Dillon Prendergast

CAP5137

Professor Liu

14 December 2018

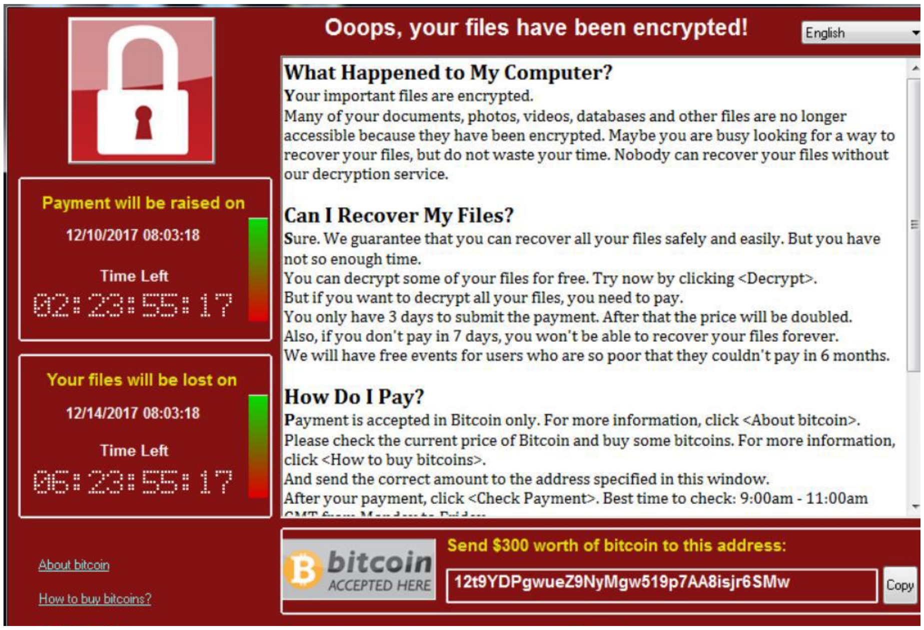
Hands-On Project Report

~~For my hands-on project I will be analyzing the ransomware malware found at http://www.cs.fsu.edu/~liux/courses/reversing/assignments/malware/ransomware.zip. This is a ransomware type of malware and it is recognized in the VirusTotal database~~.

For greater accessibility and interest I will be analyzing a piece of WannaCry ransomware, WanaCrypt 2.0.

A ransomware is a type of malware that will encrypt the victim's files making them inaccessible without paying a ransom, usually in the form of bitcoin. As such, I expect to be able to identify how the ransomware can access the victim's files and overwrite them with the encryption. I also hope to be able to identify the encryption algorithm that is being used for the malware and possibly reverse it.

To protect my own machine, I will be analyzing this malware in a VirtualBox Windows 7 machine that I have set up.



Ooops, your files have been encrypted!

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

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

How Do I Pay?
Payment is accepted in Bitcoin only. For more information, click <About bitcoin>.
Please check the current price of Bitcoin and buy some bitcoins. For more information, click <How to buy bitcoins>.
And send the correct amount to the address specified in this window.
After your payment, click <Check Payment>. Best time to check: 9:00am - 11:00am

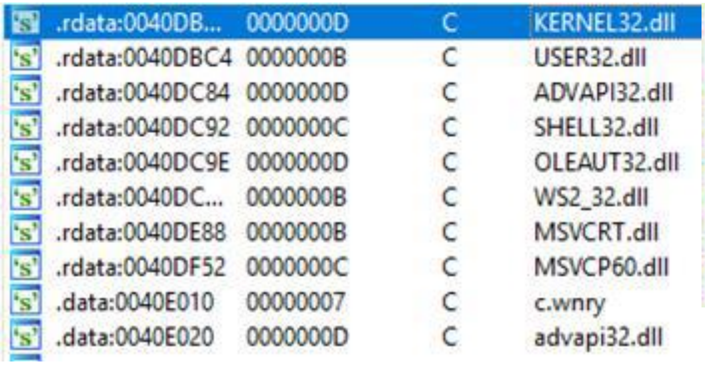
Payment will be raised on
12/10/2017 08:03:18
Time Left
02:23:55:17

Your files will be lost on
12/14/2017 08:03:18
Time Left
06:23:55:17

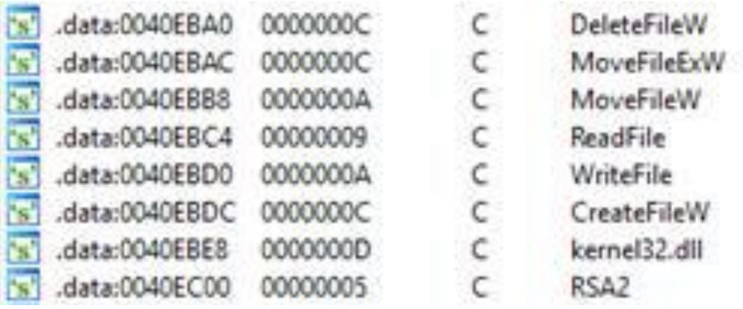
Send $300 worth of bitcoin to this address:
12t9YDPgwueZ9NyMgw519p7AA8isjr6SMw

When the malware is executed, the screen quickly becomes locked behind this paywall.

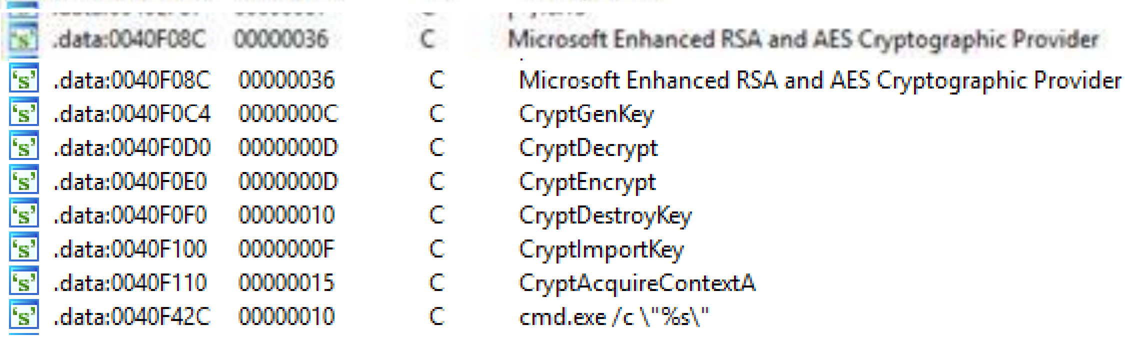
After resetting my VM, I open the executable file into IDA.



.rdata:0040DB...	0000000D	C	KERNEL32.dll
.rdata:0040DBC4	0000000B	C	USER32.dll
.rdata:0040DC84	0000000D	C	ADVAPI32.dll
.rdata:0040DC92	0000000C	C	SHELL32.dll
.rdata:0040DC9E	0000000D	C	OLEAUT32.dll
.rdata:0040DC...	0000000B	C	WS2_32.dll
.rdata:0040DE88	0000000B	C	MSVCRT.dll
.rdata:0040DF52	0000000C	C	MSVCP60.dll
.data:0040E010	00000007	C	c.wnry
.data:0040E020	0000000D	C	advapi32.dll

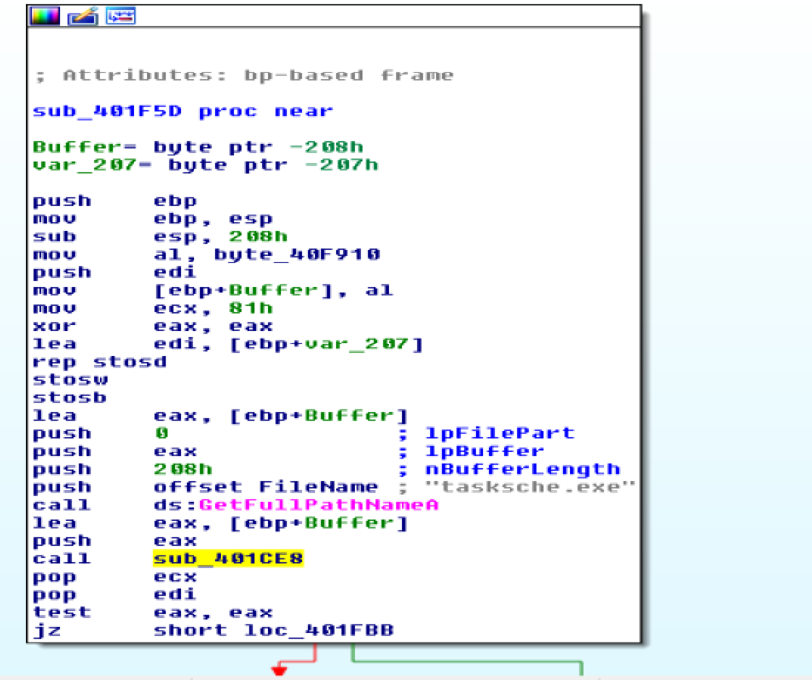


.data:0040EBA0	0000000C	C	DeleteFileW
.data:0040EBAC	0000000C	C	MoveFileExW
.data:0040EBB8	0000000A	C	MoveFileW
.data:0040EBC4	00000009	C	ReadFile
.data:0040EBD0	0000000A	C	WriteFile
.data:0040EBDC	0000000C	C	CreateFileW
.data:0040EBE8	0000000D	C	kernel32.dll
.data:0040EC00	00000005	C	RSA2



.data:0040F08C	00000036	C	Microsoft Enhanced RSA and AES Cryptographic Provider
.data:0040F08C	00000036	C	Microsoft Enhanced RSA and AES Cryptographic Provider
.data:0040F0C4	0000000C	C	CryptGenKey
.data:0040F0D0	0000000D	C	CryptDecrypt
.data:0040F0E0	0000000D	C	CryptEncrypt
.data:0040F0F0	00000010	C	CryptDestroyKey
.data:0040F100	0000000F	C	CryptImportKey
.data:0040F110	00000015	C	CryptAcquireContextA
.data:0040F42C	00000010	C	cmd.exe /c \"%s\"

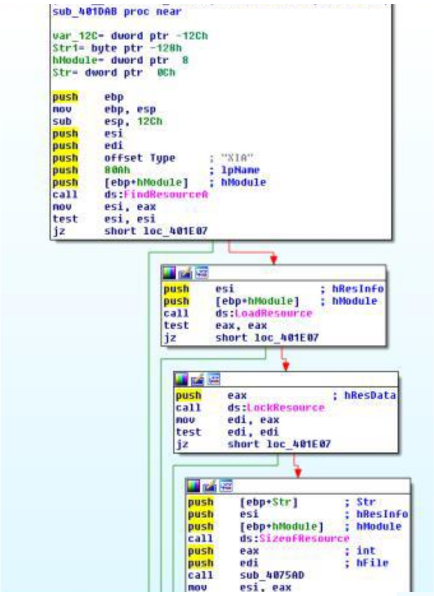
I started out with the Strings View, where I found loads of potentially useful things to look for. Specifically, I saw multiple *.dll* files that will likely be created/called and additional file manipulation commands. I also saw RSA and AES encryption mentioned, which will likely be used for the actual ransomware file encryption.



```
; Attributes: bp-based frame

sub_401F5D proc near

Buffer= byte ptr -208h
var_207= byte ptr -207h

push    ebp
mov     ebp, esp
sub     esp, 208h
mov     al, byte_40F910
push    edi
mov     [ebp+Buffer], al
mov     ecx, 81h
xor     eax, eax
lea     edi, [ebp+var_207]
rep stosd
stosw
stosb
lea     eax, [ebp+Buffer]
push    0                   ; lpFilePart
push    eax                 ; lpBuffer
push    208h                ; nBufferLength
push    offset FileName     ; "tasksche.exe"
call    ds:GetFullPathNameA
lea     eax, [ebp+Buffer]
push    eax
call    sub_401CE8
pop     ecx
pop     edi
test    eax, eax
jz      short loc_401FBB
```

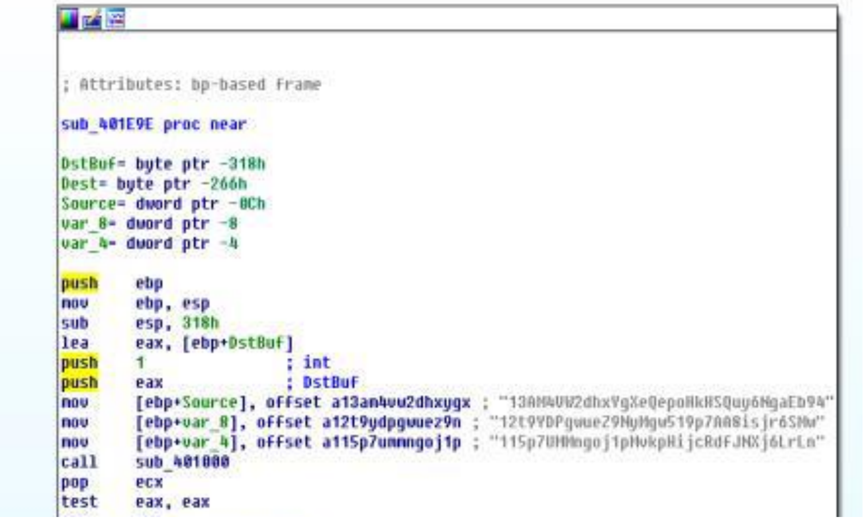
The program begins by loading a filename from *byte_40F910* into *al*. From there it is moved into *eax* for use in *ds:GetModuleFileNameA* along with an offset for *tasksch.exe*.

The return of this is a filename which is then used in *sub_401CE8.* This sub uses the string in conjunction with *‘cmd.exe /C %s’* to launce a service under *tasksche.exe.*



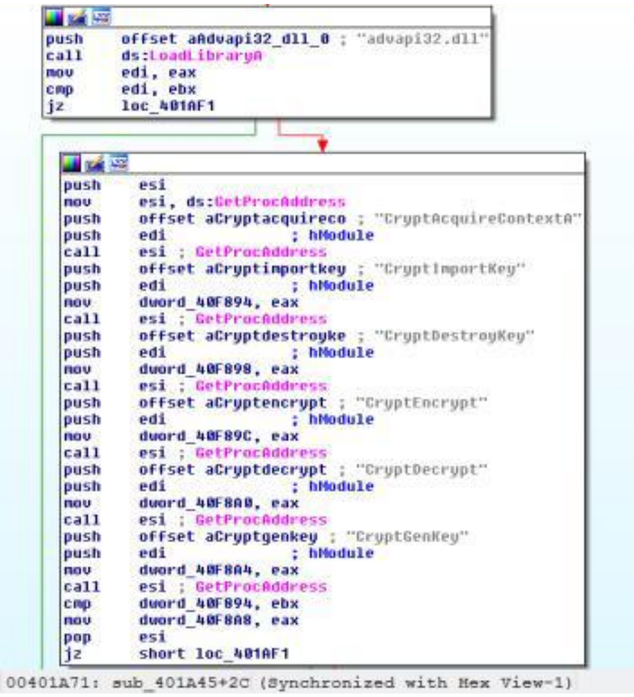
```
sub_401DAB proc near

var_12C= dword ptr -12Ch
Str1= byte ptr -128h
hModule= dword ptr  8
Str= dword ptr  0Ch

push    ebp
mov     ebp, esp
sub     esp, 12Ch
push    esi
push    edi
push    offset Type     ; "XIA"
push    80Ah            ; lpName
push    [ebp+hModule]   ; hModule
call    ds:FindResourceA
mov     esi, eax
test    esi, esi
jz      short loc_401E07

push    esi             ; hResInfo
push    [ebp+hModule]   ; hModule
call    ds:LoadResource
test    eax, eax
jz      short loc_401E07

push    eax             ; hResData
call    ds:LockResource
mov     edi, eax
test    edi, edi
jz      short loc_401E07

push    [ebp+Str]       ; Str
push    esi             ; hResInfo
push    [ebp+hModule]   ; hModule
call    ds:SizeofResource
push    eax             ; int
push    edi             ; hFile
call    sub_4075AD
mov     esi, eax
```

Also in sub_401DAB, an XIA file is unzipped with 'WNcry@20l7'. Which contains config files for wncry.



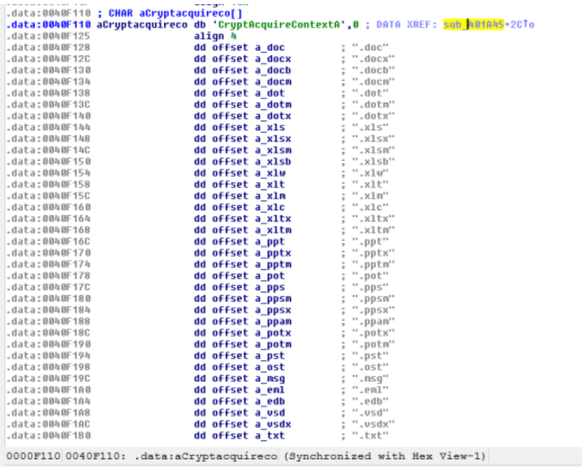
```
; Attributes: bp-based frame

sub_401E9E proc near

DstBuf= byte ptr -318h
Dest= byte ptr -266h
Source= dword ptr -0Ch
var_8= dword ptr -8
var_4= dword ptr -4

push    ebp
mov     ebp, esp
sub     esp, 318h
lea     eax, [ebp+DstBuf]
push    1               ; int
push    eax             ; DstBuf
mov     [ebp+Source], offset a13an4vu2dhxygx ; "13AM4VW2dhxYgXeQepoHkHSQuy6NgaEb94"
mov     [ebp+var_8], offset a12t9ydpgwuez9n ; "12t9YDPgwueZ9NyMgw519p7AA8isjr6SMw"
mov     [ebp+var_4], offset a115p7ummngoj1p ; "115p7UMMngoj1pMvkpHijcRdfJNXj6LrLn"
call    sub_401000
pop     ecx
test    eax, eax
```

From there we go to *sub_401E9E*, where we can see the bitcoin wallet address that was referenced in the lock screen that we initially saw.



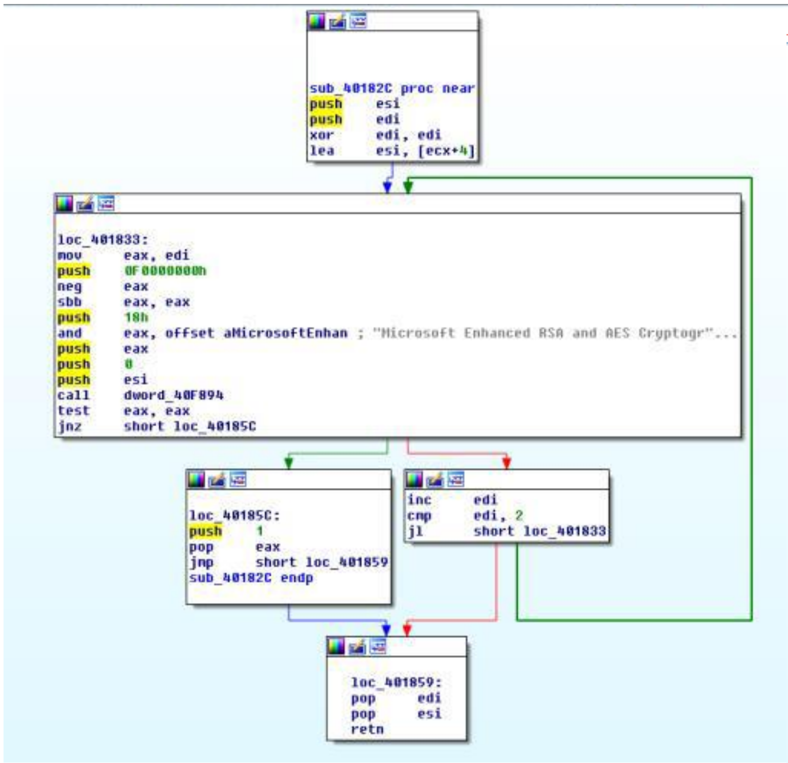
```
push    offset aAdvapi32_dll_0 ; "advapi32.dll"
call    ds:LoadLibraryA
mov     edi, eax
cmp     edi, ebx
jz      loc_401AF1

push    esi
mov     esi, ds:GetProcAddress
push    offset aCryptacquireco ; "CryptAcquireContextA"
push    edi             ; hModule
call    esi ; GetProcAddress
push    offset aCryptimportkey ; "CryptImportKey"
push    edi             ; hModule
mov     dword_40F894, eax
call    esi ; GetProcAddress
push    offset aCryptdestroyke ; "CryptDestroyKey"
push    edi             ; hModule
mov     dword_40F898, eax
call    esi ; GetProcAddress
push    offset aCryptencrypt ; "CryptEncrypt"
push    edi             ; hModule
mov     dword_40F89C, eax
call    esi ; GetProcAddress
push    offset aCryptdecrypt ; "CryptDecrypt"
push    edi             ; hModule
mov     dword_40F8A0, eax
call    esi ; GetProcAddress
push    offset aCryptgenkey ; "CryptGenKey"
push    edi             ; hModule
mov     dword_40F8A4, eax
call    esi ; GetProcAddress
cmp     dword_40F894, ebx
mov     dword_40F8A8, eax
pop     esi
jz      short loc_401AF1
00401A71: sub_401A45+2C (Synchronized with Hex View-1)
```

We see *sub_401000* being called on two separate occasions with different parameters. The first one uses '*Attrb +h'* and then it uses *'lcacls ./grant Everyone:F /T /C /Q'*. These parameters are used with the discretionary access controls for files, which can grant '*Everyone'* access to the files.

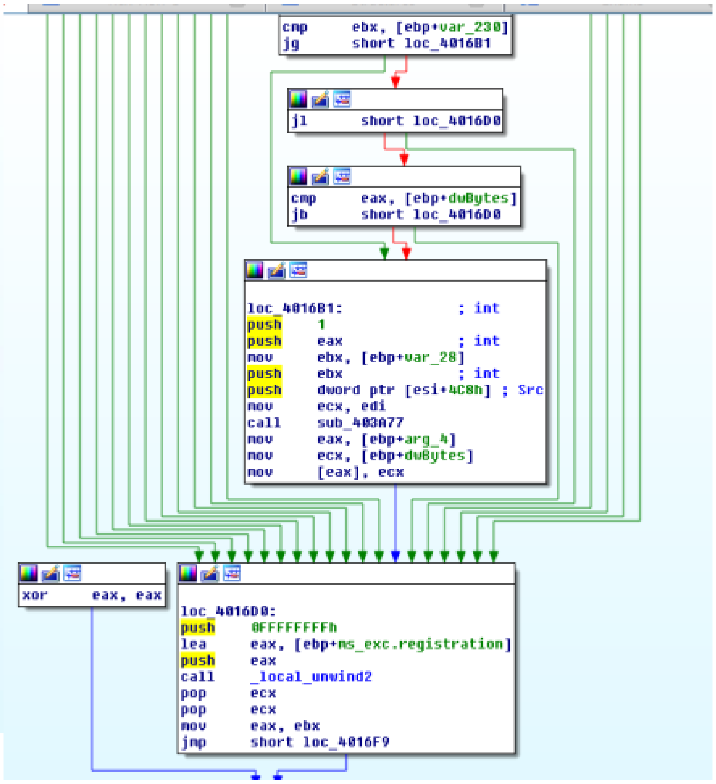


```
.data:0040F110 ; CHAR aCryptacquireco[]
.data:0040F110 aCryptacquireco db 'CryptAcquireContextA',0 ; DATA XREF: sub_401A45+2C↑o
.data:0040F125                 align 4
.data:0040F128                 dd offset a_doc         ; ".doc"
.data:0040F12C                 dd offset a_docx        ; ".docx"
.data:0040F130                 dd offset a_docb        ; ".docb"
.data:0040F134                 dd offset a_docm        ; ".docm"
.data:0040F138                 dd offset a_dot         ; ".dot"
.data:0040F13C                 dd offset a_dotm        ; ".dotm"
.data:0040F140                 dd offset a_dotx        ; ".dotx"
.data:0040F144                 dd offset a_xls         ; ".xls"
.data:0040F148                 dd offset a_xlsx        ; ".xlsx"
.data:0040F14C                 dd offset a_xlsm        ; ".xlsm"
.data:0040F150                 dd offset a_xlsb        ; ".xlsb"
.data:0040F154                 dd offset a_xlw         ; ".xlw"
.data:0040F158                 dd offset a_xlt         ; ".xlt"
.data:0040F15C                 dd offset a_xlm         ; ".xlm"
.data:0040F160                 dd offset a_xlc         ; ".xlc"
.data:0040F164                 dd offset a_xltx        ; ".xltx"
.data:0040F168                 dd offset a_xltm        ; ".xltm"
.data:0040F16C                 dd offset a_ppt         ; ".ppt"
.data:0040F170                 dd offset a_pptx        ; ".pptx"
.data:0040F174                 dd offset a_pptm        ; ".pptm"
.data:0040F178                 dd offset a_pot         ; ".pot"
.data:0040F17C                 dd offset a_pps         ; ".pps"
.data:0040F180                 dd offset a_ppsm        ; ".ppsm"
.data:0040F184                 dd offset a_ppsx        ; ".ppsx"
.data:0040F188                 dd offset a_ppam        ; ".ppam"
.data:0040F18C                 dd offset a_potx        ; ".potx"
.data:0040F190                 dd offset a_potm        ; ".potm"
.data:0040F194                 dd offset a_pst         ; ".pst"
.data:0040F198                 dd offset a_ost         ; ".ost"
.data:0040F19C                 dd offset a_msg         ; ".msg"
.data:0040F1A0                 dd offset a_eml         ; ".eml"
.data:0040F1A4                 dd offset a_edb         ; ".edb"
.data:0040F1A8                 dd offset a_vsd         ; ".vsd"
.data:0040F1AC                 dd offset a_vsdx        ; ".vsdx"
.data:0040F1B0                 dd offset a_txt         ; ".txt"
0000F110 0040F110: .data:aCryptacquireco (Synchronized with Hex View-1)
```

After this we see '*Crypt Decrypt', 'Crypt Acquire' and 'CryptImportKey'* being used, which when we look further into *'CryptAcquireContext'* we see a list of file types that the malware will look to encrypt.

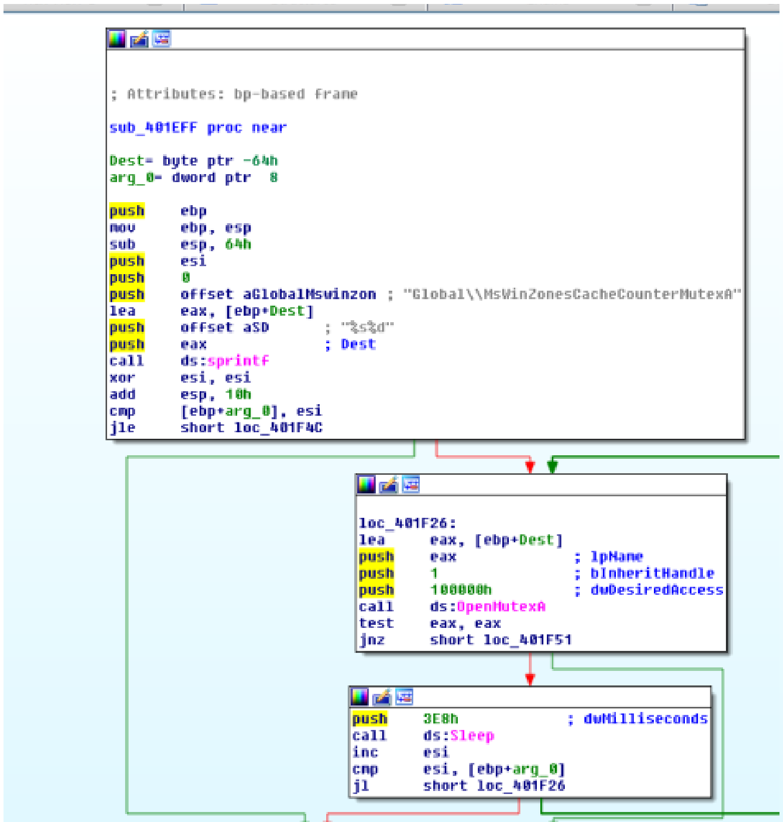


```
sub_40182C proc near
push    esi
push    edi
xor     edi, edi
lea     esi, [ecx+4]

loc_401833:
mov     eax, edi
push    0F0000000h
neg     eax
sbb     eax, eax
push    18h
and     eax, offset aMicrosoftEnhan ; "Microsoft Enhanced RSA and AES Cryptogr"...
push    eax
push    0
push    esi
call    dword_40F894
test    eax, eax
jnz     short loc_40185C

loc_40185C:
push    1
pop     eax
jmp     short loc_401859
sub_40182C endp

inc     edi
cmp     edi, 2
jl      short loc_401833

loc_401859:
pop     edi
pop     esi
retn
```



```
cmp     ebx, [ebp+var_230]
jg      short loc_4016B1

jl      short loc_4016D0

cmp     eax, [ebp+dwBytes]
jb      short loc_4016D0

loc_4016B1:                 ; int
push    1
push    eax                 ; int
mov     ebx, [ebp+var_28]
push    ebx                 ; int
push    dword ptr [esi+4C8h] ; Src
mov     ecx, edi
call    sub_403A77
mov     eax, [ebp+arg_4]
mov     ecx, [ebp+dwBytes]
mov     [eax], ecx

xor     eax, eax

loc_4016D0:
push    0FFFFFFFFh
lea     eax, [ebp+ms_exc.registration]
push    eax
call    _local_unwind2
pop     ecx
pop     ecx
mov     eax, ebx
jmp     short loc_4016F9
```

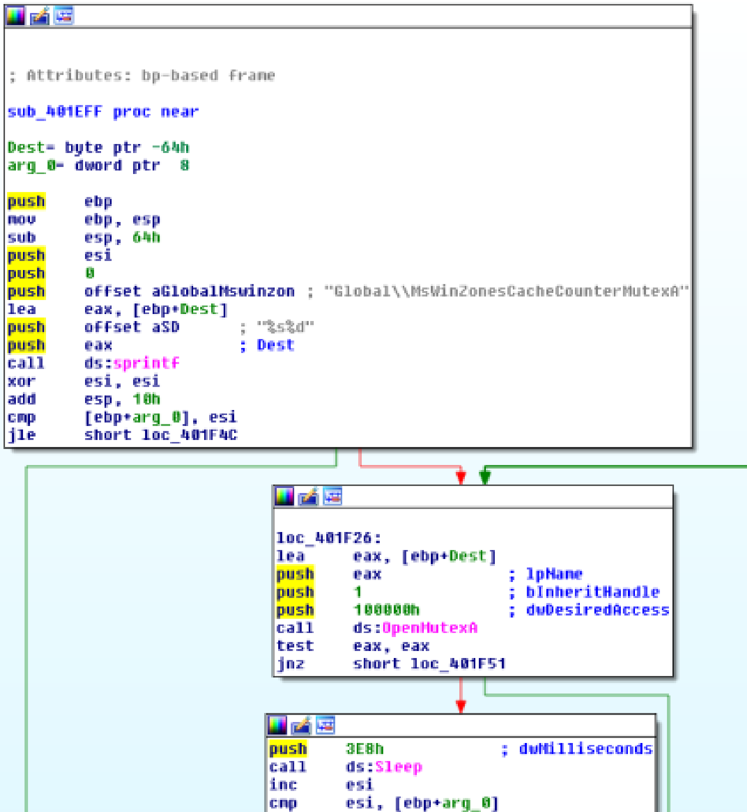
Immediately after, in *sub_40182C* the keys are created for AES encryption, and then the AES keys are got for the files in *sub_4014A6.*



```
; Attributes: bp-based frame

sub_401EFF proc near

Dest= byte ptr -64h
arg_0= dword ptr  8

push    ebp
mov     ebp, esp
sub     esp, 64h
push    esi
push    0
push    offset aGlobalMswinzon ; "Global\\MsWinZonesCacheCounterMutexA"
lea     eax, [ebp+Dest]
push    offset aSD      ; "%s%d"
push    eax             ; Dest
call    ds:sprintf
xor     esi, esi
add     esp, 10h
cmp     [ebp+arg_0], esi
jle     short loc_401F4C
```

```
loc_401F26:
lea     eax, [ebp+Dest]
push    eax             ; lpName
push    1               ; bInheritHandle
push    100000h         ; dwDesiredAccess
call    ds:OpenMutexA
test    eax, eax
jnz     short loc_401F51
```

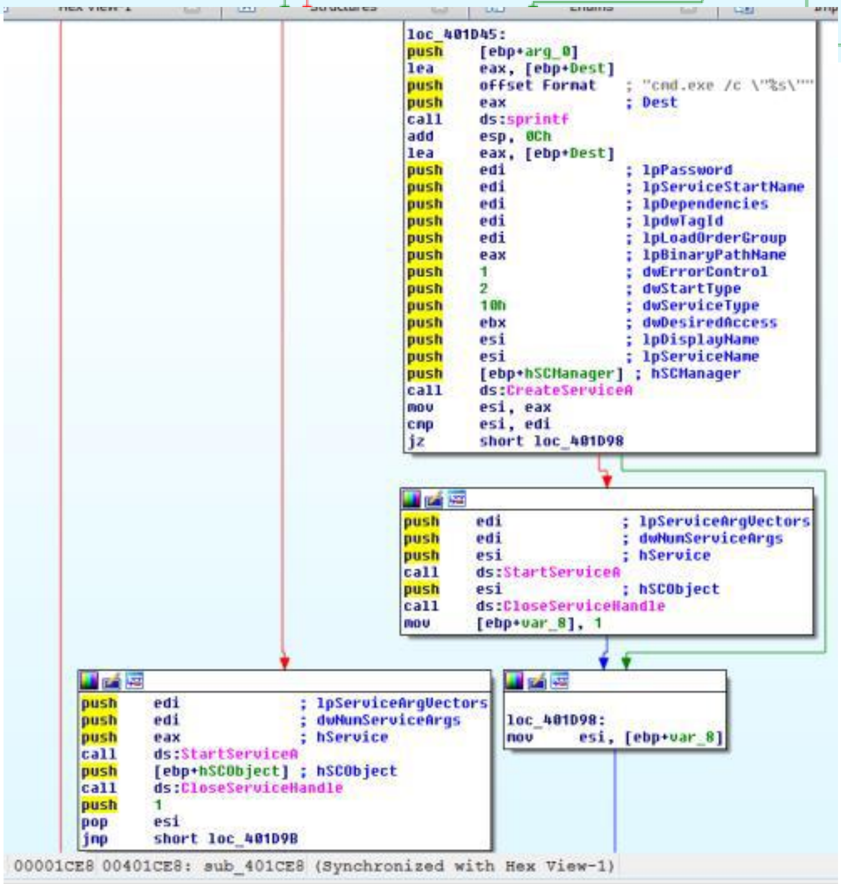
```
push    3E8h            ; dwMilliseconds
call    ds:Sleep
inc     esi
cmp     esi, [ebp+arg_0]
jl      short loc_401F26
```



```
; Attributes: bp-based frame

sub_401EFF proc near

Dest= byte ptr -64h
arg_0= dword ptr  8

push    ebp
mov     ebp, esp
sub     esp, 64h
push    esi
push    0
push    offset aGlobalMswinzon ; "Global\\MsWinZonesCacheCounterMutexA"
lea     eax, [ebp+Dest]
push    offset aSD      ; "%s%d"
push    eax             ; Dest
call    ds:sprintf
xor     esi, esi
add     esp, 10h
cmp     [ebp+arg_0], esi
jle     short loc_401F4C
```

```
loc_401F26:
lea     eax, [ebp+Dest]
push    eax             ; lpName
push    1               ; bInheritHandle
push    100000h         ; dwDesiredAccess
call    ds:OpenMutexA
test    eax, eax
jnz     short loc_401F51
```

```
push    3E8h            ; dwMilliseconds
call    ds:Sleep
inc     esi
cmp     esi, [ebp+arg_0]
jl      short loc_401F26
```



```
loc_401D45:
push    [ebp+arg_0]
lea     eax, [ebp+Dest]
push    offset Format   ; "cmd.exe /c \"%s\""
push    eax             ; Dest
call    ds:sprintf
add     esp, 0Ch
lea     eax, [ebp+Dest]
push    edi             ; lpPassword
push    edi             ; lpServiceStartName
push    edi             ; lpDependencies
push    edi             ; lpdwTagId
push    edi             ; lpLoadOrderGroup
push    eax             ; lpBinaryPathName
push    1               ; dwErrorControl
push    2               ; dwStartType
push    10h             ; dwServiceType
push    ebx             ; dwDesiredAccess
push    esi             ; lpDisplayName
push    esi             ; lpServiceName
push    [ebp+hSCManager] ; hSCManager
call    ds:CreateServiceA
mov     esi, eax
cmp     esi, edi
jz      short loc_401D98
```

```
push    edi             ; lpServiceArgVectors
push    edi             ; dwNumServiceArgs
push    esi             ; hService
call    ds:StartServiceA
push    esi             ; hSCObject
call    ds:CloseServiceHandle
mov     [ebp+var_8], 1
```

```
push    edi             ; lpServiceArgVectors
push    edi             ; dwNumServiceArgs
push    eax             ; hService
call    ds:StartServiceA
push    [ebp+hSCObject] ; hSCObject
call    ds:CloseServiceHandle
push    1
pop     esi
jmp     short loc_401D9B
```

```
loc_401D98:
mov     esi, [ebp+var_8]
```

```
00001CE8 00401CE8: sub_401CE8 (Synchronized with Hex View-1)
```

In *sub_401EFF* a mutex is created for the threads that will be encrypting files. Finally, back in *sub_401CE8* the encrypted files overwrite the unencrypted files and the machine is infected.