



Malware Analysis and Detection

Assignment-2

Submitted by:

Muhammad Osama Khalid

20I-1955 (MSCNS)

Section: MSCNS

Table of Contents

Malware-1 W32.SecretKAN.Trojan	1
Task-1 What are the different segments or sections in case of each malware?	2
Task-2 What are different functions, imports and exports of each Malware?	3
Task-3 What is flow of functions in case of each malware? Is there any suspicious function? Give detail (name, arguments, call mechanism) of suspicious functions?	4
Task-3.1 Function FUN_004030d0.....	4
Task-3.2 FUN_00403b00.....	5
Task-3.3 FUN_004031a0	7
Task-3.4 FUN_00404a30	9
Task-3.5 FUN_00406710.....	11
Task-4 What DLL's any malware includes? Are there any suspicious functionality called by these DLL's?	12
Malware-2 Password-Stealer (003bbfec1)	13
Task-1 What are the different segments or sections in case of each malware?	15
Task-2 What are different functions, imports and exports of each Malware?	15
Task-3 What is flow of functions in case of each malware? Is there any suspicious function? Give detail (name, arguments, call mechanism) of suspicious functions?	16
Task-3.1 FUN_10005	16
Task-3.2 05 FUN_100007	18
Task-3.3 FUN_10009	19
Task-3.4 FUN_10008	21
Task-4 What DLL's any malware includes? Are there any suspicious functionality called by these DLL's?	22
Malware-3 in32Tnega.bXRKZUB	25
Task-1 What are the different segments or sections in case of each malware?	26
Task-2 What are different functions, imports and exports of each Malware?	26
Task-3 What is flow of functions in case of each malware? Is there any suspicious function? Give detail (name, arguments, call mechanism) of suspicious functions?	28
Task-3.1 FUN_00401550	28
Task-4 What DLL's any malware includes? Are there any suspicious functionality called by these DLL's?	29
Malware-4 Trojan.GenericKD.3652107	31

Task-1 What are the different segments or sections in case of each malware?	32
Task-2 What are different functions, imports and exports of each Malware?	33
Task-3 What is flow of functions in case of each malware? Is there any suspicious function? Give detail (name, arguments, call mechanism) of suspicious functions?	34
Task-3.1 FUN_crtInitCritSecNoSpinCount.....	34
Task-3.2 FUN_lock_file2.....	35
Task-3.3 FUN_0040bae8	36
Task-4 What DLL's any malware includes? Are there any suspicious functionality called by these DLL's?	37

List of Figures

Figure 1: Malware-1: Opening malware sample in Ghidra	1
Figure 2: Malware-1: Basic details exreacted by Ghidra about the malware sample	1
Figure 3: Malware-1: Ghidra auto analyze the malware	2
Figure 4: Malware-1: Magicbyte of the malware	2
Figure 5: Malware-1: Details of the sections used by the malware	3
Figure 6: Malware-1: Details of the functions, imports and exports used by malware	3
Figure 7: Malware-1: Malware intract with some programs installed on the machine	4
Figure 8: Malware-1: Programs and DLLs that is accessed by the malicious function	4
Figure 9: Malware-1: Details of Cryptomining hash algorithm.....	5
Figure 10: Malware-1: Other details of the crypto miner	5
Figure 11: Malware-1: Details of Cryptomining hash algorithm.....	6
Figure 12: Malware-1: Other details of the cryptominer	6
Figure 13: Malware-1: Other details of the cryptominer	7
Figure 14: Malware-1: Details of CPU usage of the victim machine when malware connects to nicehash.com	7
Figure 15: Malware-1: CPU usage details and the details of DLL's used by the malware	8
Figure 16: Malware-1: CPU usage details and the details of DLL's used by the malware	8
Figure 17: Malware-1: Details of the mining process	9
Figure 18: Malware-1: Details of the mining process	9
Figure 19: Malware-1: Details of the mining process	10
Figure 20: Malware-1: Details of the mining process	10
Figure 21: Malware-1: Deletes the zone file on completion of the mining process.....	11
Figure 22: Malware-1: Deleting the zone file and the DLL's used by the malware	11
Figure 23: List of DLLs used by the malware with their functions	12
Figure 24: Malware-1: Malicioys DLL fucniton used by malware	12
Figure 25: Malicioys DLL fucniton used by malware.....	13
Figure 26: Malware-2: Opening malware in Ghidra	13
Figure 27: Malware-2: Basic details exreacted by Ghidra about the malware sample	14
Figure 28: Malware-2: Ghidra auto analyze the malware	14
Figure 29: Malware-2: Magic byte and details of the sections used by malware	15
Figure 30: Malware-2: Details of the functions, imports and exports used by malware	15
Figure 31: Malware-2: Different functions used by malware	16
Figure 32: Malware-2: Malware capturing details of the host machine	16
Figure 33: Malware-2: Malware capturing details of the host machine	17
Figure 34: Malware-2: Malicious DLL fuctions used by malware	17
Figure 35: Malware-2: Gathering the details of Victims facebook account	18
Figure 36: Malware-2: Gathering the details of Victims facebook account	18
Figure 37: Malware-2: Malicious DLL fuctions used by malware	19
Figure 38: Malware-2: Malware collects the informaion of the email account of the victim	19
Figure 39: : Malware-2: Malware collects the informaion of the email account of the victim	20
Figure 40: Malware-2: Malicious DLL fuctions used by malware	20
Figure 41: Malware-2: Connects to some malicious server.....	21
Figure 42: Malware-2: Connects to some malicious server also the DLL it used durint the process	21

Figure 43: List of DLLs used by the malware with their functions	22
Figure 44: Malware-2 Malicious DLL functions used by malware	22
Figure 45: Malware-2 Malicious DLL functions used by malware	23
Figure 46: Malware-2 Malicious DLL functions used by malware	23
Figure 47: Malware-2 Malicious DLL functions used by malware	24
Figure 48: Malware-2 Malicious DLL functions used by malware	24
Figure 49: Malware-3: Opening malware in Ghidra	25
Figure 50: Malware-3: Basic details exreacted by Ghidra about the malware sample	25
Figure 51: Malware-3: Ghidra auto analyze the malware	26
Figure 52: Malware-3: Magic byte and details of the sections used by malware	26
Figure 53: Malware-3: Details of the functions, imports and exports used by malware	27
Figure 54: Malware-3: Different functions used by malware.....	27
Figure 55: Malware-3: Malware collects the information of the running process and instlling some malicious software	28
Figure 56: Malware-3: Malware installing some malicious softeare on victim mahcine	28
Figure 57: Malware-3: Malicious DLL fuctions used by malware	29
Figure 58: List of DLLs used by the malware with their functions	29
Figure 59: Malware-3 Malicious DLL functions used by malware	30
Figure 60: Malware-3 Malicious DLL functions used by malware	30
Figure 61: Malware-4: Opening malware in Ghidra	31
Figure 62: Malware-4: Basic details exreacted by Ghidra about the malware sample	31
Figure 63: Malware-4: Ghidra auto analyze the malware	32
Figure 64: Malware-4: Magic byte and details of the sections used by malware	32
Figure 65: Malware-4: Details of the functions, imports and exports used by malware	33
Figure 66: Malware-4: Different functions used by malware.....	33
Figure 67: Malware-4: Malware initialize the critical section.....	34
Figure 68: Malware-4: Malicious DLL fuctions used by malware	34
Figure 69: Malware-4: Malware enters into the critical section to perform malicious action.....	35
Figure 70: Malware-4: Malicious DLL fuctions used by malware	35
Figure 71: Malware-4: Malware deletes the critical section after performing its malicious actions.....	36
Figure 72: Malware-4: Malicious DLL fuctions used by malware	36
Figure 73: Malware-4: List of DLLs used by the malware with their functions	37
Figure 74: Malware-4: List of DLLs used by the malware with their functions	37
Figure 75: Malware-4 Malicious DLL functions used by malware	38
Figure 76: Malware-4 Malicious DLL functions used by malware	38
Figure 77: Malware-4 Malicious DLL functions used by malware	39

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 Ghidra.

Malware-1 W32.SecretKAN.Trojan

This malware belongs to the family of trojans. To analyze malware I open it in Ghidra and then perform auto analysis on the malware sample.

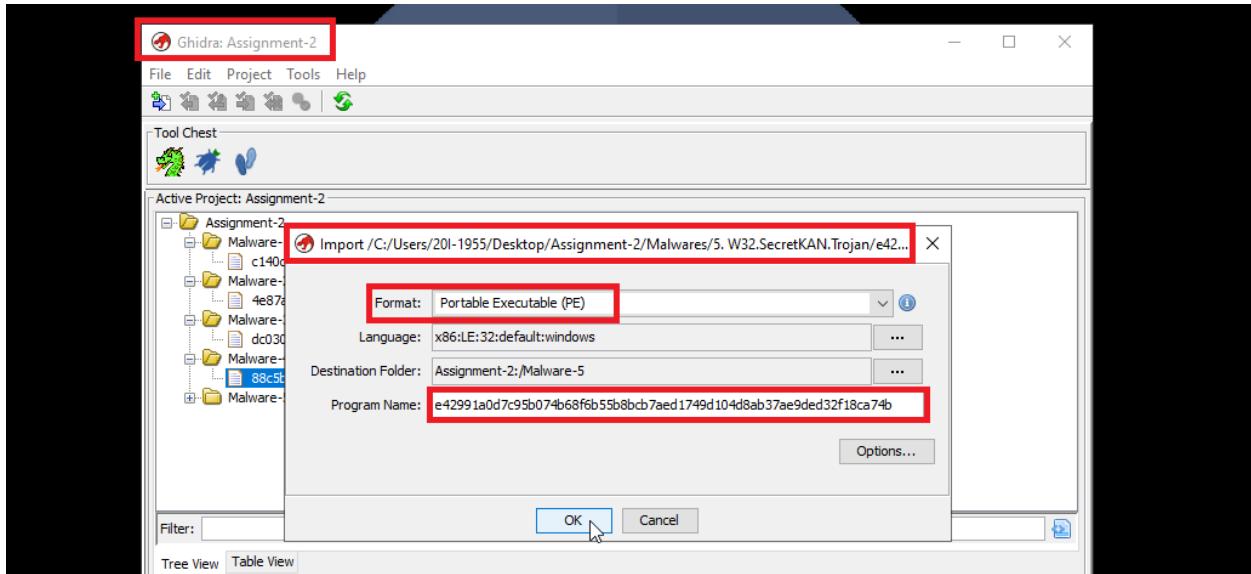


Figure 1: Malware-1: Opening malware sample in Ghidra

On opening the malware sample in the Ghidra I found that the malware is PE executable.

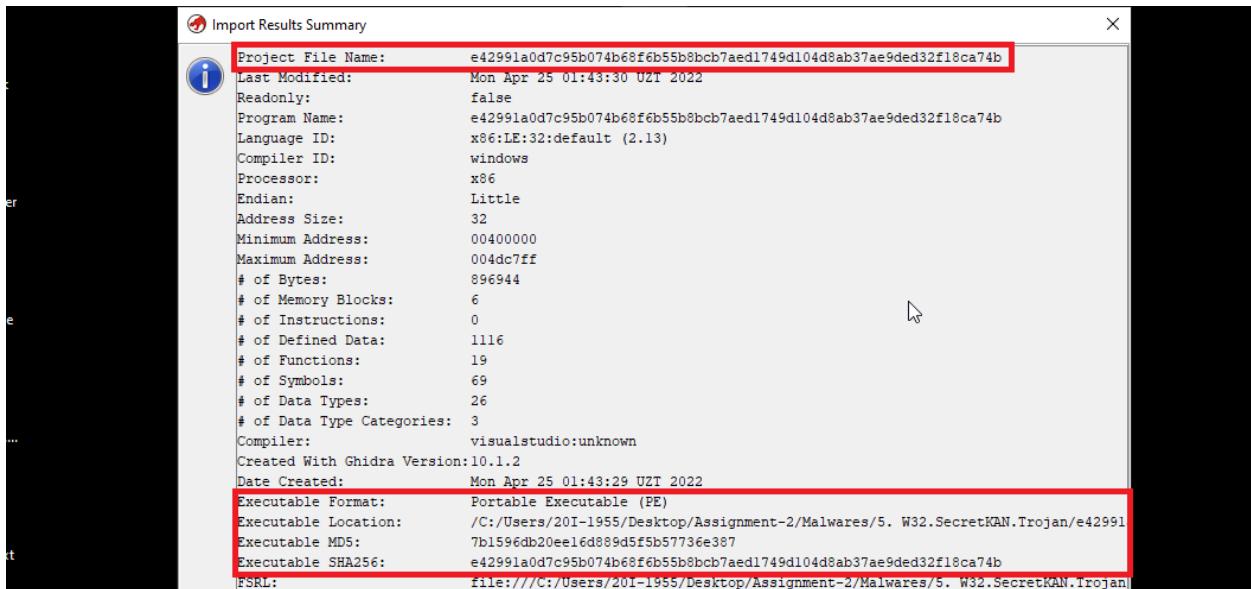


Figure 2: Malware-1: Basic details extracted by Ghidra about the malware sample

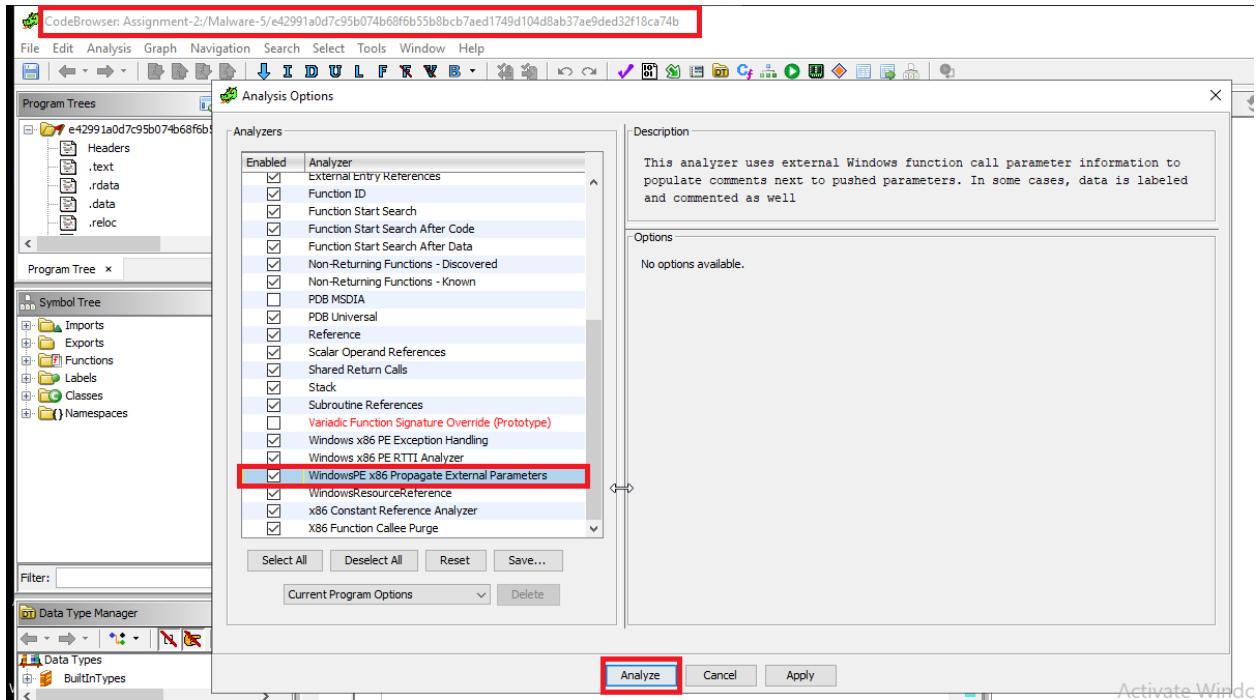


Figure 3: Malware-1: Ghidra auto analyze the malware

Task-1 What are the different segments or sections in case of each malware?

After Ghidra complete the auto analysis I found that the magic byte of the malware which is MZ, moreover I found the details of different sections used by malware that includes ".text", ".rdata", ".data", ".reloc".

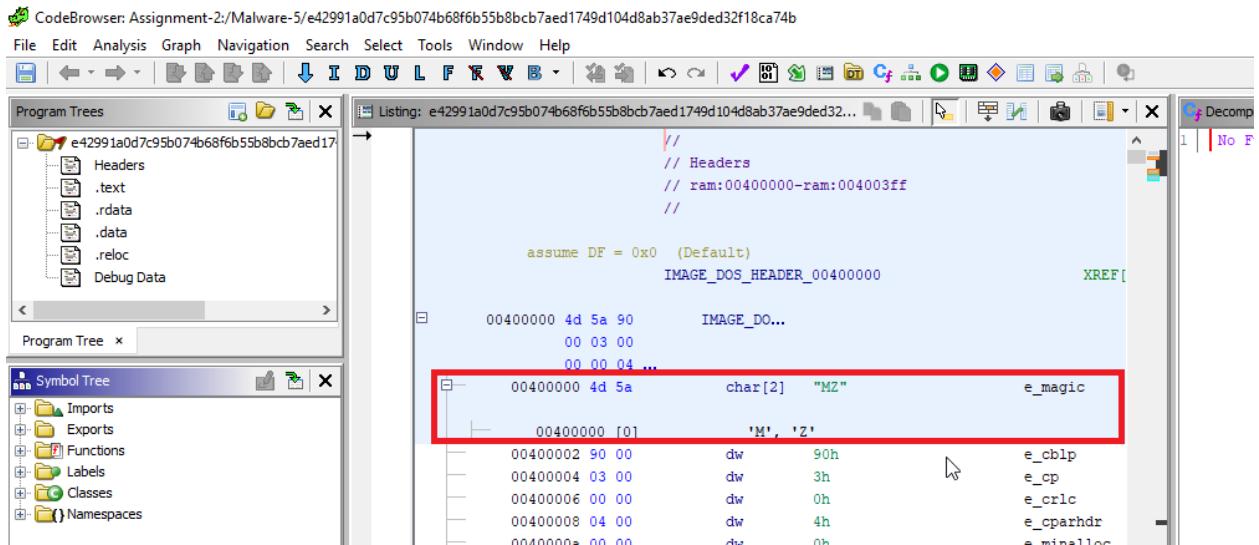


Figure 4: Malware-1: Magicbyte of the malware

CodeBrowser: Assignment-2:/Malware-5/e42991a0d7c95b074b68f6b55b8bcb7aed1749d104d8ab37ae9ded32f18

File Edit Analysis Graph Navigation Search Select Tools Window Help

Program Trees

Listing: e42991a0d7c95b074b68f6b55b8bcb7aed1749d104d8a

```

// Headers
// ram:0040
//
assume DF = 0x0 (Default)
IMAGE_DOS_H

00400000 4d 5a 90      IMAGE_I...
00 03 00
00 00 04 ...
00400000 4d 5a          char

```

Program Tree x

Symbol Tree

Figure 5: Malware-1: Details of the sections used by the malware

Task-2 What are different functions, imports and exports of each Malware?

To perform its functionality malware imports different DLL's that includes "ADVAPI32.dll", "KERNEL32.dll", "OLE32.dll", "SHELL32.dll", "USER32.dll", "WININET.dll". Moreover it also use different functions to perform its functionality.

CodeBrowser: Assignment-2:/Malware-5/e42991a0d7c95b074b68f6b55b8bcb7aed1749d104d8ab37ae9ded32f18ca74b

File Edit Analysis Graph Navigation Search Select Tools Window Help

Program Trees

Listing: e42991a0d7c95b074b68f6b55b8bcb7aed1749d104d8ab37ae9ded32...

Symbol Tree

```

// Headers
// ram:00400000-ram:004003ff
//
assume DF = 0x0 (Default)
IMAGE_DOS_HEADER_00400000 XREF[...]
00400000 4d 5a 90      IMAGE_DO...
00 03 00
00 00 04 ...
00400000 4d 5a          char[2] "MZ"           e_magic
00400000 [0]            'M', 'Z'
00400002 90 00          dw   90h           e_cblp
00400004 03 00          dw   3h            e_cp
00400006 00 00          dw   0h            e_crlc
00400008 04 00          dw   4h            e_cparhdr
0040000a 00 00          dw   0h            e_minalloc
0040000c ff ff          dw   FFFFh         e_maxalloc
0040000e 00 00          dw   0h            e_ss
00400010 b8 00          dw   B8h           e_sp
00400012 00 00          dw   0h            e_csum
00400014 00 00          dw   0h            e_ip
00400016 nn nn          dw   nn             e_ip

```

Figure 6: Malware-1: Details of the functions, imports and exports used by malware

Task-3 What is flow of functions in case of each malware? Is there any suspicious function?

Give detail (name, arguments, call mechanism) of suspicious functions?

The flow of the funcitons of this malware is that if first interact with the system programs like "notepad.exe", "explorer.exe", "svchost.exe", etc. then connects the computer to some minning pool to perfrom crypto mining also checking the CPU running status. After perform the mining it connect backs to the attacker to send the results back to him/her. Moreover after completing the mining process it deletes the zone file.

Task-3.1 Function FUN_004030d0

In this function malare interact with the system programs which includes "notepad.exe", "explorer.exe", "svchost.exe", etc.

```

CodeBrowser: Assignment-2-/Malware-5/e42991a0d7c95b074b68f6b55b8bcd7aed1749d104d8ab37ae9ded32f18ca74b
File Edit Analysis Graph Navigation Search Select Tools Window Help
I D U L F R B X Listing: e42991a0d7c95b074b68f6b55b8bcd7aed1...
Decompile: FUN_004030d0 - (e42991a0d7c95b074b68f6b55b8bcd7aed1749d104d8ab37ae9ded32f18ca74b)
Program Trees X
e42991a0d7c95b074b68f6b55b8bcd7aed1749d104d8ab37ae9ded32f18ca74b
Headers
.text
.rdata
.data
.reloc
Debug Data
Symbol Tree X
Imports
Exports
entry
Functions
entry
FUN_004040
FUN_004041
FUN_004042
FUN_004043
FUN_00404030
FUN_0040318
FUN_0040340
FUN_004030d0
FUN_00403180
FUN_00403400
Filter:
Listed symbols:
004030c4 cc ?? CCh
004030c5 cc ?? CCh
004030c6 cc ?? CCh
004030c7 cc ?? CCh
004030c8 cc ?? CCh
004030c9 cc ?? CCh
004030ca cc ?? CCh
004030cb cc ?? CCh
004030cc cc ?? CCh
004030cd cc ?? CCh
004030ce cc ?? CCh
004030cf cc ?? CCh
***** FUNCTION *****
|undefined4 __cdecl FUN_004030d0 (LPCWSTR param_1)
+-----+
undefined4 EAX=0x0 <RETN>
LPCWSTR Stack[0x4]:4 param_1
undefined1 Stack[-0x20c... local_20c
FUN_004030d0
004030d0 55 PUSH EBP
004030d1 8b ec MOV EBP,ESP
004030d3 81 ec 08 SUB ESP,0x208
Decompile:
1
2 undefined4 __cdecl FUN_004030d0 (LPCWSTR param_1)
3
4 {
5     UINT iVar1;
6     int iVar2;
7     wchar_t *pwVar3;
8     wchar_t *pwVar4;
9     WCHAR local_20c [260];
10
11     if (DAT_004da4a4 == 0) {
12         if (DAT_004da4a8 == 0) {
13             pwVar3 = L"\\""/System32\wuapp.exe";
14             pwVar4 = L"\\""/System32\svchost.exe";
15             goto LAB_00403118;
16         }
17         else if (DAT_004da4a8 == 0) {
18             pwVar3 = L"\\""/notepad.exe";
19             pwVar4 = L"\\""/explorer.exe";
20             goto LAB_00403118;
21         }
22         else {
23             pwVar3 = L"\\""/SysROW64\wuapp.exe";
24             pwVar4 = L"\\""/SysROW64\svchost.exe";
25             LAB_00403118:
26             iVar1 = GetWindowsDirectoryW(local_20c,0x104);
}

```

Figure 7: Malware-1: Malware interact with some programs installed on the machine

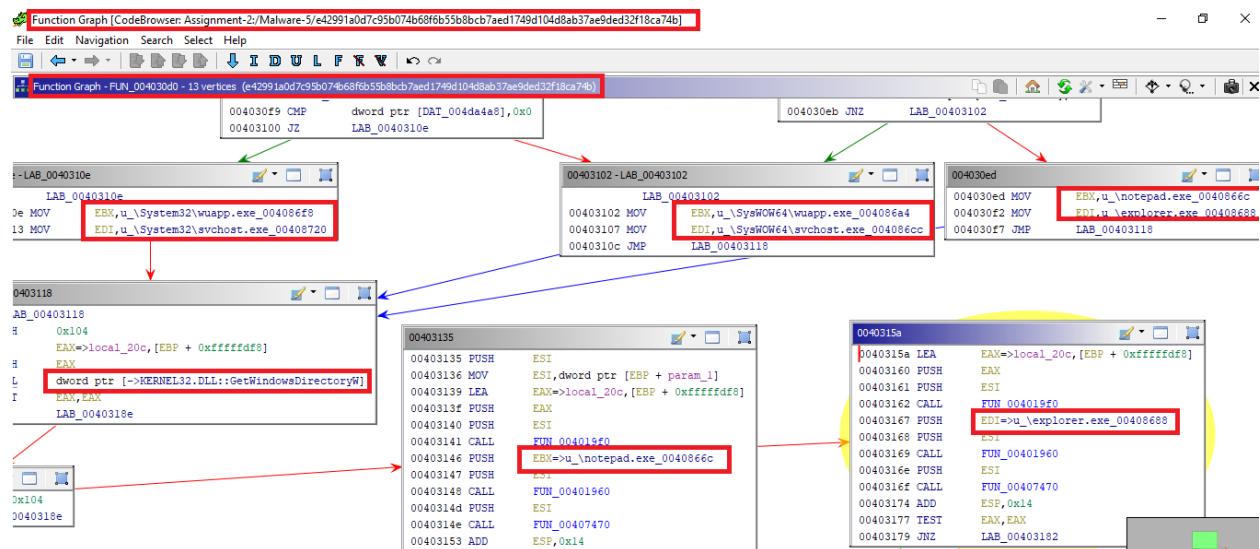


Figure 8: Malware-1: Programs and DLLs that is accessed by the malicious function

Task-3.2 FUN_00403b00

In this function I found the details of the cryptomining that malware performs ranging from cryptocurrency mining hashing algorithms “CryptoNight”, “CryptoNight-lite”, “CryptoNight-heavy” that is used by certain Proof of Work blockchains. Also I found the details of the mining pool, api, worker-id, access token etc.

```

CodeBrowser: Assignment-2/Malware-5/e42991a0d7c95b074b68f6b55b8bcd7aed1749d104d8ab37ae9ded32f18ca74b
File Edit Analysis Graph Navigation Search Select Tools Window Help
D U L F K V B C G P S T M I
Program Trees Listing: e42991a0d7c95b074b68f6b55b8bcd7aed1749d104d8ab37ae9ded32f18ca74b...
e42991a0d7c95b074b68f6b55b8bcd7aed1749d104d8ab37ae9ded32f18ca74b
Headers .text .data .data .reloc Debug Data
* FUNCTION
undefined4 _cdcc1 FUN_00403b00 int
EAX:4 <RETURN>
Stack[0x4]:4 param_1
char * Stack[0x8]:4 param_2
char * Stack[0xc]:4 param_3
char * Stack[0x10]:4 param_4
int Stack[0x14]:4 param_5
undefined1 Stack[-0xc]:1 local_c
undefined8 Stack[-0x14]:8 local_14
undefined4 Stack[-0x18]:4 local_18
undefined4 Stack[-0x1c]:4 local_lc
undefined4 Stack[-0x20]:4 local_20
undefined1[16] Stack[-0x30]... local_30
FUN_004017d0((int)local_468,"(\r\n\t\"algo\": \\"cryptonight\\",");
if ((DAT_004dab48 == 3) || (DAT_004dab48 == 4)) {
    FUN_004017d0((int)local_468,"(\r\n\t\"algo\": \\"cryptonight-lite\\",");
}
if (DAT_004dab48 == 5) {
    FUN_004017d0((int)local_468,"(\r\n\t\"algo\": \\"cryptonight-heavy\\",");
}
FUN_004016d0(local_468,"(\r\n\t\"background\": false,");
FUN_004016d0(local_468,"(\r\n\t\"colors\": true,");
FUN_004016d0(local_468,"(\r\n\t\"retry\": 5,");
FUN_004016d0(local_468,"(\r\n\t\"retry-pause\": 5,");
FUN_004016d0(local_468,"(\r\n\t\"syslog\": false,");
FUN_004016d0(local_468,"(\r\n\t\"print-time\": 60,");
FUN_004016d0(local_468,"(\r\n\t\"av\": 0,");
FUN_004016d0(local_468,"(\r\n\t\"safe\": false,");
FUN_004016d0(local_468,"(\r\n\t\"cpu-priority\": null,");
FUN_004016d0(local_468,"(\r\n\t\"cpu-affinity\": null,");
FUN_004016d0(local_468,"(\r\n\t\"threads\": ");
FUN_004016d0(local_468,"local_c");
FUN_004016d0(local_468,"(\r\n\t\"pools\": ");
FUN_004016d0(local_468,"(\r\n\t\t\"url\": \"");
FUN_004016d0(local_468,"param_2");
FUN_004016d0(local_468,"(\r\n\t\t\"");
FUN_004016d0(local_468,"(\r\n\t\t\t\"user\": \"");
FUN_004016d0(local_468,"param_3");
FUN_004016d0(local_468,"(\r\n\t\t\t\"");
FUN_004016d0(local_468,"(\r\n\t\t\t\t\"pass\": \"");
FUN_004016d0(local_468,"(\r\n\t\t\t\t\t\"");

```

Figure 9: Malware-1: Details of Cryptomining hash algorithm

```

CodeBrowser: Assignment-2/Malware-5/e42991a0d7c95b074b68f6b55b8bcd7aed1749d104d8ab37ae9ded32f18ca74b
File Edit Analysis Graph Navigation Search Select Tools Window Help
D U L F K V B C G P S T M I
Program Trees Listing: e42991a0d7c95b074b68f6b55b8bcd7aed1749d104d8ab37ae9ded32f18ca74b...
e42991a0d7c95b074b68f6b55b8bcd7aed1749d104d8ab37ae9ded32f18ca74b
Headers .text .data .data .reloc Debug Data
* FUNCTION
undefined4 _cdcc1 FUN_00403b00 int
EAX:4 <RETURN>
Stack[0x4]:4 param_1
char * Stack[0x8]:4 param_2
char * Stack[0xc]:4 param_3
char * Stack[0x10]:4 param_4
int Stack[0x14]:4 param_5
undefined1 Stack[-0xc]:1 local_c
undefined8 Stack[-0x14]:8 local_14
undefined4 Stack[-0x18]:4 local_18
undefined4 Stack[-0x1c]:4 local_lc
undefined4 Stack[-0x20]:4 local_20
FUN_004016d0(local_468,"(\r\n\t\"pass\": \"");
FUN_004016d0(local_468,param_4);
FUN_004016d0(local_468,"(\r\n\t");
if ((DAT_004dab48 == 0) {
    pcVar5 = "(\r\n\t\t\"keepalive\": false";
} else {
    pcVar5 = "(\r\n\t\t\"keepalive\": true";
}
FUN_004016d0(local_468,pcVar5);
if (param_5 == 0) {
    pcVar5 = "(\r\n\t\t\t\"nicehash\": false";
} else {
    pcVar5 = "(\r\n\t\t\t\"nicehash\": true";
}
FUN_004016d0(local_468,pcVar5);
if ((DAT_004dab48 == 2) || (DAT_004dab48 == 3)) {
    FUN_004016d0(local_468,"(\r\n\t\t\t\"variant\": 0");
}
if (((DAT_004dab48 == 1) || (DAT_004dab48 == 4)) || (DAT_004dab48 == 5)) {
    FUN_004016d0(local_468,"(\r\n\t\t\t\"variant\": 1");
}
FUN_004016d0(local_468,"(\r\n\t\t\t\t\"");
FUN_004016d0(local_468,"(\r\n\t\t\t\t");
FUN_004016d0(local_468,"(\r\n\t\t\t\t\t\"api\": (");
FUN_004016d0(local_468,"(\r\n\t\t\t\t\t\"port\": 0,");
FUN_004016d0(local_468,"(\r\n\t\t\t\t\t\"access-token\": null,");
FUN_004016d0(local_468,"(\r\n\t\t\t\t\t\"worker-id\": null");

```

Figure 10: Malware-1: Other details of the crypto miner

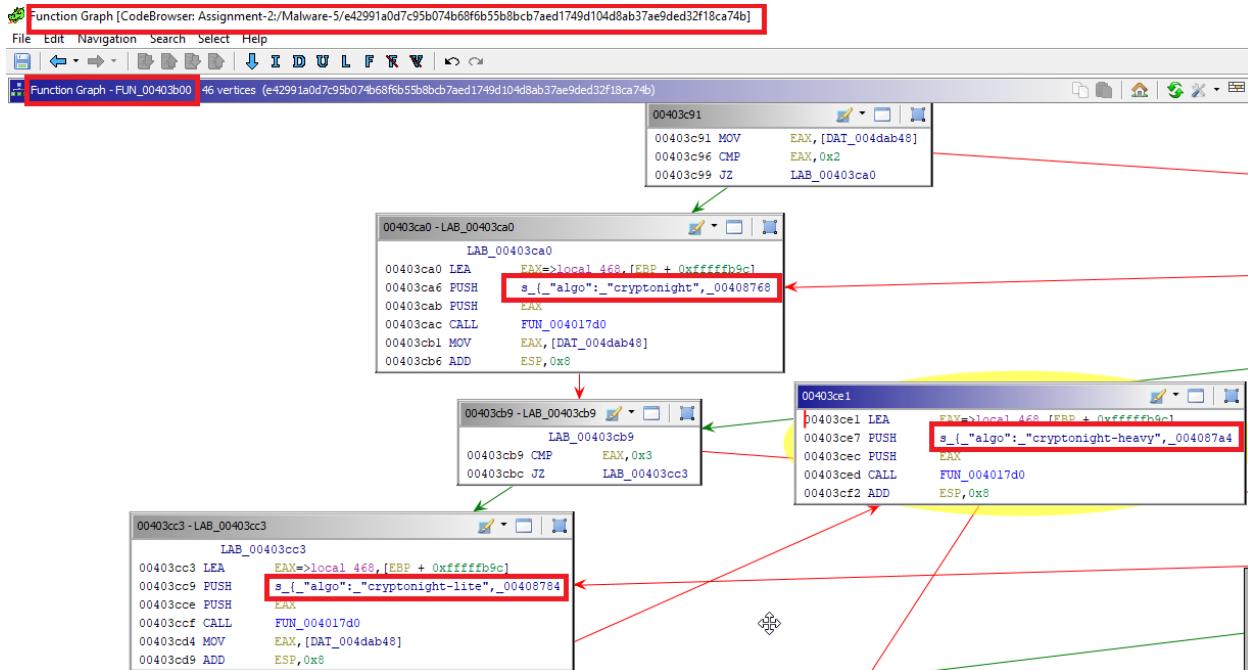


Figure 11: Malware-1: Details of Cryptomining hash algorithm

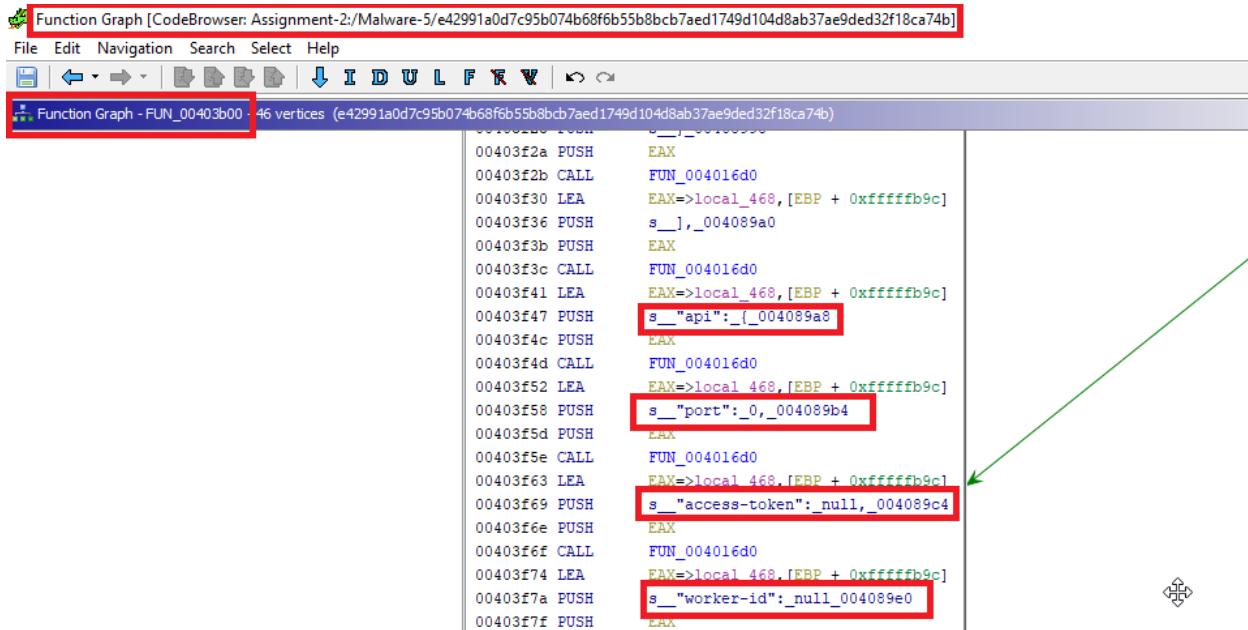


Figure 12: Malware-1: Other details of the cryptominer

```

00403d80 LEA    EAX=>local_468,[EBP + 0xfffffb9c]
00403d86 PUSH   s__"cpu-priority":_null,_004088e0
00403d8b PUSH   EAX
00403d8c CALL   FUN_004016d0
00403d91 LEA    EAX=>local_468,[EBP + 0xfffffb9c]
00403d97 PUSH   s__"cpu-affinity":_null,_00408888
00403d9c PUSH   EAX
00403d9d CALL   FUN_004016d0
00403da2 LEA    EAX=>local_468,[EBP + 0xfffffb9c]
00403da8 PUSH   s__"threads":_004088a4
00403dad PUSH   EAX
00403dae CALL   FUN_004016d0
00403db3 LEA    EAX=>local_c,[EBP + -0x8]
00403dbe PUSH   EAX
00403dc0 CALL   FUN_004016d0
00403dc3 LEA    EAX=>local_468,[EBP + 0xfffffb9c]
00403dc9 PUSH   DAT_004088b4
00403dce PUSH   EAX
00403dcf CALL   FUN_004016d0
00403dd4 LEA    EAX=>local_468,[EBP + 0xfffffb9c]
00403dda PUSH   s__"pools":_004088b8
00403ddf PUSH   EAX
00403de0 CALL   FUN_004016d0
00403de5 LEA    EAX=>local_468,[EBP + 0xfffffb9c]
00403deb PUSH   s__"l_004088c8
00403df0 PUSH   EAX
00403df1 CALL   FUN_004016d0
00403df6 LEA    EAX=>local_468,[EBP + 0xfffffb9c]
00403dfc PUSH   s__"url":_004088d0
00403d01 RETW

```

Figure 13: Malware-1: Other details of the cryptominer

Task-3.3 FUN_004031a0

In this function I found the detail of the CPU usage used by the malware while it connects to “nicehash.com” which is a cryptocurrency mining platform on which it connects the sellers that have hashing power with the buyers that want hashing power to mine their cryptocurrencies.

```

TEST    EAX,EAX
PUSH    s_TASKMGR_00408b20
CMOVZ   ECX,ESI
PUSH    s_TASKMGR_00408b28
MOV     dword ptr [DAT_004da490],ECX
CALL    FUN_004017a0
MOV     ECX,dword ptr [DAT_004da494]
TEST    EAX,EAX
PUSH    s_50%CPU_00408b30
CMOVZ   ECX,ESI
PUSH    s_1THREAD_00408b3c
MOV     dword ptr [DAT_004da494],ECX
CALL    FUN_004017a0
MOV     ECX,dword ptr [DAT_004da498]
TEST    EAX,EAX
PUSH    s_50%CPU_00408b44
CMOVZ   ECX,ESI
PUSH    s_50%CPU_00408b50
MOV     dword ptr [DAT_004da498],ECX

```

```

56 iVar6 = FUN_004017a0((DAT_00408b18,&DAT_00408b10));
57 if (iVar6 == 0) {
58     DAT_004da490 = 1;
59 }
60 iVar6 = FUN_004017a0((byte *)"TASKMGR", (byte *)"TASKMGR");
61 if (iVar6 == 0) {
62     DAT_004da494 = 1;
63 }
64 iVar6 = FUN_004017a0((byte *)"1THREAD", (byte *)"50%CPU");
65 if (iVar6 == 0) {
66     DAT_004da498 = 1;
67 }
68 iVar6 = FUN_004017a0((byte *)"50%CPU", (byte *)"50%CPU");
69 if (iVar6 == 0) {
70     DAT_004da498 = 2;
71 }
72 iVar6 = FUN_004017a0((byte *)"100%CPU", (byte *)"100%CPU");
73 if (iVar6 == 0) {
74     DAT_004da49c = 1;
75 }
76 DAT_004da4b0 = 0xle;
77 FUN_00401a90((DAT_004da2c8,0,0x100);
78 __sanitizer::internal_memcpy((DAT_004da2c8,&DAT_00408b80,DAT_004da3c8
79 FUN_00401b00((int)"0125789244697858",0x10,(int)&DAT_004da2c8,DAT_004da
80 uVar3 = FUN_00401a90((int)&DAT_004da2c8,DAT_004da3c0);
81 FUN_00401a90((int)&DAT_004da3cc,0x80);
82 __sanitizer::internal_memcpy((DAT_004da3cc,&DAT_00408c30,DAT_004da44c
83 FUN_00401b00((int)"0125789244697858",0x10,(int)&DAT_004da3cc,DAT_004da
84 uVar4 = FUN_00401a90((int)&DAT_004da3cc,DAT_004da44c);
85 if ((uVar3 == 0x9532289) && (uVar4 == 0x1e303451)) {
86     pcVar5 = FUN_004018c0((DAT_004da3cc,"nicehash.com"));
87     if (pcVar5 != (char *)0x0) {

```

Figure 14: Malware-1: Details of CPU usage of the victim machine when malware connects to nicehash.com

Figure 15: Malware-1: CPU usage details and the details of DLL's used by the malware

Figure 16: Malware-1: CPU usage details and the details of DLL's used by the malware

Task-3.4 FUN_00404a30

In this function I found the details of the mining address, mining pool, miner proxy and miner password used by the cryptominer to mine cryptocurrency.

```

1 undefined4 __cdecl FUN_00404a30(int param_1, byte *param_2, byte *param_3, byte *param_4)
2 
3 {
4     int iVar1;
5     char *pcVar2;
6     uint uVar3;
7 
8     iVar1 = FUN_004017a0(param_2, (byte *)"Miner");
9     if ((iVar1 == 0) && (iVar1 = FUN_004017a0(param_3, (byte *)"address")) != 0) {
10        pcVar2 = (char *)FUN_004017f0((char *)param_4);
11        uVar3 = FUN_00401840(pcVar2);
12        if (uVar3 < 0x100) {
13            FUN_004017d0(param_1 + 0x4c8, pcVar2);
14        }
15        FUN_00401500(pcVar2);
16    }
17 
18    iVar1 = FUN_004017a0(param_2, (byte *)"Miner");
19    if ((iVar1 == 0) && (iVar1 = FUN_004017a0(param_3, (byte *)"poolport")) != 0) {
20        pcVar2 = (char *)FUN_004017f0((char *)param_4);
21        uVar3 = FUN_00401840(pcVar2);
22        if (uVar3 < 0x80) {
23            FUN_004017d0(param_1, pcVar2);
24        }
25        FUN_00401500(pcVar2);
26        return 1;
27    }
28 
29    iVar1 = FUN_004017a0(param_2, (byte *)"Miner");
30    if ((iVar1 == 0) && (iVar1 = FUN_004017a0(param_3, (byte *)"password")) != 0) {
31        pcVar2 = (char *)FUN_004017f0((char *)param_4);
32        uVar3 = FUN_00401840(pcVar2);
33        if (uVar3 < 0x40) {
34            FUN_004017d0(param_1, pcVar2);
35        }
36        FUN_00401500(pcVar2);
37        return 1;
38    }
39 }

```

Figure 17: Malware-1: Details of the mining process

```

58 iVar1 = FUN_004017a0(param_2, (byte *)"Update");
59 if ((iVar1 == 0) && (iVar1 = FUN_004017a0(param_3, (byte *)"update_url")) != 0) {
60     pcVar2 = (char *)FUN_004017f0((char *)param_4);
61     uVar3 = FUN_00401840(pcVar2);
62     if (uVar3 < 0x200) {
63         FUN_004017d0(param_1 + 0xc4, pcVar2);
64     }
65     FUN_00401500(pcVar2);
66     return 1;
67 }
68 
69 iVar1 = FUN_004017a0(param_2, (byte *)"Update");
70 if ((iVar1 == 0) && (iVar1 = FUN_004017a0(param_3, (byte *)"config_url")) != 0) {
71     pcVar2 = (char *)FUN_004017f0((char *)param_4);
72     uVar3 = FUN_00401840(pcVar2);
73     if (uVar3 < 0x200) {
74         FUN_004017d0(param_1 + 0x2c4, pcVar2);
75     }
76     FUN_00401500(pcVar2);
77     return 1;
78 }
79 
80 iVar1 = FUN_004017a0(param_2, (byte *)"Update");
81 if ((iVar1 == 0) && (iVar1 = FUN_004017a0(param_3, (byte *)"knock_time")) != 0) {
82     pcVar2 = (char *)FUN_004017f0((char *)param_4);
83     if ((char)uVar3 == '\0') {
84         return 1;
85     }
86     iVar1 = FUN_004014c0((char *)param_4);
87     *(int *)param_1 + 0x4c4) = iVar1;
88     return 1;
89 }

```

Figure 18: Malware-1: Details of the mining process

The screenshot shows the Ghidra interface with several windows open:

- Program Trees**: Shows the file structure of the malware binary.
- Listing**: Displays the assembly code for function `00404a30`. It includes instructions like `undefined4 cdec1 FUN_00404a30(int param_1, byte * param_2)` and `Stack[0x4]:4 param_1`.
- Decompiler**: Shows the decompiled C-like pseudocode for the same function. It includes conditional logic based on parameters `iVar1` and `iVar2`, and function calls like `FUN_004017a0`.
- Symbol Tree**: Lists various symbols and functions used in the code.
- Data Type Manager**: Shows the data types defined in the project.
- Registers**: Shows the current state of registers EAX, EBX, ECX, ESP, and ECSP.
- Call Graph**: Shows the call graph for the function `00404a30`.
- Function Call History**: Shows a history of function calls.
- Function Graph**: Shows the function graph for `00404a30`.

Figure 19: Malware-1: Details of the mining process

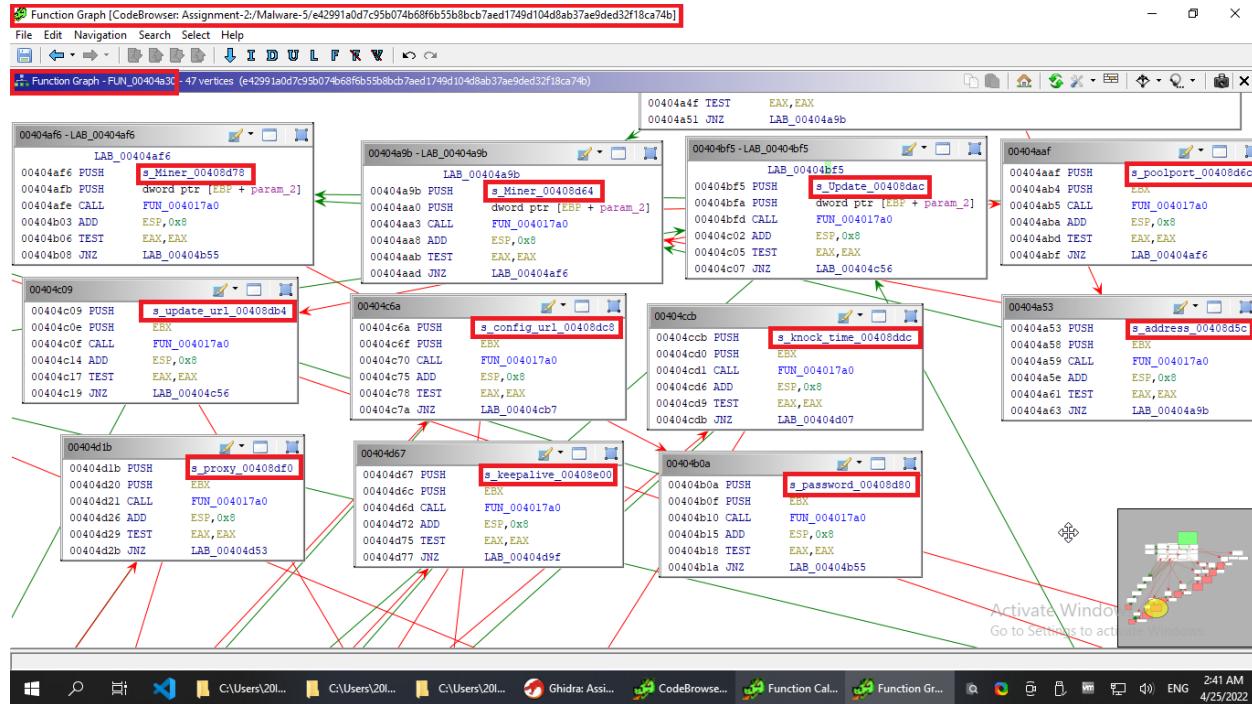


Figure 20: Malware-1: Details of the mining process

Task-3.5 FUN_00406710

After that I found that when the mining process completes malware deletes the zone file used by it.

```

75     local_3c = 0x18;
76     local_38._0_8_ = SUB128(CONCAT84(local_38._0_8_,&1));
77     local_38._0_12_ = CONCAT48(0x40,local_38._0_8_);
78     local_38 = ZEXT1216(local_38._0_12_);
79     local_28 = 0;
80     iVar3 = (*DAT_004da244)(local_c,0x120116,&local_34);
81 ;
82     if (iVar3 == 0) {
83         local_14 = iVar3;
84         local_10 = iVar3;
85         iVar3 = (*DAT_004da250)(local_c,0,0,&local_24);
86         if (iVar3 == 0) {
87             (*DAT_004da218)(local_c);
88             VirtualFree(lphddress,0,0x8000);
89             FUN_004019f0((int)local_25c,param_1);
90             FUN_004019e0(local_25c,L"Zone.Identifier");
91             DeleteFileW(local_25c);
92             return true;
93         }
94     }
95     (*DAT_004da218)();

```

Figure 21: Malware-1: Deletes the zone file on completion of the mining process

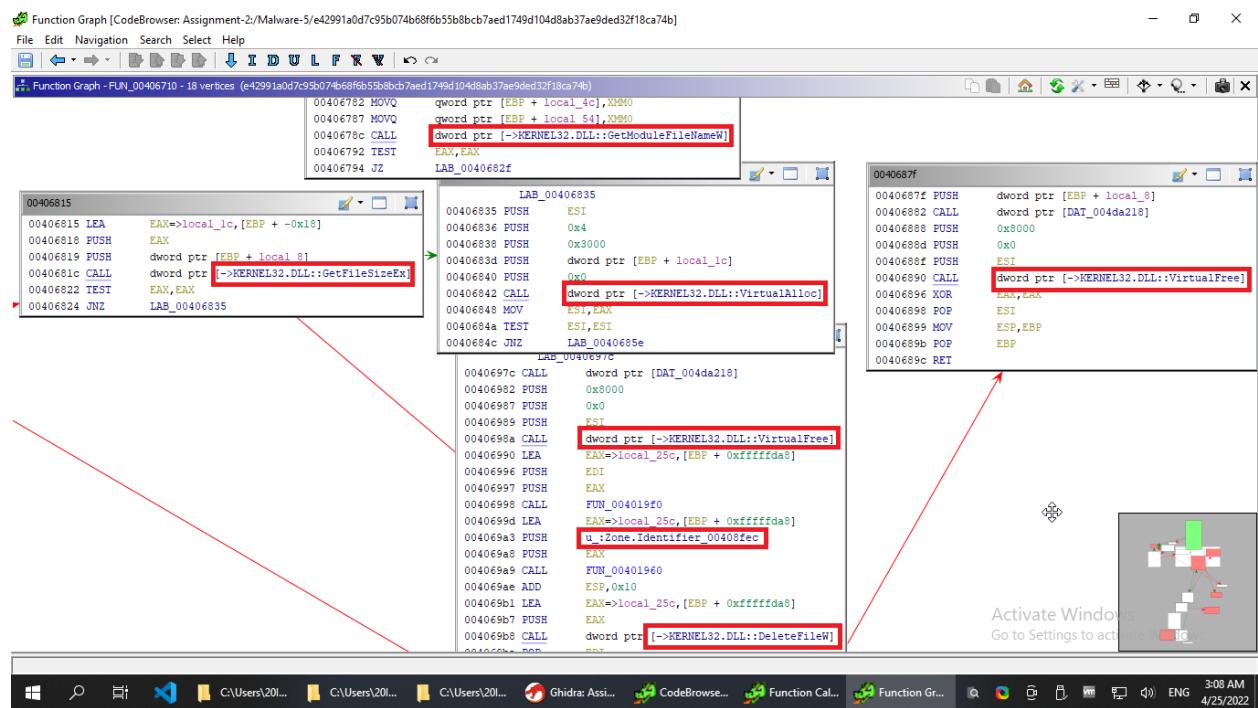


Figure 22: Malware-1: Deleting the zone file and the DLL's used by the malware

Task-4 What DLL's any malware includes? Are there any suspicious functionality called by these DLL's?

The DLL's used by this malware includes "ADVAPI32.dll", "KERNEL32.dll", "OLE32.dll", "SHELL32.dll", "USER32.dll", "WININET.dll".

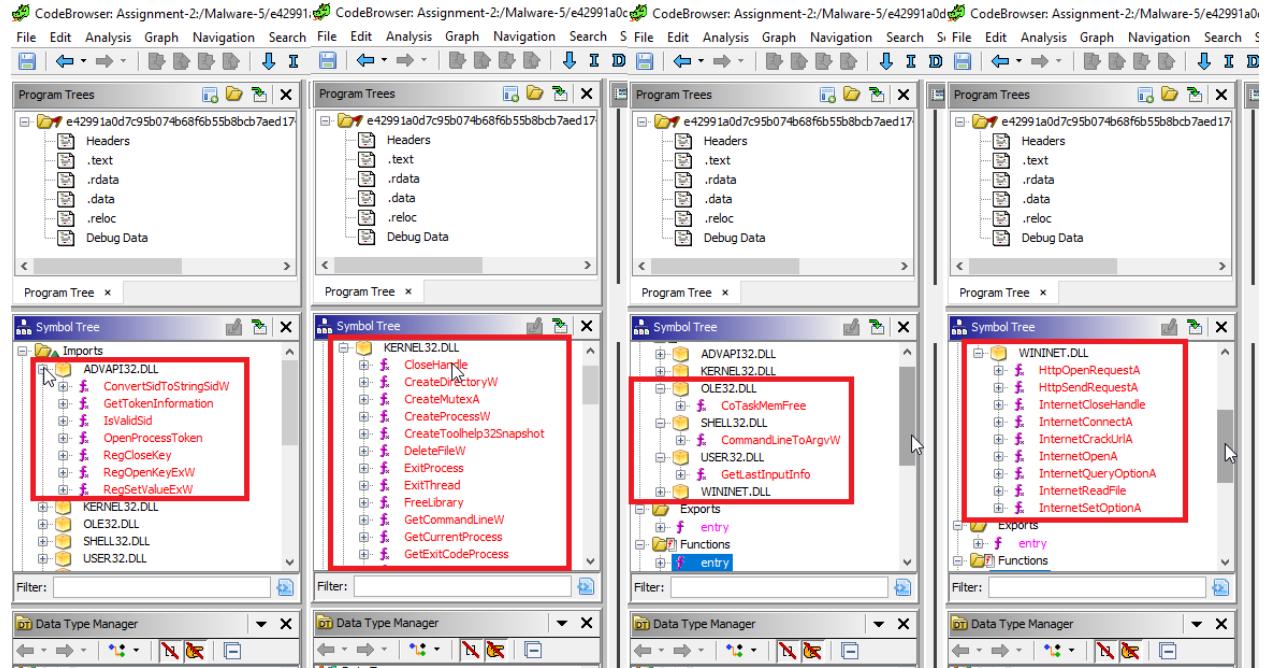


Figure 23: List of DLLs used by the malware with their functions

The suspicious functionality called by the DLLs includes the "Kernel32.dll" calling its function "GetWindowsDirectoryW" when the malicious function of malware "FUN_004030d0" is interact with the system programs including "notepad.exe", "explorer.exe", "svchost.exe" etc.

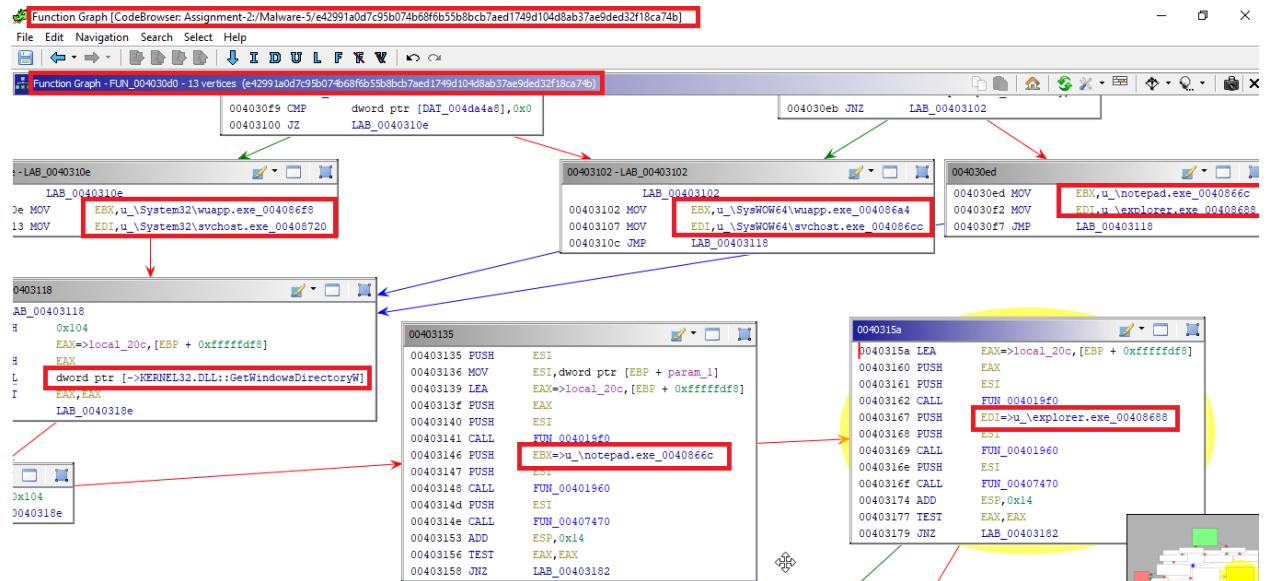


Figure 24: Malicious DLL function used by malware

Another suspicious functionality called by the “Kernel32.dll” DLLs includes “GetModuleFileName”, “GetFileSizeEx”, “DeleteFileW”, “VirtualAlloc” , and “VirtualFree” when the malicious function of the malware “FUN_00406710” deletes the zone file after completion of the mining process.

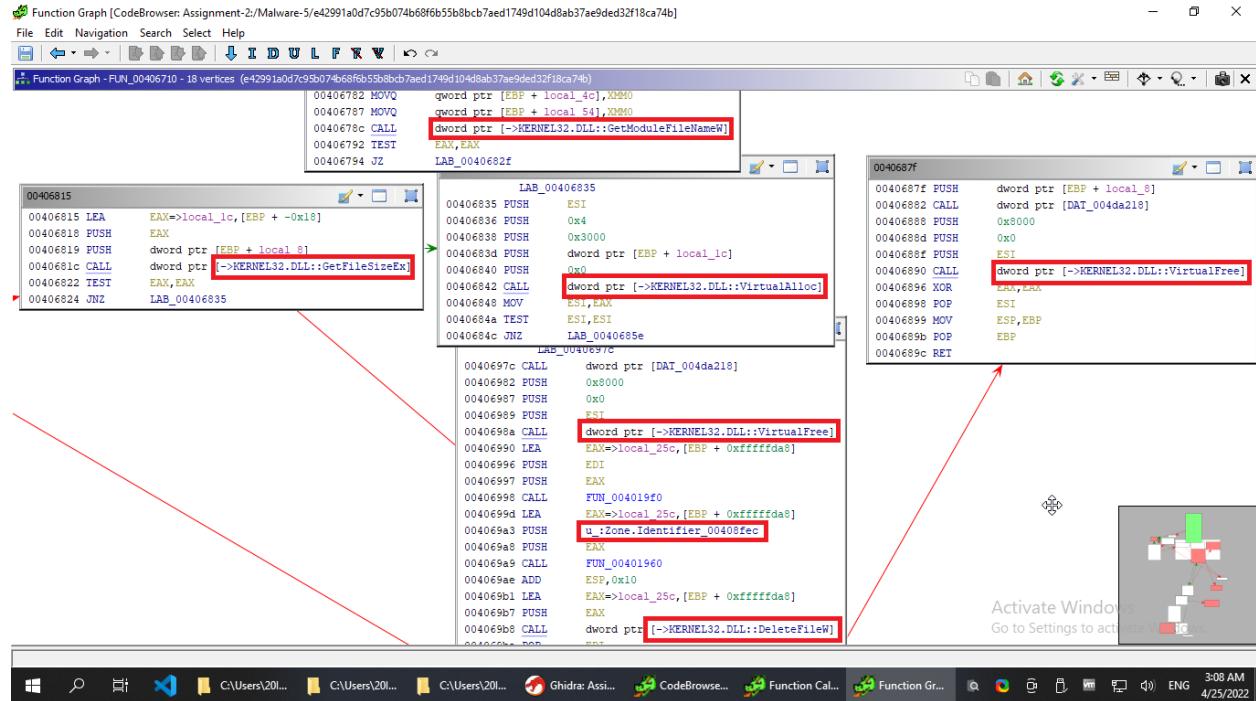


Figure 25: Malicious DLL function used by malware

Malware-2 Password-Stealer (003bbfec1)

This is a type of malware that steals the account information of the victim machine. To analyze malware I open it in Ghidra and then perform auto analysis on the malware sample.

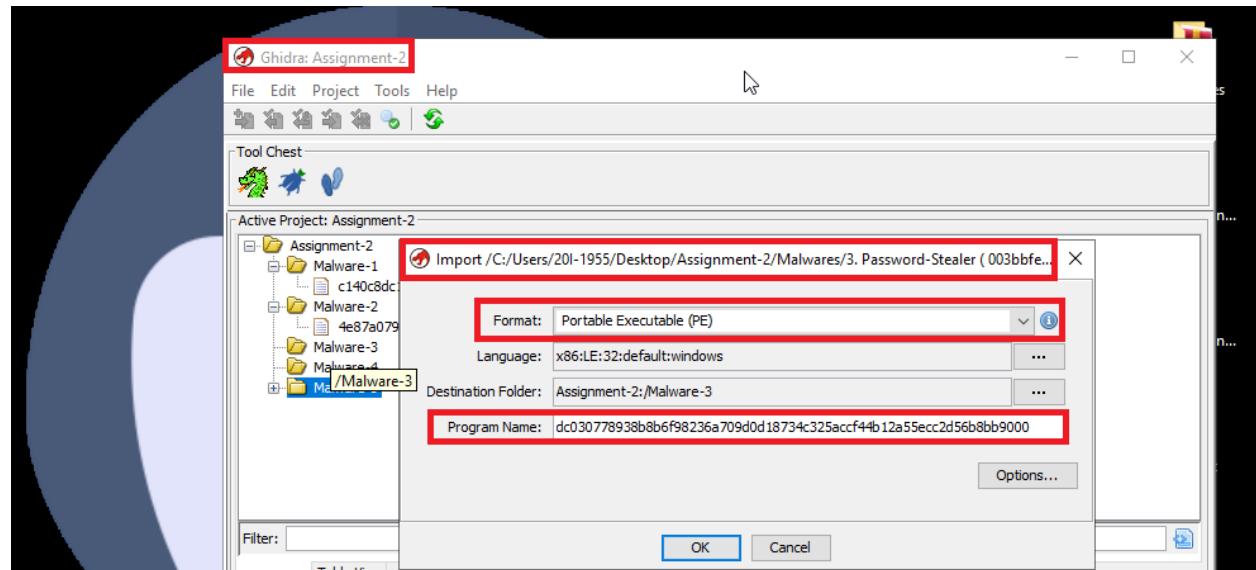


Figure 26: Malware-2: Opening malware in Ghidra

On opening the malware sample in the Ghidra I found that the malware is PE executable.



Figure 27: Malware-2: Basic details extracted by Ghidra about the malware sample

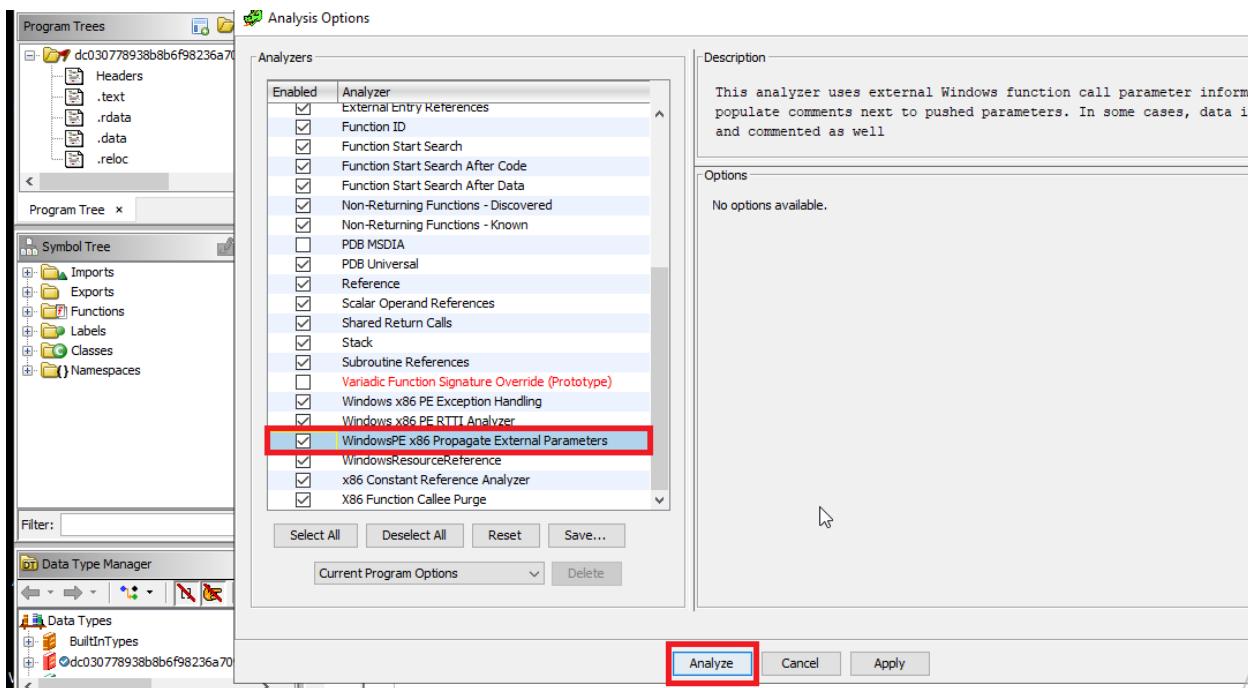


Figure 28: Malware-2: Ghidra auto analyze the malware

Task-1 What are the different segments or sections in case of each malware?

After Ghidra complete the auto analysis I found that the magic byte of the malware which is MZ, moreover I found the details of different sections used by malware that includes “.text”, “.rdata”, “.data”, “.reloc”.

CodeBrowser: Assignment-2/Malware-3/dc030778938b8b6f98236a709d0d18734c325acff44b12a55ecc2d56b8bb9000

File Edit Analysis Graph Navigation Search Select Tools Window Help

Program Trees Listing: dc030778938b8b6f98236a709d0d18734c325acff44b12a55ecc2d56b8bb9000

Program Tree x

Symbol Tree x

Imports Exports Functions f entry FUN_1000 Labels

```

        // Headers
        // ram:10000000-ram:100003ff
        //

assume DF = 0x0 (Default)
IMAGE_DOS_HEADER_10000000
10000000 4d 5a 90    IMAGE_DOS...
00 03 00
00 00 04

10000000 4d 5a      char[2]  "MZ"
10000000 [0]          "M", 'Z'

10000002 00 00      dw      0h
10000004 03 00      dw      3h
10000006 00 00      dw      0h
10000008 04 00      dw      4h
1000000a 00 00      dw      0h

```

e_magic XREF[1]: 100000b4 (*)

e_cwp XREF[1]: 1 e_cp Pages in file

e_crcl XREF[1]: 1 e_cparhdr Relocations

e_cparhdr XREF[1]: 1 e_minalloc Size of header in ...

e_minalloc XREF[1]: 1

Figure 29: Malware-2: Magic byte and details of the sections used by malware

Task-2 What are different functions, imports and exports of each Malware?

To perform its functionality malware imports different DLL's that includes “ADVAPI32.dll”, “KERNEL32.dll”, “OLE32.dll”, “SHLWAPI.dll”, “URLMON.dll”, “USER32.dll”, “USERENV.dll”, “WININET.dll” and “WSOCK32.dll”. Moreover it also uses different functions to perform its functionality.

Program Trees Listing: dc030778938b8b6f98236a709d0d18734c325acff44b12a55ecc2d56b8bb9000

Program Tree x

Symbol Tree x

Imports Exports Functions f entry FUN_1000afe3 Labels

```

MOV     dword ptr [hKey_1000f15f],0x00000001
f1 00 10
01 00 00 80

CALL   FUN_1000aed4
e8 c9 fb
ff ff

CALL   FUN_1000afe3
e8 d3 fc
ff ff

LEAVE
c9

RET
c3

```

XREF[1]: LAB_1000b2fc

XREF[1]: LAB_1000b306

XREF[1]: LAB_1000b30b

XREF[1]: LAB_1000b310

XREF[1]: LAB_1000b311

XREF[2]: undefined8 _fastcall entry(undefined4 param_1, undefined4 param_2, undefined4 param_3)

XREF[2]: undefined8 EDX:4, EAX:4 <RETURN>

XREF[2]: undefined4 ECX:4 param_1

XREF[2]: undefined4 EDX:4 param_2

XREF[2]: undefined4 Stack[0x4]:4 param_3

XREF[2]: entry

PUSH EBP
1000b312 55

MOV EBP,ESP
1000b313 8b ec

Figure 30: Malware-2: Details of the functions, imports and exports used by malware

The screenshot shows the IDA Pro interface. On the left, the Program Tree and Symbol Tree panes are visible. The Symbol Tree pane is highlighted with a red box around the entry point and its sub-functions: FUN_1000, FUN_10001, FUN_100010, FUN_1000100, FUN_10001026, FUN_10001051, FUN_10001091, FUN_100010d6, FUN_100011, FUN_1000114c, and FUN_100011aa. The main pane displays assembly code for various functions, including LAB_1000b2fc, LAB_1000b306, and LAB_1000b30b. The assembly code includes instructions like MOV, CALL, LEAVE, RET, and PUSH.

Figure 31: Malware-2: Different functions used by malware

Task-3 What is flow of functions in case of each malware? Is there any suspicious function? Give detail (name, arguments, call mechanism) of suspicious functions?

The flow of the functions of this malware is that it first collects the information of the victim machine that includes name of host machine, username and password, collects the information of the victim's Facebook account and collects the information of the mail server including email accounts and then uploads that details to some malicious link.

Task-3.1 FUN_10005

In this function malware collects the details of the victim host machine including the username, password, details of the remote directory and its port number.

The screenshot shows the IDA Pro interface with the assembly view. The assembly code for the FUN_10005 function is displayed, showing various PUSH, CALL, MOV, and POP instructions. Red boxes highlight specific memory locations and function names: s_Password_1000ff37, s_HostName_1000ff40, s_UserName_1000ff49, and s_RemoteDirectory_1000ff52. The right pane shows the corresponding C-like pseudocode for these sections, involving parameters like param_1, param_2, and local variables. The pseudocode includes function calls like FUN_10001d2a and parameter assignments like "param_2,local_814,s_RemoteDirectory_1000ff52".

Figure 32: Malware-2: Malware capturing details of the host machine

Function Graph [CodeBrowser: Assignment-2-/Malware-3/dc030778938b8bf98236a709d0d18734c325acfc44b12a55ecc2d56b8bb9000]

```

10005b59 CALL    FUN_10001871
10005b5e PUSH   0x0
10005b60 PUSH   _s_Password_1000ff37
10005b65 PUSH   dword ptr [EBP + local_814]
10005b66 PUSH   dword ptr [EBP + param_2]
10005b67 CALL    FUN_10001d2a
10005b73 MOV    dword ptr [EBP + local_820],EAX
10005b79 PUSH   0x0
10005b7b PUSH   _sHostName_1000ff40
10005b80 PUSH   dword ptr [EBP + local_814]
10005b81 PUSH   dword ptr [EBP + param_2]
10005b89 CALL    FUN_10001d2a
10005b8b MOV    dword ptr [EBP + local_818],EAX
10005b94 PUSH   0x0
10005b96 PUSH   _s_UserName_1000ff49
10005b9b PUSH   dword ptr [EBP + local_814]
10005ba1 PUSH   dword ptr [EBP + param_2]
10005ba4 CALL    FUN_10001d2a
10005ba9 MOV    dword ptr [EBP + local_81c],EAX
10005baf PUSH   0x0
10005bb1 PUSH   _s_RemoteDirectory_1000ff52
10005bb6 PUSH   dword ptr [EBP + local_814]
10005bb8 PUSH   dword ptr [EBP + param_2]
10005bbf CALL    FUN_10001d2a
10005bc4 MOV    dword ptr [EBP + local_828],EAX
10005bca LEA    EAX=>local_824,[EBP + 0xfffffff7e0]
10005bd0 PUSH   FAX
10005bd1 PUSH   _s_PortNumber_1000ff62
10005bd6 PUSH   dword ptr [EBP + local_814]
10005bd8 PUSH   dword ptr [hKey_1000f15f]
10005be2 CALL    FUN_10001d2a
10005be7 AND    EAX,EAX
10005be9 JZ     LAB_10005c04

```

Figure 33: Malware-2: Malware capturing details of the host machine

Function Graph [CodeBrowser: Assignment-2-/Malware-3/dc030778938b8bf98236a709d0d18734c325acfc44b12a55ecc2d56b8bb9000]

```

10005ae1 - FUN_10005ae1
undefined __stdcall FUN_10005ae1(int * param_1, HKEY param_2, LPCSTR param_3)
undefined AL:1 <RETURN>
int * Stack[0x4]:4 param_1
HKEY Stack[0x8]:4 param_2
LPCSTR Stack[0xc]:4 param_3
undefined4 Stack[-0x8]:4 local_8
undefined4 Stack[-0xc]:4 local_c
undefined4 Stack[-0x10]:4 local_10
undefined1 Stack[-0x10..-0x10] local_810
undefined4 Stack[-0x14..-0x14] local_814
undefined4 Stack[-0x18..-0x18] local_818
undefined4 Stack[-0x1c..-0x1c] local_81c
undefined4 Stack[-0x20..-0x20] local_820
undefined4 Stack[-0x24..-0x24] local_824
undefined4 Stack[-0x28..-0x28] local_828
undefined4 Stack[-0x2c..-0x2c] local_82c
undefined4 Stack[-0x30..-0x30] local_830
undefined4 Stack[-0x34..-0x34] local_834
        FUN_10005ae1
10005ae1 PUSH   EBP
10005ae2 MOV    EBP,ESP
10005ae4 ADD    ESP,0xfffffff7d0
10005aea LEA    EAX=>local_8,[EBP + -0x4]
10005aed PUSH   EAX
10005afe PUSH   dword ptr [EBP + param_3]
10005af1 PUSH   dword ptr [EBP + param_2]
10005af4 CALL    ADVAPI32.DLL::RegOpenKeyA
10005af9 AND    EAX,EAX
10005afb JNZ   LAB_10005d43

```

LAB_10005b08 - LAB_10005b08

```

10005b08 MOV    dword ptr [EBP + local_10],0x7ff
10005b0f PUSH   0x0
10005b11 PUSH   0x0
10005b13 PUSH   0x0
10005b15 PUSH   0x0
10005b17 LEA    EAX=>local_10,[EBP + -0xc]
10005b1a PUSH   EAX
10005b1b LEA    EAX=>local_810,[EBP + 0xfffffff7f4]
10005b21 PUSH   EAX
10005b22 PUSH   dword ptr [EBP + local_c]
10005b25 PUSH   dword ptr [EBP + local_8]
10005b28 CALL   ADVAPI32.DLL::RegEnumKeyExA
10005b2d AND    EAX,EAX
10005b2f JZ     LAB_10005b36

```

LAB_10005d3b - LAB_10005d3b

```

10005d3b PUSH   dword ptr [EBP + local_8]
10005d3e CALL   ADVAPI32.DLL::RegCloseKey

```

Figure 34: Malicious DLL fuctions used by malware

Task-3.2 05 FUN_100007

In this function malware collects the details of the victim facebook profile.

```

        00    CMP    dword ptr [EBP + local_2c], 0x0
        10007e1d 83 7d d8 00
        10007e21 74 08    JZ    LAB_10007e2b
        10007e23 ff 75 d8    PUSH   dword ptr [EBP + local_2c]
        10007e26 e8 a7 36    CALL   OLE32.DLL:CoTaskMemFree
        00 00

        LAB_10007e2b
        10007e2b eb 8c    JMP    LAB_10007db9

        LAB_10007e2d
        10007e2d 8b 55 f8    MOV    EDX,dword ptr [EBP + local_c]
        10007e30 8b 12    MOV    EDX,dword ptr [EDX]
        10007e32 ff 75 f8    PUSH   dword ptr [EBP + local_c]
        10007e35 ff 52 08    CALL   dword ptr [EDX + 0x8]

        LAB_10007e38
        10007e38 8b 55 fc    MOV    EDX,dword ptr [EBP + local_8]
        10007e3b 8b 12    MOV    EDX,dword ptr [EDX]
        10007e3d ff 75 fc    PUSH   dword ptr [EBP + local_8]
        10007e40 ff 52 08    CALL   dword ptr [EDX + 0x8]

        LAB_10007e43
        10007e43 6a 00    PUSH   0x0
        10007e45 68 7d 03    PUSH   u_http://www.facebook.com/_1001037d
        01 10
        10007e4a ff 75 08    PUSH   dword ptr [EBP + param_1]
        10007e4d e8 90 fd    CALL   FUN_10007be2
        ff ff

        10007e52 c9    LEAVE
        10007e53 c2 04 00    RET    0x4

        LAB_10007e43
        10007e43 PUSH 0x0
        10007e45 PUSH u_http://www.facebook.com/_1001037d
        10007e4a PUSH dword ptr [EBP + param_1]
        10007e4d CALL FUN_10007be2
        10007e52 LEAVE
        10007e53 RET 0x4

        LAB_10007e38
        10007e38 MOV EDX,dword ptr [EBP + local_8]
        10007e3b MOV EDX,dword ptr [EDX]
        10007e3d PUSH dword ptr [EBP + local_8]
    
```

Figure 35: Malware-2: Gathering the details of Victims facebook account

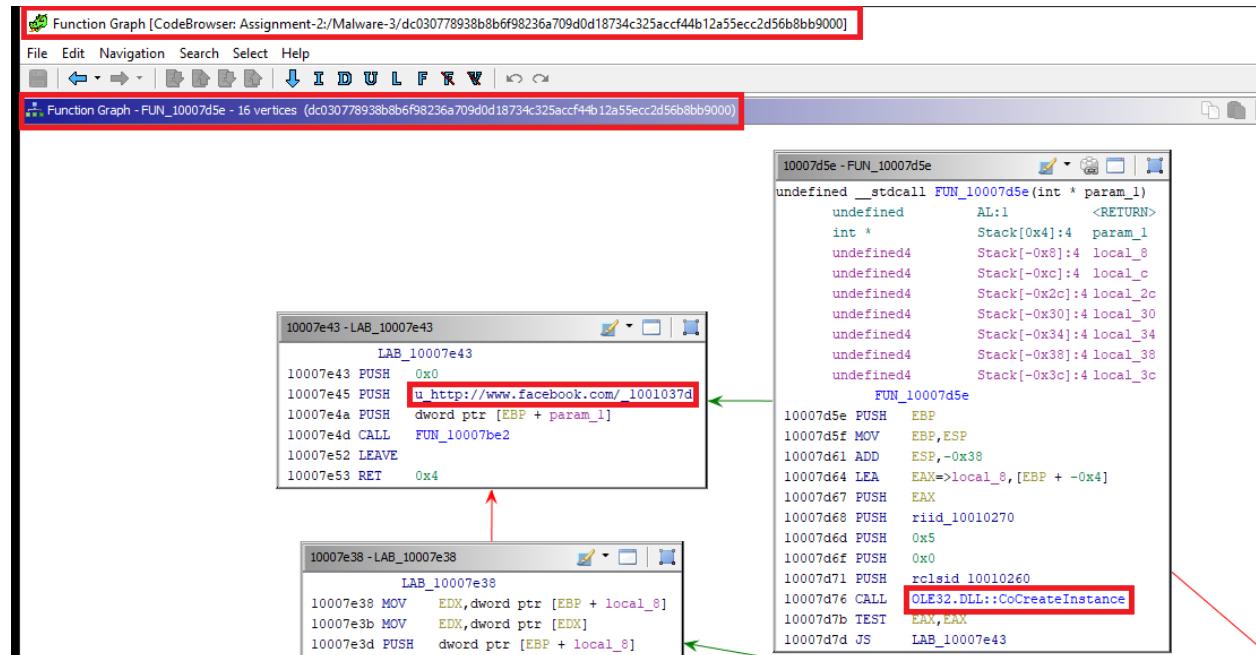


Figure 36: Malware-2: Gathering the details of Victims facebook account

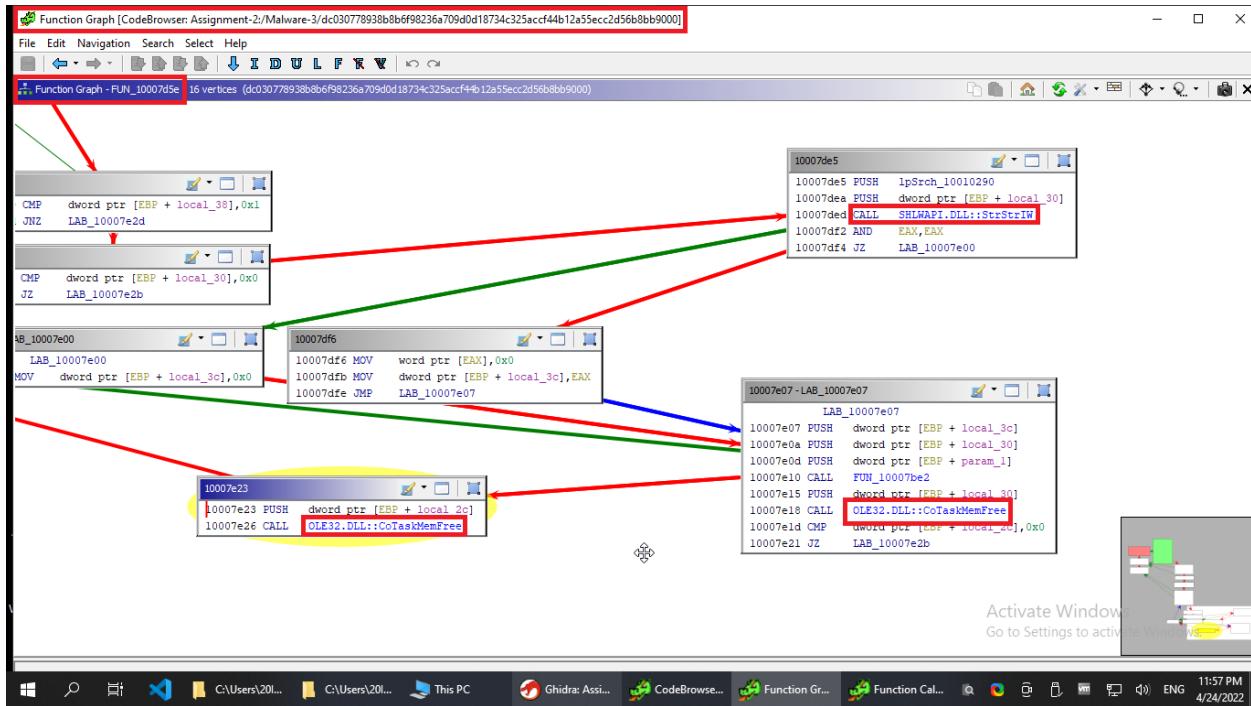


Figure 37: Malicious DLL functions used by malware

Task-3.3 FUN_10009

In this function malware collects the information of the victim email account including its email address and password.

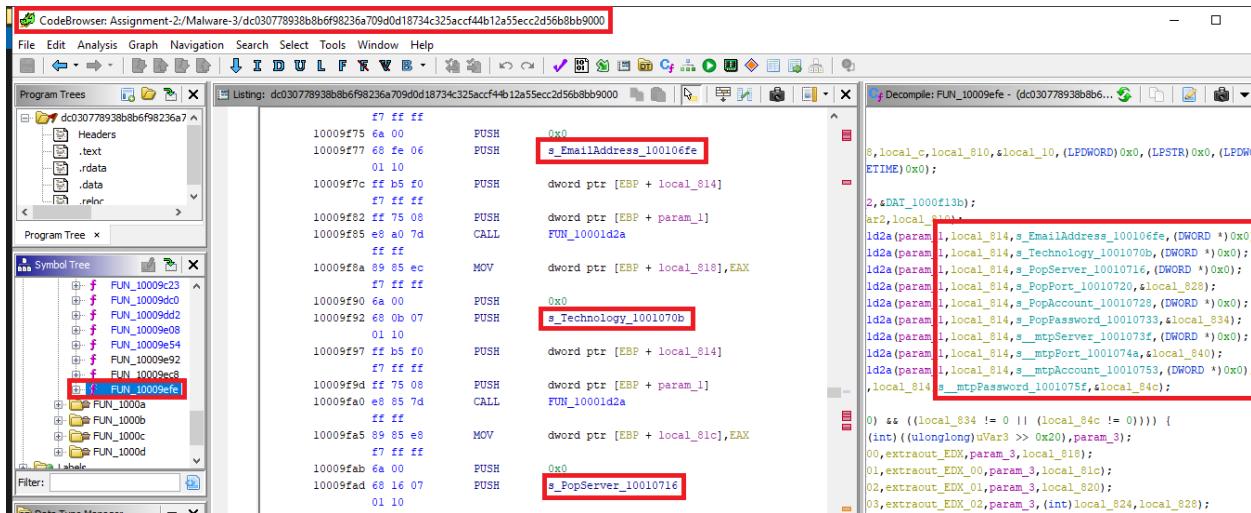


Figure 38: Malware collects the information of the email account of the victim

The screenshot shows the Ghidra Function Graph interface. The main window displays the assembly code for the function `10009ef0 - LAB_10009ef0`. The code involves pushing various parameters onto the stack and calling functions like `FUN_10001584` and `FUN_10001588`. Several memory locations are highlighted with red boxes, including `s_EmailAddress_100106fe`, `s_Technology_1001070b`, `s_PopServer_1001071d`, `s_PopPort_10010720`, and `s_MtpPort_1001074a`. A red arrow points from the left side of the graph towards the right, indicating the flow of control through the function. The right pane shows the assembly code for `10009f25 - LAB_10009f25`, which includes calls to `ADVAPI32.DLL::RegCloseKey` and `ADVAPI32.DLL::RegOpenKeyA`.

Figure 39: Malware collects the information of the email account of the victim

This screenshot shows the Ghidra Function Graph interface. The main window displays the assembly code for the function `10009ef0 - LAB_10009ef0`. The code includes calls to `ADVAPI32.DLL::RegCloseKey` and `ADVAPI32.DLL::RegOpenKeyA`. A red box highlights the call to `ADVAPI32.DLL::RegOpenKeyA`. A blue arrow points from the left side of the graph towards the right, indicating the flow of control through the function. The right pane shows the assembly code for `10009f25 - LAB_10009f25`, which includes calls to `ADVAPI32.DLL::RegEnumKeyExA` and `ADVAPI32.DLL::RegCloseKey`.

Figure 40: Malicious DLL functions used by malware

Task-3.4 FUN_10008

In this function malware connects to the some http and ftp server maybe to upload the collected informiton to attacker the server.

```

CodeBrowser: Assignment-2/Malware-3/dc030778938b8b6f98236a709dd18734c325accf44b12a55ecc2d56b8bb9000
File Edit Analysis Graph Navigation Search Select Tools Window Help
I D U L F R V B ...
Program Trees X
Listing: dc030778938b8b6f98236a709dd18734c325accf44b12a55ecc2d56b8bb9000
10008f6e ff 75 fc PUSH dword ptr [EBP + local_8]
10008f71 e8 12 89 CALL FUN_10001888
10008f76 89 45 c8 MOV dword ptr [EBP + local_3c],EAX
10008f79 ff 75 fc PUSH dword ptr [EBP + local_8]
10008f7c ff 75 c8 PUSH dword ptr [EBP + local_3c]
10008f7f ff 75 f4 PUSH dword ptr [EBP + local_10]
10008f82 e8 38 89 CALL FUN_100018bf
10008f87 68 cb 04 PUSH s_ftp://_100104cb
10008f8c e8 09 24 CALL KERNEL32.DLL::lstrlenA
10008f91 50 PUSH EAX
10008f92 68 cb 04 PUSH s_ftp://_100104cb
10008f97 ff 75 c8 PUSH dword ptr [EBP + local_3c]
10008f9a ed ab 25 CALL SHLWAPI.DLL::StrCmpNIA
10008f9f 23 c0 AND EAX,EAX
10008fa1 74 18 JZ LAB_10008fbh
10008fa3 d2 d2 04 PUSH s_http://_100104d2
10008fa8 e8 ed 23 CALL KERNEL32.DLL::lstrlenA
10008fad 50 PUSH EAX
10008fae d2 d2 04 PUSH s_http://_100104d2
10008fb0 01 10
Decompile: FUN_10008e59 - (dc030778938b8b6f98236a709dd18734c325accf44b12a55ecc2d56b8bb9000)
{
    if (local_38 != 0) {
        for (; iVar2 != 0; iVar2 = uVar2 - 1) {
            *puVar6 = *puVars;
            puVars = puVar5 + 1;
            puVar5 = puVar6 + 1;
        }
    }

    cal_20 = local_38;
    calFree(local_34);
    if ((local_8 != 0) && (local_14 != 0)) && (local_20 != 0) {
        lpStr1 = (LPCSTR)FUN_10001888(local_8);
        FUN_100018bf(local_10,lpStr1,local_8);
        iVar1 = lstrlenA(s_ftp://_100104cb);
        iVar1 = StrCmpNIA(lpStr1,s_ftp://_100104cb,iVar1);
        uVar3 = extraout_EBX;
        uVar4 = extraout_EDX;
        if (iVar1 != 0) {
            iVar1 = lstrlenA(s_http://_100104d2);
            iVar1 = StrCmpNIA(lpStr1,s_http://_100104d2,iVar1);
            uVar3 = extraout_ECX_00;
            uVar4 = extraout_EDX_00;
        }
        if (iVar1 != 0) {
            iVar1 = lstrlenA(s_https://_100104da);
            iVar1 = StrCmpNIA(lpStr1,s_https://_100104da,iVar1);
            uVar3 = extraout_ECX_00;
            uVar4 = extraout_EDX_00;
        }
    }
}

```

Figure 41: Malware-2: Connects to some malicious server

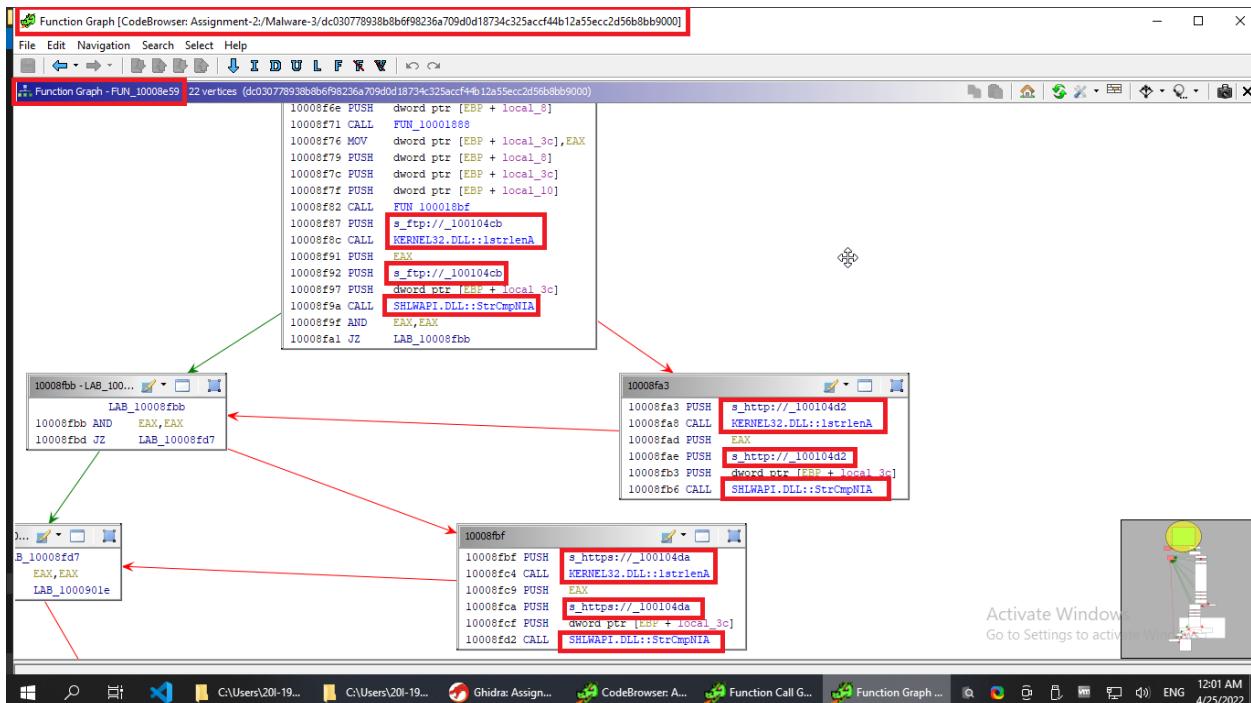


Figure 42: Malware-2: Connects to some malicious server also the DLL it used durint the process

Task-4 What DLL's any malware includes? Are there any suspicious functionality called by these DLL's?

The DLL's used by this malware includes "ADVAPI32.dll", "KERNEL32.dll", "OLE32.dll", "SHLWAPI.dll", "URLMON.dll", "USER32.dll", "USERENV.dll", "WININET.dll" and "WSOCK32.dll".

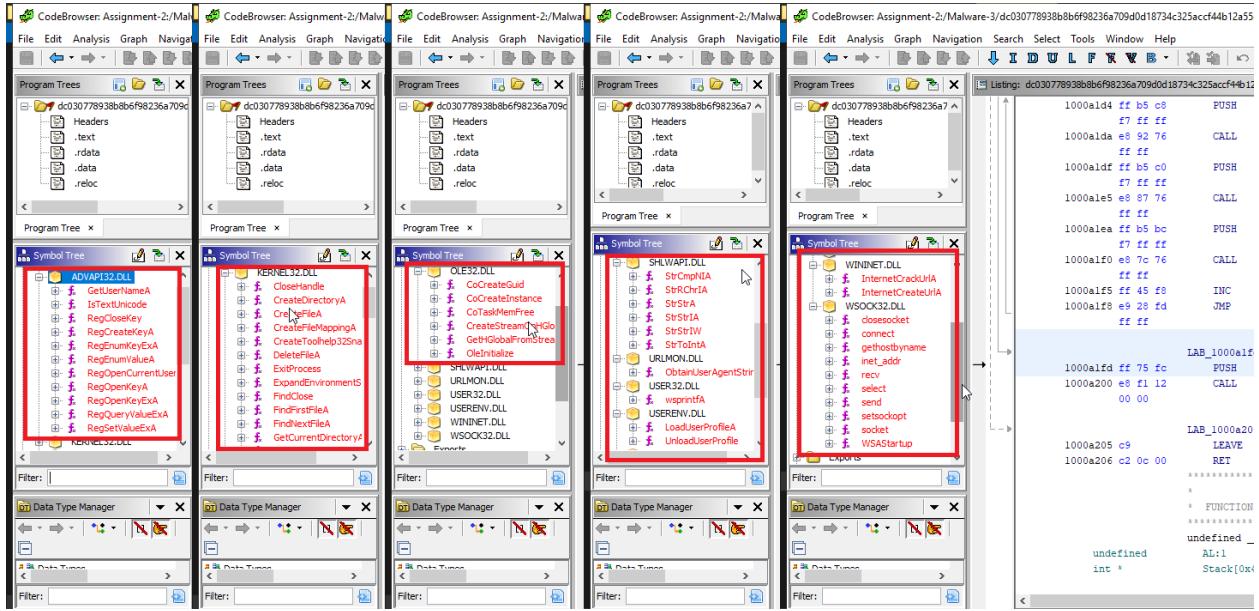


Figure 43: List of DLLs used by the malware with their functions

The suspicious functionality called by the DLLs includes the "ADVAPI32.dll" calling its function "RegOpenKeyA", "RegEnumKeyExA", and "RegCloseKey" when the malicious function of malware "FUN_10005ae1" collects the information of the victims host machine.

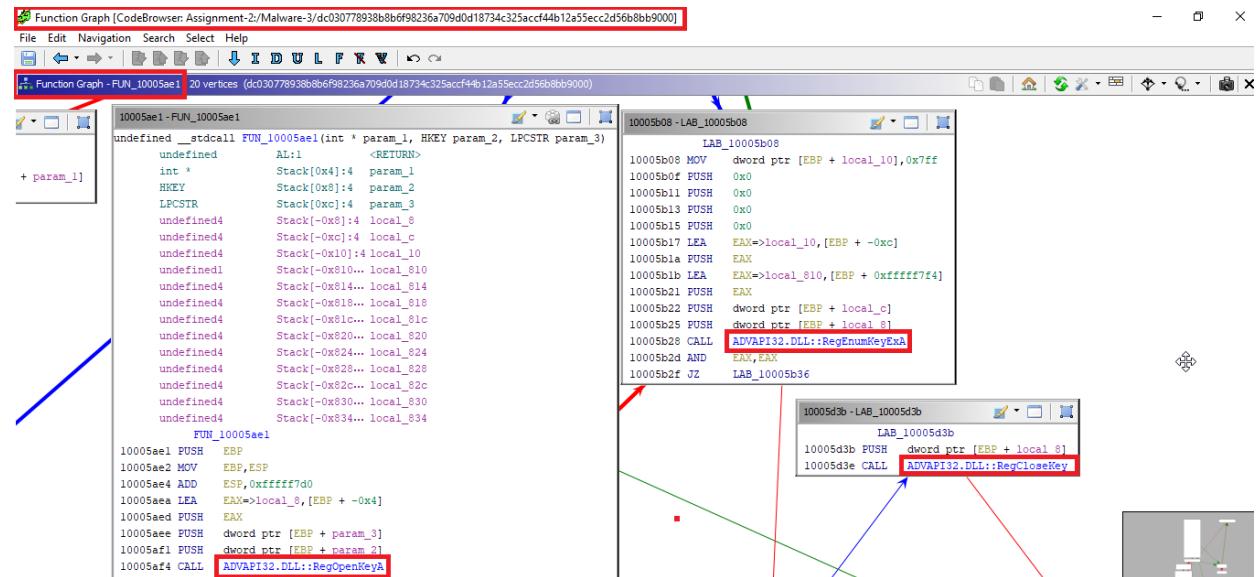


Figure 44: Malware-2 Malicious DLL functions used by malware

The suspicious functionality called by the DLLs includes the “OLE32.DLL” calling its function “CoCreateInstance”, “CoTaskMemFree”, and “SHLWAPI.dll” calling its function “StrStrIW” when the malicious function of malware “FUN_10007d5e” collects the information of the victims facebook profile.

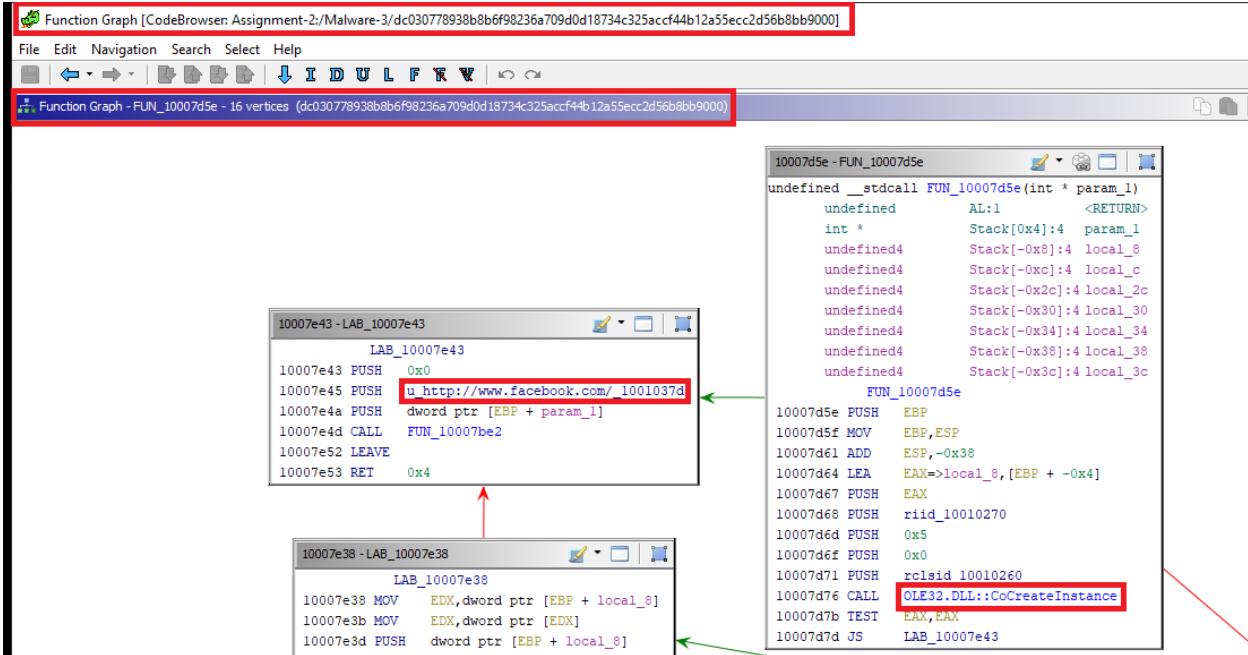


Figure 45: Malware-2 Malicious DLL functions used by malware

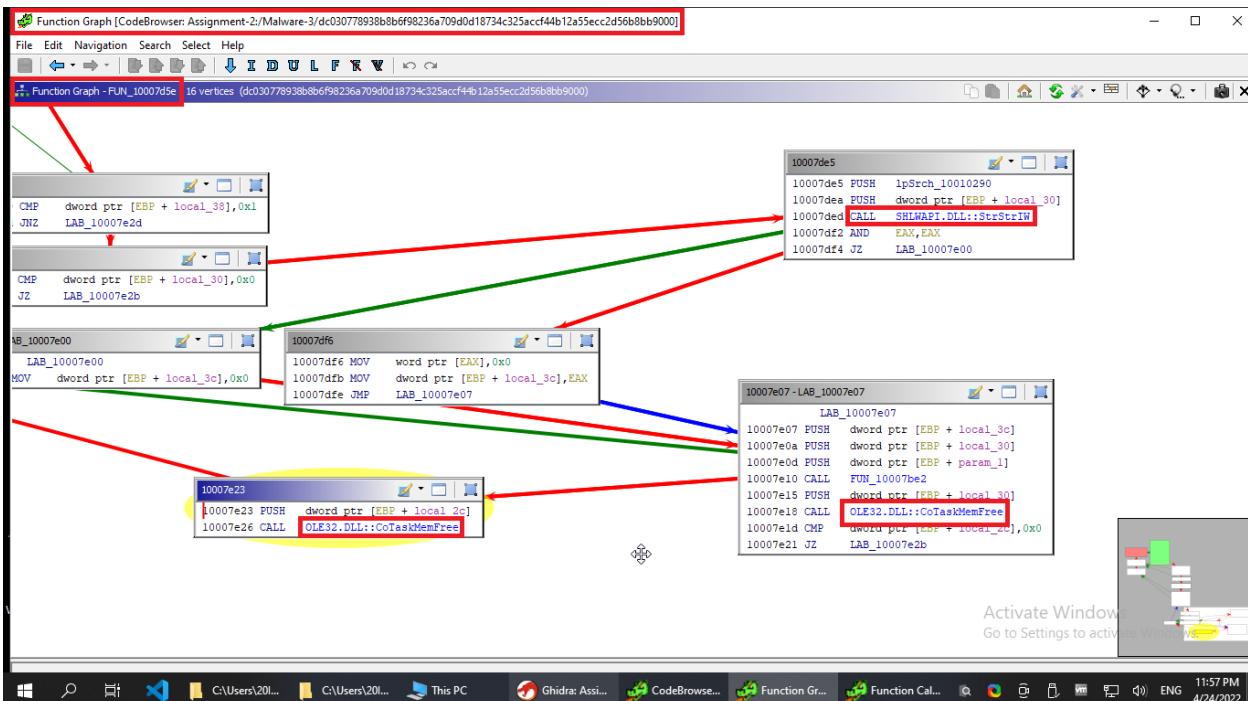


Figure 46: Malware-2 Malicious DLL functions used by malware

Another suspicious functionality called by the DLLs includes the “ADVAPI32.dll” calling its function “RegOpenKeyA”, “RegEnumKeyExA”, and “RegCloseKey” when the malicious function of malware “FUN_10009efe” collects the information of the victim email account including its email address and password.

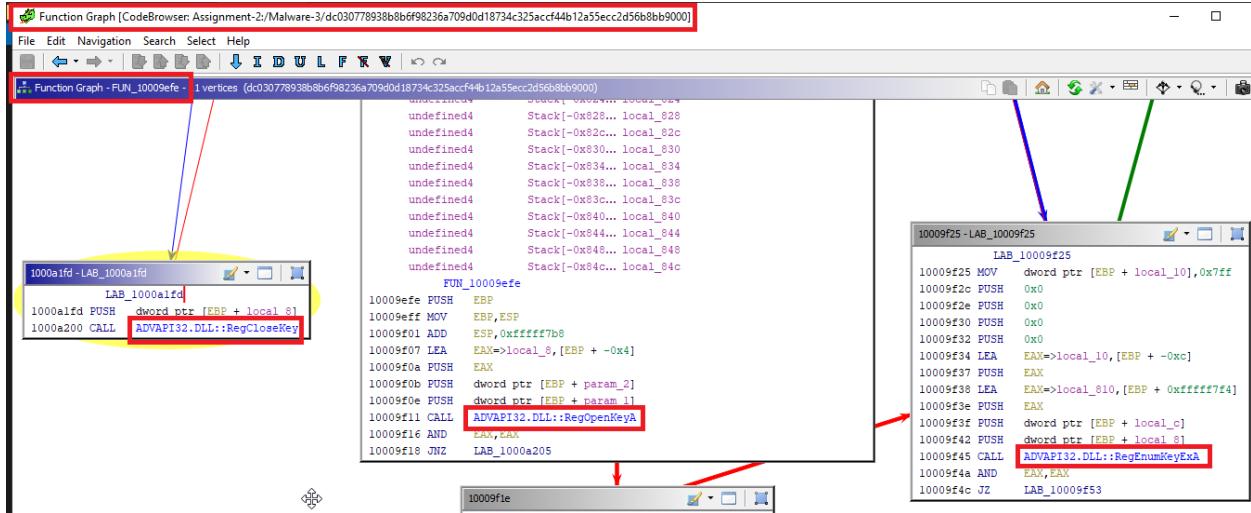


Figure 47: Malware-2 Malicious DLL functions used by malware

Another suspicious functionality called by the DLLs includes the “KERNEL32.dll” calling its function “IstrlenA” and “SHLWAPI.dll” calling its function “StrCmpNIA” when the malicious function of malware “FUN_10008e59” connects to some malicious http and ftp server.

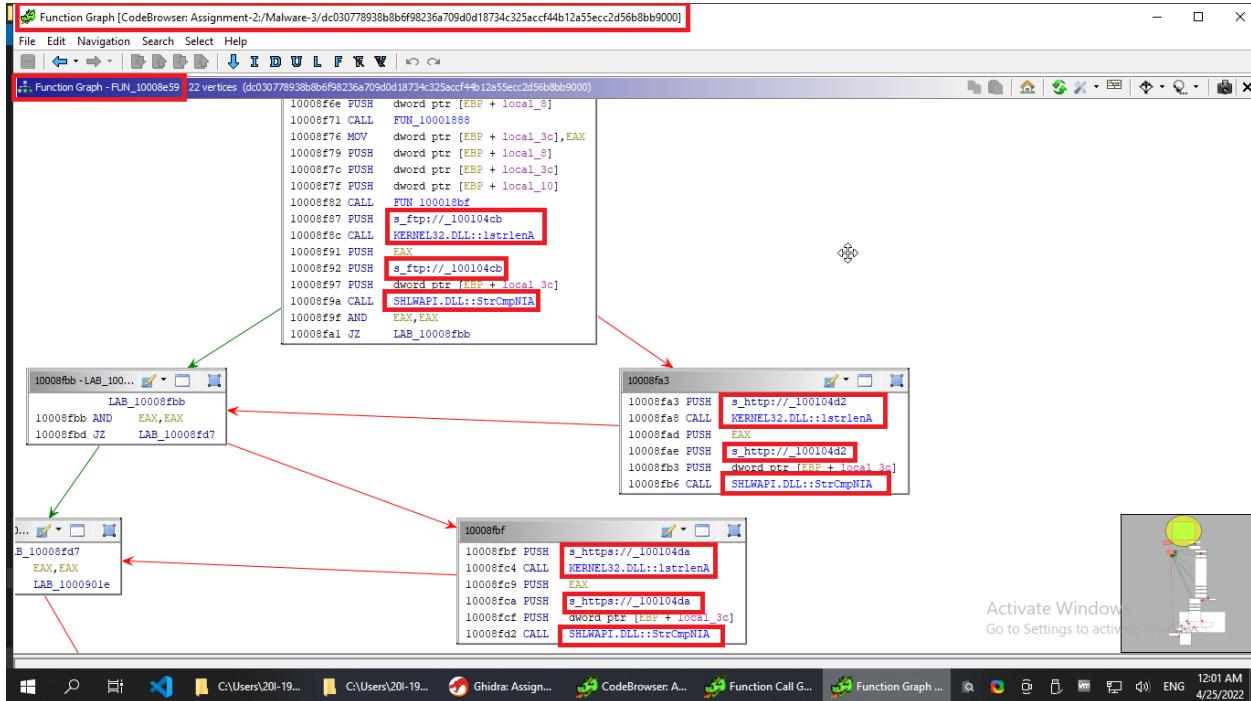


Figure 48: Malware-2 Malicious DLL functions used by malware

Malware-3 in32Tnega.bXRKZUB

This malware belongs to the family of trojans. To analyze malware I open it in Ghidra and then perform auto analysis on the malware sample.

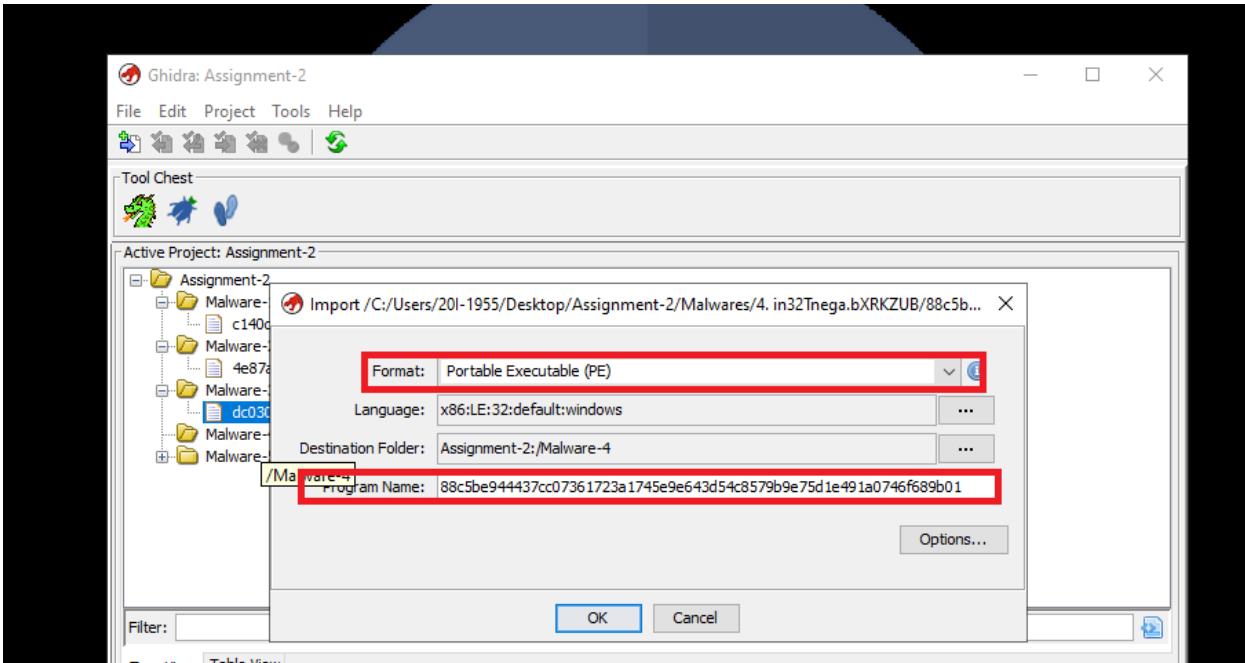


Figure 49: Malware-3: Opening malware in Ghidra

On opening the malware sample in the Ghidra I found that the malware is PE executable.

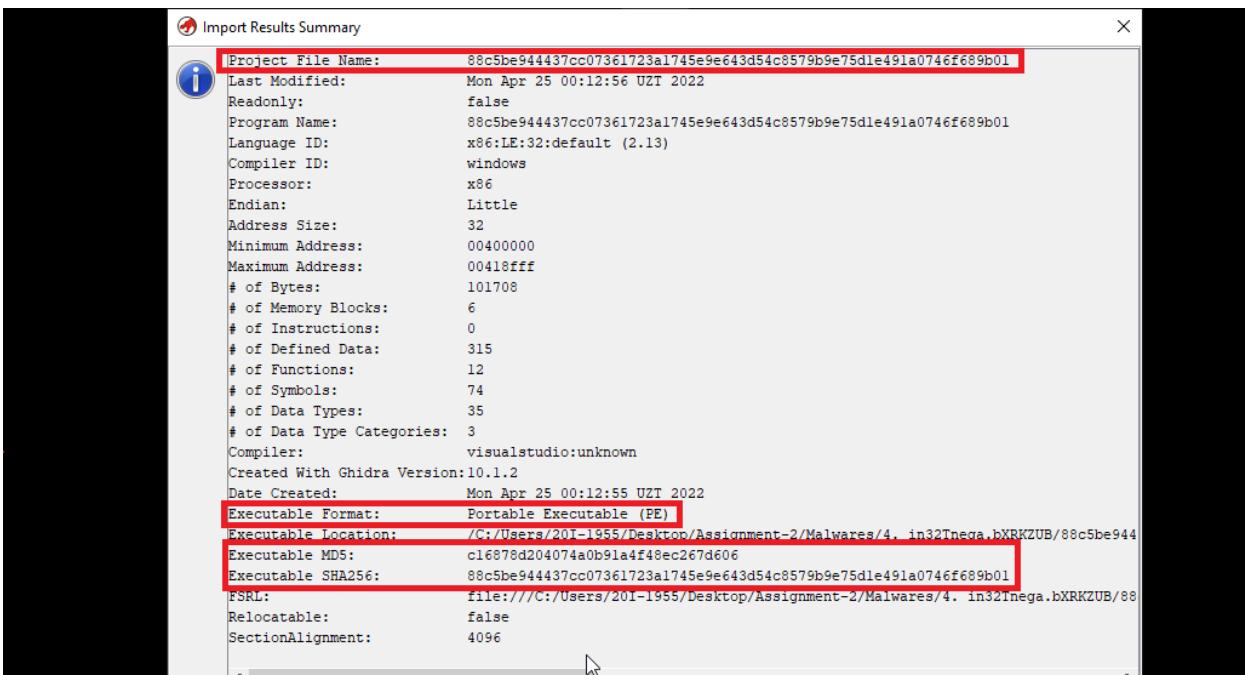


Figure 50: Malware-3: Basic details extracted by Ghidra about the malware sample

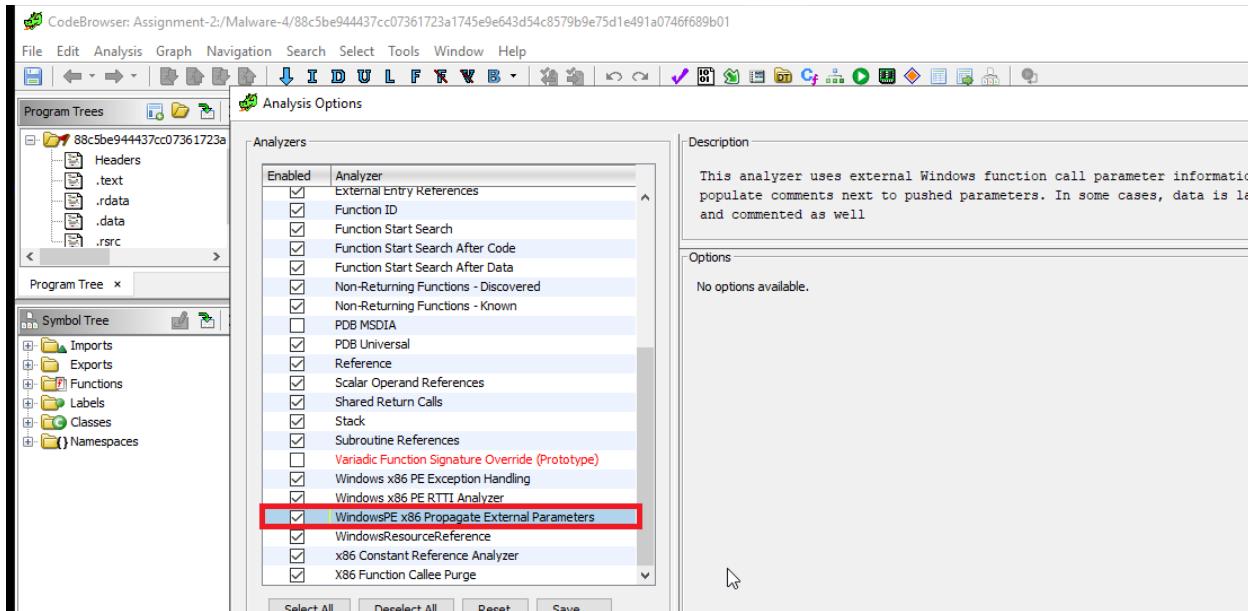


Figure 51: Malware-3: Ghidra auto analyze the malware

Task-1 What are the different segments or sections in case of each malware?

After Ghidra complete the auto analysis I found that the magic byte of the malware which is MZ, moreover I found the details of different sections used by malware that includes “.text”, “.rdata”, “.data”, “.rsrc”.

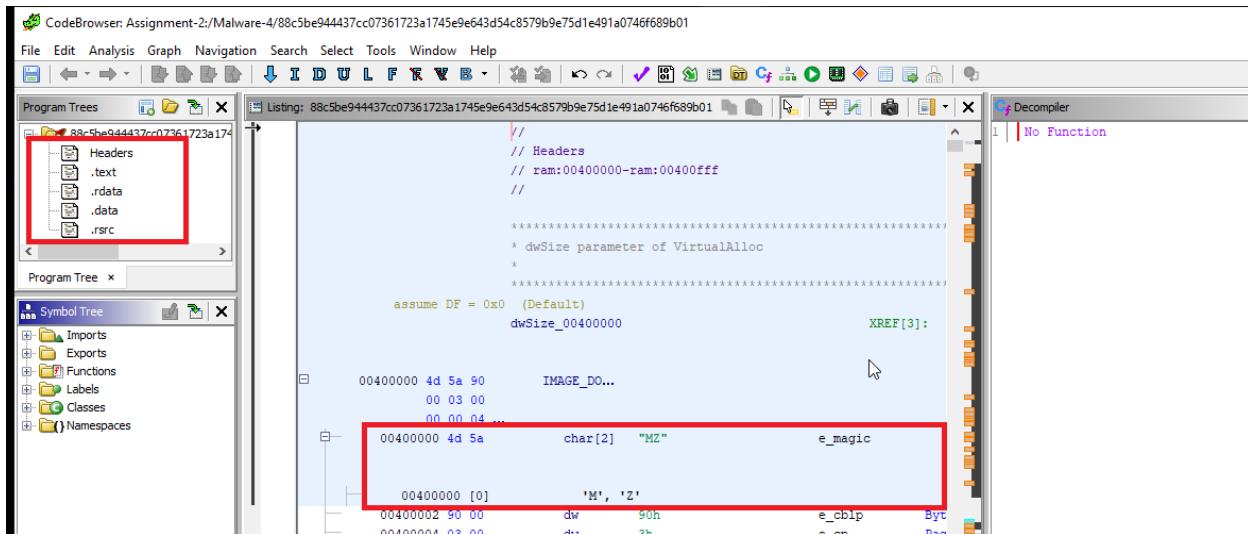


Figure 52: Malware-3: Magic byte and details of the sections used by malware

Task-2 What are different functions, imports and exports of each Malware?

To perform its functionality malware imports different DLL's that includes “KERNEL32.dll”, “SHELL32.dll” and “USER32.dll”. Moreover it also uses different functions to perform its functionality.

The screenshot shows the CodeBrowser interface with the following details:

- Program Trees:** Shows the file structure of the malware binary.
- Symbol Tree:** Displays the symbol table with imports from KERNEL32.DLL, SHELL32.DLL, and USER32.DLL, and exports including a function named "entry".
- Listing:** Shows the assembly listing for the malware. A specific instruction at address 00400000 is highlighted: `00400000 4d 5a 90 IMAGE_D...`. This is a call to the Windows API function VirtualAlloc.
- Decompiler:** A partially visible column on the right showing decompiled code.

Figure 53: Malware-3: Details of the functions, imports and exports used by malware

The screenshot shows the CodeBrowser interface with the following details:

- Program Trees:** Shows the file structure of the malware binary.
- Symbol Tree:** Displays the symbol table with many local functions named FUN_0040 through FUN_004c.
- Listing:** Shows the assembly listing for the malware. A specific instruction at address 00400000 is highlighted: `00400000 4d 5a 90 IMAGE_D...`. This is a call to the Windows API function VirtualAlloc.
- Decompiler:** A partially visible column on the right showing decompiled code.

Figure 54: Malware-3: Different functions used by malware

Task-3 What is flow of functions in case of each malware? Is there any suspicious function?

Give detail (name, arguments, call mechanism) of suspicious functions?

The flow of the funcitons of this malware is that it installs some malicious software on the victim machine without notifying him/her.

Task-3.1 FUN_00401550

In this function malware collects information of the running processes and then install some malicious software on the victim machine.

```

CodeBrowser: Assignment-2/Malware-4/88c5be944437cc07361723a1745e9e643d54c8579b9e75d1e491a0746f689b01

File Edit Analysis Graph Navigation Search Select Tools Window Help

Program Trees Listing: 88c5be944437cc07361723a1745e9e643d54c8579b9e75d1e491a0746f689b01
Decompile: FUN_00401550 (88c5be944437cc07361723a1745e9e643d54c8579b9e75d1e491a0746f689b01)

404((byte **)pVar2,local_160,&local_10);
acut_var,bVar1 == 0) break;
l_c,0x405,0,0);
local_6,0x65,(LPCSTR)local_168;
l_30,(byte *)0x0,local_26c,(byte *)local_168,(byte *)0x0);
e **pVar2,(LPCTSTR)local_370,local_10);
,(char *)local_168;
9340(local_168,s_pbin.dat_00411080);
nt *)0x0) {
PCSTR)local_370,(char *)local_26c);

);
_);
*local_168,(byte *)0x0,local_26c,byte *)s.setup.exe_004110
ssA((LPCSTR)local_168,(LPSTR)0x0,(LPSECURITY_ATTRIBUTES)0x0,
(LPSECURITY_ATTRIBUTES)0x0,0,0,(LVOID)0x0,(LPCSTR)local_
&local_20);

r *)local_168,(byte *)0x0,local_168,(LPCSTR)UR0,0x10);
s_HAI_Process_could_not_be_started_004

```

Figure 55: Malware-3: Malware collects the information of the running process and instlling some malicious software

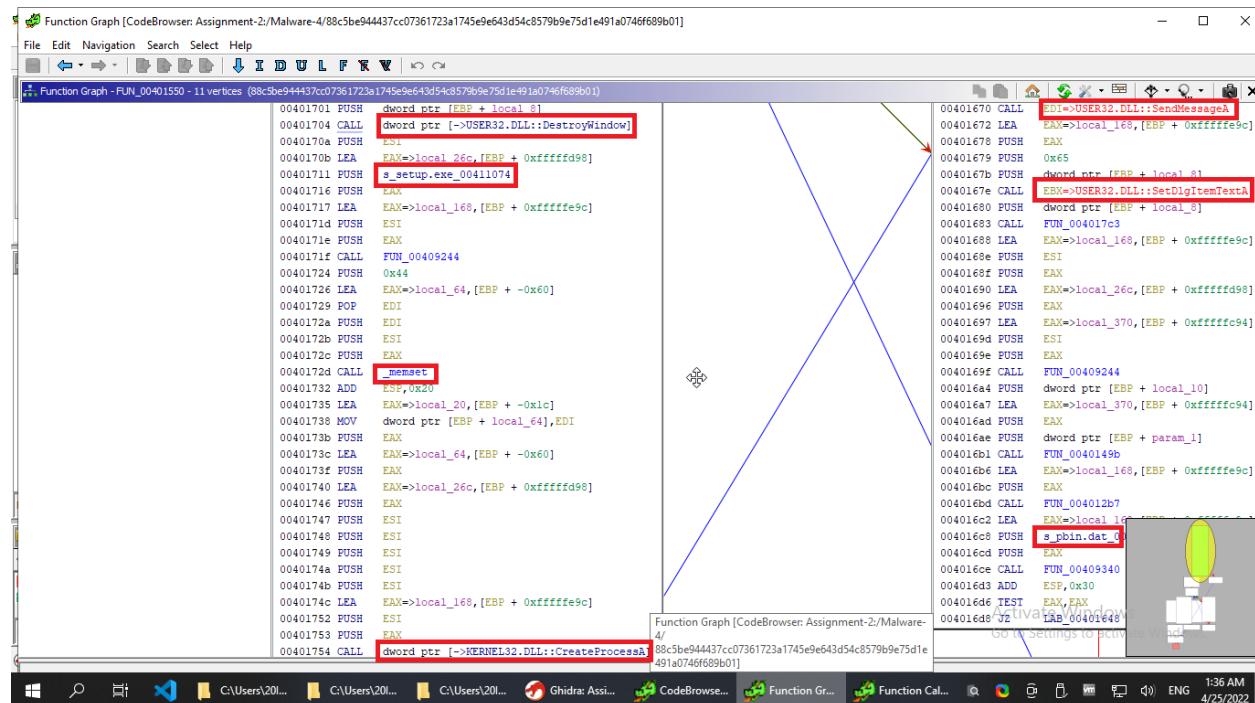


Figure 56: Malware-3: Malware installing some malicious softeare on victim mahcine

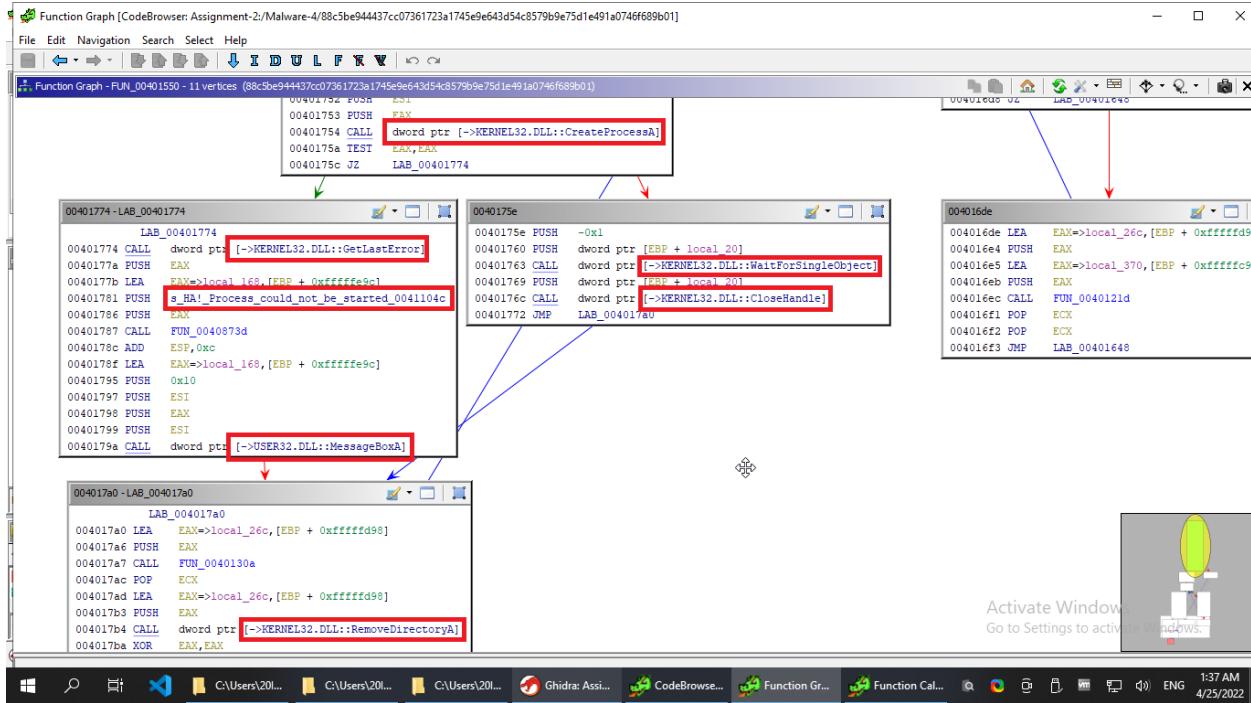


Figure 57: Malware-3: Malicious DLL fuctions used by malware

Task-4 What DLL's any malware includes? Are there any suspicious functionality called by these DLL's?

The DLL's used by this malware includes "KERNEL32.dll", "SHELL32.dll" and "USER32.dll".

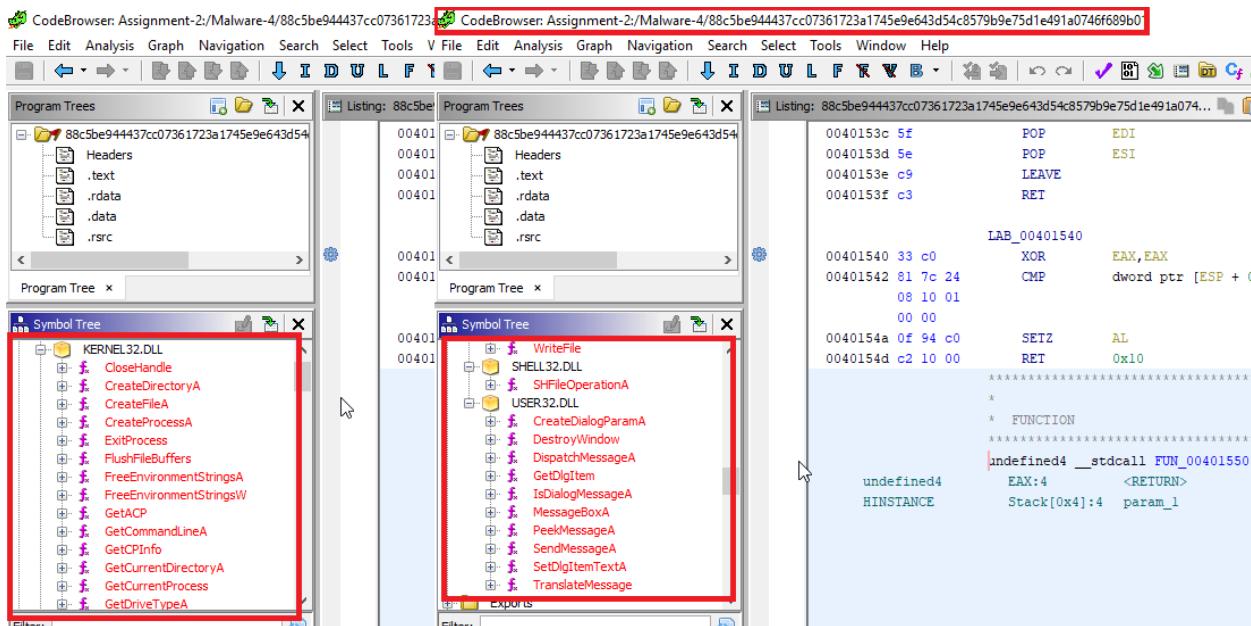


Figure 58: List of DLLs used by the malware with their functions

The suspicious functionality called by the DLLs includes the "User32.DLL" calling its function "SendMessageA", "SetDlgItemTextA", "DestroyWindow" and "MessageBoxA" and "KERNEL32.dll" calling

its functions “CreateProcessA”, “RemoveDirectoryA”, “WaitForSingleObject” and “CloseHandle” when the malicious function of malware “FUN_00401550” installs some malicious software on victims machine.

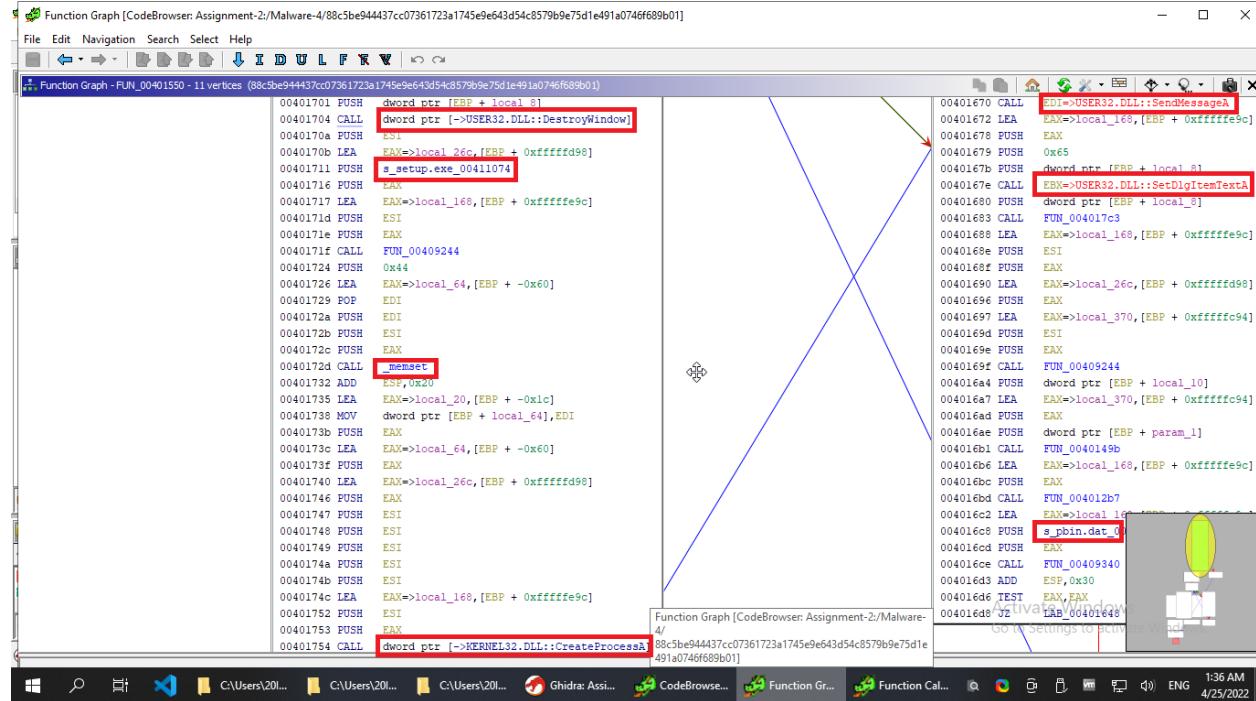


Figure 59: Malware-3 Malicious DLL functions used by malware

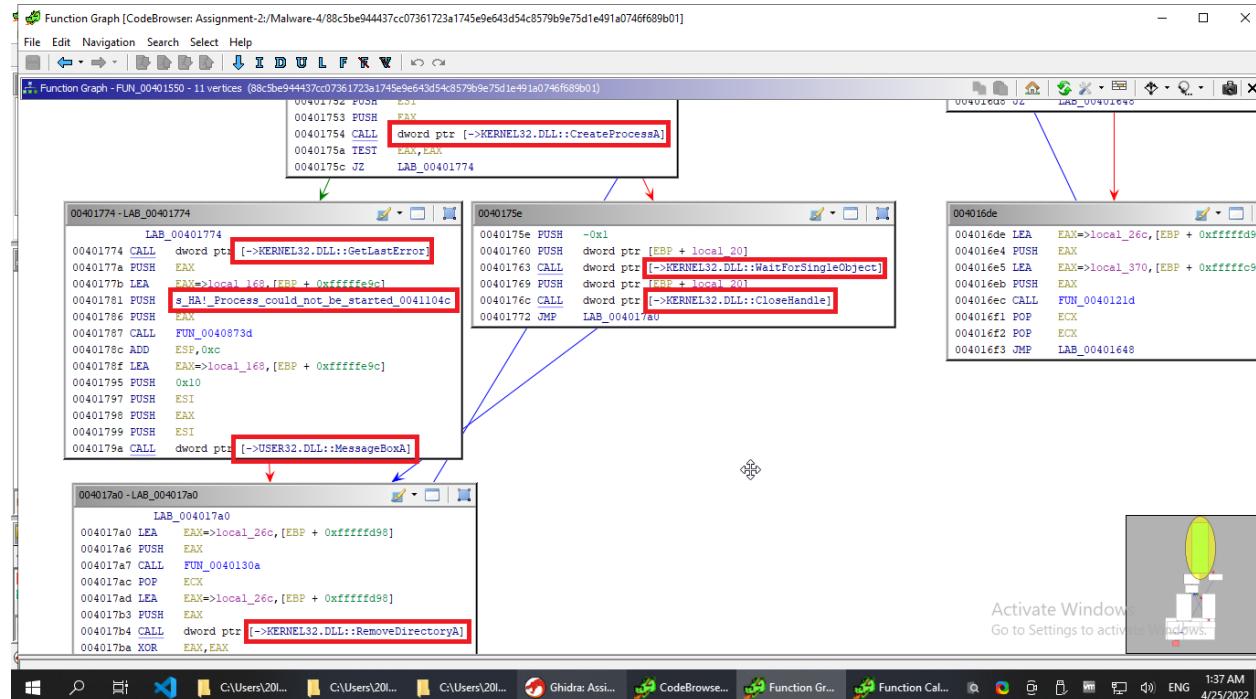


Figure 60: Malware-3 Malicious DLL functions used by malware

Malware-4 Trojan.GenericKD.3652107

This malware belongs to the family of trojans. To analyze malware I open it in Ghidra and then perform auto analysis on the malware sample.

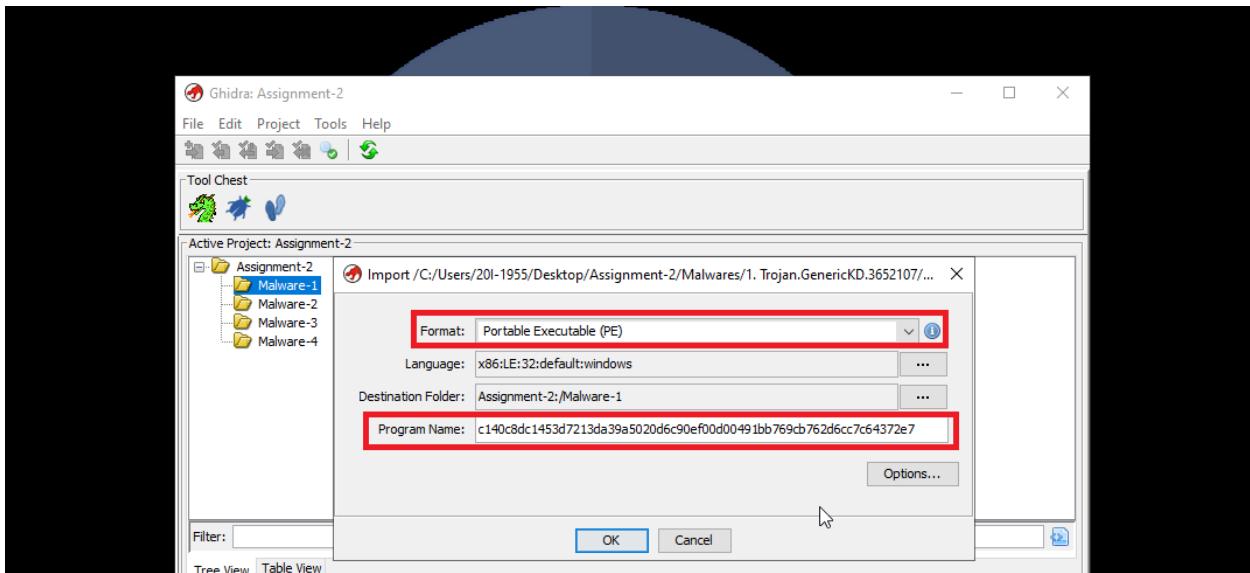


Figure 61: Malware-4: Opening malware in Ghidra

On opening the malware sample in the Ghidra I found that the malware is PE executable.

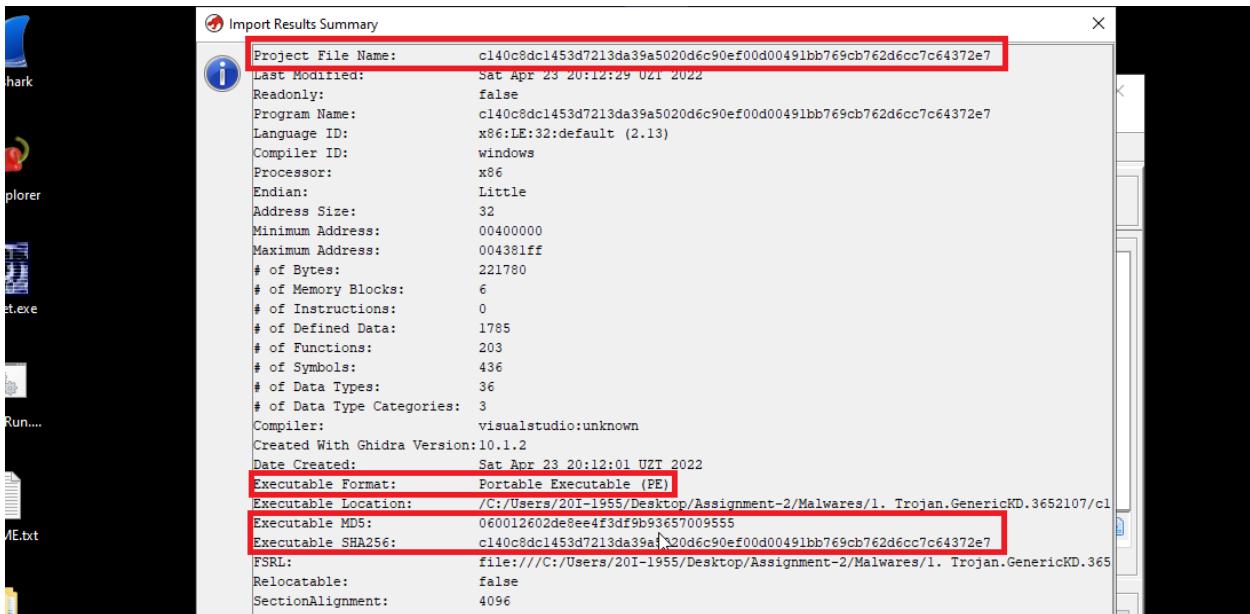


Figure 62: Malware-4: Basic details extracted by Ghidra about the malware sample

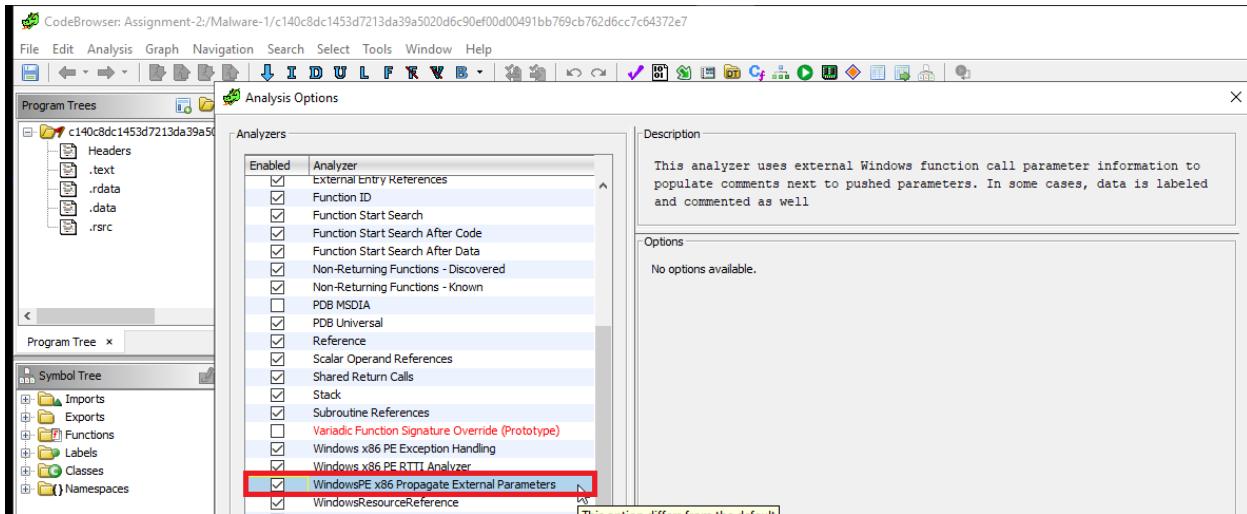


Figure 63: Malware-4: Ghidra auto analyze the malware

Task-1 What are the different segments or sections in case of each malware?

After Ghidra complete the auto analysis I found that the magic byte oif the malware which is MZ, moreover I found the details of different sections used by malware that includes “.text”, “.rdata”, “.data”, “.rsrc”.

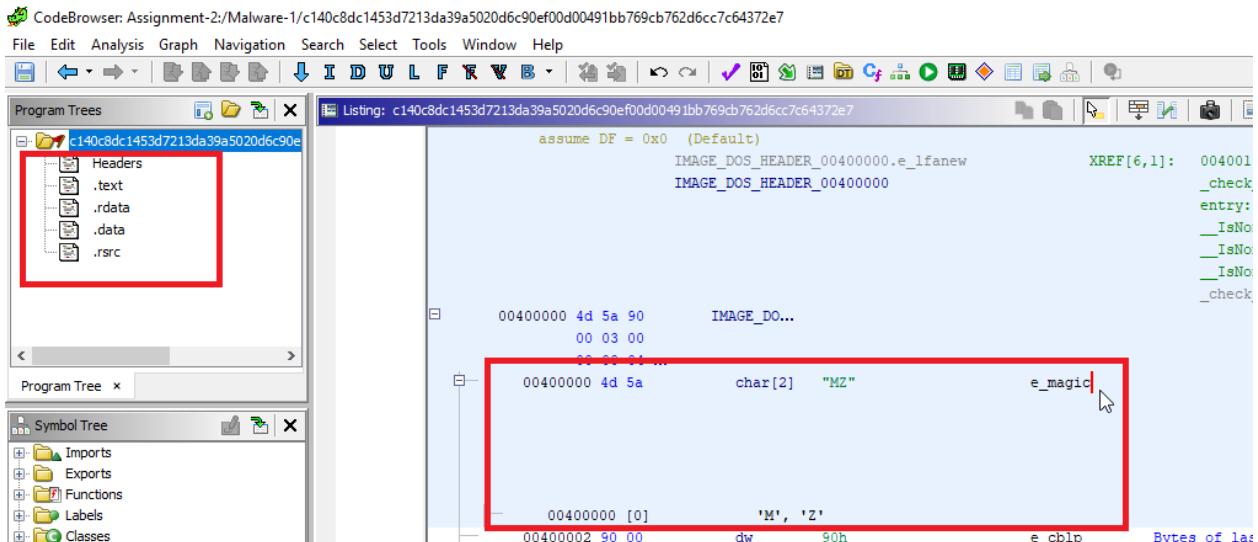


Figure 64: Malware-4: Magic byte and details of the sections used by malware

Task-2 What are different functions, imports and exports of each Malware?

To perform its functionality malware imports different DLL's that includes "ADVAPI32.dll", "COMCTL32.dll", "GDI32.dll", "KERNEL32.DLL", "OLE32.DLL", "SHELL32.dll", "SHLWAPI.dll", "USER32.dll" and "VERSION.dll". Moreover it also use different funcitons to perform it functionality.

CodeBrowser: Assignment-2/Malware-1/c140c8dc1453d7213da39a5020d6c90ef00d00491bb769cb762d6cc7c64372e7

File Edit Analysis Graph Navigation Search Select Tools Window Help

Program Trees

Listing: c140c8dc1453d7213da39a5020d6c90ef00d00491bb769cb762d6cc7c64372e7

```

assume DF = 0x0 (Default)
IMAGE_DOS_HEADER_00400000.e_lfanew
IMAGE_DOS_HEADER_00400000

00400000 4d 5a 90      IMAGE_DOS...
00 03 00
00 00 04 ...
00400000 4d 5a      char[2]  "MZ"   e_
00400000 [0]          'M', 'Z'
00400002 90 00      dw     90h   e_
00400004 03 00      dw     3h    e_
00400006 00 00      dw     0h    e_

```

Symbol Tree

Imports

- ADVAPI32.DLL
- COMCTL32.DLL
- GDI32.DLL
- KERNEL32.DLL
- OLE32.DLL
- SHELL32.DLL
- SHLWAPI.DLL
- USER32.DLL
- VERSION.dll

Exports

Figure 65: Malware-4: Details of the functions, imports and exports used by malware

CodeBrowser: Assignment-2/Malware-1/c140c8dc1453d7213da39a5020d6c90ef00d00491bb769cb762d6cc7c64372e7

File Edit Analysis Graph Navigation Search Select Tools Window Help

Program Trees

Listing: c140c8dc1453d7213da39a5020d6c90ef00d00491bb769cb762d6cc7c64372e7

```

assume DF = 0x0 (Default)
IMAGE_DOS_HEADER_00400000.e_lfanew
IMAGE_DOS_HEADER_00400000

00400000 4d 5a 90      IMAGE_DOS...
00 03 00
00 00 04 ...
00400000 4d 5a      char[2]  "MZ"   e_magic
00400000 [0]          'M', 'Z'
00400002 90 00      dw     90h   e_cblp  Bytes of last ...
00400004 03 00      dw     3h    e_cp    Pages in file ...
00400006 00 00      dw     0h    e_crlc  Relocations ...
00400008 04 00      dw     4h    e_cparhdr Size of header ...
0040000a 00 00      dw     0h    e_minalloc Minimum extra ...
0040000c ff ff      dw     FFFFh   e_maxalloc Maximum extra ...

```

Symbol Tree

Functions

- @_EH4
 - f @_EH4_CallFilterFunc@8
 - f @_EH4_LocalUnwind@16
- _CPToCID
- entry
- F
 - FID_conflict:
 - FUN_004
 - getSystemCP
- s
 - setSBCS
 - setSBUpLow
 - strchr
 - strtol
 - x ismbtotype_

Figure 66: Malware-4: Different functions used by malware

Task-3 What is flow of functions in case of each malware? Is there any suspicious function?

Give detail (name, arguments, call mechanism) of suspicious functions?

The flow of the funcitons of this malware is that it initialize the critical section, enter into the critical section perform sme malicious actions there and then deletes that critical section.

Task-3.1 FUN_crtInitCritSecNoSpinCount

In this function malware initialize the critical section to access it to perform malicious action.

The screenshot shows the CodeBrowser interface with two main panes. The left pane displays assembly code from the listing tab, and the right pane shows the corresponding decompiled C code from the decompile tab. The assembly code includes instructions like `VirtualFree`, `InitializeCriticalSection`, and `EnterCriticalSection`. The decompiled C code shows a function definition:

```

1 /* Library Function - Single Match
2 __crtInitCritSecNoSpinCount@8
3
4 Library: Visual Studio 2005 Release */
5
6 undefined4 __crtInitCritSecNoSpinCount@8(LPCRITICAL_SE...
7 {
8     InitializeCriticalSection(param_1);
9     return 1;
10 }
11
12 }
13

```

Figure 67: Malware-4: Malware initialize the critical section

The screenshot shows the CodeBrowser interface with a function graph tab selected. The graph visualization shows the flow of control between different parts of the function. The title bar indicates the current view is a function graph for the `__crtInitCritSecNoSpinCount` function.

The screenshot shows the assembly code for the `__crtInitCritSecNoSpinCount@8` function. The code includes calls to `InitializeCriticalSection` and `EnterCriticalSection`, which are highlighted with red boxes. The assembly code is as follows:

```

00411417 - __crtInitCritSecNoSpinCount@8
undefined4 __stdcall __crtInitCritSecNoSpinCount@8(LPCRITICAL_SE...
    undefined4 EAX:4 <RETURN>
    LPCRITICAL_SEC... Stack[0x4]:4 param_1
    crtInitCritSecNoS...
00411417 PUSH dword ptr [ESP + param_1]
0041141b CALL dword ptr [→KERNEL32.DLL::InitializeCriticalSection]
00411421 XOR EAX,EAX
00411423 INC EAX
00411424 RET 0x8

```

Figure 68: Malware-4: Malicious DLL fuctions used by malware

Task-3.2 FUN_lock_file2

In this function malware enter to the critical section to access it to perform malicious action.

```

/* Library Function - Single Match
 * _lock_file2
 *
 * Library: Visual Studio 2005 Release */
void __cdecl _lock_file2(int _Index, void * _File)
{
    if (_Index < 0x1) {
        _lock(_Index + 0x10);
        return;
    }
    EnterCriticalSection((LPCRITICAL_SECTION)(
        _File));
}

```

Figure 69: Malware-4: Malware enters into the critical section to perform malicious action

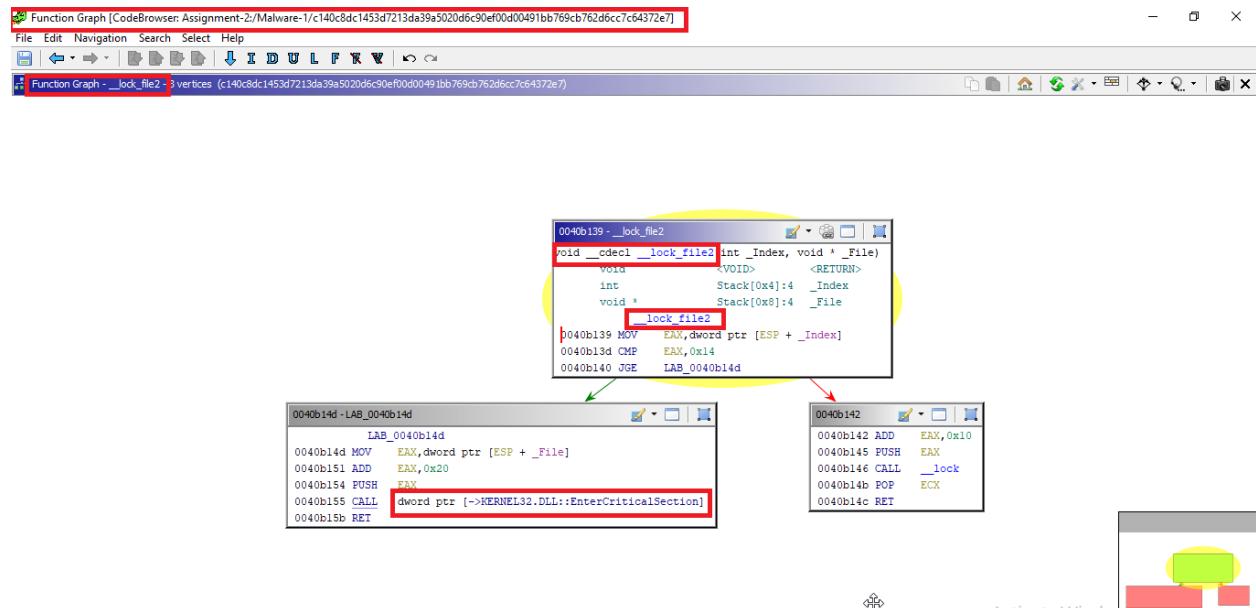


Figure 70: Malicious DLL fuctions used by malware

Task-3.3 FUN_0040bae8

In this function malware deletes to the critical section to after performing its malicious action.

```

CodeBrowser: Assignment-2/Malware-1/c140c8dc1453d7213da39a5020d6c90ef00d00491bb769cb762d6cc7c64372e7
File Edit Analysis Graph Search Select Tools Window Help
Program Trees
c140c8dc1453d7213da39a5020d6c90ef00d00491bb769cb762d6cc7c64372e7
Headers
Text
.rdata
.data
.src
Program Tree
Symbol Tree
Exports
Functions
CPt0CID
entry
local_8
local_20
local_24
local_28
local_2c
local_44
local_48
local_74
Filter:
0040bae7 c3          RET
0040bae8 al 88 e5    MOV    EAX, [DAT_0042e588]
0040bae8 42 00
0040bae8 83 ff ff    CMP    EAX, -0x1
0040bae8 74 16        JZ     LAB_0040bb08
0040bae8 50          PUSH   EAX
0040bae8 ff 35 60    PUSH   dword ptr [DAT_00434760]
0040bae8 47 43 00
0040bae8 e8 54 ff    CALL   _decode_pointer
0040bae8 ff ff
0040bae8 59          POP    ECX
0040bae8 ff d0        CALL   EAX
0040bb01 83 0d 88    OR     dword ptr [DAT_0042e588], 0xffffffff
0040bb01 e5 42 00 ff
LAB_0040bb08
0040bb08 al 8c e5    MOV    EAX, [dwTlsIndex_0042e58c]
XREF[1]: 0040bb08
undefined _stdcall FUN_0040bae8(void)
FUN_0040bae8
XREF[2]: 0040bae7
void FUN_0040bae8(void)
{
    LPCRITICAL_SECTION lpCriticalSection;
    code *pcVar1;
    LPCRITICAL_SECTION *pp_Var2;
    int iVar3;

    if (DAT_0042e588 != -1) {
        iVar3 = DAT_0042e588;
        pcVar1 = (code *)_decode_pointer(DAT_00434760);
        pp_Var2 = (LPCRITICAL_SECTION *)iVar3;
        DAT_0042e588 = -1;
    }
    if (dwTlsIndex_0042e58c != 0xffffffff) {
        TlsFree(dwTlsIndex_0042e58c);
        dwTlsIndex_0042e58c = 0xffffffff;
    }
    pp_Var2 = &lpCriticalSection_0042e6a0;
    do {
        lpCriticalSection = *pp_Var2;
        if ((lpCriticalSection != (LPCRITICAL_SECTION)0x0) {
            DeleteCriticalSection(lpCriticalSection);
            _free(lpCriticalSection);
            *pp_Var2 = (LPCRITICAL_SECTION)0x0;
        }
    }
}

```

Figure 71: Malware-4: Malware deletes the critical section after performing its malicious actions

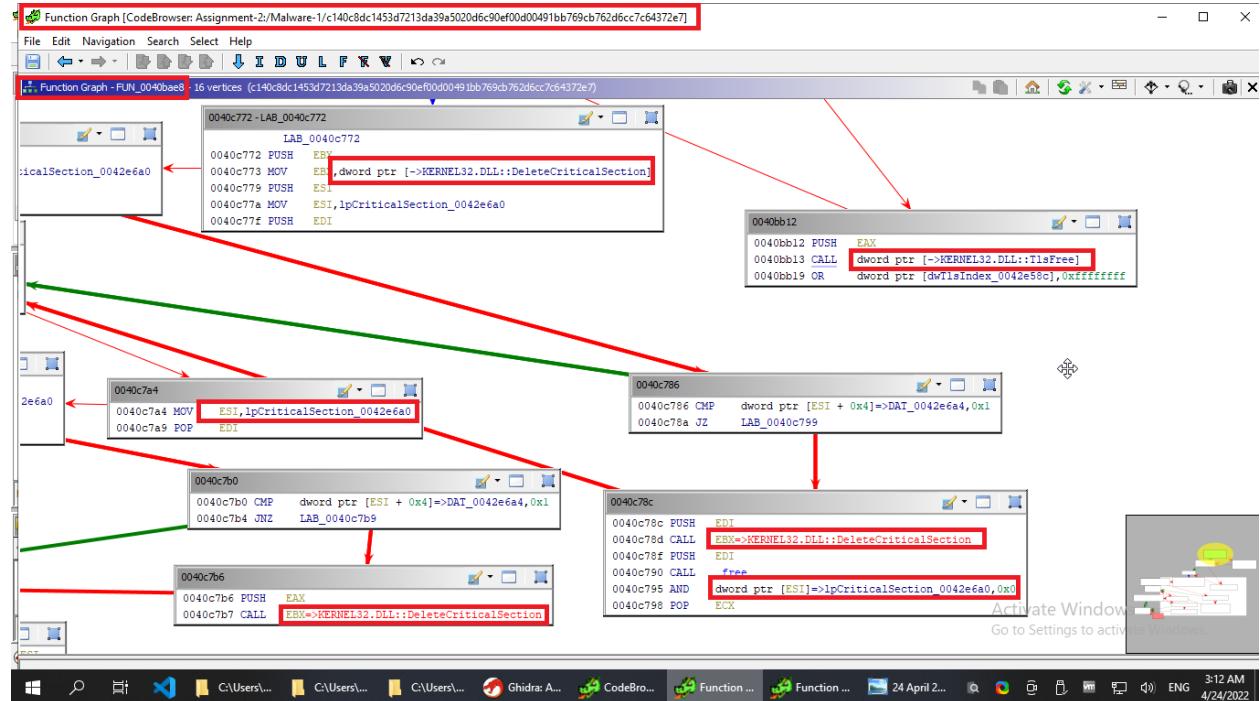


Figure 72: Malicious DLL fuctions used by malware

Task-4 What DLL's any malware includes? Are there any suspicious functionality called by these DLL's?

The DLL's used by this malware includes includes “ADVAPI32.dll”, “COMCTL32.dll”, “GDI32.dll”, “KERNEL32.DLL”, “OLE32.DLL”, “SHELL32.dll”, “SHLWAPI.dll”, “USER32.dll” and “VERSION.dll”

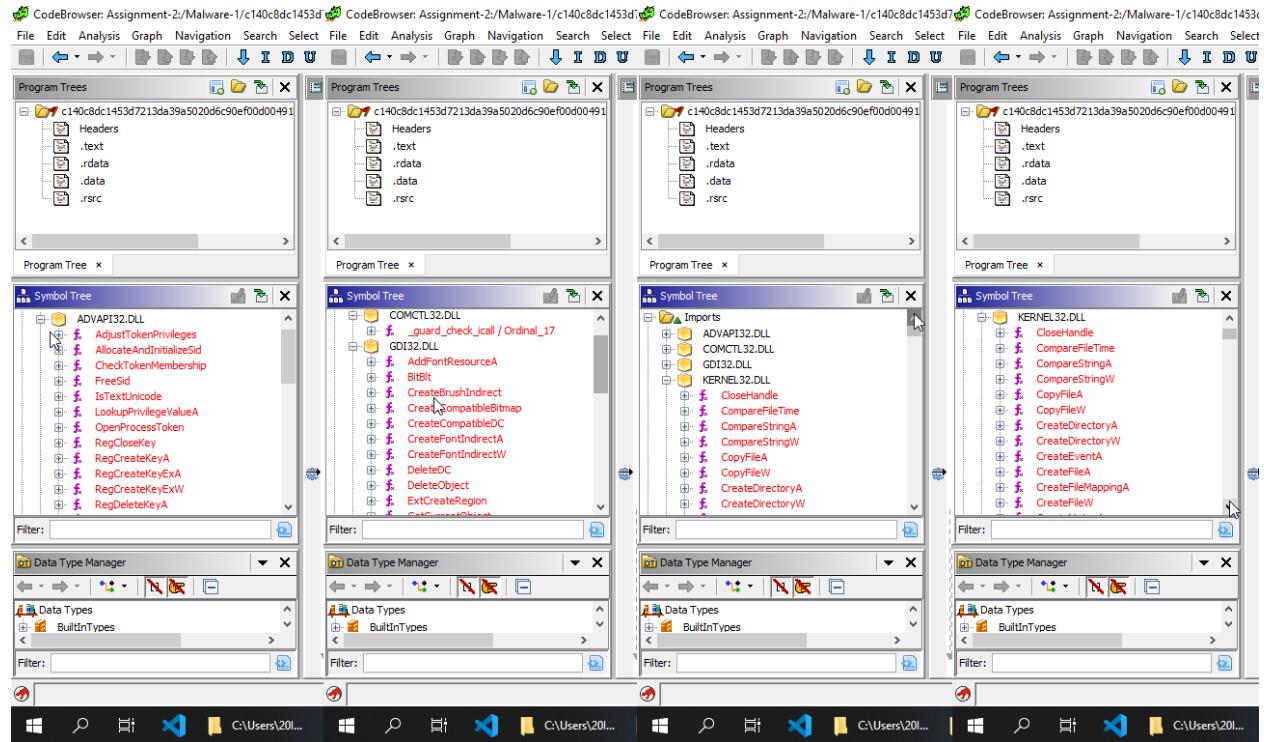


Figure 73: Malware-4: List of DLLs used by the malware with their functions

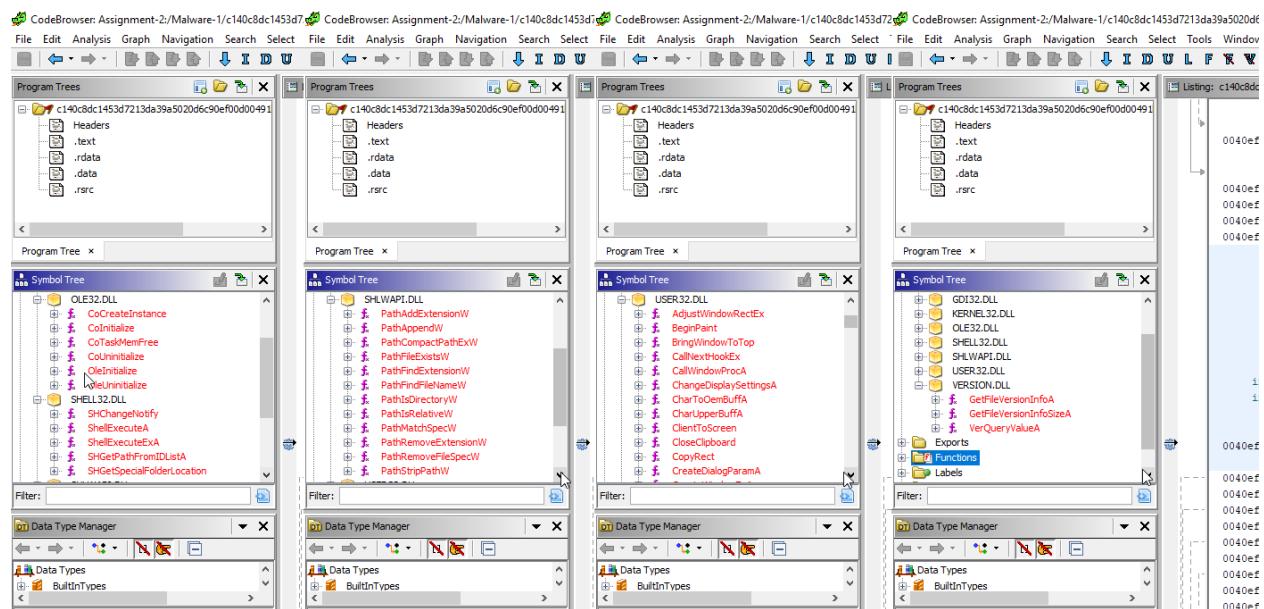


Figure 74: Malware-4: List of DLLs used by the malware with their functions

The suspicious functionality called by the DLLs includes the “KERNEL32.dll” calling its function “InitializeCriticalSection” when the malicious function of malware “FUN_crtInitCritSecNoSpinCount ” initialize the critical seciton.

```

00411417 - __crtInitCritSecNoSpinCount@8
undefined4 __stdcall __crtInitCritSecNoSpinCount@8 (LPCRITICAL SEC...
    undefined4     EAX:4      <RETURN>
    LPCRITICAL SEC... Stack[0x4] param_1
    crtInitCritSecNoS...
00411417 PUSH    dword ptr [ESP + param_1]
0041141b CALL    dword ptr [->KERNEL32.DLL::InitializeCriticalSection]
00411421 XOR     EAX,EAX
00411423 INC     EAX
00411424 RET     0x8

```

Figure 75: Malware-4 Malicious DLL functions used by malware

The suspicious functionality called by the DLLs includes the “KERNEL32.dll” calling its function “EnterCriticalSection” when the malicious function of malware “FUN_lock_file2 ” enters the critical seciton.

```

0040b139 - _lock_file2
void __cdecl _lock_file2(int _Index, void * _File)
    void     *VOID>             <RETURN>
    int      Stack[0x4]:4 _Index
    void *   Stack[0x8]:4 _File
    lock file2
0040b139 MOV     EAX,dword ptr [ESP + _Index]
0040b13d CMP     EAX,0x14
0040b140 JGE     LAB_0040b14d
0040b14d LAB_0040b14d
    LAB_0040b14d
0040b14d MOV     EAX,dword ptr [ESP + _File]
0040b151 ADD     EAX,0x20
0040b154 PUSH   EAX
0040b155 CALL    dword ptr [->KERNEL32.DLL::EnterCriticalSection]
0040b15b RET
0040b142
    0040b142 ADD     EAX,0x10
    0040b145 PUSH   EAX
    0040b146 CALL    _lock
    0040b14b POP    ECX
    0040b14c RET

```

Figure 76: Malware-4 Malicious DLL functions used by malware

The suspicious functionality called by the DLLs includes the “KERNEL32.dll” calling its function “DeleteCriticalSection” and “TlsFree” when the malicious function of malware “FUN_0040bae8” deletes the critical section after performing the malicious actions..

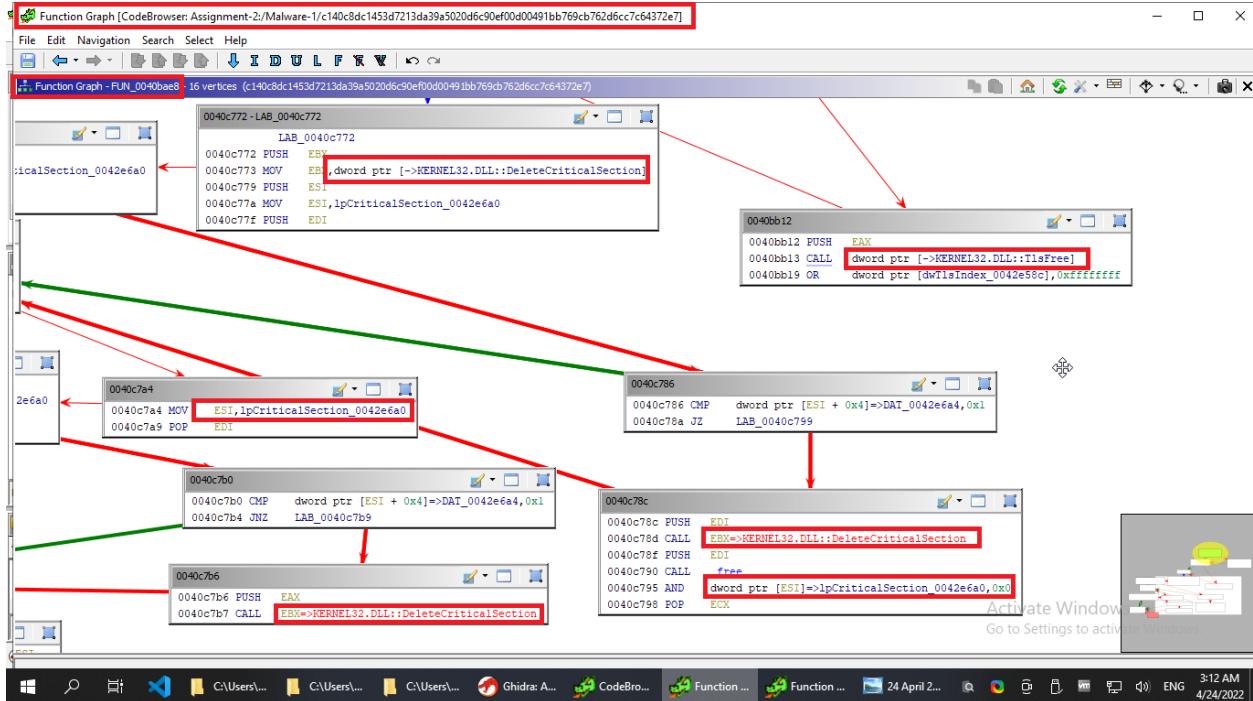


Figure 77: Malware-4 Malicious DLL functions used by malware