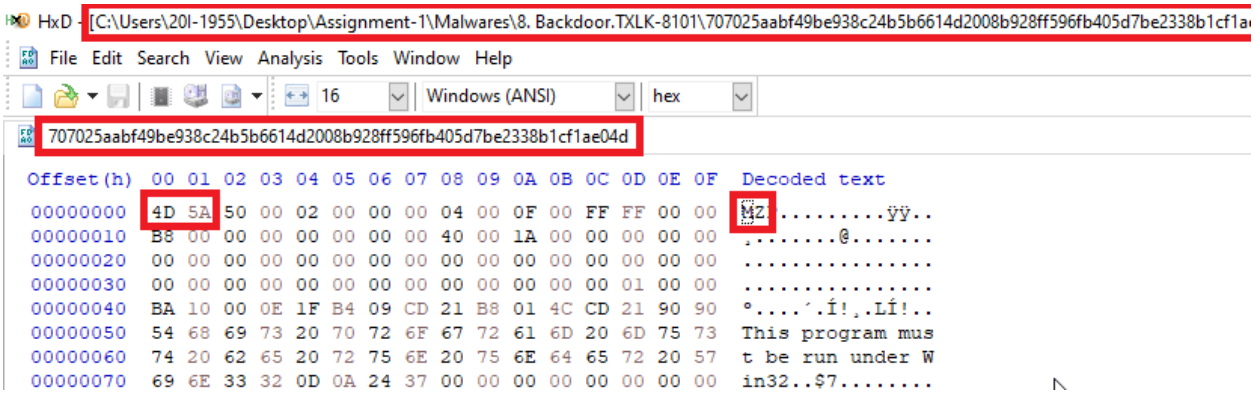

Figure 171: Malware-8 Magic Byte through HxD

Tool-3 CFF Explorer

To find the magic byte of the malware file I open the file in the CFF Explorer, then from the left panel I click on "**DOS Header**". In the "**DOS Header**" I find the magic byte in the "e_magic" column that is "**5A 4D**" which is equivalent to "**4D 5A**" because in CFF Explorer hexadecimal are represented in right to left order. So "**4D = M**" and "**5A = Z**" mean that the malware file is executable.

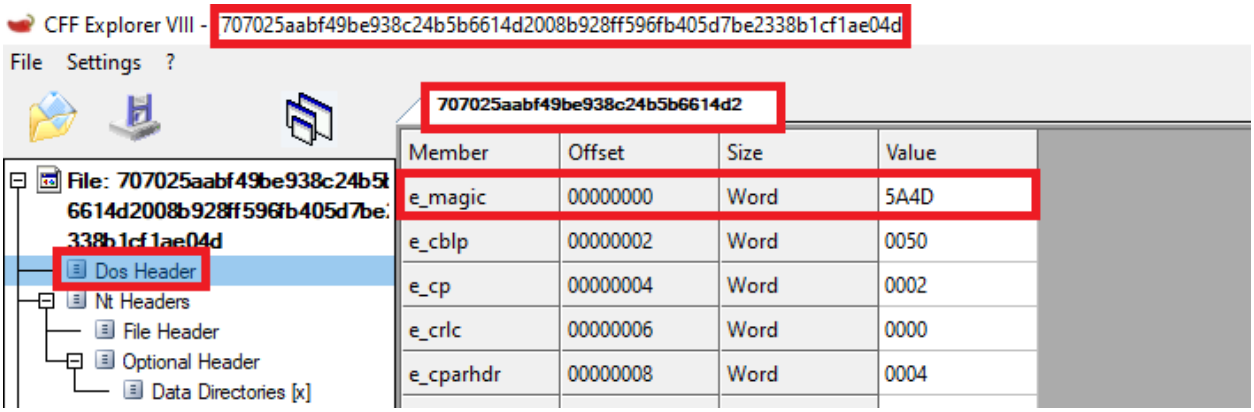

Figure 172: Malware-8 Magic Byte through CFF Explorer

Task-1.2 File Signature

A file signature is used to identify the content of the file. This information is stored in the file's header. To find the file signature of the malware I have used two different tools PE Studio and CFF Explorer.

Tool-1 PE Studio

To find the file signature of the malware file I open the file in the PE Studio, then from the left panel I click on "**file-header**" and then from the right panel I found the column named "**signature**". From there I found that the signature of the file is "**PE**" which means that this file is portable executable.

Figure 173: Malware-8 File Signature through PE Studio

Tool-2 CFF Explorer

To find the file signature of the malware file I open the file in the CFF Explorer, then from the left panel I click on "**Nt Headers**" and then from the right panel I found the column named "**signature**". From there I found that the signature of the file is "**45 50**" which is equivalent to "**50 45**" because in CFF Explorer hexadecimal are represented in right to left order. So "**50 = P**" and "**45 = E**" which means that this file is portable executable.

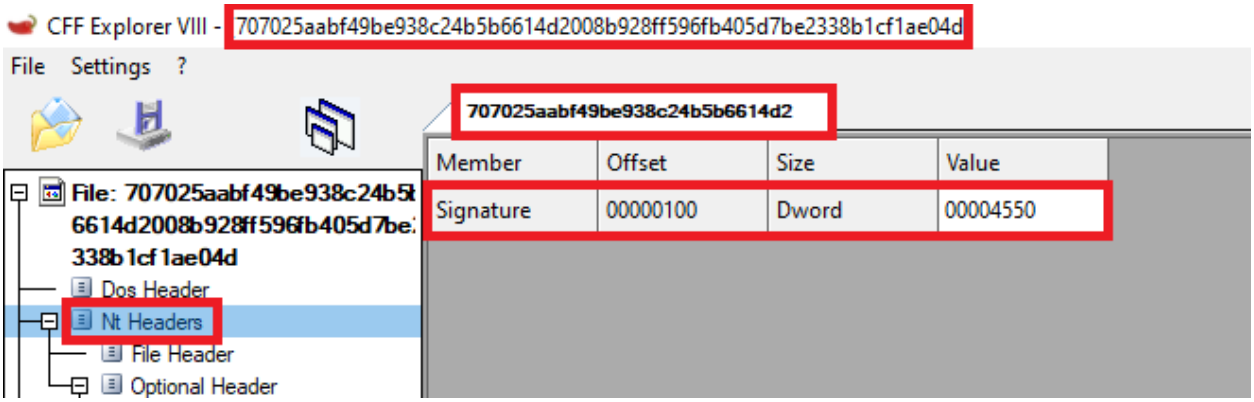

Figure 174: Malware-8 File Signature through CFF Explorer

Task-1.3 Machine Information

To find the machine information of the malware I have used two different tools PE Studio and CFF Explorer.

Tool-1 PE Studio

To find the file signature of the malware file I open the file in the PE Studio, then from the left panel I click on "**file-header**" and then from the right panel I found the column named "**machine**". From there I found that this malware is designed for Intel-based machines. Other than machine information I also find that when the malware is created from the "**compiler-stamp**" column. This malware is created on Jun 19, 1992, on Friday at 11:11 PM. Also find that this malware file has 8 sections and the size of the optional header is 224 bytes.

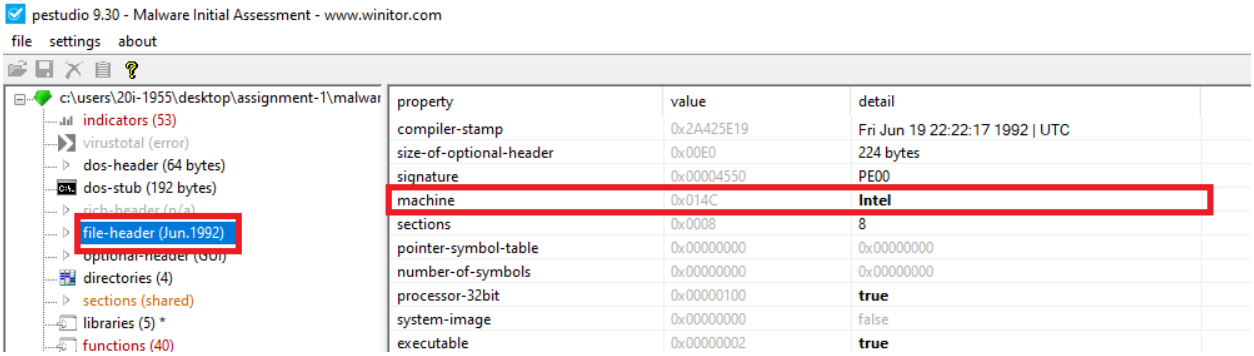

Figure 175: Malware-8 Machine Information through PE Studio

Tool-2 CFF Explorer

To find the file signature of the malware file I open the file in the CFF Explorer, then from the left panel I click on "**file header**" and then from the right panel I found the column named "**Machine**". From there I found that this malware is designed for Intel 386 based machines.

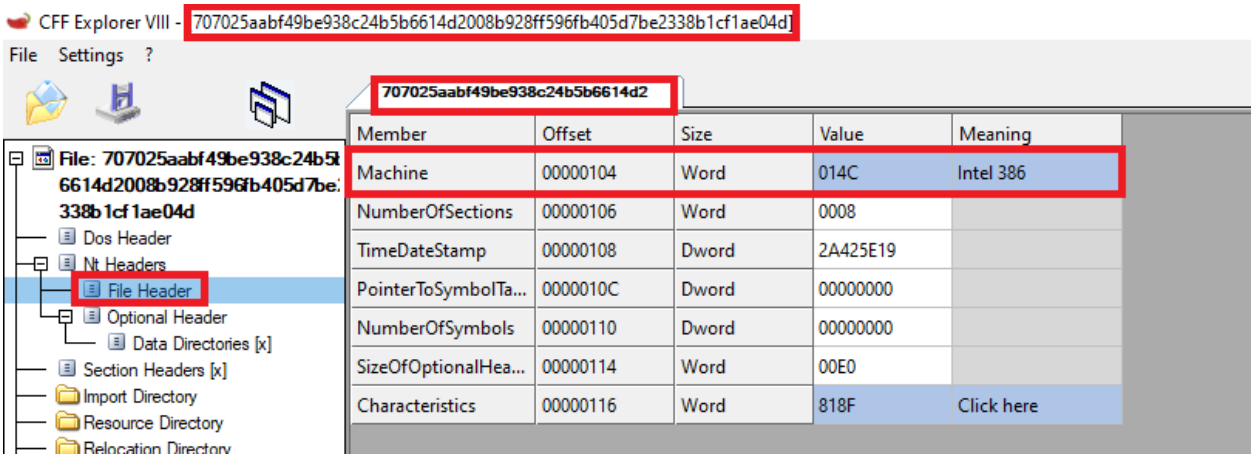

Figure 176: Malware-8 Machine Information through CFF Explorer

Task-1.4 Exe Type

Exe type is used to identify which type of executable it is either 32-bit or 64-bit and to find that I used two different tools PE Studio and CFF Explorer.

Tool-1 PE Studio

To find the exe type of the malware file I open the file in the PE Studio, then from the left panel I click on "**optional-header**" and then from the right panel I found the column named "**magic**". From there I found that the exe type of this malware file is "0x010B" which means that this malware file is a 32-bit portable executable. Other than exe type I also find the entry point of the malware which is "CODE".

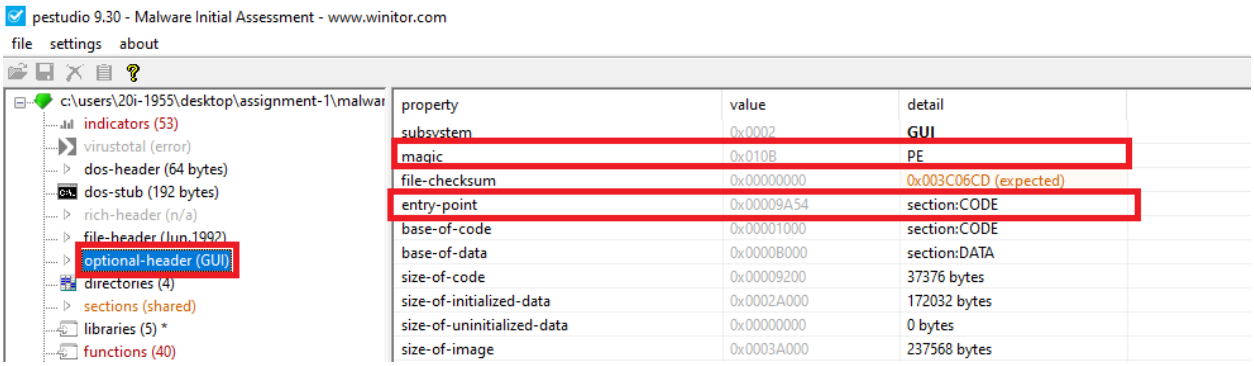

Figure 177: Malware-8 Exe type through PE Studio

Tool-2 CFF Explorer

To find the file signature of the malware file I open the file in the CFF Explorer, then from the left panel I click on "**optional header**" and then from the right panel I found the column named "**Magic**". From there I found that the exe type of this malware file is "010B" which means that this malware file is a 32-bit portable executable. Other than exe type I also find the entry point of the malware which is "CODE".

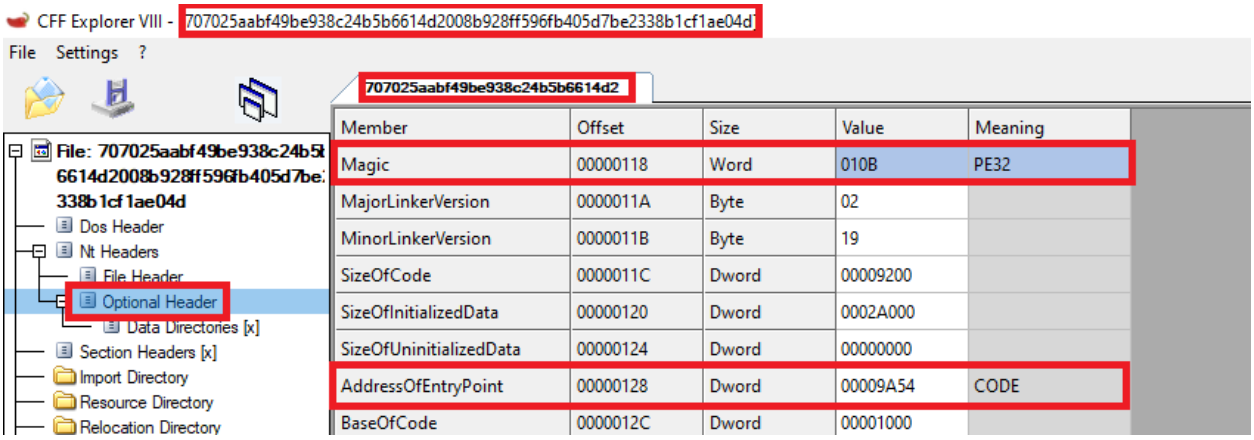

Figure 178: Malware-8 Exe type through CFF Explorer

Task-1.5 File Type

File Type is used to identify which type of file it is. I used two different tools Python Magic Library and TrID Net.

Tool-1 Python Magic Library

To find the file type of the malware using the python magic library I write the following code.

```
import magic

mgicByte = magic.from_file('C:\\Users\\20I-1955\\Desktop\\Assignment-
1\\Malwares\\8. Backdoor.TXLK-
8101\\707025aabf49be938c24b5b6614d2008b928ff596fb405d7be2338b1cf1ae04d\\707025a
abf49be938c24b5b6614d2008b928ff596fb405d7be2338b1cf1ae04d')

print(mgicByte)
```

After running the code, I found that the file type of malware is "PE32 executable (GUI) Intel 80386, for MS Windows".

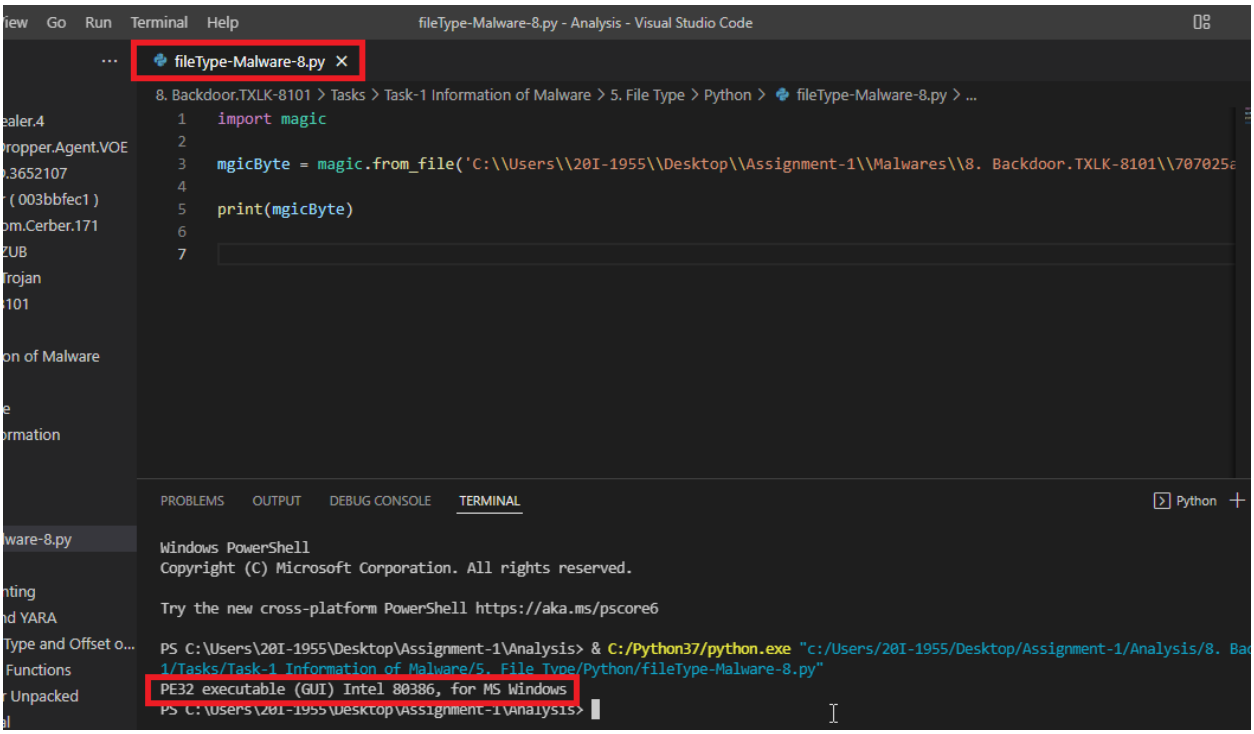

Figure 179: Malware-8 File Type through Python Magic Library

Tool-2 TrID Net

To find the file type of the malware using TrID Net I open the file in TrID Net and find that the malware file is "Inno Setup installer" as it has the "80.4%" match.

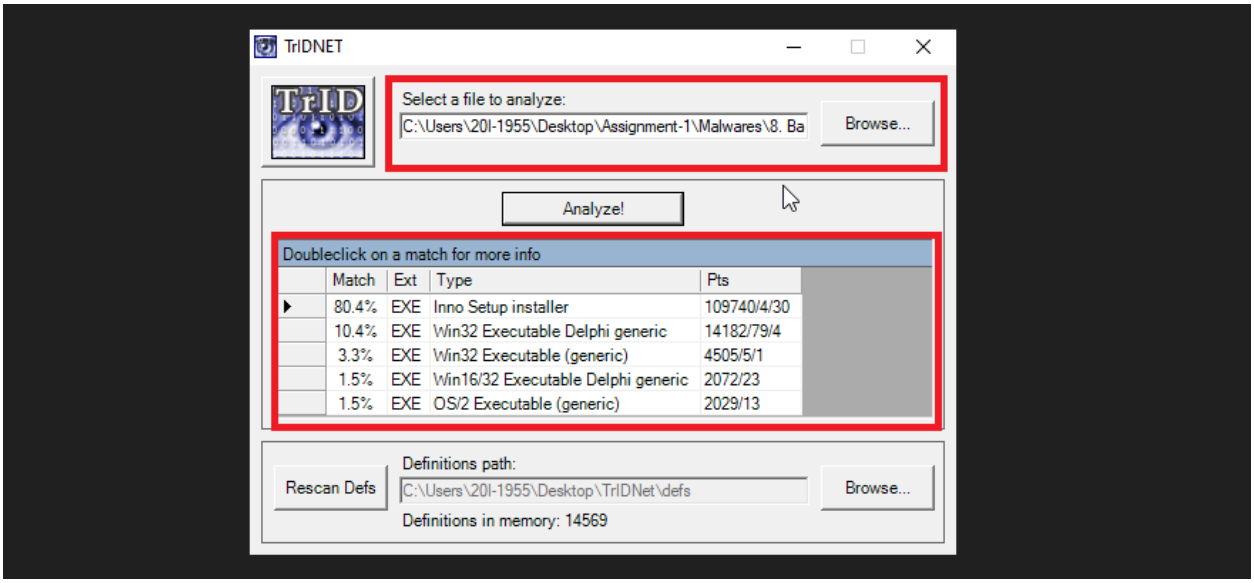

Figure 180: Malware-8 File Type through TrID Net

Task-2 What is the fingerprinting information of the malware file?

To find the hashes of the malware file I have used two different tools HashCalc and Python hashlib library.

Tool-1 HashCalc

To find the hash function of the file in HashCalc I open the file in HashCalc and find the hash of the file.

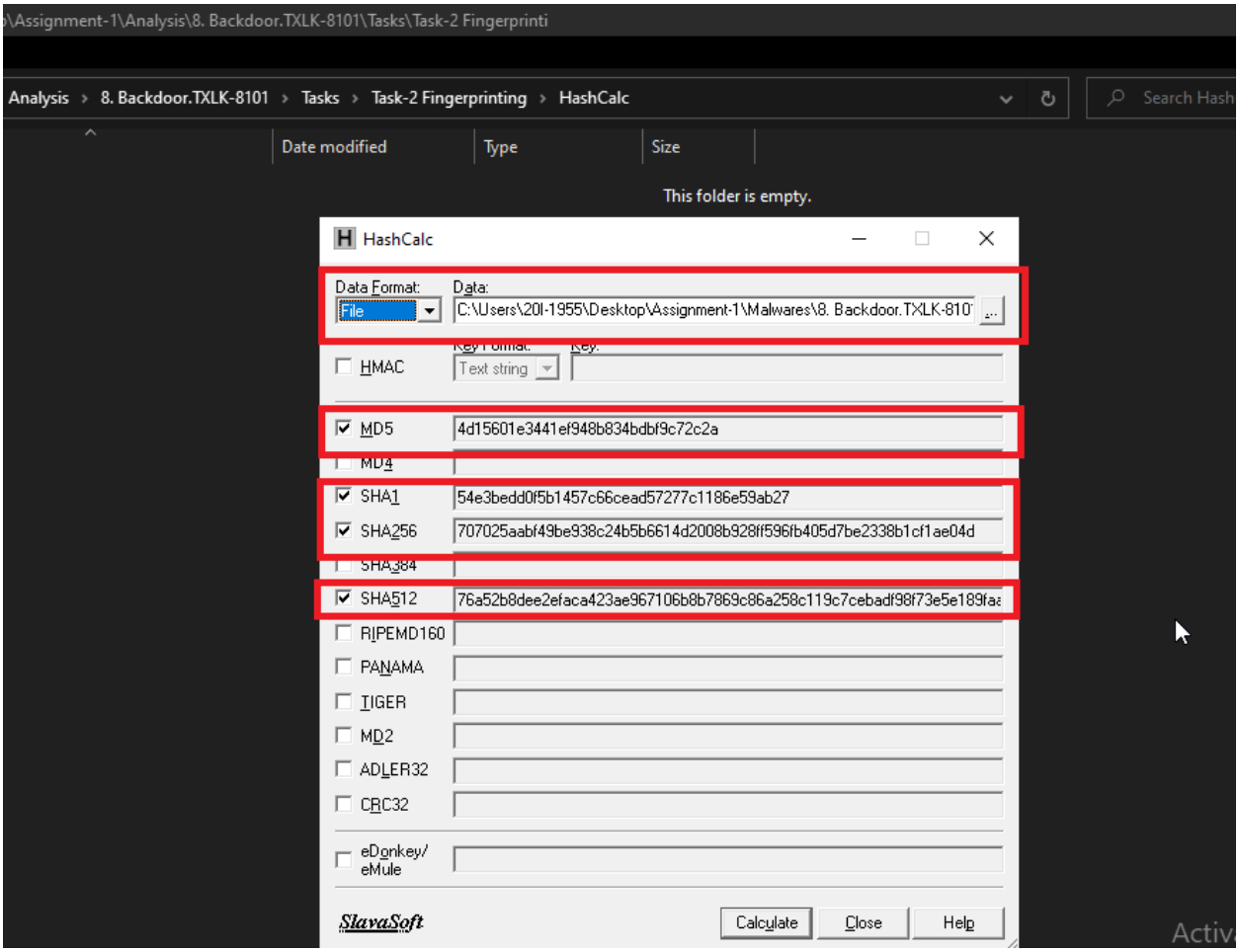

Figure 181: Malware-8 Fingerprinting through HashCalc

Tool-2 Python Hashlib Library

To find the hash of the malware using the python Hashlib library I write the following code.

```
import hashlib

hashMd5 = hashlib.md5(open("C:\\Users\\20I-1955\\Desktop\\Assignment-
1\\Malwares\\8. Backdoor.TXLK-
8101\\707025aabf49be938c24b5b6614d2008b928ff596fb405d7be2338b1cf1ae04d\\707025a
abf49be938c24b5b6614d2008b928ff596fb405d7be2338b1cf1ae04d",
"rb").read()).hexdigest()
```

```
hashSha1 = hashlib.sha1(open("C:\\Users\\20I-1955\\Desktop\\Assignment-
1\\Malwares\\8. Backdoor.TXLK-
8101\\707025aabf49be938c24b5b6614d2008b928ff596fb405d7be2338b1cf1ae04d\\707025a
abf49be938c24b5b6614d2008b928ff596fb405d7be2338b1cf1ae04d",
"rb").read()).hexdigest()
hashSha256 = hashlib.sha256(open("C:\\Users\\20I-1955\\Desktop\\Assignment-
1\\Malwares\\8. Backdoor.TXLK-
8101\\707025aabf49be938c24b5b6614d2008b928ff596fb405d7be2338b1cf1ae04d\\707025a
abf49be938c24b5b6614d2008b928ff596fb405d7be2338b1cf1ae04d",
"rb").read()).hexdigest()
hashSha512 = hashlib.sha512(open("C:\\Users\\20I-1955\\Desktop\\Assignment-
1\\Malwares\\8. Backdoor.TXLK-
8101\\707025aabf49be938c24b5b6614d2008b928ff596fb405d7be2338b1cf1ae04d\\707025a
abf49be938c24b5b6614d2008b928ff596fb405d7be2338b1cf1ae04d",
"rb").read()).hexdigest()

print("MD5:    ",hashMd5)
print("SHA1:   ",hashSha1)
print("SHA256: ",hashSha256)
print("SHA512: ",hashSha512)
```

After running the code, I found that the hashes of the malware file

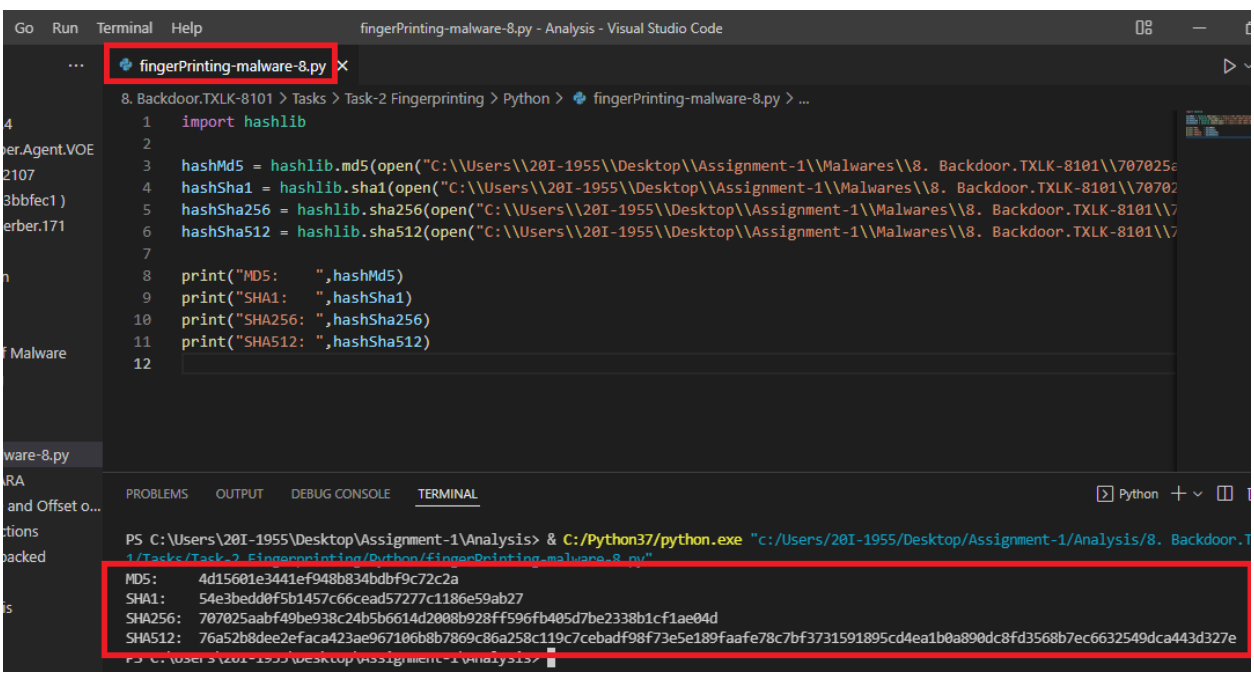

Figure 182: Malware-8 Fingerprinting through Python Hashlib Library

Task-3 What type of malicious strings are in the Malware file?

To extract the strings from the malware file I use the tool called "**Strings**" and then to scan the malware according to the extracted strings I write the YARA rule.

Tool-1 Strings

To extract the strings from the malware file and save them in the text file I run the following command

```
strings malware file path > text file name
```

After running the command all the strings are extracted from the malware file and stored in the text file.

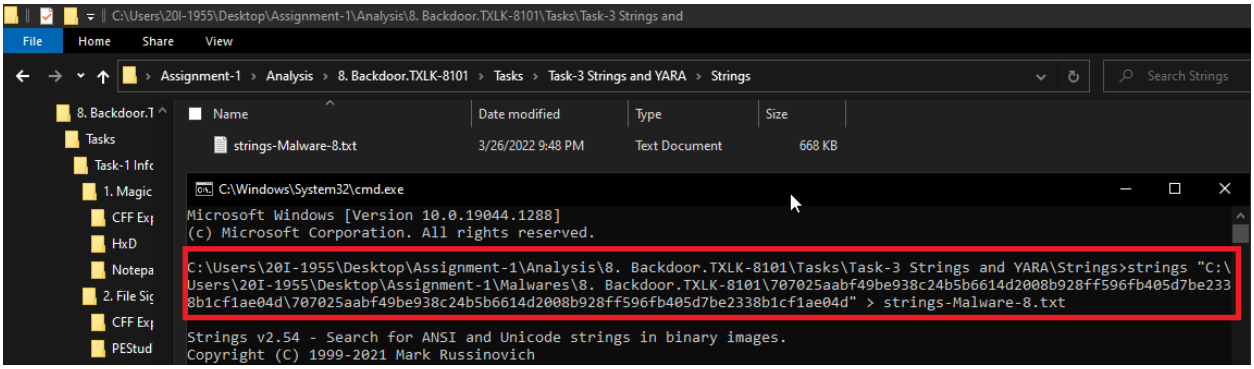

Figure 183: Malware-8 Strings Extraction through Strings Tool

After that, I open the text file to check the strings that were extracted from the malware and found some malicious strings.

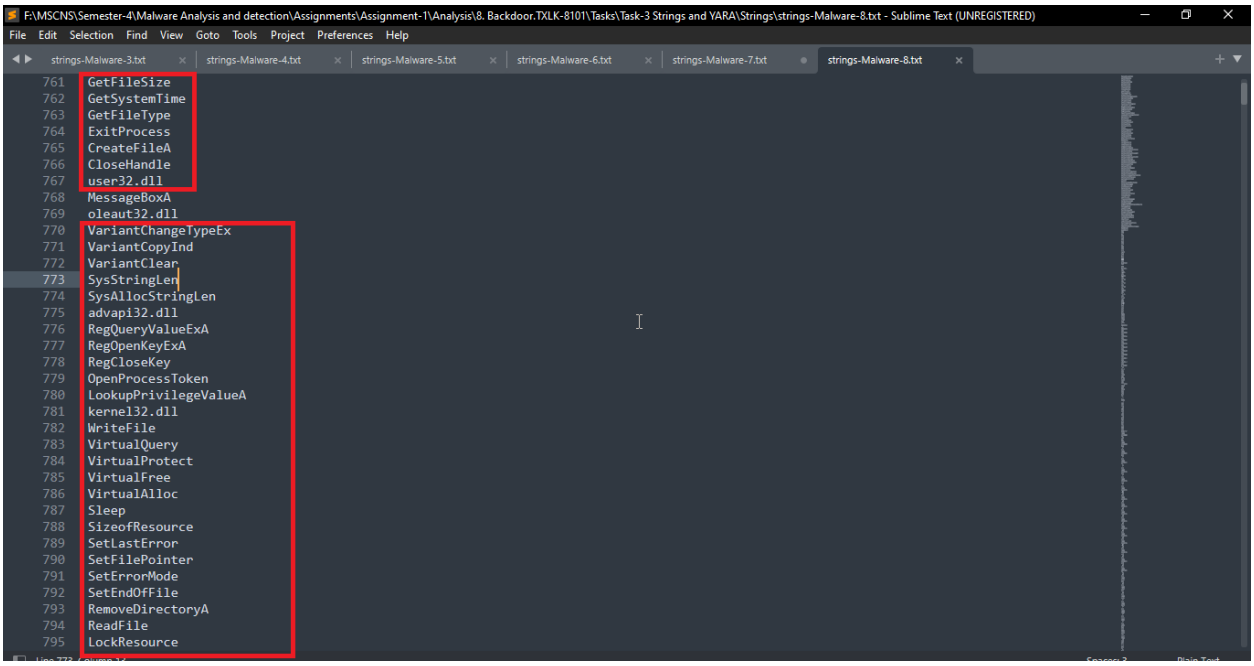

Figure 184: Malware-8 Strings Extracted through Strings Tool

Tool-2 YARA Rule

After extracting the string my next task is to write the YARA rule that scans the file and triggers the warning if it finds the malicious string in the malware file. For that, I write the following rule

```
rule malware_Backdoor_TXLK_8101{
    meta:
        malwareName = "Backdoor.TXLK-8101"
        ruleAuthor = "20I-1955_Muhammad Osama Khalid"
    strings:
        $str1 = "RegOpenKey" nocase
        $str2 = "RegQueryValue" nocase
        $str3 = "RegCloseKey" nocase
        $str4 = "kernel32" nocase
        $str5 = "shell32" nocase
    condition:
        any of them
}
```

Note: These strings are extracted from analyzing all the malware samples that are given for this assignment.

After that, I run the rule and find that the rule that I write works successfully as it detects the malicious string extracted above in the malware file.

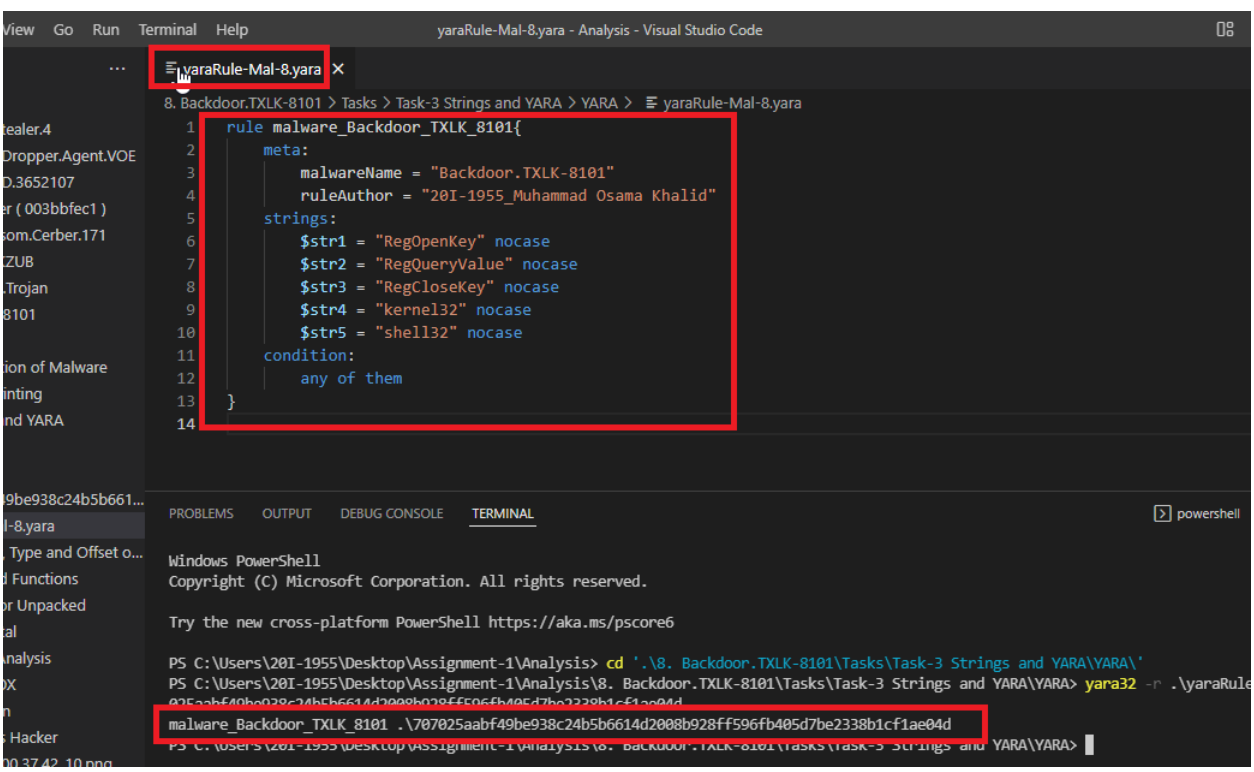

Figure 185: Malware-8 Malware detection through YARA Rule

Task-4 Number, type, and offset of different sections of Malware?

Sections contain the important information about the file as it describes how the executable is made. To find the number, type, and offset of different sections of malware files I use three different tools PE Studio and Exeinfo.

Tool-1 PE Studio

To find the number, type, and offset of different sections of the malware file I open the file in the PE Studio, then from the left panel first I click on "**file-header**" and then from the right panel I found the column named "**sections**". From there I found that this malware file has three sections and to find then what they are I click on "**sections**" in the left panel. After that from the right panel, I found the sections that this malware file has.

No. of Section	Type	Offset
1	CODE	0x00401000
2	DATA	0x0040B000
3	BSS	0x0040C000
4	.idata	0x0040D000
5	.tls	0x0040E000
6	.rdata	0x0040F000
7	.reloc	0x00410000
8	.rsrc	0x00411000

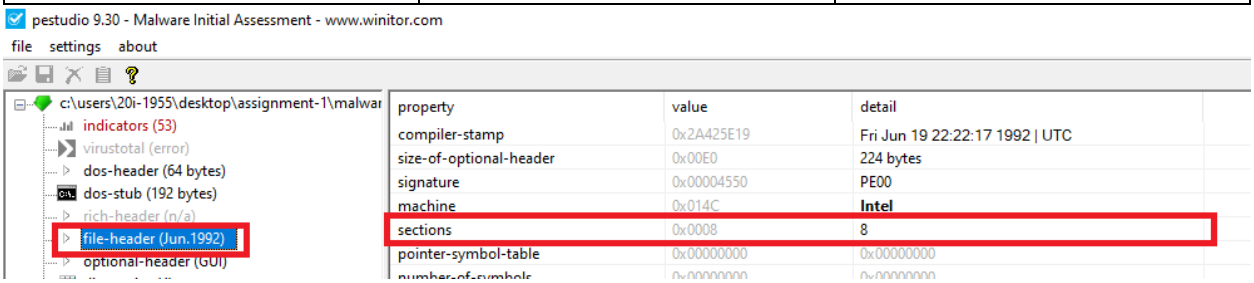

Figure 186: Malware-8 Information of Sections through PE Studio

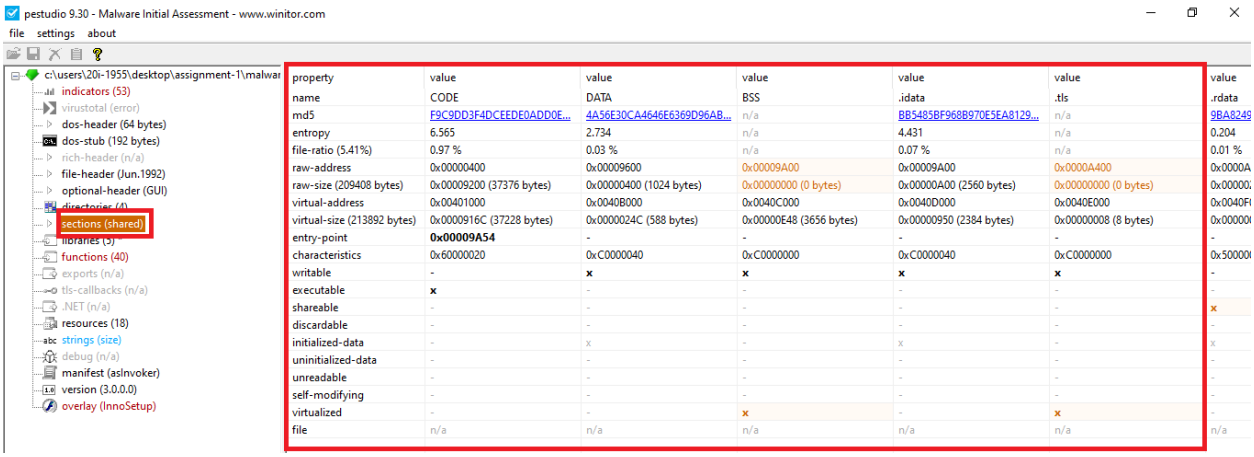

Figure 187: Malware-8 Information of Sections through PE Studio

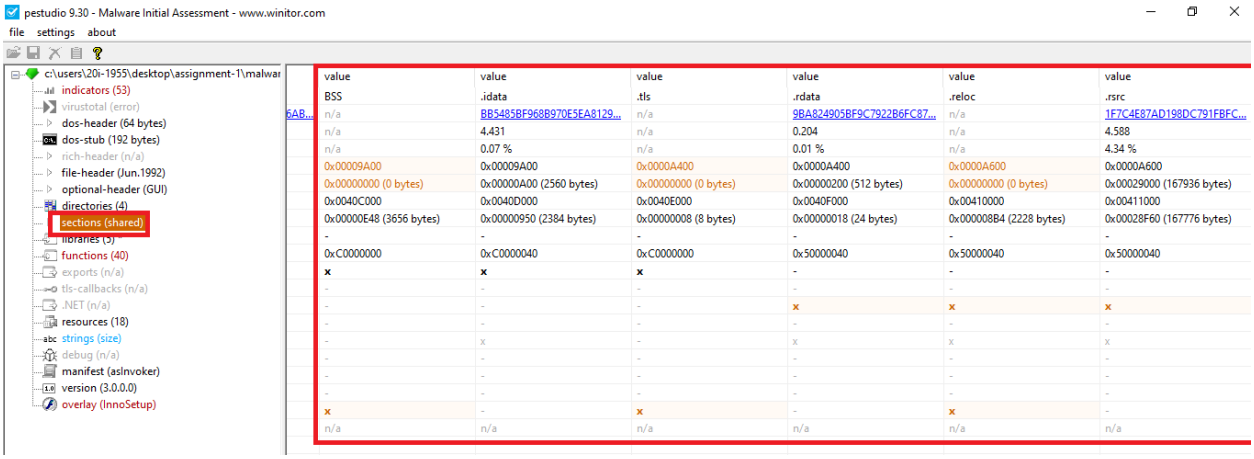

Figure 188: Malware-8 Information of Sections through PE Studio

Task-5 Which DLLs and their number of functions are used by Malware?

To find what DLLs and functions this malware used I used the tool PE Studio.

Tool-1 PE Studio

To find the DLLs and their number of functions used by malware files I open the file in the PE Studio, then from the left panel first I click on "**libraries**" and then from the right pane I found the list of DLLs and their functions used by this malware.

No. of DLL	Name	No. of functions
1	kernel32.dll	28
2	user32.dll	1
3	oleaut32.dll	5
4	advapi32.dll	5
5	comctl32.dll	1

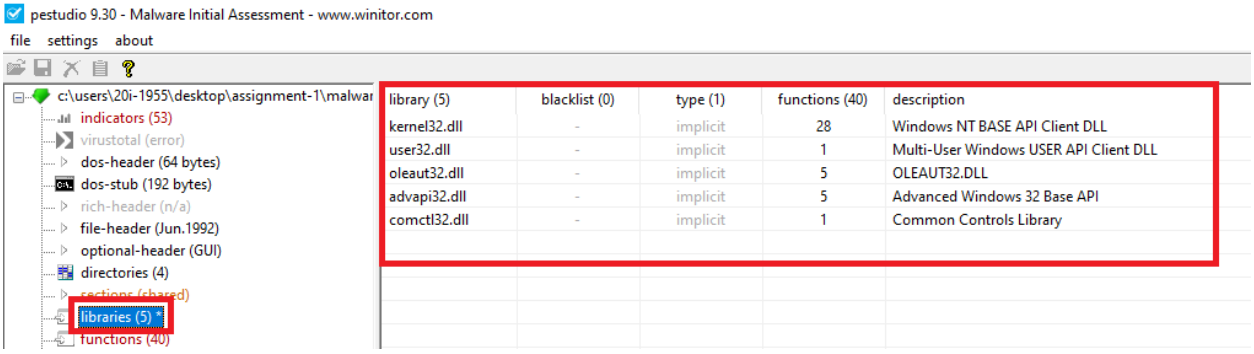

Figure 189: Malware-8 Information of DLLs and functions through PE Studio

Task-6 What is packing information if Malware is packed?

To check whether the malware is packed or not I use the tool Exeinfo.

Tool-1 Exeinfo

To find whether the malware file is packed or not I open the file in Exeinfo and at the bottom I found that the malware is packed with “Inno Setup Module [SFX]” and we can unpack it with “InnoExtractor”.

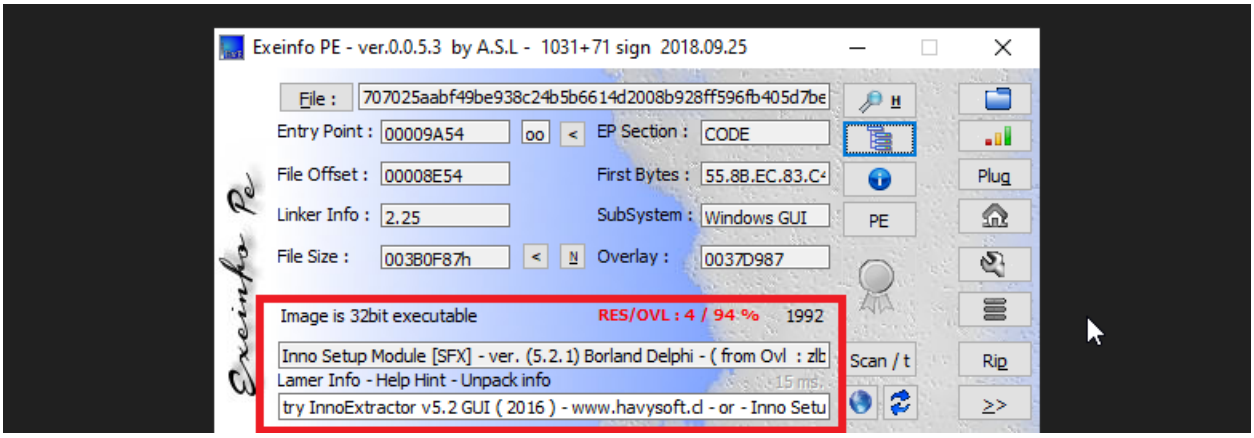

Figure 190: Malware-8 Packed or unpacked malware information through Exeinfo

Task-7 VirusTotal analysis of the malware file

To scan the malware file from the virus total I visit the website and upload the malware file. After some time, I found that surprisingly only 1 antivirus engine marked this malware file as malicious. To my surprise, I find that famous antivirus search engines failed to detect this malware.

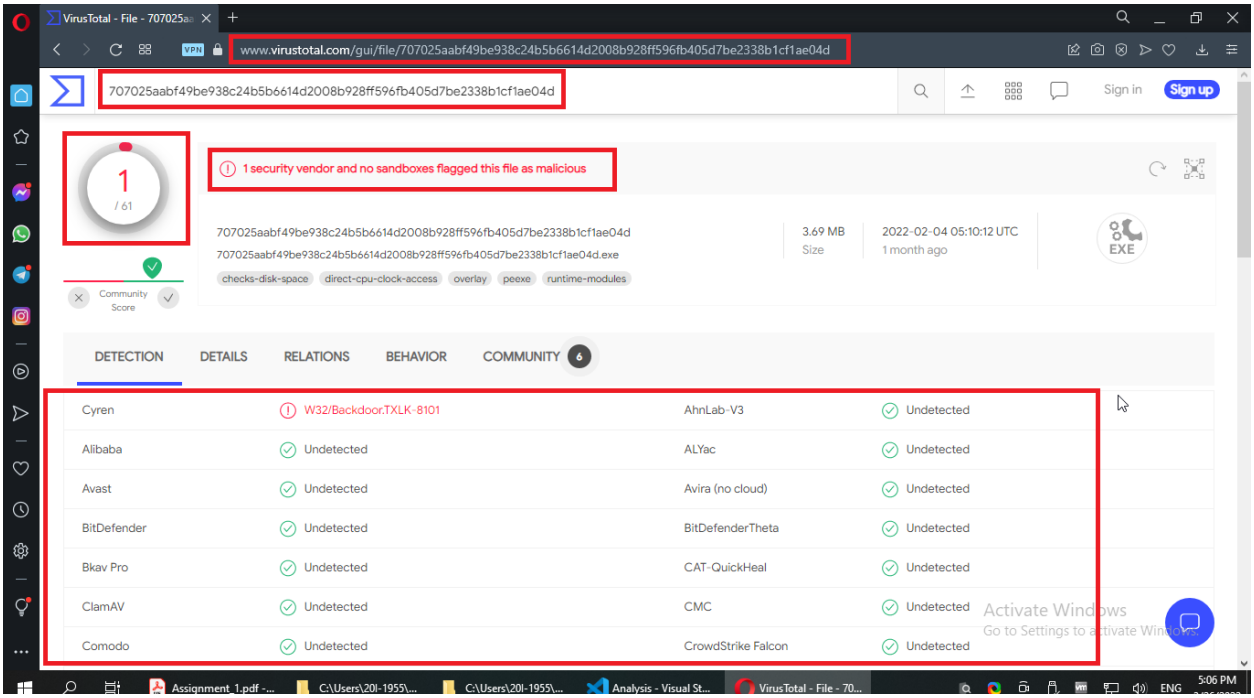

Figure 191: Malware-8 VirusTotal Analysis

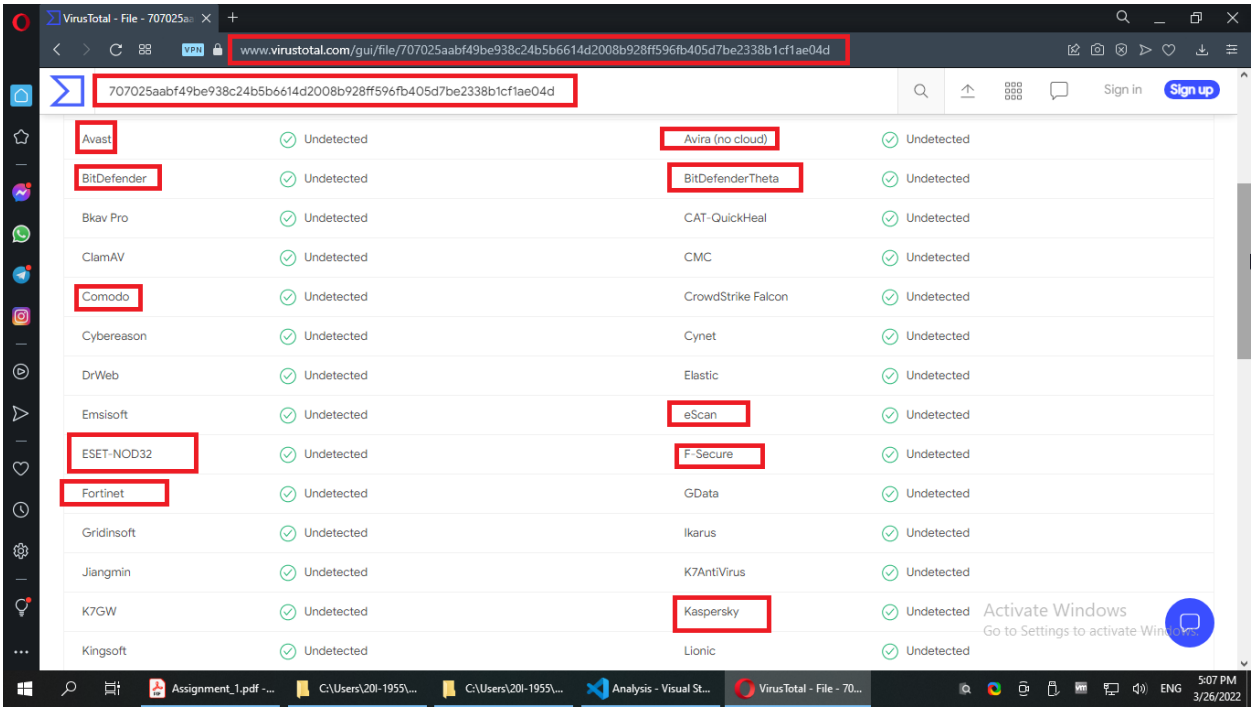

Figure 192: Malware-8 VirusTotal Analysis

Moreover, in the details tab, I find more information about the malware that I already explained in the previous tasks.

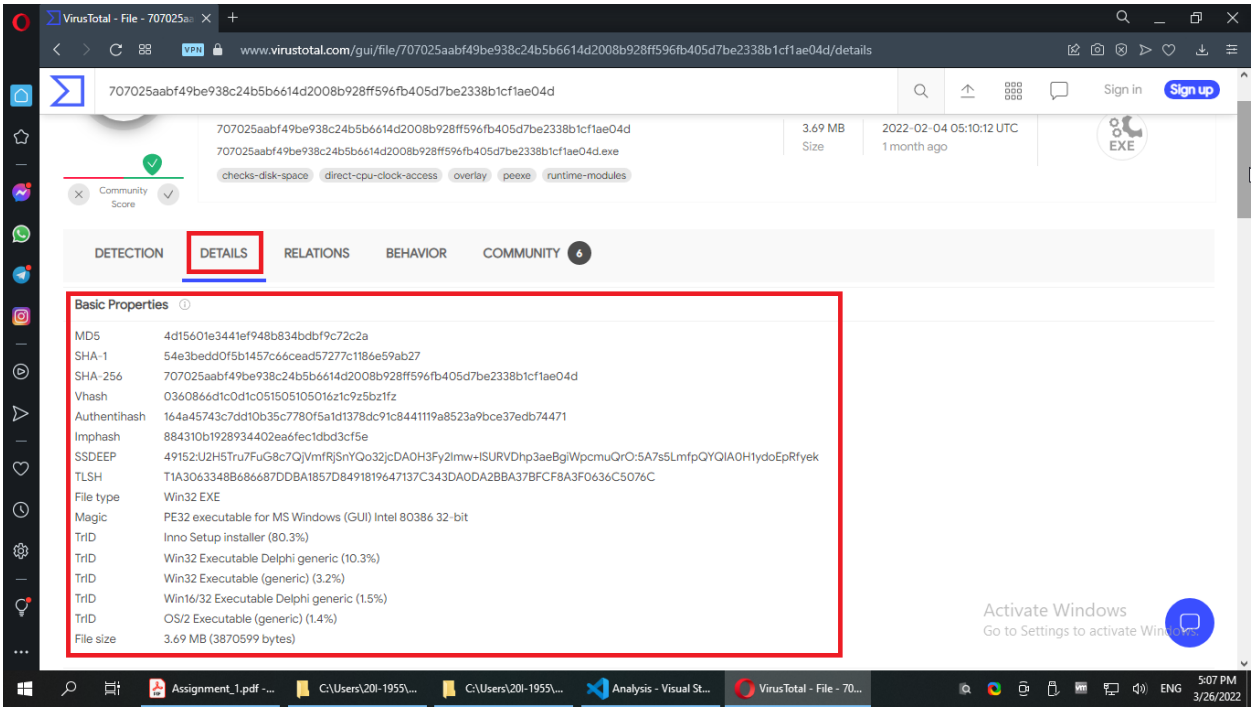

Figure 193: Malware-8 VirusTotal Analysis

Task-8 Analysis of the malware file through Hybrid Analysis

To scan the malware file from Hybrid Analysis I visit the website and upload the malware file. After some time, I found that the file is triggered as malicious by only one antivirus search engine, and VirusTotal and CrowdStrike Falcon search engines marked this malware file as clean.

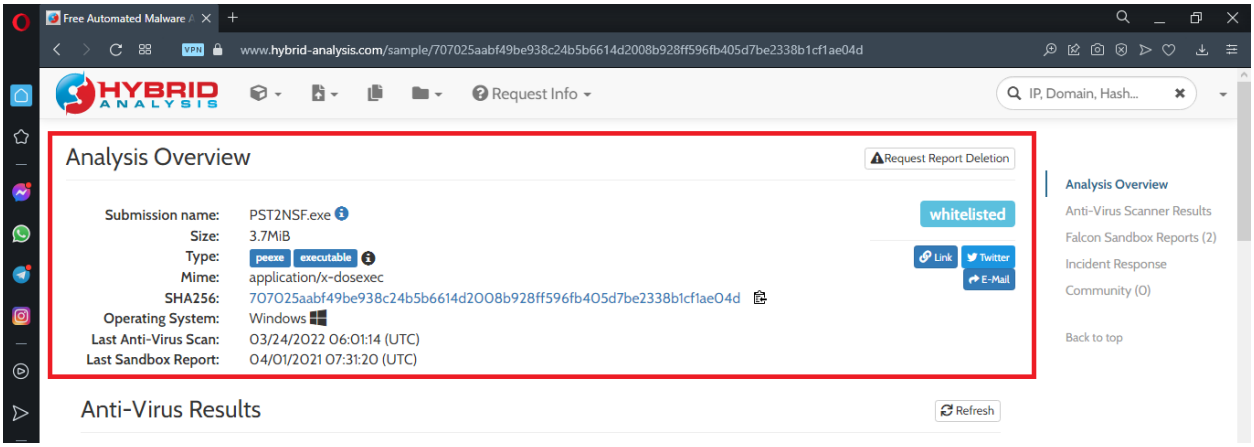

Figure 194: Malware-8 Analysis through Hybrid Analysis

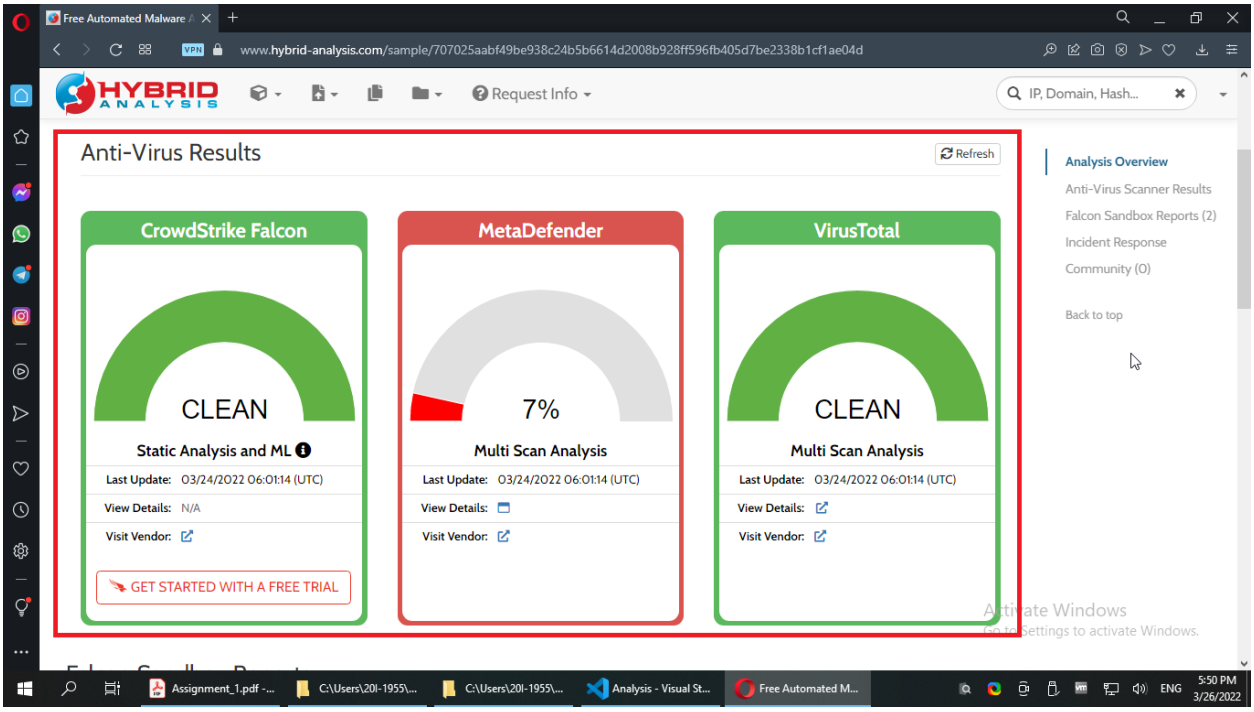

Figure 195: Malware-8 Analysis through Hybrid Analysis