

WANNACRY MALWARE REVERSE ENGINEERING



BY EMMANUEL WILLIAM FRIMPONG

DATE: 12/7/2023

This analysis was done from the entry point of the exe file downwards to the end of the graph flow. The analysis was carried out in the IDA pro program.

- **GetModuleFileNameA**

Retrieves the fully qualified path for the file that contains the specified module.

The statement "Retrieves the fully qualified path for the file that contains the specified module" refers to the function of retrieving the complete file path of a specific module (like a DLL or an EXE file) in a system.

Parameters

nSize – 208h – the size of the buffer to receive the module

lpFilename - The full path of the module to be retrieved

hModule – handle to the module that is to be retrieved

```
mov     ecx, 81h
xor     eax, eax
lea     edi, [ebp+var_208]
rep stosd
stosw
stosb
lea     eax, [ebp+Filename]
push   208h          ; nSize
xor     ebx, ebx
push   eax          ; lpFilename
push   ebx          ; hModule
call   ds:GetModuleFileNameA
push   offset DisplayName
call   sub_401225
pop    ecx
call   ds:___p__argc
cmp    dword ptr [eax], 2
jnz    short loc_40208E
```

Explanation: The statement "Retrieves the fully qualified path for the file that contains the specified module," refers to the function that retrieves the complete file path of a specific module (like a DLL or an EXE file) in a system.

- **CopyFileA**

Copies an existing file to a new file

Parameters

lpNewFileName - The name of the new file.

lpExistingFileName - The name of an existing file.

bFailIfExists - f this parameter is TRUE and the new file specified by lpNewFileName already exists, the function fails. If this parameter is FALSE and the new file already exists, the function overwrites the existing file and succeeds.

```
SHORT loc_40200E
mov     esi, offset FileName ; "tasksche.exe"
push   ebx                  ; bFailIfExists
lea    eax, [ebp+Filename]
push   esi                  ; lpNewFileName
push   eax                  ; lpExistingFileName
call   ds:CopyFileA
push   esi                  ; lpFileName
call   ds:GetFileAttributesA
cmp    eax, 0FFFFFFFh
jz     short loc_40208E

call   sub_401F5D
```

Explanation: This function seems to be copying an old file and renaming it into “tasksche.exe”.

- **GetFileAttributesA**

Retrieves file system attributes for a specified file or directory.

Attributes of the File

- 1.Name. Every file carries a name by which the file is recognized in the file system. ...
- Identifier. Along with the name, Each File has its own extension which identifies the type of the file. ...
- Type. ...
- Location. ...
- Size. ...
- Protection. ...
- Time and Date.

Parameters

lpFileName - The name of the file or directory.

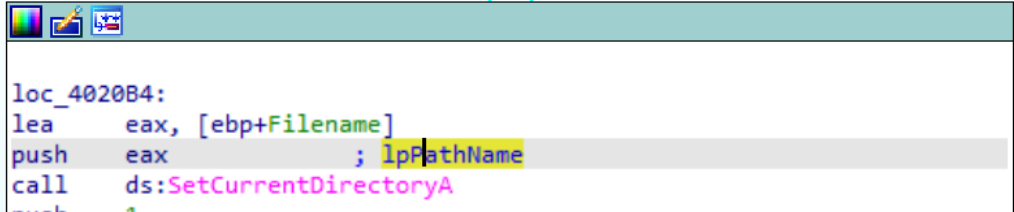
Explanation: This function gets the metadata of the “tasksche.exe” which has been copied.

- **SetCurrentDirectoryA**

Changes the current directory for the current process

Parameter

lpPathName - The path to the new current directory. This parameter may specify a relative path or a full path.



```
loc_4020B4:
lea  eax, [ebp+Filename]
push  eax          ; lpPathName
call  ds:SetCurrentDirectoryA
```

Explanation: This function copies the exe file “tasksche.exe,” to the current process’s directory.

Moving into this function



```
push  1
call  sub_4010FD
mov   [esp+6F4h+Str], offset Str ; "wNcry@2o17"
push  ebx
call  sub_401DAB
```

- **FindResourceA**

Determines the location of a resource with the specified type and name in the specified module.

Parameters

hModule - A handle to the module whose portable executable file or an accompanying MUI file contains the resource. If this parameter is NULL, the function searches the module used to create the current process.

MUI – A language that allows a program to run multiple languages.

lpName - The name of the resource.

```

sub_401DAB proc near

Src= 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

```

Explanation: This function finds the resource of a process, presumably the current process.

- **LoadResource**

Retrieves a handle that can be used to obtain a pointer to the first byte of the specified resource in memory.

Parameters

hModule - A handle to the module whose executable file contains the resource. If hModule is NULL, the system loads the resource from the module that was used to create the current process.

hResInfo - A handle to the resource to be loaded.

```

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

```

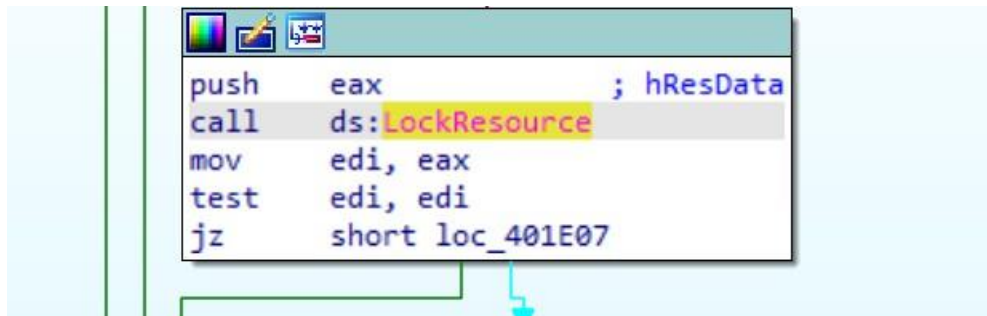
Explanation: As said of this function, it retrieves the handle to the first byte of the current process's memory.

- **LockResource**

Retrieves a pointer to the specified resource in memory.

Parameters

hResData - A handle to the resource to be accessed.



A screenshot of a debugger's assembly window showing the implementation of the LockResource function. The code is as follows:

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

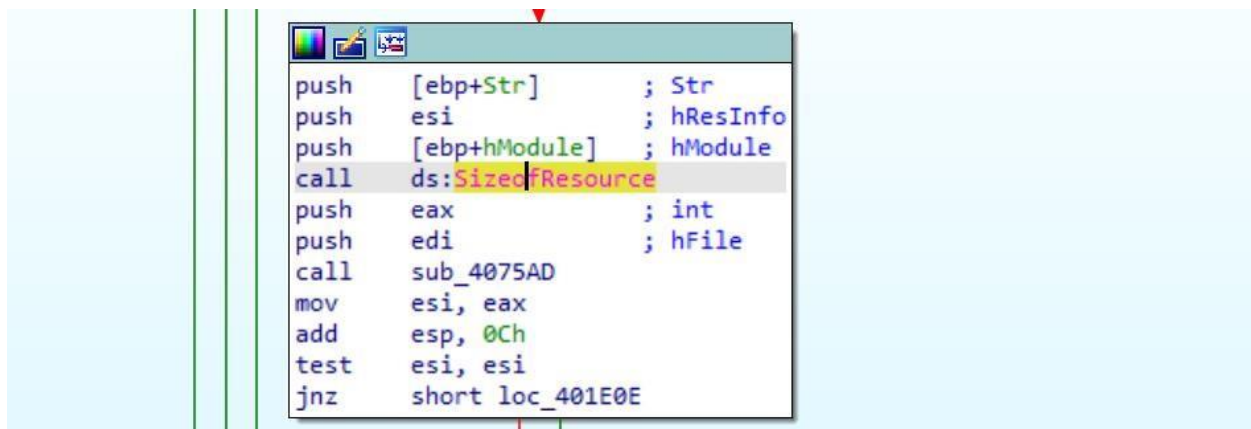
- **SizeofResource**

Retrieves the size, in bytes, of the specified resource.

Parameters

hModule - A handle to the module whose executable file contains the resource.

hResInfo - A handle to the resource.



A screenshot of a debugger's assembly window showing the implementation of the SizeofResource function. The code is as follows:

```
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
add     esp, 0Ch
test    esi, esi
jnz     short loc_401E0E
```

Entering into this function

```
push    ebx
push    4Ah ; 'j'
xor     eax, eax
pop     ecx
lea     edi, [ebp+Str1]
rep stosd
lea     eax, [ebp+Src]
push    eax           ; Src
push    0FFFFFFFh    ; int
push    esi           ; int
call   sub_4075C4
mov     ebx, [ebp+Src]
add     esp, 0Ch
xor     edi, edi
test    ebx, ebx
jle    short loc_401E8F
```

Entering into the subfunction

```
v  eax, 80000h
p  short loc_4075FD
```

```
loc_4075F0:
mov     ecx, [ecx+4]
push    eax           ; Src
push    [esp+4+arg_4] ; int
call   sub_406C40
```

- **ReadFile**

Reads the file

```
lea     eax, [ebp+lpBuffer]
push    0           ; lpOverlapped
push    eax         ; lpNumberOfBytesRead
push    edi         ; nNumberOfBytesToRead
push    [ebp+lpBuffer] ; lpBuffer
push    dword ptr [esi+4] ; hFile
call   ds:ReadFile
test    eax, eax
jnz    short loc_405DB9
```

```
loc_405DB9:
mov     eax, [esi+4]
mov     ebx, [esi+8]
lea     ecx, [esi+12]
cmp     ecx, ebx
jbe     short loc_405DB9
```

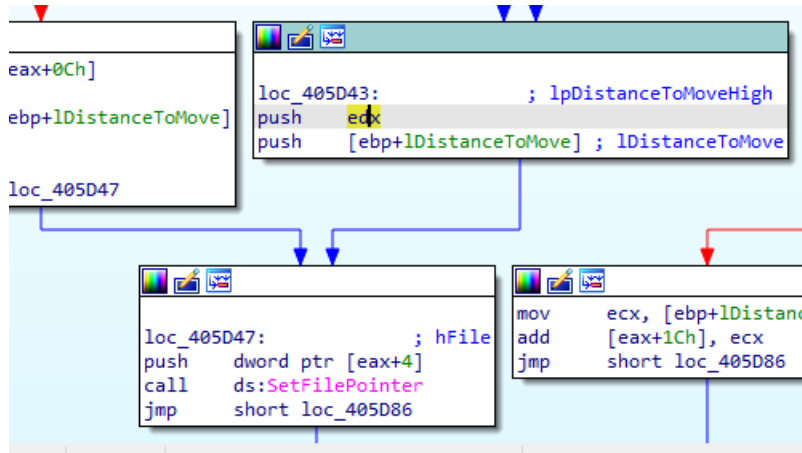
- **SetFilePointer**

Moves the file pointer of the specified file.

Parameters

hFile - A handle to the file.

lDistanceToMove - The low order 32-bits of a signed value that specifies the number of bytes to move the file pointer.



- **SystemTimeToFileTime**

Converts a system time to file time format.

Parameters

lpSystemTime - A pointer to a SYSTEMTIME structure that contains the system time to be converted from UTC to file time format.

lpFileTime - A pointer to a FILETIME structure to receive the converted system time.

```
mov     ecx, [ebp+lSystemTime]
shr     ecx, 5
push    eax                ; lpFileTime
lea     eax, [ebp+SystemTime]
and     ecx, 3Fh
push    eax                ; lpSystemTime
mov     [ebp+SystemTime.wMinute], cx
call    ds:SystemTimeToFileTime
mov     eax, [ebp+FileTime.dwLowDateTime]
mov     edx, [ebp+FileTime.dwHighDateTime]
leave
retn
sub_406B23 endp
```

Explanation: The SystemTimeToFileTime function converts the time of the local machine to the format of the time of the file presumably, the “tasksche.exe”. This is to make sure that the correct time set for some execution or scheduling of some process would be initiated at the right time as expected.

- **LocalFileTimeToFiletime**

Converts a local file time to a file time based on the Coordinated Universal Time (UTC).

Parameters

lpLocalFileTime - A pointer to a FILETIME structure that specifies the local file time to be converted into a UTC-based file time.

lpFileTime - A pointer to a FILETIME structure to receive the converted UTC-based file time. This parameter cannot be the same as the lpLocalFileTime parameter.

```

push    ecx
lea     eax, [ebp+FileTime]
push    eax           ; lpFileTime
lea     eax, [ebp+LocalFileTime]
push    eax           ; lpLocalFileTime
mov     [ebp+LocalFileTime.dwHighDateTime], edx
call    ds:LocalFileTimeToFileTime
mov     eax, [ebp+FileTime.dwLowDateTime]
mov     ecx, [ebp+FileTime.dwHighDateTime]
cmp     [ebp+var_C], 4
mov     ebx, dword ptr [ebp+Str1]
mov     [esi+10Ch], eax
mov     [esi+114h], eax
mov     [esi+11Ch], eax
mov     [esi+110h], ecx

```

- **CreateFileA**

Creates or opens a file or I/O device.

Parameters

lpFileName - The name of the file or device to be created or opened.

dwDesiredAccess - The requested access to the file or device, which can be summarized as read, write, both or 0 to indicate neither).

The value `0xC0000000` (or `40000000h` in hexadecimal) is a combination of several access rights:

- `GENERIC_READ` (0x80000000): Allows the caller to read the object.
- `GENERIC_WRITE` (0x40000000): Allows the caller to write to the object.
- `GENERIC_EXECUTE` (0x20000000): Allows the caller to execute the object.

dwShareMode - The requested sharing mode of the file or device, which can be read, write, both, delete, all of these, or none. If this parameter is zero and CreateFile succeeds, the file or device cannot be shared and cannot be opened again until the handle to the file or device is closed.

Value	Meaning
0 0x00000000	Prevents other processes from opening a file or device if they request delete, read, or write access.

lpSecurityAttributes - A pointer to a SECURITY_ATTRIBUTES structure that contains two separate but related data members. If this parameter is NULL, the handle returned by CreateFile cannot be inherited by any child processes the application may create and the file or device associated with the returned handle gets a default security descriptor.

dwCreationDisposition - An action to take on a file or device that exists or does not exist.

Value	Meaning
CREATE_ALWAYS 2	<p>Creates a new file, always.</p> <p>If the specified file exists and is writable, the function truncates the file, the function succeeds, and last-error code is set to ERROR_ALREADY_EXISTS (183).</p> <p>If the specified file does not exist and is a valid path, a new file is created, the function succeeds, and the last-error code is set to zero.</p> <p>For more information, see the Remarks section of this topic.</p>

dwFlagsAndAttributes - The file or device attributes and flags, FILE_ATTRIBUTE_NORMAL being the most common default value for files.

```

loc_407312:
xor     eax, eax
push   eax           ; hTemplateFile
push   [ebp+dwFlagsAndAttributes] ; dwFlagsAndAttributes
push   2             ; dwCreationDisposition
push   eax           ; lpSecurityAttributes
push   eax           ; dwShareMode
lea    eax, [ebp+FileName]
push   40000000h    ; dwDesiredAccess
push   eax           ; lpFileName
call   ds:CreateFileA

```

Explanation: This function gives the “tasksche.exe” writing permissions. Also, the exe prevents other processes from opening a file or a device if those processes would request a read, write or delete access based on the **ShareMode** configuration hijacking the machine.

- **CloseHandle**

Closes an open object handle.

```

push   [ebp+hFile] ; hObject
call   ds:CloseHandle

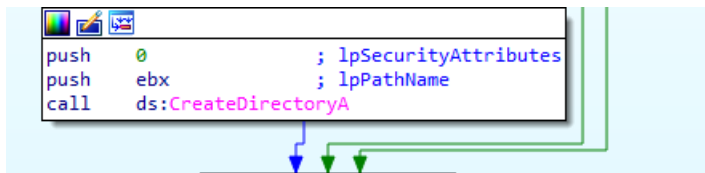
```

- **CreateDirectoryA**

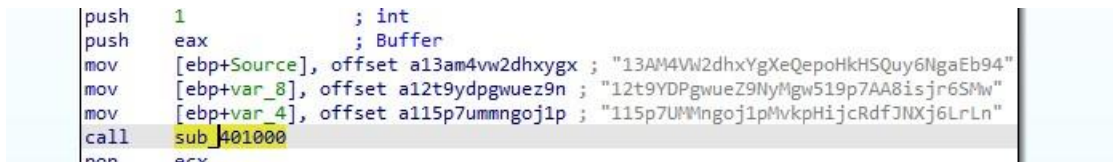
Creates a new directory.

Parameters

lpSecurityAttributes - A pointer to a SECURITY_ATTRIBUTES structure. The lpSecurityDescriptor member of the structure specifies a security descriptor for the new directory. If lpSecurityAttributes is NULL, the directory gets a default security descriptor.

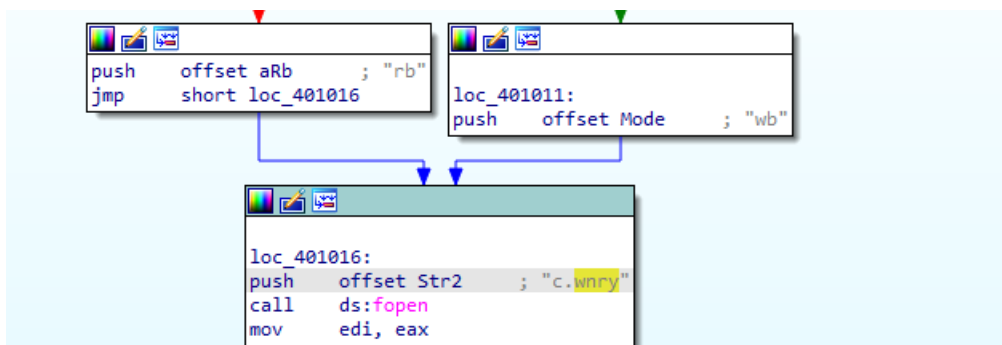


Encryption keys being passed to this function



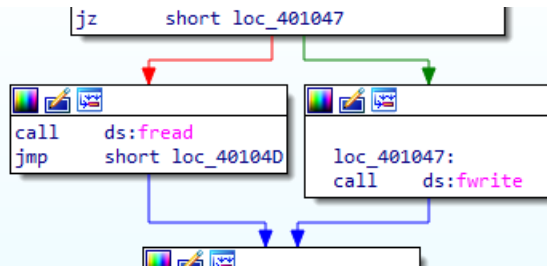
Explanation: From the above image, it looks like the encryption keys are being passed from a file to the function to execute a particular function.

- **fopen**



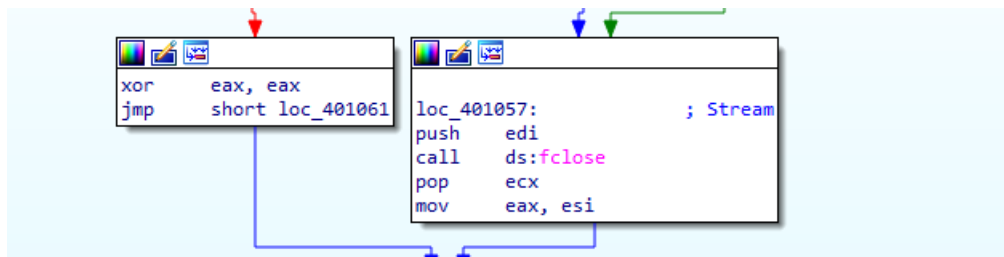
Explanation: The **fopen** function is used to create a file. From the assembly, it looks like the "c.wnry" is a configuration file that is being created with this function.

- **fread and fwrite**



Explanation: This is the function used to write the encryption keys to the configuration file "c.wnry".

fclose



Explanation: The file is then closed after successful addition of the keys.

- **Icacls with attrib +h** to grant access to directories and subdirectories and addition of hidden attribute to hide files.

```

push    ebx                ; lpExitCode
push    ebx                ; dwMilliseconds
push    offset CommandLine ; "attrib +h ."
call    sub_401064
push    ebx                ; lpExitCode
push    ebx                ; dwMilliseconds
push    offset aIcaclsGrantEve ; "icacls . /grant Everyone:F /T /C /Q"
call    sub_401064
  
```

Explanation: In the above image, it suggests that the configuration file which is created in the current process directory is now being hidden.

- **CreateProcess**

Creates a new process and its primary thread. The new process runs in the security context of the calling process.

Parameters

lpApplicationName - The name of the module to be executed.

lpProcessAttributes - A pointer to a SECURITY_ATTRIBUTES structure that determines whether the returned handle to the new process object can be inherited by child processes. If lpProcessAttributes is NULL, the handle cannot be inherited.

lpThreadAttributes - A pointer to a SECURITY_ATTRIBUTES structure that determines whether the returned handle to the new thread object can be inherited by child processes. If lpThreadAttributes is NULL, the handle cannot be inherited.

bInheritHandles - If this parameter is TRUE, each inheritable handle in the calling process is inherited by the new process. If the parameter is FALSE, the handles are not inherited.

dwCreationFlags - The flags that control the priority class and the creation of the process.

CREATE_NO_WINDOW 0x08000000	The process is a console application that is being run without a console window. Therefore, the console handle for the application is not set. This flag is ignored if the application is not a console application, or if it is used with either CREATE_NEW_CONSOLE or DETACHED_PROCESS.
---------------------------------------	----------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------

lpEnvironment - A pointer to the environment block for the new process. If this parameter is NULL, the new process uses the environment of the calling process.

lpCurrentDirectory - The full path to the current directory for the process. The string can also specify a UNC path. If this parameter is NULL, the new process will have the same current drive and directory as the calling process.

lpStartupInfo - A pointer to a STARTUPINFO or STARTUPINFOEX structure.

lpProcessInformation - A pointer to a PROCESS_INFORMATION structure that receives identification information about the new process.

```
pup    esi
mov    [ebp+StartupInfo.wShowWindow], esi
push   eax                ; lpProcessInformation
lea   eax, [ebp+StartupInfo]
push   eax                ; lpStartupInfo
push   esi                ; lpCurrentDirectory
push   esi                ; lpEnvironment
push   8000000h          ; dwCreationFlags
push   esi                ; bInheritHandles
push   esi                ; lpThreadAttributes
push   esi                ; lpProcessAttributes
mov    [ebp+StartupInfo.dwFlags], edi
push   [ebp+lpCommandLine] ; lpCommandLine
push   esi                ; lpApplicationName
call   ds:CreateProcessA
test   eax, eax
jz     short loc_4010F7
```

Explanation: This explains that the commandline command “attrib +h” and other threads are executed to hide files and folders.

- **WaitForSingleObject**

```
push [ebp+dwMilliseconds] ; dwMilliseconds
push [ebp+ProcessInformation.hProcess] ; hHandle
call ds:WaitForSingleObject
test eax, eax
jz short loc_4010D2
```

Explanation: This function is to make sure that only one thread is accessing a particular resource at a time in a situation where multiple threads are sharing the same resources.

- **TerminateProcess**

Terminates the specified process and all of its threads.

Parameters

The **TerminateProcess** function is a Windows API function that is used to cause a process to exit. It takes two parameters: a handle to the process to be terminated and an exit code. The exit code is a 32-bit unsigned integer that is used by the operating system to determine why the process terminated ¹.

The value **0xFFFFFFFF** (or **0FFFFFFFFh** in hexadecimal) is often used as the exit code when a process is terminated abnormally, such as when it is killed by another process or when the system is shutting down. This value is defined as **STILL_ACTIVE** in the Windows API and indicates that the process is still active ³.

hProcess – the handle to the process to terminate.

```
push 0FFFFFFFFh ; uExitCode
push [ebp+ProcessInformation.hProcess] ; hProcess
call ds:TerminateProcess
```

Explanation: This function kills the specified process that is indicated in the “hProcess”.

- **GetExitCodeProcess**

Retrieves the termination status of the specified process.

Parameters

hProcess - A handle to the process.

lpExitCode - A pointer to a variable to receive the process termination status.

```
push    [ebp+lpExitCode] ; lpExitCode
push    [ebp+ProcessInformation.hProcess] ; hProcess
call    ds:|GetExitCodeProcess
```

Explanation: This function would then retrieve the exit status to determine whether a process was successfully terminated or not.

- **CloseHandle**

```
loc_4010E3:                ; hObject
push    [ebp+ProcessInformation.hProcess]
mov     esi, ds:CloseHandle
call    esi ; CloseHandle
push    [ebp+ProcessInformation.hThread] ; hObject
call    esi ; CloseHandle
mov     eax, edi
jmp     short loc_4010F9
```

Explanation: This function closes the process in which the “tasksche.exe” has been injected into.

- Now calling libraries

```
push    offset ModuleName ; "kernel32.dll"
call    ds:LoadLibraryA
mov     edi, eax
cmp     edi, ebx
jz      loc_4017D8
```

```
push    esi
mov     esi, ds:GetProcAddress
push    offset ProcName ; "CreateFileW"
push    edi ; hModule
call    esi ; GetProcAddress
push    offset aWritefile ; "WriteFile"
push    edi ; hModule
mov     dword_40F878, eax
call    esi ; GetProcAddress
push    offset aReadfile ; "ReadFile"
push    edi ; hModule
mov     dword_40F87C, eax
call    esi ; GetProcAddress
push    offset aMovefilew ; "MoveFileW"
push    edi ; hModule
mov     dword_40F880, eax
call    esi ; GetProcAddress
push    offset aMovefileexw ; "MoveFileExW"
push    edi ; hModule
mov     dword_40F884, eax
call    esi ; GetProcAddress
```

```
push    offset aDeletefilew ; "DeleteFileW"
push    edi ; hModule
mov     dword_40F888, eax
call    esi ; GetProcAddress
push    offset aClosehandle ; "CloseHandle"
push    edi ; hModule
mov     dword_40F88C, eax
call    esi ; GetProcAddress
```

Explanation: These indicates some libraries which have been used by the “tasksche.exe” file.

- **InitializeCriticalSection**

The `InitializeCriticalSection` function is a Windows API function that initializes a critical section object. A critical section object is a synchronization object that allows one thread at a time to access a resource or section of code. It is used to prevent multiple threads from accessing shared resources simultaneously, which can lead to data inconsistency or corruption ¹.

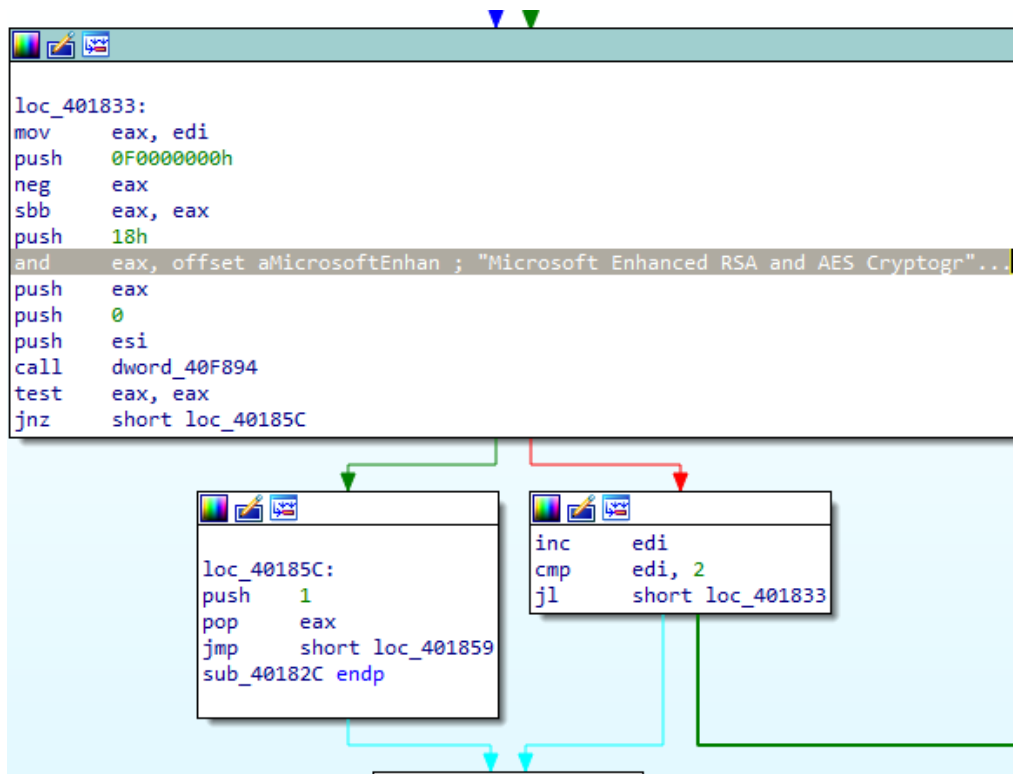
Parameters

lpCriticalSection

The `lpCriticalSection` parameter is a pointer to the critical section object to be initialized. The process is responsible for allocating the memory used by a critical section object, which it can do by declaring a variable of type `CRITICAL_SECTION`. Before using a critical section, some thread of the process must initialize the object ¹.

```
lea    eax, [esi+10h]
push   eax ; lpCriticalSection
mov    dword ptr [esi], offset off_4081EC
call   ds:InitializeCriticalSection
mov    eax, esi
```

“and” bitwise comparison to make sure “Microsoft Enhanced RSA and AES Cryptography at eax register.



Explanation: This bitwise operation verifies that the encryption keys that are passed to the configuration file is the correct keys.

- **CreateFileA**

Creates or opens a file or I/O device.

Parameters

lpFileName - The name of the file or device to be created or opened.

dwDesiredAccess - The requested access to the file or device, which can be summarized as read, write, both or 0 to indicate neither).

The value `0xC0000000` (or `40000000h` in hexadecimal) is a combination of several access rights:

- `GENERIC_READ` (0x80000000): Allows the caller to read the object.
- `GENERIC_WRITE` (0x40000000): Allows the caller to write to the object.
- `GENERIC_EXECUTE` (0x20000000): Allows the caller to execute the object.

dwShareMode - The requested sharing mode of the file or device, which can be read, write, both, delete, all of these, or none. If this parameter is zero and CreateFile succeeds, the file or device cannot be shared and cannot be opened again until the handle to the file or device is closed.

`FILE_SHARE_READ`
0x00000001

Enables subsequent open operations on a file or device to request read access. Otherwise, other processes cannot open the file or device if they request read access.

If this flag is not specified, but the file or device has been opened for read access, the function fails.

lpSecurityAttributes - A pointer to a SECURITY_ATTRIBUTES structure that contains two separate but related data members. If this parameter is NULL, the handle returned by CreateFile cannot be inherited by any child processes the application may create and the file or device associated with the returned handle gets a default security descriptor.

```

xor     esi, esi
mov     [ebp+NumberOfBytesRead], esi

```

dwCreationDisposition - An action to take on a file or device that exists or does not exist.

`OPEN_EXISTING`
3

Opens a file or device, only if it exists.

If the specified file or device does not exist, the function fails and the last-error code is set to `ERROR_FILE_NOT_FOUND` (2).

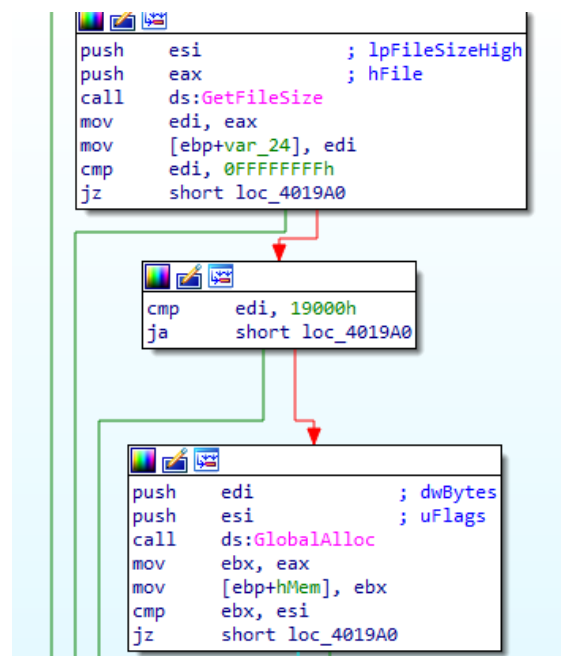
For more information about devices, see the Remarks section.

dwFlagsAndAttributes - The file or device attributes and flags, FILE_ATTRIBUTE_NORMAL being the most common default value for files.

```
mov     [ebp+ms_exc.registration.TryLevel], esi
push   esi           ; hTemplateFile
push   esi           ; dwFlagsAndAttributes
push   3             ; dwCreationDisposition
push   esi           ; lpSecurityAttributes
push   1             ; dwShareMode
push   80000000h     ; dwDesiredAccess
push   [ebp+lpFileName] ; lpFileName
call   ds:CreateFileA
mov    [ebp+hFile].eax
```

Explanation: In this, it seems the “tasksche.exe” is being executed and then a read permission is now allowed. However, this read permission is not granted to any other spawned processes or child processes. This explains why the prompt would be displayed on the infected machine but the user cannot see the other processes that are going on.

- **GetFileSize and GlobalAlloc**



CryptReleaseContext

The `CryptReleaseContext` function releases the handle of a [cryptographic service provider](#) (CSP) and a [key container](#). At each call to this function, the [reference count](#) on the CSP is reduced by one. When the reference count reaches zero, the [context](#) is fully released and it can no longer be used by any function in the application.

An application calls this function after finishing the use of the CSP. After this function is called, the released CSP handle is no longer valid. This function does not destroy [key containers](#) or [key pairs](#).

Parameters

[in] `hProv`

Handle of a cryptographic service provider (CSP) created by a call to [CryptAcquireContext](#).

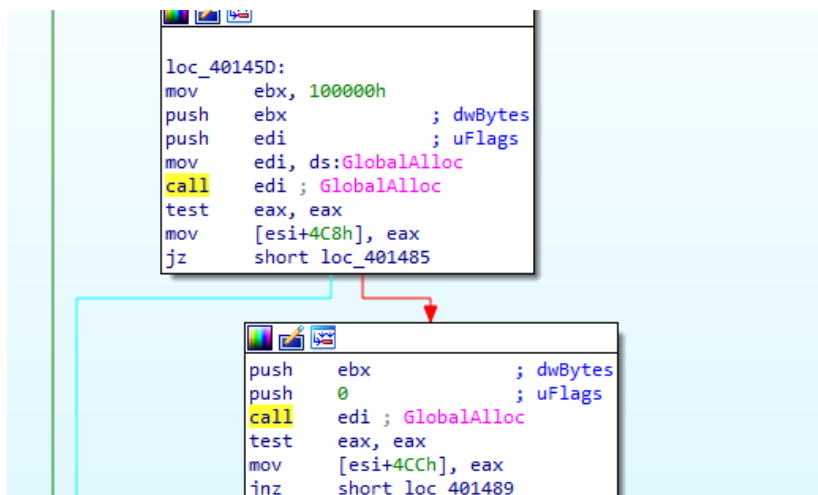
[in] `dwFlags`

Reserved for future use and must be zero. If `dwFlags` is not set to zero, this function returns `FALSE` but the CSP is released.

```
push    0                ; dwFlags
push    eax              ; hProv
call    ds:CryptReleaseContext
and     dword ptr [esi+4], 0
```

Explanation: This function alerts that there should be a link to the cryptographic service provider who would perform encryption of the infected device. It seems the encryption is being performed at least three times based on the encryption keys that have been recovered.

Multiple GlobalAlloc API called



Entering into function that uses "t.wnry" as a parameter.

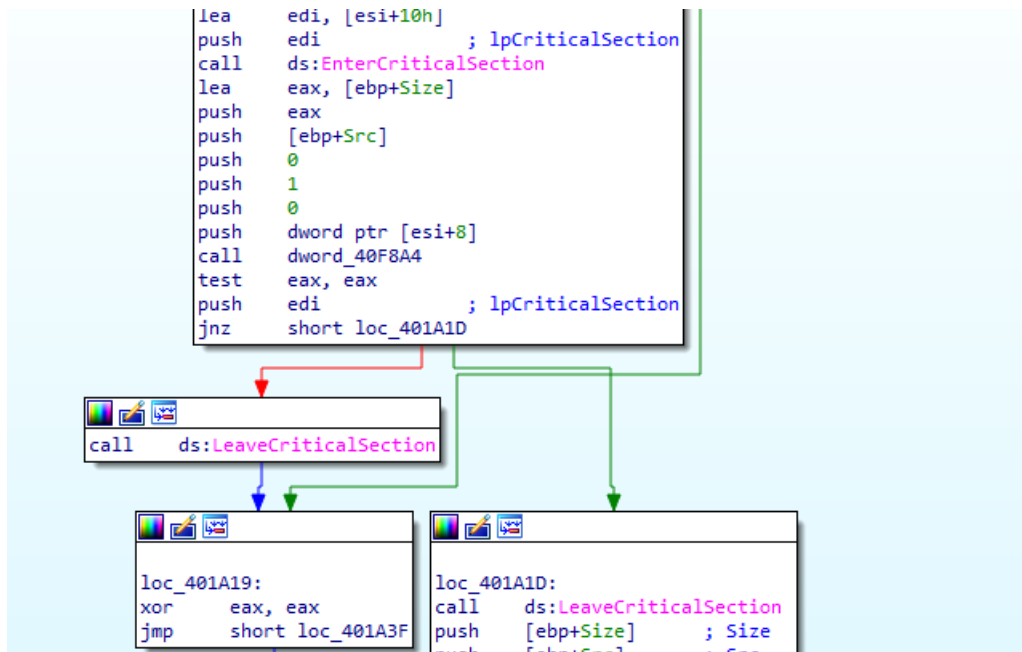
```
push    eax                ; int
push    offset aTwnry     ; "t.wnry"
mov     [ebp+var_4], ebx
call    sub_4014A6
cmn    eax, ebx
```

Explanation: The above is another file that is being used by the executable.

There's a call to a function that uses a memory address containing "WANACRY" as a parameter.

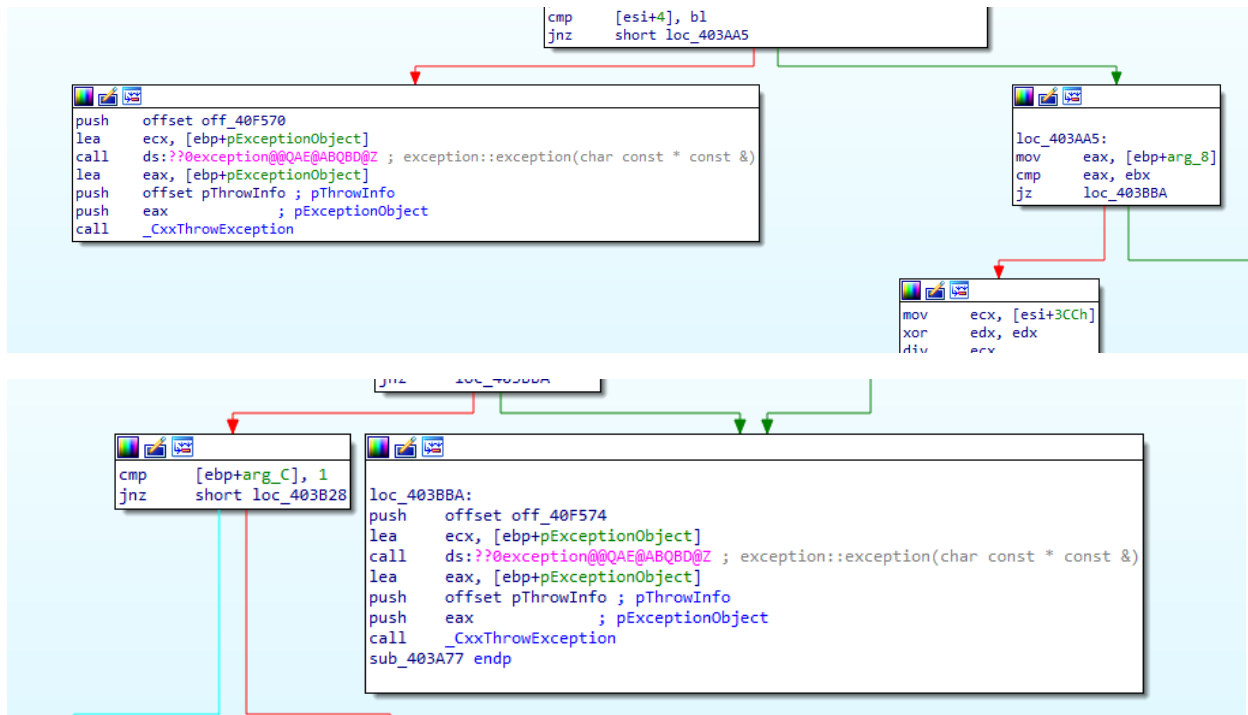
```
push    8                  ; Size
push    offset aWanacry   ; "WANACRY!"
lea     eax, [ebp+Buf1]
push    eax                ; Buf1
call    memcmp
```

Functions to enter critical section and leave critical section.



Coming out of the function.

Multiple exception-handling functions to handle exceptions in creating a process



- Importation of kernel32.dll library

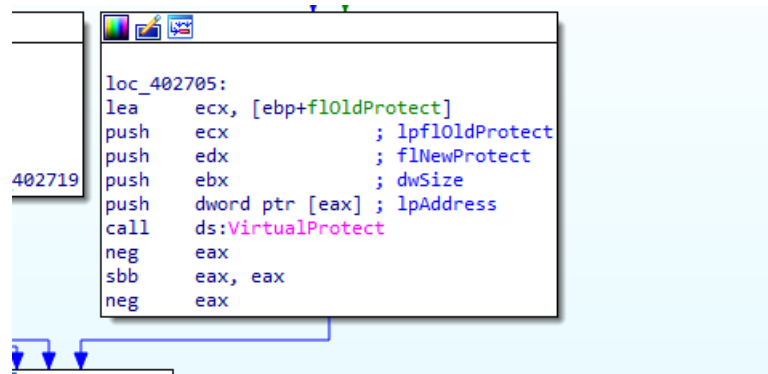
```
loc_40228C:
push  offset ModuleName ; "kernel32.dll"
call  ds:GetModuleHandleA
test   eax, eax
jz     loc_40243D
```

Using it to get systeminfo

```
push  0
push  offset aGetnativesyste ; "GetNativeSystemInfo"
push  eax
call  [ebp+arg_14]
add   esp, 0Ch
test  eax, eax
jz    loc_40243D
```

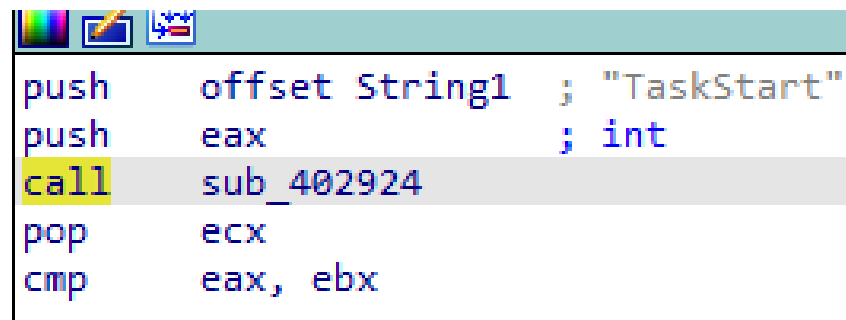
VirtualProtect

Changes the protection on a region of committed pages in the virtual address space of the calling process.



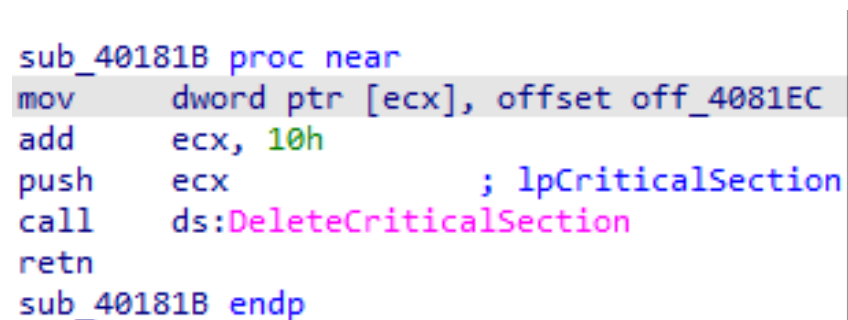
```
loc_402705:
lea    ecx, [ebp+flOldProtect]
push  ecx          ; lpOldProtect
push  edx          ; flNewProtect
push  ebx          ; dwSize
push  dword ptr [eax] ; lpAddress
call  ds:VirtualProtect
neg    eax
sbb   eax, eax
neg    eax
```

Next is calling a function that indicates the start of a task.



```
push  offset String1 ; "TaskStart"
push  eax            ; int
call  sub_402924
pop   ecx
cmp   eax, ebx
---
```

Then it deletes the critical selection.



```
sub_40181B proc near
mov   dword ptr [ecx], offset off_4081EC
add   ecx, 10h
push  ecx          ; lpCriticalSection
call  ds>DeleteCriticalSection
retn
sub_40181B endp
```

c.wnry

```
push    esi
call    sub_4075C4
lea     eax, [ebp+Str1]
push    offset Str2 ; "c.wnry"
push    eax          ; Str1
call    strcmp
add     esp, 14h
```

TaskStart

```
push    offset String1 ; "TaskStart"
push    eax             ; int
call    sub_402924
```

Explanation: The remaining images show that the kernel32.dll library is being used to obtain the system information and then there is a deletion of something after the task has started.

CONCLUSION

This executable is a dropper that has another executable embedded in it. When it infects a machine, it hijacks the machine by injecting a process in the current process and preventing read, write, and delete permissions to any process that would require them. It then creates an executable "tasksche.exe" within the directory of the current process. It then creates some configuration files i.e. "c.wnry" and "t.wnry" files. "c.wnry" contains assumed to be keys for encryption or decryption. Presumably, the "t.wnry" configuration file would contain URLs to the cryptography service provider.

When it executes the tasksche.exe a connection is made to cryptographic service providers to encrypt the data on the machine. It then allows read access but not write and delete access. This read access could display some information to the user. This might be to intimidate the user and provide directions on how to resolve the infection. Since time was called in the instructions, it could be that time was used as a measurement to achieve an aim that is not determined yet.

LIMITATION

Some difficulties were encountered in the analysis. Some of these difficulties were:

1. FLIRT's inability to identify all functions.
2. Limited analysis since it was solely reverse engineering.

The second limitation prevented access to files and exe which is created by the exe. Thus, "c.wnry" which presumably contains the keys, "t.wnry" which is assumed to contain some URLs and the tasksche.exe were not analyzed to see what they do and contain.