Practical Malware Analysis & Triage

Malware Analysis Report

WannaHusky_RansomWare

May 2024 | AnikshaShetty | v1.0

Table of Contents

Executive Summary

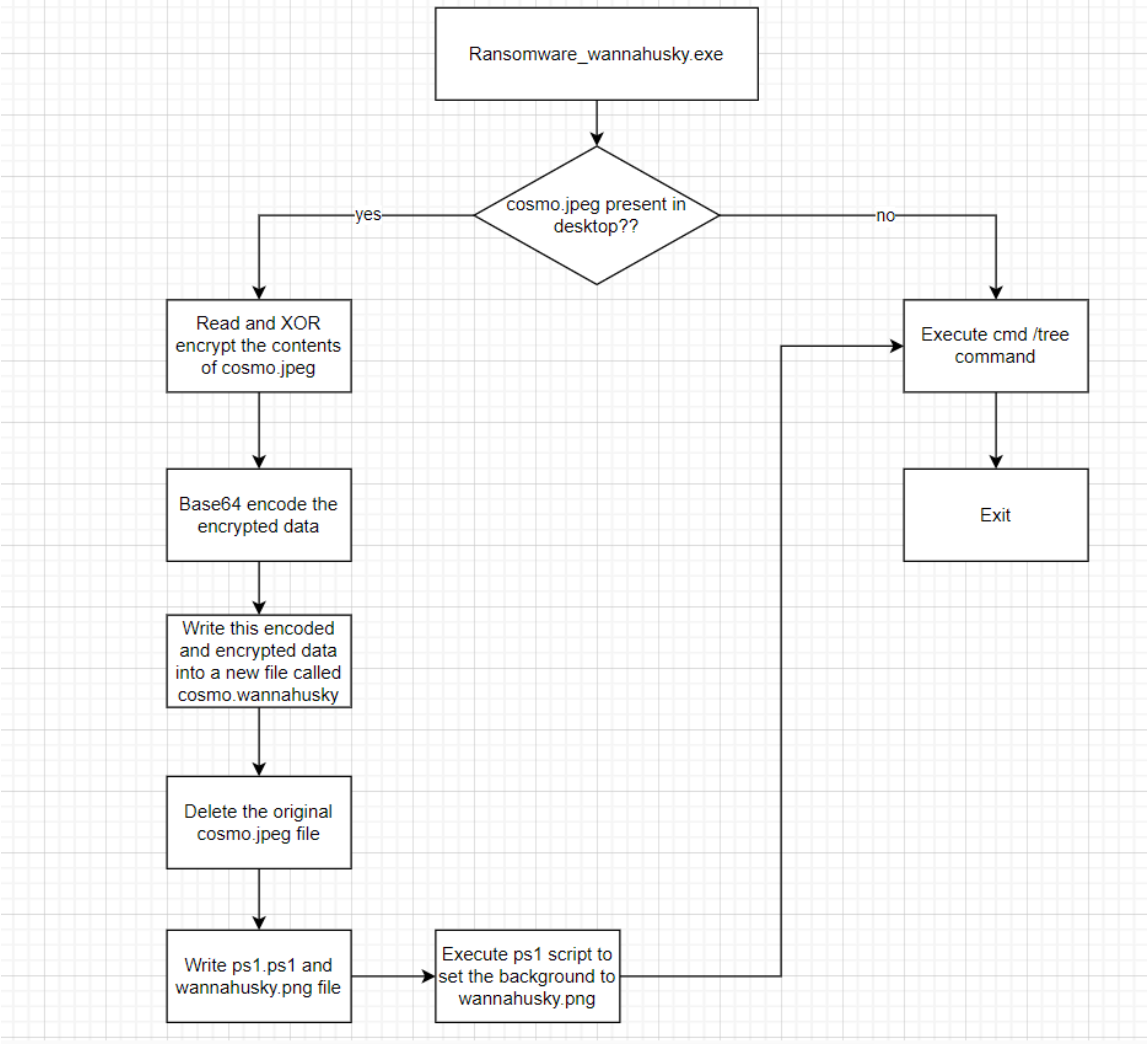
sha256sum	3D35CEBCF40705C23124FDC4656A7F400A316B8E96F1F9E0C187E82A9D17DCA3

WannaHusky is a nim compiled ransomware sample that encrypts and deletes a particular file(cosmo.jpeg) on the victim's desktop and demands for a ransom payment in order to recover the file. It consists of two parts : encryption routine and the creation of ps1 script to change the background/desktop wallpaper. Symptoms of infection include deletion of cosmo.jpeg file, appearance of cosmo.wannahusky file (base64 encoded encrypted cosmo.jpeg file), wannahusky.png(desktop wallpaper/background image) and a command window with tree view of all directories/sub directories within the system.

High-Level Technical Summary

Firstly, the binary checks for the presence of cosmo.jpeg file on desktop, if it is not present then exits the program. If it is present, then calls the encryption routine which generates a 16 byte xor key and performs a bit wise xor operation on data in cosmo.jpeg file. The encrypted data is then base64 encoded and written into a new file called cosmo.wannahusky and the original cosmo.jpeg file is deleted off the system. It then creates a new file “Wannahusky.png” and ps1.ps1. The powershell script aims at changing the background/desktop wallpaper to Wannahusky.png. Powershell is executed and lastly the “cmd /tree c:” command is run to list all the directories within the C: drive.

Execution Flow



Malware Composition

MalwareName consists of the following components:

FileName	Sha256
ps1.ps1	d6317f374f879cd4e67fb4e9ddc0d283926489f4c0d6cf07d912a247e5cfde99
wannahusky.png	07b3e2937388ac6394a08d35f3a66a80dde38c63b3c459729e2471022961f562

WannaHusky.png
Background image that contains the ransom note.

ps1.ps1
Powershell script that is used to set WannaHusky.png as the new background

Basic Static Analysis

{Screenshots and description about basic static artifacts and methods}

Floss - String Analysis

Interesting finds : ps1 script being created in desktop, wallpaper is being set to wannahusky.png, cosmo.jpeg is likely renamed as cosmo.wannahusky and some cryptographic functions.

ascii	13	section:.rdata	-	-	-	writeDataImpl
ascii	9	section:.rdata	-	-	-	flushImpl
ascii	4	section:.rdata	-	-	-	data
ascii	3	section:.rdata	-	-	-	pos
ascii	23	section:.rdata	-	-	-	@cannot write to stream
ascii	9	section:.rdata	-	-	-	@tree C:\
ascii	12	section:.rdata	-	-	-	@powershell
ascii	37	section:.rdata	-	-	-	using System.Runtime.InteropServices;
ascii	16	section:.rdata	-	-	-	namespace Win32{
ascii	23	section:.rdata	-	-	-	public class Wallpaper{
ascii	47	section:.rdata	-	-	-	[DllImport("user32.dll", CharSet=CharSet.Auto)]
ascii	101	section:.rdata	-	-	-	static extern int SystemParametersInfo (int uAction , int uParam , string lpvParam , int fuWinIni) ;
ascii	48	section:.rdata	-	-	-	public static void SetWallpaper(string thePath){
ascii	37	section:.rdata	-	-	-	SystemParametersInfo(20,0,thePath,3);
ascii	14	section:.rdata	-	-	-	add-type $code
ascii	23	section:.rdata	-	-	-	$currDir = Get-Location
ascii	31	section:.rdata	-	-	-	$wallpaper = ".\WANNAHUSKY.PNG"
ascii	58	section:.rdata	-	-	-	$fullpath = Join-Path -path $currDir -ChildPath $wallpaper
ascii	42	section:.rdata	-	-	-	[Win32.Wallpaper]::SetWallpaper($fullpath)
ascii	3	section:.rdata	-	-	-	PNG

```
oserr.nim
raiseOSError
@USERPROFILE
@unknown OS error
@Additional info:
ABCDEFGHIJKLMNOPQRSTUVWXYZabcdefghijklmnopqrstuvwxyz0123456789-_ABCDEFGHIJKLMNOPQRSTUVWXYZabcdefghijklmnopqrstuvwxyz0123456789+/
@bcmode.nim(456, 9) `
ctx.sizeKey() <= len(key)`
@bcmode.nim(455, 9) `
ctx.sizeBlock() <= len(iv)`
bCryptGenRandom
queryProcessCycleTime
queryUnbiasedInterruptTime
queryIdleProcessorCycleTime
coresCount
hIntel
IOError
streams.nim
```

```
readLineImpl
readDataImpl
peekDataImpl
writeDataImpl
flushImpl
data
@cannot write to stream
@tree C:\
@Desktop\ps1.ps1
@powershell
@Desktop\ps1.ps1
@$code = @'
using System.Runtime.InteropServices;
namespace Win32{

    public class Wallpaper{
      [DllImport("user32.dll", CharSet=CharSet.Auto)]
      static  extern int SystemParametersInfo (int uAction , int uParam , string lpvParam , int fuWinIni) ;
      public static void SetWallpaper(string thePath){
         SystemParametersInfo(20,0,thePath,3);
      }
    }
add-type $code
$currDir = Get-Location
$wallpaper = ".\WANNAHUSKY.PNG"
$fullpath = Join-Path -path $currDir -ChildPath $wallpaper
[Win32.Wallpaper]::SetWallpaper($fullpath)
@Desktop\WANNAHUSKY.png
IHDR
sRGB
gAMA
\tpHYs
```

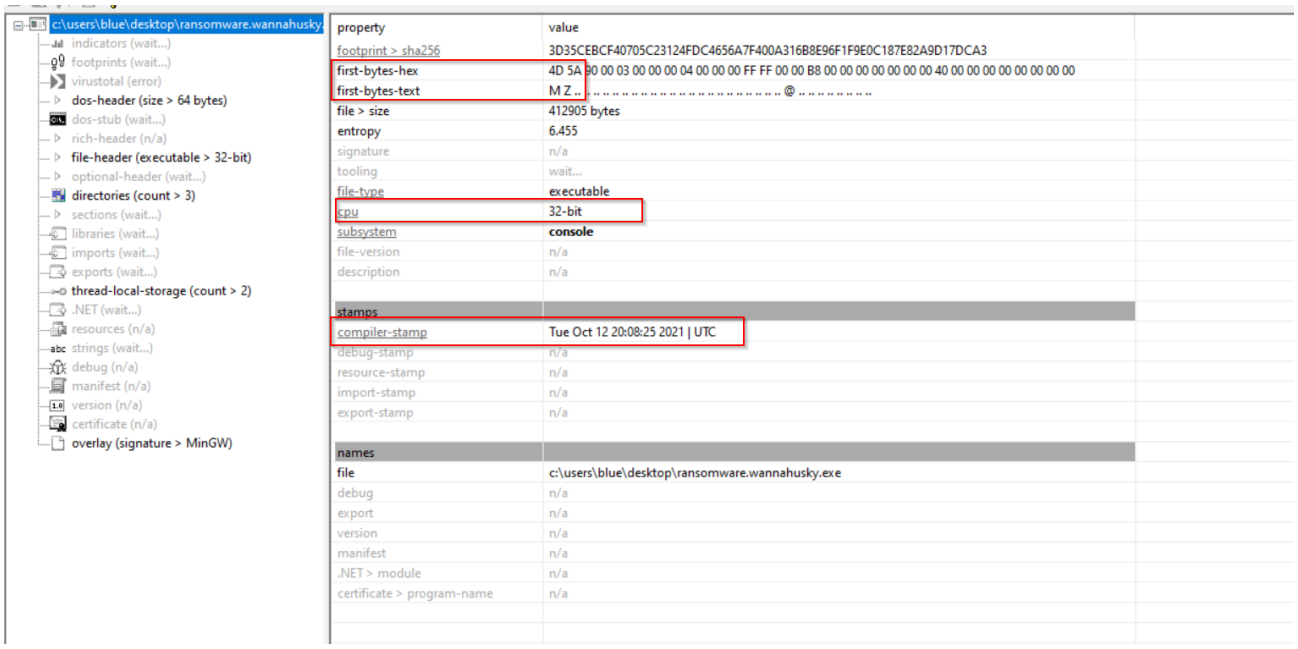
```
?5SZ
%OCZIE
I|w7
z0Gz
1RgOB
@Z-B
IEND
@Desktop\cosmo.WANNAHUSKY
@bcmode.nim(503, 9) `len(input) <= len(output)`
@COSMO
@Desktop\target\cosmo.WANNAHUSKY
@Desktop\cosmo.jpeg
Unknown error
_matherr(): %s in %s(%g, %g)  (retval=%g)
Argument domain error (DOMAIN)
Argument singularity (SIGN)
Overflow range error (OVERFLOW)
The result is too small to be represented (UNDERFLOW)
Total loss of significance (TLOSS)
Partial loss of significance (PLOSS)
Mingw-w64 runtime failure:
Address %p has no image-section
  VirtualQuery failed for %d bytes at address %p
  VirtualProtect failed with code 0x%x
  Unknown pseudo relocation protocol version %d.
  Unknown pseudo relocation bit size %d.
```

writeDataImpl

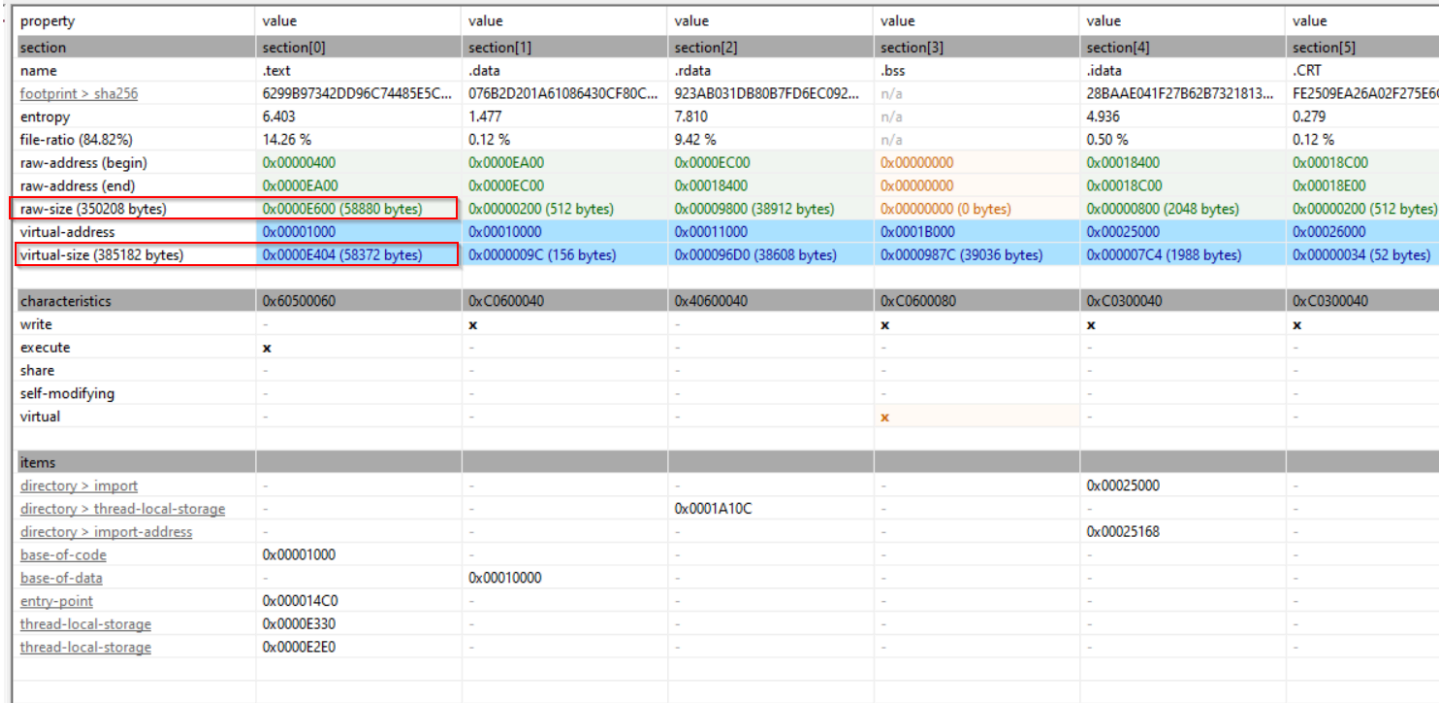
```
@tree C:\
@Desktop\ps1.ps1
@powershell
@Desktop\ps1.ps1
@$code = @'
using System.Runtime.InteropServices;
namespace Win32{
	public class Wallpaper{
	[DllImport("user32.dll", CharSet=CharSet.Auto)]
	static  extern int SystemParametersInfo (int uAction , int uParam , string
lpvParam , int fuWinIni) ;
	public static void SetWallpaper(string thePath){
	SystemParametersInfo(20,0,thePath,3);
	}
	}
add-type $code
$currDir = Get-Location
$wallpaper = ".\WANNAHUSKY.PNG"
$fullpath = Join-Path -path $currDir -ChildPath $wallpaper
[Win32.Wallpaper]::SetWallpaper($fullpath)
@Desktop\WANNAHUSKY.png
@Desktop\cosmo.WANNAHUSKY
@bcmode.nim(503, 9) `len(input) <= len(output)`
@COSMO
@Desktop\target\cosmo.WANNAHUSKY
@Desktop\cosmo.jpeg
@USERPROFILE
@Additional info:
ABCDEFGHIJKLMNOPQRSTUVWXYZabcdefghijklmnopqrstuvwxyz0123456789-_ABC
DEFGHIJKLMNOPQRSTUVWXYZabcdefghijklmnopqrstuvwxyz0123456789+/
@bcmode.nim(456, 9) `
ctx.sizeKey() <= len(key)`
@bcmode.nim(455, 9) `
ctx.sizeBlock() <= len(iv)`
bCryptGenRandom
queryProcessCycleTime
queryUnbiasedInterruptTime
queryIdleProcessorCycleTime
```

PEStudio

This is a 32 bit binary, compiled on oct 12 2021



Not packed, virtual size and raw size does not have significant difference



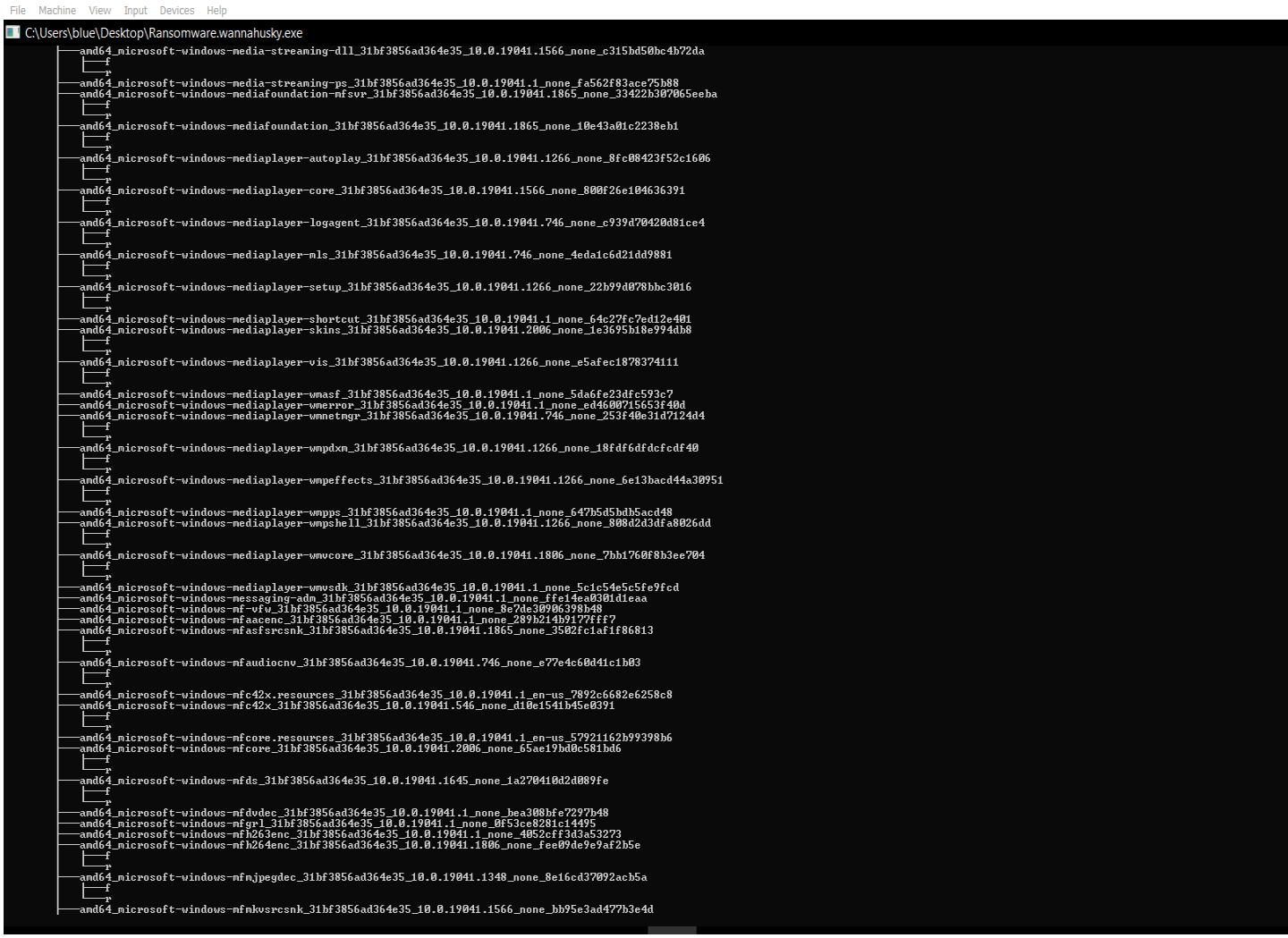
property	value	value	value	value	value	value
section	section[0]	section[1]	section[2]	section[3]	section[4]	section[5]
name	.text	.data	.rdata	.bss	.idata	.CRT
footprint > sha256	6299B97342DD96C74485E5C...	076B2D201A61086430CF80C...	923AB031DB80B7FD6EC092...	n/a	28BAAE041F27B62B7321813...	FE2509EA26A02F275E6
entropy	6.403	1.477	7.810	n/a	4.936	0.279
file-ratio (84.82%)	14.26 %	0.12 %	9.42 %	n/a	0.50 %	0.12 %
raw-address (begin)	0x00000400	0x0000EA00	0x0000EC00	0x00000000	0x00018400	0x00018C00
raw-address (end)	0x0000EA00	0x0000EC00	0x00018400	0x00000000	0x00018C00	0x00018E00
raw-size (350208 bytes)	0x0000E600 (58880 bytes)	0x00000200 (512 bytes)	0x00009800 (38912 bytes)	0x00000000 (0 bytes)	0x00000800 (2048 bytes)	0x00000200 (512 bytes)
virtual-address	0x00001000	0x00010000	0x00011000	0x0001B000	0x00025000	0x00026000
virtual-size (385182 bytes)	0x0000E404 (58372 bytes)	0x0000009C (156 bytes)	0x000096D0 (38608 bytes)	0x0000987C (39036 bytes)	0x000007C4 (1988 bytes)	0x00000034 (52 bytes)
characteristics	0x60500060	0xC0600040	0x40600040	0xC0600080	0xC0300040	0xC0300040
write	-	x	-	x	x	x
execute	x	-	-	-	-	-
share	-	-	-	-	-	-
self-modifying	-	-	-	-	-	-
virtual	-	-	-	x	-	-
items						
directory > import	-	-	-	-	0x00025000	-
directory > thread-local-storage	-	-	0x0001A10C	-	-	-
directory > import-address	-	-	-	-	0x00025168	-
base-of-code	0x00001000	-	-	-	-	-
base-of-data	-	0x00010000	-	-	-	-
entry-point	0x000014C0	-	-	-	-	-
thread-local-storage	0x0000E330	-	-	-	-	-
thread-local-storage	0x0000E2E0	-	-	-	-	-

Basic Dynamic Analysis

{Screenshots and description about basic dynamic artifacts and methods}

Initial Detonation

Running the binary without inet simulation, pops up a black terminal window(cmd??) with some operation happening.
On the desktop new files appear - cosmo.wannahusky, wannahusky.png and ps1 script.
After some time, this window and the ps1 script along with the cosmo.jpeg file disappears.



ProcMon Analysis

Looking at the procmon logs, we can see cosmo.jpeg file is first read and then written into cosmo.wannahusky and then there is a powershell script that gets created in desktop which is then executed using cmd and finally cmd /tree is executed.

New file creation logs :

Time o...	Process Name	PID	Operation	Path	Result
6:40:28....	Ransomware.w...	5412	CreateFile	C:\Users\blue\Desktop\cosmo.jpeg	SUCCESS
6:40:28....	Ransomware.w...	5412	QueryStandardI...	C:\Users\blue\Desktop\cosmo.jpeg	SUCCESS
6:40:28....	Ransomware.w...	5412	ReadFile	C:\Users\blue\Desktop\cosmo.jpeg	SUCCESS
6:40:28....	Ransomware.w...	5412	ReadFile	C:\Users\blue\Desktop\cosmo.jpeg	SUCCESS
6:40:28....	Ransomware.w...	5412	ReadFile	C:\Users\blue\Desktop\cosmo.jpeg	END OF FIL
6:40:28....	Ransomware.w...	5412	ReadFile	C:\Windows\SysWOW64\msvcrt.dll	SUCCESS
6:40:28....	Ransomware.w...	5412	CloseFile	C:\Users\blue\Desktop\cosmo.jpeg	SUCCESS
6:40:28....	Ransomware.w...	5412	CreateFile	C:\Users\blue\Desktop\cosmo.WANNAHUSKY	SUCCESS
6:40:28....	Ransomware.w...	5412	WriteFile	C:\Users\blue\Desktop\cosmo.WANNAHUSKY	SUCCESS
6:40:28....	Ransomware.w...	5412	WriteFile	C:\Users\blue\Desktop\cosmo.WANNAHUSKY	SUCCESS
6:40:28....	Ransomware.w...	5412	WriteFile	C:\Users\blue\Desktop\cosmo.WANNAHUSKY	SUCCESS
6:40:28....	Ransomware.w...	5412	WriteFile	C:\Users\blue\Desktop\cosmo.WANNAHUSKY	SUCCESS
6:40:28....	Ransomware.w...	5412	WriteFile	C:\Users\blue\Desktop\cosmo.WANNAHUSKY	SUCCESS
6:40:28....	Ransomware.w...	5412	WriteFile	C:\Users\blue\Desktop\cosmo.WANNAHUSKY	SUCCESS
6:40:28....	Ransomware.w...	5412	WriteFile	C:\Users\blue\Desktop\cosmo.WANNAHUSKY	SUCCESS
6:40:28....	Ransomware.w...	5412	WriteFile	C:\Users\blue\Desktop\cosmo.WANNAHUSKY	SUCCESS
6:40:28....	Ransomware.w...	5412	WriteFile	C:\Users\blue\Desktop\cosmo.WANNAHUSKY	SUCCESS
6:40:28....	Ransomware.w...	5412	WriteFile	C:\Users\blue\Desktop\cosmo.WANNAHUSKY	SUCCESS
6:40:28....	Ransomware.w...	5412	WriteFile	C:\Users\blue\Desktop\cosmo.WANNAHUSKY	SUCCESS
6:40:28....	Ransomware.w...	5412	WriteFile	C:\Users\blue\Desktop\cosmo.WANNAHUSKY	SUCCESS
6:40:28....	Ransomware.w...	5412	WriteFile	C:\Users\blue\Desktop\cosmo.WANNAHUSKY	SUCCESS
6:40:28....	Ransomware.w...	5412	WriteFile	C:\Users\blue\Desktop\cosmo.WANNAHUSKY	SUCCESS
6:40:28....	Ransomware.w...	5412	WriteFile	C:\Users\blue\Desktop\cosmo.WANNAHUSKY	SUCCESS
6:40:28....	Ransomware.w...	5412	CloseFile	C:\Users\blue\Desktop\cosmo.WANNAHUSKY	SUCCESS
6:40:28....	Ransomware.w...	5412	ReadFile	C:\Users\blue\Desktop\Ransomware.wannahusky.exe	SUCCESS
6:40:28....	Ransomware.w...	5412	CreateFile	C:\Users\blue\Desktop\WANNAHUSKY.png	SUCCESS
6:40:28....	Ransomware.w...	5412	WriteFile	C:\Users\blue\Desktop\WANNAHUSKY.png	SUCCESS
6:40:28....	Ransomware.w...	5412	WriteFile	C:\Users\blue\Desktop\WANNAHUSKY.png	SUCCESS
6:40:28....	Ransomware.w...	5412	CloseFile	C:\Users\blue\Desktop\WANNAHUSKY.png	SUCCESS
6:40:28....	Ransomware.w...	5412	CreateFile	C:\Users\blue\Desktop\cosmo.jpeg	SUCCESS
6:40:28....	Ransomware.w...	5412	QueryAttributeT...	C:\Users\blue\Desktop\cosmo.jpeg	SUCCESS
6:40:28....	Ransomware.w...	5412	CloseFile	C:\Users\blue\Desktop\cosmo.jpeg	SUCCESS
6:40:28....	Ransomware.w...	5412	CreateFile	C:\Users\blue\Desktop\ps1.ps1	SUCCESS
6:40:28....	Ransomware.w...	5412	WriteFile	C:\Users\blue\Desktop\ps1.ps1	SUCCESS
6:40:28....	Ransomware.w...	5412	CloseFile	C:\Users\blue\Desktop\ps1.ps1	SUCCESS
6:40:28....	Ransomware.w...	5412	ReadFile	C:\Windows\SysWOW64\msvcrt.dll	SUCCESS
6:40:28....	Ransomware.w...	5412	CreateFile	C:\Windows\SysWOW64\cmd.exe	SUCCESS
6:40:28....	Ransomware.w...	5412	QueryBasicInfor...	C:\Windows\SysWOW64\cmd.exe	SUCCESS

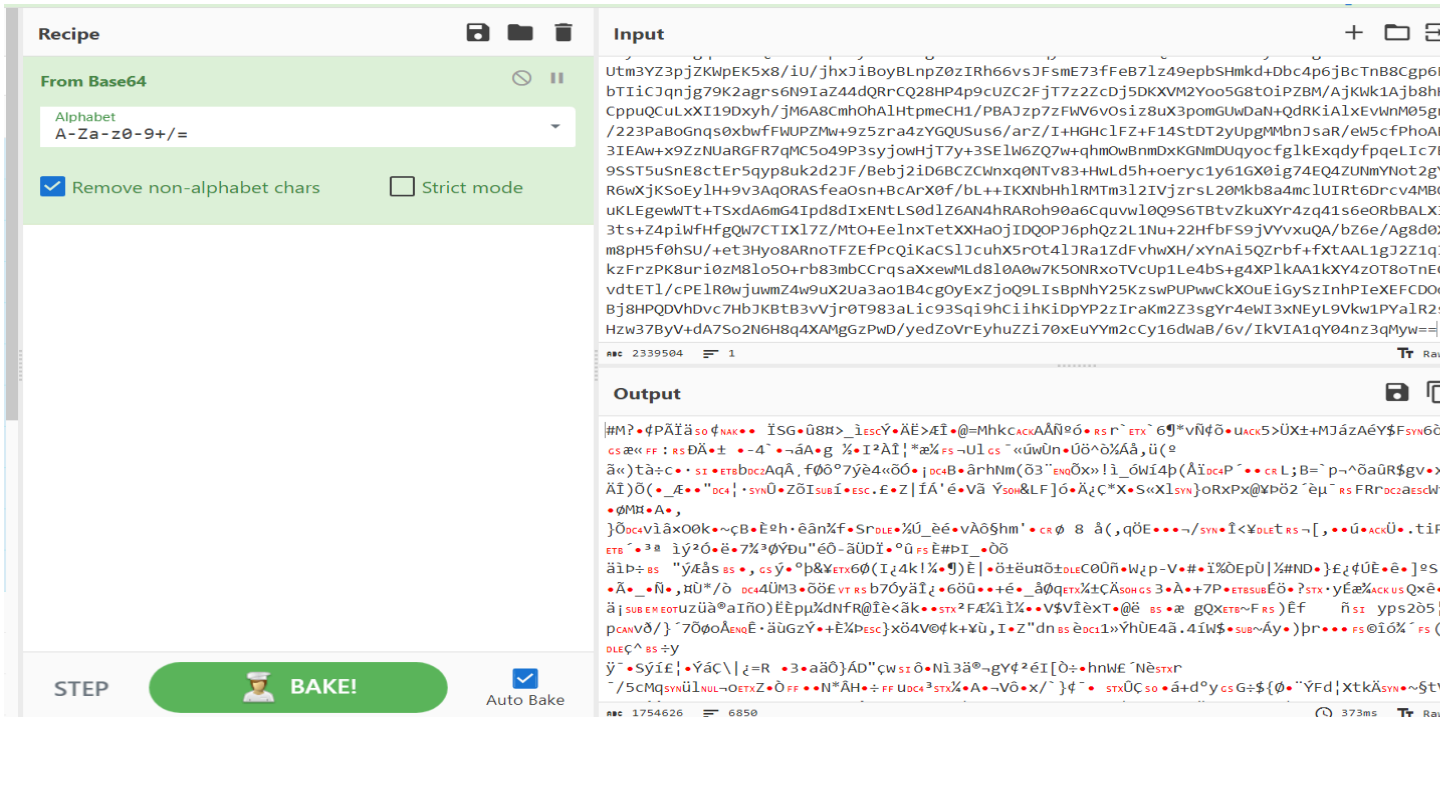
Process logs showing powershell and cmd execution :

Time o...	Process Name	PID	Operation	Path	Result	Detail
6:40:28....	Ransomware.w...	5412	Process Start		SUCCESS	Parent PID: 3164, Command line: "C:\Users\blue\Desktop\Ransomware.wannahusky.exe" , Current directo
6:40:28....	Ransomware.w...	5412	Process Create	C:\Windows\System32\Conhost.exe	SUCCESS	PID: 5132, Command line: \??\C:\Windows\system32\conhost.exe 0xffffffff -ForceV1
6:40:28....	Ransomware.w...	5412	Process Create	C:\Windows\SysWOW64\cmd.exe	SUCCESS	PID: 5392, Command line: C:\Windows\system32\cmd.exe /c powershell C:\Users\blue\Desktop\ps1.ps1
6:40:30....	Ransomware.w...	5412	Process Create	C:\Windows\SysWOW64\cmd.exe	SUCCESS	PID: 1180, Command line: C:\Windows\system32\cmd.exe /c tree C:\
6:40:58....	Ransomware.w...	5412	Process Exit		SUCCESS	Exit Status: 0, User Time: 0.1875000 seconds, Kernel Time: 0.0000000 seconds, Private Bytes: 10,846,208,

Wannahusky.png ---> Ransom note stating the file cosmo.jpg has been encrypted and we need to pay 100 huskycoins to recover the file

Cosmo.wannahusky ---> contains base64 encoded encrypted data.

Decoding b64 gives encrypted data :



WannaHusky_RansomWare
May 2024
v1.0

Advanced Static Analysis

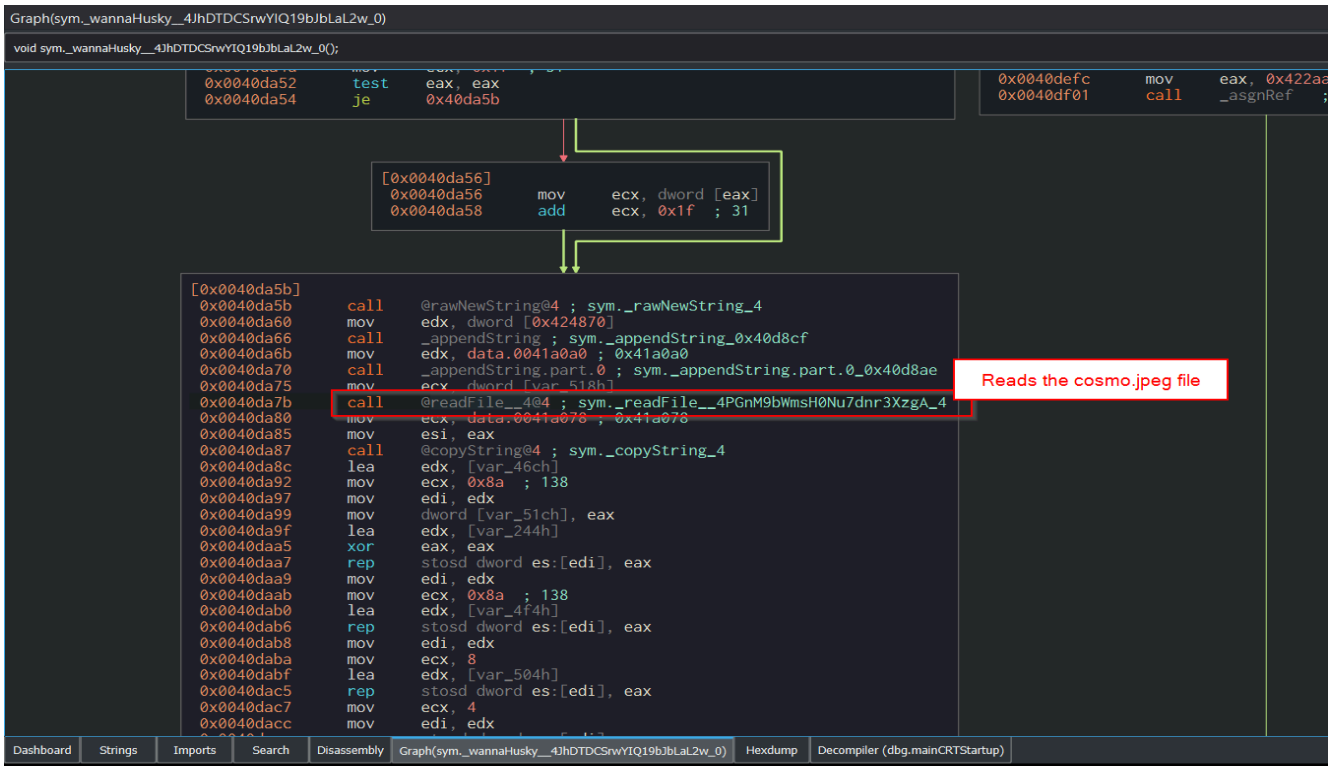
{Screenshots and description about findings during advanced static analysis}

Loading the binary in cutter :

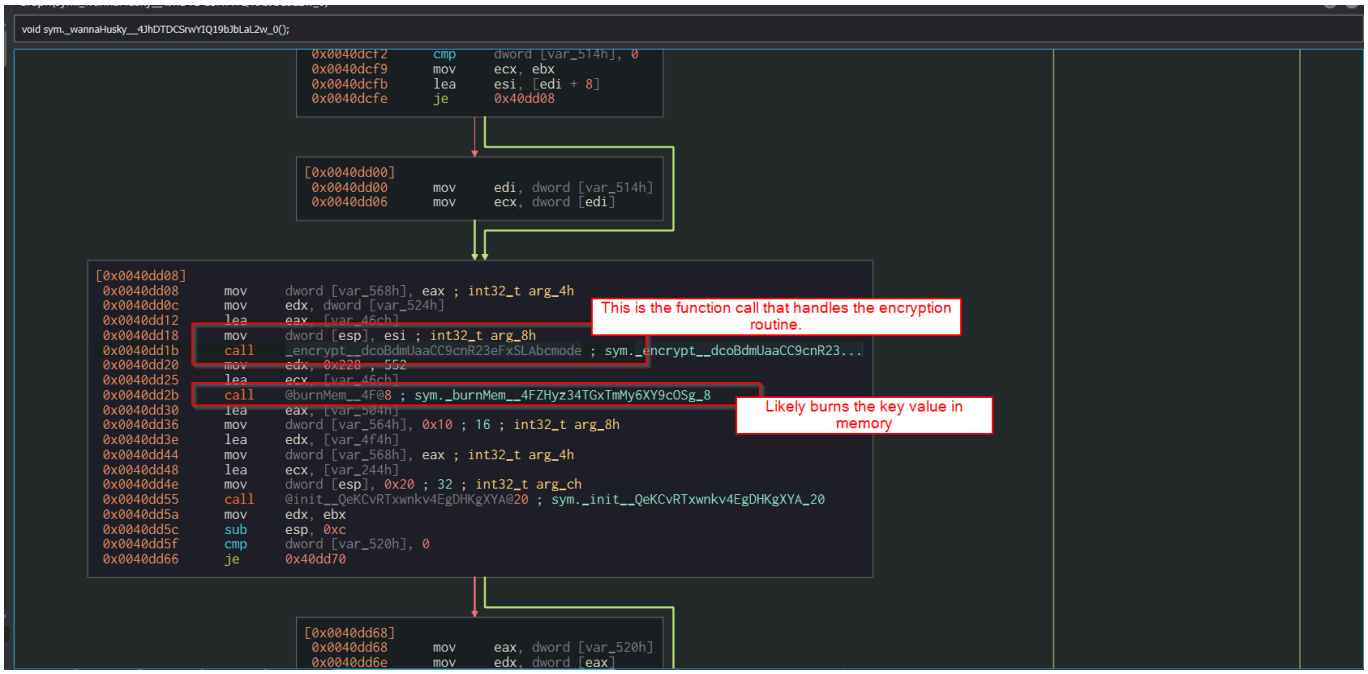
Main program of the binary starts here :

Interesting function call here is the wannaHusky ---> this function contains the functionality to find and read cosmo.jpeg and the encryption routine.

Looking into the wannacry function in depth, it initially constructs the path for cosmo.jpeg and then passes this path to the readfile function which reads the contents of cosmo.jpeg file.



Then there is an encryption routine call, after which the burn memory call is made to likely clear the encryption key stored in memory.



Encryption routine call

Looks like it performs a bit wise xor operation on each byte of data. Lets get into debugging now.

```
uint32_t encrypt_dcoBdmUaaCC9cnR23eFxSLAbcmode (int32_t arg_4h, uint32_t arg_8h) {
    int32_t var_38h;
    int32_t var_34h;
    int32_t var_28h;
    int32_t var_24h;
    uint32_t var_20h;
    int32_t var_10h;
    esi = eax;
    var_24h = edx;
    var_20h = ecx;
    if (ecx > arg_8h) {
        ecx = data.0041a040;
        @failedAssertImpl_W9cjVocn1tjhW7p7xohJj6A@4 ();
    }
    ebx = *((esi + 0x224));
    edi = 0;
    do {
        if (edi >= var_20h) {
            *((esi + 0x224)) = ebx;
            esp = &var_10h;
            return;
        }
        if (ebx == 0) {
            edx = esi + 0x204;
            eax = esi + 0x1e4;
            ecx = esi;
            edx = eax;
            @encrypt_py6wg79aBw8iTzUm11Z7JOA@20 (0x20, 0x20, edx);
            edx = 0x20;
            ecx = esi + 0x1e4;
            @inc128_vRz5m42fv3XKwSYgATX55Q@8 ();
        }
        if (edi >= arg_8h) {
            eax = arg_8h;
            edx = eax - 1;
            _raiseIndexError2 (edi, edx);
        }
        if (edi >= var_20h) {
```

unknown encrypt function that does some shift right and left and xor operation

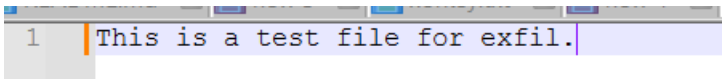
```
            eax = var_20h;
            edx = eax - 1;
            _raiseIndexError2 (edi, edx);
        }
        if (ebx > 0x1f) {
            _raiseIndexError2 (ebx, 0x1f);
        }
        eax = var_24h;
        dl = *((eax + edi));
        eax = arg_4h;
        dl ^= *((esi + ebx + 0x204));
        *((eax + edi)) = dl;
        eax = edi + 1;
        edi ^= eax;
        if (ebx < 0x1f) {
            if (eax >= 0) {
                goto label_0;
            }
            _raiseOverflow (eax);
            eax = var_28h;
        }
label_0:
        ebx++;
        edi = eax;
        ebx &= 0xf;
    } while (1);
}
```

Bit wise xor operation on each byte of data. esi + ebx + 0x204 should be the xor key.
resulted encrypted data is stored in eax + edi

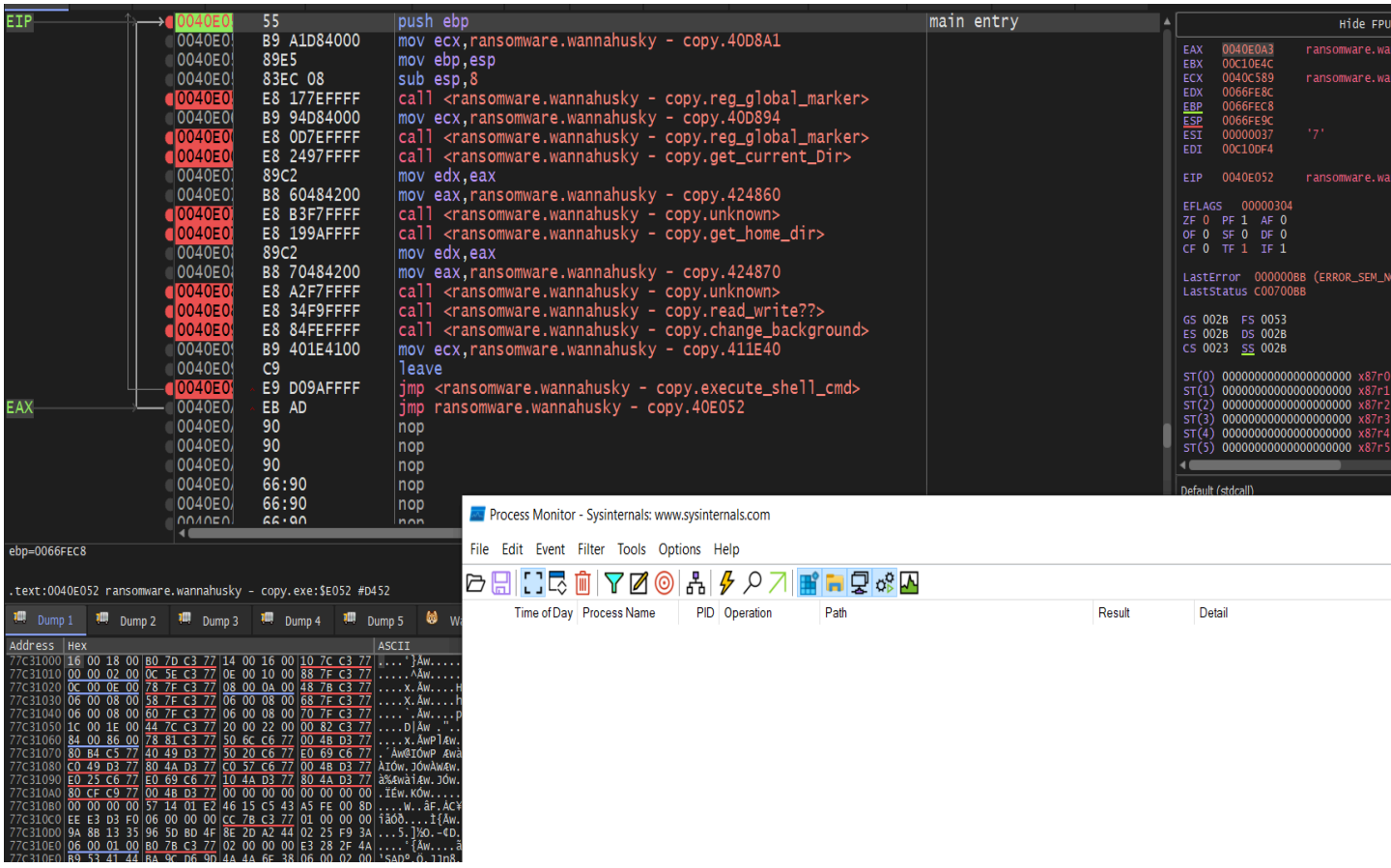
Advanced Dynamic Analysis

{Screenshots and description about advanced dynamic artifacts and methods}

Created a small text file "This is a test file for exfil" and named it as cosmo.jpeg to reduce the time of encryption and see if we can decrypt the data.



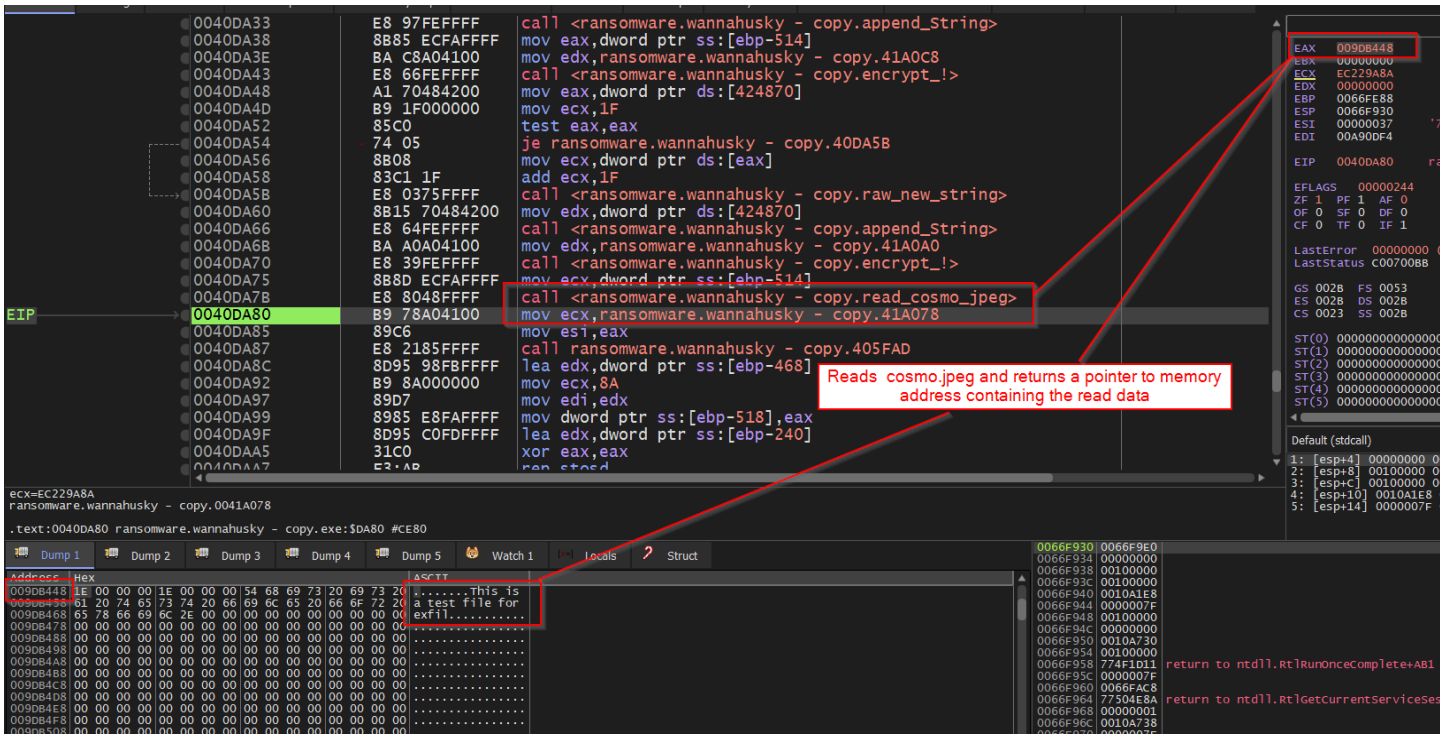
Main entry

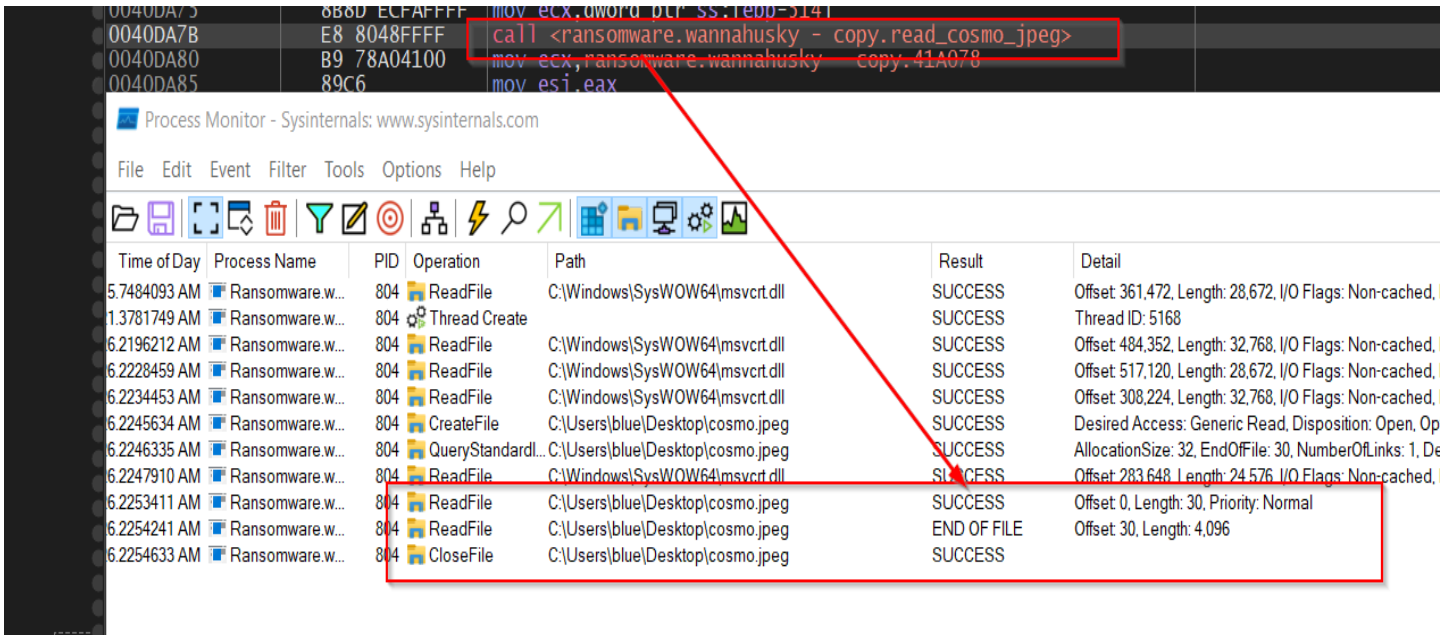


Stepping in through the wannacry function and going over each instruction.

ReadFile

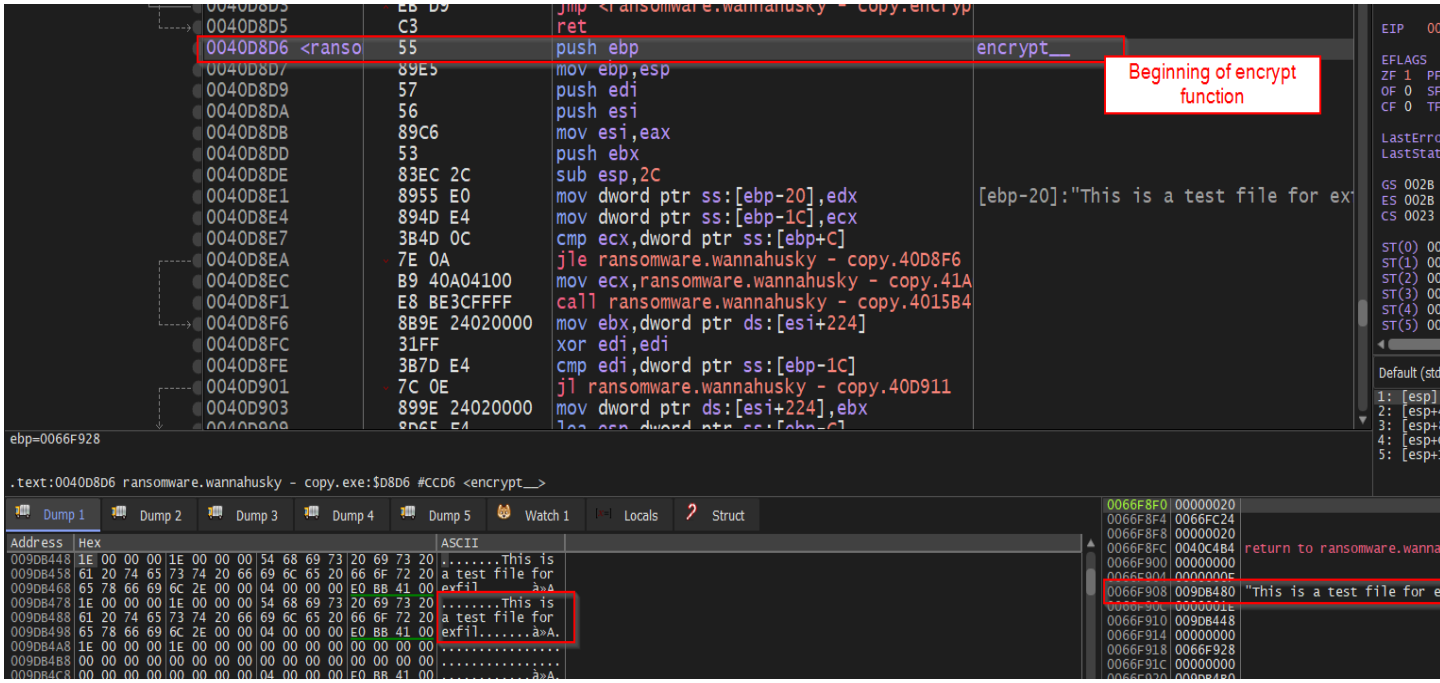
Initially there is a call to fileread which reads and stores the content of the data in memory.



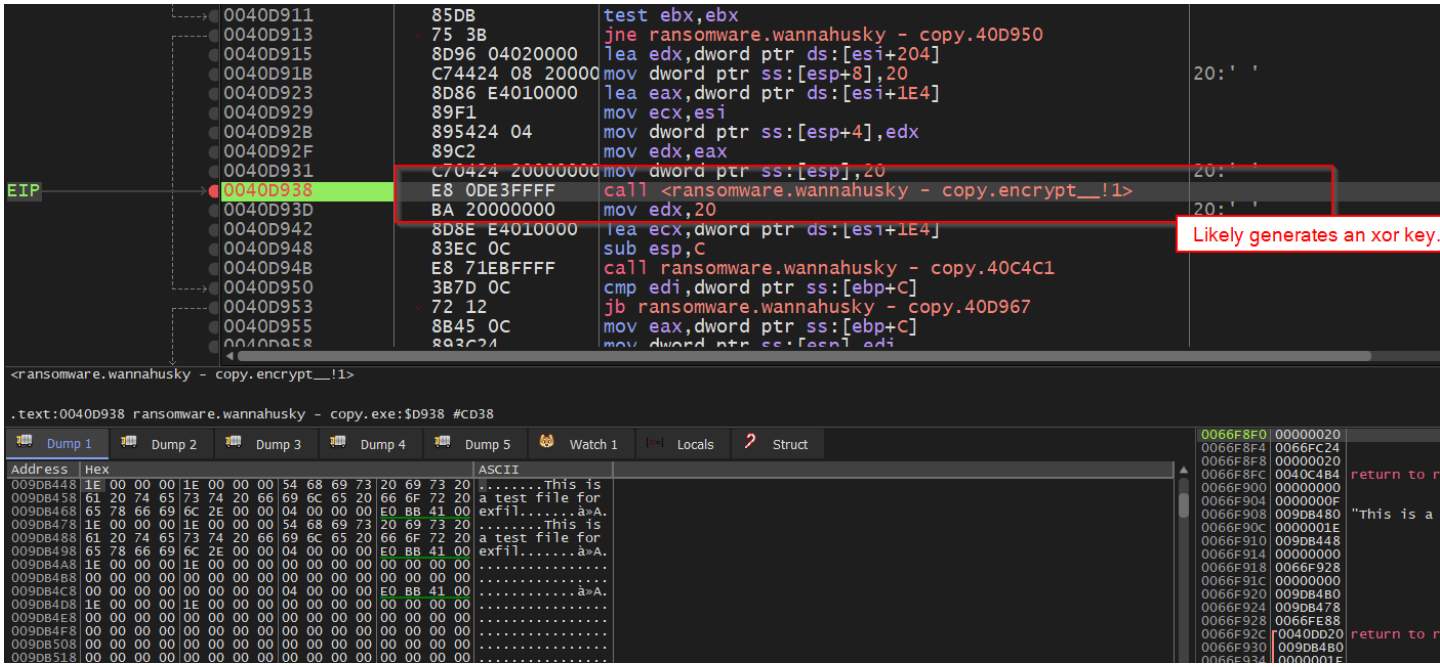


Encryption Routine

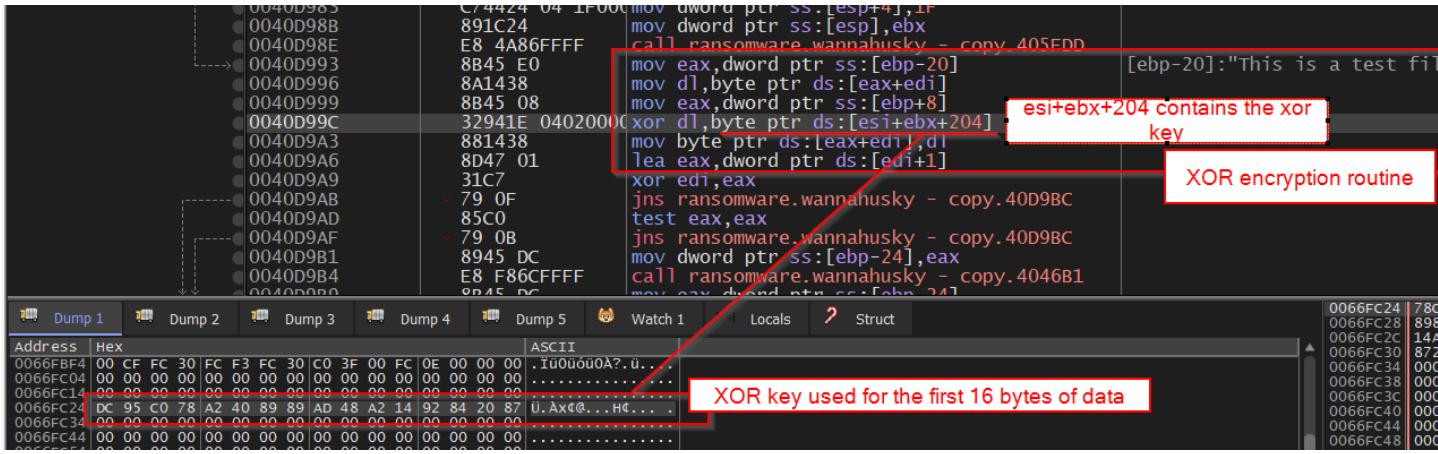
Skipping to the memory location of encryption call "0x0040d8d6" and adding a break point and stepping over each instruction within the call.

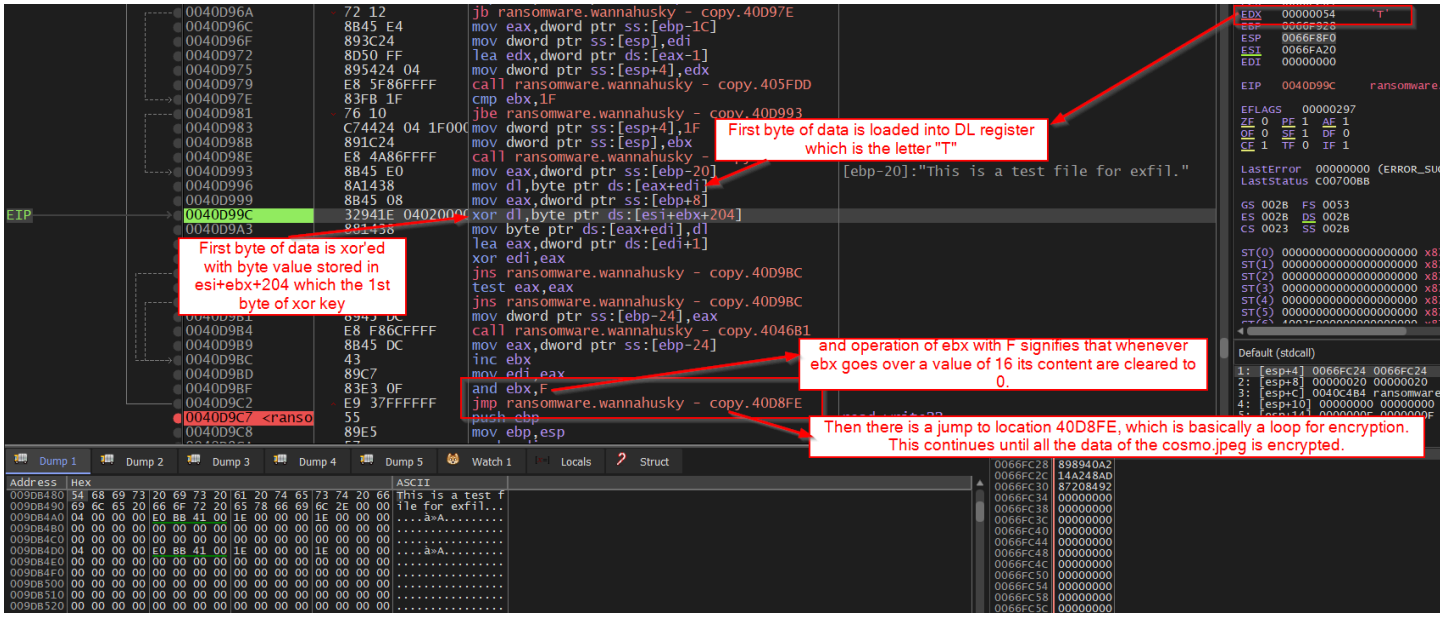


Analyzing the execution flow, firstly there is a call to encrypt__! which does some shift and xor operation. After further review was found to be generating the xor key for encryption.

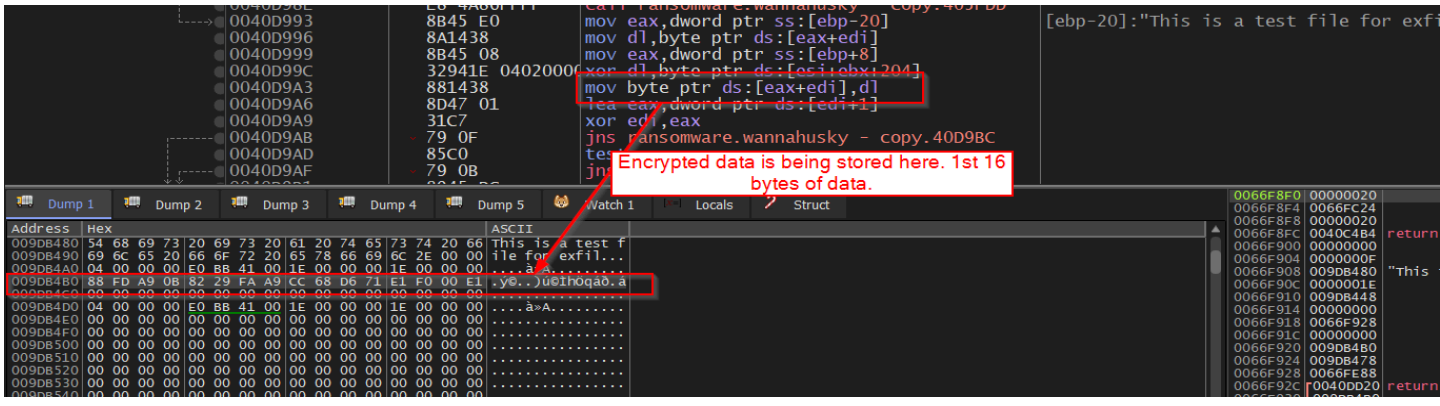


We can see the encryption is carried out within a loop, by performing a bit wise xor operation on each byte of data with each byte of key at a time.

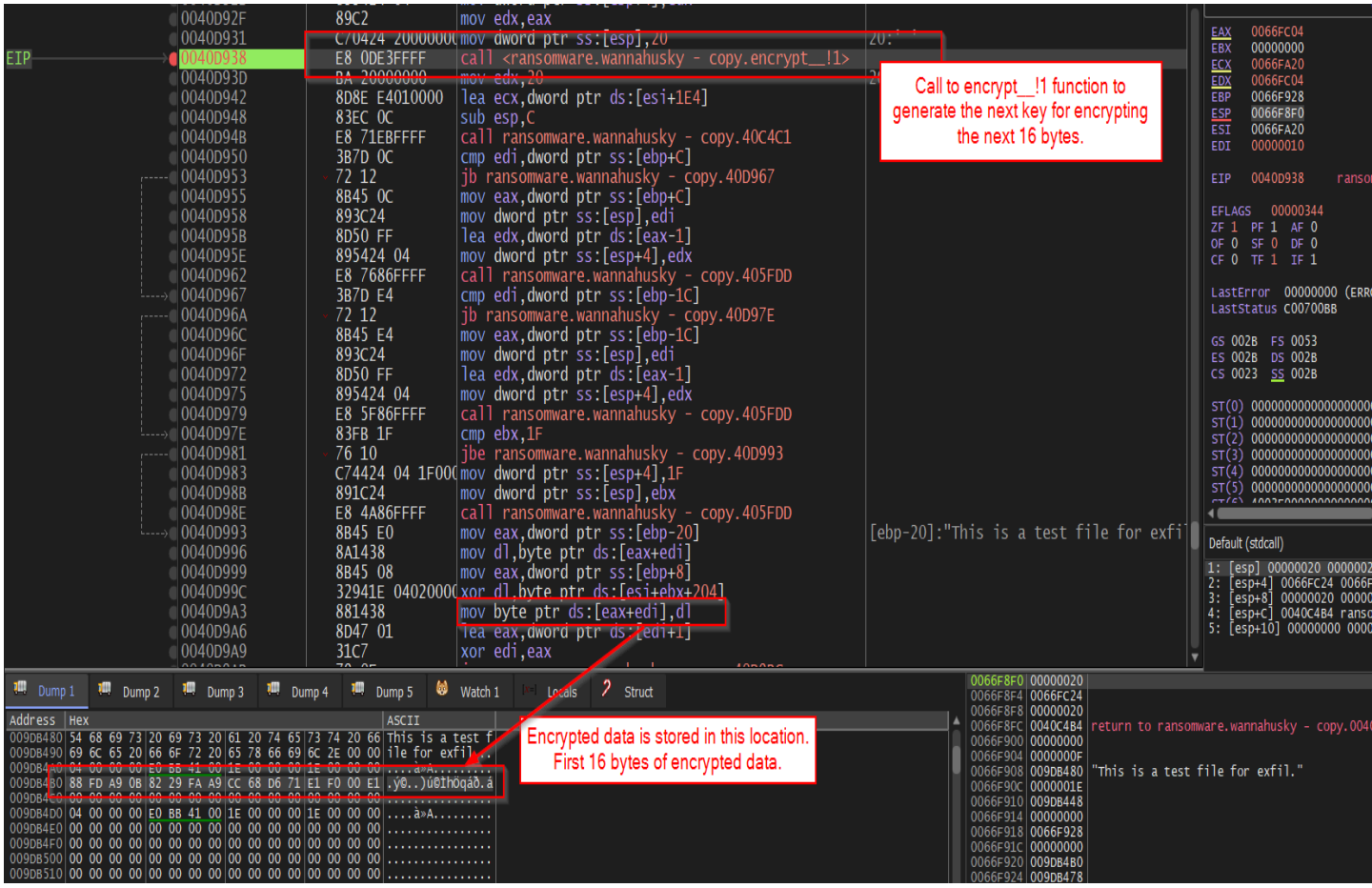




XOR'ed data is stored in the location pointed by eax+edi



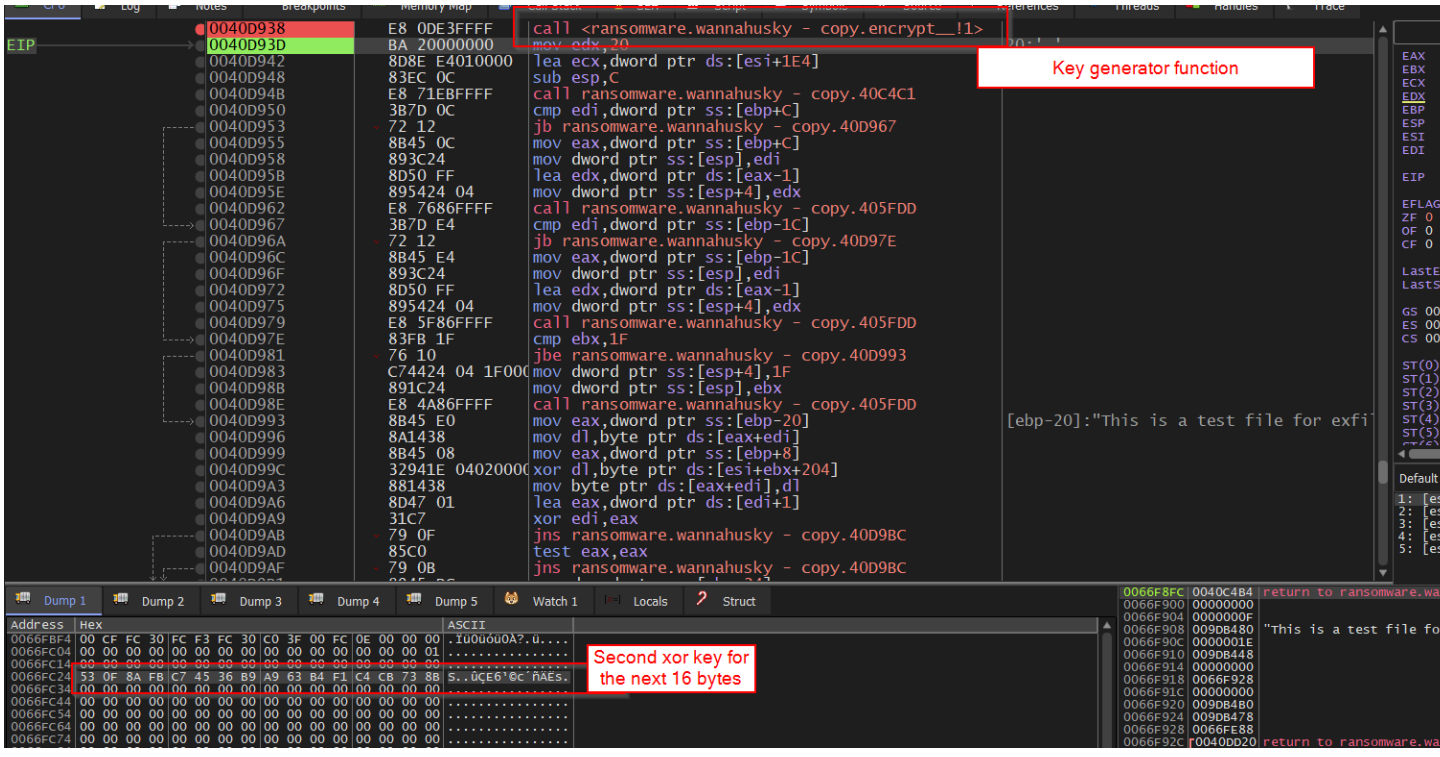
After encrypting the initial 16 bytes of data, there's again a call to keygenerator function. And this new key is used to encrypt the next 16 bytes of data.



Key Generation

First xor key generation :

Second xor key generation :



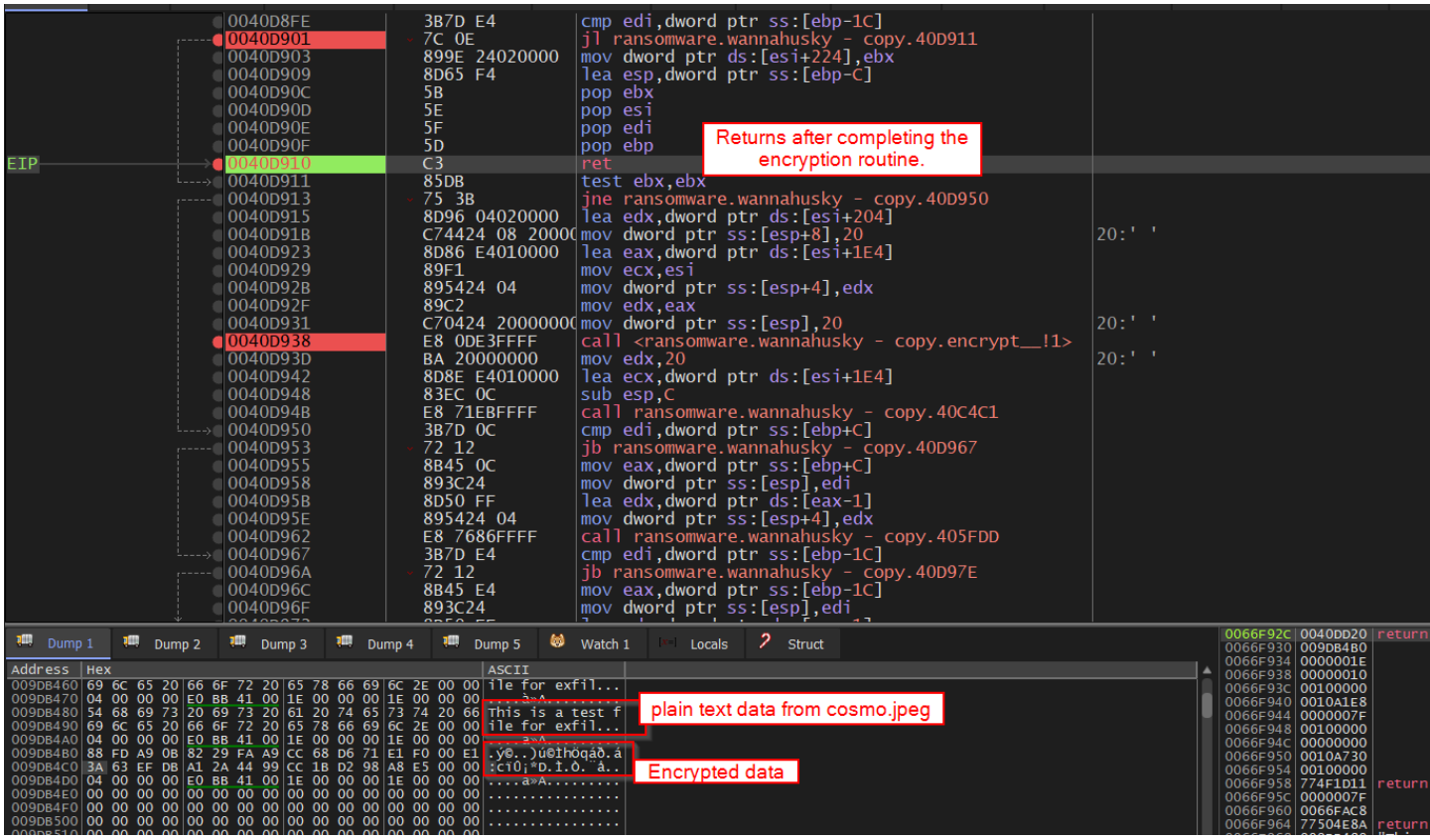
WannaHusky_RansomWare

May 2024

v1.0

Encryption routine conclusion

So to conclude, the program generates a random 16 byte key each round of encryption where 16 bytes data gets encrypted. Implying every 16 byte of data is encrypted with randomly generated 16 byte xor key.

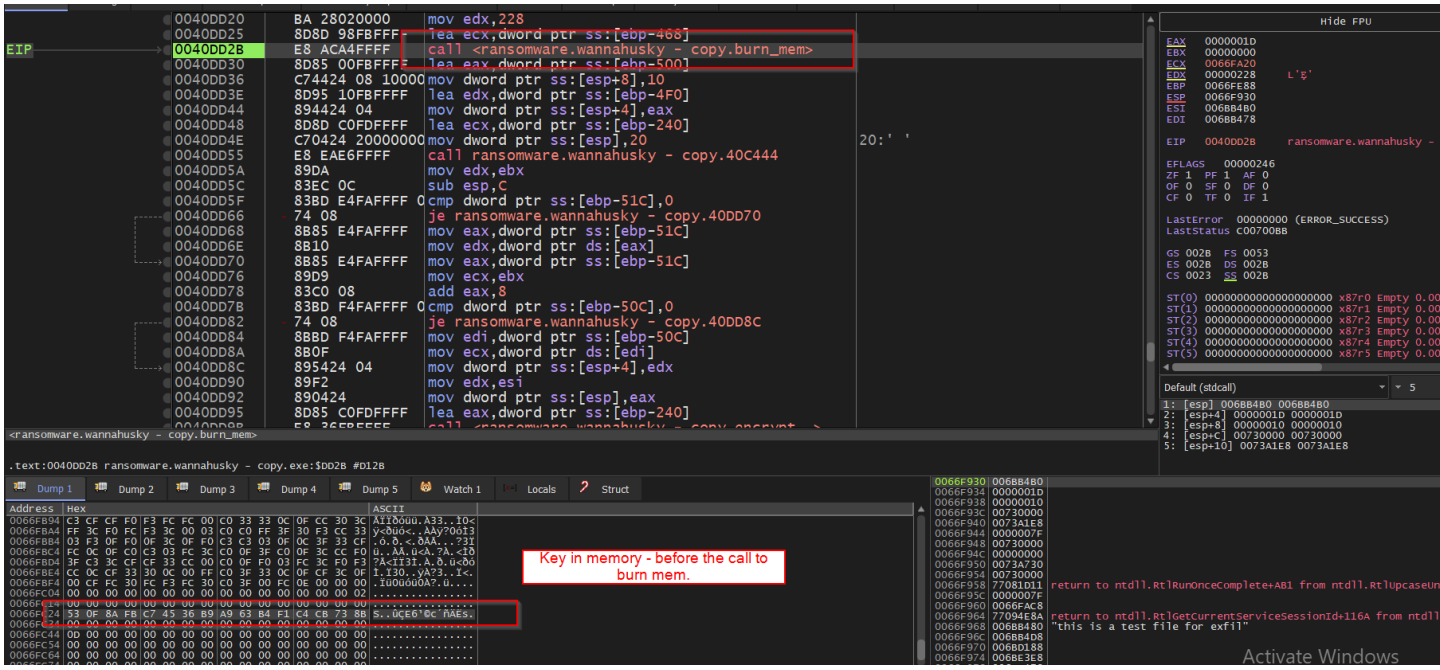


One thing about xor operation is that, if you have the xor key and the encrypted data. Performing another round of xor operation on the encrypted data with the same key will give you the original plain text data.

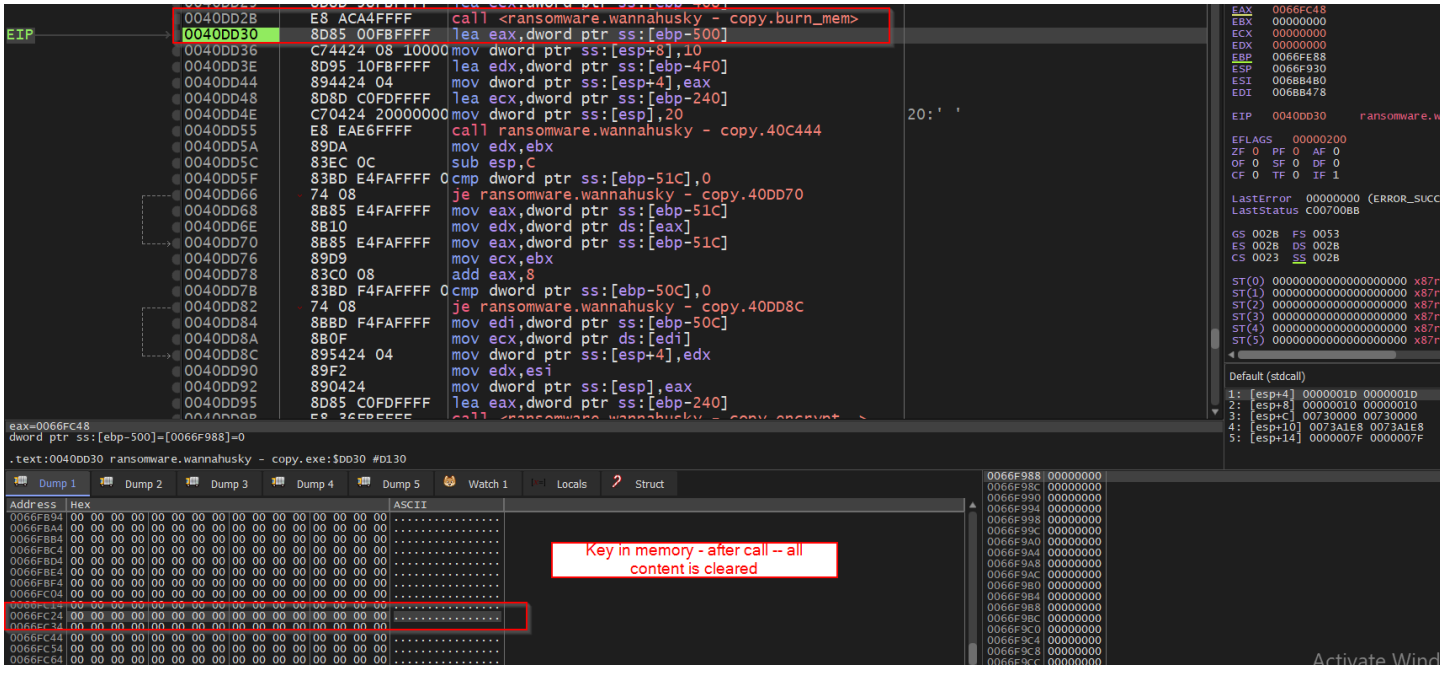
So I saved both the keys from the memory in a file on my desktop so as to decrypt the encrypted data found in the wannahusky file.

After this burnmem function is called which clears the content of key stored in memory.

Before Call :

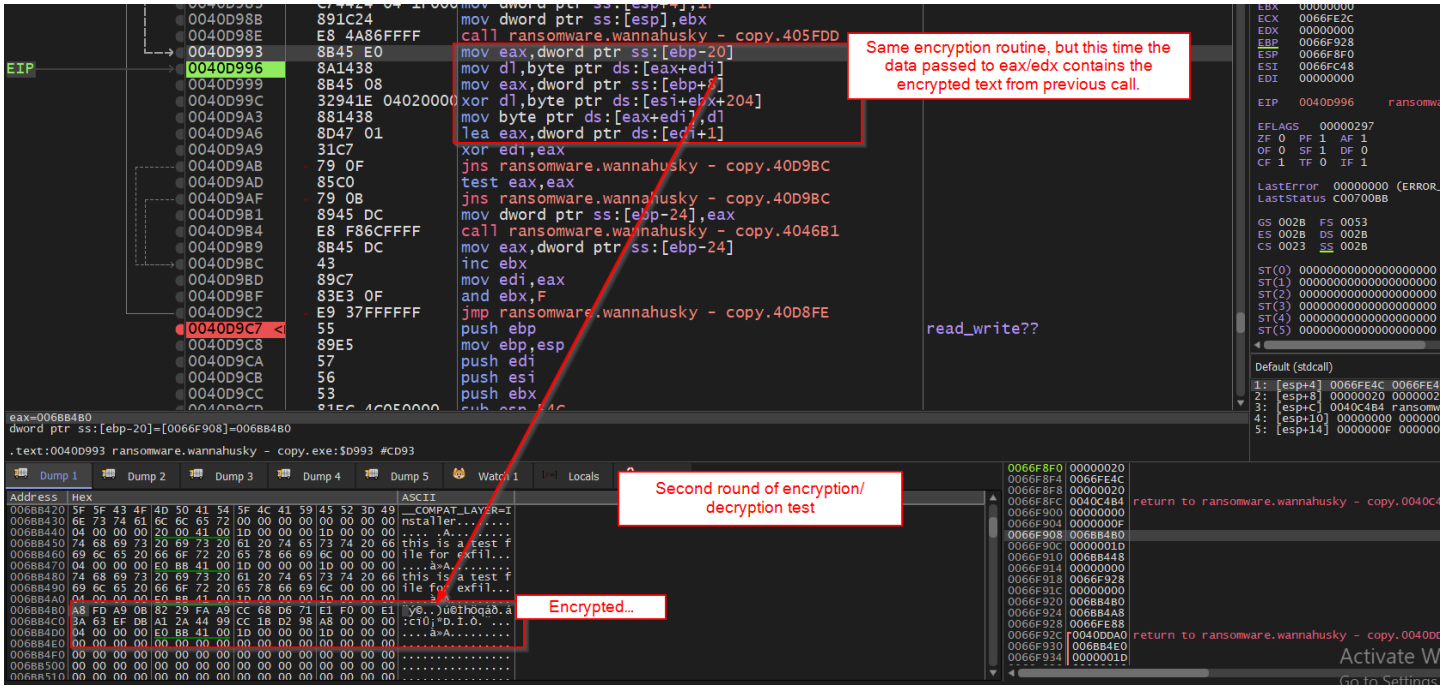


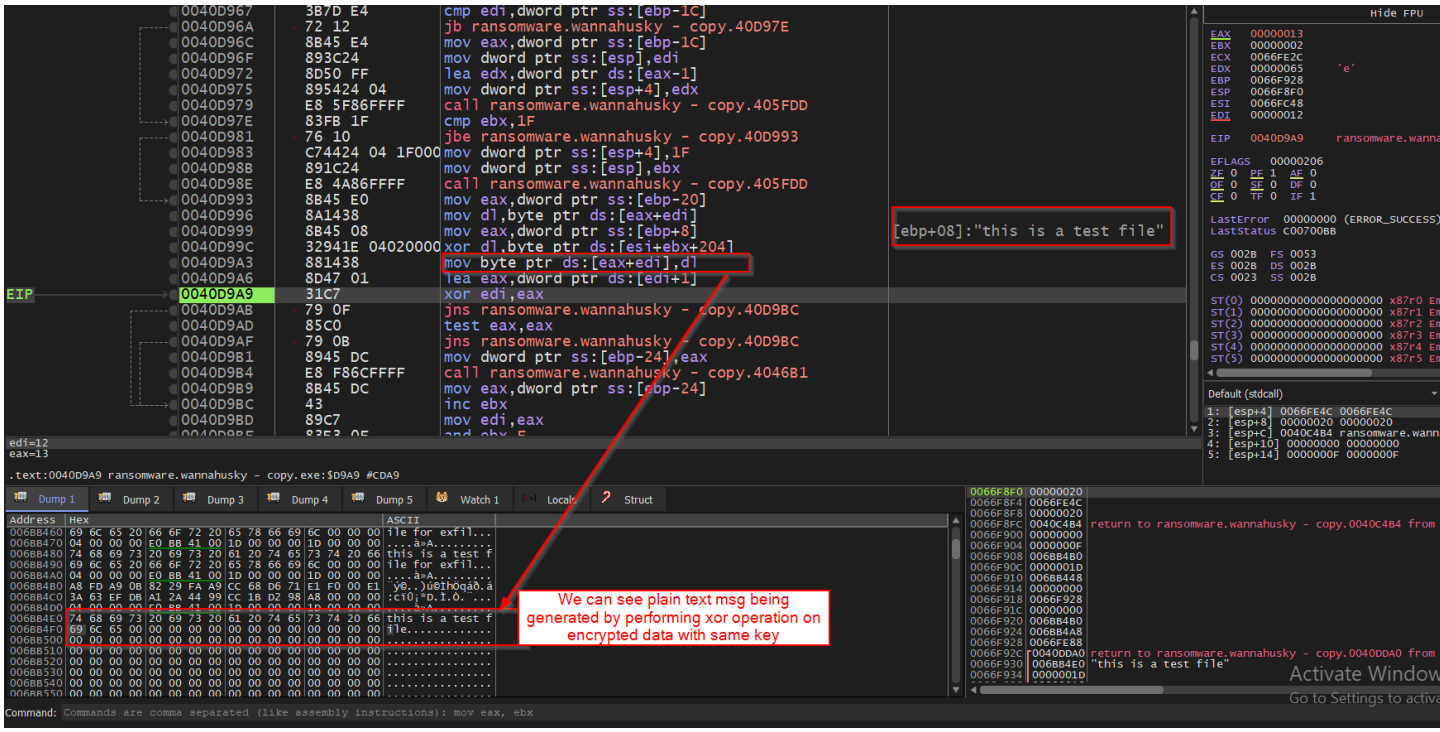
After call :



Encryption validation

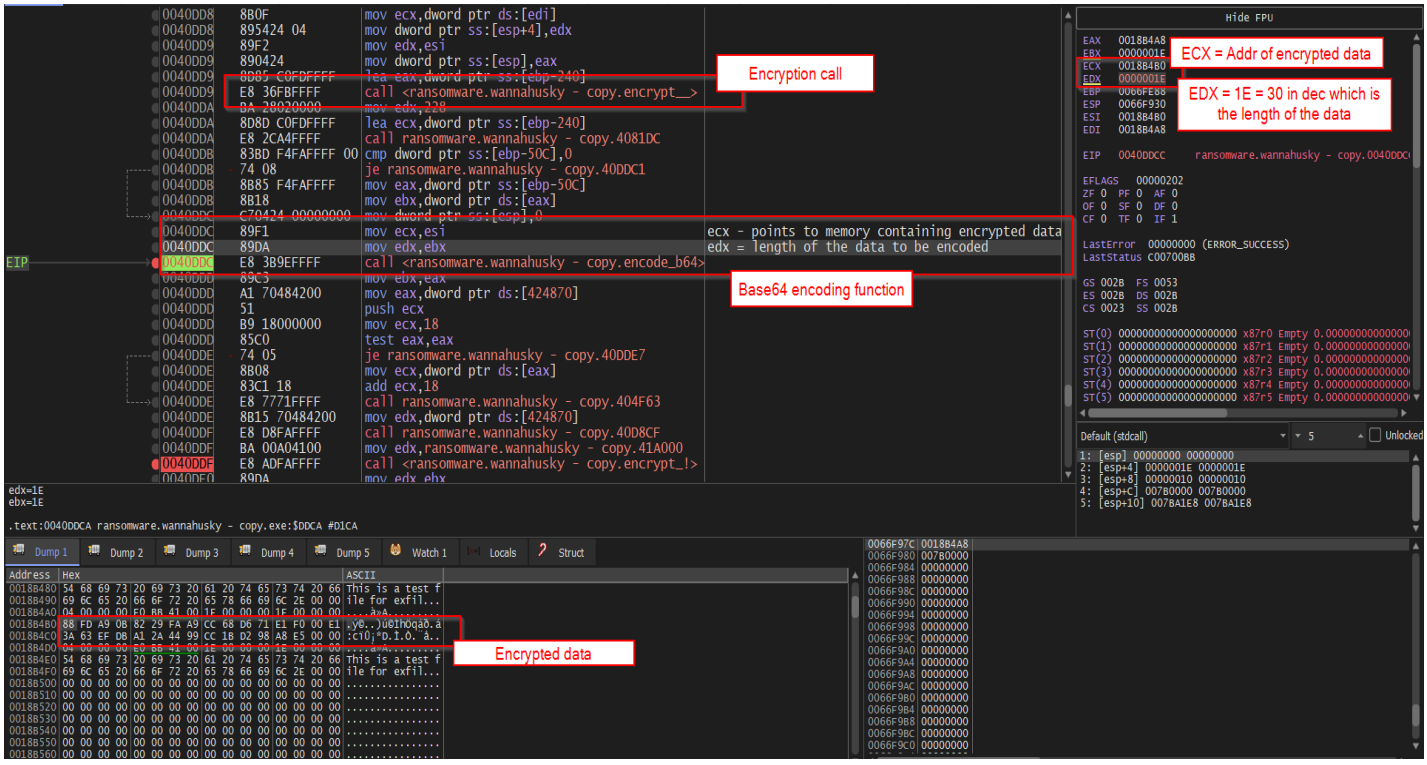
The program calls the encryption routine again, likely to validate the key is correct?? Generates the same set of keys and performs an xor operation on the previously encrypted data which results in the original plain text contents.

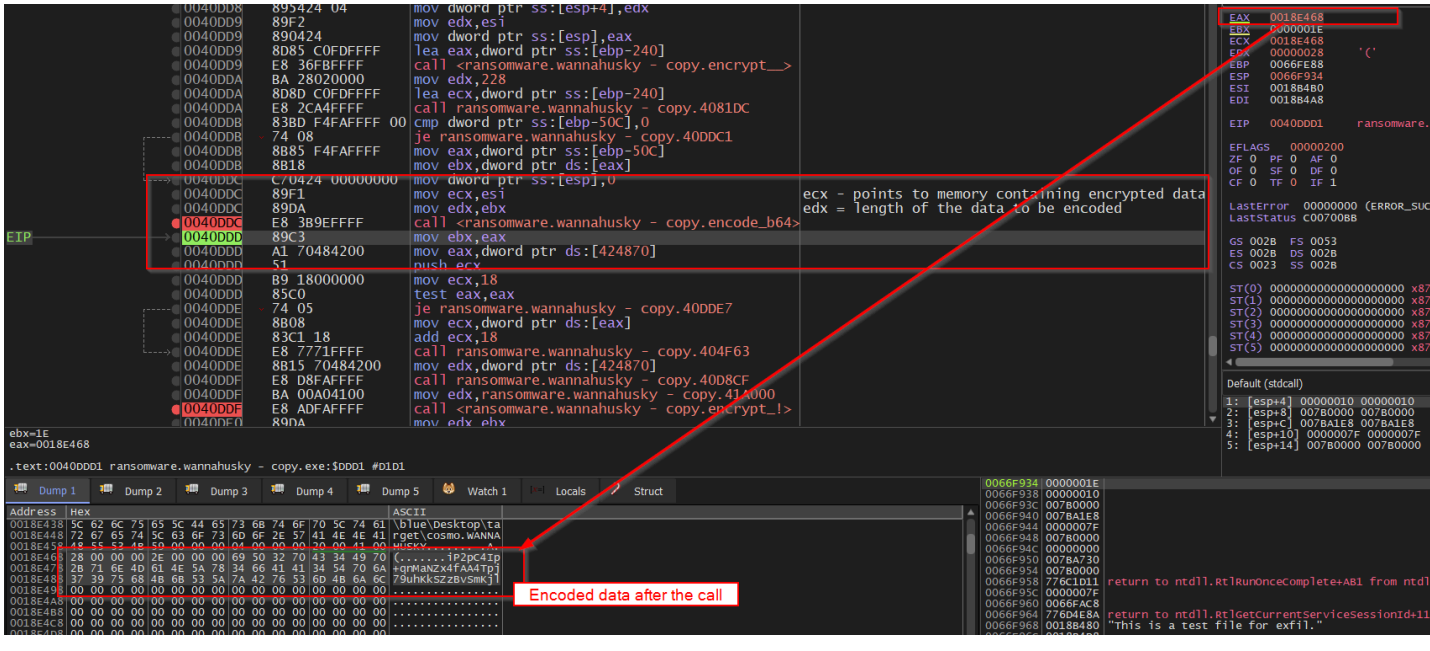




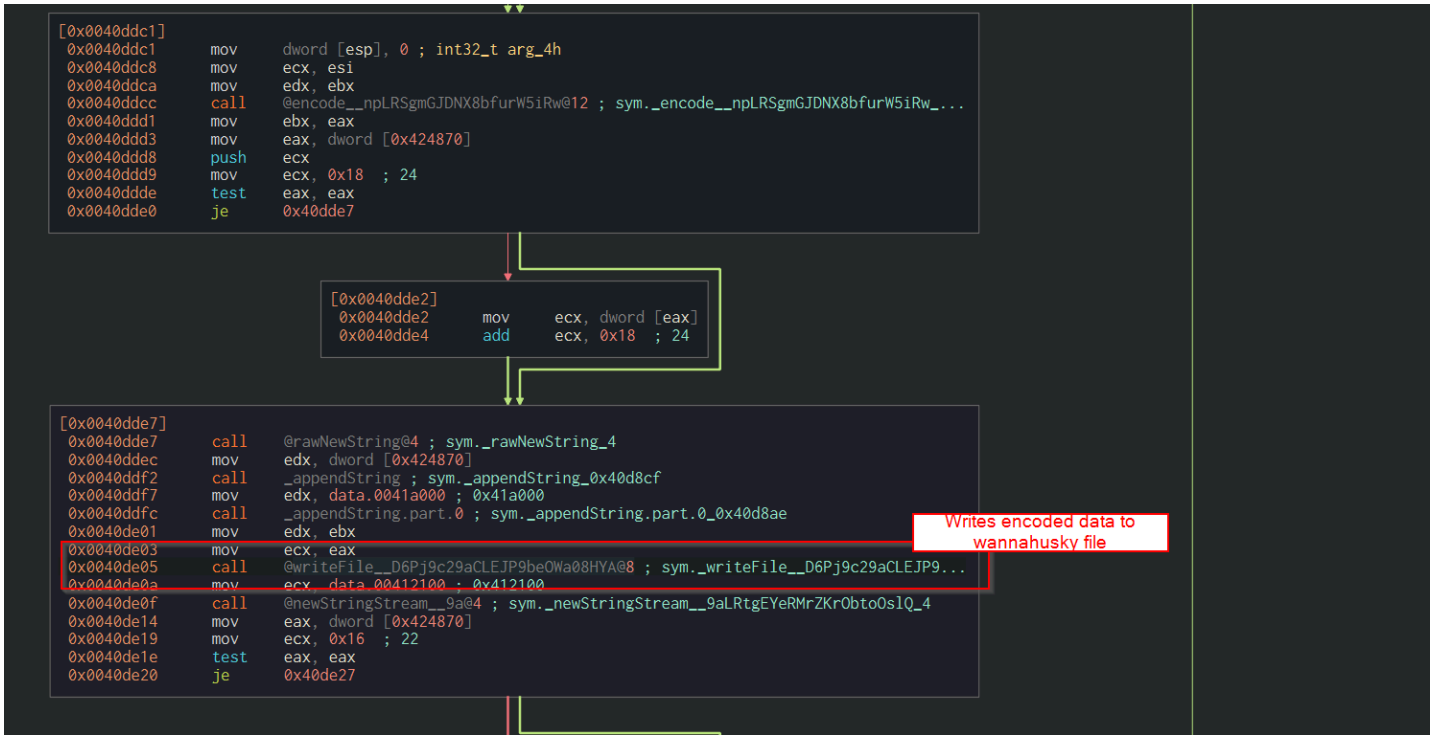
Encode function

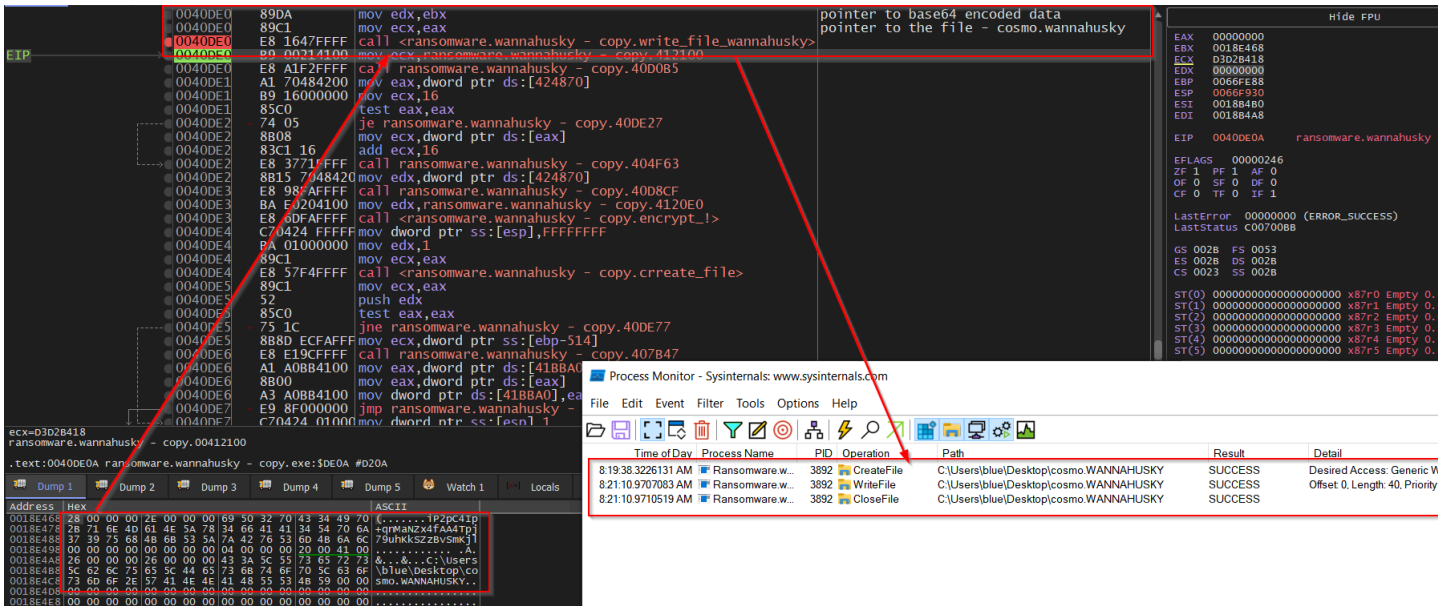
After this call, the encrypted data is passed to another function which converts it into a base64 encoded string.





Then writes this data into cosmo.wannahusky file. The write file function takes two arguments, 1 pointer to encoded data and the other file path of cosmo.wannahusky

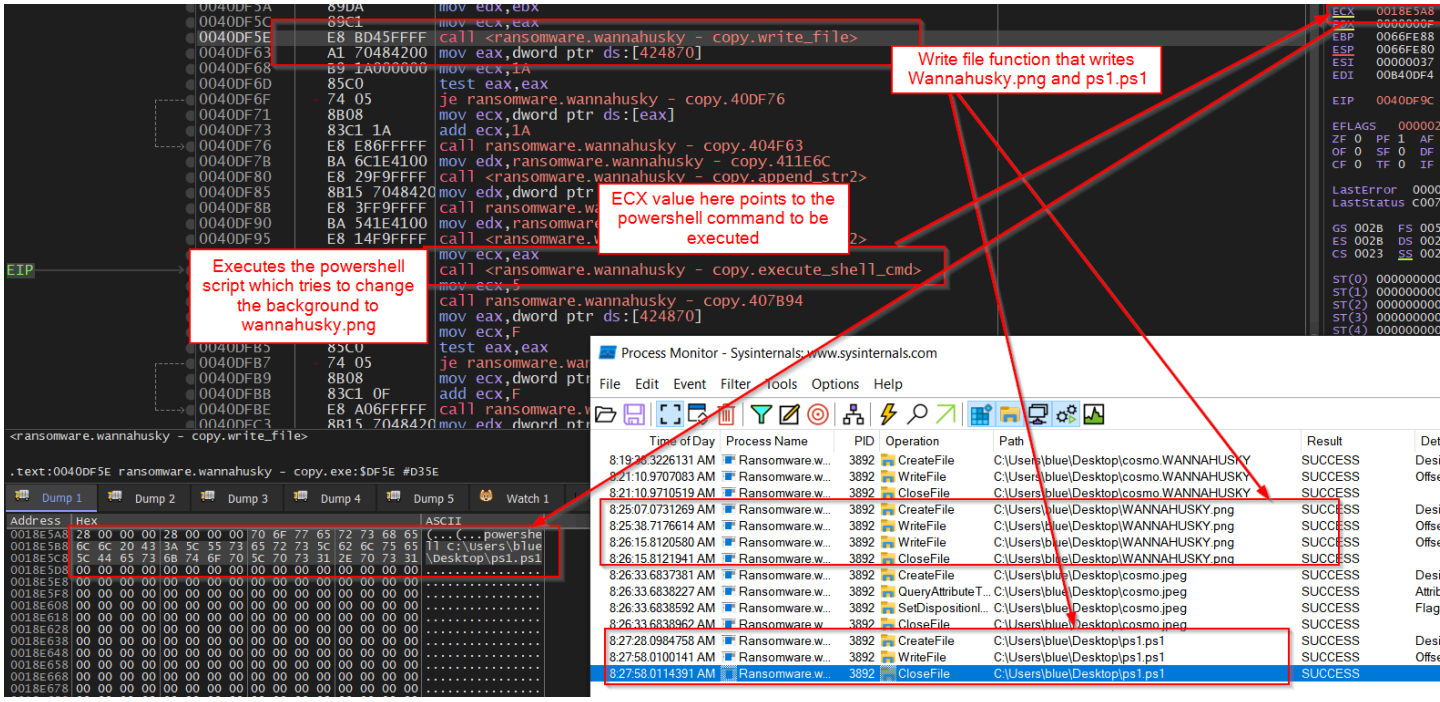


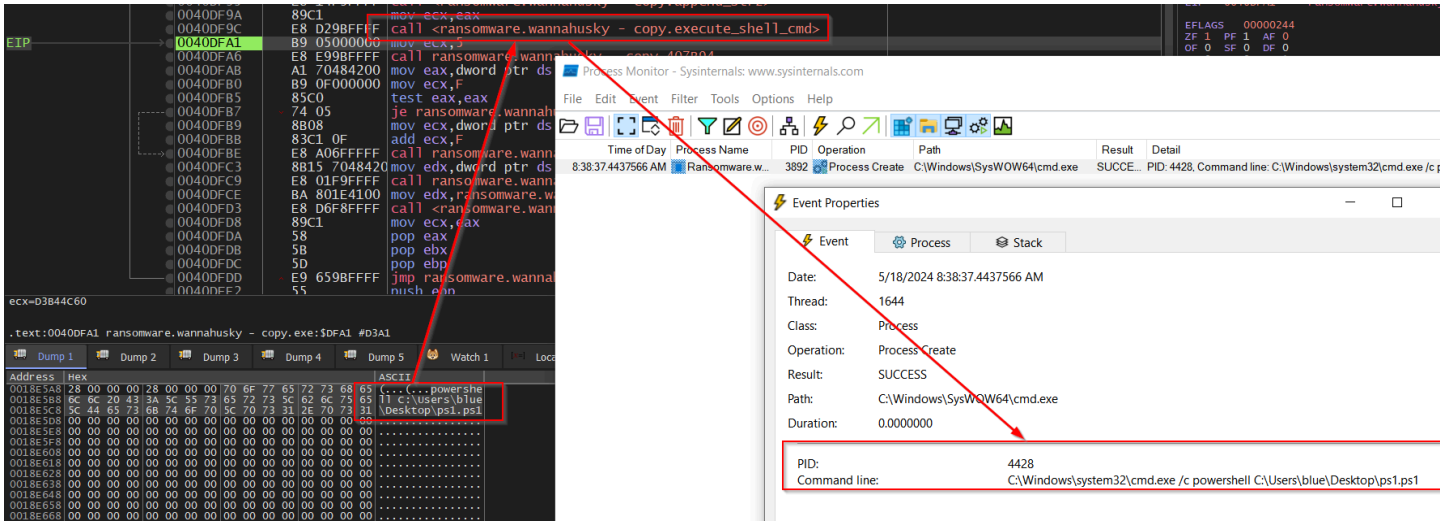


Change Background

Then there is a call to changebackground function which writes the wannahusky.png and ps1.ps1 file on disk and then executes the newly written powershell script.

This powershell script tries to change the background/desktop wallpaper to wannahusky.png.





Dropped files

Powershell script : Ps1.ps1

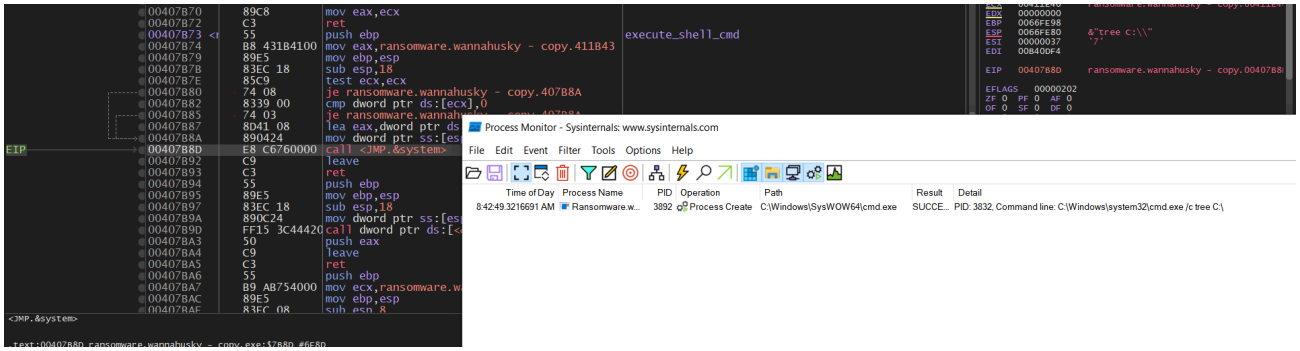
```powershell
$code = @'
using System.Runtime.InteropServices;

namespace Win32{

    public class Wallpaper{

      [DllImport("user32.dll", CharSet=CharSet.Auto)]
      static  extern int SystemParametersInfo (int uAction , int uParam , string lpvParam , int fuWinIni) ;

      public static void SetWallpaper(string thePath){
        SystemParametersInfo(20,0,thePath,3);
      }
    }
}
'@
add-type $code

$currDir = Get-Location
$wallpaper = ".\WANNAHUSKY.PNG"
$fullpath = Join-Path -path $currDir -ChildPath $wallpaper

[Win32.Wallpaper]::SetWallpaper($fullpath)
```

WannaHusky.Png :

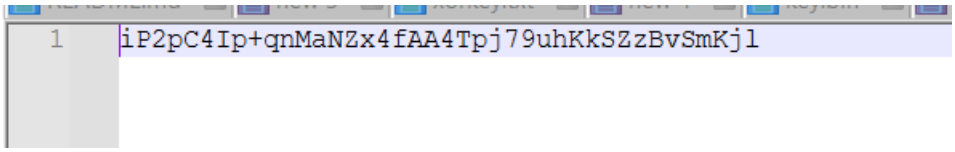
WannaHusky_RansomWare
May 2024
v1.0

Command Execution

And then finally executes cmd /tree C: command that lists all the file system paths for c drive in a tree format -- which is the window that we see during initial detonation.



Copy of the encrypted cosmo file on desktop :

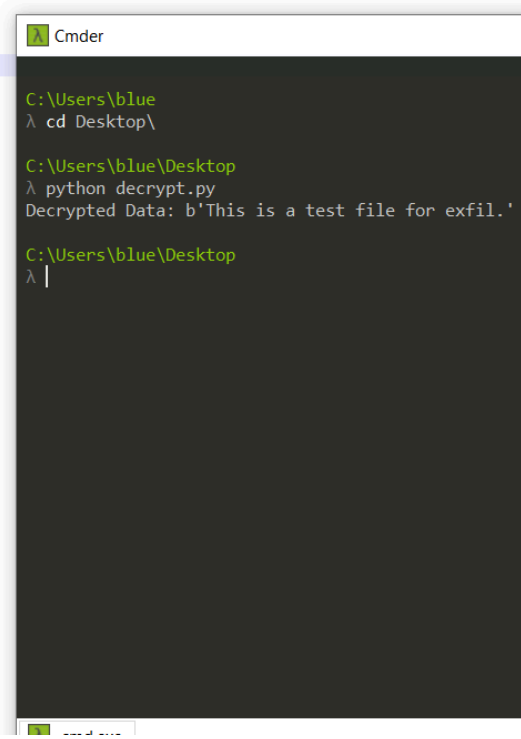


WannaHusky_RansomWare
May 2024
v1.0

Decryption Routine

I saved the keys used for encryption on my desktop.
Writing a simple python script to decrypt the above contents using the saved keys :
Running the script, we can get the encrypted data back :

```python
import base64

def xor_decrypt(data, key):
    decrypted = ''
    for i in range(len(data)):
        decrypted += chr(data[i] ^ key[i % len(key)])
    return decrypted

def main():
    # Base64 encoded xor encrypted data
    encoded_data = b'iP2pC4Ip+qnMaNZx4fAA4Tpj79uhKkSZzBvSmKjl'

    # XOR key
    with open("key1.bin", "rb") as keyfile:
        xor_key_1 = keyfile.read();

    with open("key2.bin", "rb") as keyfile2:
        xor_key_2 = keyfile2.read();

    # Decode the Base64 encoded data
    decoded_data = base64.b64decode(encoded_data)

    # Decrypt the decoded data using XOR
    decrypted_data = xor_decrypt(decoded_data[:16], xor_key_1)
    decrypted_data += xor_decrypt(decoded_data[16:], xor_key_2)

    print("Decrypted Data:", decrypted_data.encode())

if __name__ == "__main__":
    main()
```



Indicators of Compromise

The full list of IOCs can be found in the Appendices.

Network Indicators

{Description of network indicators}

N/A - no network indicators found for this sample.

Host-based Indicators

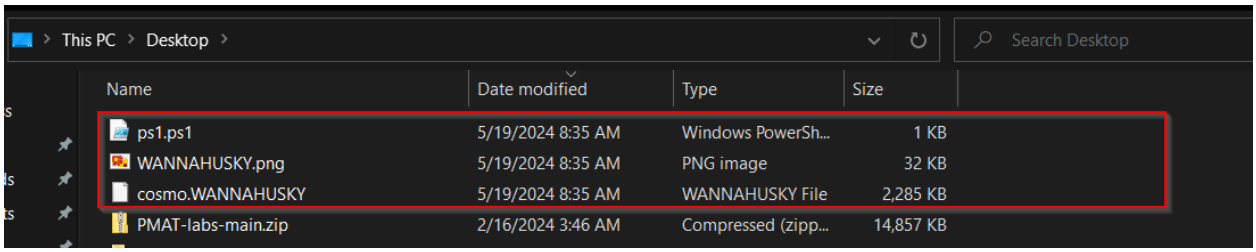
{Description of host-based indicators}

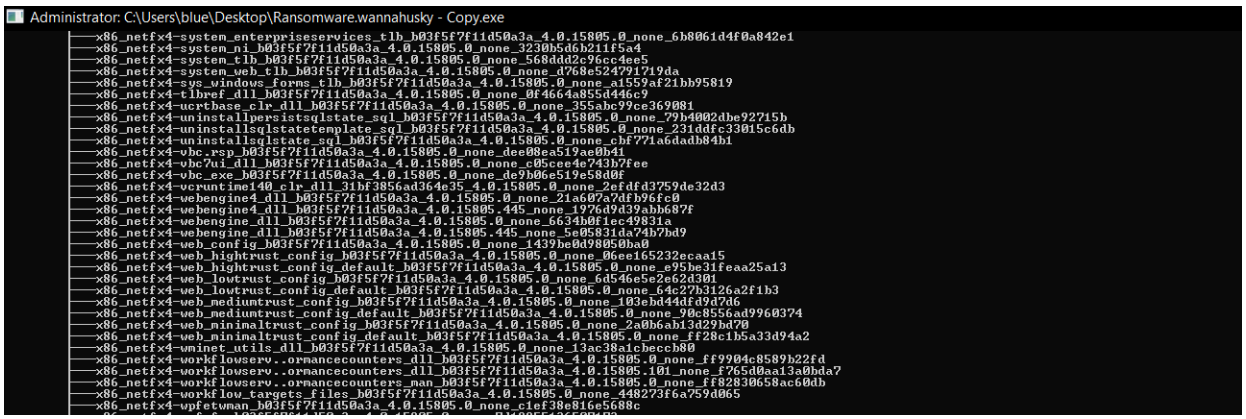
Powershell script - ps1.ps1 ===> changes the background/desktop wallpaper to wannahusky.png

WannaHusky.png ===> image file containing the ransom note

Cosmo.wannahusky ===> encrypted cosmo.jpeg file

Execution of cmd /tree C:/ command ===> likely to hide the powershell window





Rules & Signatures

Yara Rules

```
rule RANSOMWARE_WANNAHUSKY {
    meta:

        author = "Aniksha Shetty"
        description = "Detect malicious ransomware sample, from wannahusky
family"
        created = "05-19-2024"
    strings:
        $target_file = "@Desktop\\cosmo.jpeg" ascii
        $background_file = "\\WANNAHUSKY.PNG" ascii
        $powershell = "@Desktop\\ps1.ps1" ascii
        $encrypted_file = "@Desktop\\cosmo.WANNAHUSKY"
        $base_64 =
"ABCDEFGHIJKLMNOPQRSTUVWXYZabcdefghijklmnopqrstuvwxyz0123456789-_ABCDEFGHI
JKLMNOPQRSTUVWXYZabcdefghijklmnopqrstuvwxyz0123456789+/" ascii
        $compile_lang = "@nim" ascii
        $crypt_api = "bCryptGenRandom" ascii
        $pe_magic_byte = "MZ"

    condition:
        $pe_magic_byte at 0 and $compile_lang and $crypt_api and
        ($background_file or $powershell) or
        ($target_file and $base_64 and $encrypted_file)

}
```