Reverse Engineering Malware Assignment

Type	32 bit Windows Executable
Filename	Wannacry.Ransomware
Md5hash	84c82835a5d21bbcf75a61706d8ab549
URL Download	https://github.com/ytisf/theZoo/blob/master/malwares/Binaries/Ransomware.WannaCry

By Ryan Ng and Chua Zhe Yu

Last Update: 29/06/2020

Table of Contents

Lab Setup

The lab setup consists of VMware Workstation 15 Pro, Interactive Dissambler (IDA) Pro and OllyDBG.

It is crucial that the malware sample is separated from the host system to prevent any damage to the host system and hence a virtual machine will be used to analyse and execute the malware in a sandbox environment.

The malware sample will be placed in the virtual machine, where it will be disassembled using IDA Pro and OllyDBG.

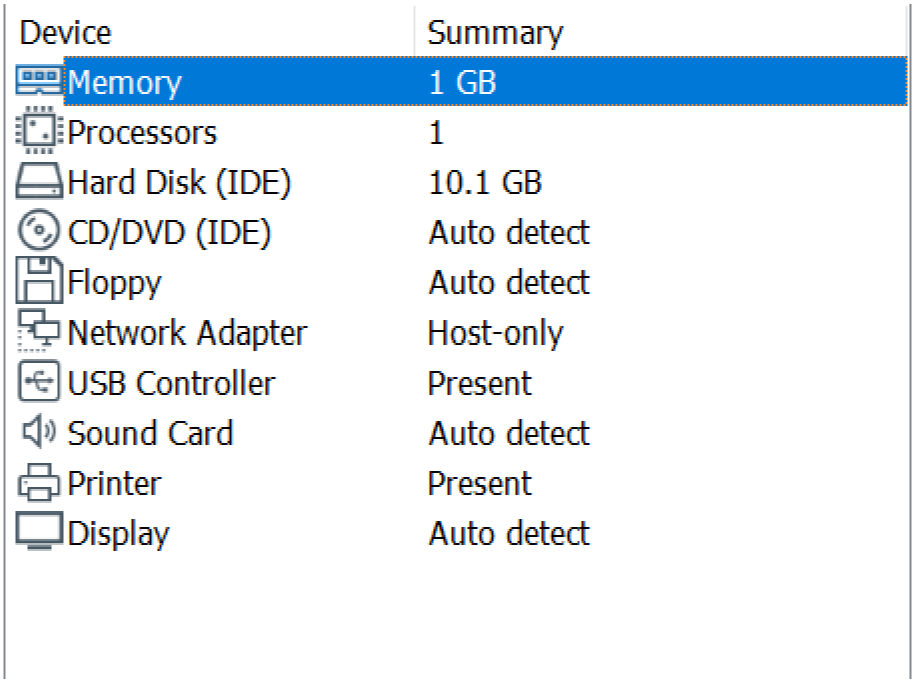
OllyDBG will be used to patch the malware. The malicious codes will be replaced by no operation codes so the malware will not be to execute.

1.1 VMware Setup

Version of VMware : Workstation 15 Pro

Host OS : Windows 10

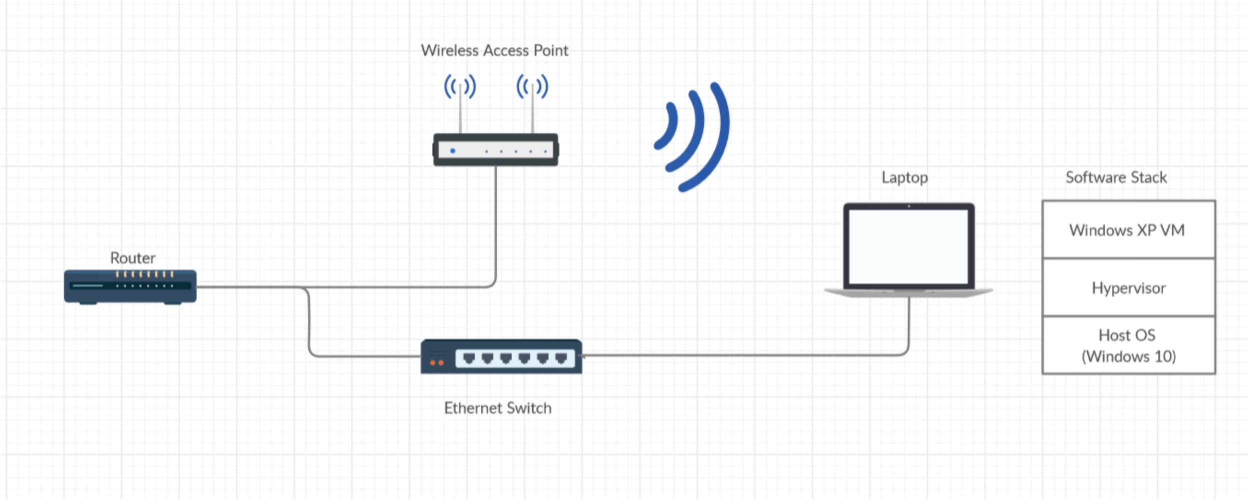
Guest OS (Virtual Machine) : Windows XP



Device	Summary
Memory	1 GB
Processors	1
Hard Disk (IDE)	10.1 GB
CD/DVD (IDE)	Auto detect
Floppy	Auto detect
Network Adapter	Host-only
USB Controller	Present
Sound Card	Auto detect
Printer	Present
Display	Auto detect

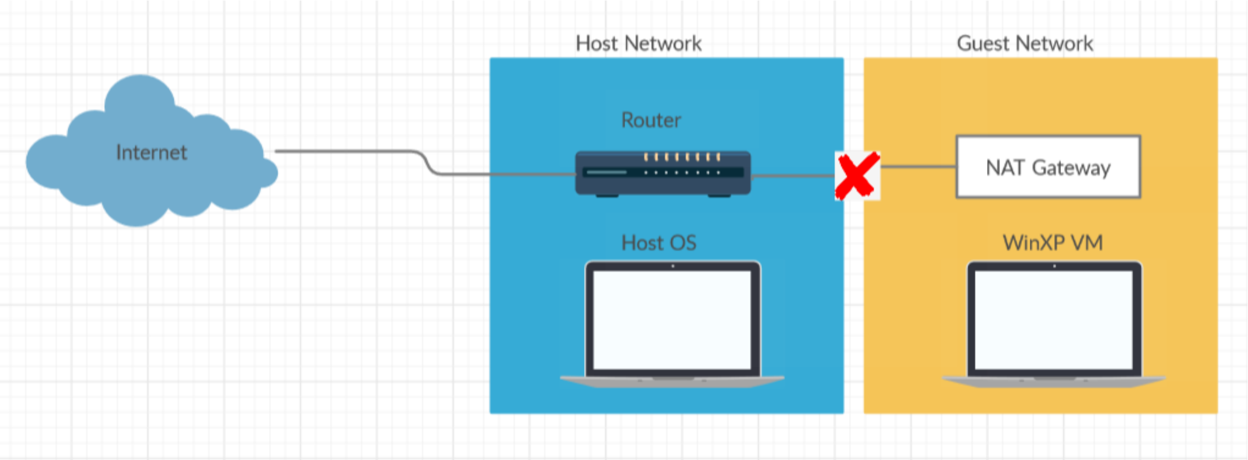
VMware Workstation Pro 15 is installed and running on the Windows 10 Host Machine. The virtual machine operates on Windows XP with 1GB of Memory, 10GB of hard disk space and uses Host-only adapter mode.

1.2 Network Diagram



Above shows the physical setup. The laptop is either connected to the switch, which is in turn connected to the router via a switch physical ethernet cable or through a wireless access point connected to (or sometimes built into) the router. There is a Windows XP Virtual Machine as depicted above running on the host computer by using the hypervisor will allow for virtualization software to be run. The malware needs to be executed for dynamic analysis in an isolated environment like the virtual machine so it does not pose any dangers to the Host OS potentially getting infected. Using the VMWare Workstation Pro VM Manager also allows us to take snapshots, which is critical so that we can revert back to the original state before dynamic analysis and rectify potential issues caused by the malware (i.e. encrypting the files).

1.3 Network Configuration

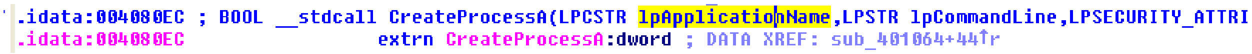


The host machine acts as a router for the Guest Network and this is done through a NAT Gateway on the Guest Network, which allows the Guest Network to use a private IP leased by the Host Network. In this manner, both Host and Guest can communicate with the Internet. However, due to safety precautions, we were advised to disable Internet connection on the Guest (Windows XP) machine.

Passive Information Gathering (IDA Pro)

1.1 Imported APIs

1.1.1 Windows API 1: CreateProcessA



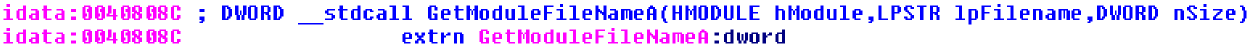
```
.idata:004080EC ; BOOL __stdcall CreateProcessA(LPCSTR lpApplicationName,LPSTR lpCommandLine,LPSECURITY_ATTRI
.idata:004080EC                 extrn CreateProcessA:dword ; DATA XREF: sub_401064+44↑r
```

Purpose: Creates and launches a new process when a logical ‘AND’ is performed value stored in eax (which is the sum of the value stored in ebp and StartupInfo) which equates to zero and thereafter the program proceeds to XOR the value in eax with itself.

Parameters: lpApplicationName, lpCommandLine, lpProcessAttributes, lpThreadAttributes, bInheritHandles, dwCreationFlags, lpEnvironment, lpCurrentDirectory, lpStartupInfo, lpProcessInformation

MSDN: https://docs.microsoft.com/en-us/windows/win32/api/processthreadsapi/nf-processthreadsapi-createprocessa

1.1.2 Windows API 2: GetModuleFileNameA



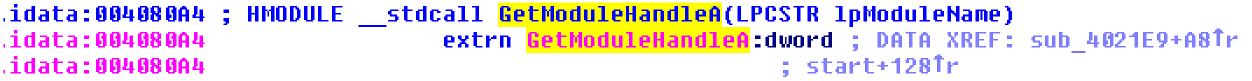
```
idata:0040808C ; DWORD __stdcall GetModuleFileNameA(HMODULE hModule,LPSTR lpFilename,DWORD nSize)
idata:0040808C                  extrn GetModuleFileNameA:dword
```

Purpose: Returns the path of a path in the specified module

Parameters: hModule, lpFilename, nSize

MSDN: https://docs.microsoft.com/en-us/windows/win32/api/libloaderapi/nf-libloaderapi-getmodulefilenamea

1.1.3 Windows API 3: GetModuleHandleA



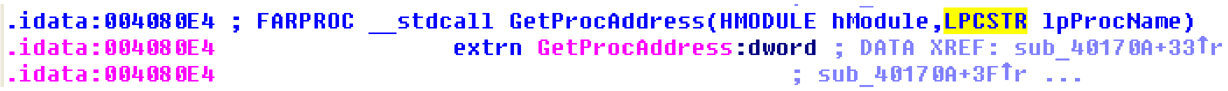
```
.idata:004080A4 ; HMODULE __stdcall GetModuleHandleA(LPCSTR lpModuleName)
.idata:004080A4                 extrn GetModuleHandleA:dword ; DATA XREF: sub_4021E9+A8↑r
.idata:004080A4                                              ; start+128↑r
```

Purpose: Used to access a loaded module in memory. This is used by the malware for code modification or injection during runtime.

Parameters: lpModuleName

MSDN: https://docs.microsoft.com/en-us/windows/win32/api/libloaderapi/nf-libloaderapi-getmodulehandlea

1.1.4 Windows API 4: GetProcAddress



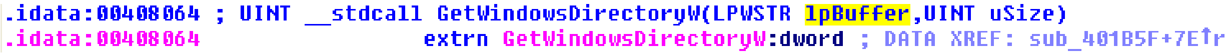
```
.idata:004080E4 ; FARPROC __stdcall GetProcAddress(HMODULE hModule,LPCSTR lpProcName)
.idata:004080E4                 extrn GetProcAddress:dword ; DATA XREF: sub_40170A+33↑r
.idata:004080E4                                         ; sub_40170A+3F↑r ...
```

Purpose: Retrieves the address of a DLL loaded into memory. Used to import functions for other DLLs. Several instances of this in the malware suggests many DLL dependencies that the malware requires.

Parameters: hModule (returned by LoadLibraryA), lpProcName

MSDN: https://docs.microsoft.com/en-us/windows/win32/api/libloaderapi/nf-libloaderapi-getprocaddress

1.1.5 Windows API 5: GetWindowsDirectoryW



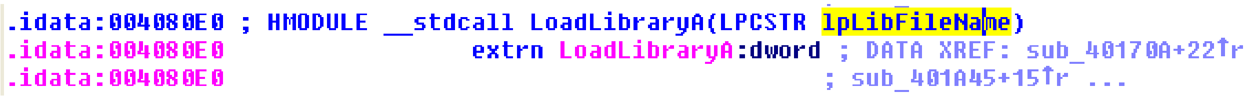
```
.idata:00408064 ; UINT __stdcall GetWindowsDirectoryW(LPWSTR lpBuffer,UINT uSize)
.idata:00408064                 extrn GetWindowsDirectoryW:dword ; DATA XREF: sub_401B5F+7E↑r
```

Purpose: Used to return the full Windows file path (C:\\ProgramData). Allows malware to install additional malicious programs.

Parameters: lpBuffer, uSize

MSDN: https://docs.microsoft.com/en-us/windows/win32/api/sysinfoapi/nf-sysinfoapi-getwindowsdirectoryw

1.1.6 Windows API 6: LoadLibraryA



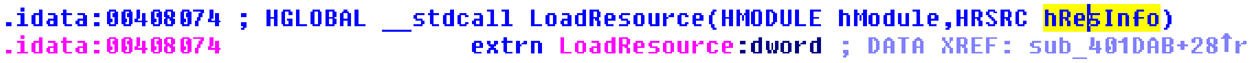
```
.idata:004080E0 ; HMODULE __stdcall LoadLibraryA(LPCSTR lpLibFileName)
.idata:004080E0                 extrn LoadLibraryA:dword ; DATA XREF: sub_40170A+22↑r
.idata:004080E0                                         ; sub_401A45+15↑r ...
```

Purpose: Loads a new DLL (aAdvapi32.dll which is an impersonation of the legitimate Advapi32.dll which is meant for event tracing) into memory. Very commonly used by Win32 programs so may not be detected as malicious.

Parameters: lpLibFileName

MSDN: https://docs.microsoft.com/en-us/windows/win32/api/libloaderapi/nf-libloaderapi-loadlibrarya

1.1.7 Windows API 7: LoadResource



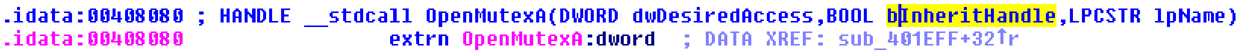
```
.idata:00408074 ; HGLOBAL __stdcall LoadResource(HMODULE hModule,HRSRC hResInfo)
.idata:00408074                     extrn LoadResource:dword ; DATA XREF: sub_401DAB+28↑r
```

Purpose: Loads resource from a PE file to memory

Parameters: hModule, hResInfo

MSDN: https://docs.microsoft.com/en-us/windows/win32/api/libloaderapi/nf-libloaderapi-loadresource

1.1.8 Windows API 8: OpenMutexA



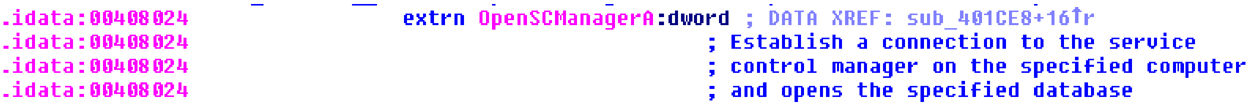
```
.idata:00408080 ; HANDLE __stdcall OpenMutexA(DWORD dwDesiredAccess,BOOL bInheritHandle,LPCSTR lpName)
.idata:00408080                     extrn OpenMutexA:dword  ; DATA XREF: sub_401EFF+32↑r
```

Purpose: This API call opens a handle to a mutual exclusion object which is used by the malware to ensure that only itself is running on the system. In this case the Mutex is named Global\MsWinZonesCacheCounterMutexA as seen in the screenshot above. This is a host-based indicator that indicates compromise of the machine.

Parameters: DesiredAccess, bInheritHandle, lpName

MSDN: No references

1.1.9 Windows API 9: OpenSCManagerA



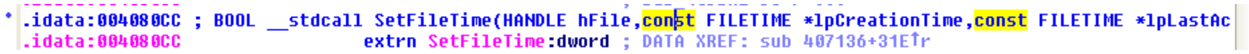
```
.idata:00408024                     extrn OpenSCManagerA:dword ; DATA XREF: sub_401CE8+16↑r
.idata:00408024                                                ; Establish a connection to the service
.idata:00408024                                                ; control manager on the specified computer
.idata:00408024                                                ; and opens the specified database
```

Purpose: This API call opens the Microsoft Security Center Service Version 2 or mssecsvc2.0 on the specified computer *(lpMachineName)* as mssecsvc.exe, which is an executable file. This is important because there is a need for the malware to call the function before manipulating the services by calling other functions. It also opens a specified database *(lpDatabaseName)*.

Parameters: lpMachineName, lpDatabaseName, dwDesiredAccess

MSDN: https://docs.microsoft.com/en-us/windows/win32/api/winsvc/nf-winsvc-openscmanagera

1.1.10 Windows API 10: SetFileTime



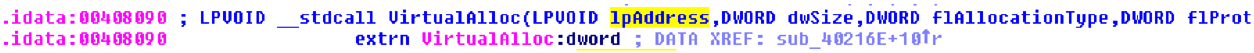
```
.idata:004080CC ; BOOL __stdcall SetFileTime(HANDLE hFile,const FILETIME *lpCreationTime,const FILETIME *lpLastAc
.idata:004080CC                 extrn SetFileTime:dword ; DATA XREF: sub 407136+31E↑r
```

Purpose: Modify the last modified/access time to cover up tracks and mask malicious activity.

Parameters: *lpCreationTime, *lpLastAccessTime, *lpLastWriteTime

MSDN: https://docs.microsoft.com/en-us/windows/win32/api/fileapi/nf-fileapi-setfiletime

1.1.11 Windows API 11: VirtualAlloc



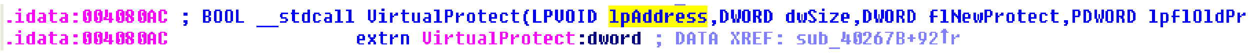
```
.idata:00408090 ; LPVOID __stdcall VirtualAlloc(LPVOID lpAddress,DWORD dwSize,DWORD flAllocationType,DWORD flProt
.idata:00408090                 extrn VirtualAlloc:dword ; DATA XREF: sub_40216E+10↑r
```

Purpose: Can be used for process injection

Parameters: lpAddress, dwSize, flAllocationType, flProtect

MSDN: https://docs.microsoft.com/en-us/windows/win32/api/memoryapi/nf-memoryapi-virtualalloc

1.1.12 Windows API 12: VirtualProtect



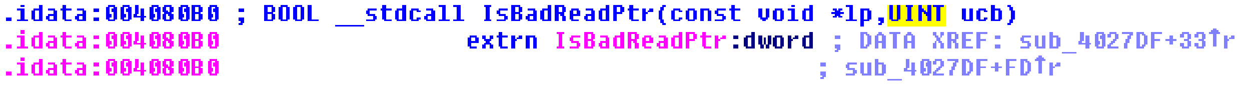
```
.idata:004080AC ; BOOL __stdcall VirtualProtect(LPVOID lpAddress,DWORD dwSize,DWORD flNewProtect,PDWORD lpflOldPr
.idata:004080AC                 extrn VirtualProtect:dword ; DATA XREF: sub_40267B+92↑r
```

Purpose: Enables malware to modify permissions of memory from read to executable

Parameters: lpAddress, dwSize, flNewProtect, lpflOldProtect

MSDN: https://docs.microsoft.com/en-us/windows/win32/api/memoryapi/nf-memoryapi-virtualprotect

1.1.13 Windows API 13:IsBadReadPtr



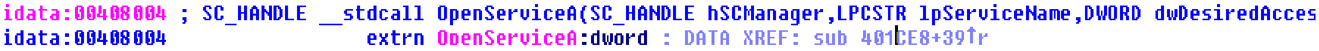
```
.idata:004080B0 ; BOOL __stdcall IsBadReadPtr(const void *lp,UINT ucb)
.idata:004080B0                 extrn IsBadReadPtr:dword ; DATA XREF: sub_4027DF+33↑r
.idata:004080B0                                         ; sub_4027DF+FD↑r
```

Purpose: This API call verifies that the calling process has read access to the specified range of memory. This is a dangerous API that is described by Microsoft to be obsolete and should not be used. The pointer referenced may not be valid or that the memory pointed to is safe to use. It should only be used for debugging purposes but is used in the malware in this case.

Parameters: *lp, ucb

MSDN: https://docs.microsoft.com/en-us/windows/win32/api/winbase/nf-winbase-isbadreadptr

1.1.14 Windows API 14:OpenServiceA



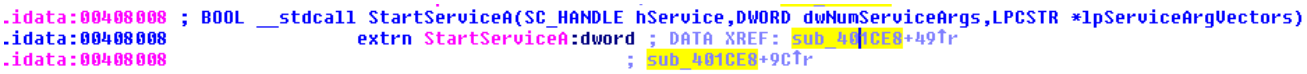
```
idata:00408004 ; SC_HANDLE __stdcall OpenServiceA(SC_HANDLE hSCManager,LPCSTR lpServiceName,DWORD dwDesiredAcces
idata:00408004                 extrn OpenServiceA:dword ; DATA XREF: sub 401CE8+39↑r
```

Purpose: Opens a service used by the malware

Parameters: hSCManager, lpServiceName, dwDesiredAccess

MSDN: https://docs.microsoft.com/en-us/windows/win32/api/winsvc/nf-winsvc-openservicea

1.1.15 Windows API 15:StartServiceA



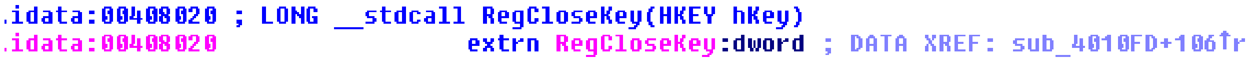
```
.idata:00408008 ; BOOL __stdcall StartServiceA(SC_HANDLE hService,DWORD dwNumServiceArgs,LPCSTR *lpServiceArgVectors)
.idata:00408008                 extrn StartServiceA:dword ; DATA XREF: sub_401CE8+49↑r
.idata:00408008                                           ; sub_401CE8+9C↑r
```

Purpose: Starts a service

Parameters: hService, dwNumServiceArgs, *lpServiceArgVectors

MSDN: https://docs.microsoft.com/en-us/windows/win32/api/winsvc/nf-winsvc-startservicea

1.1.16 Windows API 16:RegCloseKey



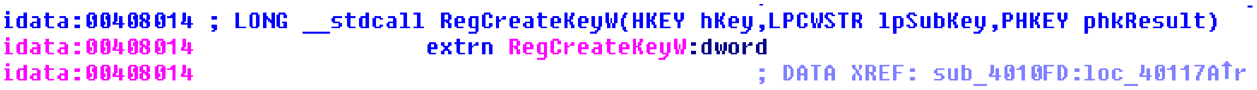
```
.idata:00408020 ; LONG __stdcall RegCloseKey(HKEY hKey)
.idata:00408020                 extrn RegCloseKey:dword ; DATA XREF: sub_4010FD+106↑r
```

Purpose: Closes a handle to the specified registry key.

Parameters: hKey

MSDN: https://docs.microsoft.com/en-us/windows/win32/api/winreg/nf-winreg-regclosekey

1.1.17 Windows API 17: RegCreateKeyW



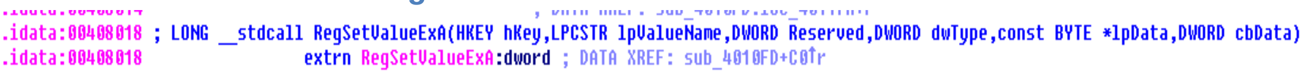
```
idata:00408014 ; LONG __stdcall RegCreateKeyW(HKEY hKey,LPCWSTR lpSubKey,PHKEY phkResult)
idata:00408014                 extrn RegCreateKeyW:dword
idata:00408014                                          ; DATA XREF: sub_4010FD:loc_40117A↑r
```

Purpose: Create registry key for 16 bit applications

Parameters: hKey, lpSubKey, phkResult

MSDN: https://docs.microsoft.com/en-us/windows/win32/api/winreg/nf-winreg-regcreatekeyw

1.1.18 Windows API 18: RegSetValueExA



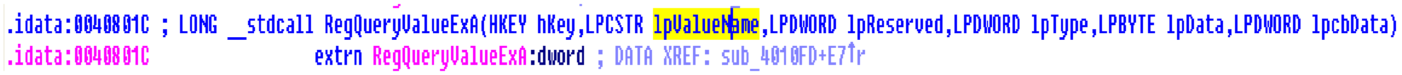
```
.idata:00408018 ; LONG __stdcall RegSetValueExA(HKEY hKey,LPCSTR lpValueName,DWORD Reserved,DWORD dwType,const BYTE *lpData,DWORD cbData)
.idata:00408018                 extrn RegSetValueExA:dword ; DATA XREF: sub_4010FD+C0↑r
```

Purpose: Set registry value

Parameters: hKey, lpValueName, Reserved, dwType, *lpData, cbData

MSDN: https://docs.microsoft.com/en-us/windows/win32/api/winreg/nf-winreg-regsetvalueexa

1.1.19 Windows API 19: RegQueryValueExA



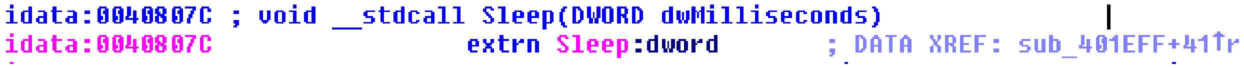
```
.idata:0040801C ; LONG __stdcall RegQueryValueExA(HKEY hKey,LPCSTR lpValueName,LPDWORD lpReserved,LPDWORD lpType,LPBYTE lpData,LPDWORD lpcbData)
.idata:0040801C                 extrn RegQueryValueExA:dword ; DATA XREF: sub_4010FD+E7↑r
```

Purpose: Retrieves the type and data for the specified value name associated with an open registry key.

Parameters: hKey, lpValueName, lpReserved, lpType, lpData, lpcbData

MSDN: https://docs.microsoft.com/en-us/windows/win32/api/winreg/nf-winreg-regqueryvalueexa

1.1.20 Windows API 20: Sleep



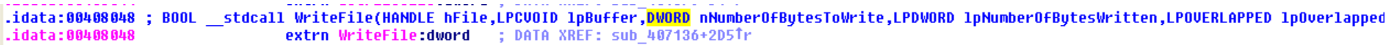
```
idata:0040807C ; void __stdcall Sleep(DWORD dwMilliseconds)
idata:0040807C                 extrn Sleep:dword       ; DATA XREF: sub_401EFF+41↑r
```

Purpose: Make the malware action undetectable for a set period of time to evade antivirus detection

Parameters: dwMilliseconds

MSDN: https://docs.microsoft.com/en-us/windows/win32/api/synchapi/nf-synchapi-sleep

1.1.21 Windows API 21: WriteFile



```
.idata:00408048 ; BOOL __stdcall WriteFile(HANDLE hFile,LPCVOID lpBuffer,DWORD nNumberOfBytesToWrite,LPDWORD lpNumberOfBytesWritten,LPOVERLAPPED lpOverlapped
.idata:00408048                 extrn WriteFile:dword   ; DATA XREF: sub_407136+2D5↑r
```

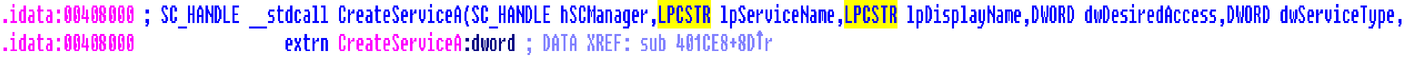
Purpose: Write data to a specified file

Parameters: hFile, lpBuffer, nNumberOfBytesToWrite, lpNumberOfBytesWritten, lpOverlapped

MSDN: https://docs.microsoft.com/en-us/windows/win32/api/fileapi/nf-fileapi-writefile

1.1.22 Windows API 22: CreateServiceA

1.1.23



```
.idata:00408000 ; SC_HANDLE __stdcall CreateServiceA(SC_HANDLE hSCManager,LPCSTR lpServiceName,LPCSTR lpDisplayName,DWORD dwDesiredAccess,DWORD dwServiceType,
.idata:00408000                 extrn CreateServiceA:dword ; DATA XREF: sub 401CE8+8D↑r
```

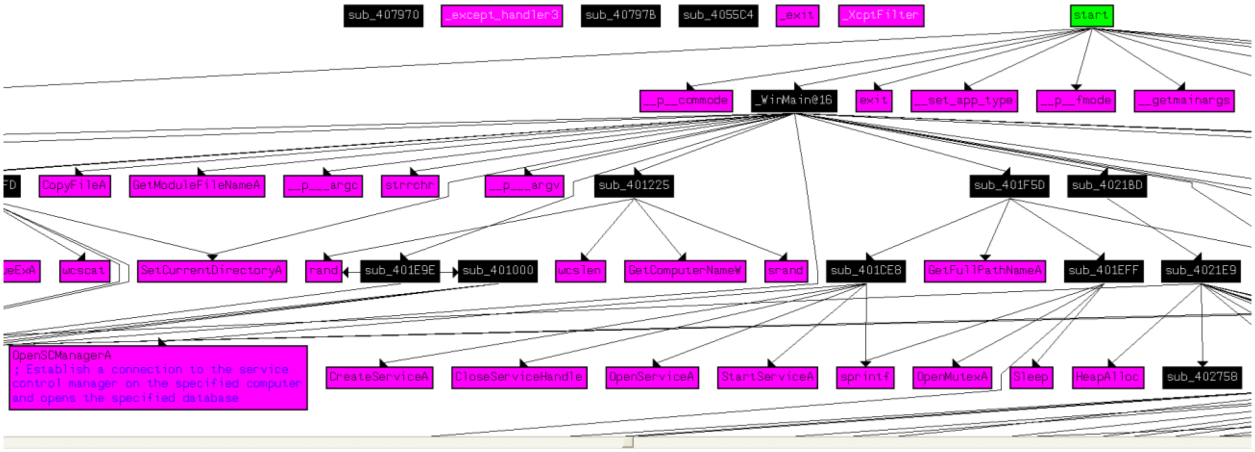
Purpose: Creates a service that can be started at boot time for persistence and loading of kernel drivers.

Parameters: lpServiceName, lpDisplayName, dwDesiredAccess, dwServiceType, dwStartType, dwErrorControl, lpBinaryPathName, lpLoadOrderGroup, lpdwTagId, lpDependencies, lpServiceStartName, lpPassword, hSCManager

MSDN: https://docs.microsoft.com/en-us/windows/win32/api/winsvc/nf-winsvc-createservicea

Code Analysis (IDA Pro)

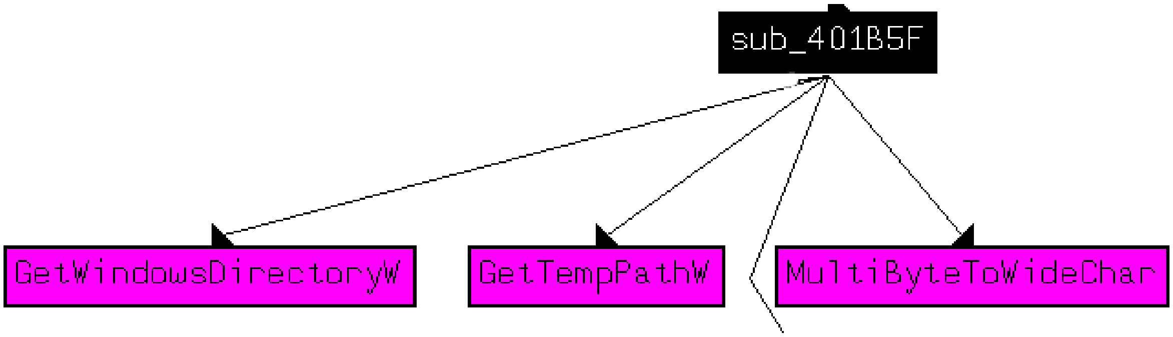
2.1 Graph of Major Subroutines from Main Function



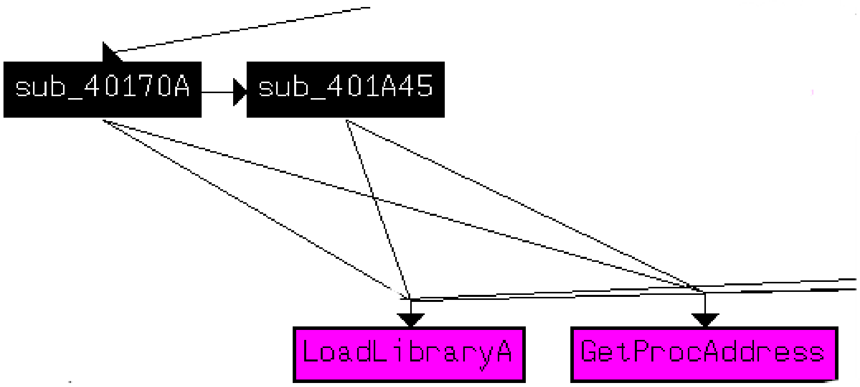
; Establish a connection to the service control manager on the specified computer and opens the specified database

Important Subroutines to Analyse:

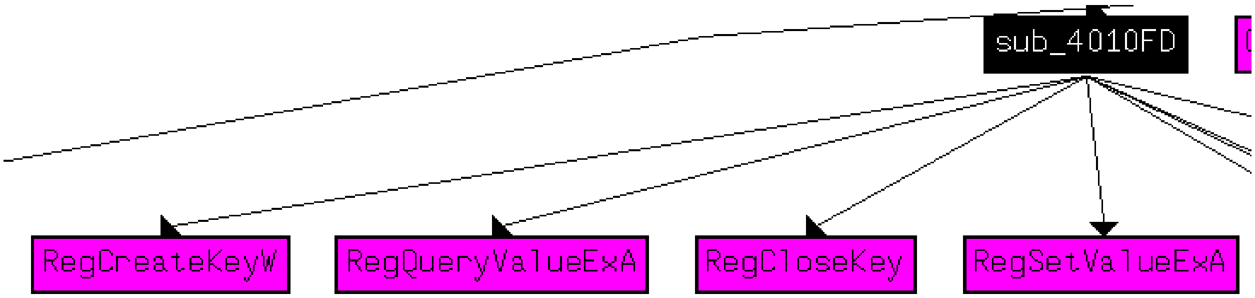
sub_401B5F (GetWindowsDirectoryW)



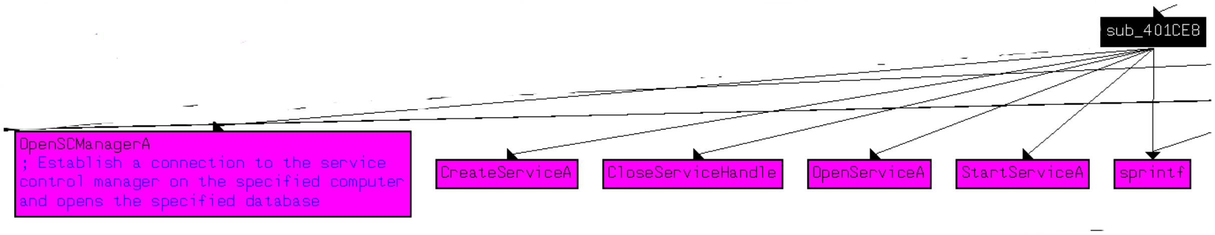
sub_40170A (LoadLibraryA, GetProcAddress) and sub_401A45 (LoadLibraryA, the keys stuff)



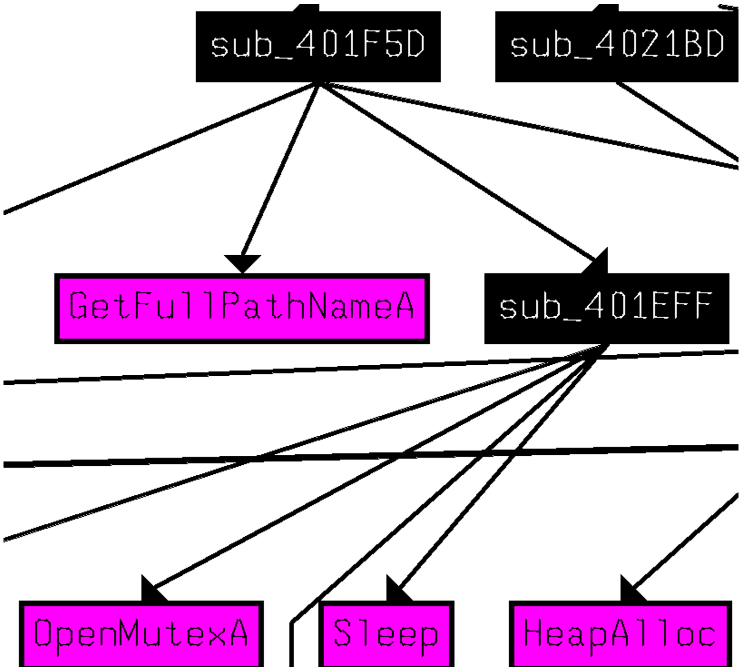
sub_4010FD (RegCreateKeyW, RegQueryValueExA, RegCloseKey, RegSetValueA)



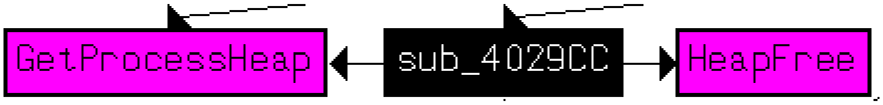
sub_401CE8 (OpenSCManager, OpenServiceA, StartServiceA)


sub_401CE8
OpenSCManagerA
; Establish a connection to the service control manager on the specified computer and opens the specified database
CreateServiceA
CloseServiceHandle
OpenServiceA
StartServiceA
sprintf

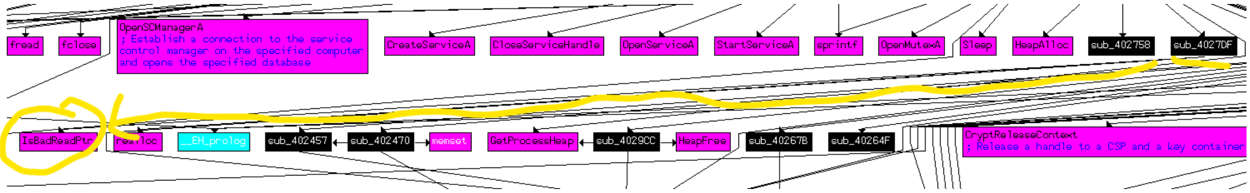
sub_401EFF (OpenMutexA, Sleep)



sub_4029CC (GetProcessheap, HeapFree)

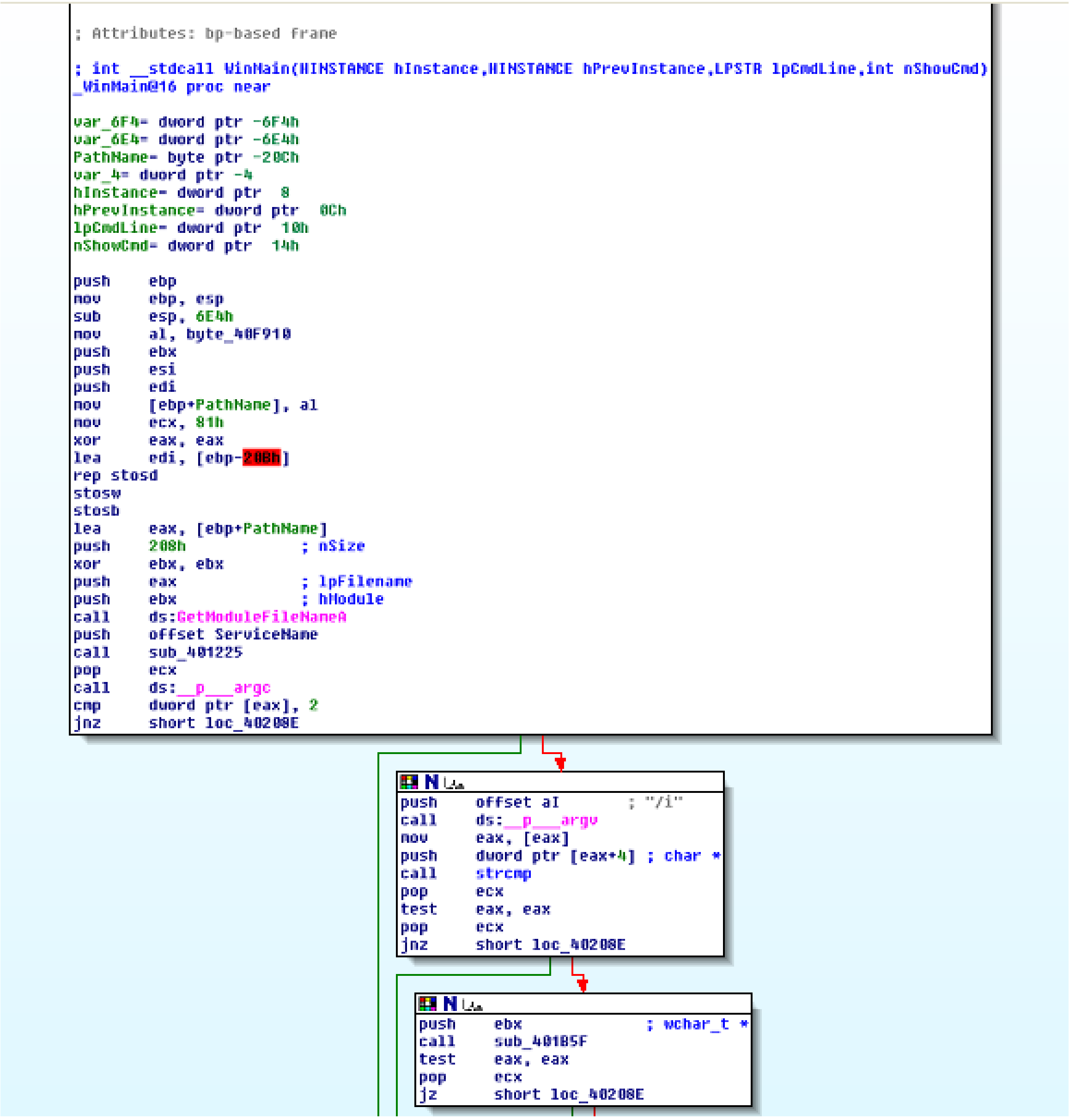


sub_4027DF (IsReadBadPtr)



Our major subroutine is WinMain. The graph below shows our major subroutine.

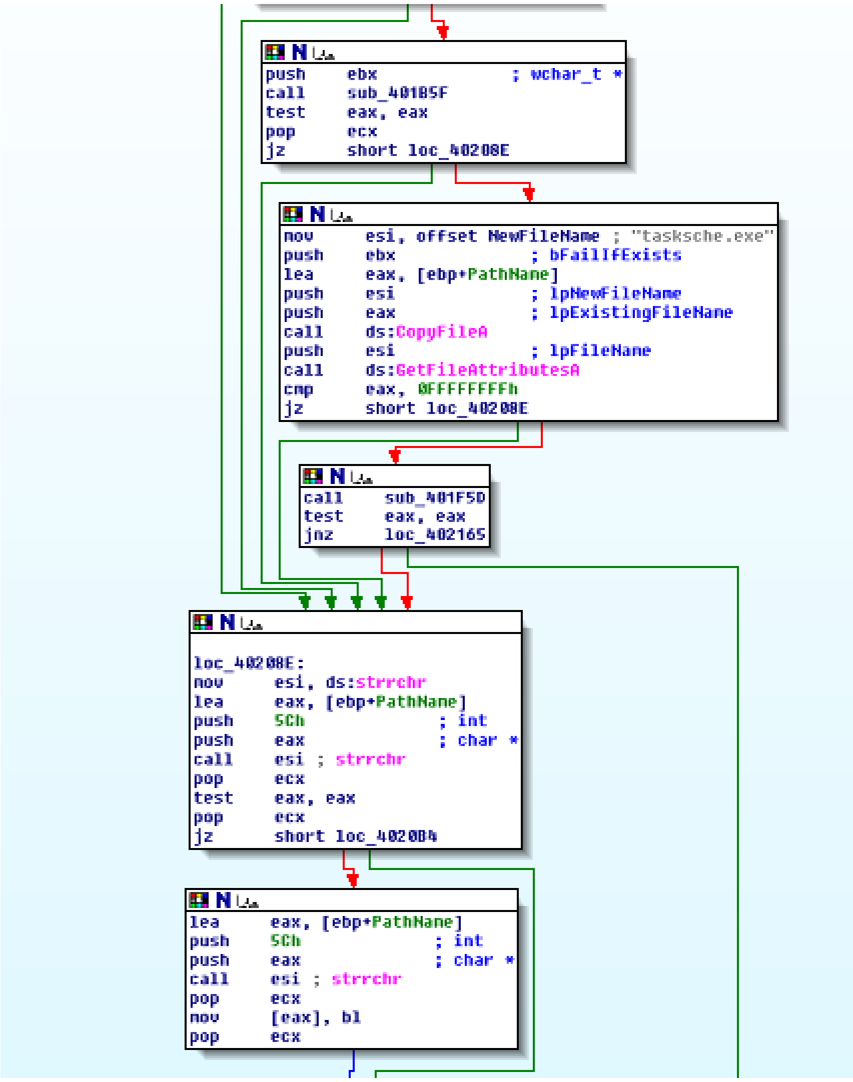
Graph View Flow of major subroutine



```
; Attributes: bp-based frame

; int __stdcall WinMain(HINSTANCE hInstance,HINSTANCE hPrevInstance,LPSTR lpCmdLine,int nShowCmd)
_WinMain@16 proc near

var_6F4= dword ptr -6F4h
var_6E4= dword ptr -6E4h
PathName= byte ptr -20Ch
var_4= dword ptr -4
hInstance= dword ptr  8
hPrevInstance= dword ptr  0Ch
lpCmdLine= dword ptr  10h
nShowCmd= dword ptr  14h

push    ebp
mov     ebp, esp
sub     esp, 6E4h
mov     al, byte_40F910
push    ebx
push    esi
push    edi
mov     [ebp+PathName], al
mov     ecx, 81h
xor     eax, eax
lea     edi, [ebp-20Bh]
rep stosd
stosw
stosb
lea     eax, [ebp+PathName]
push    208h            ; nSize
xor     ebx, ebx
push    eax             ; lpFilename
push    ebx             ; hModule
call    ds:GetModuleFileNameA
push    offset ServiceName
call    sub_401225
pop     ecx
call    ds:__p___argc
cmp     dword ptr [eax], 2
jnz     short loc_40208E
```

```
push    offset aI       ; "/i"
call    ds:__p___argv
mov     eax, [eax]
push    dword ptr [eax+4] ; char *
call    strcmp
pop     ecx
test    eax, eax
pop     ecx
jnz     short loc_40208E
```

```
push    ebx             ; wchar_t *
call    sub_40185F
test    eax, eax
pop     ecx
jz      short loc_40208E
```



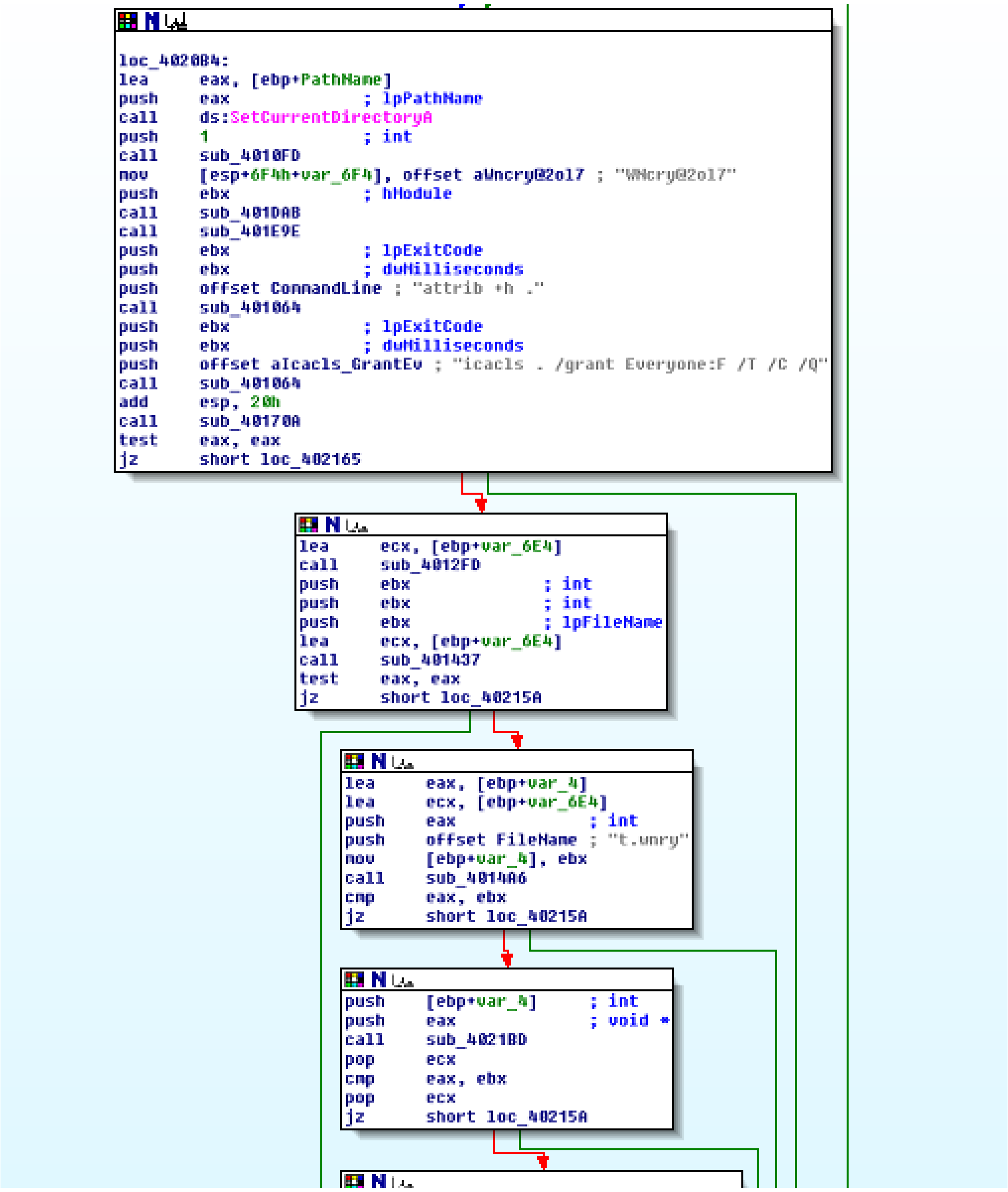
```
push    ebx                 ; wchar_t *
call    sub_40185F
test    eax, eax
pop     ecx
jz      short loc_40208E
```

```
mov     esi, offset NewFileName ; "tasksche.exe"
push    ebx                 ; bFailIfExists
lea     eax, [ebp+PathName]
push    esi                 ; lpNewFileName
push    eax                 ; lpExistingFileName
call    ds:CopyFileA
push    esi                 ; lpFileName
call    ds:GetFileAttributesA
cmp     eax, 0FFFFFFFFh
jz      short loc_40208E
```

```
call    sub_401F5D
test    eax, eax
jnz     loc_402165
```

```

loc_40208E:
mov     esi, ds:strrchr
lea     eax, [ebp+PathName]
push    5Ch                 ; int
push    eax                 ; char *
call    esi ; strrchr
pop     ecx
test    eax, eax
pop     ecx
jz      short loc_4020B4
```

```
lea     eax, [ebp+PathName]
push    5Ch                 ; int
push    eax                 ; char *
call    esi ; strrchr
pop     ecx
mov     [eax], bl
pop     ecx
```

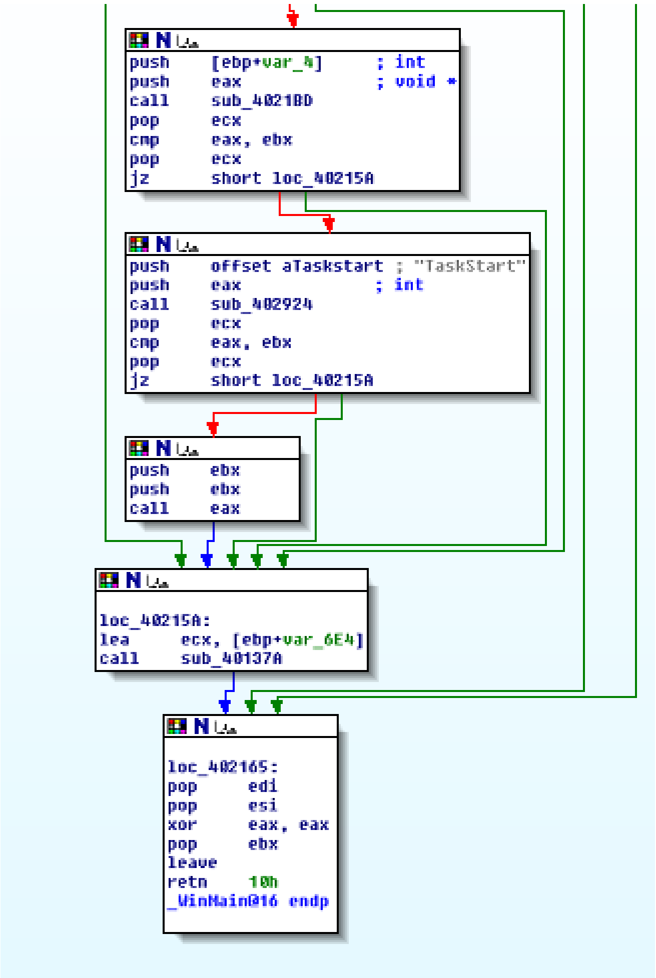


```
loc_402084:
lea     eax, [ebp+PathName]
push    eax                 ; lpPathName
call    ds:SetCurrentDirectoryA
push    1                   ; int
call    sub_4010FD
mov     [esp+6F4h+var_6F4], offset aWncry@2ol7 ; "WNcry@2ol7"
push    ebx                 ; hModule
call    sub_401DAB
call    sub_401E9E
push    ebx                 ; lpExitCode
push    ebx                 ; dwMilliseconds
push    offset CommandLine ; "attrib +h ."
call    sub_401064
push    ebx                 ; lpExitCode
push    ebx                 ; dwMilliseconds
push    offset aIcacls_GrantEv ; "icacls . /grant Everyone:F /T /C /Q"
call    sub_401064
add     esp, 20h
call    sub_40170A
test    eax, eax
jz      short loc_402165
```

```
lea     ecx, [ebp+var_6E4]
call    sub_4012FD
push    ebx                 ; int
push    ebx                 ; int
push    ebx                 ; lpFileName
lea     ecx, [ebp+var_6E4]
call    sub_401437
test    eax, eax
jz      short loc_40215A
```

```
lea     eax, [ebp+var_4]
lea     ecx, [ebp+var_6E4]
push    eax                 ; int
push    offset FileName ; "t.wnry"
mov     [ebp+var_4], ebx
call    sub_4014A6
cmp     eax, ebx
jz      short loc_40215A
```

```
push    [ebp+var_4]         ; int
push    eax                 ; void *
call    sub_4021BD
pop     ecx
cmp     eax, ebx
pop     ecx
jz      short loc_40215A
```



```
push    [ebp+var_4]     ; int
push    eax             ; void *
call    sub_4021BD
pop     ecx
cmp     eax, ebx
pop     ecx
jz      short loc_40215A
```

```
push    offset aTaskstart ; "TaskStart"
push    eax             ; int
call    sub_402924
pop     ecx
cmp     eax, ebx
pop     ecx
jz      short loc_40215A
```

```
push    ebx
push    ebx
call    eax
```

```
loc_40215A:
lea     ecx, [ebp+var_6E4]
call    sub_40137A
```

```
loc_402165:
pop     edi
pop     esi
xor     eax, eax
pop     ebx
leave
retn    10h
_WinMain@16 endp
```

Purpose of major subroutine:

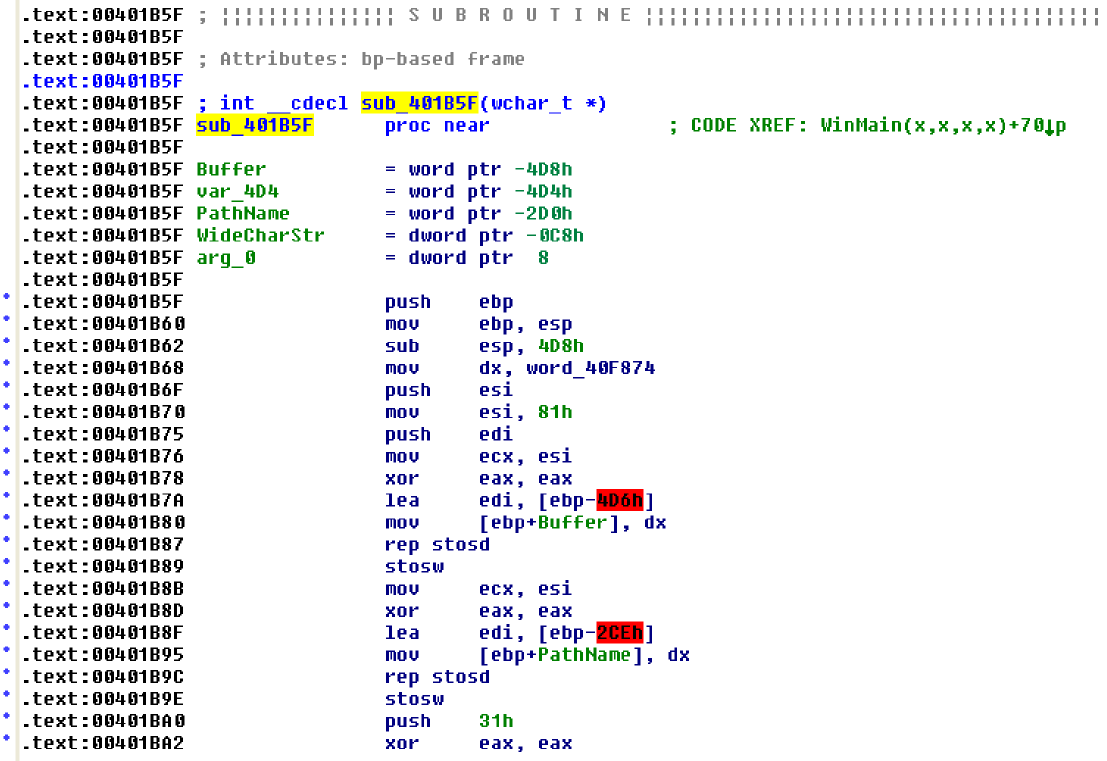
For the major subroutine, it calls GetModuleFileNameA in the beginning to install the main executable of the malware, in this case tasksche.exe. It will check if the argument /i exists which means to install before proceeding to download itself in the victim computer as shown using CopyFileA and GetFileAttributesA. This executable contains a resource zip file "XIA" which will be downloaded using sub_401DAB and contains files that malware will unzip using the password "WNcry@2017" as shown above.

It will proceed to call sub_401064, which will use the API LoadLibraryA to load a malicious DLL and proceed to get the various files on the victim computer.

After some processing, the malware will decrypt t.wnry, which is an encrypted ransomware DLL and export it to TaskStart to begin encrypting the files using a variety of different encryption algorithms such as RSA and AES.

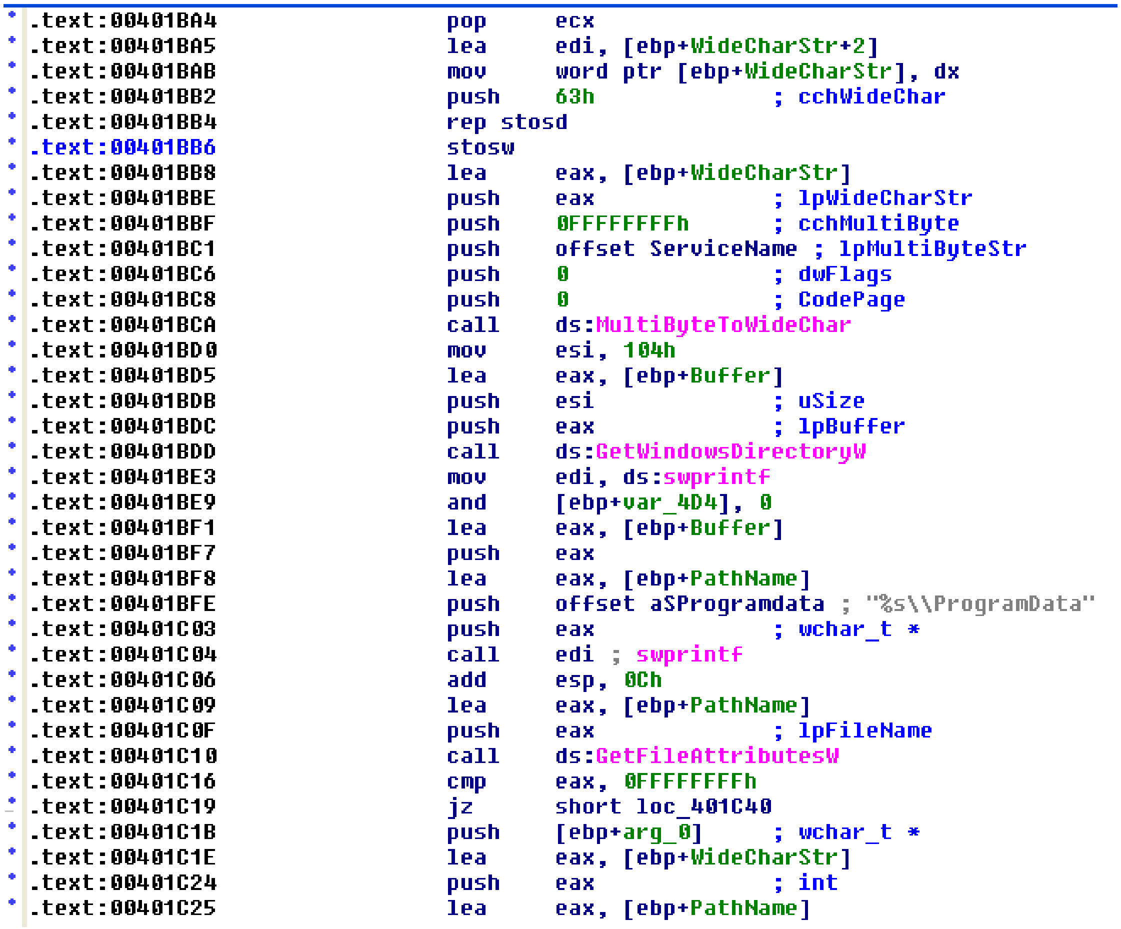
2.2 Description of Subroutines

2.2.1 Subroutine 1: sub_401B5F



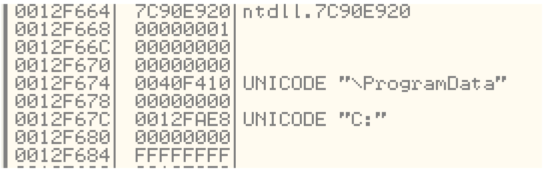
```
.text:00401B5F ; ¦¦¦¦¦¦¦¦¦¦¦¦¦¦¦ S U B R O U T I N E ¦¦¦¦¦¦¦¦¦¦¦¦¦¦¦¦¦¦¦¦¦¦¦¦¦¦¦¦¦¦¦¦¦¦¦¦¦¦¦
.text:00401B5F
.text:00401B5F ; Attributes: bp-based frame
.text:00401B5F
.text:00401B5F ; int __cdecl sub_401B5F(wchar_t *)
.text:00401B5F sub_401B5F      proc near               ; CODE XREF: WinMain(x,x,x,x)+70↓p
.text:00401B5F
.text:00401B5F Buffer          = word ptr -4D8h
.text:00401B5F var_4D4         = word ptr -4D4h
.text:00401B5F PathName        = word ptr -2D0h
.text:00401B5F WideCharStr     = dword ptr -0C8h
.text:00401B5F arg_0           = dword ptr  8
.text:00401B5F
.text:00401B5F                 push    ebp
.text:00401B60                 mov     ebp, esp
.text:00401B62                 sub     esp, 4D8h
.text:00401B68                 mov     dx, word_40F874
.text:00401B6F                 push    esi
.text:00401B70                 mov     esi, 81h
.text:00401B75                 push    edi
.text:00401B76                 mov     ecx, esi
.text:00401B78                 xor     eax, eax
.text:00401B7A                 lea     edi, [ebp-4D6h]
.text:00401B80                 mov     [ebp+Buffer], dx
.text:00401B87                 rep stosd
.text:00401B89                 stosw
.text:00401B8B                 mov     ecx, esi
.text:00401B8D                 xor     eax, eax
.text:00401B8F                 lea     edi, [ebp-2CEh]
.text:00401B95                 mov     [ebp+PathName], dx
.text:00401B9C                 rep stosd
.text:00401B9E                 stosw
.text:00401BA0                 push    31h
.text:00401BA2                 xor     eax, eax
```

The buffer size and the file path are provided as arguments as shown above. The code first allocates a value to the filename and the buffer.

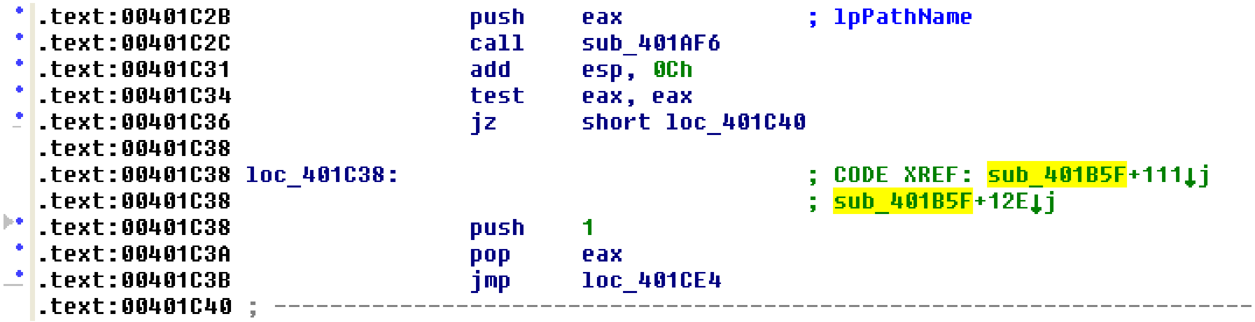


```
.text:00401BA4                 pop     ecx
.text:00401BA5                 lea     edi, [ebp+WideCharStr+2]
.text:00401BAB                 mov     word ptr [ebp+WideCharStr], dx
.text:00401BB2                 push    63h             ; cchWideChar
.text:00401BB4                 rep stosd
.text:00401BB6                 stosw
.text:00401BB8                 lea     eax, [ebp+WideCharStr]
.text:00401BBE                 push    eax             ; lpWideCharStr
.text:00401BBF                 push    0FFFFFFFFh      ; cchMultiByte
.text:00401BC1                 push    offset ServiceName ; lpMultiByteStr
.text:00401BC6                 push    0               ; dwFlags
.text:00401BC8                 push    0               ; CodePage
.text:00401BCA                 call    ds:MultiByteToWideChar
.text:00401BD0                 mov     esi, 104h
.text:00401BD5                 lea     eax, [ebp+Buffer]
.text:00401BDB                 push    esi             ; uSize
.text:00401BDC                 push    eax             ; lpBuffer
.text:00401BDD                 call    ds:GetWindowsDirectoryW
.text:00401BE3                 mov     edi, ds:swprintf
.text:00401BE9                 and     [ebp+var_4D4], 0
.text:00401BF1                 lea     eax, [ebp+Buffer]
.text:00401BF7                 push    eax
.text:00401BF8                 lea     eax, [ebp+PathName]
.text:00401BFE                 push    offset aSProgramdata ; "%s\\ProgramData"
.text:00401C03                 push    eax             ; wchar_t *
.text:00401C04                 call    edi ; swprintf
.text:00401C06                 add     esp, 0Ch
.text:00401C09                 lea     eax, [ebp+PathName]
.text:00401C0F                 push    eax             ; lpFileName
.text:00401C10                 call    ds:GetFileAttributesW
.text:00401C16                 cmp     eax, 0FFFFFFFFh
.text:00401C19                 jz      short loc_401C40
.text:00401C1B                 push    [ebp+arg_0]     ; wchar_t *
.text:00401C1E                 lea     eax, [ebp+WideCharStr]
.text:00401C24                 push    eax             ; int
.text:00401C25                 lea     eax, [ebp+PathName]
```

If ecx is zero in the main program, this subroutine gets executed. This subroutine gets information about the system that it is running on through the use of imported APIs such as GetWindowsDirectory and GetFileAttributesW. This allows the malware to find a location to download and install its malicious payload for future use. The path used is the drive letter denoted by the placeholder %s which is a string and then the path of \\ProgramData which is a legitimate Windows folder used to store program information for later execution.



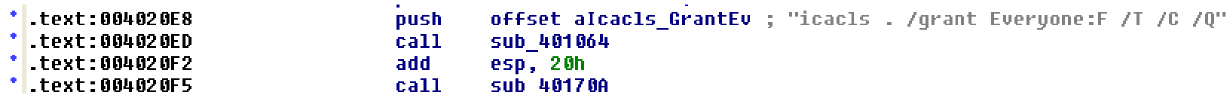
```
0012F664  7C90E920 ntdll.7C90E920
0012F668  00000001
0012F66C  00000000
0012F670  00000000
0012F674  0040F410 UNICODE "\ProgramData"
0012F678  00000000
0012F67C  0012FAE8 UNICODE "C:"
0012F680  00000000
0012F684  FFFFFFFF
```



```
.text:00401C2B                 push    eax             ; lpPathName
.text:00401C2C                 call    sub_401AF6
.text:00401C31                 add     esp, 0Ch
.text:00401C34                 test    eax, eax
.text:00401C36                 jz      short loc_401C40
.text:00401C38
.text:00401C38 loc_401C38:                             ; CODE XREF: sub_401B5F+111↓j
.text:00401C38                                         ; sub_401B5F+12E↓j
.text:00401C38                 push    1
.text:00401C3A                 pop     eax
.text:00401C3B                 jmp     loc_401CE4
.text:00401C40 ; ---------------------------------------------------------------------------
```

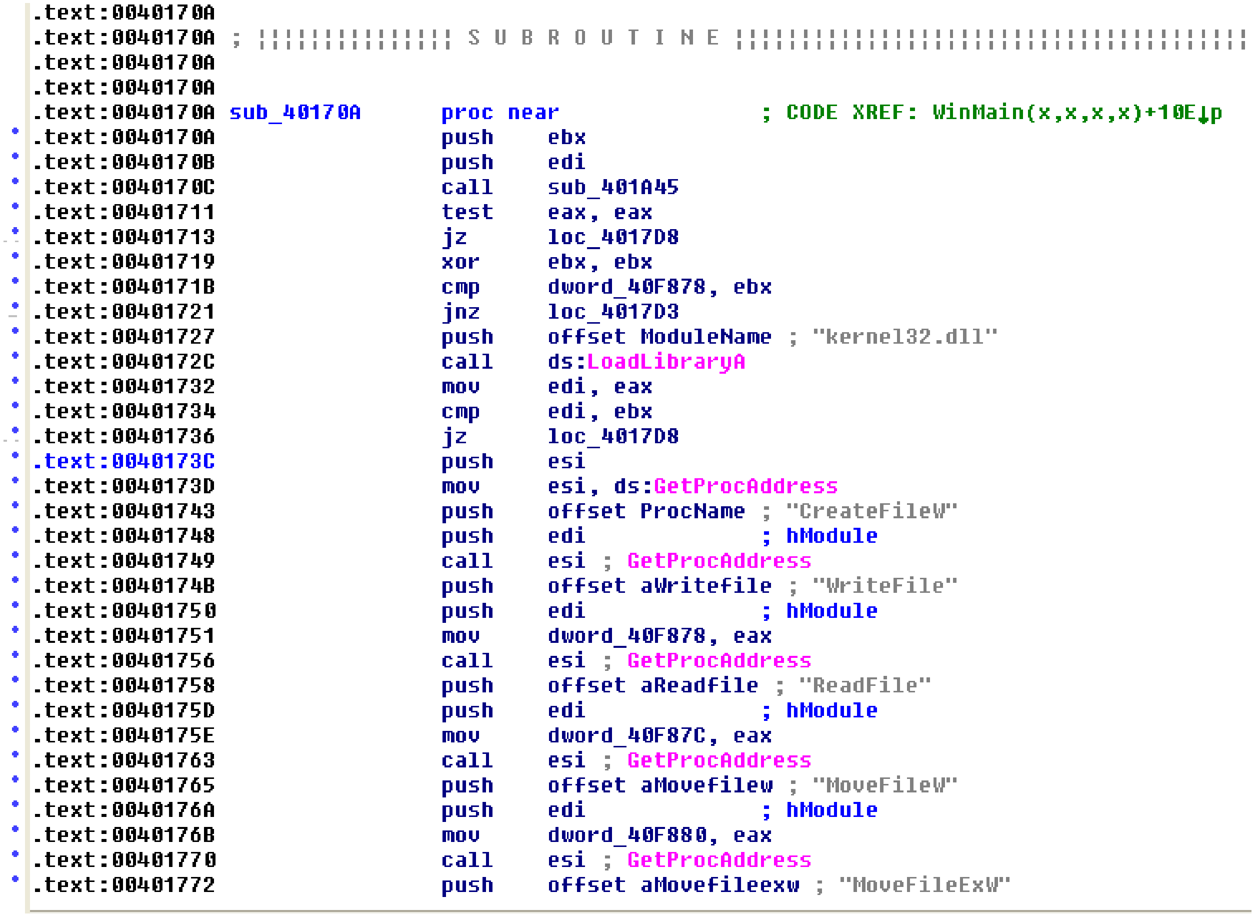
If the pointer to pathname is supplied, the program jumps to another subroutine (that is not shown). Otherwise, the program gets ended when it jumps to loc_401CE4.

2.2.2 Subroutine 2: sub_40170A



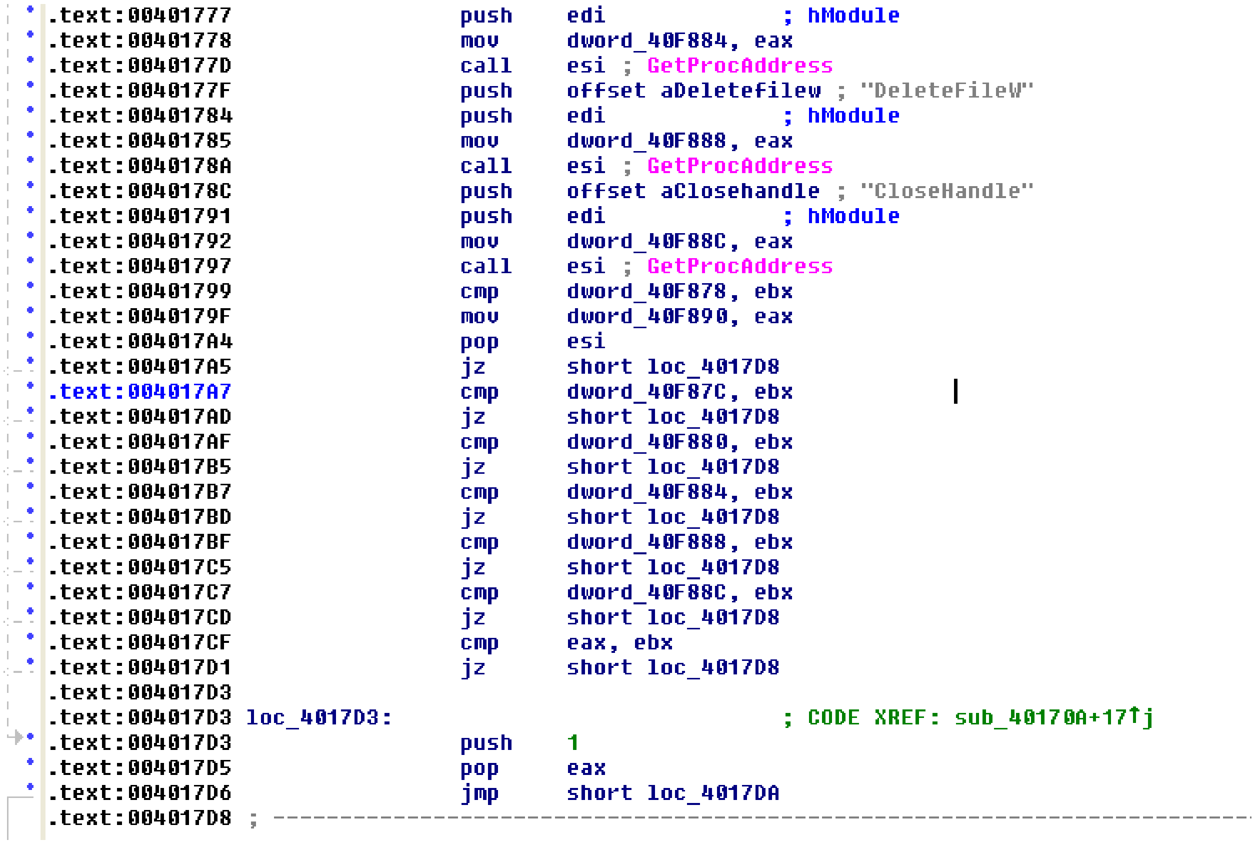
```
.text:004020E8                 push    offset aIcacls_GrantEv ; "icacls . /grant Everyone:F /T /C /Q"
.text:004020ED                 call    sub_401064
.text:004020F2                 add     esp, 20h
.text:004020F5                 call    sub_40170A
```

Before this subroutine is called the malware permission is granted to all users (including the malware) to the newly created folder. This subroutine is then called after sub_401064 is executed.



```
.text:0040170A
.text:0040170A ; ¦¦¦¦¦¦¦¦¦¦¦¦¦¦¦ S U B R O U T I N E ¦¦¦¦¦¦¦¦¦¦¦¦¦¦¦¦¦¦¦¦¦¦¦¦¦¦¦¦¦¦¦¦¦¦¦¦¦¦¦¦
.text:0040170A
.text:0040170A
.text:0040170A sub_40170A      proc near               ; CODE XREF: WinMain(x,x,x,x)+10E↓p
.text:0040170A                 push    ebx
.text:0040170B                 push    edi
.text:0040170C                 call    sub_401A45
.text:00401711                 test    eax, eax
.text:00401713                 jz      loc_4017D8
.text:00401719                 xor     ebx, ebx
.text:0040171B                 cmp     dword_40F878, ebx
.text:00401721                 jnz     loc_4017D3
.text:00401727                 push    offset ModuleName ; "kernel32.dll"
.text:0040172C                 call    ds:LoadLibraryA
.text:00401732                 mov     edi, eax
.text:00401734                 cmp     edi, ebx
.text:00401736                 jz      loc_4017D8
.text:0040173C                 push    esi
.text:0040173D                 mov     esi, ds:GetProcAddress
.text:00401743                 push    offset ProcName ; "CreateFileW"
.text:00401748                 push    edi             ; hModule
.text:00401749                 call    esi ; GetProcAddress
.text:0040174B                 push    offset aWritefile ; "WriteFile"
.text:00401750                 push    edi             ; hModule
.text:00401751                 mov     dword_40F878, eax
.text:00401756                 call    esi ; GetProcAddress
.text:00401758                 push    offset aReadfile ; "ReadFile"
.text:0040175D                 push    edi             ; hModule
.text:0040175E                 mov     dword_40F87C, eax
.text:00401763                 call    esi ; GetProcAddress
.text:00401765                 push    offset aMovefilew ; "MoveFileW"
.text:0040176A                 push    edi             ; hModule
.text:0040176B                 mov     dword_40F880, eax
.text:00401770                 call    esi ; GetProcAddress
.text:00401772                 push    offset aMovefileexw ; "MoveFileExW"
```

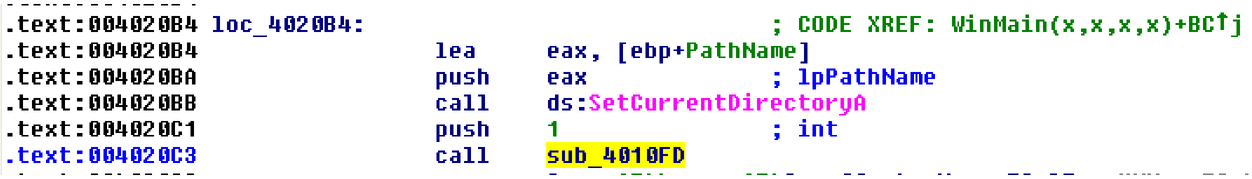
This subroutine takes no arguments as input. It calls sub_401A45 and then if the value stored in eax is zero, it jumps to the location loc_4017D8. If not it performs an xor for ebx with itself and compares ebx with a preset dword value. If dword is smaller than ebx it will jump to loc_4017D3. Otherwise it will set the Module Name as kernel32.dll and call the LoadLibrary API which calls the malicious aAdvapi32.dll. Thereafter, there are many GetProcAddress API calls which call the various API from the DLLs.



```
.text:00401777                 push    edi             ; hModule
.text:00401778                 mov     dword_40F884, eax
.text:0040177D                 call    esi ; GetProcAddress
.text:0040177F                 push    offset aDeletefilew ; "DeleteFileW"
.text:00401784                 push    edi             ; hModule
.text:00401785                 mov     dword_40F888, eax
.text:0040178A                 call    esi ; GetProcAddress
.text:0040178C                 push    offset aClosehandle ; "CloseHandle"
.text:00401791                 push    edi             ; hModule
.text:00401792                 mov     dword_40F88C, eax
.text:00401797                 call    esi ; GetProcAddress
.text:00401799                 cmp     dword_40F878, ebx
.text:0040179F                 mov     dword_40F890, eax
.text:004017A4                 pop     esi
.text:004017A5                 jz      short loc_4017D8
.text:004017A7                 cmp     dword_40F87C, ebx
.text:004017AD                 jz      short loc_4017D8
.text:004017AF                 cmp     dword_40F880, ebx
.text:004017B5                 jz      short loc_4017D8
.text:004017B7                 cmp     dword_40F884, ebx
.text:004017BD                 jz      short loc_4017D8
.text:004017BF                 cmp     dword_40F888, ebx
.text:004017C5                 jz      short loc_4017D8
.text:004017C7                 cmp     dword_40F88C, ebx
.text:004017CD                 jz      short loc_4017D8
.text:004017CF                 cmp     eax, ebx
.text:004017D1                 jz      short loc_4017D8
.text:004017D3
.text:004017D3 loc_4017D3:                             ; CODE XREF: sub_40170A+17↑j
.text:004017D3                 push    1
.text:004017D5                 pop     eax
.text:004017D6                 jmp     short loc_4017DA
.text:004017D8 ; ---------------------------------------------------------------------------
```

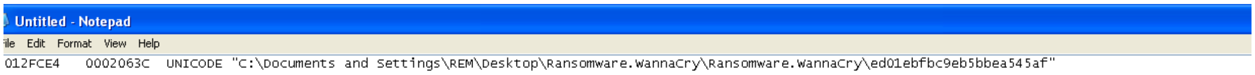
Thereafter, there are various comparisons between ebx and different dword values, which determine the next code block that the malware executes. If these are not fulfilled the program moves on to loc_4017DA.

2.2.3 Subroutine 3: sub_4010FD

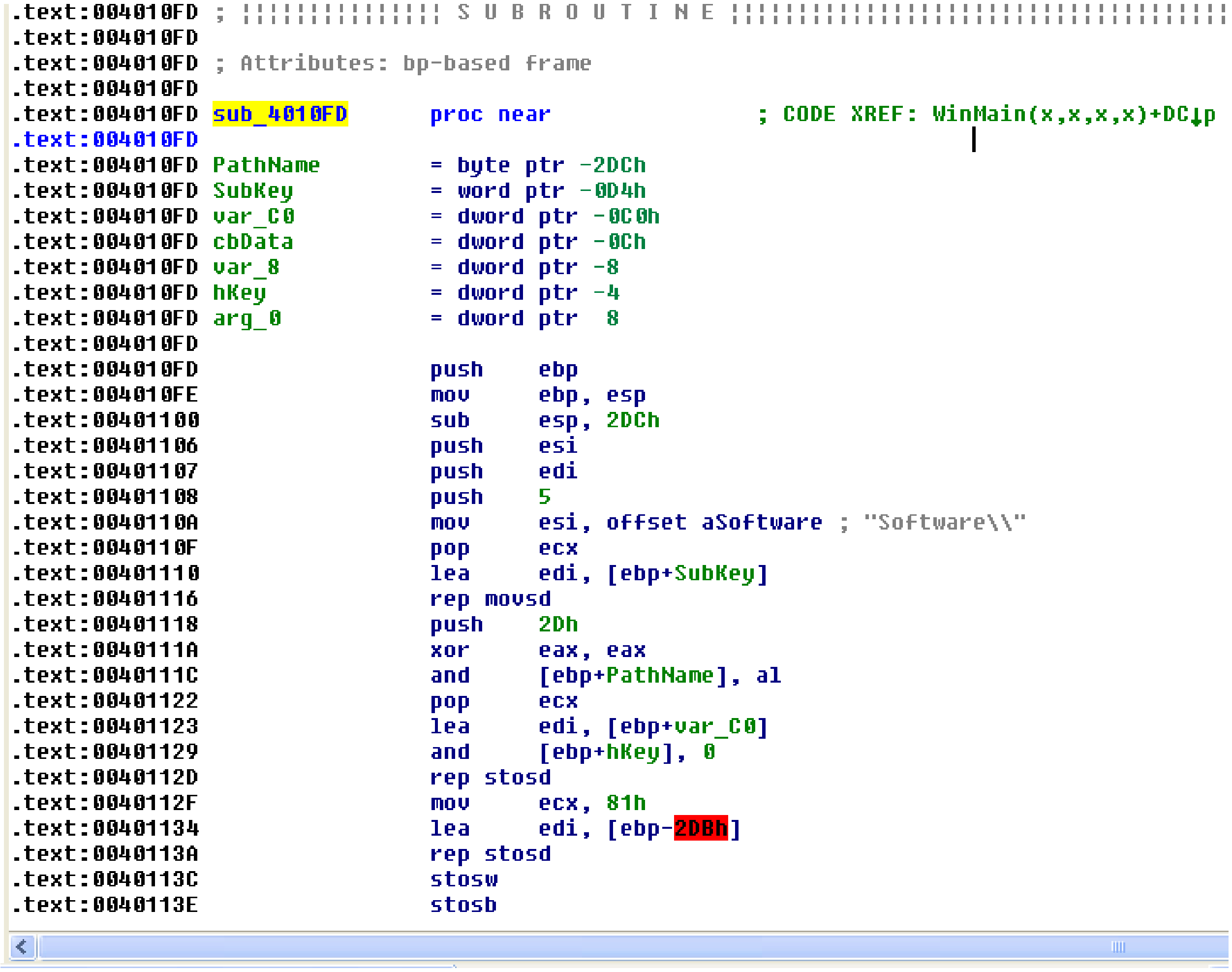


```
.text:004020B4 loc_4020B4:                             ; CODE XREF: WinMain(x,x,x,x)+BC↑j
.text:004020B4                 lea     eax, [ebp+PathName]
.text:004020BA                 push    eax             ; lpPathName
.text:004020BB                 call    ds:SetCurrentDirectoryA
.text:004020C1                 push    1               ; int
.text:004020C3                 call    sub_4010FD
```

From the main program, the pathname as well as the pointer to the pathname is supplied to the SetCurrentDirectoryA API to enable the malware to be executed in the directory that it is currently in. Thereafter the subroutine is called with the current working directory passed as the PathName.

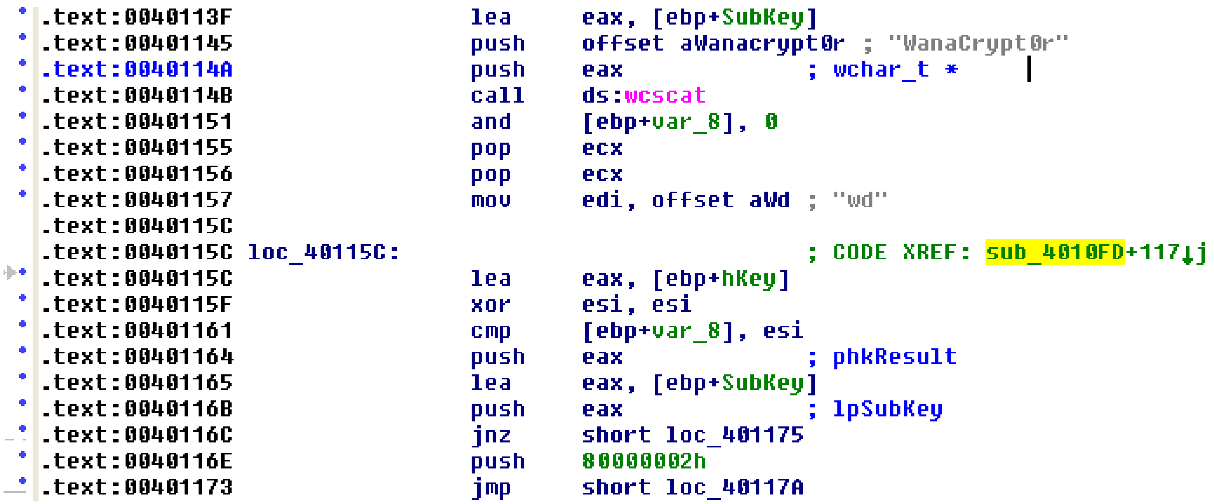


```
012FCE4    0002063C   UNICODE "C:\Documents and Settings\REM\Desktop\Ransomware.WannaCry\Ransomware.WannaCry\ed01ebfbc9eb5bbea545af"
```



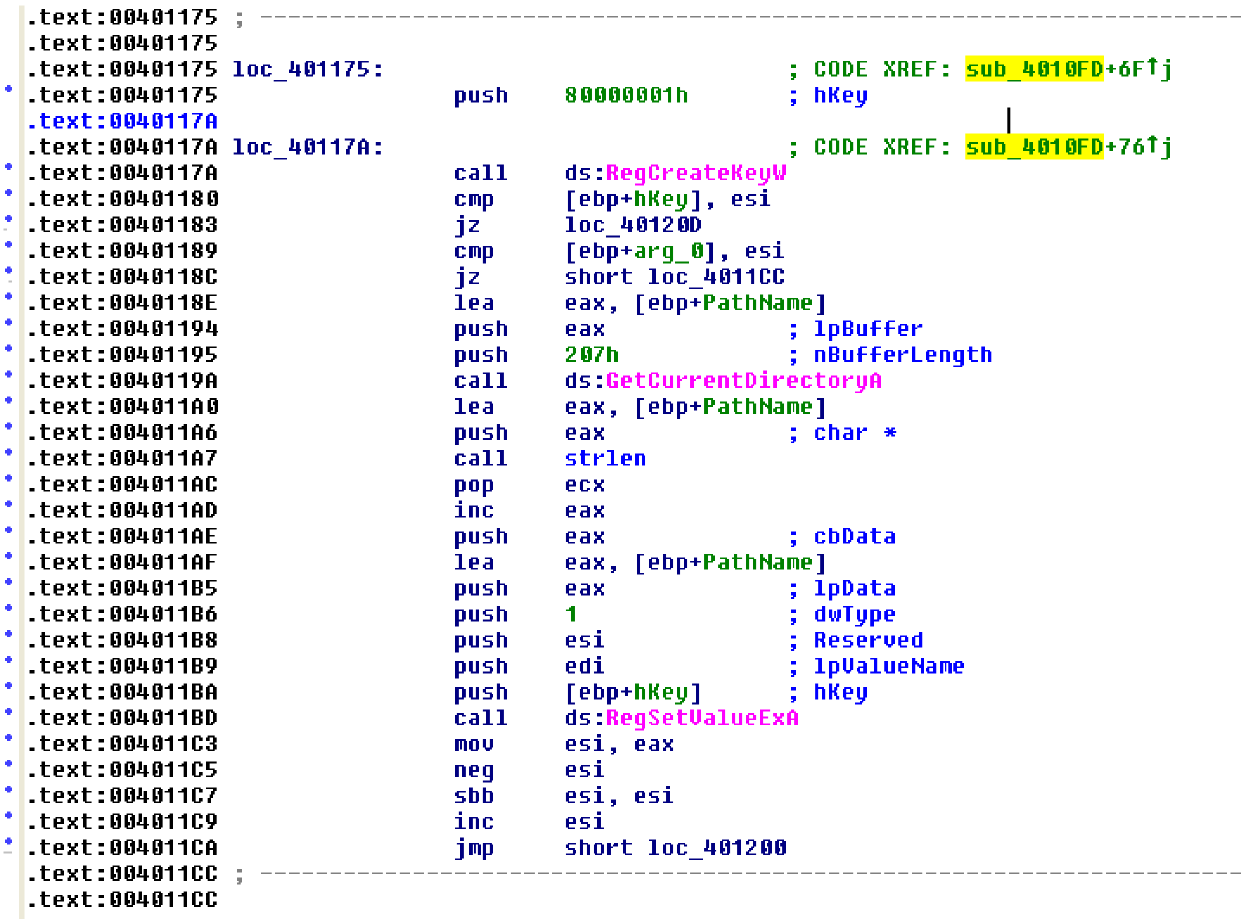
```
.text:004010FD ; ||||||||||||||| S U B R O U T I N E |||||||||||||||||||||||||||||||||||||||
.text:004010FD
.text:004010FD ; Attributes: bp-based frame
.text:004010FD
.text:004010FD sub_4010FD      proc near               ; CODE XREF: WinMain(x,x,x,x)+DC↓p
.text:004010FD
.text:004010FD PathName        = byte ptr -2DCh
.text:004010FD SubKey          = word ptr -0D4h
.text:004010FD var_C0          = dword ptr -0C0h
.text:004010FD cbData          = dword ptr -0Ch
.text:004010FD var_8           = dword ptr -8
.text:004010FD hKey            = dword ptr -4
.text:004010FD arg_0           = dword ptr  8
.text:004010FD
.text:004010FD                 push    ebp
.text:004010FE                 mov     ebp, esp
.text:00401100                 sub     esp, 2DCh
.text:00401106                 push    esi
.text:00401107                 push    edi
.text:00401108                 push    5
.text:0040110A                 mov     esi, offset aSoftware ; "Software\\"
.text:0040110F                 pop     ecx
.text:00401110                 lea     edi, [ebp+SubKey]
.text:00401116                 rep movsd
.text:00401118                 push    2Dh
.text:0040111A                 xor     eax, eax
.text:0040111C                 and     [ebp+PathName], al
.text:00401122                 pop     ecx
.text:00401123                 lea     edi, [ebp+var_C0]
.text:00401129                 and     [ebp+hKey], 0
.text:0040112D                 rep stosd
.text:0040112F                 mov     ecx, 81h
.text:00401134                 lea     edi, [ebp-2DBh]
.text:0040113A                 rep stosd
.text:0040113C                 stosw
.text:0040113E                 stosb
```

The malware starts by creating memory to make a function. It then copies the malware from the Software\\ folder to the current working directory (in this case Desktop) It then calls 2 subroutines to get the hkey argument which is essential later on when it calls the RegCreateKeyW API to create persistence of the malware.

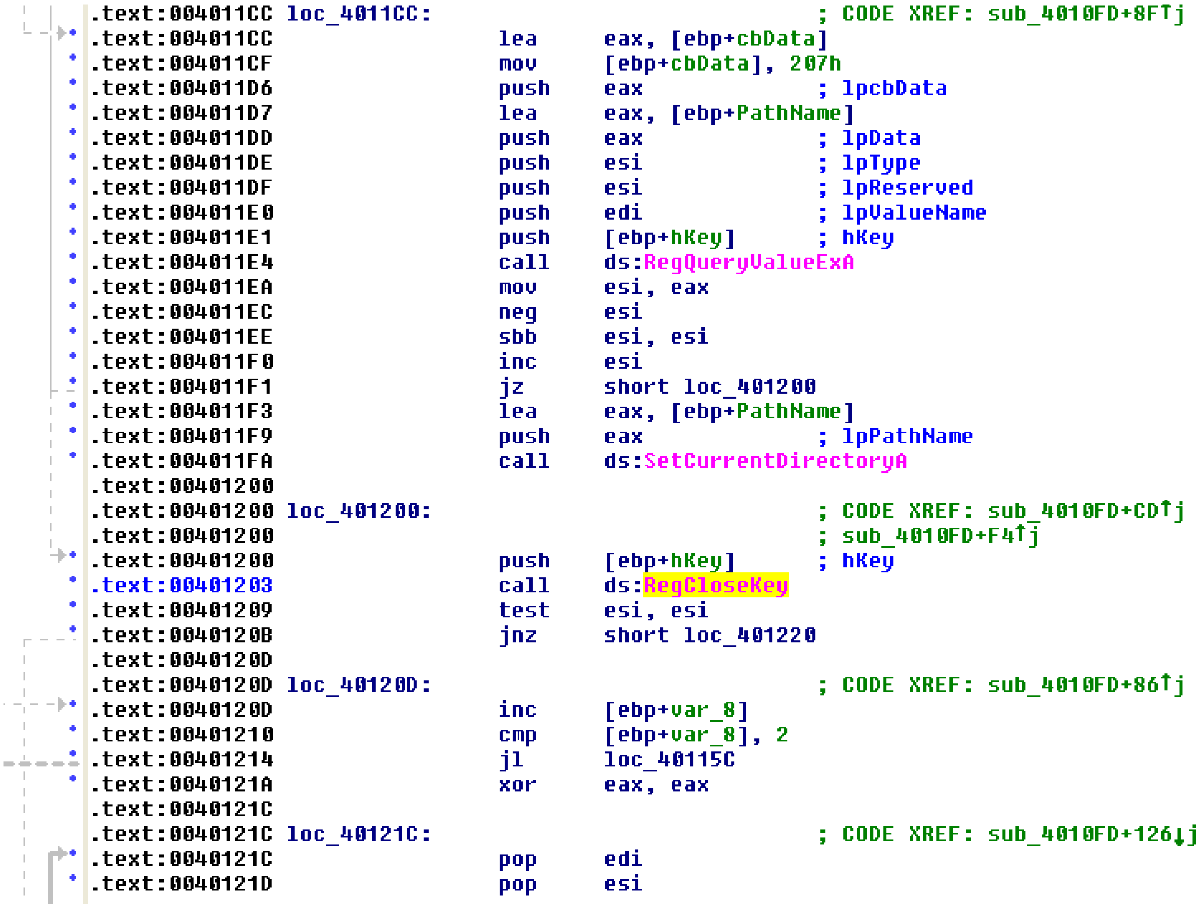


```
.text:0040113F                 lea     eax, [ebp+SubKey]
.text:00401145                 push    offset aWanacrypt0r ; "WanaCrypt0r"
.text:0040114A                 push    eax             ; wchar_t *
.text:0040114B                 call    ds:wcscat
.text:00401151                 and     [ebp+var_8], 0
.text:00401155                 pop     ecx
.text:00401156                 pop     ecx
.text:00401157                 mov     edi, offset aWd ; "wd"
.text:0040115C
.text:0040115C loc_40115C:                             ; CODE XREF: sub_4010FD+117↓j
.text:0040115C                 lea     eax, [ebp+hKey]
.text:0040115F                 xor     esi, esi
.text:00401161                 cmp     [ebp+var_8], esi
.text:00401164                 push    eax             ; phkResult
.text:00401165                 lea     eax, [ebp+SubKey]
.text:0040116B                 push    eax             ; lpSubKey
.text:0040116C                 jnz     short loc_401175
.text:0040116E                 push    80000002h
.text:00401173                 jmp     short loc_40117A
```

Thereafter, the malware calls the GetCurrentDirectoryA API for it to be able set the registry value later on in the following API RegSetValueExA and thereafter it performs some arithmetic before jumping on to loc_401200 as shown below.

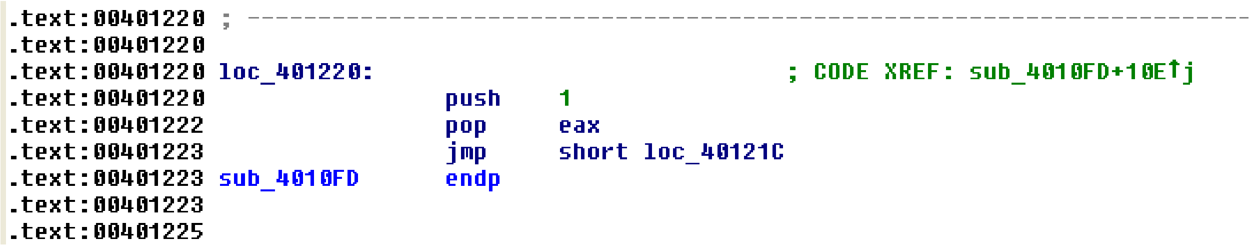


```
.text:00401175 ; ---------------------------------------------------------------------------
.text:00401175
.text:00401175 loc_401175:                             ; CODE XREF: sub_4010FD+6F↑j
.text:00401175                 push    80000001h       ; hKey
.text:0040117A
.text:0040117A loc_40117A:                             ; CODE XREF: sub_4010FD+76↑j
.text:0040117A                 call    ds:RegCreateKeyW
.text:00401180                 cmp     [ebp+hKey], esi
.text:00401183                 jz      loc_40120D
.text:00401189                 cmp     [ebp+arg_0], esi
.text:0040118C                 jz      short loc_4011CC
.text:0040118E                 lea     eax, [ebp+PathName]
.text:00401194                 push    eax             ; lpBuffer
.text:00401195                 push    207h            ; nBufferLength
.text:0040119A                 call    ds:GetCurrentDirectoryA
.text:004011A0                 lea     eax, [ebp+PathName]
.text:004011A6                 push    eax             ; char *
.text:004011A7                 call    strlen
.text:004011AC                 pop     ecx
.text:004011AD                 inc     eax
.text:004011AE                 push    eax             ; cbData
.text:004011AF                 lea     eax, [ebp+PathName]
.text:004011B5                 push    eax             ; lpData
.text:004011B6                 push    1               ; dwType
.text:004011B8                 push    esi             ; Reserved
.text:004011B9                 push    edi             ; lpValueName
.text:004011BA                 push    [ebp+hKey]      ; hKey
.text:004011BD                 call    ds:RegSetValueExA
.text:004011C3                 mov     esi, eax
.text:004011C5                 neg     esi
.text:004011C7                 sbb     esi, esi
.text:004011C9                 inc     esi
.text:004011CA                 jmp     short loc_401200
.text:004011CC ; ---------------------------------------------------------------------------
.text:004011CC
```



```
.text:004011CC loc_4011CC:                            ; CODE XREF: sub_4010FD+8F↑j
.text:004011CC                 lea     eax, [ebp+cbData]
.text:004011CF                 mov     [ebp+cbData], 207h
.text:004011D6                 push    eax             ; lpcbData
.text:004011D7                 lea     eax, [ebp+PathName]
.text:004011DD                 push    eax             ; lpData
.text:004011DE                 push    esi             ; lpType
.text:004011DF                 push    esi             ; lpReserved
.text:004011E0                 push    edi             ; lpValueName
.text:004011E1                 push    [ebp+hKey]      ; hKey
.text:004011E4                 call    ds:RegQueryValueExA
.text:004011EA                 mov     esi, eax
.text:004011EC                 neg     esi
.text:004011EE                 sbb     esi, esi
.text:004011F0                 inc     esi
.text:004011F1                 jz      short loc_401200
.text:004011F3                 lea     eax, [ebp+PathName]
.text:004011F9                 push    eax             ; lpPathName
.text:004011FA                 call    ds:SetCurrentDirectoryA
.text:00401200
.text:00401200 loc_401200:                            ; CODE XREF: sub_4010FD+CD↑j
.text:00401200                                         ; sub_4010FD+F4↑j
.text:00401200                 push    [ebp+hKey]      ; hKey
.text:00401203                 call    ds:RegCloseKey
.text:00401209                 test    esi, esi
.text:0040120B                 jnz     short loc_401220
.text:0040120D
.text:0040120D loc_40120D:                            ; CODE XREF: sub_4010FD+86↑j
.text:0040120D                 inc     [ebp+var_8]
.text:00401210                 cmp     [ebp+var_8], 2
.text:00401214                 jl      loc_40115C
.text:0040121A                 xor     eax, eax
.text:0040121C
.text:0040121C loc_40121C:                            ; CODE XREF: sub_4010FD+126↓j
.text:0040121C                 pop     edi
.text:0040121D                 pop     esi
```

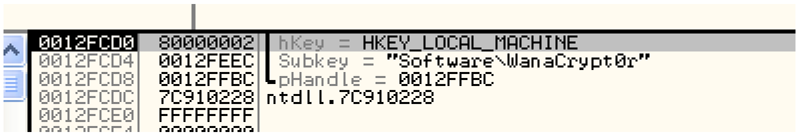
Thereafter the malware calls the RegQueryValueExA API to retrieve the type and the value of the newly created registry key to perhaps confirm that it exists before performing RegCloseKey, which saves the registry key and thus makes the malware persistent even on reboot. Thereafter if esi is not set to 0, the malware will perform a jump to loc_401220 and the subroutine will eventually end after all the registers and pointers have been popped off the stack (after loc_40121C, it will proceed to loc_401223), where endp is called and the program is hence terminated.



```
.text:00401220 ; ---------------------------------------------------------------------------
.text:00401220
.text:00401220 loc_401220:                            ; CODE XREF: sub_4010FD+10E↑j
.text:00401220                 push    1
.text:00401222                 pop     eax
.text:00401223                 jmp     short loc_40121C
.text:00401223 sub_4010FD      endp
.text:00401223
.text:00401225
```

sub_4010FD values in OllyDBG

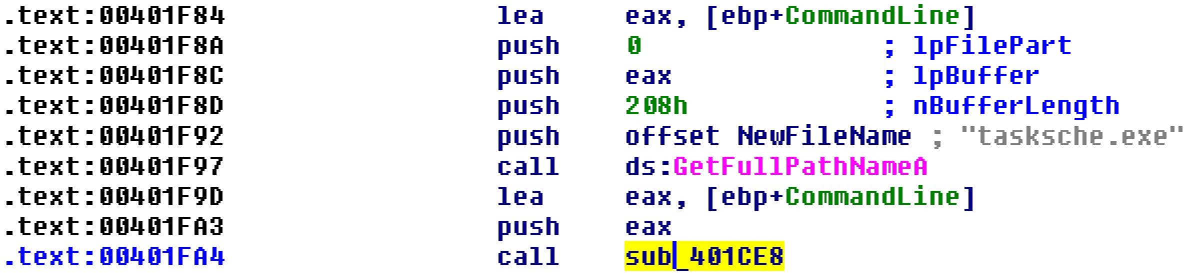
Values for RegCreateKeyW



```
0012FCD0  80000002  hKey = HKEY_LOCAL_MACHINE
0012FCD4  0012FEEC  Subkey = "Software\WanaCrypt0r"
0012FCD8  0012FFBC  pHandle = 0012FFBC
0012FCDC  7C910228 ntdll.7C910228
0012FCE0  FFFFFFFF
0012FCE4  00000000
```

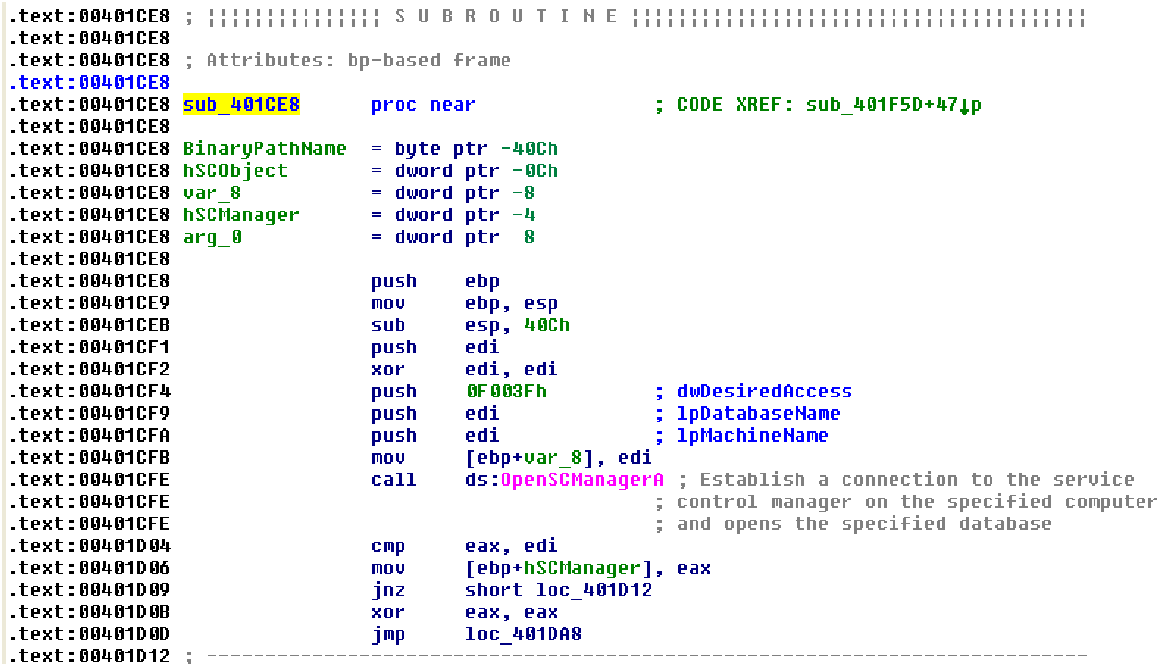
2.2.4 Subroutine 4: sub_401CE8

The subroutine is called from sub_401F5D.



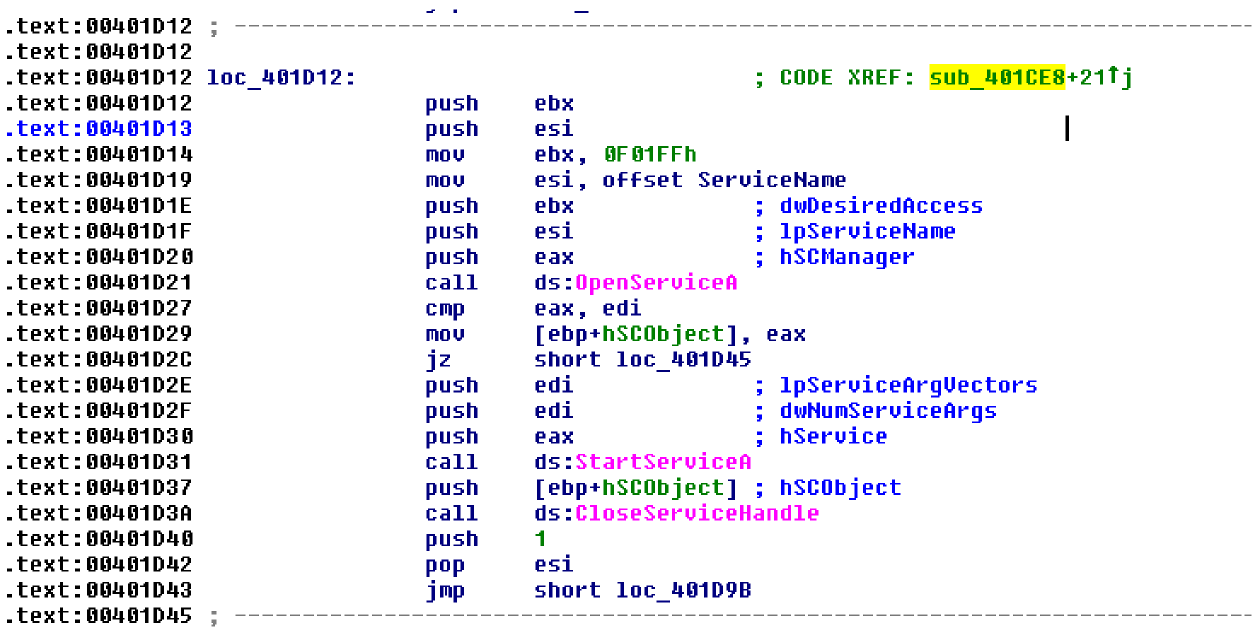
```
.text:00401F84                 lea     eax, [ebp+CommandLine]
.text:00401F8A                 push    0               ; lpFilePart
.text:00401F8C                 push    eax             ; lpBuffer
.text:00401F8D                 push    208h            ; nBufferLength
.text:00401F92                 push    offset NewFileName ; "tasksche.exe"
.text:00401F97                 call    ds:GetFullPathNameA
.text:00401F9D                 lea     eax, [ebp+CommandLine]
.text:00401FA3                 push    eax
.text:00401FA4                 call    sub_401CE8
```

The file is renamed to taskche.exe before the GetFilePathNameA API is called and thereafter, the subroutine is called.



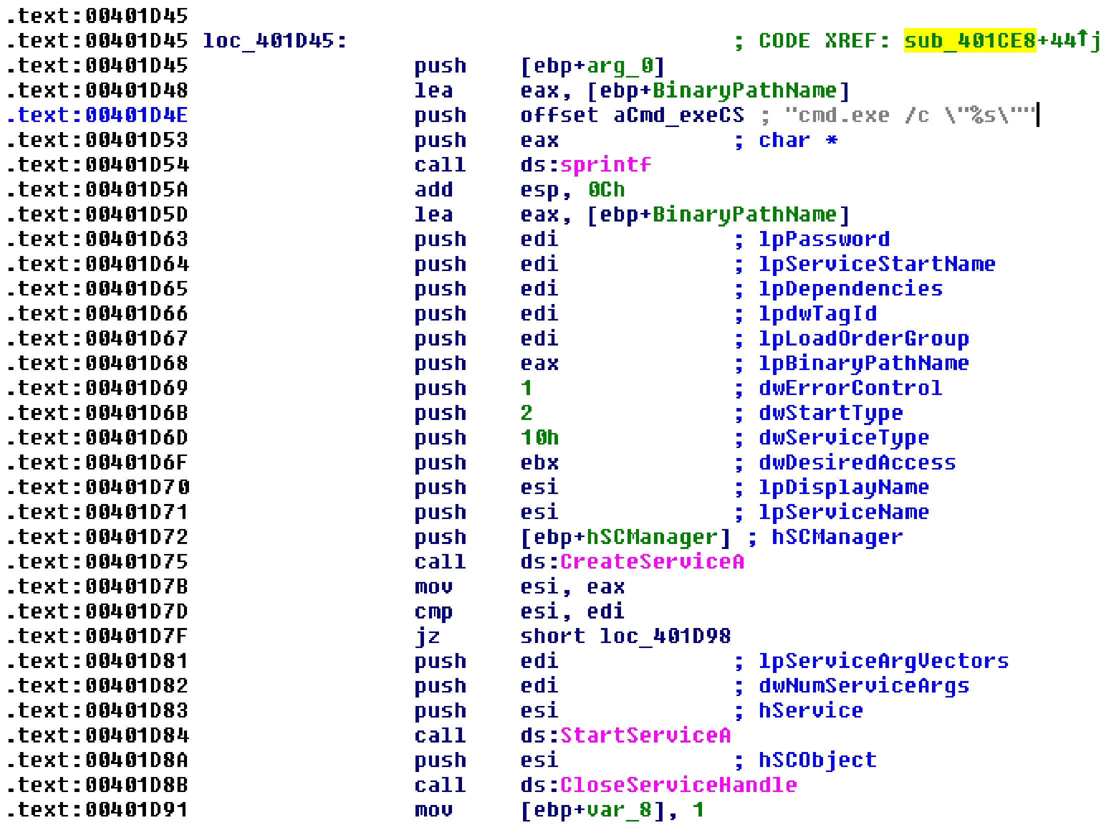
```
.text:00401CE8 ; ¦¦¦¦¦¦¦¦¦¦¦¦¦¦¦ S U B R O U T I N E ¦¦¦¦¦¦¦¦¦¦¦¦¦¦¦¦¦¦¦¦¦¦¦¦¦¦¦¦¦¦¦¦¦¦¦¦¦¦
.text:00401CE8
.text:00401CE8 ; Attributes: bp-based frame
.text:00401CE8
.text:00401CE8 sub_401CE8      proc near               ; CODE XREF: sub_401F5D+47↓p
.text:00401CE8
.text:00401CE8 BinaryPathName  = byte ptr -40Ch
.text:00401CE8 hSCObject       = dword ptr -0Ch
.text:00401CE8 var_8           = dword ptr -8
.text:00401CE8 hSCManager      = dword ptr -4
.text:00401CE8 arg_0           = dword ptr  8
.text:00401CE8
.text:00401CE8                 push    ebp
.text:00401CE9                 mov     ebp, esp
.text:00401CEB                 sub     esp, 40Ch
.text:00401CF1                 push    edi
.text:00401CF2                 xor     edi, edi
.text:00401CF4                 push    0F003Fh         ; dwDesiredAccess
.text:00401CF9                 push    edi             ; lpDatabaseName
.text:00401CFA                 push    edi             ; lpMachineName
.text:00401CFB                 mov     [ebp+var_8], edi
.text:00401CFE                 call    ds:OpenSCManagerA ; Establish a connection to the service
.text:00401CFE                                         ; control manager on the specified computer
.text:00401CFE                                         ; and opens the specified database
.text:00401D04                 cmp     eax, edi
.text:00401D06                 mov     [ebp+hSCManager], eax
.text:00401D09                 jnz     short loc_401D12
.text:00401D0B                 xor     eax, eax
.text:00401D0D                 jmp     loc_401DA8
.text:00401D12 ; ---------------------------------------------------------------------------
```

In this subroutine, the malware opens a connection to the service manager to enable functions to be imported and thereafter it performs a few arithmetic operations before moving on to open the service and start/run it.



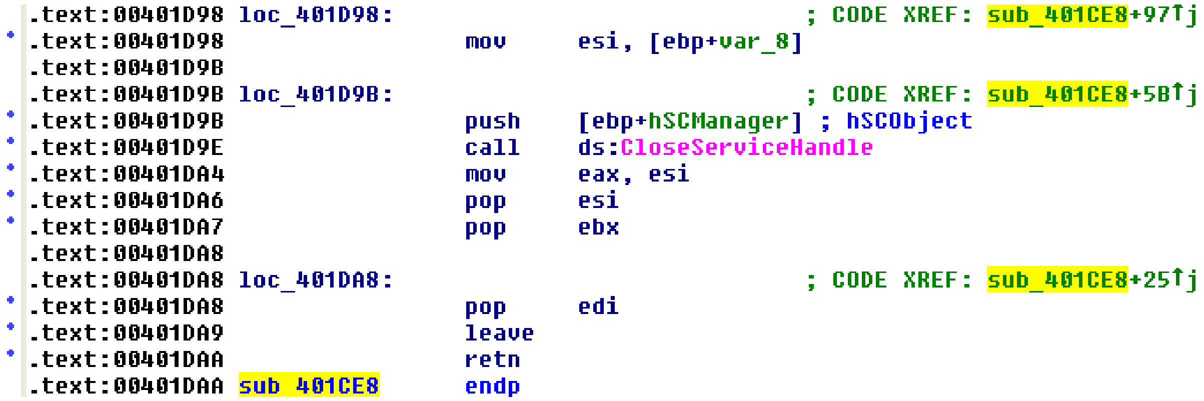
```
.text:00401D12 ; ---------------------------------------------------------------------------
.text:00401D12
.text:00401D12 loc_401D12:                             ; CODE XREF: sub_401CE8+21↑j
.text:00401D12                 push    ebx
.text:00401D13                 push    esi
.text:00401D14                 mov     ebx, 0F01FFh
.text:00401D19                 mov     esi, offset ServiceName
.text:00401D1E                 push    ebx             ; dwDesiredAccess
.text:00401D1F                 push    esi             ; lpServiceName
.text:00401D20                 push    eax             ; hSCManager
.text:00401D21                 call    ds:OpenServiceA
.text:00401D27                 cmp     eax, edi
.text:00401D29                 mov     [ebp+hSCObject], eax
.text:00401D2C                 jz      short loc_401D45
.text:00401D2E                 push    edi             ; lpServiceArgVectors
.text:00401D2F                 push    edi             ; dwNumServiceArgs
.text:00401D30                 push    eax             ; hService
.text:00401D31                 call    ds:StartServiceA
.text:00401D37                 push    [ebp+hSCObject] ; hSCObject
.text:00401D3A                 call    ds:CloseServiceHandle
.text:00401D40                 push    1
.text:00401D42                 pop     esi
.text:00401D43                 jmp     short loc_401D9B
.text:00401D45 ; ---------------------------------------------------------------------------
```

If the result of the comparison between eax and edi sets the zero flag after the OpenService API is called, which suggests that the service has not yet been created it jumps to loc_401D45 as shown below with a /c argument and the path supplied to the command prompt application cmd.exe in order to Call the CreateService Method and then after the service is created is it actually started so this serves as an alternative code path that the malware takes in case the service has yet to been created when the subroutine is first executed.



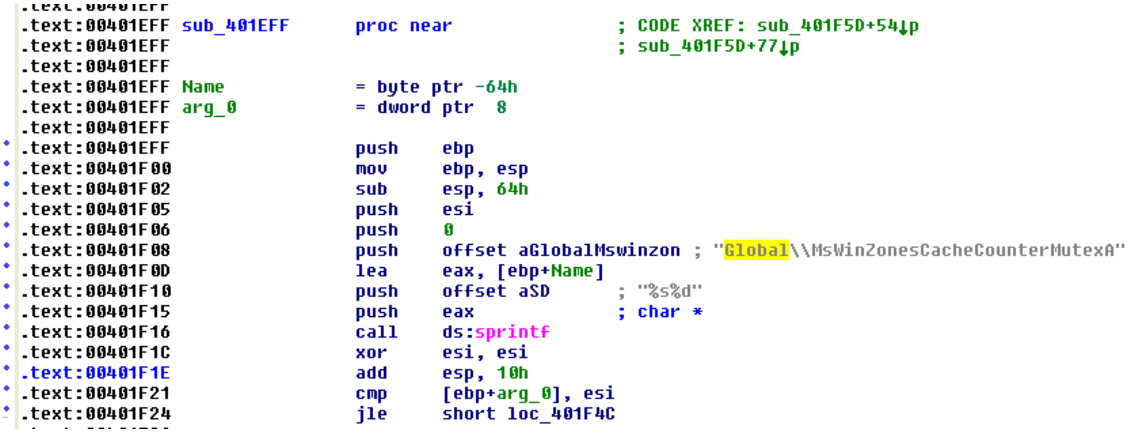
```
.text:00401D45
.text:00401D45 loc_401D45:                             ; CODE XREF: sub_401CE8+44↑j
.text:00401D45                 push    [ebp+arg_0]
.text:00401D48                 lea     eax, [ebp+BinaryPathName]
.text:00401D4E                 push    offset aCmd_exeCS ; "cmd.exe /c \"%s\""
.text:00401D53                 push    eax             ; char *
.text:00401D54                 call    ds:sprintf
.text:00401D5A                 add     esp, 0Ch
.text:00401D5D                 lea     eax, [ebp+BinaryPathName]
.text:00401D63                 push    edi             ; lpPassword
.text:00401D64                 push    edi             ; lpServiceStartName
.text:00401D65                 push    edi             ; lpDependencies
.text:00401D66                 push    edi             ; lpdwTagId
.text:00401D67                 push    edi             ; lpLoadOrderGroup
.text:00401D68                 push    eax             ; lpBinaryPathName
.text:00401D69                 push    1               ; dwErrorControl
.text:00401D6B                 push    2               ; dwStartType
.text:00401D6D                 push    10h             ; dwServiceType
.text:00401D6F                 push    ebx             ; dwDesiredAccess
.text:00401D70                 push    esi             ; lpDisplayName
.text:00401D71                 push    esi             ; lpServiceName
.text:00401D72                 push    [ebp+hSCManager] ; hSCManager
.text:00401D75                 call    ds:CreateServiceA
.text:00401D7B                 mov     esi, eax
.text:00401D7D                 cmp     esi, edi
.text:00401D7F                 jz      short loc_401D98
.text:00401D81                 push    edi             ; lpServiceArgVectors
.text:00401D82                 push    edi             ; dwNumServiceArgs
.text:00401D83                 push    esi             ; hService
.text:00401D84                 call    ds:StartServiceA
.text:00401D8A                 push    esi             ; hSCObject
.text:00401D8B                 call    ds:CloseServiceHandle
.text:00401D91                 mov     [ebp+var_8], 1
```

Thereafter, the CloseServiceHandle API is called to halt the service execution and the program is terminated shortly after popping the registers and pointers from the stack (free the memory).

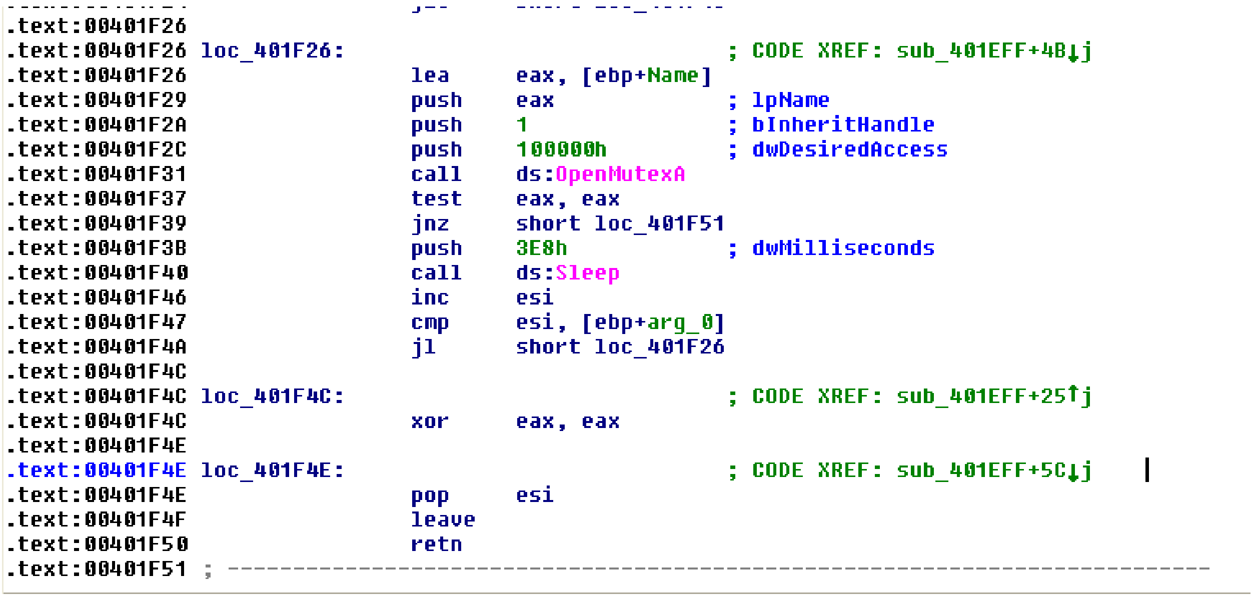


```
.text:00401D98 loc_401D98:                             ; CODE XREF: sub_401CE8+97↑j
.text:00401D98                 mov     esi, [ebp+var_8]
.text:00401D9B
.text:00401D9B loc_401D9B:                             ; CODE XREF: sub_401CE8+5B↑j
.text:00401D9B                 push    [ebp+hSCManager] ; hSCObject
.text:00401D9E                 call    ds:CloseServiceHandle
.text:00401DA4                 mov     eax, esi
.text:00401DA6                 pop     esi
.text:00401DA7                 pop     ebx
.text:00401DA8
.text:00401DA8 loc_401DA8:                             ; CODE XREF: sub_401CE8+25↑j
.text:00401DA8                 pop     edi
.text:00401DA9                 leave
.text:00401DAA                 retn
.text:00401DAA sub_401CE8      endp
```

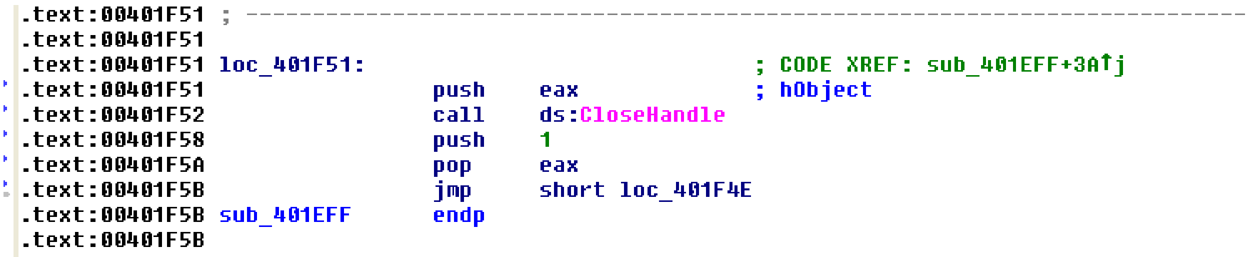
2.2.5 Subroutine 5: sub_401EFF



```
.text:00401EFF sub_401EFF      proc near               ; CODE XREF: sub_401F5D+54↓p
.text:00401EFF                                         ; sub_401F5D+77↓p
.text:00401EFF
.text:00401EFF Name            = byte ptr -64h
.text:00401EFF arg_0           = dword ptr  8
.text:00401EFF
.text:00401EFF                 push    ebp
.text:00401F00                 mov     ebp, esp
.text:00401F02                 sub     esp, 64h
.text:00401F05                 push    esi
.text:00401F06                 push    0
.text:00401F08                 push    offset aGlobalMswinzon ; "Global\\MsWinZonesCacheCounterMutexA"
.text:00401F0D                 lea     eax, [ebp+Name]
.text:00401F10                 push    offset aSD      ; "%s%d"
.text:00401F15                 push    eax             ; char *
.text:00401F16                 call    ds:sprintf
.text:00401F1C                 xor     esi, esi
.text:00401F1E                 add     esp, 10h
.text:00401F21                 cmp     [ebp+arg_0], esi
.text:00401F24                 jle     short loc_401F4C
```



```
.text:00401F26
.text:00401F26 loc_401F26:                             ; CODE XREF: sub_401EFF+4B↓j
.text:00401F26                 lea     eax, [ebp+Name]
.text:00401F29                 push    eax             ; lpName
.text:00401F2A                 push    1               ; bInheritHandle
.text:00401F2C                 push    100000h         ; dwDesiredAccess
.text:00401F31                 call    ds:OpenMutexA
.text:00401F37                 test    eax, eax
.text:00401F39                 jnz     short loc_401F51
.text:00401F3B                 push    3E8h            ; dwMilliseconds
.text:00401F40                 call    ds:Sleep
.text:00401F46                 inc     esi
.text:00401F47                 cmp     esi, [ebp+arg_0]
.text:00401F4A                 jl      short loc_401F26
.text:00401F4C
.text:00401F4C loc_401F4C:                             ; CODE XREF: sub_401EFF+25↑j
.text:00401F4C                 xor     eax, eax
.text:00401F4E
.text:00401F4E loc_401F4E:                             ; CODE XREF: sub_401EFF+5C↓j
.text:00401F4E                 pop     esi
.text:00401F4F                 leave
.text:00401F50                 retn
.text:00401F51 ; ---------------------------------------------------------------------------
```



```
.text:00401F51 ; ---------------------------------------------------------------------------
.text:00401F51
.text:00401F51 loc_401F51:                             ; CODE XREF: sub_401EFF+3A↑j
.text:00401F51                 push    eax             ; hObject
.text:00401F52                 call    ds:CloseHandle
.text:00401F58                 push    1
.text:00401F5A                 pop     eax
.text:00401F5B                 jmp     short loc_401F4E
.text:00401F5B sub_401EFF      endp
.text:00401F5B
```

In this subroutine. the malware using APIs such as OpenMutexA and Sleep. These APIs are malicious as OpenMutexA ensures that only the Malware is running on the machine and Sleep is mainly used by malware to avoid detection by the machine.

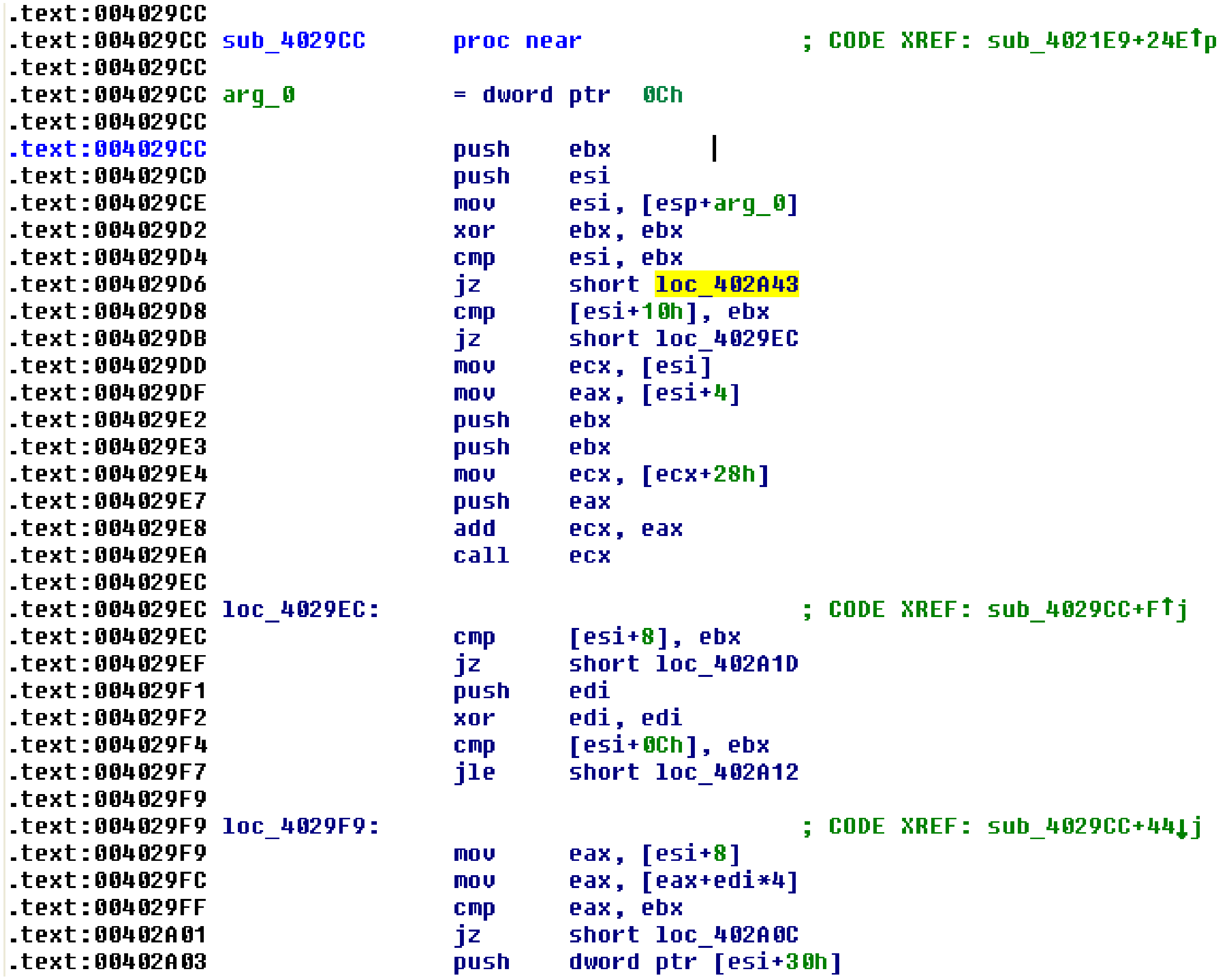
There are arguments being passed into OpenMutexA such as inheritable being set to true and access being 100000. It opens the MuteX object known as

Global\\MsWinZonesCacheCounterMutexA and 100000 being the size of the stack reserve.

3E8h is also pushed into the data segment Sleep, which means the malware will sleep for 1 second. It will also increase the value of esi by 1.

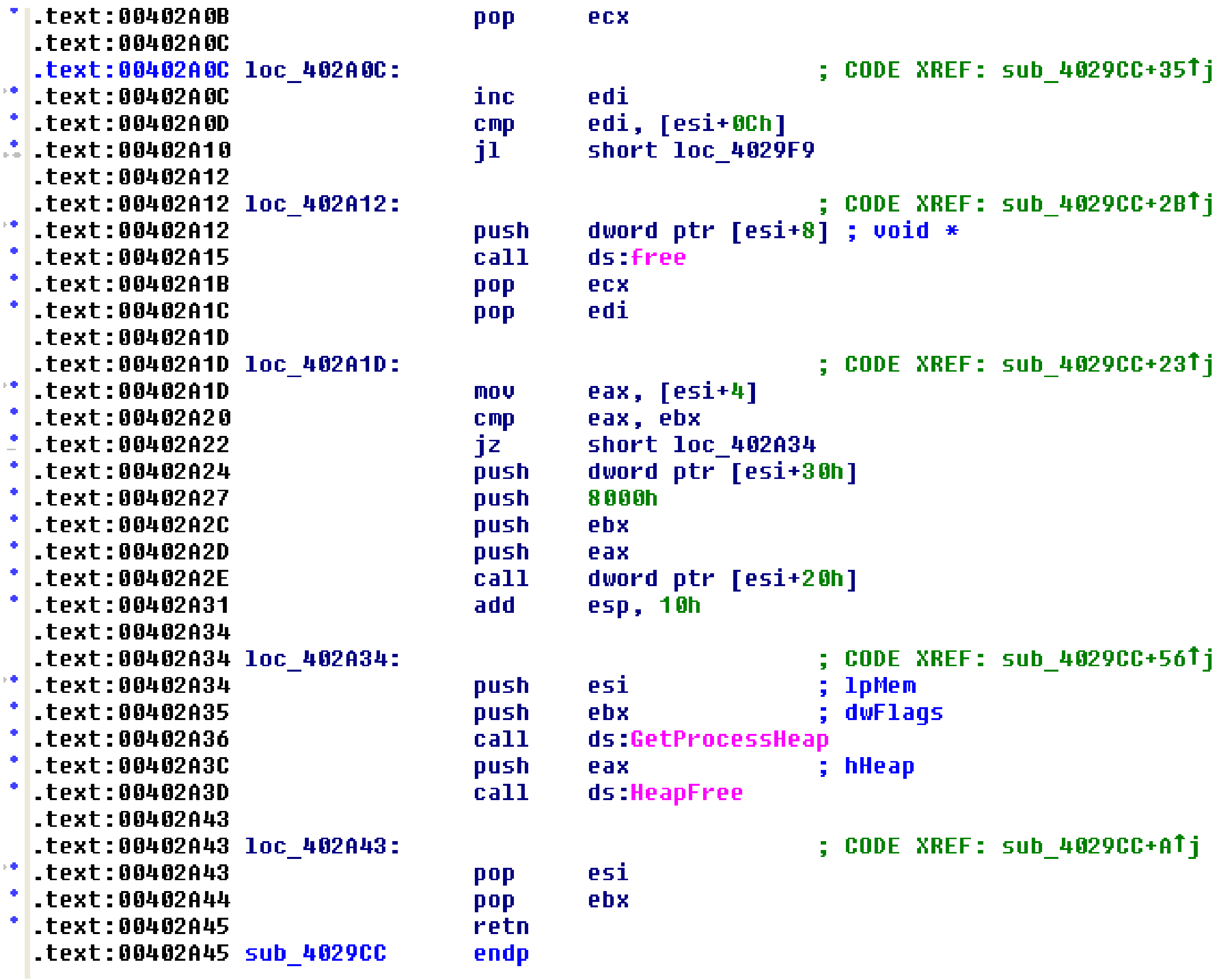
The value stored in [ebp + arg_0] is compared with esi and if esi is larger than the other value there is a formation of a loop structure where the code proceeds back to the beginning of loc 401F26_until [ebp + arg_0] > esi and thereafter eax is XORed with itself. Thereafter the value of esi is returned and the program jumps to loc_401F4E before termination.

2.2.6 Subroutine 6: sub_4029CC



```
.text:004029CC
.text:004029CC sub_4029CC      proc near               ; CODE XREF: sub_4021E9+24E↑p
.text:004029CC
.text:004029CC arg_0           = dword ptr  0Ch
.text:004029CC
.text:004029CC                 push    ebx
.text:004029CD                 push    esi
.text:004029CE                 mov     esi, [esp+arg_0]
.text:004029D2                 xor     ebx, ebx
.text:004029D4                 cmp     esi, ebx
.text:004029D6                 jz      short loc_402A43
.text:004029D8                 cmp     [esi+10h], ebx
.text:004029DB                 jz      short loc_4029EC
.text:004029DD                 mov     ecx, [esi]
.text:004029DF                 mov     eax, [esi+4]
.text:004029E2                 push    ebx
.text:004029E3                 push    ebx
.text:004029E4                 mov     ecx, [ecx+28h]
.text:004029E7                 push    eax
.text:004029E8                 add     ecx, eax
.text:004029EA                 call    ecx
.text:004029EC
.text:004029EC loc_4029EC:                             ; CODE XREF: sub_4029CC+F↑j
.text:004029EC                 cmp     [esi+8], ebx
.text:004029EF                 jz      short loc_402A1D
.text:004029F1                 push    edi
.text:004029F2                 xor     edi, edi
.text:004029F4                 cmp     [esi+0Ch], ebx
.text:004029F7                 jle     short loc_402A12
.text:004029F9
.text:004029F9 loc_4029F9:                             ; CODE XREF: sub_4029CC+44↓j
.text:004029F9                 mov     eax, [esi+8]
.text:004029FC                 mov     eax, [eax+edi*4]
.text:004029FF                 cmp     eax, ebx
.text:00402A01                 jz      short loc_402A0C
.text:00402A03                 push    dword ptr [esi+30h]
```

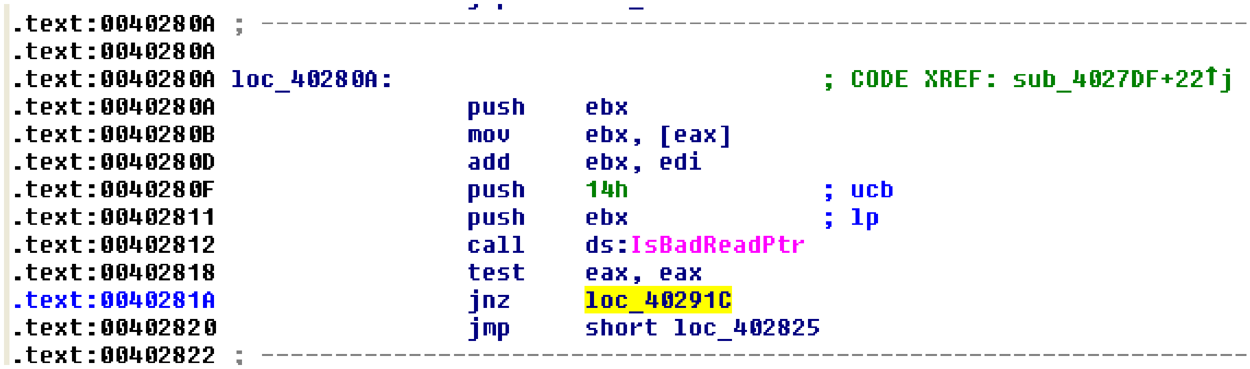
If the comparison of ebx and esi results in the zero flag being set (i.e. if esi > ebx) the program will exit immediately. Otherwise, it will jump to the next location in the subroutine, where another condition is checked where [esi + 0Ch] is greater than or equal to ebx (note that either one of the values stored in the register may be negative). And if this is the case, it will jump directly to loc_402A12, where the allocated memory is freed. If not it will continue executing the code at loc_4029F9.



```
.text:00402A0B                 pop     ecx
.text:00402A0C
.text:00402A0C loc_402A0C:                             ; CODE XREF: sub_4029CC+35↑j
.text:00402A0C                 inc     edi
.text:00402A0D                 cmp     edi, [esi+0Ch]
.text:00402A10                 jl      short loc_4029F9
.text:00402A12
.text:00402A12 loc_402A12:                             ; CODE XREF: sub_4029CC+2B↑j
.text:00402A12                 push    dword ptr [esi+8] ; void *
.text:00402A15                 call    ds:free
.text:00402A1B                 pop     ecx
.text:00402A1C                 pop     edi
.text:00402A1D
.text:00402A1D loc_402A1D:                             ; CODE XREF: sub_4029CC+23↑j
.text:00402A1D                 mov     eax, [esi+4]
.text:00402A20                 cmp     eax, ebx
.text:00402A22                 jz      short loc_402A34
.text:00402A24                 push    dword ptr [esi+30h]
.text:00402A27                 push    8000h
.text:00402A2C                 push    ebx
.text:00402A2D                 push    eax
.text:00402A2E                 call    dword ptr [esi+20h]
.text:00402A31                 add     esp, 10h
.text:00402A34
.text:00402A34 loc_402A34:                             ; CODE XREF: sub_4029CC+56↑j
.text:00402A34                 push    esi             ; lpMem
.text:00402A35                 push    ebx             ; dwFlags
.text:00402A36                 call    ds:GetProcessHeap
.text:00402A3C                 push    eax             ; hHeap
.text:00402A3D                 call    ds:HeapFree
.text:00402A43
.text:00402A43 loc_402A43:                             ; CODE XREF: sub_4029CC+A↑j
.text:00402A43                 pop     esi
.text:00402A44                 pop     ebx
.text:00402A45                 retn
.text:00402A45 sub_4029CC      endp
```

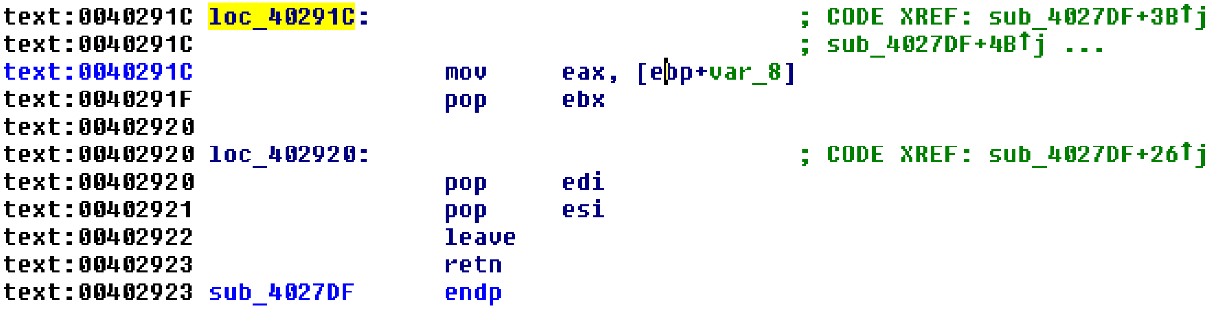
For loc_402A34, which is the most interesting for API calls in this subroutine where GetProcessHeap and HeapFree are called, which basically just calls the heap functions and frees the memory in the heap for use.

2.2.7 Subroutine 7: sub_4027DF



```
.text:0040280A ; ---------------------------------------------------------------------------
.text:0040280A
.text:0040280A loc_40280A:                             ; CODE XREF: sub_4027DF+22↑j
.text:0040280A                 push    ebx
.text:0040280B                 mov     ebx, [eax]
.text:0040280D                 add     ebx, edi
.text:0040280F                 push    14h             ; ucb
.text:00402811                 push    ebx             ; lp
.text:00402812                 call    ds:IsBadReadPtr
.text:00402818                 test    eax, eax
.text:0040281A                 jnz     loc_40291C
.text:00402820                 jmp     short loc_402825
.text:00402822 ; ---------------------------------------------------------------------------
```

This subroutine determines if the program should continue executing or terminate. It calls the API IsBadReadPtr, which determines if it has read access to a range of memory values, which is returned and stored in the data segment. If the value stored in eax is zero then the program will proceed on execution of the code at loc_402825). On the other hand if eax is not zero, the code will be terminated and the malware will stop execution as seen in the code below where endp is called.



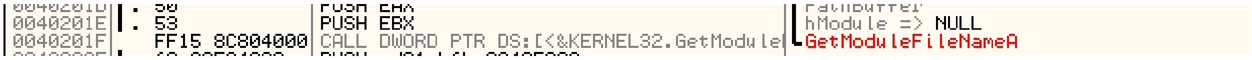
```
text:0040291C loc_40291C:                             ; CODE XREF: sub_4027DF+3B↑j
text:0040291C                                         ; sub_4027DF+4B↑j ...
text:0040291C                 mov     eax, [ebp+var_8]
text:0040291F                 pop     ebx
text:00402920
text:00402920 loc_402920:                             ; CODE XREF: sub_4027DF+26↑j
text:00402920                 pop     edi
text:00402921                 pop     esi
text:00402922                 leave
text:00402923                 retn
text:00402923 sub_4027DF      endp
```

Patching (OllyDBG)

4.1. Main Routine

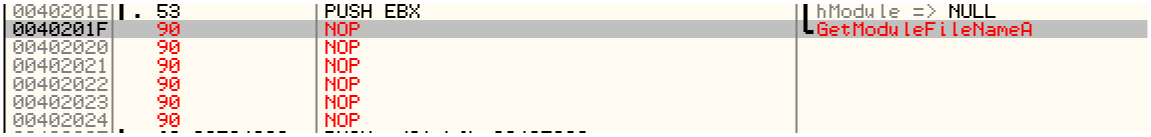
4.1.1. GetModuleFileNameA

Before



```
0040201D . 50            PUSH EAX                                  PathBuffer
0040201E . 53            PUSH EBX                                  hModule => NULL
0040201F   FF15 8C804000 CALL DWORD PTR DS:[<&KERNEL32.GetModule  GetModuleFileNameA
```

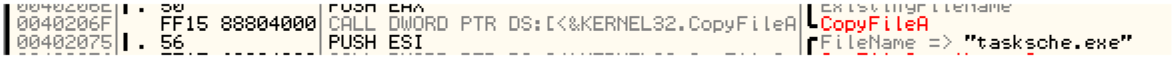
Replaced with NOP



```
0040201E . 53            PUSH EBX                                  hModule => NULL
0040201F   90            NOP                                       GetModuleFileNameA
00402020   90            NOP
00402021   90            NOP
00402022   90            NOP
00402023   90            NOP
00402024   90            NOP
```

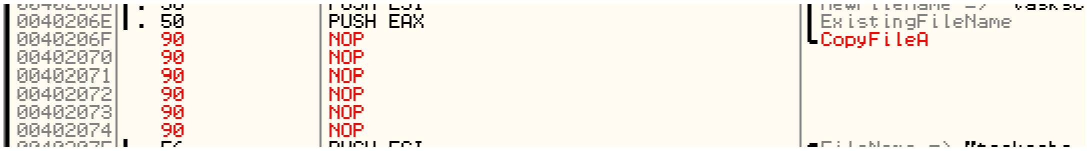
4.1.2. CopyFileA

Before



```
0040206E . 50            PUSH EAX                                  ExistingFileName
0040206F   FF15 88804000 CALL DWORD PTR DS:[<&KERNEL32.CopyFileA  CopyFileA
00402075 . 56            PUSH ESI                                  FileName => "tasksche.exe"
```

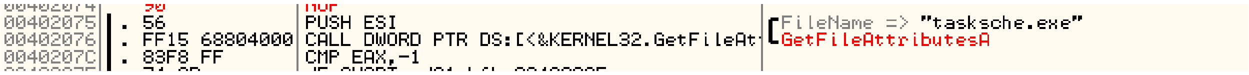
Replaced with NOP



```
0040206D . 56            PUSH ESI                                  NewFileName => "tasksc
0040206E . 50            PUSH EAX                                  ExistingFileName
0040206F   90            NOP                                       CopyFileA
00402070   90            NOP
00402071   90            NOP
00402072   90            NOP
00402073   90            NOP
00402074   90            NOP
```

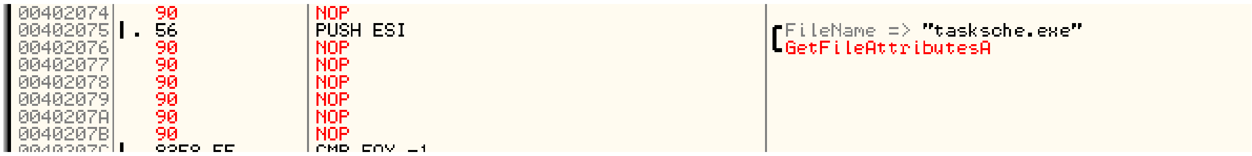
4.1.3. GetFileAttributesA

Before



```
00402074   90            NOP
00402075 . 56            PUSH ESI                                  FileName => "tasksche.exe"
00402076 . FF15 68804000 CALL DWORD PTR DS:[<&KERNEL32.GetFileAt  GetFileAttributesA
0040207C . 83F8 FF       CMP EAX,-1
```

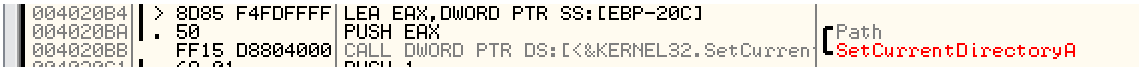
Replaced with NOP



```
00402074    90            NOP
00402075 |. 56            PUSH ESI                          [FileName => "tasksche.exe"
00402076    90            NOP                               [GetFileAttributesA
00402077    90            NOP
00402078    90            NOP
00402079    90            NOP
0040207A    90            NOP
0040207B    90            NOP
0040207C |. 83F8 FF       CMP EAX,-1
```

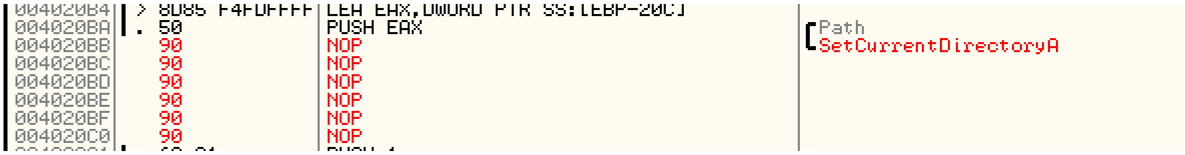
4.1.4. SetCurrentDirectoryA

Before



```
004020B4 |> 8D85 F4FDFFFF LEA EAX,DWORD PTR SS:[EBP-20C]
004020BA |. 50            PUSH EAX                          [Path
004020BB |. FF15 D8804000 CALL DWORD PTR DS:[<&KERNEL32.SetCurren [SetCurrentDirectoryA
004020C1 |. 6A 01         PUSH 1
```

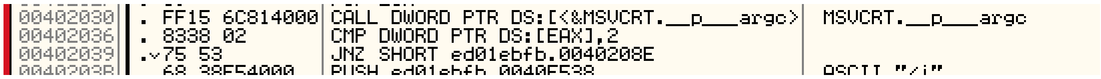
Replaced with NOP



```
004020B4 |> 8D85 F4FDFFFF LEA EAX,DWORD PTR SS:[EBP-20C]
004020BA |. 50            PUSH EAX                          [Path
004020BB    90            NOP                               [SetCurrentDirectoryA
004020BC    90            NOP
004020BD    90            NOP
004020BE    90            NOP
004020BF    90            NOP
004020C0    90            NOP
004020C1 |. 6A 01         PUSH 1
```

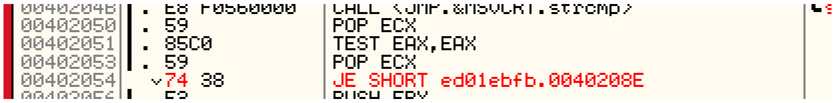
4.1.5 Reversing Jump Condition for Main Routine

Before (Jumping to loc_40208E)



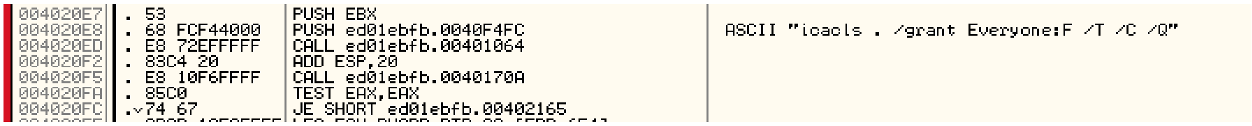
```
00402030 |. FF15 6C814000 CALL DWORD PTR DS:[<&MSVCRT.__p___argc>  MSVCRT.__p___argc
00402036 |. 8338 02       CMP DWORD PTR DS:[EAX],2
00402039 |.v75 53         JNZ SHORT ed01ebfb.0040208E
0040203B |. 68 38F54000   PUSH ed01ebfb.0040F538              ASCII "/i"
```

Reversing the jump



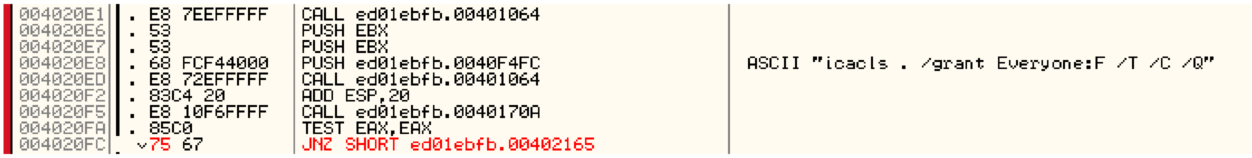
```
0040204B |. E8 F0560000   CALL <JMP.&MSVCRT.strcmp>
00402050 |. 59            POP ECX
00402051 |. 85C0          TEST EAX,EAX
00402053 |. 59            POP ECX
00402054 |.v74 38         JE SHORT ed01ebfb.0040208E
00402056 |. 53            PUSH EBX
```

Before (Jumping to loc_402165)



```
004020E7 |. 53            PUSH EBX
004020E8 |. 68 FCF44000   PUSH ed01ebfb.0040F4FC             ASCII "icacls . /grant Everyone:F /T /C /Q"
004020ED |. E8 72EFFFFF   CALL ed01ebfb.00401064
004020F2 |. 83C4 20       ADD ESP,20
004020F5 |. E8 10F6FFFF   CALL ed01ebfb.0040170A
004020FA |. 85C0          TEST EAX,EAX
004020FC |.v74 67         JE SHORT ed01ebfb.00402165
```

Reversing the jump to end the main routine

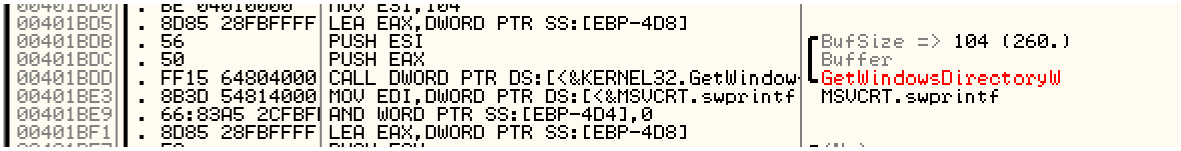


```
004020E1  . E8 7EEFFFFF    CALL ed01ebfb.00401064
004020E6  . 53             PUSH EBX
004020E7  . 53             PUSH EBX
004020E8  . 68 FCF44000    PUSH ed01ebfb.0040F4FC                 ASCII "icacls . /grant Everyone:F /T /C /Q"
004020ED  . E8 72EFFFFF    CALL ed01ebfb.00401064
004020F2  . 83C4 20        ADD ESP,20
004020F5  . E8 10F6FFFF    CALL ed01ebfb.0040170A
004020FA  . 85C0           TEST EAX,EAX
004020FC  .v75 67          JNZ SHORT ed01ebfb.00402165
```

4.2 Subroutine 1 : sub_401B5F

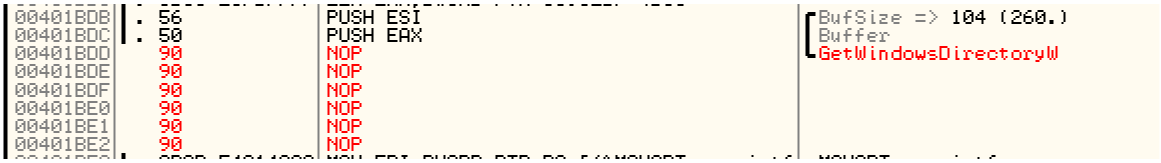
4.2.1 GetWindowsDirectoryW

Before



```
00401BD0  . BE 04010000    MOV ESI,104
00401BD5  . 8D85 28FBFFFF  LEA EAX,DWORD PTR SS:[EBP-4D8]
00401BDB  . 56             PUSH ESI                                   [BufSize => 104 (260.)
00401BDC  . 50             PUSH EAX                                   |Buffer
00401BDD  . FF15 64804000  CALL DWORD PTR DS:[<&KERNEL32.GetWindow    [GetWindowsDirectoryW
00401BE3  . 8B3D 54814000  MOV EDI,DWORD PTR DS:[<&MSVCRT.swprintf    MSVCRT.swprintf
00401BE9  . 66:83A5 2CFBFI AND WORD PTR SS:[EBP-4D4],0
00401BF1  . 8D85 28FBFFFF  LEA EAX,DWORD PTR SS:[EBP-4D8]
```

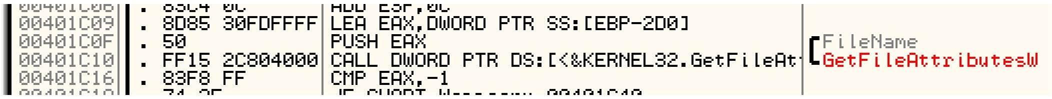
Replaced with NOP



```
00401BDB  . 56             PUSH ESI                                   [BufSize => 104 (260.)
00401BDC  . 50             PUSH EAX                                   |Buffer
00401BDD    90             NOP                                        [GetWindowsDirectoryW
00401BDE    90             NOP
00401BDF    90             NOP
00401BE0    90             NOP
00401BE1    90             NOP
00401BE2    90             NOP
```

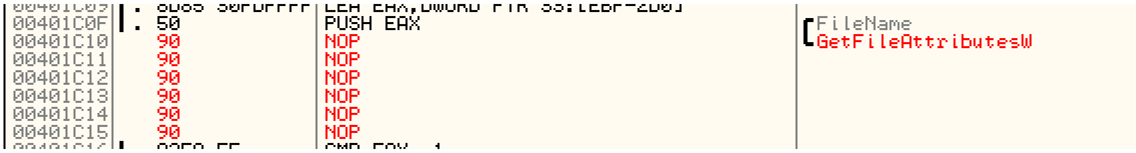
4.2.2 GetFileAttributesW

Before



```
00401C06  . 83C4 0C        ADD ESP,0C
00401C09  . 8D85 30FDFFFF  LEA EAX,DWORD PTR SS:[EBP-2D0]
00401C0F  . 50             PUSH EAX                                   [FileName
00401C10  . FF15 2C804000  CALL DWORD PTR DS:[<&KERNEL32.GetFileAt    [GetFileAttributesW
00401C16  . 83F8 FF        CMP EAX,-1
```

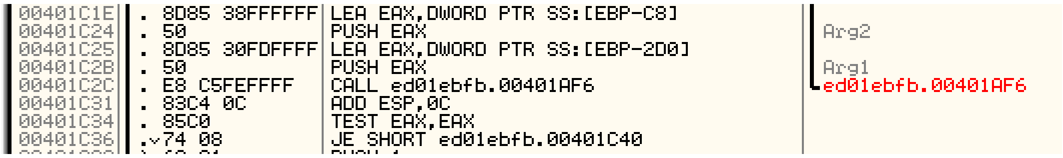
Replaced with NOP



```
00401C09  . 8D85 30FDFFFF  LEA EAX,DWORD PTR SS:[EBP-2D0]
00401C0F  . 50             PUSH EAX                                   [FileName
00401C10    90             NOP                                        [GetFileAttributesW
00401C11    90             NOP
00401C12    90             NOP
00401C13    90             NOP
00401C14    90             NOP
00401C15    90             NOP
```

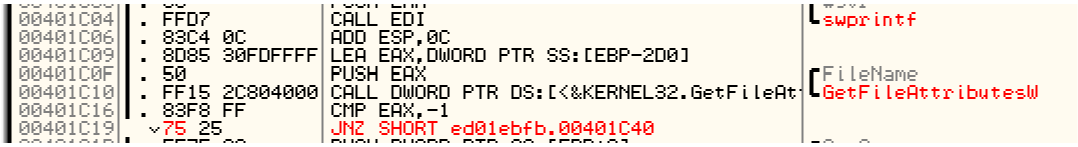
4.2.3 Reversing jumps

Before (jumping to loc_401C40)



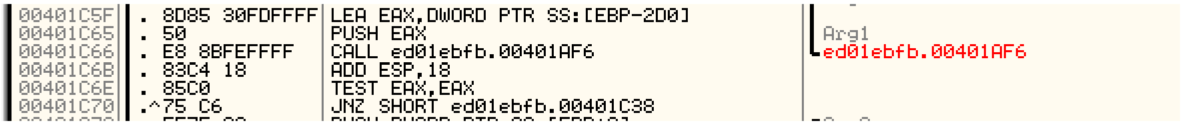
```
00401C1E  . 8D85 38FFFFFF LEA EAX,DWORD PTR SS:[EBP-C8]
00401C24  . 50            PUSH EAX                          Arg2
00401C25  . 8D85 30FDFFFF LEA EAX,DWORD PTR SS:[EBP-2D0]
00401C2B  . 50            PUSH EAX                          Arg1
00401C2C  . E8 C5FEFFFF   CALL ed01ebfb.00401AF6            ed01ebfb.00401AF6
00401C31  . 83C4 0C       ADD ESP,0C
00401C34  . 85C0          TEST EAX,EAX
00401C36  .v74 08         JE SHORT ed01ebfb.00401C40
```

Reversing jump



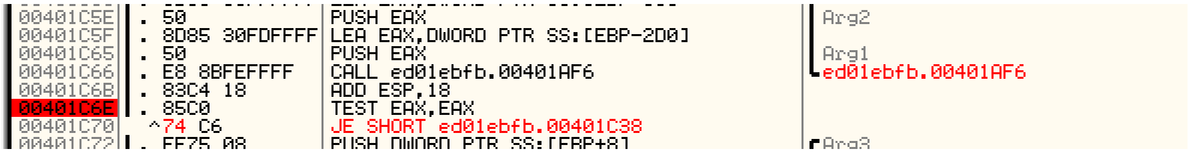
```
00401C04  . FFD7          CALL EDI                          swprintf
00401C06  . 83C4 0C       ADD ESP,0C
00401C09  . 8D85 30FDFFFF LEA EAX,DWORD PTR SS:[EBP-2D0]
00401C0F  . 50            PUSH EAX                          FileName
00401C10  . FF15 2C804000 CALL DWORD PTR DS:[<&KERNEL32.GetFileAt GetFileAttributesW
00401C16  . 83F8 FF       CMP EAX,-1
00401C19   v75 25         JNZ SHORT ed01ebfb.00401C40
```

Before (jumping to loc_401C38 which then jumps to end the subroutine)



```
00401C5F  . 8D85 30FDFFFF LEA EAX,DWORD PTR SS:[EBP-2D0]
00401C65  . 50            PUSH EAX                          Arg1
00401C66  . E8 8BFEFFFF   CALL ed01ebfb.00401AF6            ed01ebfb.00401AF6
00401C6B  . 83C4 18       ADD ESP,18
00401C6E  . 85C0          TEST EAX,EAX
00401C70  .^75 C6         JNZ SHORT ed01ebfb.00401C38
```

Reversing jump

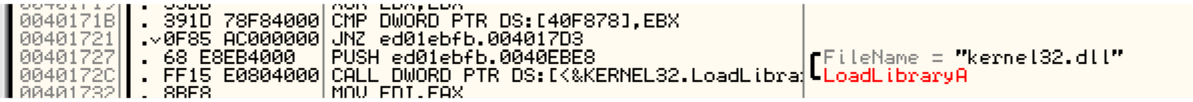


```
00401C5E  . 50            PUSH EAX                          Arg2
00401C5F  . 8D85 30FDFFFF LEA EAX,DWORD PTR SS:[EBP-2D0]
00401C65  . 50            PUSH EAX                          Arg1
00401C66  . E8 8BFEFFFF   CALL ed01ebfb.00401AF6            ed01ebfb.00401AF6
00401C6B  . 83C4 18       ADD ESP,18
00401C6E  . 85C0          TEST EAX,EAX
00401C70   ^74 C6         JE SHORT ed01ebfb.00401C38
00401C72  . FF75 08       PUSH DWORD PTR SS:[EBP+8]         Arg3
```

4.3 Subroutine 2: sub_40170A

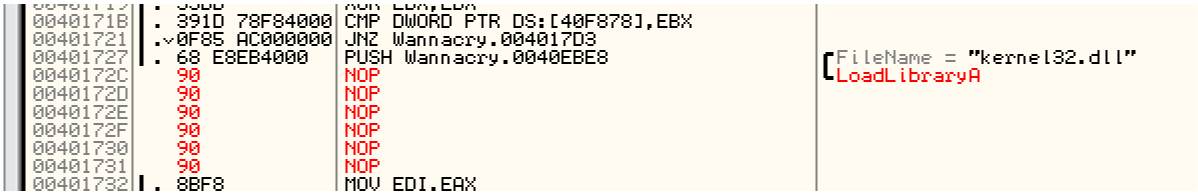
4.3.1 LoadLibraryA

Before



```
0040171B  . 391D 78F84000 CMP DWORD PTR DS:[40F878],EBX
00401721  .v0F85 AC000000 JNZ ed01ebfb.004017D3
00401727  . 68 E8EB4000   PUSH ed01ebfb.0040EBE8            FileName = "kernel32.dll"
0040172C  . FF15 E0804000 CALL DWORD PTR DS:[<&KERNEL32.LoadLibra: LoadLibraryA
00401732  . 8BF8          MOV EDI,EAX
```

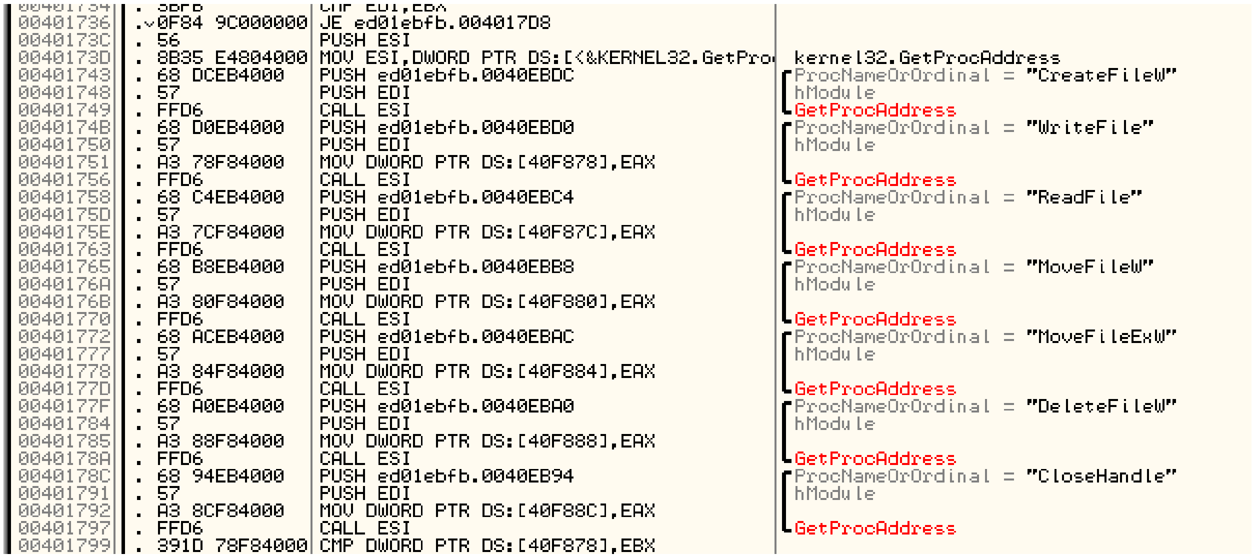
Replaced with NOP



```
0040171B  . 391D 78F84000 CMP DWORD PTR DS:[40F878],EBX
00401721  .v0F85 AC000000 JNZ Wannacry.004017D3
00401727  . 68 E8EB4000   PUSH Wannacry.0040EBE8            FileName = "kernel32.dll"
0040172C    90            NOP                               LoadLibraryA
0040172D    90            NOP
0040172E    90            NOP
0040172F    90            NOP
00401730    90            NOP
00401731    90            NOP
00401732  . 8BF8          MOV EDI,EAX
```

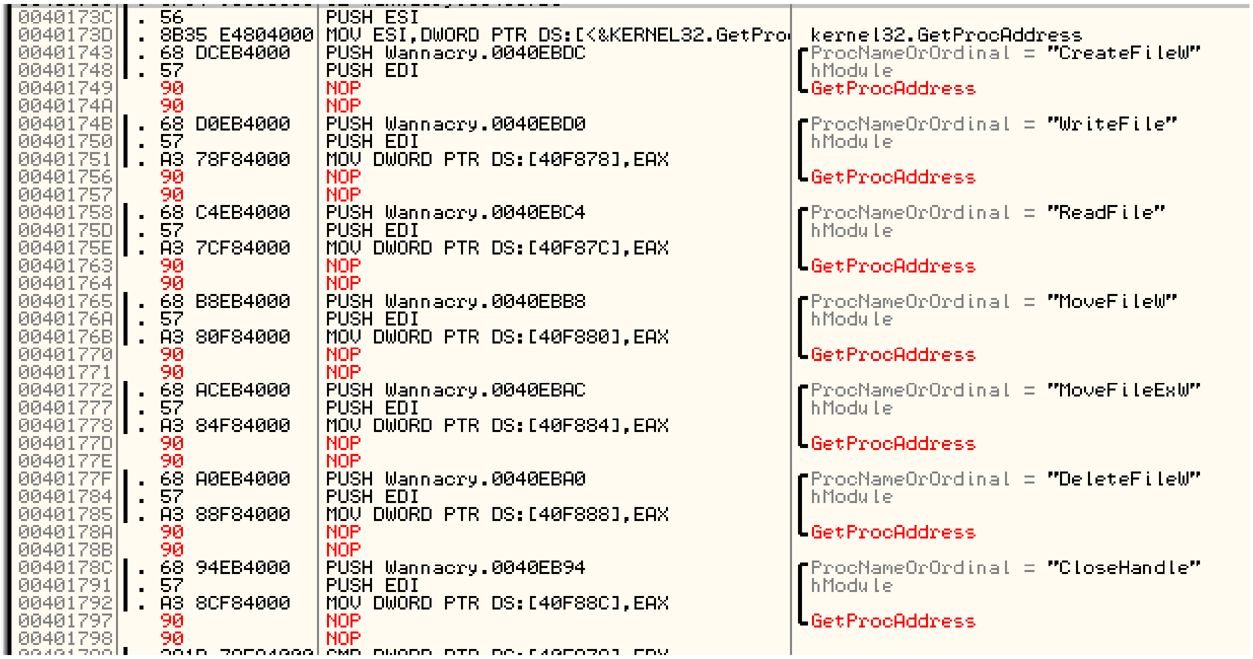
4.3.2 GetProcAddress

Before



```
00401734  . 3BFB            CMP EDI,EBX
00401736  .v0F84 9C000000   JE ed01ebfb.004017D8
0040173C  . 56              PUSH ESI
0040173D  . 8B35 E4804000   MOV ESI,DWORD PTR DS:[<&KERNEL32.GetPro   kernel32.GetProcAddress
00401743  . 68 DCEB4000     PUSH ed01ebfb.0040EBDC                    ProcNameOrOrdinal = "CreateFileW"
00401748  . 57              PUSH EDI                                  hModule
00401749  . FFD6            CALL ESI                                  GetProcAddress
0040174B  . 68 D0EB4000     PUSH ed01ebfb.0040EBD0                    ProcNameOrOrdinal = "WriteFile"
00401750  . 57              PUSH EDI                                  hModule
00401751  . A3 78F84000     MOV DWORD PTR DS:[40F878],EAX
00401756  . FFD6            CALL ESI                                  GetProcAddress
00401758  . 68 C4EB4000     PUSH ed01ebfb.0040EBC4                    ProcNameOrOrdinal = "ReadFile"
0040175D  . 57              PUSH EDI                                  hModule
0040175E  . A3 7CF84000     MOV DWORD PTR DS:[40F87C],EAX
00401763  . FFD6            CALL ESI                                  GetProcAddress
00401765  . 68 B8EB4000     PUSH ed01ebfb.0040EBB8                    ProcNameOrOrdinal = "MoveFileW"
0040176A  . 57              PUSH EDI                                  hModule
0040176B  . A3 80F84000     MOV DWORD PTR DS:[40F880],EAX
00401770  . FFD6            CALL ESI                                  GetProcAddress
00401772  . 68 ACEB4000     PUSH ed01ebfb.0040EBAC                    ProcNameOrOrdinal = "MoveFileExW"
00401777  . 57              PUSH EDI                                  hModule
00401778  . A3 84F84000     MOV DWORD PTR DS:[40F884],EAX
0040177D  . FFD6            CALL ESI                                  GetProcAddress
0040177F  . 68 A0EB4000     PUSH ed01ebfb.0040EBA0                    ProcNameOrOrdinal = "DeleteFileW"
00401784  . 57              PUSH EDI                                  hModule
00401785  . A3 88F84000     MOV DWORD PTR DS:[40F888],EAX
0040178A  . FFD6            CALL ESI                                  GetProcAddress
0040178C  . 68 94EB4000     PUSH ed01ebfb.0040EB94                    ProcNameOrOrdinal = "CloseHandle"
00401791  . 57              PUSH EDI                                  hModule
00401792  . A3 8CF84000     MOV DWORD PTR DS:[40F88C],EAX
00401797  . FFD6            CALL ESI                                  GetProcAddress
00401799  . 391D 78F84000   CMP DWORD PTR DS:[40F878],EBX
```

Replaced with NOP

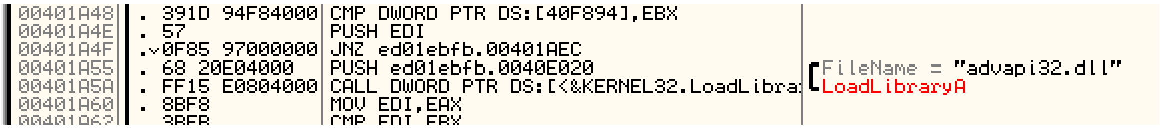


```
0040173C  . 56              PUSH ESI
0040173D  . 8B35 E4804000   MOV ESI,DWORD PTR DS:[<&KERNEL32.GetPro   kernel32.GetProcAddress
00401743  . 68 DCEB4000     PUSH Wannacry.0040EBDC                    ProcNameOrOrdinal = "CreateFileW"
00401748  . 57              PUSH EDI                                  hModule
00401749    90              NOP                                       GetProcAddress
0040174A    90              NOP
0040174B  . 68 D0EB4000     PUSH Wannacry.0040EBD0                    ProcNameOrOrdinal = "WriteFile"
00401750  . 57              PUSH EDI                                  hModule
00401751  . A3 78F84000     MOV DWORD PTR DS:[40F878],EAX
00401756    90              NOP                                       GetProcAddress
00401757    90              NOP
00401758  . 68 C4EB4000     PUSH Wannacry.0040EBC4                    ProcNameOrOrdinal = "ReadFile"
0040175D  . 57              PUSH EDI                                  hModule
0040175E  . A3 7CF84000     MOV DWORD PTR DS:[40F87C],EAX
00401763    90              NOP                                       GetProcAddress
00401764    90              NOP
00401765  . 68 B8EB4000     PUSH Wannacry.0040EBB8                    ProcNameOrOrdinal = "MoveFileW"
0040176A  . 57              PUSH EDI                                  hModule
0040176B  . A3 80F84000     MOV DWORD PTR DS:[40F880],EAX
00401770    90              NOP                                       GetProcAddress
00401771    90              NOP
00401772  . 68 ACEB4000     PUSH Wannacry.0040EBAC                    ProcNameOrOrdinal = "MoveFileExW"
00401777  . 57              PUSH EDI                                  hModule
00401778  . A3 84F84000     MOV DWORD PTR DS:[40F884],EAX
0040177D    90              NOP                                       GetProcAddress
0040177E    90              NOP
0040177F  . 68 A0EB4000     PUSH Wannacry.0040EBA0                    ProcNameOrOrdinal = "DeleteFileW"
00401784  . 57              PUSH EDI                                  hModule
00401785  . A3 88F84000     MOV DWORD PTR DS:[40F888],EAX
0040178A    90              NOP                                       GetProcAddress
0040178B    90              NOP
0040178C  . 68 94EB4000     PUSH Wannacry.0040EB94                    ProcNameOrOrdinal = "CloseHandle"
00401791  . 57              PUSH EDI                                  hModule
00401792  . A3 8CF84000     MOV DWORD PTR DS:[40F88C],EAX
00401797    90              NOP                                       GetProcAddress
00401798    90              NOP
```

4.3 Subroutine 3 : sub_401A45

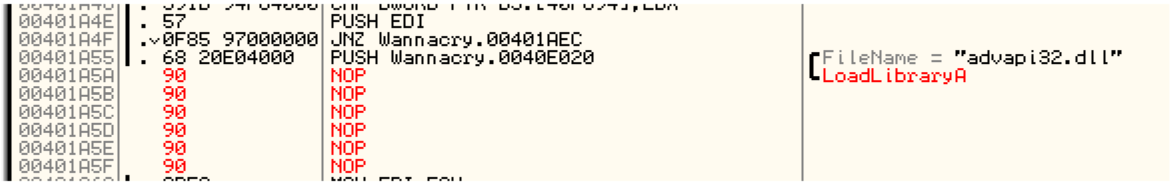
4.3.1 LoadLibraryA

Before



```
00401A48  . 391D 94F84000  CMP DWORD PTR DS:[40F894],EBX
00401A4E  . 57             PUSH EDI
00401A4F  .v0F85 97000000  JNZ ed01ebfb.00401AEC
00401A55  . 68 20E04000    PUSH ed01ebfb.0040E020                  [FileName = "advapi32.dll"
00401A5A  . FF15 E0804000  CALL DWORD PTR DS:[<&KERNEL32.LoadLibra  LoadLibraryA
00401A60  . 8BF8           MOV EDI,EAX
00401A62  . 3BFB           CMP EDI,EBX
```

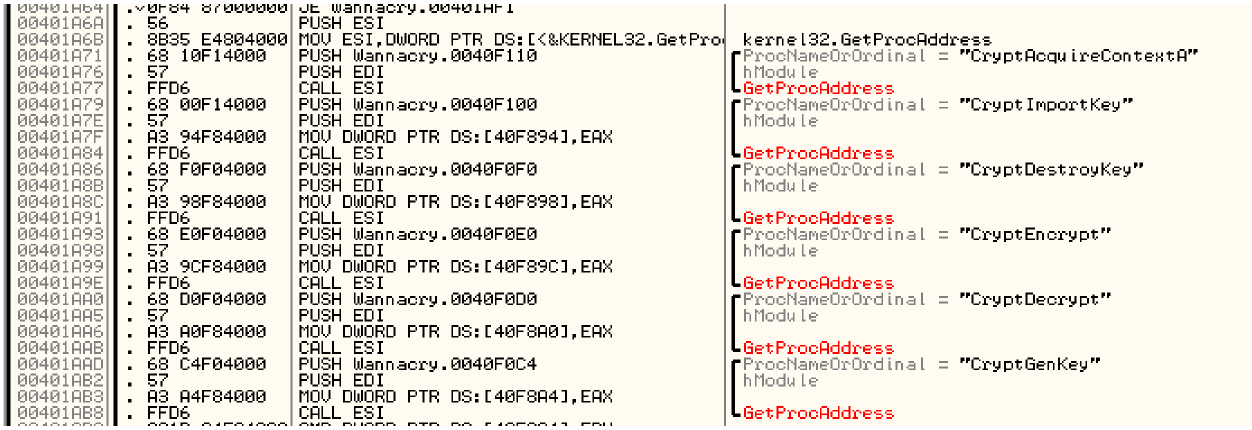
Replaced with NOP



```
00401A48  . 391D 94F84000  CMP DWORD PTR DS:[40F894],EBX
00401A4E  . 57             PUSH EDI
00401A4F  .v0F85 97000000  JNZ Wannacry.00401AEC
00401A55  . 68 20E04000    PUSH Wannacry.0040E020                  [FileName = "advapi32.dll"
00401A5A    90             NOP                                     LoadLibraryA
00401A5B    90             NOP
00401A5C    90             NOP
00401A5D    90             NOP
00401A5E    90             NOP
00401A5F    90             NOP
00401A60  . 8BF8           MOV EDI,EAX
```

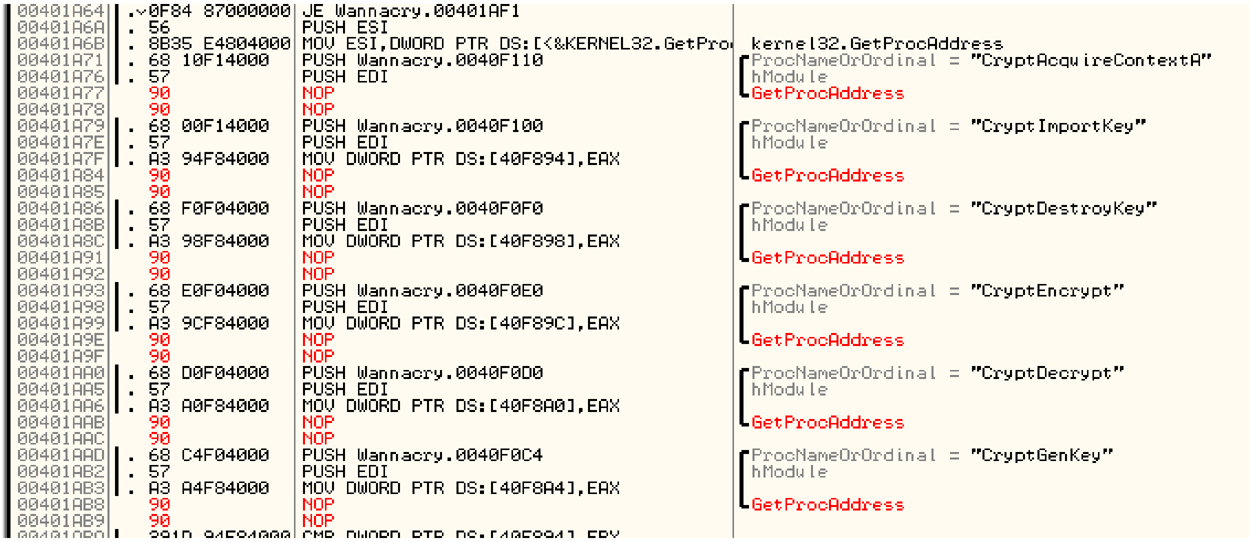
4.3.2 Get ProcAddress

Before



```
00401A64  .v0F84 87000000  JE Wannacry.00401AF1
00401A6A  . 56             PUSH ESI
00401A6B  . 8B35 E4804000  MOV ESI,DWORD PTR DS:[<&KERNEL32.GetPro   kernel32.GetProcAddress
00401A71  . 68 10F14000    PUSH Wannacry.0040F110                   [ProcNameOrOrdinal = "CryptAcquireContextA"
00401A76  . 57             PUSH EDI                                  hModule
00401A77  . FFD6           CALL ESI                                  GetProcAddress
00401A79  . 68 00F14000    PUSH Wannacry.0040F100                   [ProcNameOrOrdinal = "CryptImportKey"
00401A7E  . 57             PUSH EDI                                  hModule
00401A7F  . A3 94F84000    MOV DWORD PTR DS:[40F894],EAX
00401A84  . FFD6           CALL ESI                                  GetProcAddress
00401A86  . 68 F0F04000    PUSH Wannacry.0040F0F0                   [ProcNameOrOrdinal = "CryptDestroyKey"
00401A8B  . 57             PUSH EDI                                  hModule
00401A8C  . A3 98F84000    MOV DWORD PTR DS:[40F898],EAX
00401A91  . FFD6           CALL ESI                                  GetProcAddress
00401A93  . 68 E0F04000    PUSH Wannacry.0040F0E0                   [ProcNameOrOrdinal = "CryptEncrypt"
00401A98  . 57             PUSH EDI                                  hModule
00401A99  . A3 9CF84000    MOV DWORD PTR DS:[40F89C],EAX
00401A9E  . FFD6           CALL ESI                                  GetProcAddress
00401AA0  . 68 D0F04000    PUSH Wannacry.0040F0D0                   [ProcNameOrOrdinal = "CryptDecrypt"
00401AA5  . 57             PUSH EDI                                  hModule
00401AA6  . A3 A0F84000    MOV DWORD PTR DS:[40F8A0],EAX
00401AAB  . FFD6           CALL ESI                                  GetProcAddress
00401AAD  . 68 C4F04000    PUSH Wannacry.0040F0C4                   [ProcNameOrOrdinal = "CryptGenKey"
00401AB2  . 57             PUSH EDI                                  hModule
00401AB3  . A3 A4F84000    MOV DWORD PTR DS:[40F8A4],EAX
00401AB8  . FFD6           CALL ESI                                  GetProcAddress
```

Replaced with NOP

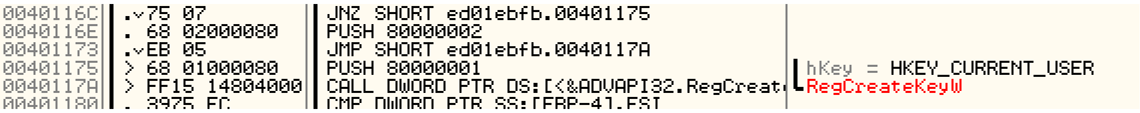


```
00401A64 .v0F84 87000000 JE Wannacry.00401AF1
00401A6A . 56            PUSH ESI
00401A6B . 8B35 E4804000 MOV ESI,DWORD PTR DS:[<&KERNEL32.GetPro  kernel32.GetProcAddress
00401A71 . 68 10F14000   PUSH Wannacry.0040F110                   ┌ProcNameOrOrdinal = "CryptAcquireContextA"
00401A76 . 57            PUSH EDI                                 │hModule
00401A77   90            NOP                                      └GetProcAddress
00401A78   90            NOP
00401A79 . 68 00F14000   PUSH Wannacry.0040F100                   ┌ProcNameOrOrdinal = "CryptImportKey"
00401A7E . 57            PUSH EDI                                 │hModule
00401A7F . A3 94F84000   MOV DWORD PTR DS:[40F894],EAX            │
00401A84   90            NOP                                      └GetProcAddress
00401A85   90            NOP
00401A86 . 68 F0F04000   PUSH Wannacry.0040F0F0                   ┌ProcNameOrOrdinal = "CryptDestroyKey"
00401A8B . 57            PUSH EDI                                 │hModule
00401A8C . A3 98F84000   MOV DWORD PTR DS:[40F898],EAX            │
00401A91   90            NOP                                      └GetProcAddress
00401A92   90            NOP
00401A93 . 68 E0F04000   PUSH Wannacry.0040F0E0                   ┌ProcNameOrOrdinal = "CryptEncrypt"
00401A98 . 57            PUSH EDI                                 │hModule
00401A99 . A3 9CF84000   MOV DWORD PTR DS:[40F89C],EAX            │
00401A9E   90            NOP                                      └GetProcAddress
00401A9F   90            NOP
00401AA0 . 68 D0F04000   PUSH Wannacry.0040F0D0                   ┌ProcNameOrOrdinal = "CryptDecrypt"
00401AA5 . 57            PUSH EDI                                 │hModule
00401AA6 . A3 A0F84000   MOV DWORD PTR DS:[40F8A0],EAX            │
00401AAB   90            NOP                                      └GetProcAddress
00401AAC   90            NOP
00401AAD . 68 C4F04000   PUSH Wannacry.0040F0C4                   ┌ProcNameOrOrdinal = "CryptGenKey"
00401AB2 . 57            PUSH EDI                                 │hModule
00401AB3 . A3 A4F84000   MOV DWORD PTR DS:[40F8A4],EAX            │
00401AB8   90            NOP                                      └GetProcAddress
00401AB9   90            NOP
00401ABA . 391D 94F84000 CMP DWORD PTR DS:[40F894],EBX
```

4.4 Subroutine 4: sub_4010FD

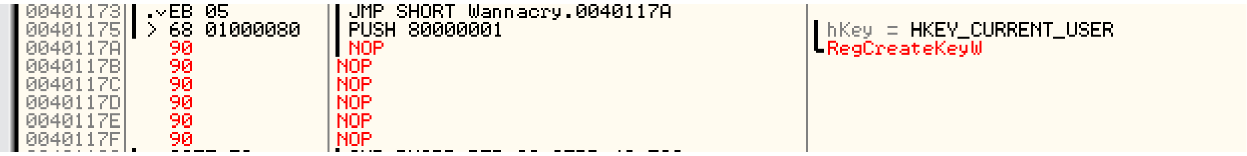
4.4.1 RegCreateKeyW

Before



```
0040116C .v75 07          JNZ SHORT ed01ebfb.00401175
0040116E . 68 02000080    PUSH 80000002
00401173 .vEB 05          JMP SHORT ed01ebfb.0040117A
00401175 > 68 01000080    PUSH 80000001                           ┌hKey = HKEY_CURRENT_USER
0040117A > FF15 14804000  CALL DWORD PTR DS:[<&ADVAPI32.RegCreat  └RegCreateKeyW
00401180 . 3975 FC        CMP DWORD PTR SS:[EBP-4],ESI
```

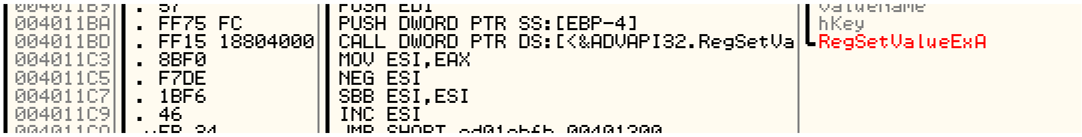
Replaced with NOP



```
00401173 .vEB 05          JMP SHORT Wannacry.0040117A
00401175 > 68 01000080    PUSH 80000001                           ┌hKey = HKEY_CURRENT_USER
0040117A   90             NOP                                     └RegCreateKeyW
0040117B   90             NOP
0040117C   90             NOP
0040117D   90             NOP
0040117E   90             NOP
0040117F   90             NOP
```

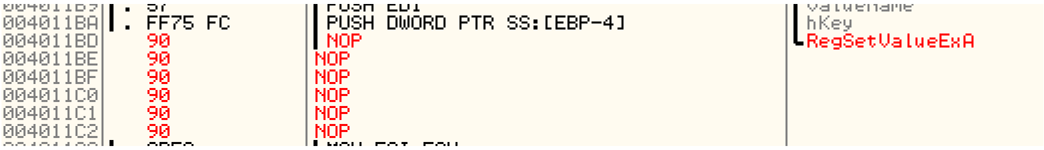
4.4.2 RegSetValueExA

Before



```
004011B9 . 57             PUSH EDI                                │ValueName
004011BA . FF75 FC        PUSH DWORD PTR SS:[EBP-4]               │hKey
004011BD . FF15 18804000  CALL DWORD PTR DS:[<&ADVAPI32.RegSetVa  └RegSetValueExA
004011C3 . 8BF0           MOV ESI,EAX
004011C5 . F7DE           NEG ESI
004011C7 . 1BF6           SBB ESI,ESI
004011C9 . 46             INC ESI
004011CA .vEB 34          JMP SHORT ed01ebfb.00401200
```

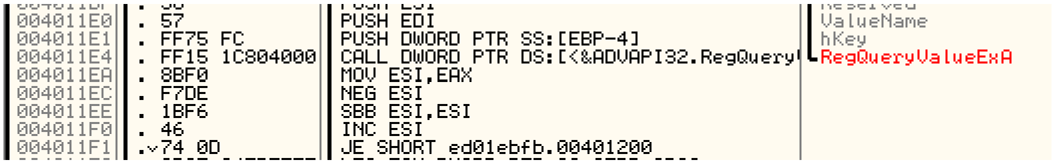
Replaced with NOP



```
004011BA  .  FF75 FC       PUSH DWORD PTR SS:[EBP-4]      hKey
004011BD     90            NOP                            RegSetValueExA
004011BE     90            NOP
004011BF     90            NOP
004011C0     90            NOP
004011C1     90            NOP
004011C2     90            NOP
```

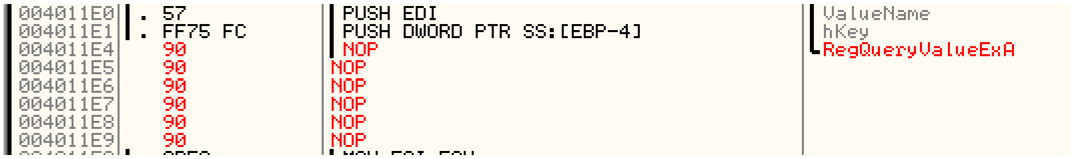
4.4.3 RegQueryValueExA

Before



```
004011E0  .  57             PUSH EDI                                   ValueName
004011E1  .  FF75 FC        PUSH DWORD PTR SS:[EBP-4]                  hKey
004011E4  .  FF15 1C804000  CALL DWORD PTR DS:[<&ADVAPI32.RegQuery     RegQueryValueExA
004011EA  .  8BF0           MOV ESI,EAX
004011EC  .  F7DE           NEG ESI
004011EE  .  1BF6           SBB ESI,ESI
004011F0  .  46             INC ESI
004011F1  .v 74 0D          JE SHORT ed01ebfb.00401200
```

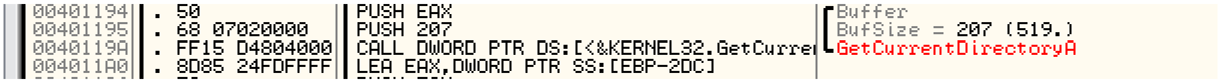
Replaced with NOP



```
004011E0  .  57             PUSH EDI                                   ValueName
004011E1  .  FF75 FC        PUSH DWORD PTR SS:[EBP-4]                  hKey
004011E4     90             NOP                                        RegQueryValueExA
004011E5     90             NOP
004011E6     90             NOP
004011E7     90             NOP
004011E8     90             NOP
004011E9     90             NOP
```

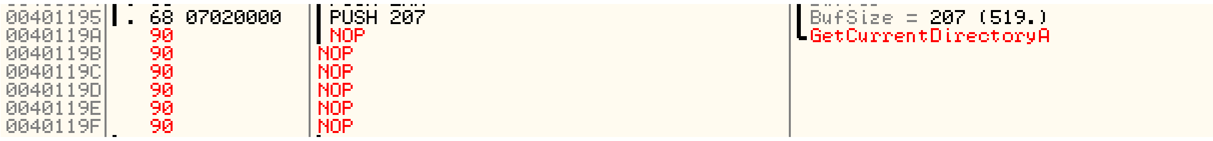
4.4.4 GetCurrentDirectoryA

Before



```
00401194  .  50             PUSH EAX                                   Buffer
00401195  .  68 07020000    PUSH 207                                   BufSize = 207 (519.)
0040119A  .  FF15 D4804000  CALL DWORD PTR DS:[<&KERNEL32.GetCurre     GetCurrentDirectoryA
004011A0  .  8D85 24FDFFFF  LEA EAX,DWORD PTR SS:[EBP-2DC]
```

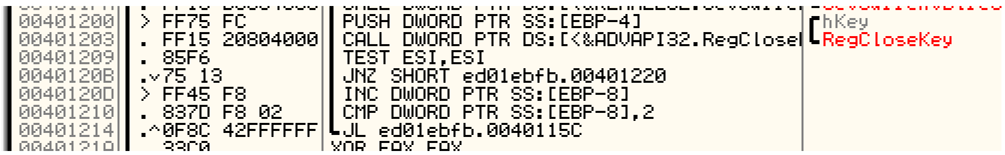
Replaced with NOP



```
00401195  .  68 07020000    PUSH 207                                   BufSize = 207 (519.)
0040119A     90             NOP                                        GetCurrentDirectoryA
0040119B     90             NOP
0040119C     90             NOP
0040119D     90             NOP
0040119E     90             NOP
0040119F     90             NOP
```

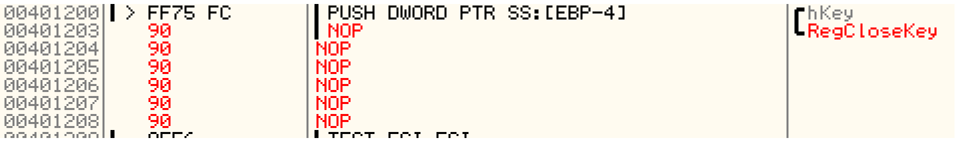
4.4.5 RegCloseKey

Before



```
00401200  >  FF75 FC        PUSH DWORD PTR SS:[EBP-4]                  hKey
00401203  .  FF15 20804000  CALL DWORD PTR DS:[<&ADVAPI32.RegClose     RegCloseKey
00401209  .  85F6           TEST ESI,ESI
0040120B  .v 75 13          JNZ SHORT ed01ebfb.00401220
0040120D  >  FF45 F8        INC DWORD PTR SS:[EBP-8]
00401210  .  837D F8 02     CMP DWORD PTR SS:[EBP-8],2
00401214  .^ 0F8C 42FFFFFF  JL ed01ebfb.0040115C
```

Replaced with NOP

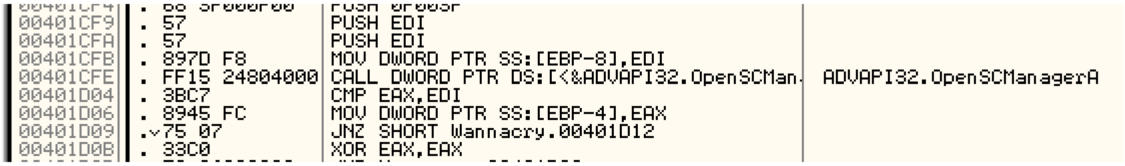


```
00401200  > FF75 FC        PUSH DWORD PTR SS:[EBP-4]          [hKey
00401203    90             NOP                                [RegCloseKey
00401204    90             NOP
00401205    90             NOP
00401206    90             NOP
00401207    90             NOP
00401208    90             NOP
```

4.5 Subroutine 5: sub_401CE8

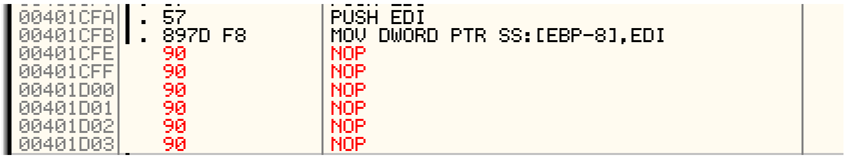
4.5.1 OpenSCManagerA

Before



```
00401CF9  . 57             PUSH EDI
00401CFA  . 57             PUSH EDI
00401CFB  . 897D F8        MOV DWORD PTR SS:[EBP-8],EDI
00401CFE  . FF15 24804000  CALL DWORD PTR DS:[<&ADVAPI32.OpenSCMan  ADVAPI32.OpenSCManagerA
00401D04  . 3BC7           CMP EAX,EDI
00401D06  . 8945 FC        MOV DWORD PTR SS:[EBP-4],EAX
00401D09  .v75 07          JNZ SHORT Wannacry.00401D12
00401D0B  . 33C0           XOR EAX,EAX
```

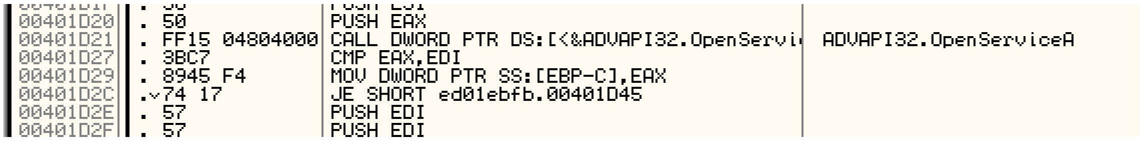
Replaced with NOP



```
00401CFA  . 57             PUSH EDI
00401CFB  . 897D F8        MOV DWORD PTR SS:[EBP-8],EDI
00401CFE    90             NOP
00401CFF    90             NOP
00401D00    90             NOP
00401D01    90             NOP
00401D02    90             NOP
00401D03    90             NOP
```

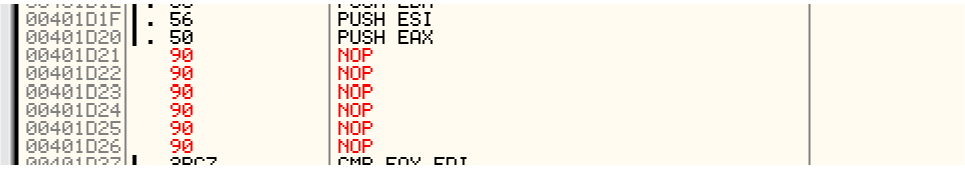
4.5.2 OpenServiceA

Before



```
00401D20  . 50             PUSH EAX
00401D21  . FF15 04804000  CALL DWORD PTR DS:[<&ADVAPI32.OpenServi  ADVAPI32.OpenServiceA
00401D27  . 3BC7           CMP EAX,EDI
00401D29  . 8945 F4        MOV DWORD PTR SS:[EBP-C],EAX
00401D2C  .v74 17          JE SHORT ed01ebfb.00401D45
00401D2E  . 57             PUSH EDI
00401D2F  . 57             PUSH EDI
```

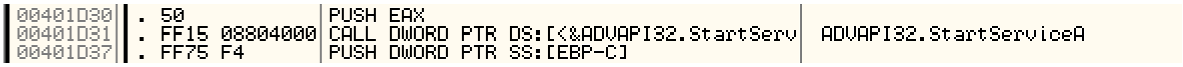
Replaced with NOP



```
00401D1F  . 56             PUSH ESI
00401D20  . 50             PUSH EAX
00401D21    90             NOP
00401D22    90             NOP
00401D23    90             NOP
00401D24    90             NOP
00401D25    90             NOP
00401D26    90             NOP
```

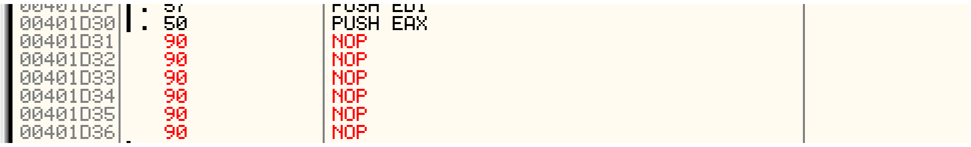
4.5.3 StartServiceA

Before



```
00401D30 . 50            PUSH EAX
00401D31 . FF15 08804000 CALL DWORD PTR DS:[<&ADVAPI32.StartServ   ADVAPI32.StartServiceA
00401D37 . FF75 F4       PUSH DWORD PTR SS:[EBP-C]
```

Replaced with NOP

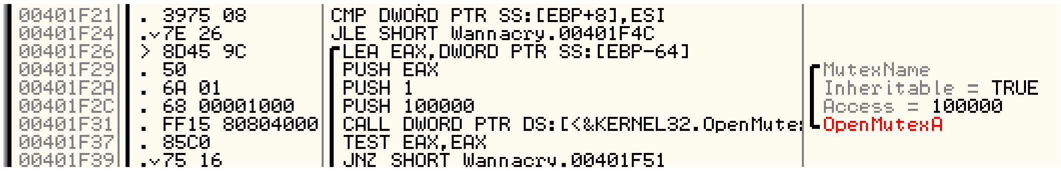


```
00401D2F . 57            PUSH EDI
00401D30 . 50            PUSH EAX
00401D31   90            NOP
00401D32   90            NOP
00401D33   90            NOP
00401D34   90            NOP
00401D35   90            NOP
00401D36   90            NOP
```

4.6 Subroutine 6: sub_401EFF

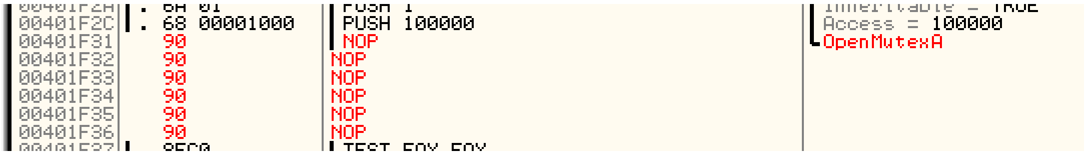
4.6.1 OpenMutexA

Before



```
00401F21 .  3975 08       CMP DWORD PTR SS:[EBP+8],ESI
00401F24 .v7E 26          JLE SHORT Wannacry.00401F4C
00401F26 >  8D45 9C       ┌LEA EAX,DWORD PTR SS:[EBP-64]
00401F29 .  50            │PUSH EAX                             ┌MutexName
00401F2A .  6A 01         │PUSH 1                               │Inheritable = TRUE
00401F2C .  68 00001000   │PUSH 100000                          │Access = 100000
00401F31 .  FF15 80804000 │CALL DWORD PTR DS:[<&KERNEL32.OpenMute └OpenMutexA
00401F37 .  85C0          │TEST EAX,EAX
00401F39 .v75 16          │JNZ SHORT Wannacry.00401F51
```

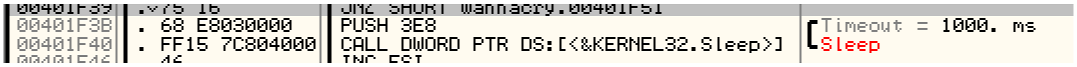
Replaced with NOP



```
00401F2A . 6A 01         PUSH 1                               Inheritable = TRUE
00401F2C . 68 00001000   PUSH 100000                          Access = 100000
00401F31   90            NOP                                  OpenMutexA
00401F32   90            NOP
00401F33   90            NOP
00401F34   90            NOP
00401F35   90            NOP
00401F36   90            NOP
00401F37   85C0          TEST EAX,EAX
```

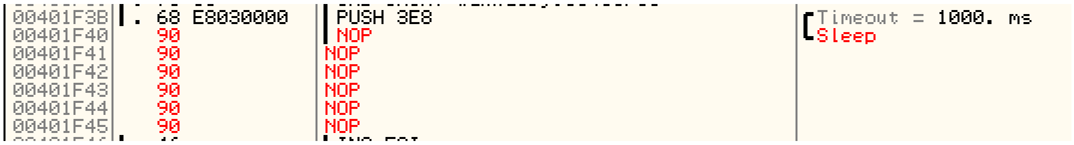
4.6.2 Sleep

Before



```
00401F39 .v75 16          JNZ SHORT Wannacry.00401F51
00401F3B .  68 E8030000   PUSH 3E8                                 ┌Timeout = 1000. ms
00401F40 .  FF15 7C804000 CALL DWORD PTR DS:[<&KERNEL32.Sleep>]    └Sleep
00401F46    46            INC ESI
```

Replaced with NOP

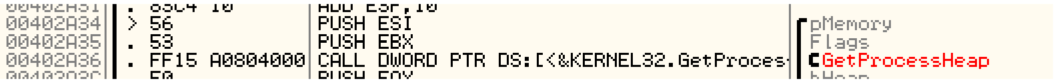


```
00401F3B . 68 E8030000   PUSH 3E8                              ┌Timeout = 1000. ms
00401F40   90            NOP                                   └Sleep
00401F41   90            NOP
00401F42   90            NOP
00401F43   90            NOP
00401F44   90            NOP
00401F45   90            NOP
00401F46   46            INC ESI
```

4.7 Subroutine 7: sub_4029CC

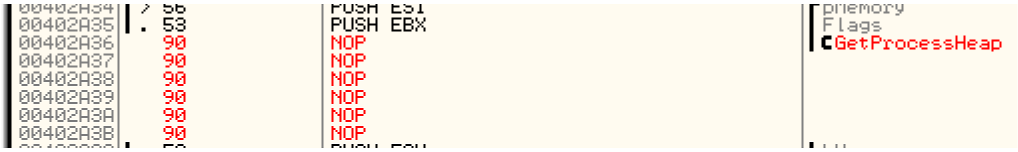
4.7.1 GetProcessHeap

Before



```
00402A31 . 83C4 10          ADD ESP,10
00402A34 > 56               PUSH ESI                                  pMemory
00402A35 . 53               PUSH EBX                                  Flags
00402A36 . FF15 A0804000    CALL DWORD PTR DS:[<&KERNEL32.GetProces   GetProcessHeap
00402A3C . 50               PUSH EAX                                  hHeap
```

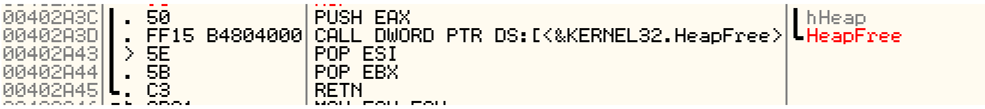
Replaced with NOP



```
00402A34 > 56               PUSH ESI                                  pMemory
00402A35 . 53               PUSH EBX                                  Flags
00402A36   90               NOP                                       GetProcessHeap
00402A37   90               NOP
00402A38   90               NOP
00402A39   90               NOP
00402A3A   90               NOP
00402A3B   90               NOP
```

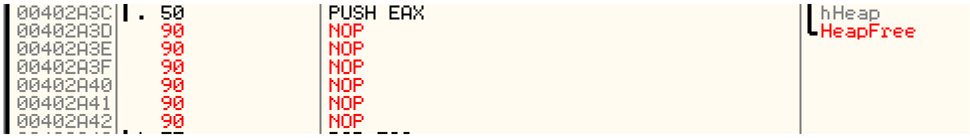
4.7.2 HeapFree

Before



```
00402A3C . 50               PUSH EAX                                  hHeap
00402A3D . FF15 B4804000    CALL DWORD PTR DS:[<&KERNEL32.HeapFree>   HeapFree
00402A43 > 5E               POP ESI
00402A44 . 5B               POP EBX
00402A45 . C3               RETN
```

Replaced with NOP

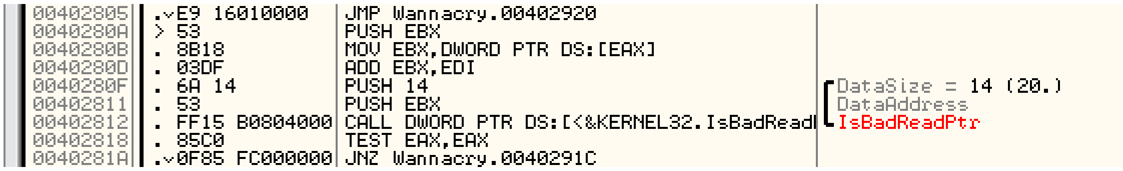


```
00402A3C . 50               PUSH EAX                                  hHeap
00402A3D   90               NOP                                       HeapFree
00402A3E   90               NOP
00402A3F   90               NOP
00402A40   90               NOP
00402A41   90               NOP
00402A42   90               NOP
```

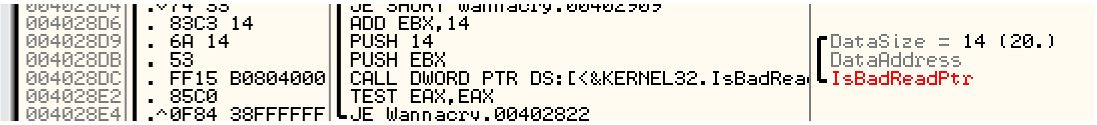
4.8 Subroutine 8 : sub_4027DF

4.8.1 IsBadReadPtr

Before

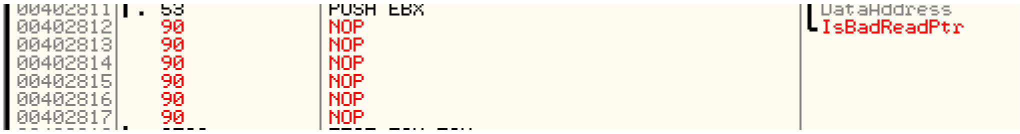


```
00402805 .vE9 16010000      JMP Wannacry.00402920
0040280A > 53               PUSH EBX
0040280B . 8B18             MOV EBX,DWORD PTR DS:[EAX]
0040280D . 03DF             ADD EBX,EDI
0040280F . 6A 14            PUSH 14                                   DataSize = 14 (20.)
00402811 . 53               PUSH EBX                                  DataAddress
00402812 . FF15 B0804000    CALL DWORD PTR DS:[<&KERNEL32.IsBadReadP  IsBadReadPtr
00402818 . 85C0             TEST EAX,EAX
0040281A .v0F85 FC000000    JNZ Wannacry.0040291C
```

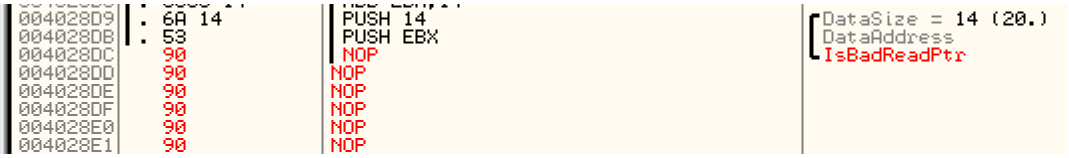


```
004028D4 .v74 33            JE SHORT Wannacry.00402909
004028D6 . 83C3 14          ADD EBX,14
004028D9 . 6A 14            PUSH 14                                   DataSize = 14 (20.)
004028DB . 53               PUSH EBX                                  DataAddress
004028DC . FF15 B0804000    CALL DWORD PTR DS:[<&KERNEL32.IsBadRea    IsBadReadPtr
004028E2 . 85C0             TEST EAX,EAX
004028E4 .^0F84 38FFFFFF    JE Wannacry.00402822
```

Replaced with NOP

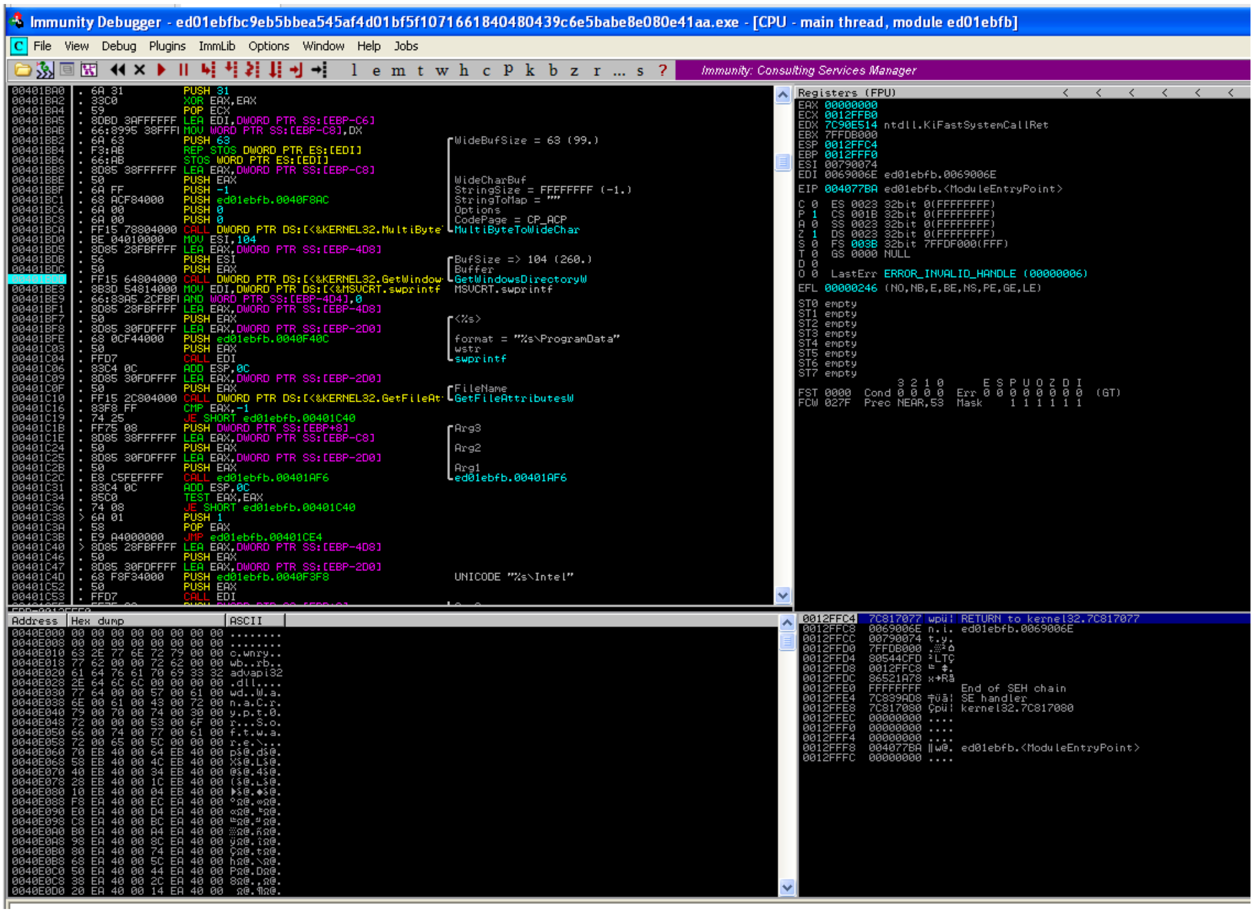


```
00402811 . 53    PUSH EBX      DataAddress
00402812   90    NOP           IsBadReadPtr
00402813   90    NOP
00402814   90    NOP
00402815   90    NOP
00402816   90    NOP
00402817   90    NOP
```



```
004028D9 . 6A 14    PUSH 14     DataSize = 14 (20.)
004028DB . 53       PUSH EBX    DataAddress
004028DC   90       NOP         IsBadReadPtr
004028DD   90       NOP
004028DE   90       NOP
004028DF   90       NOP
004028E0   90       NOP
004028E1   90       NOP
```

Immunity Debugger



Immunity Debugger - ed01ebfbc9eb5bbea545af4d01bf5f1071661840480439c6e5babe8e080e41aa.exe - [CPU - main thread, module ed01ebfb]

File
View
Debug
Plugins
ImmLib
Options
Window
Help
Jobs
l e m t w h c P k b z r ... s ?
Immunity: Consulting Services Manager

```
00401BA0 . 6A 31          PUSH 31
00401BA2 . 33C0           XOR EAX,EAX
00401BA4 . 59             POP ECX
00401BA5 . 8DBD 3AFFFFFF  LEA EDI,DWORD PTR SS:[EBP-C6]
00401BAB . 66:8995 38FFFI MOV WORD PTR SS:[EBP-C8],DX
00401BB2 . 6A 63          PUSH 63                                  ┌WideBufSize = 63 (99.)
00401BB4 . F3:AB          REP STOS DWORD PTR ES:[EDI]              │
00401BB6 . 66:AB          STOS WORD PTR ES:[EDI]                   │
00401BB8 . 8D85 38FFFFFF  LEA EAX,DWORD PTR SS:[EBP-C8]            │
00401BBE . 50             PUSH EAX                                 │WideCharBuf
00401BBF . 6A FF          PUSH -1                                  │StringSize = FFFFFFFF (-1.)
00401BC1 . 68 ACF84000    PUSH ed01ebfb.0040F8AC                   │StringToMap = ""
00401BC6 . 6A 00          PUSH 0                                   │Options
00401BC8 . 6A 00          PUSH 0                                   │CodePage = CP_ACP
00401BCA . FF15 78804000  CALL DWORD PTR DS:[<&KERNEL32.MultiByte' └MultiByteToWideChar
00401BD0 . BE 04010000    MOV ESI,104
00401BD5 . 8D85 28FBFFFF  LEA EAX,DWORD PTR SS:[EBP-4D8]
00401BDB . 56             PUSH ESI                                 ┌BufSize => 104 (260.)
00401BDC . 50             PUSH EAX                                 │Buffer
00401BDD . FF15 64804000  CALL DWORD PTR DS:[<&KERNEL32.GetWindow· └GetWindowsDirectoryW
00401BE3 . 8B3D 54814000  MOV EDI,DWORD PTR DS:[<&MSVCRT.swprintf   MSVCRT.swprintf
00401BE9 . 66:83A5 2CFBFI AND WORD PTR SS:[EBP-4D4],0
00401BF1 . 8D85 28FBFFFF  LEA EAX,DWORD PTR SS:[EBP-4D8]
00401BF7 . 50             PUSH EAX                                 ┌<%s>
00401BF8 . 8D85 30FDFFFF  LEA EAX,DWORD PTR SS:[EBP-2D0]           │
00401BFE . 68 0CF44000    PUSH ed01ebfb.0040F40C                   │format = "%s\ProgramData"
00401C03 . 50             PUSH EAX                                 │wstr
00401C04 . FFD7           CALL EDI                                 └swprintf
00401C06 . 83C4 0C        ADD ESP,0C
00401C09 . 8D85 30FDFFFF  LEA EAX,DWORD PTR SS:[EBP-2D0]
00401C0F . 50             PUSH EAX                                 ┌FileName
00401C10 . FF15 2C804000  CALL DWORD PTR DS:[<&KERNEL32.GetFileAt· └GetFileAttributesW
00401C16 . 83F8 FF        CMP EAX,-1
00401C19 . 74 25          JE SHORT ed01ebfb.00401C40
00401C1B . FF75 08        PUSH DWORD PTR SS:[EBP+8]                ┌Arg3
00401C1E . 8D85 38FFFFFF  LEA EAX,DWORD PTR SS:[EBP-C8]            │
00401C24 . 50             PUSH EAX                                 │Arg2
00401C25 . 8D85 30FDFFFF  LEA EAX,DWORD PTR SS:[EBP-2D0]           │
00401C2B . 50             PUSH EAX                                 │Arg1
00401C2C . E8 C5FEFFFF    CALL ed01ebfb.00401AF6                   └ed01ebfb.00401AF6
00401C31 . 83C4 0C        ADD ESP,0C
00401C34 . 85C0           TEST EAX,EAX
00401C36 . 74 08          JE SHORT ed01ebfb.00401C40
00401C38 > 6A 01          PUSH 1
00401C3A . 58             POP EAX
00401C3B . E9 A4000000    JMP ed01ebfb.00401CE4
00401C40 > 8D85 28FBFFFF  LEA EAX,DWORD PTR SS:[EBP-4D8]
00401C46 . 50             PUSH EAX
00401C47 . 8D85 30FDFFFF  LEA EAX,DWORD PTR SS:[EBP-2D0]
00401C4D . 68 F8F34000    PUSH ed01ebfb.0040F3F8                    UNICODE "%s\Intel"
00401C52 . 50             PUSH EAX
00401C53 . FFD7           CALL EDI
```

```
Registers (FPU)
EAX 00000000
ECX 0012FFB0
EDX 7C90E514 ntdll.KiFastSystemCallRet
EBX 7FFDB000
ESP 0012FFC4
EBP 0012FFF0
ESI 00790074
EDI 0069006E ed01ebfb.0069006E

EIP 004077BA ed01ebfb.<ModuleEntryPoint>

C 0  ES 0023 32bit 0(FFFFFFFF)
P 1  CS 001B 32bit 0(FFFFFFFF)
A 0  SS 0023 32bit 0(FFFFFFFF)
Z 1  DS 0023 32bit 0(FFFFFFFF)
S 0  FS 003B 32bit 7FFDF000(FFF)
T 0  GS 0000 NULL
D 0
O 0  LastErr ERROR_INVALID_HANDLE (00000006)

EFL 00000246 (NO,NB,E,BE,NS,PE,GE,LE)

ST0 empty
ST1 empty
ST2 empty
ST3 empty
ST4 empty
ST5 empty
ST6 empty
ST7 empty
               3 2 1 0      E S P U O Z D I
FST 0000  Cond 0 0 0 0  Err 0 0 0 0 0 0 0 0  (GT)
FCW 027F  Prec NEAR,53  Mask    1 1 1 1 1 1
```

```
Address  Hex dump                 ASCII
0040E000 00 00 00 00 00 00 00 00 ........
0040E008 00 00 00 00 00 00 00 00 ........
0040E010 63 2E 77 6E 72 79 00 00 c.wnry..
0040E018 77 62 00 00 72 62 00 00 wb..rb..
0040E020 61 64 76 61 70 69 33 32 advapi32
0040E028 2E 64 6C 6C 00 00 00 00 .dll....
0040E030 77 64 00 00 57 00 61 00 wd..W.a.
0040E038 6E 00 61 00 43 00 72 00 n.a.C.r.
0040E040 79 00 70 00 74 00 30 00 y.p.t.0.
0040E048 72 00 00 00 53 00 6F 00 r...S.o.
0040E050 66 00 74 00 77 00 61 00 f.t.w.a.
0040E058 72 00 65 00 5C 00 00 00 r.e.\...
0040E060 70 EB 40 00 64 EB 40 00 pδ@.dδ@.
0040E068 58 EB 40 00 4C EB 40 00 Xδ@.Lδ@.
0040E070 40 EB 40 00 34 EB 40 00 @δ@.4δ@.
0040E078 28 EB 40 00 1C EB 40 00 (δ@.∟δ@.
0040E080 10 EB 40 00 04 EB 40 00 ►δ@.♦δ@.
0040E088 F8 EA 40 00 EC EA 40 00 °Ω@.∞Ω@.
0040E090 E0 EA 40 00 D4 EA 40 00 αΩ@.╘Ω@.
0040E098 C8 EA 40 00 BC EA 40 00 ╚Ω@.╝Ω@.
0040E0A0 B0 EA 40 00 A4 EA 40 00 ░Ω@.ñΩ@.
0040E0A8 98 EA 40 00 8C EA 40 00 ÿΩ@.îΩ@.
0040E0B0 80 EA 40 00 74 EA 40 00 ÇΩ@.tΩ@.
0040E0B8 68 EA 40 00 5C EA 40 00 hΩ@.\Ω@.
0040E0C0 50 EA 40 00 44 EA 40 00 PΩ@.DΩ@.
0040E0C8 38 EA 40 00 2C EA 40 00 8Ω@.,Ω@.
0040E0D0 20 EA 40 00 14 EA 40 00  Ω@.¶Ω@.
```

```
0012FFC4  7C817077 wpü¦ RETURN to kernel32.7C817077
0012FFC8  0069006E n.i. ed01ebfb.0069006E
0012FFCC  00790074 t.y.
0012FFD0  7FFDB000 .░²⌂
0012FFD4  80544CFD ²LTÇ
0012FFD8  0012FFC8 ╚ ↕.
0012FFDC  86521A78 x→Rå
0012FFE0  FFFFFFFF      End of SEH chain
0012FFE4  7C839AD8 ╪üâ¦ SE handler
0012FFE8  7C817080 Çpü¦ kernel32.7C817080
0012FFEC  00000000 ....
0012FFF0  00000000 ....
0012FFF4  00000000 ....
0012FFF8  004077BA ║w@. ed01ebfb.<ModuleEntryPoint>
0012FFFC  00000000 ....
```

We felt that immunity debugger could also be used as an alternative to Ollydbg due to it being able to colour code the assembly codes so it is much easier to use. It also has additional comments and can be used for exploit development and thus perhaps we can use this to give us an alternative perspective compared to Ollydbg.

General Analysis

What type of malware is it?

It is a ransomware that encrypts the user files. The files will be encrypted and the malware will demand a ransom of $300 worth of bitcoin for it to decrypt the files.

What are the functionalities of the malware?

The original malware first surfaced in 2017 as a result of the EternalBlue exploit. It was able to use the backdoor in the to be able to drop the main aspect of the ransomware which is tasksche.exe. The exe file is the one we will be analysing for this assignment (without the worm).

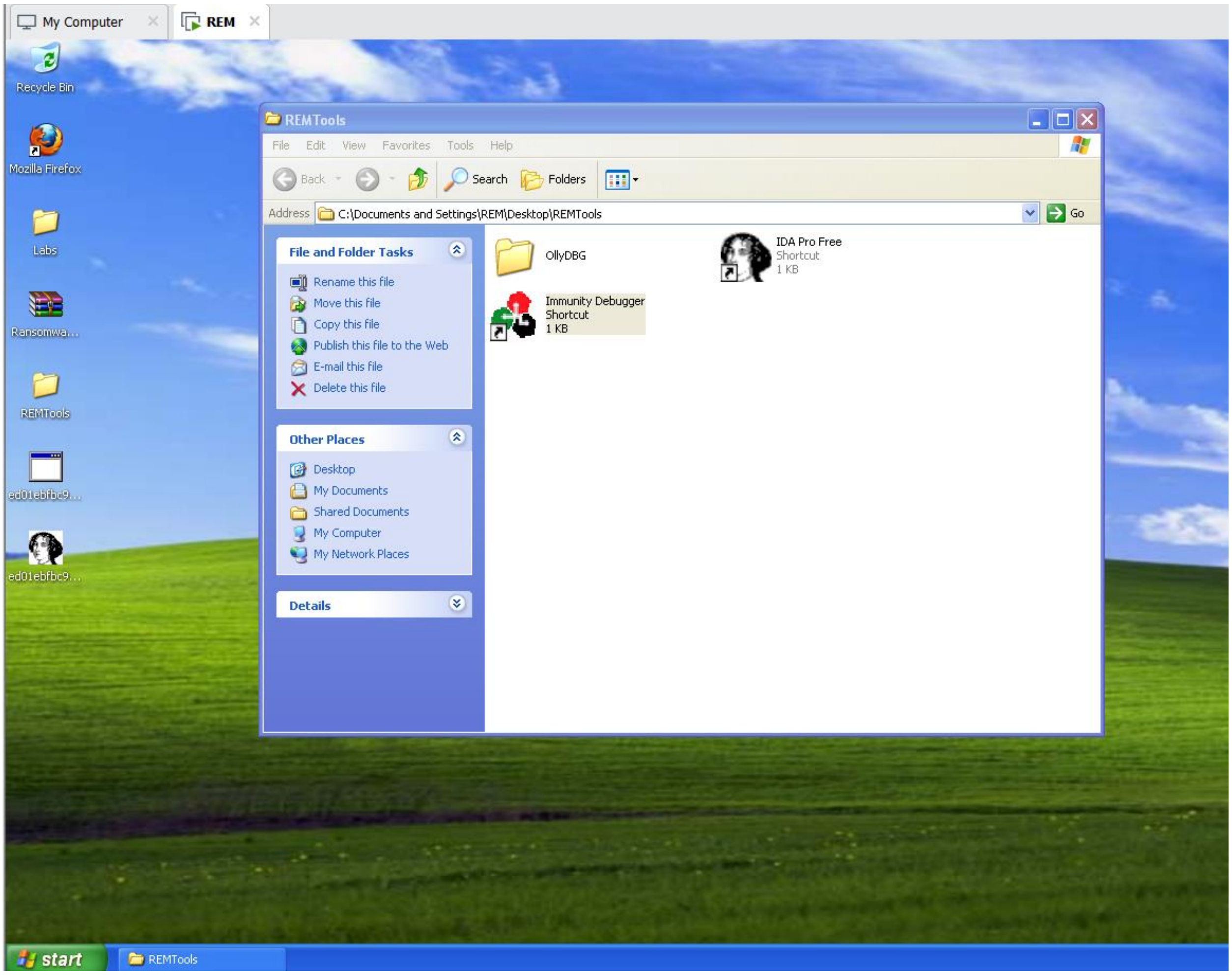
From there, the malware will be able to do the encryption of the files that are on the victim's machine. It also ensures that it only runs once by using the mutex API.

To ensure that users will not be infected by the malware, Microsoft Ms-1710 is patched to address SMB vulnerability exploited by the malware. Marcus Hutchins of MalwareTech registered the kill switch for the worm version. This ensures when the domain is registered the malware will not be able to run.

Were you able to interact with the malware? How?

We were able to interact with it through dynamic analysis by running it literally.

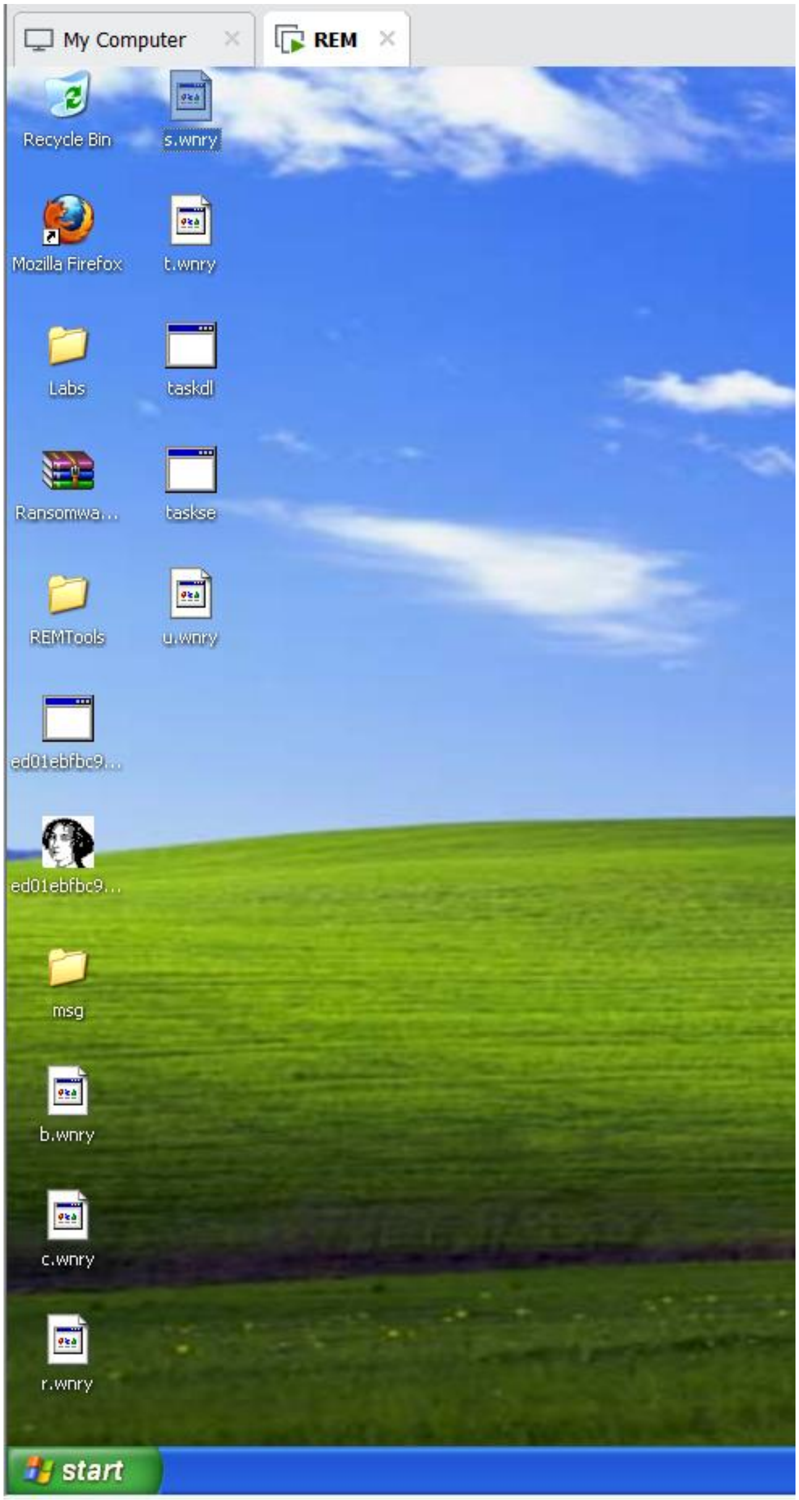
Before



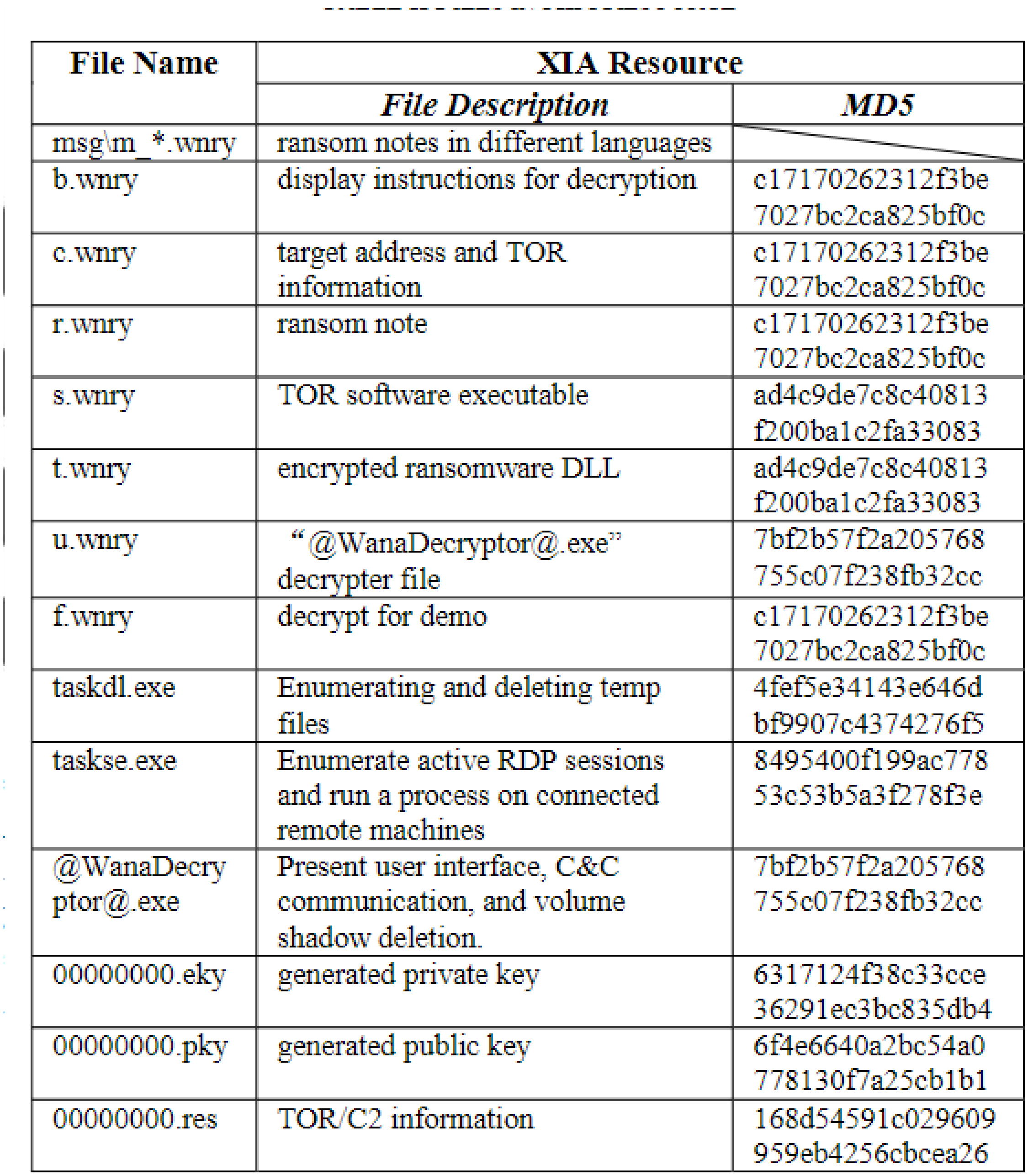
C:\Documents and Settings\REM\Desktop\REMTools

When the malware is run (i.e. double clicked), it first unzips the files that were in the resource zip file “XIA” as previously described in the major subroutine.

After:

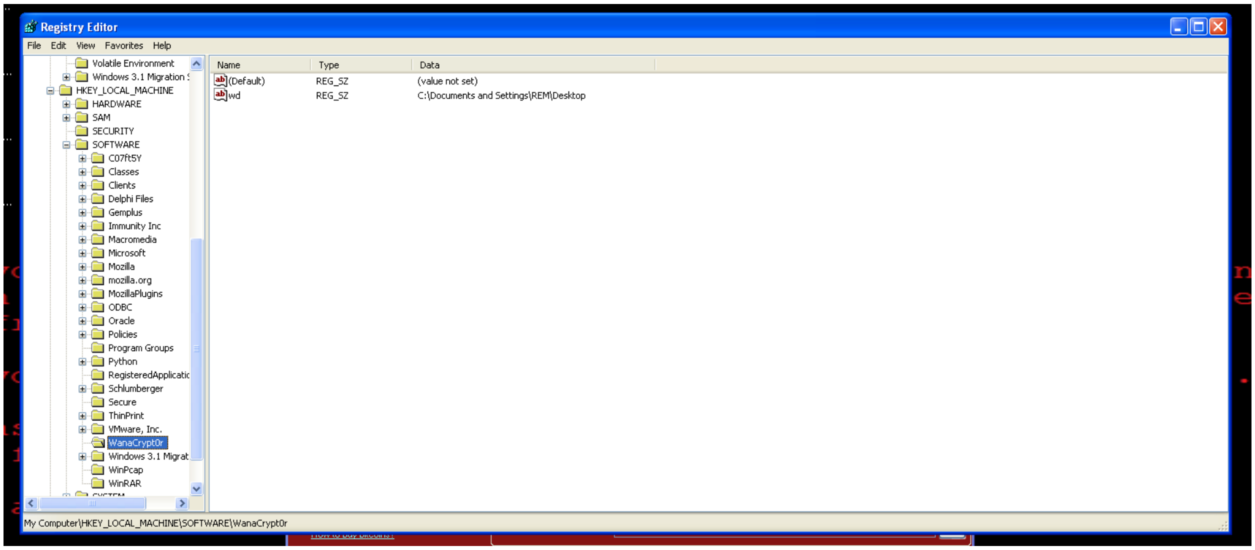


When the malware is first run, the wallpaper is still the default (bliss), although some additional programs are already loaded onto the victim’s machine



File Name	XIA Resource	
	File Description	*MD5*
msg\m_*.wnry	ransom notes in different languages	
b.wnry	display instructions for decryption	c17170262312f3be 7027bc2ca825bf0c
c.wnry	target address and TOR information	c17170262312f3be 7027bc2ca825bf0c
r.wnry	ransom note	c17170262312f3be 7027bc2ca825bf0c
s.wnry	TOR software executable	ad4c9de7c8c40813 f200ba1c2fa33083
t.wnry	encrypted ransomware DLL	ad4c9de7c8c40813 f200ba1c2fa33083
u.wnry	"@WanaDecryptor@.exe" decrypter file	7bf2b57f2a205768 755c07f238fb32cc
f.wnry	decrypt for demo	c17170262312f3be 7027bc2ca825bf0c
taskdl.exe	Enumerating and deleting temp files	4fef5e34143e646d bf9907c4374276f5
taskse.exe	Enumerate active RDP sessions and run a process on connected remote machines	8495400f199ac778 53c53b5a3f278f3e
@WanaDecryptor@.exe	Present user interface, C&C communication, and volume shadow deletion.	7bf2b57f2a205768 755c07f238fb32cc
00000000.eky	generated private key	6317124f38c33cce 36291ec3bc835db4
00000000.pky	generated public key	6f4e6640a2bc54a0 778130f7a25cb1b1
00000000.res	TOR/C2 information	168d54591c029609 959eb4256cbcea26

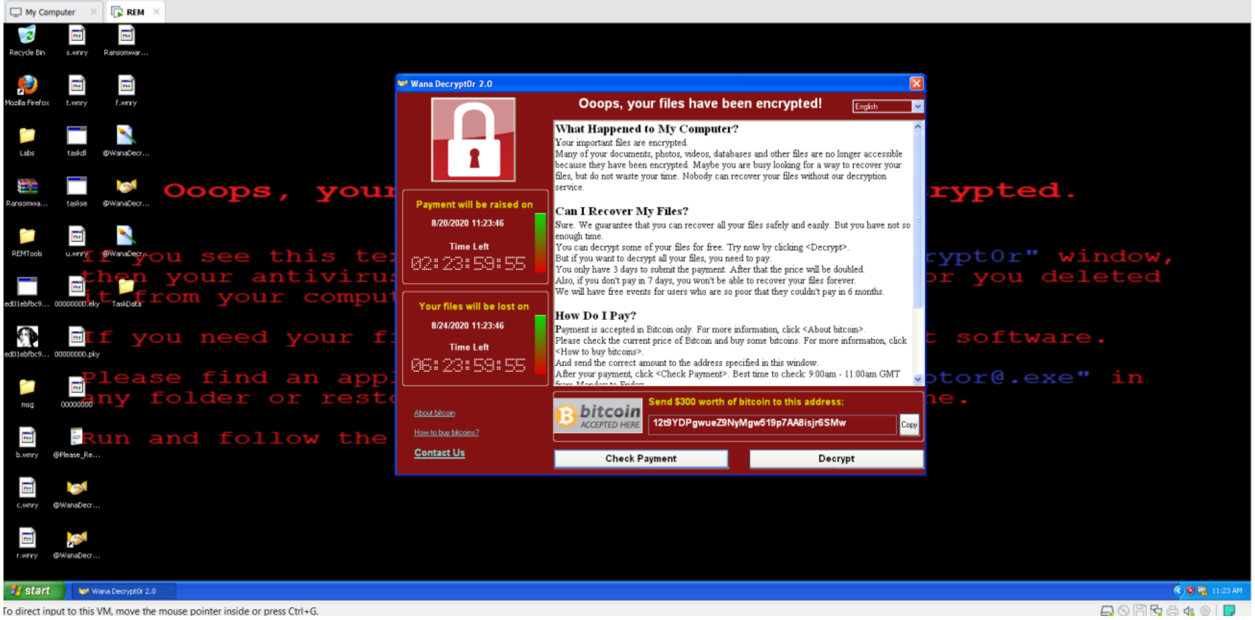
The programs have various purposes in order to demand the ransom, decrypt the information, send other details to the C2 Server (if this was encapsulated in the worm).



Name	Type	Data
(Default)	REG_SZ	(value not set)
wd	REG_SZ	C:\Documents and Settings\REM\Desktop

My Computer\HKEY_LOCAL_MACHINE\SOFTWARE\WanaCrypt0r

It also creates a registry key under Local Machine> Software> WannaCryptor as shown above.



Wana Decrypt0r 2.0

Ooops, your files have been encrypted!

English

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.
After your payment, click <Check Payment>. Best time to check: 9:00am - 11:00am GMT from Monday to Friday.

Payment will be raised on
8/20/2020 11:23:46
Time Left
02:23:59:55

Your files will be lost on
8/24/2020 11:23:46
Time Left
06:23:59:55

About bitcoin
How to buy bitcoins?
Contact Us

bitcoin ACCEPTED HERE

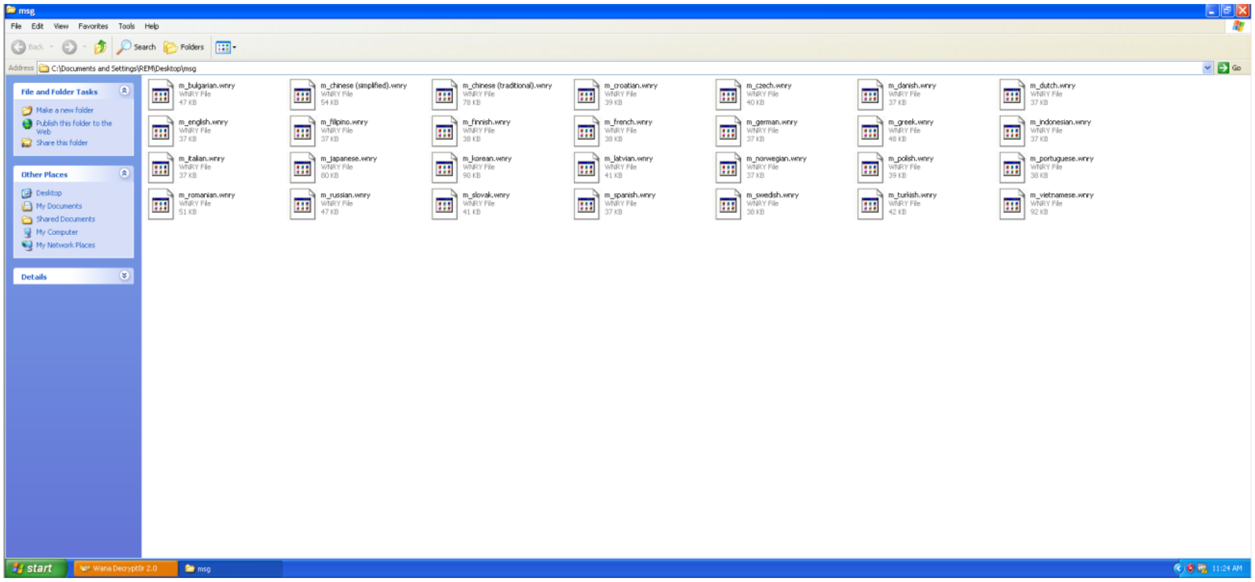
Send $300 worth of bitcoin to this address:
12t9YDPgwueZ9NyMgw519p7AA8isjr6SMw

Copy

Check Payment
Decrypt

To direct input to this VM, move the mouse pointer inside or press Ctrl+G.

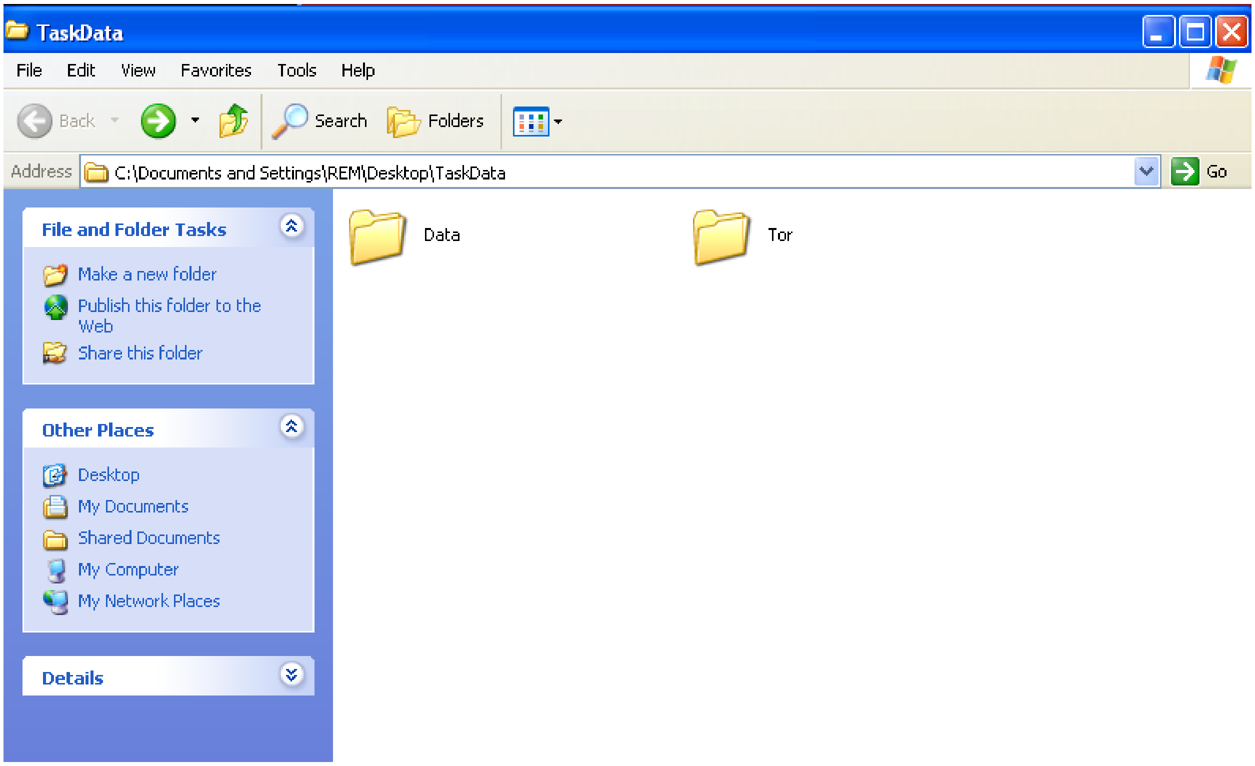
Thereafter, a default wallpaper (b.wnry) demanding the ransom is displayed and the wannacryptor application shows the ransom note as well as the bitcoin address to send the ransom to and this is persistent even if the victim tries to close the window (pop up after a few seconds).



Address C:\Documents and Settings\REM\Desktop\msg

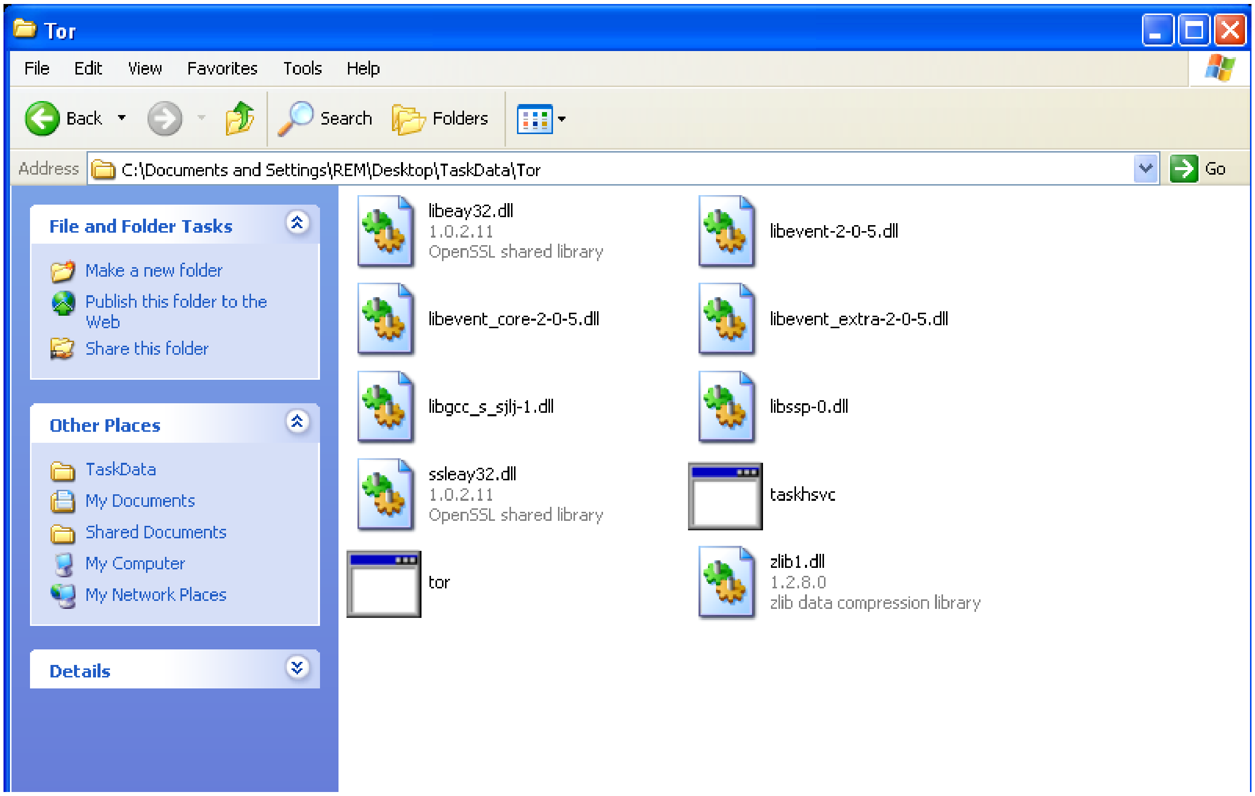
m_bulgarian.wnry WNRY File 47 KB	m_chinese (simplified).wnry WNRY File 54 KB	m_chinese (traditional).wnry WNRY File 78 KB	m_croatian.wnry WNRY File 39 KB	m_czech.wnry WNRY File 40 KB	m_danish.wnry WNRY File 37 KB	m_dutch.wnry WNRY File 37 KB
m_english.wnry WNRY File 37 KB	m_filipino.wnry WNRY File 37 KB	m_finnish.wnry WNRY File 38 KB	m_french.wnry WNRY File 38 KB	m_german.wnry WNRY File 37 KB	m_greek.wnry WNRY File 48 KB	m_indonesian.wnry WNRY File 37 KB
m_italian.wnry WNRY File 37 KB	m_japanese.wnry WNRY File 80 KB	m_korean.wnry WNRY File 90 KB	m_latvian.wnry WNRY File 41 KB	m_norwegian.wnry WNRY File 37 KB	m_polish.wnry WNRY File 39 KB	m_portuguese.wnry WNRY File 38 KB
m_romanian.wnry WNRY File 51 KB	m_russian.wnry WNRY File 47 KB	m_slovak.wnry WNRY File 41 KB	m_spanish.wnry WNRY File 37 KB	m_swedish.wnry WNRY File 38 KB	m_turkish.wnry WNRY File 42 KB	m_vietnamese.wnry WNRY File 92 KB

Messages will be loaded in the directory msg with the language is it in as the ransom notes as shown with the table above.



Address C:\Documents and Settings\REM\Desktop\TaskData

The taskdata folder contains the data (which will be populated if the victim's machine was connected to the Internet) but in our case it is not as described in the network diagram and the Tor folder which is elaborated on below.



C:\Documents and Settings\REM\Desktop\TaskData\Tor

libeay32.dll
1.0.2.11
OpenSSL shared library

libevent-2-0-5.dll

libevent_core-2-0-5.dll

libevent_extra-2-0-5.dll

libgcc_s_sjlj-1.dll

libssp-0.dll

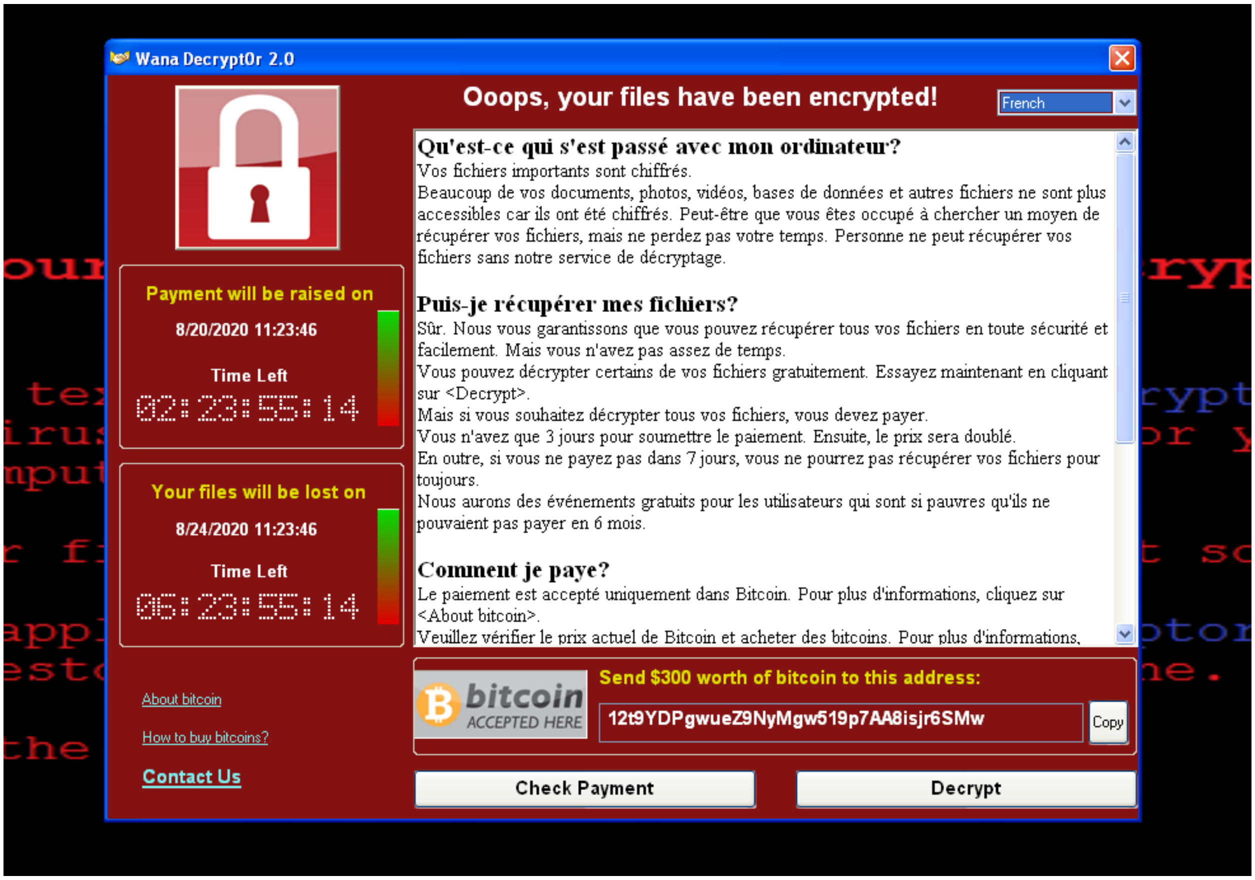
ssleay32.dll
1.0.2.11
OpenSSL shared library

taskhsvc

tor

zlib1.dll
1.2.8.0
zlib data compression library

The Tor directory contains files which enable the C2 server to communicate with the infected machine and further propagate the malware through the network.



Wana Decrypt0r 2.0

Ooops, your files have been encrypted!

French

Qu'est-ce qui s'est passé avec mon ordinateur?
Vos fichiers importants sont chiffrés.
Beaucoup de vos documents, photos, vidéos, bases de données et autres fichiers ne sont plus accessibles car ils ont été chiffrés. Peut-être que vous êtes occupé à chercher un moyen de récupérer vos fichiers, mais ne perdez pas votre temps. Personne ne peut récupérer vos fichiers sans notre service de décryptage.

Puis-je récupérer mes fichiers?
Sûr. Nous vous garantissons que vous pouvez récupérer tous vos fichiers en toute sécurité et facilement. Mais vous n'avez pas assez de temps.
Vous pouvez décrypter certains de vos fichiers gratuitement. Essayez maintenant en cliquant sur <Decrypt>.
Mais si vous souhaitez décrypter tous vos fichiers, vous devez payer.
Vous n'avez que 3 jours pour soumettre le paiement. Ensuite, le prix sera doublé.
En outre, si vous ne payez pas dans 7 jours, vous ne pourrez pas récupérer vos fichiers pour toujours.
Nous aurons des événements gratuits pour les utilisateurs qui sont si pauvres qu'ils ne pouvaient pas payer en 6 mois.

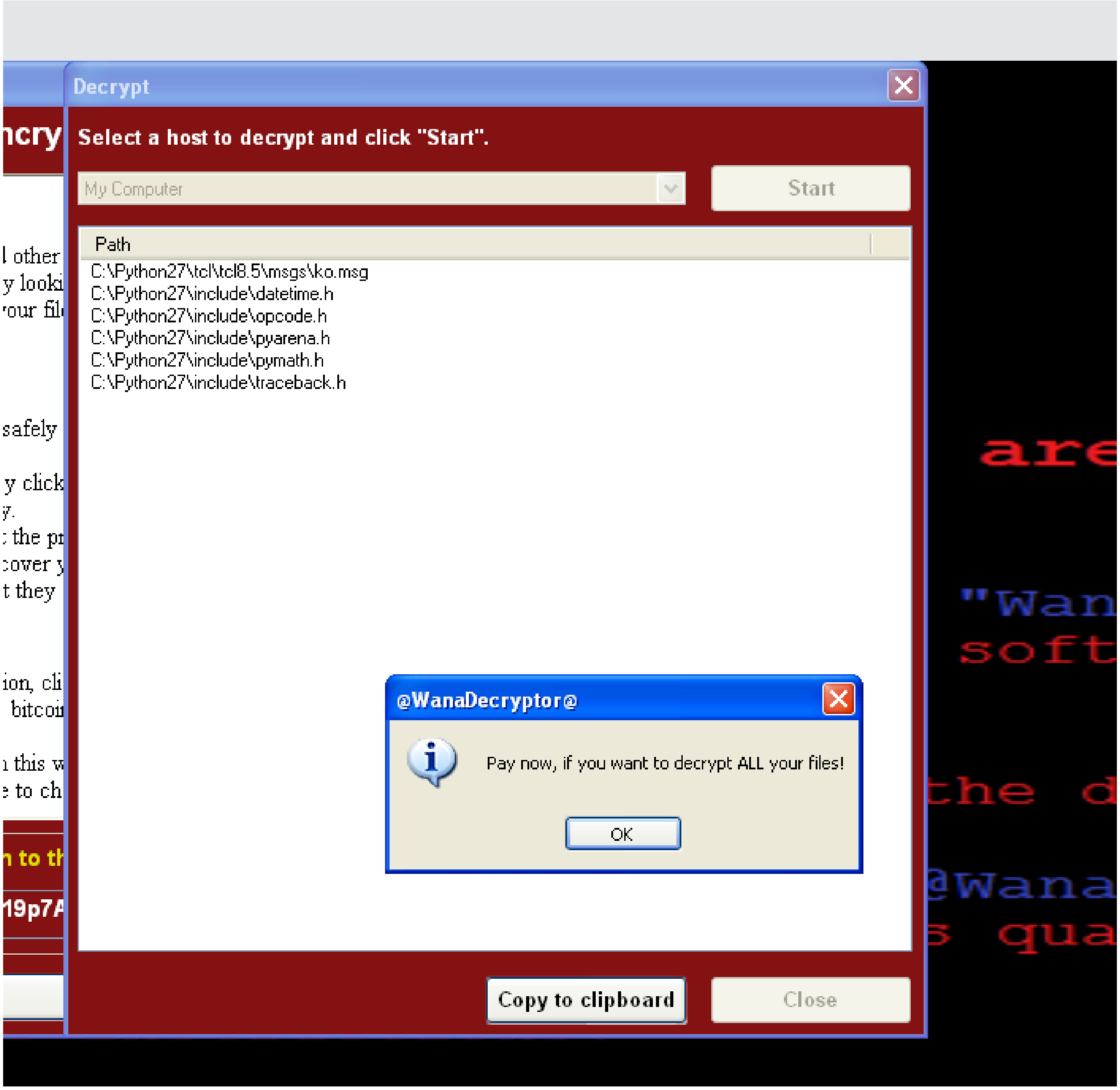
Comment je paye?
Le paiement est accepté uniquement dans Bitcoin. Pour plus d'informations, cliquez sur <About bitcoin>.
Veuillez vérifier le prix actuel de Bitcoin et acheter des bitcoins. Pour plus d'informations,

Payment will be raised on
8/20/2020 11:23:46
Time Left
02:23:55:14

Your files will be lost on
8/24/2020 11:23:46
Time Left
06:23:55:14

Send $300 worth of bitcoin to this address:
12t9YDPgwueZ9NyMgw519p7AA8isjr6SMw

Additional features of the malware include being able to change the language of the popup and being able to Decrypt some of the files for free but still require the payment for $300 of bitcoin for it to decrypt all the files. It is able to decrypt some python files for free.



Select a host to decrypt and click "Start".

```
C:\Python27\tcl\tcl8.5\msgs\ko.msg
C:\Python27\include\datetime.h
C:\Python27\include\opcode.h
C:\Python27\include\pyarena.h
C:\Python27\include\pymath.h
C:\Python27\include\traceback.h
```

Pay now, if you want to decrypt ALL your files!

References

https://techtalk.pcmatic.com/2017/05/16/wanacrypt0r-dive-code/

https://www.coursehero.com/file/33565476/20180369-finalpaperpdf/