

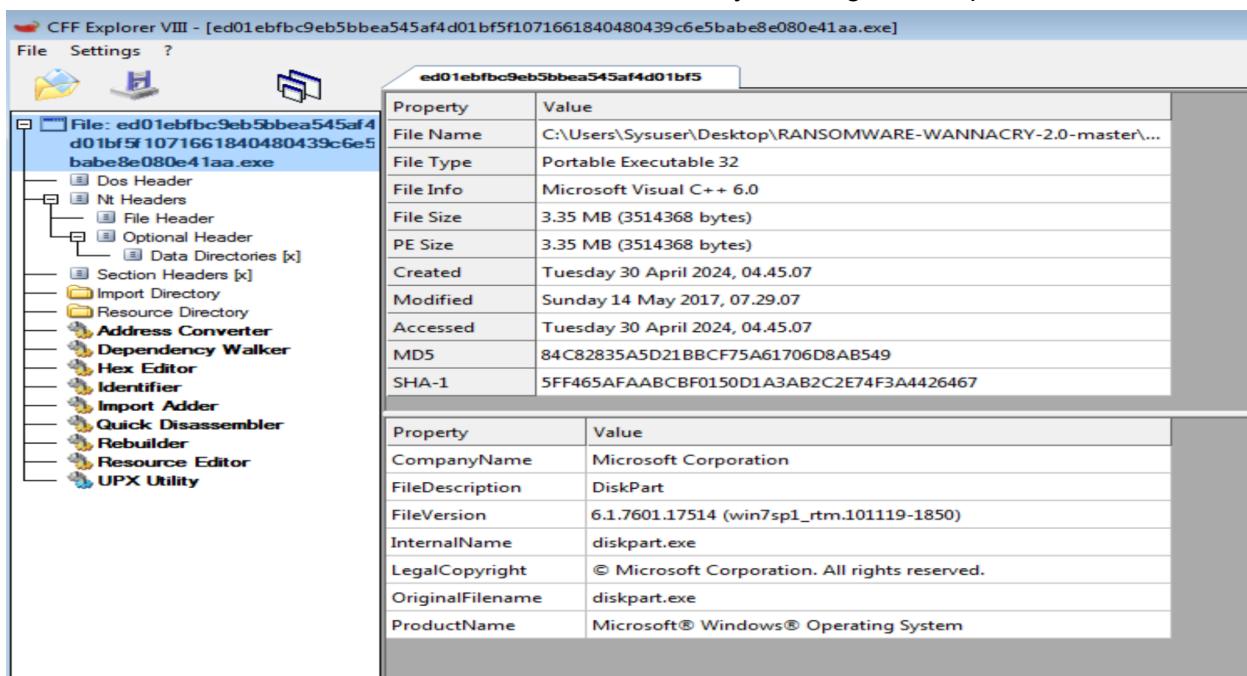
For this CSCE 451 reverse engineering project, I downloaded and analyzed WannaCry, one of the most infamous and destructive pieces of ransomware in recent history. WannaCry was initially created and spread in 2017, affecting networks throughout the world. This paper will explain the process I used to analyze the malware using static and dynamic analysis.

Initial Setup:

The WannaCry sample which I downloaded was from the following URL:
<https://github.com/chronosmiki/RANSOMWARE-WANNACRY-2.0>. I then loaded the .zip file into my VM and unzipped it there, inside which was an executable.

Static Analysis:

First, to check the properties of the file, including whether any packing utility was used to pack or obfuscate the malware, I started with some static analysis using CFF Explorer.



The screenshot shows the CFF Explorer VIII interface. The left pane displays a tree view of the file's sections and headers, with the main section selected. The right pane contains two tables of properties.

Property	Value
File Name	C:\Users\Sysuser\Desktop\RANSOMWARE-WANNACRY-2.0-master\... ed01ebfbc9eb5bbea545af4d01bf5f1071661840480439c6e5bab...e080e41aa.exe
File Type	Portable Executable 32
File Info	Microsoft Visual C++ 6.0
File Size	3.35 MB (3514368 bytes)
PE Size	3.35 MB (3514368 bytes)
Created	Tuesday 30 April 2024, 04.45.07
Modified	Sunday 14 May 2017, 07.29.07
Accessed	Tuesday 30 April 2024, 04.45.07
MD5	84C82835A5D21BBCF75A61706D8AB549
SHA-1	5FF465AFAABCBF0150D1A3AB2C2E74F3A4426467

Property	Value
CompanyName	Microsoft Corporation
FileDescription	DiskPart
FileVersion	6.1.7601.17514 (win7sp1_rtm.101119-1850)
InternalName	diskpart.exe
LegalCopyright	© Microsoft Corporation. All rights reserved.
OriginalFilename	diskpart.exe
ProductName	Microsoft® Windows® Operating System

It looks like no packing utility was used that appears in “File Info”. This seems to fit with the intent of a ransomware file like this, as packing a piece of malware usually helps with avoiding detection and making reverse engineering harder, while the primary goal of a file like this is to deny availability and damage file systems quickly, not evade detection for a long period of time.

We can also see that this file was created using C++ in May 2017, which was shortly before it attacked networks worldwide. Additionally, we can see in the lower window that the file was originally called diskpart.exe.

ed01ebfb9eb5bbea545af4d01bf5		xppacked.exe	xppacked.exe	HTTPbot.exe
Member	Offset	Size	Value	Meaning
Machine	000000FC	Word	014C	Intel 386
NumberOfSections	000000FE	Word	0004	
TimeDateStamp	00000100	Dword	4CE78F41	
PointerToSymbolTable	00000104	Dword	00000000	
NumberOfSymbols	00000108	Dword	00000000	
SizeOfOptionalHeader	0000010C	Word	00E0	
Characteristics	0000010E	Word	010F	Click here

Characteristics

File is executable
 File is a DLL
 System File
 Relocation info stripped from file
 Line numbers stripped from file
 Local symbols stripped from file
 Aggressively trim working set
 App can handle >2gb address space
 Bytes of machine word are reversed (low)
 32 bit word machine
 Debugging info stripped from file in .DBG file
 If Image is on removable media, copy and run from the swap
 If Image is on Net, copy and run from the swap file
 File should only be run on a UP machine
 Bytes of machine word are reversed (high)

We also see that this is a 32-bit executable.

For further analysis, we will load this file into Ghidra. Immediately on analyzing the file and looking at significant strings, we see several file extension names:

0040d453	- unzip 0.15 Copyrig...	"- unzip 0.15 Copyright 1998 Gilles Vollant "	ds
0040e9b4	.123	u".123"	unicode
0040e3d8	.3dm	u".3dm"	unicode
0040e3c0	.3ds	u".3ds"	unicode
0040e750	.3g2	u".3g2"	unicode
0040e738	.3gp	u".3gp"	unicode
0040e94c	.602	u".602"	unicode
0040f588	.?AVexception@@	".?AVexception@@"	ds
0040f858	.?AVtype_info@@	".?AVtype_info@@"	ds
0040e4fc	.accdb	u".accdb"	unicode
0040e8e0	.aes	u".aes"	unicode
0040e8d4	.ARC	u".ARC"	unicode
0040e4bc	.asc	u".asc"	unicode
0040e708	.ASF	u".ASF"	unicode
0040e5c8	.asm	u".asm"	unicode
0040e664	.asp	u".asp"	unicode
0040e714	.avi	u".avi"	unicode
0040e854	.backup	u".backup"	unicode
0040e8a4	.bak	u".bak"	unicode
0040e5e8	.bat	u".bat"	unicode
0040e818	.bmp	u".bmp"	unicode
0040e640	.brd	u".brd"	unicode
0040e8bc	.bz2	u".bz2"	unicode
0040e7e8	.cgm	u".cgm"	unicode
0040e690	.class	u".class"	unicode
0040e5dc	.cmd	u".cmd"	unicode
0040e5a8	.cpp	u".cpp"	unicode
0040e354	.crt	u".crt"	unicode
0040e360	.csr	u".csr"	unicode
0040e9cc	.csv	u".csv"	unicode
00400240	.data	".data"	char[8]
0040e520	.dbf	u".dbf"	unicode
0040e628	.dch	u".dch"	unicode
0040e330	.der	u".der"	unicode
0040e414	.dif	u".dif"	unicode
0040e61c	.dip	u".dip"	unicode
0040e798	.djvu	u".djvu"	unicode
0040eb70	.doc	u".doc"	unicode
0040eh58	.doch	u".doch"	unicode

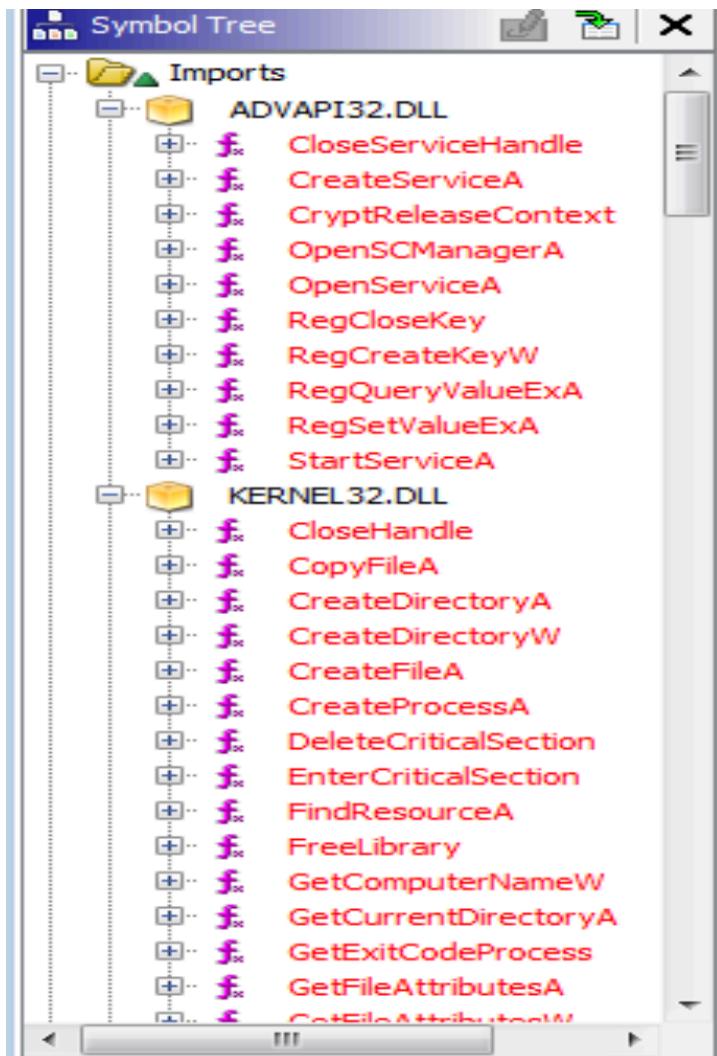
Defined Strings - 378 items			
Location	String Value	String Representation	Data Type
0040e984	.dwg	u".dwg"	unicode
0040e9fc	.edb	u".edb"	unicode
0040ea08	.eml	u".eml"	unicode
0040e6cc	.fla	u".fla"	unicode
0040e75c	.flv	u".flv"	unicode
0040e538	.frm	u".frm"	unicode
0040e800	.gif	u".gif"	unicode
0040e8ec	.gpg	u".gpg"	unicode
0040e958	.hwp	u".hwp"	unicode
0040e55c	.ibd	u".ibd"	unicode
0040e848	.iso	u".iso"	unicode
0040e684	.jar	u".jar"	unicode
0040e678	.java	u".java"	unicode
0040e830	.jpeg	u".jpeg"	unicode
0040e824	.jpg	u".jpg"	unicode
0040e64c	.jsp	u".jsp"	unicode
0040e348	.key	u".key"	unicode
0040e4a4	.lay	u".lay"	unicode
0040e4b0	.lay6	u".lay6"	unicode
0040e574	.ldf	u".ldf"	unicode
0040e780	.m3u	u".m3u"	unicode
0040e78c	.m4u	u".m4u"	unicode
0040e3cc	.max	u".max"	unicode
0040e50c	.mdb	u".mdb"	unicode
0040e568	.mdf	u".mdf"	unicode
0040e774	.mid	u".mid"	unicode
0040e744	.mkv	u".mkv"	unicode
0040e498	.mml	u".mml"	unicode
0040e720	.mov	u".mov"	unicode
0040e6a8	.mp3	u".mp3"	unicode
0040e72c	.mp4	u".mp4"	unicode
0040e6fc	.mpeg	u".mpeg"	unicode
0040e6e4	.mpg	u".mpg"	unicode
0040ea14	.msg	u".msg"	unicode
0040e544	.myd	u".myd"	unicode
0040e550	.myi	u".myi"	unicode
0040e7c4	.nef	u".nef"	unicode
0040e52c	.odb	u".odb"	unicode
0040e734	.	u". "	unicode

Defined Strings - 378 items			
Location	String Value	String Representation	Data Type
0040e474	.odg	u".odg"	unicode
0040e438	.odp	u".odp"	unicode
0040e3e4	.ods	u".ods"	unicode
0040e384	.odt	u".odt"	unicode
0040e970	.onetoc2	u".onetoc2"	unicode
0040ea20	.ost	u".ost"	unicode
0040e480	.otg	u".otg"	unicode
0040e444	.otp	u".otp"	unicode
0040e3f0	.ots	u".ots"	unicode
0040e390	.ott	u".ott"	unicode
0040e36c	.p12	u".p12"	unicode
0040e8c8	.PAQ	u".PAQ"	unicode
0040e5b4	.pas	u".pas"	unicode
0040e990	.pdf	u".pdf"	unicode
0040e378	.pem	u".pem"	unicode
0040e33c	.pxf	u".pxf"	unicode
0040e658	.php	u".php"	unicode
0040e80c	.png	u".png"	unicode
0040ea80	.pot	u".pot"	unicode
0040ea38	.potm	u".potm"	unicode
0040ea44	.potx	u".potx"	unicode
0040ea50	.ppam	u".ppam"	unicode
0040ea74	.pps	u".pps"	unicode
0040ea68	.ppsm	u".ppsm"	unicode
0040ea5c	.ppsx	u".ppsx"	unicode
0040eaa4	.ppt	u".ppt"	unicode
0040ea8c	.pptm	u".pptm"	unicode
0040ea98	.pptx	u".pptx"	unicode
0040e5f4	.ps1	u".ps1"	unicode
0040e7b8	.psd	u".psd"	unicode
0040ea2c	.pst	u".pst"	unicode
0040e870	.rar	u".rar"	unicode
0040e7f4	.raw	u".raw"	unicode
00400218	.rdata	u".rdata"	char[8]
00400268	.rsrc	u".rsrc"	char[8]
0040e9c0	.rtf	u".rtf"	unicode
0040e634	.sch	u".sch"	unicode
0040e91c	.sldm	u".sldm"	unicode
0040e928	.sldx	u".sldx"	unicode

Defined Strings - 378 items			
Location	String Value	String Representation	Data Type
0040e928	.sldx	u".sldx"	unicode
0040e420	.slk	u".slk"	unicode
0040e580	.sln	u".sln"	unicode
0040e964	.snt	u".snt"	unicode
0040e4f0	.sql	u".sql"	unicode
0040e4c8	.sqlite3	u".sqlite3"	unicode
0040e4dc	.sqitedb	u".sqitedb"	unicode
0040e408	.stc	u".stc"	unicode
0040e45c	.std	u".std"	unicode
0040e934	.sti	u".sti"	unicode
0040e3a8	.stw	u".stw"	unicode
0040e58c	.suo	u".suo"	unicode
0040e7a4	.svg	u".svg"	unicode
0040e6c0	.swf	u".swf"	unicode
0040e3fc	.sxc	u".sxc"	unicode
0040e450	.sxd	u".sxd"	unicode
0040e940	.sxi	u".sxi"	unicode
0040e48c	.sxm	u".sxm"	unicode
0040e39c	.sxw	u".sxw"	unicode
0040e898	.tar	u".tar"	unicode
0040e8b0	.tbk	u".tbk"	unicode
004001f0	.text	".text"	char[8]
0040e88c	.tgz	u".tgz"	unicode
0040e7dc	.tif	u".tif"	unicode
0040e7d0	.tiff	u".tiff"	unicode
0040e9d8	.txt	u".txt"	unicode
0040e468	.uop	u".uop"	unicode
0040e3b4	.uot	u".uot"	unicode
0040e600	.vbs	u".vbs"	unicode
0040e83c	.vcd	u".vcd"	unicode
0040e910	.vdi	u".vdi"	unicode
0040e904	.vmdk	u".vmdk"	unicode
0040e8f8	.vmx	u".vmx"	unicode
0040e6f0	.vob	u".vob"	unicode
0040e9f0	.vsd	u".vsd"	unicode
0040e9e4	.vsdx	u".vsdx"	unicode
0040e6b4	.wav	u".wav"	unicode
0040e42c	.wb2	u".wb2"	unicode
0040eada4	.xim	u".xim"	unicode
0040eb1c	.xls	u".xls"	unicode
0040eaef8	.xlsb	u".xlsb"	unicode
0040eb04	.xlsm	u".xlsm"	unicode
0040eb10	.xlsx	u".xlsx"	unicode
0040eae0	.xlt	u".xlt"	unicode
0040eab0	.xltm	u".xltm"	unicode
0040eabc	.xltx	u".xltx"	unicode
0040eaec	.xlw	u".xlw"	unicode
0040e864	.zip	u".zip"	unicode
007597ae	040904B0	u"040904B0"	unicode
0040f7b0	1.1.3	"1.1.3"	ds
0040f440	115p7UMMngoj1pM...	"115p7UMMngoj1pMvkphijcRdfJNkj6LrLn"	ds
0040f464	12t9YDPgwueZ9Ny...	"12t9YDPgwueZ9NyMgw519p7AA8isjr6SMw"	ds
0040f488	13AM4VW2dhwYgXe...	"13AM4VW2dhwYgXeQepoHkHSQuy6NgaEb94"	ds
00759a4c	6.1.7601.17514	u"6.1.7601.17514"	unicode
00759868	6.1.7601.17514 (wi...	u"6.1.7601.17514 (win7sp1_rtm.101119-1850)"	unicode

Given that WannaCry is a ransomware program, the several file extensions noted here could be the file types the malware targets. Interestingly, the strings ".exe" and ".dll" are not seen here, which suggests that WannaCry isn't encrypting executables and system files, as is the norm for ransomware. This could be useful later if we need to execute a decryption tool.

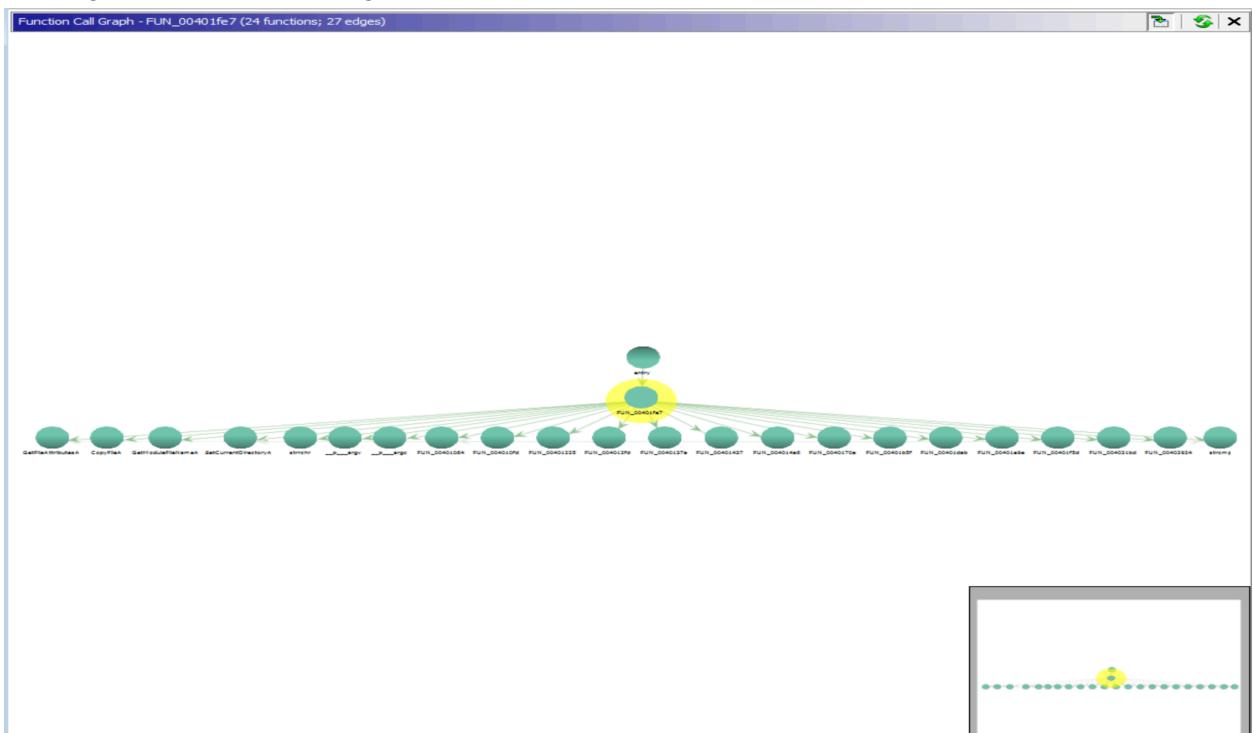
Looking at the imports on the left-hand side, we see RegCloseKey, RegCreateKey, RegSetValue, etc, which imply that this malware likely is making edits to the registry keys, as well as CreateFileA to create new files. Additionally, we see CryptReleaseContext, which is a method for using Cryptographic Service Providers (CSP). This is another common feature of ransomware, as the payment is usually demanded in cryptocurrency.



Looking at the functions, we see several unnamed functions with different input arguments:

Label	Function Signature	Function Size
FUN_00401000	004... uint FUN_0040100...	100
FUN_00401064	004... undefined4 FUN_0...	153
FUN_004010fd	004... undefined4 FUN_0...	296
FUN_00401225	004... undefined4 FUN_0...	216
FUN_004012fd	004... undefined4 * FUN...	97
FUN_0040135e	004... void * FUN_00401...	28
FUN_0040137a	004... undefined4 FUN_00...	84
FUN_004013ce	004... undefined4 FUN_0...	105
FUN_00401437	004... undefined4 FUN_0...	111
FUN_004014a6	004... byte * FUN_00401...	588
FUN_0040170a	004... undefined4 FUN_0...	211
FUN_004017dd	004... undefined4 * FUN...	34
FUN_004017ff	004... undefined4 * FUN...	28
FUN_0040181b	004... undefined4 FUN_0...	17
FUN_0040182c	004... undefined4 FUN_0...	53
FUN_00401861	004... undefined4 FUN_0...	88
FUN_004018b9	004... undefined4 FUN_0...	64
FUN_004018f9	004... undefined4 FUN_0...	199
FUN_004019e1	004... undefined4 FUN_0...	100
FUN_00401a45	004... undefined4 FUN_0...	177
FUN_00401af6	004... undefined4 FUN_0...	105
FUN_00401b5f	004... uint FUN_00401b5...	393
FUN_00401ce8	004... undefined4 FUN_0...	195
FUN_00401dab	004... undefined4 FUN_0...	243
FUN_00401e9e	004... undefined4 FUN_00...	97
FUN_00401eff	004... undefined4 FUN_0...	94
FUN_00401f5d	004... undefined4 FUN_0...	138
FUN_00401fe7	004... undefined4 FUN_0...	391
FUN_004021bd	004... undefined4 FUN_00...	44
FUN_004021e9	004... uint * FUN_00402...	622
FUN_00402457	004... undefined4 FUN_0...	25
FUN_00402470	004... undefined4 FUN_0...	219
FUN_0040254b	004... uint FUN_0040254...	260
FUN_0040264f	004... int FUN_0040264...	44
FUN_0040267b	004... uint FUN_0040267...	162
FUN_0040271d	004... undefined4 FUN_0...	59
FUN_00402758	004... uint FUN_0040275...	135
FUN_004027df	004... int FUN_004027df...	325
FUN_00402924	004... int FUN_00402924	168

Viewing the function tree to get a clearer view of how these functions interact with each other:



The entry function calls FUN_104fe7, which then calls all the other functions in the malware. This can be seen near the end of “entry”:

```
    local_r0 = local_r0 + 4;
LAB_004078ad:
} while ((*local_78 != 0) && (*local_78 < 0x21));
local_60.dwFlags = 0;
GetStartupInfoA((LPSTARTUPINFOA)&local_60);
GetModuleHandleA((LPCSTR)0x0);
local_6c = FUN_00401fe7();
/* WARNING: Subroutine does not return */
exit(local_6c);
}
```

When we navigate to the FUN_00401fe7 function, we see several more function calls within this method, as expected from all the different functions called by FUN_00401fe7.

```
SetCurrentDirectoryA(&local_210);
FUN_004010fd(1);

*(undefined2 *)puVar4 = 0;
*(undefined *)((int)puVar4 + 2) = 0;
wcscat((wchar_t *)local_d8, (wchar_t *)&_Source_0040e034);
local_c = 0;
do {
    if (local_c == 0) {
        hKey = (HKEY)0x80000002;
    }
    else {
        hKey = (HKEY)0x80000001;
    }
    RegCreateKeyW(hKey, (LPCWSTR)local_d8, (PHKEY)&local_8);
    if (local_8 != (HKEY)0x0) {
        if (param_1 == 0) {
            local_10 = 0x207;
            LVar2 = RegQueryValueExA(local_8, &DAT_0040e030, (LPDWORD)0x0, (LPDWORD)&local_10);
            bVar6 = LVar2 == 0;
            if (bVar6) {
                SetCurrentDirectoryA((LPCSTR)&local_2e0);
            }
        }
        else {
            GetCurrentDirectoryA(0x207, (LPSTR)&local_2e0);
            sVar1 = strlen((char *)&local_2e0);
            LVar2 = RegSetValueExA(local_8, &DAT_0040e030, 0, 1, &local_2e0, sVar1);
            bVar6 = LVar2 == 0;
        }
    }
}
```

Looking at the first function call, it takes in an argument 1, and handles several registry edits, including creating keys and setting them to new values. However, the actual values of the registry keys and values created are not directly visible, so I used dynamic analysis with x32dbg to find that data, as will be shown later.

Looking further down in the function, we also find another method FUN_00401e9e to assign one of 3 randomly generated cryptocurrency tokens (as shown earlier alongside the rest of the extracted strings). The assembly instructions for the method are shown below first, followed by a reconstruction of the source code with variable names and comments.

	FUN_00401e9e	XREF [1] :	FUN
00401e9e	55	PUSH	EBP
00401e9f	8b ec	MOV	EBP, ESP
00401ea1	81 ec 18	SUB	ESP, 0x318
	03 00 00		
00401ea7	8d 85 e8 fc ff ff	LEA	EAX=>local_31c, [0xfffffce8 + EBP]
00401ead	6a 01	PUSH	0x1
00401eaf	50	PUSH	EAX
00401eb0	c7 45 f4 88 f4 40 00	MOV	dword ptr [EBP + token_array[0]], s_13AM4VW2dhx...
00401eb7	c7 45 f8 64 f4 40 00	MOV	dword ptr [EBP + token_array[1]], s_12t9YDPgwue...
00401ebc	c7 45 fc 40 f4 40 00	MOV	dword ptr [EBP + token_array[2]], s_115p7UMMng...
00401ec5	e8 36 f1 ff ff	CALL	File_ReadWrite
00401eca	59	POP	ECX
00401ecb	85 c0	TEST	EAX, EAX
00401ecd	59	POP	ECX
00401ece	74 2d	JZ	LAB_00401efd
00401edo	ff 15 20 81 40 00	CALL	dword ptr [->MSVCRT.DLL::rand]
00401ed6	6a 03	PUSH	0x3
00401ed8	99	CDQ	
00401ed9	59	POP	ECX
00401eda	f7 f9	IDIV	ECX
00401edc	8d 85 9a fd ff ff	LEA	randomNum=>local_26a, [0xfffffd9a + EBP]
00401ee2	ff 74 95 f4	PUSH	dword ptr [EBP + EDX*0x4 + -0xc]
00401ee6	50	PUSH	randomNum
00401ee7	e8 bc 57 00 00	CALL	strcpy
00401eec	8d 85 e8 fc ff ff	LEA	randomNum=>local_31c, [0xfffffce8 + EBP]
00401ef2	6a 00	PUSH	0x0
00401ef4	50	PUSH	randomNum
00401ef5	e8 06 f1	CALL	File_ReadWrite
00401efa	83 c4 10	ADD	ESP, 0x10
			LAB_00401efd
00401efd	c9	LEAVE	
00401efe	c3	RET	

Using the decompiled output in Ghidra as a template, I reconstructed the source code in C syntax:

```

1 void FUN_00401e9e(void)
2 {
3     uint fileInput;
4     int randomNum;
5     undefined fileOutput [178];
6     char new_token [602];
7     char *token_array [3];
8
9     token_array[0] = s_13AM4VW2dhwYgXeQepoHkHSQuy6NgaEb_0040f488;
10    token_array[1] = s_12t9YDPgwueZ9NyMgw519p7AA8isjr6S_0040f464;
11    token_array[2] = s_115p7UMMngoj1pMvkpHijcRdfJNXj6Lr_0040f440;
12    fileInput = File_ReadWrite(fileOutput,1);
13    if (fileInput != 0) {
14        randomNum = rand();
15        strcpy(new_token,token_array[randomNum % 3]);
16        File_ReadWrite(fileOutput,0);
17    }
18    return;
19 }
20
21 }
22

```

```

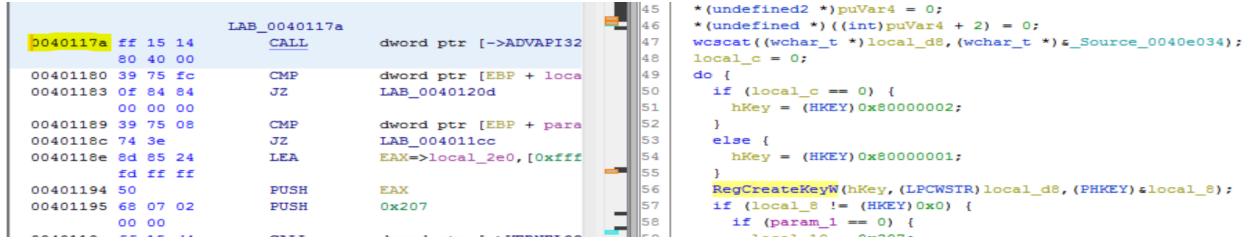
void FUN_00401e9e(void)
{
    uint fileInput;
    int randomNum;
    undefined fileOutput [178];
    char new_token [602]; // the token that will be chosen
    char *token_array [3]; // array that will take in 3 cryptographic tokens

    token_array[0] = s_13AM4WW2dhwYgXeQepoHkHSQuy6NgaEb_0040f488;
    token_array[1] = s_12t9YDPgwueZ9NyMgw519p7ZA8isj6r6S_0040f464;
    token_array[2] = s_115p7UMMngoj1pMvkrpHijcRdfEJUNXj6Lr_0040f440;
    fileInput = File_ReadWrite(fileOutput, 1); // fileOutput read from c.wnry
    // (as defined in File_ReadWrite, when arg2 is 1, read into arg1),
    // then passed to fileInput
    if (fileInput != 0) { // checks to make sure c.wnry was read properly
        // random number is generated, then reduced mod 3 to generate some number between 0 and 2
        // then that number is passed as an index for which token is picked
        randomNum = rand();
        strcpy(new_token, token_array[randomNum % 3]);
        File_ReadWrite(fileOutput, 0); // file written back to c.wnry (argument 0 = write)
    }
    return;
}

```

Debugging:

Looking at the registry edits found in the function earlier, we can now try to find the actual registry keys and values that are called with dynamic analysis. I loaded the file into x32dbg, and found the address at which the value was being set:



The screenshot shows a debugger interface with two panes. The left pane displays assembly code in Intel notation, and the right pane displays corresponding C code. The assembly code includes instructions like CALL, CMP, JZ, LEA, PUSH, and MOV. The C code is a snippet of Win32 API calls, specifically dealing with registry keys and values.

```
0040117a ff 15 14           LAB_0040117a      CALL    dword ptr [->ADVAPI32]
00401180 39 75 fc           CMP     dword ptr [EBP + local_d8]
00401183 0f 84 84           JZ    LAB_0040120d
0040118c 00 00 00
00401189 39 75 08           CMP     dword ptr [EBP + para]
0040118c 74 3e               JZ    LAB_004011cc
0040118e 8d 85 24           LEA    EAX=>local_2e0,[0xffff
00401194 50                 PUSH   EAX
00401195 68 07 02           PUSH   0x207
00401196 00 00
----- -----
```

```
45 * (undefined2 *)puVar4 = 0;
46 *(undefined *)((int)puVar4 + 2) = 0;
47 wscat((wchar_t *)local_d8,(wchar_t *)a_Source_0040e034);
48 local_c = 0;
49 do {
50     if (local_c == 0) {
51         hKey = (HKEY)0x80000002;
52     }
53     else {
54         hKey = (HKEY)0x80000001;
55     }
56     RegCreateKeyW(hKey,(LPCWSTR)local_d8,(PHKEY)&local_8);
57     if (local_8 != (HKEY)0x0) {
58         if (param_1 == 0) {
----- -----
```

Then, I set a breakpoint at that address and ran the code, and the registry key was then shown in the EAX register:

```

        Hide FPU

EAX  0018F71C      L"Software\\WanaCryptor"
EBX  00000000
ECX  0040E034      L"WanaCryptor"
EDX  0018F746
EBP  0018F7F0
ESP  0018F500
ESI  00000000
EDI  0040E030      "wd"

EIP  0040117A      ed01ebfbc9eb5bbea545af4d01bf5f1071661840480439c6e5babe8e0

EFLAGS  00000344
ZF 1  PF 1  AF 0
DF 0  SF 0  DF 0
CF 0  TF 1  IF 1

LastError  000000CB (ERROR_ENVVAR_NOT_FOUND)
LastStatus  C0000100 (STATUS_VARIABLE_NOT_FOUND)

GS 002B  FS 0053
ES 002B  DS 002B
CS 0023  SS 002B

ST(0) 00000000000000000000000000000000 x87r0 Empty 0.00000000000000000000000000000000
ST(1) 00000000000000000000000000000000 x87r1 Empty 0.00000000000000000000000000000000
ST(2) 00000000000000000000000000000000 x87r2 Empty 0.00000000000000000000000000000000
ST(3) 00000000000000000000000000000000 x87r3 Empty 0.00000000000000000000000000000000
ST(4) 00000000000000000000000000000000 x87r4 Empty 0.00000000000000000000000000000000
ST(5) 00000000000000000000000000000000 x87r5 Empty 0.00000000000000000000000000000000
ST(6) 00000000000000000000000000000000 x87r6 Empty 0.00000000000000000000000000000000
ST(7) 00000000000000000000000000000000 x87r7 Empty 0.00000000000000000000000000000000

x87TagWord FFFF
x87TW_0 3 (Empty)    x87TW_1 3 (Empty)
x87TW_2 3 (Empty)    x87TW_3 3 (Empty)
<   !!!   
```

Default (stdcall)

1: [esp]	80000002	80000002
2: [esp+4]	0018F71C	0018F71C L"Software\\WanaCryptor"
3: [esp+8]	0018F7EC	0018F7EC
4: [esp+C]	0018FEE8	0018FEE8
5: [esp+10]	75D8DBAE	<msvcrt.strrchr> (75D8DBAE)

0018F500	80000002	L"Software\\WanaCryptor"
0018F504	0018F71C	
0018F508	0018F7EC	
0018F50C	0018FEE8	
0018F510	75D8DBAE	
0018F514	00000000	
0018F518	00000000	
0018F51C	00000000	
0018F520	00000000	
0018F524	00000000	
0018F528	00000000	
0018F52C	00000000	
0018F530	00000000	
0018F534	00000000	
0018F538	00000000	
0018F53C	00000000	
0018F540	00000000	
0018F544	00000000	
0018F548	00000000	
0018F54C	00000000	
0018F550	00000000	

Looking in RegEdit, there is a directory called “Software”, and based on this method, it looks like the malware when run will create a new directory called Software\\WanaCryptor as the registry key.

Next, to find the registry value itself, I ran it with a breakpoint at the address at which RegSetValue was called:

```

004011b9 57        PUSH    EDI=>DAT_0040e030
004011ba ff 75 fc  PUSH    dword ptr [EBP + local_d8]
004011bd ff 15 18  CALL    dword ptr [->ADVAPI32]
004011c3 80 40 00
004011c5 80 f0      MOV     ESI,EAX
004011c5 f7 de      NEG     ESI
004011c7 1b f6      SBB     ESI,ESI
004011c9 1b          INC     ESI
004011ca eb 34      JMP    LAB_004011cc
004011cc 8d 45 f4  LEA    EAX=>local_10,[EBP + local_10]
004011cf c7 45 f4  MOV    EAX,dword ptr [EBP + local_10]
004011d6 07 02 00 00
004011d7 8d 85 24  LEA    EAX=>local_2eo,[0xffff]
004011d8 fd ff ff
004011dd 50        PUSH    EAX
004011de 56        PUSH    ESI
----- -->
    LAB_004011cc
004011cc 8d 45 f4  LEA    EAX=>local_10,[EBP + local_10]
004011cf c7 45 f4  MOV    EAX,dword ptr [EBP + local_10]
004011d6 50        PUSH    EAX
004011d7 8d 85 24  LEA    EAX=>local_2eo,[0xffff]
004011d8 fd ff ff
004011dd 50        PUSH    EAX
004011de 56        PUSH    ESI
----- -->

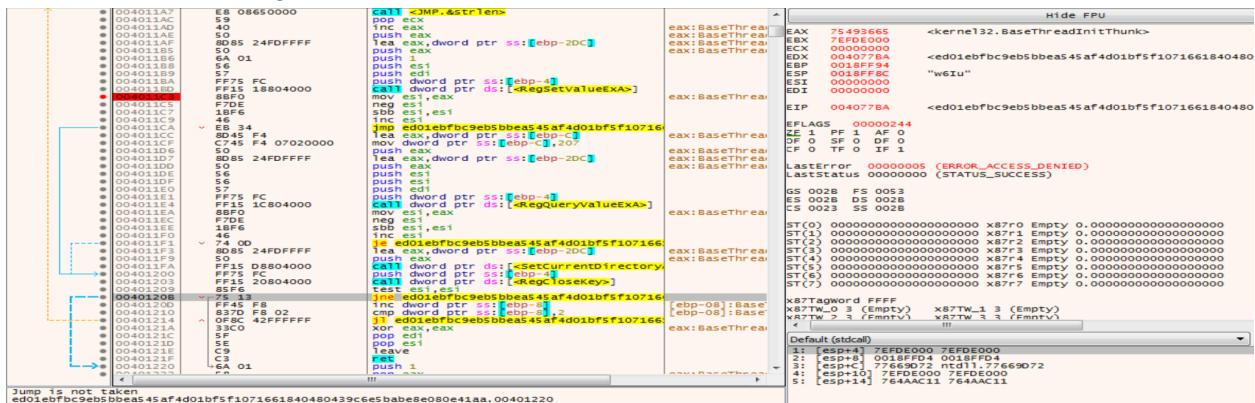
```

```

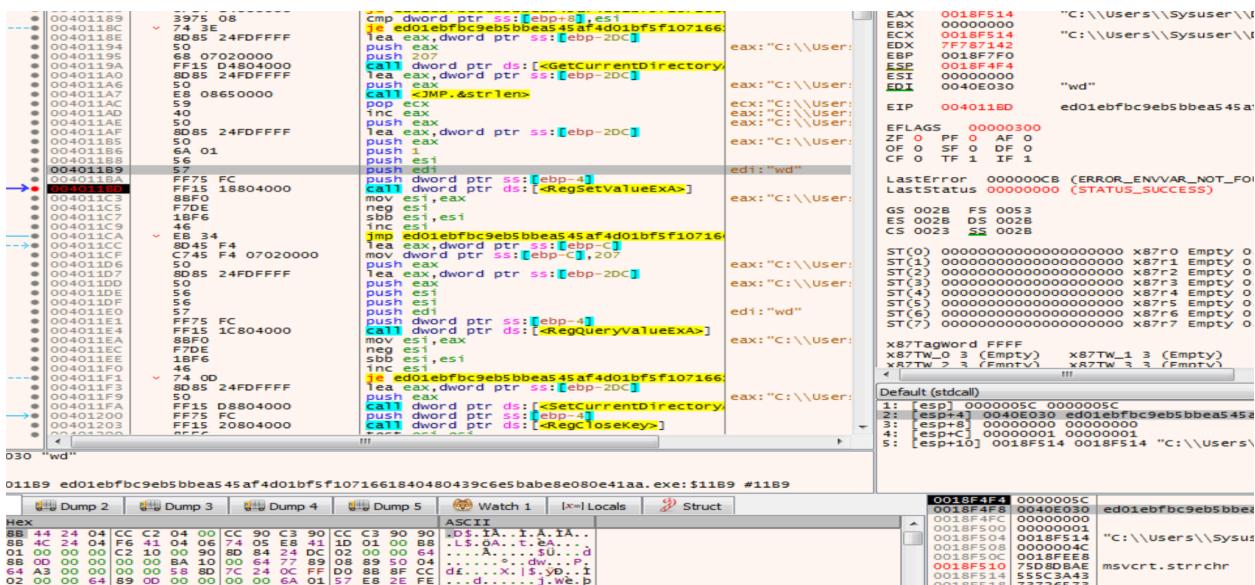
hKey = (HKEY) 0x80000001;
}
RegCreateKeyW(hKey, (LPCWSTR)local_d8, (PHKEY)&local_e);
if (local_8 != (HKEY) 0x0) {
    if (param_1 == 0) {
        local_10 = 0x207;
        LVar2 = RegQueryValueExA(local_8,&DAT_0040e030,(LPDWORD) 0x0
                                   &local_10);
        bVar6 = LVar2 == 0;
    }
    if (bVar6) {
        SetCurrentDirectoryA((LPCSTR)&local_2eo);
    }
} else {
    GetCurrentDirectoryA(0x207,(LPSTR)&local_2eo);
    sVar1 = strlen((char *) &local_2eo);
    LVar2 = RegSetValueExA(local_8,&DAT_0040e030,0,1,&local_2eo
                          &bVar6);
    LVar2 = RegSetValueExA(local_8,&DAT_0040e030,0,1,&local_2eo
                          &bVar6);
    bVar6 = LVar2 == 0;
}

```

First I tried running with a breakpoint set for the instruction after the call, but no registry values were shown in the registers.



I restarted the debugger and ran it again, setting the code to break at the function call for RegSetValueExA, and this time received more useful output.



```

EAX  0018F514    "C:\\\\Users\\\\Sysuser\\\\Desktop\\\\RANSOMWARE-WANNACRY-2.0-master\\\\Ransomware.wannaCry"
EBX  00000000
ECX  0018F514    "C:\\\\Users\\\\Sysuser\\\\Desktop\\\\RANSOMWARE-WANNACRY-2.0-master\\\\Ransomware.wannaCry"
EDX  7F787142
EBP  0018F7FO
ESP  0018F4F4
ESI  00000000
EDI  0040EO30    "wd"
EIP  004011BD    ed01ebfb9eb5bbea545af4d01bf5f1071661840480439c65bab8e080e41aa.004011BD

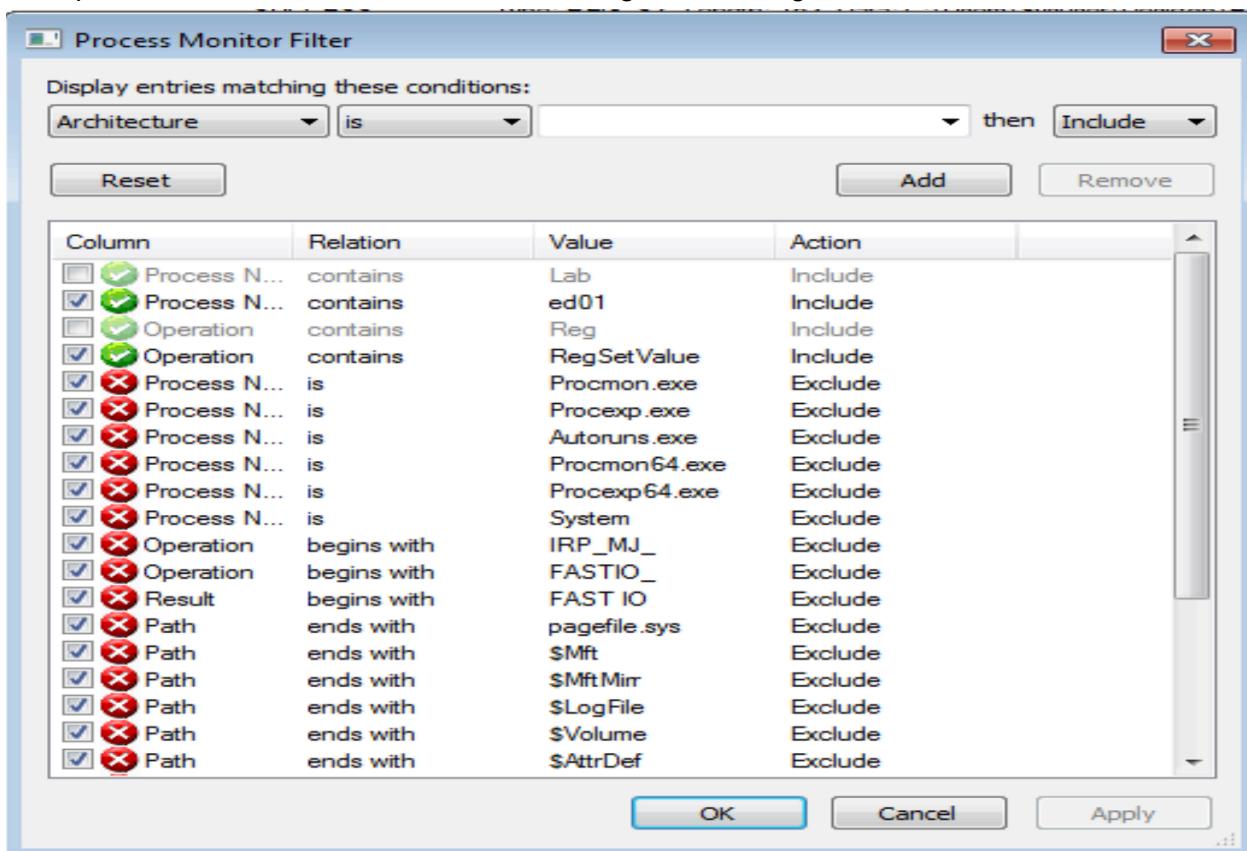
EFLAGS 00000300
ZF 0  PF 0  AF 0
OF 0  SF 0  DF 0
CF 0  TF 1  IF 1

```

Checking the EDI register, it looks like “wd” is our registry value.

Running the Malware

After checking my VM settings to make sure it was not connected to the internet and had no possible attack vectors to the host, I set up inetsim in a remnux machine, opened Wireshark, and opened ProcMon to run the malware, setting the following filters:



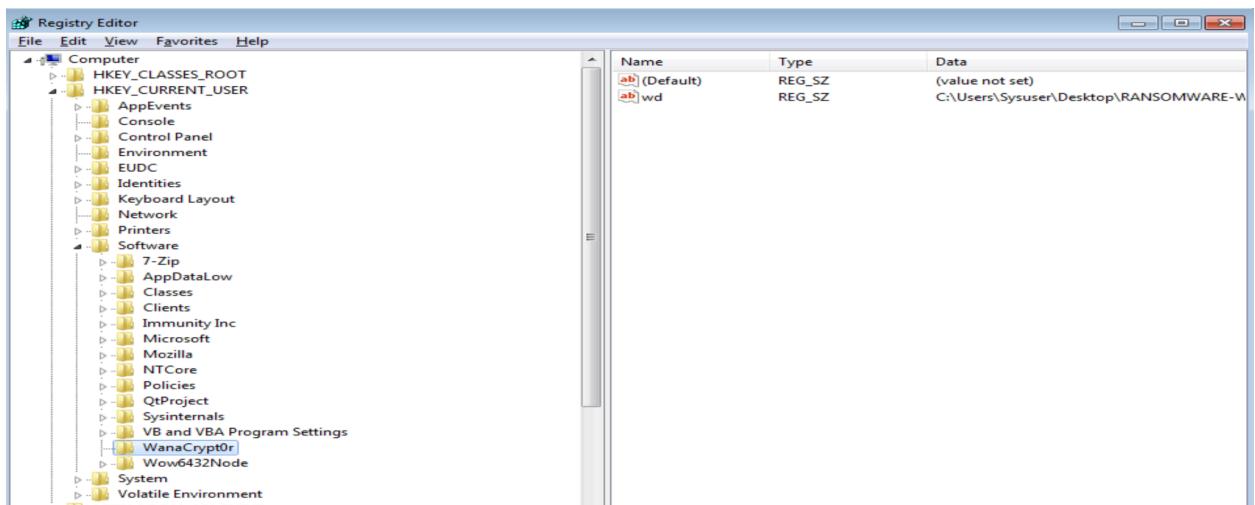
Checking the inetsim log after execution, the same DNS queries are seen as on Wireshark during the process of running the executable. Several reverse DNS lookup requests are made, which could imply attempted connection to a command-and-control server.

```

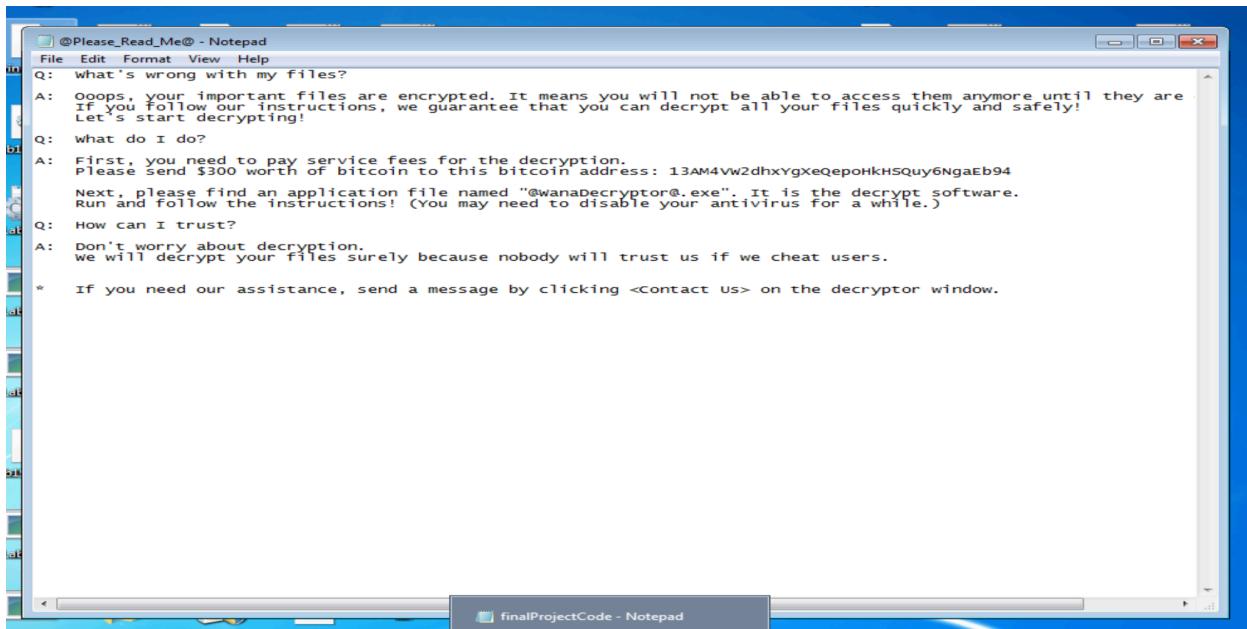
remnux@remnux: /var/log/inetsim/report
2024-04-30 18:11:03 DNS connection, type: PTR, class: IN, requested name: 45.114.11.193.in-addr.arpa
2024-04-30 18:11:07 DNS connection, type: PTR, class: IN, requested name: 151.169.101.46.in-addr.arpa
2024-04-30 18:11:07 DNS connection, type: PTR, class: IN, requested name: 38.21.59.86.in-addr.arpa
2024-04-30 18:11:26 DNS connection, type: PTR, class: IN, requested name: 49.230.47.212.in-addr.arpa
2024-04-30 18:11:26 DNS connection, type: PTR, class: IN, requested name: 189.40.188.131.in-addr.arpa
2024-04-30 18:11:36 DNS connection, type: PTR, class: IN, requested name: 222.151.101.46.in-addr.arpa
2024-04-30 18:11:46 DNS connection, type: PTR, class: IN, requested name: 52.238.254.199.in-addr.arpa
2024-04-30 18:11:56 DNS connection, type: PTR, class: IN, requested name: 93.10.7.81.in-addr.arpa
2024-04-30 18:12:32 DNS connection, type: PTR, class: IN, requested name: 212.206.109.194.in-addr.arpa
2024-04-30 18:13:08 DNS connection, type: PTR, class: IN, requested name: 246.129.210.62.in-addr.arpa
2024-04-30 18:14:30 DNS connection, type: PTR, class: IN, requested name: 244.244.23.193.in-addr.arpa
2024-04-30 18:16:28 DNS connection, type: PTR, class: IN, requested name: 240.104.10.176.in-addr.arpa
2024-04-30 18:17:42 DNS connection, type: PTR, class: IN, requested name: 39.0.31.128.in-addr.arpa
2024-04-30 18:17:42 Last simulated date in log file
===

```

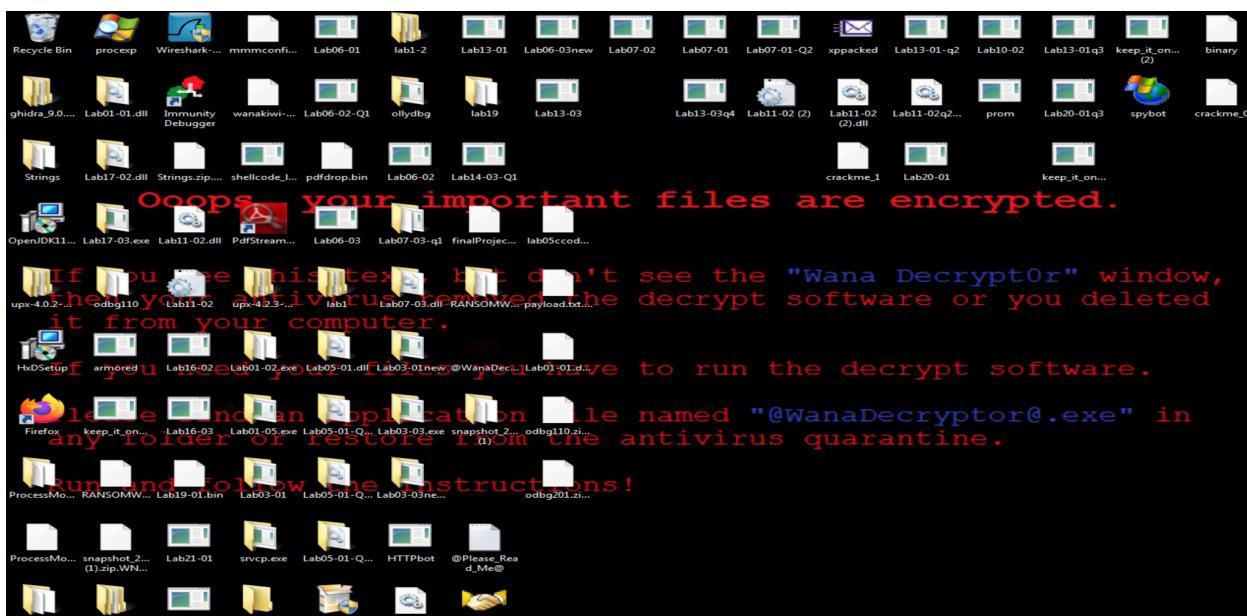
Checking process monitor, we see a new instance of “regSetValue” generated by the executable - to confirm this, I opened up the registry editor, and saw a new directory in HKCU/Software, with the same key name,WanaCrypt0r, and the same value, wd, as seen in the debugger output.



File Modifications and attempted cleanup:



This file is created, along with several new files according to the process monitor. This file seems to be a ransom note, which specifies a bitcoin address which matches one of the 3 seen in the array from the Ghidra disassembly - randomly chosen out of the 3 as per the decompiled source code.



Additionally, every non-executable file in my system has now been encrypted, and given a ".WNCRY" extension. Fortunately, I had a decryption tool WanaKiwi, which was developed after the initial WannaCry attack, and tries to reconstruct the address of the decryption key from memory. After running the decryptor and waiting for some time, the decryptor was able to recover and start decrypting my files, and then deleted the WannaCry executable itself -

however, instead of cleanly restoring my files, an encrypted copy of each file with the .WCRY extension remained, which would have to be deleted manually.