



Practical Malware Analysis & Triage

Malware Analysis Report

WannaHusky_RansomWare

May 2024 | AnikshaShetty | v1.0

Table of Contents

Table of Contents	2
Executive Summary	3
High-Level Technical Summary	4
Execution Flow	4
Malware Composition	5
WannaHusky.png	5
ps1.ps1	5
Basic Static Analysis	6
Floss - String Analysis	6
PEStudio	
This is a 32 bit binary, compiled on oct 12 2021	9
Basic Dynamic Analysis	10
Initial Detonation	10
Advanced Static Analysis	14
Encryption routine call	15
Advanced Dynamic Analysis	17
ReadFile	18
Encryption Routine	19
Key Generation	22
Encryption routine conclusion	23
Encryption validation	25
Encode function	26
Change Background	28
Dropped files	29
Command Execution	30
Decryption Routine	31
Indicators of Compromise	32
Network Indicators	32
Host-based Indicators	32
Rules & Signatures	33
Yara Rules	33

Executive Summary

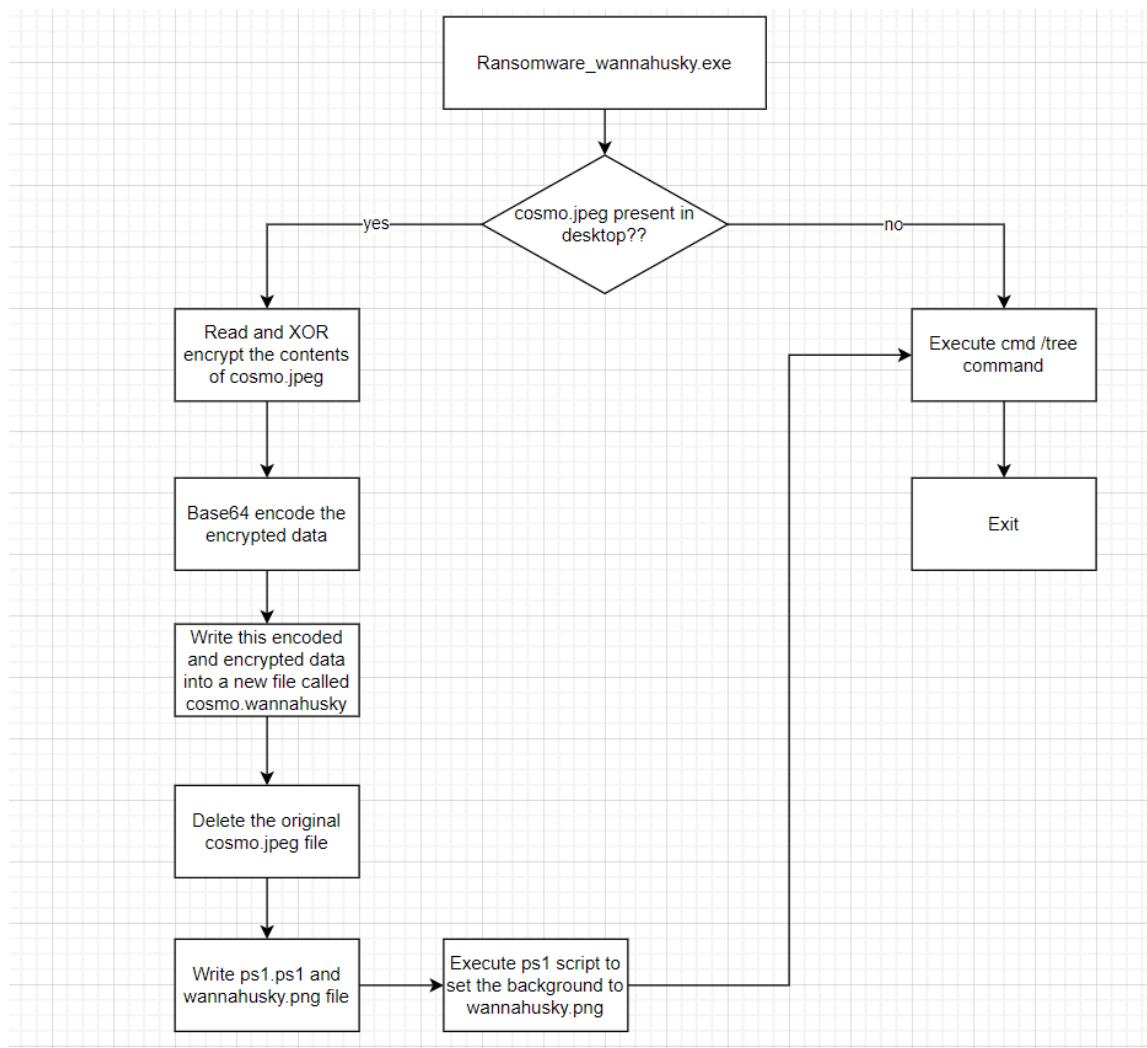
sha256sum	3D35CEBCF40705C23124FDC4656A7F400A316B8E96F1F9E0C187 E82A9D17DCA3
-----------	--

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 lpszParam , 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

```
11 oserr.nim
12 raiseOSError
13 @USERPROFILE
14 @unknown OS error
15 @Additional info:
16 ABCDEFGHIJKLMNOPQRSTUVWXYZabcdefghijklmnopqrstuvwxyz0123456789-_ ABCDEFGHIJKLMNOPQRSTUVWXYZabcdefghijklmnopqrstuvwxyz0123456789+/
17 @bcmode.nim(456, 9) ` 
18 ctx.sizeKey() <= len(key)` 
19 @bcmode.nim(455, 9) ` 
20 ctx.sizeBlock() <= len(iv)` 
21 bCryptGenRandom
22 queryProcessCycleTime
23 queryUnbiasedInterruptTime
24 queryIdleProcessorCycleTime
25 coresCount
26 hIntel
27 IOError
28 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 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
\tpHVs
. . . . .
```



```

4 ?5SZ
5 %OCZIE
6 I|w7
7 z0Gz
8 1RgOB
9 @Z-B
0 IEND
1 @Desktop\cosmo.WANNAHUSKY
2 @bcmode.nim(503, 9) `len(input) <= len(output)` 
3 @COSMO
4 @Desktop\target\cosmo.WANNAHUSKY
5 @Desktop\cosmo.jpeg
6 Unknown error
7 _matherr(): %s in %s(%g, %g)  (retval=%g)
8 Argument domain error (DOMAIN)
9 Argument singularity (SIGN)
0 Overflow range error (OVERFLOW)
1 The result is too small to be represented (UNDERFLOW)
2 Total loss of significance (TLOSS)
3 Partial loss of significance (PLOSS)
4 Mingw-w64 runtime failure:
5 Address %p has no image-section
6     VirtualQuery failed for %d bytes at address %p
7     VirtualProtect failed with code 0x%x
8     Unknown pseudo relocation protocol version %d.
9     Unknown pseudo relocation bit size %d.
```

writeDataImpl

WannaHusky_RansomWare
May 2024
v1.0

```

@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
IpvParam , 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

The screenshot shows the PEStudio interface with the following details:

- File Path:** c:\users\blue\Desktop\ransomware.wannahusky
- Properties:**
 - property value
 - footprint > sha256**: 3D35CEBCF40705C23124FDC4656A7F400A316B8E96F1F9E0C187E82A9D17DCA3
 - first-bytes-hex**: 4D 5A 90 00 03 00 00 00 04 00 00 00 FF FF 00 00 B8 00 00 00 00 00 00 40 00 00 00 00 00 00 00 00
 - first-bytes-text**: MZ ...
 - file > size**: 412905 bytes
 - entropy**: 6.455
 - signature**: n/a
 - tooling**: wait...
 - file-type**: executable
 - cpu**: 32-bit
 - subsystem**: console
 - file-version**: n/a
 - description**: n/a
- stamps:**
 - compiler-stamp**: Tue Oct 12 20:08:25 2021 | UTC
 - debug-stamp: n/a
 - resource-stamp: n/a
 - import-stamp: n/a
 - export-stamp: n/a
- names:**
 - file**: c:\users\blue\Desktop\ransomware.wannahusky.exe
 - debug: n/a
 - export: n/a
 - version: n/a
 - manifest: n/a
 - .NET > module: n/a
 - certificate > program-name: n/a

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

The screenshot shows the PEStudio interface with the following details:

property	value	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	28BAE041F27B62B7321813...	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	0x0001B800	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	0x05000060	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	-	-	-	-	-	

WannaHusky_RansomWare
May 2024
v1.0

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.

```
File Machine View Input Devices Help
C:\Users\blue\Desktop\Ransomware.wannahusky.exe
└── and64_microsoft-windows-media-streaming-dll_31bf3856ad364e35_10.0.19041.1566_none_c315hd50bc4b72da
    └── f
        └── r
            └── and64_microsoft-windows-media-streaming-ps_31bf3856ad364e35_10.0.19041.1_none_fa562f83ace75b88
                └── f
                    └── r
                        └── and64_microsoft-windows-mediafoundation-msfsrv_31bf3856ad364e35_10.0.19041.1865_none_33422b307065eeba
                            └── f
                                └── r
                                    └── and64_microsoft-windows-medialayer-autoplay_31bf3856ad364e35_10.0.19041.1865_none_10e43a01c2238eb1
                                        └── f
                                            └── r
                                                └── and64_microsoft-windows-medialayer-core_31bf3856ad364e35_10.0.19041.1566_none_800f26e104636391
                                                    └── f
                                                        └── r
                                                            └── and64_microsoft-windows-medialayer-logagent_31bf3856ad364e35_10.0.19041.746_none_c93d70420d81ce4
                                                                └── f
                                                                    └── r
                                                                        └── and64_microsoft-windows-medialayer-mls_31bf3856ad364e35_10.0.19041.746_none_4eda1c6d21dd9881
                                                                            └── f
                                                                                └── r
                                                                                    └── and64_microsoft-windows-medialayer-setup_31bf3856ad364e35_10.0.19041.1266_none_22b99d078bbc3016
                                                                                        └── f
                                                                                            └── r
                                                                                                └── and64_microsoft-windows-medialayer-shortcut_31bf3856ad364e35_10.0.19041.1_none_64c27fc7ed12a491
                                                                                             and64_microsoft-windows-medialayer-skins_31bf3856ad364e35_10.0.19041.2006_none_1e3695h18e994dh8
                                                                                                 └── f
                                                                                                     └── r
                                         and64_microsoft-windows-medialayer-vis_31bf3856ad364e35_10.0.19041.1266_none_e5afec1878374111
                                             └── f
                                                 └── r
                                                     └── and64_microsoft-windows-medialayer-wmasf_31bf3856ad364e35_10.0.19041.1_none_5da6fe23dfc593c7
                                                         and64_microsoft-windows-medialayer-wmerror_31bf3856ad364e35_10.0.19041.1_none_ef4600f715653f49d
                                                         and64_microsoft-windows-medialayer-wmnetngr_31bf3856ad364e35_10.0.19041.746_none_253f40e31d7124d4
                                                             └── f
                                                               └── r
                                                               and64_microsoft-windows-medialayer-wmpdgm_31bf3856ad364e35_10.0.19041.1266_none_18fdf6dfdcfd40
                                                               └── f
                                                               and64_microsoft-windows-medialayer-wmpeffects_31bf3856ad364e35_10.0.19041.1266_none_6e13bacd44a30951
                                                               └── f
                                                               and64_microsoft-windows-medialayer-wmpps_31bf3856ad364e35_10.0.19041.1_none_647h5d5hd15acd49
                                                               and64_microsoft-windows-medialayer-wmpshell_31bf3856ad364e35_10.0.19041.1266_none_38802b3df80826dd
                                                               └── f
                                                               and64_microsoft-windows-medialayer-wmvcore_31bf3856ad364e35_10.0.19041.1806_none_7bbf1760f8b3ee704
                                                               └── f
                                                               and64_microsoft-windows-medialayer-wmwdk_31bf3856ad364e35_10.0.19041.1_none_5c1c54e5c5fe9fc0
                                                               and64_microsoft-windows-nfaudioadm_31bf3856ad364e35_10.0.19041.1_none_f1f14e0301d1e1ea
                                                               and64_microsoft-windows-nfvc_31bf3856ad364e35_10.0.19041.1_none_8e7de30963398b48
                                                               and64_microsoft-windows-nfaacenc_31bf3856ad364e35_10.0.19041.1_none_289h214h917ff7ff?
                                                               and64_microsoft-windows-nfasfsrcsnk_31bf3856ad364e35_10.0.19041.1865_none_3502fc1af1f86813
                                                               └── f
                                                               and64_microsoft-windows-nfaudionv_31bf3856ad364e35_10.0.19041.746_none_e77e4c60d41c1b03
                                                               └── f
                                                               and64_microsoft-windows-nfc42x.resources_31bf3856ad364e35_10.0.19041.1_en-us_7892c6682e6258c8
                                                               and64_microsoft-windows-nfc42x_31bf3856ad364e35_10.0.19041.1456_none_d10e1541b45e0391
                                                               └── f
                                                               and64_microsoft-windows-nfcore_31bf3856ad364e35_10.0.19041.1_en-us_57921162b99398b6
                                                               and64_microsoft-windows-nfcore_31bf3856ad364e35_10.0.19041.2086_none_55ae19bd0c581hd6
                                                               └── f
                                                               and64_microsoft-windows-nfds_31bf3856ad364e35_10.0.19041.1645_none_1a270410d2d089fe
                                                               └── f
                                                               and64_microsoft-windows-nfddude_31bf3856ad364e35_10.0.19041.1_none_bp380hf72297h48
                                                               and64_microsoft-windows-nfsql_31bf3856ad364e35_10.0.19041.1_none_BF52e28281e1449f
                                                               and64_microsoft-windows-nf1263enc_31bf3856ad364e35_10.0.19041.1_none_4052cf3d3a53273
                                                               and64_microsoft-windows-nf1264enc_31bf3856ad364e35_10.0.19041.1886_none_fee09de9e9af2b5e
                                                               └── f
                                                               and64_microsoft-windows-nfmjpegdec_31bf3856ad364e35_10.0.19041.1348_none_8e16cd37092ach5a
                                                               └── f
                                                               and64_microsoft-windows-nfmkvsrcsnk_31bf3856ad364e35_10.0.19041.1566_none_jb95e3ad4277b3e4d
```

WannaHusky_RansomWare

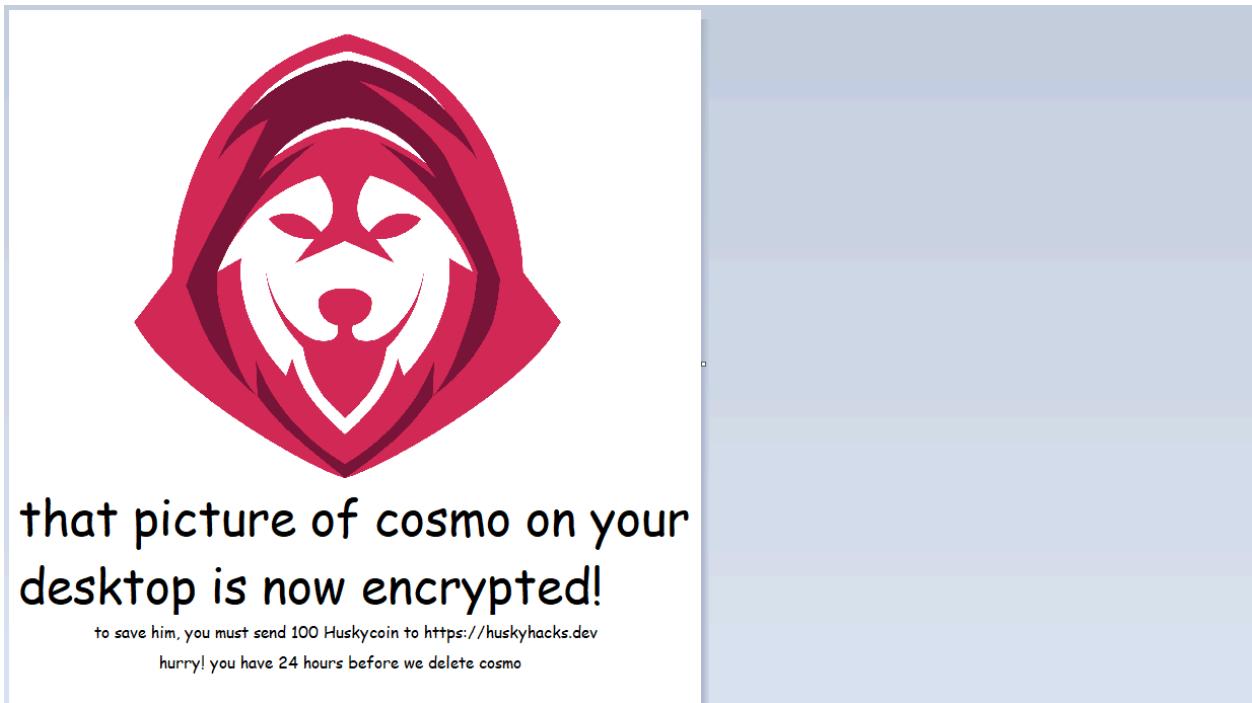
May 2024

v1.0

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 tee 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



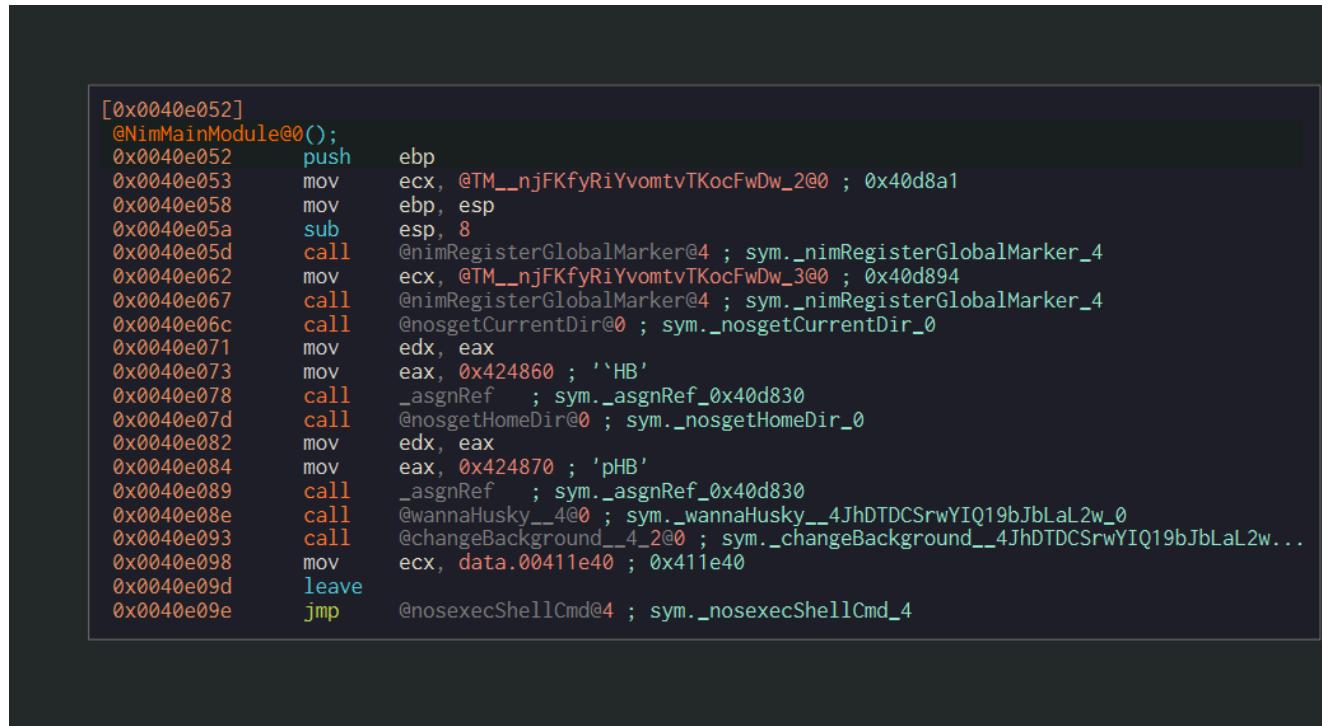
WannaHusky_RansomWare
May 2024
v1.0

Advanced Static Analysis

{Screenshots and description about findings during advanced static analysis}

Loading the binary in cutter :

Main program of the binary starts here :



The screenshot shows the assembly code for the main program entry point. The assembly code is as follows:

```
[0x0040e052]
@NimMainModule@0();
0x0040e052    push    ebp
0x0040e053    mov     ecx, @TM__njFKfyRiYvomtvTKocFwDw_2@0 ; 0x40d8a1
0x0040e058    mov     ebp, esp
0x0040e05a    sub     esp, 8
0x0040e05d    call    @nimRegisterGlobalMarker@4 ; sym._nimRegisterGlobalMarker_4
0x0040e062    mov     ecx, @TM__njFKfyRiYvomtvTKocFwDw_3@0 ; 0x40d894
0x0040e067    call    @nimRegisterGlobalMarker@4 ; sym._nimRegisterGlobalMarker_4
0x0040e06c    call    @nosgetCurrentDir@0 ; sym._nosgetCurrentDir_0
0x0040e071    mov     edx, eax
0x0040e073    mov     eax, 0x424860 ; `HB'
0x0040e078    call    _asgnRef ; sym._asgnRef_0x40d830
0x0040e07d    call    @nosgetHomeDir@0 ; sym._nosgetHomeDir_0
0x0040e082    mov     edx, eax
0x0040e084    mov     eax, 0x424870 ; 'pHB'
0x0040e089    call    _asgnRef ; sym._asgnRef_0x40d830
0x0040e08e    call    @wannaHusky__4@0 ; sym._wannaHusky__4JhDTDCSrwYIQ19bJbLaL2w_0
0x0040e093    call    @changeBackground__4_2@0 ; sym._changeBackground__4JhDTDCSrwYIQ19bJbLaL2w...
0x0040e098    mov     ecx, data.00411e40 ; 0x411e40
0x0040e09d    leave
0x0040e09e    jmp     @nosexecShellCmd@4 ; sym._nosexecShellCmd_4
```

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.

```

Graph(sym_wannaHusky_4JhDTDCSrvYIQ19bJbLaL2w_0)

void sym_wannaHusky_4JhDTDCSrvYIQ19bJbLaL2w_0();
{
    0x0040da52    test    eax, eax
    0x0040da54    je      0x40da5b
    [0x0040da56]
    0x0040da56    mov     ecx, dword [eax]
    0x0040da58    add     ecx, 0x1f ; 31
    [0x0040da5b]
    0x0040da5b    call    @rawNewString@4 ; sym._rawNewString_4
    0x0040da60    mov     edx, dword [0x424870]
    0x0040da66    call    _appendString ; sym._appendString_0x40d8cf
    0x0040da6b    mov     edx, data.00410a0 ; 0x41a0a0
    0x0040da70    call    _appendString.part.0 ; sym._appendString.part.0_0x40d8ae
    0x0040da75    mov     ecx, dword [var_518h]
    0x0040da7b    call    @readFile_404 ; sym._readFile_4PGnM9bWmsH0Nu7dnr3XzgA_4
    0x0040da80    mov     ecx, data.00410a0 ; 0x41a0a0
    0x0040da85    mov     esi, eax
    0x0040da87    call    @copyString@4 ; sym._copyString_4
    0x0040da8c    lea     edx, [var_46ch]
    0x0040da92    mov     ecx, 0x8a ; 138
    0x0040da97    mov     edi, edx
    0x0040da99    mov     dword [var_51ch], eax
    0x0040da9f    lea     edx, [var_244h]
    0x0040daa5    xor     eax, eax
    0x0040daa7    rep    stosd dword es:[edi], eax
    0x0040daa9    mov     edi, edx
    0x0040daab    mov     ecx, 0x8a ; 138
    0x0040dab0    lea     edx, [var_4f4h]
    0x0040dab6    rep    stosd dword es:[edi], eax
    0x0040dab8    mov     edi, edx
    0x0040daba    mov     ecx, 8
    0x0040dabf    lea     edx, [var_504h]
    0x0040dac5    rep    stosd dword es:[edi], eax
    0x0040dac7    mov     ecx, 4
    0x0040dacc    mov     edi, edx
}

```

Dashboard | Strings | Imports | Search | Disassembly | Graph(sym_wannaHusky_4JhDTDCSrvYIQ19bJbLaL2w_0) | Hexdump | Decompiler (dbg.mainCRTStartup)

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

```

void sym_wannaHusky_4JhDTDCSrvYIQ19bJbLaL2w_0();
{
    0x0040dcf2    cmp    dword [var_514h], 0
    0x0040dcf9    mov    ecx, ebx
    0x0040dcfb    lea    esi, [edi + 8]
    0x0040dcfe    je    0x40dd08
    [0x0040dd00]
    0x0040dd00    mov    edi, dword [var_514h]
    0x0040dd06    mov    ecx, dword [edi]
    [0x0040dd08]
    0x0040dd08    mov    dword [var_568h], eax ; int32_t arg_4h
    0x0040dd0c    mov    edx, dword [var_524h]
    0x0040dd12    lea    eax, [var_46ch]
    0x0040dd18    mov    dword [esp], esi ; int32_t arg_8h
    0x0040dd1b    call   _encrypt_dcoBdmUaaCC9cnR23eFxSLAbmode ; sym._encrypt_dcoBdmUaaCC9cnR23...
    0x0040dd20    mov    edx, 0x20 ; 32
    0x0040dd25    lea    ecx, [var_45ch]
    0x0040dd2b    call   @burnMem_4F08 ; sym._burnMem_4FZHz34TGoTmMy6XY9c0Sg_8
    0x0040dd30    lea    eax, [var_520h]
    0x0040dd36    mov    dword [var_564h], 0x10 ; 16 ; int32_t arg_4h
    0x0040dd3e    lea    edx, [var_4f4h]
    0x0040dd44    mov    dword [var_568h], eax ; int32_t arg_4h
    0x0040dd48    lea    ecx, [var_244h]
    0x0040dd4e    mov    dword [esp], 0x20 ; 32 ; int32_t arg_ch
    0x0040dd55    call   @_init_QeKvRTxwnkv4EgDHKgXYA_20 ; sym._init_QeKvRTxwnkv4EgDHKgXYA_20
    0x0040dd5a    mov    edx, ebx
    0x0040dd5c    sub    esp, 0xc
    0x0040dd5f    cmp    dword [var_520h], 0
    0x0040dd66    je    0x40dd70
    [0x0040dd68]
    0x0040dd68    mov    eax, dword [var_520h]
    0x0040dd6e    mov    edx, dword [eax]
}

```

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_py6wg79aBw8iTzUm11Z7J0A@20 (0x20, 0x20, edx);
            edx = 0x20;
            ecx = esi + 0x1e4;
            @inc128_vRz5m42Fv3KwSYgATX55Q@8 ();
        }
        if (edi >= arg_8h) {
            eax = arg_8h;
            edx = eax - 1;
            _raiseIndexError2 (edi, edx);
        }
        if (edi >= var_20h) {
            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);
}
```

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

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

The screenshot shows the assembly view of a debugger. The assembly code is as follows:

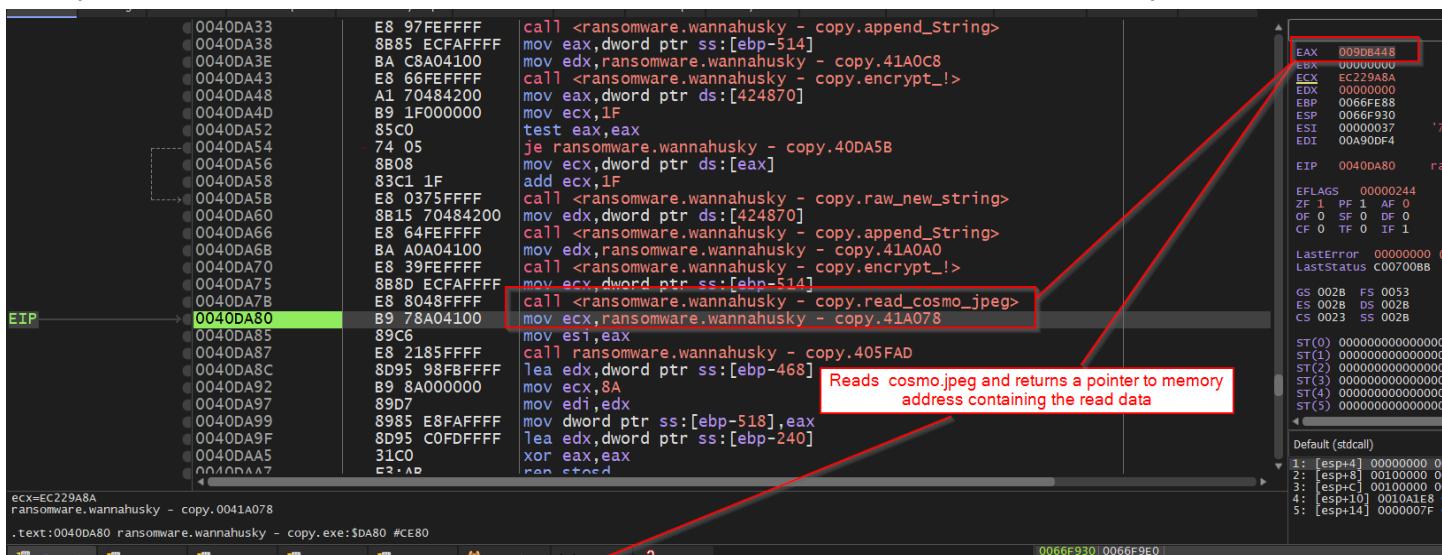
```
EIP: 0040E000 55 push ebp
      0040E001 B9 A1D84000 mov ecx, ransomware.wannahusky - copy.40D8A1
      0040E002 89E5 mov ebp, esp
      0040E003 83EC 08 sub esp, 8
      0040E004 E8 177EFFFF call <ransomware.wannahusky - copy.reg_global_marker>
      0040E005 B9 94D84000 mov ecx, ransomware.wannahusky - copy.40D894
      0040E006 E8 0D7EFFFF call <ransomware.wannahusky - copy.reg_global_marker>
      0040E007 E8 2497FFFF call <ransomware.wannahusky - copy.get_current_dir>
      0040E008 89C2 mov edx, eax
      0040E009 B8 60484200 mov eax, ransomware.wannahusky - copy.424860
      0040E00A E8 B3F7FFFF call <ransomware.wannahusky - copy.unknown>
      0040E00B E8 199AFFFF call <ransomware.wannahusky - copy.get_home_dir>
      0040E00C 89C2 mov edx, eax
      0040E00D B8 70484200 mov eax, ransomware.wannahusky - copy.424870
      0040E00E E8 A2F7FFFF call <ransomware.wannahusky - copy.unknown>
      0040E00F E8 34F9FFFF call <ransomware.wannahusky - copy.read_write??>
      0040E010 E8 84FEFFFF call <ransomware.wannahusky - copy.change_background>
      0040E011 B9 401E4100 mov ecx, ransomware.wannahusky - copy.411E40
      0040E012 C9 leave
      0040E013 E9 D09AFFFF jmp <ransomware.wannahusky - copy.execute_shell_cmd>
      0040E014 EB AD jmp ransomware.wannahusky - copy.40E052
      0040E015 90 nop
      0040E016 90 nop
      0040E017 90 nop
      0040E018 66:90 nop
      0040E019 66:90 nop
      0040E01A 66:90 nop
```

Registers and memory dump tabs are visible on the right side of the debugger interface.

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.

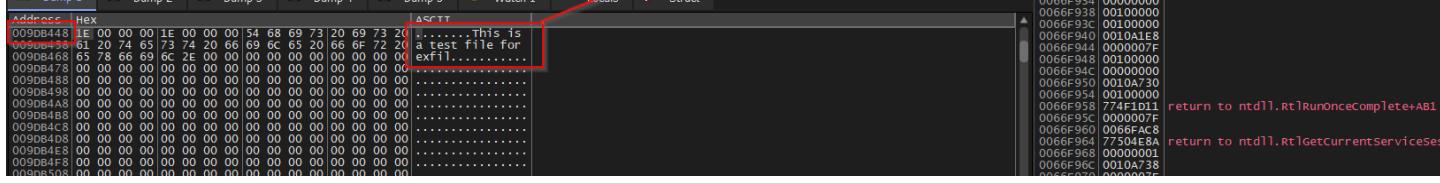


```

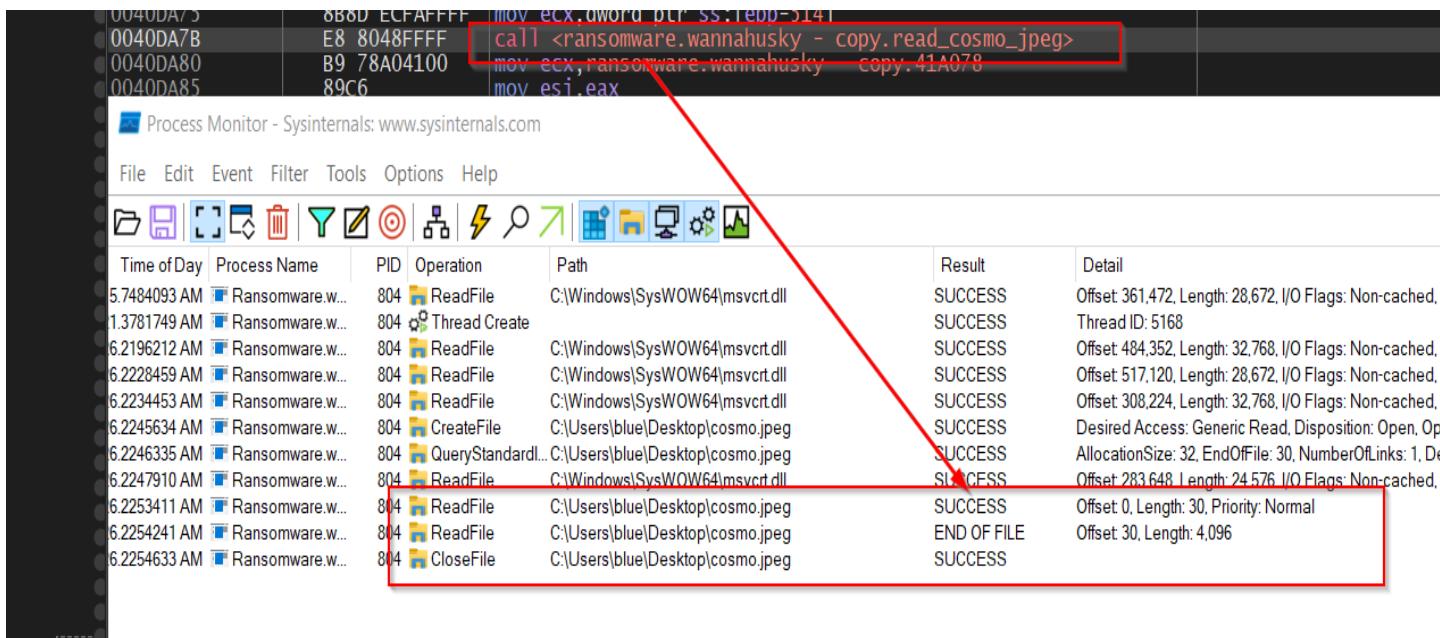
0040DA33 E8 97FFFF call <ransomware.wannahusky - copy.append_String>
0040DA38 8B85 ECFAFFFF mov eax,dword ptr ss:[ebp-514]
0040DA3E BA C8A04100 mov edx, ransomware.wannahusky - copy.41A0C8
0040DA43 E8 66FFFFFF call <ransomware.wannahusky - copy.encrypt_!>
0040DA48 A1 70484200 mov eax,dword ptr ds:[424870]
0040DA4D B9 1F000000 mov ecx,1F
0040DA52 85C0 test eax,eax
0040DA54 74 05 je ransomware.wannahusky - copy.40DAB
0040DA56 8B08 mov ecx,dword ptr ds:[eax]
0040DA58 83C1 1F add ecx,1F
0040DA5B E8 0375FFFF call <ransomware.wannahusky - copy.raw_new_string>
0040DA60 8B15 70484200 mov edx,dword ptr ds:[424870]
0040DA66 E8 64FFFFFF call <ransomware.wannahusky - copy.append_String>
0040DA6B BA A00A04100 mov edx, ransomware.wannahusky - copy.41A0A0
0040DA70 E8 39FFFFFF call <ransomware.wannahusky - copy.encrypt_!>
0040DA75 88D0 ECFAFFFF mov eax,dword ptr ss:[ebp-514]
0040DA7B E8 8048FFFF call <ransomware.wannahusky - copy.read_cosmo_jpeg>
0040DA80 B9 78A04100 mov ecx, ransomware.wannahusky - copy.41A078
0040DA85 89C6 mov esi,eax
0040DA87 E8 2185FFFF call ransomware.wannahusky - copy.405FAD
0040DA8C 8D95 98FBFFFF Teax edx,dword ptr ss:[ebp-468]
0040DA92 B9 A0000000 mov ecx,8A
0040DA97 89D7 mov edi,edx
0040DA99 8985 E8FAFFFF mov dword ptr ss:[ebp-518],eax
0040DA9F 8D95 C0FFF mov edx,dword ptr ss:[ebp-240]
0040DAA5 31C0 xor eax,eax
0040DAAB 83C4 01000000 rep stosd
0040DAAB 31C0 xor eax,eax
0040DAAB 83C4 01000000 rep stosd

eax=EC229A8A
ransomware.wannahusky - copy.0041A078
.text:0040DA80 ransomware.wannahusky - copy.exe:$0A80 #CE80

```



Address	Hex	ASCII
009DB448	LE 00 00 00 01 E0 00 00 54 68 69 73 20 69 73 24 00 20 74 00 65 73 24 20 05 69 00 65 20 66 0F 00 00 00	... This is a test file for exfil.....
009DB453	65 08 66 69 6C 2E 00	
009DB478	00 00	
009DB483	00 00	
009DB498	00 00	
009DB4A3	00 00	
009DB4D8	00 00	
009DB4E3	00 00	
009DB4F8	00 00	
009DB503	00 00	



Time of Day	Process Name	PID	Operation	Path	Result	Detail
5.7484093 AM	Ransomware.w...	804	ReadFile	C:\Windows\SysWOW64\msvcrtdll	SUCCESS	Offset: 361,472, Length: 28,672, I/O Flags: Non-cached,
1.3781749 AM	Ransomware.w...	804	Thread Create		SUCCESS	Thread ID: 5168
6.2196212 AM	Ransomware.w...	804	ReadFile	C:\Windows\SysWOW64\msvcrtdll	SUCCESS	Offset: 484,352, Length: 32,768, I/O Flags: Non-cached,
6.2228459 AM	Ransomware.w...	804	ReadFile	C:\Windows\SysWOW64\msvcrtdll	SUCCESS	Offset: 517,120, Length: 28,672, I/O Flags: Non-cached,
6.2234453 AM	Ransomware.w...	804	ReadFile	C:\Windows\SysWOW64\msvcrtdll	SUCCESS	Offset: 308,224, Length: 32,768, I/O Flags: Non-cached,
6.2245634 AM	Ransomware.w...	804	CreateFile	C:\Users\blue\Desktop\cosmo.jpeg	SUCCESS	Desired Access: Generic Read, Disposition: Open, Op...
6.2246335 AM	Ransomware.w...	804	QueryStandardI...	C:\Users\blue\Desktop\cosmo.jpeg	SUCCESS	AllocationSize: 32, EndOfFile: 30, NumberOfLinks: 1, De...
6.2247910 AM	Ransomware.w...	804	ReadFile	C:\Windows\SysWOW64\msvcrtdll	SUCCESS	Offset: 283,648, Length: 24,576, I/O Flags: Non-cached,
6.2253411 AM	Ransomware.w...	804	ReadFile	C:\Users\blue\Desktop\cosmo.jpeg	SUCCESS	Offset: 0, Length: 30, Priority: Normal
6.2254241 AM	Ransomware.w...	804	ReadFile	C:\Users\blue\Desktop\cosmo.jpeg	END OF FILE	Offset: 30, Length: 4,096
6.2254633 AM	Ransomware.w...	804	CloseFile	C:\Users\blue\Desktop\cosmo.jpeg	SUCCESS	

Encryption Routine

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

```

0040D8D3 EB D9 jmp <ransomware.wannahusky - copy.encrypt>
0040D8D5 C3 ret
0040D8D6 <rando> 55 push ebp
0040D8D7 89E5 mov ebp,esp
0040D8D9 57 push edi
0040D8DA 56 push esi
0040D8DB 89C6 mov esi,eax
0040D8DD 53 push ebx
0040D8DE 83EC 2C sub esp,2C
0040D8E1 8955 E0 mov dword ptr ss:[ebp-20],edx
0040D8E4 894D E4 mov dword ptr ss:[ebp-1C],ecx
0040D8E7 3B4D 0C cmp ecx,dword ptr ss:[ebp+C]
0040D8EA 7E OA jle ransomware.wannahusky - copy.40D8F6
0040D8EC B9 40A04100 mov ecx,ransomware.wannahusky - copy.41A
0040D8F1 E8 BE3CFFFF call ransomware.wannahusky - copy.4015B4
0040D8F6 8B9E 24020000 mov ebx,dword ptr ds:[esi+224]
0040D8FC 31FF xor edi,edi
0040D8FE 3B7D E4 cmp edi,dword ptr ss:[ebp-1C]
0040D901 7C 0E j1 ransomware.wannahusky - copy.40D911
0040D903 899E 24020000 mov dword ptr ds:[esi+224],ebx
0040D906 89E5 E1 lea esp,dword ptr ss:[ebp+C]

```

Beginning of encrypt function

ebp=0066F928

.text:0040D8D6 ransomware.wannahusky - copy.exe:\$D8D6 #CCD6 <encrypt__>

Address	Hex	ASCII	Comments
009DB448	1E 00 00 00 1E 00 00 00 54 68 69 73 20 69 73 20This is	
009DB458	61 20 74 65 73 74 20 66 69 6C 65 20 66 6F 72 20	a test file for	
009DB468	65 78 66 69 6C 2E 00 04 00 00 00 E0 BB 41 00	exfil.....à»A	
009DB478	1E 00 00 00 1E 00 00 00 54 68 69 73 20 69 73 20This is	
009DB488	61 20 74 65 73 74 20 66 69 6C 65 20 66 6F 72 20	a test file for	
009DB498	65 78 66 69 6C 2E 00 04 00 00 00 E0 BB 41 00	exfil.....à»A	
009DB4A8	1E 00 00 00 1E 00 00 00 00 00 00 00 00 00 00 00	
009DB4B8	00 00 00 00 00 00 00 00 04 00 00 E0 BB 41 00à»A	

Address	Hex	ASCII	Comments
0066F8E0	00000020		return to ransomware.wanna
0066F8F4	0066FC24		
0066F8F8	00000020		
0066F8F0	0040C4B4		
0066F900	00000000		
0066F904	0000000F		
0066F908	009DB480	"This is a test file for	
0066F910	0000001E		
0066F914	00000000		
0066F918	0066F928		
0066F91C	00000000		

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.

```

0040D911 85DB test ebx,ebx
0040D913 75 3B jne ransomware.wannahusky - copy.40D950
0040D915 8D96 04020000 lea edx,dword ptr ds:[esi+204]
0040D918 C74424 08 20000 mov dword ptr ss:[esp+8],20
0040D923 8D86 E4010000 lea eax,dword ptr ds:[esi+1E4]
0040D929 89F1 mov ecx,esi
0040D92B 895424 04 mov dword ptr ss:[esp+4],edx
0040D92F 89C2 mov edx,eax
0040D931 C70424 20000000 mov dword ptr ss:[esp],20
0040D932 E8 0DE3FFFF call <ransomware.wannahusky - copy.encrypt__!1>
0040D933 BA 20000000 mov edx,20
0040D934 8D8E E4010000 lea ecx,dword ptr ds:[esi+1E4]
0040D935 83EC 0C sub esp,C
0040D936 E8 71EBFFFF call ransomware.wannahusky - copy.40C4C1
0040D937 3B7D 0C cmp edi,dword ptr ss:[ebp+C]
0040D938 72 12 jb ransomware.wannahusky - copy.40D967
0040D939 8B45 0C mov eax,dword ptr ss:[ebp+C]
0040D93A 89C2 01 mov dword ptr ss:[ebp+C],edi

```

Likely generates an xor key.

EIP

.text:0040D938 ransomware.wannahusky - copy.exe:\$D938 #CD38

Address	Hex	ASCII	Comments
009DB448	1E 00 00 00 1E 00 00 00 54 68 69 73 20 69 73 20This is	
009DB458	61 20 74 65 73 74 20 66 69 66 65 20 66 6F 72 20	a test file for	
009DB468	65 78 66 69 6C 2E 00 04 00 00 00 E0 BB 41 00	exfil.....à»A	
009DB478	1E 00 00 00 1E 00 00 00 54 68 69 73 20 69 73 20This is	
009DB488	61 20 74 65 73 74 20 66 69 66 65 20 66 6F 72 20	a test file for	
009DB498	65 78 66 69 6C 2E 00 04 00 00 00 E0 BB 41 00	exfil.....à»A	
009DB4A8	1E 00 00 00 1E 00 00 00 00 00 00 00 00 00 00 00	
009DB4B8	00 00 00 00 00 00 00 00 04 00 00 E0 BB 41 00à»A	

Address	Hex	ASCII	Comments
0066F8E0	00000020		return to r
0066F8F4	0066FC24		
0066F8F8	00000020		
0066F8F0	0040C4B4		
0066F900	00000000		
0066F904	0000000F		
0066F908	009DB480	"This is a	
0066F910	0000001E		
0066F914	00000000		
0066F918	0066F928		
0066F91C	00000000		

WannaHusky_RansomWare

May 2024

v1.0

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 encryption routine

XOR key used for the first 16 bytes of data

First byte of data is loaded into DL register which is the letter "T"

First byte of data is xor'd with byte value stored in esi+ebx+204 which is the 1st byte of xor key

and operation of ebx with F signifies that whenever ebx goes over a value of 16 its content are cleared to 0.

Then there is a jump to location 40D8FE, which is basically a loop for encryption. This continues until all the data of the cosmo.jpeg is encrypted.

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

Encrypted data is being stored here. 1st 16 bytes of data.

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.

The screenshot shows the assembly code and memory dump of the ransomware. The assembly code includes several calls to the `<ransomware.wannahusky - copy.encrypt_!1>` function, which is highlighted with a red box. The memory dump shows the first 16 bytes of encrypted data, also highlighted with a red box. A red arrow points from the assembly code to the memory dump.

```

0040D92F 89C2 mov edx, eax
0040D931 C70424 20000000 mov dword ptr ss:[esp],20
0040D938 E8 0DE3FFFF call <ransomware.wannahusky - copy.encrypt_!1>
0040D93D BA 20000000 mov edx,20
0040D942 8D8E E4010000 lea ecx,dword ptr ds:[esi+1E4]
0040D948 83EC 0C sub esp,C
0040D94B E8 71EBFFFF call ransomware.wannahusky - copy.40C4C1
0040D950 3B7D 0C cmp edi,dword ptr ss:[ebp+C]
0040D953 72 12 jb ransomware.wannahusky - copy.40D967
0040D955 8B45 0C mov eax,dword ptr ss:[ebp+C]
0040D958 893C24 mov dword ptr ss:[esp],edi
0040D95B 8D50 FF lea edx,dword ptr ds:[eax-1]
0040D95E 895424 04 mov dword ptr ss:[esp+4],edx
0040D962 E8 7686FFFF call ransomware.wannahusky - copy.405FDD
0040D967 3B7D E4 cmp edi,dword ptr ss:[ebp-1C]
0040D96A 72 12 jb ransomware.wannahusky - copy.40D97E
0040D96C 8B45 E4 mov eax,dword ptr ss:[ebp-1C]
0040D96F 893C24 mov dword ptr ss:[esp],edi
0040D972 8D50 FF lea edx,dword ptr ds:[eax-1]
0040D975 895424 04 mov dword ptr ss:[esp+4],edx
0040D979 E8 5F86FFFF call ransomware.wannahusky - copy.405FDD
0040D97E 83FB 1F cmp ebx,1F
0040D981 76 10 jbe ransomware.wannahusky - copy.40D993
0040D983 C74424 04 1F0000 mov dword ptr ss:[esp+4],1F
0040D988 891C24 mov dword ptr ss:[esp],ebx
0040D98E E8 4A86FFFF call ransomware.wannahusky - copy.405FDD
0040D993 8B45 E0 mov eax,dword ptr ss:[ebp-20]
0040D996 8A1438 mov dl,byte ptr ds:[eax+edi]
0040D999 8B45 08 mov eax,dword ptr ss:[ebp+8]
0040D99C 32941E 04020000 xor dl,byte ptr ds:[esi+ebx+204]
0040D9A3 881438 mov byte ptr ds:[eax+edi],dl
0040D9A6 8D47 01 lea eax,dword ptr ds:[edi+1]
0040D9A9 31C7 xor edi,eax
0040D9B2 74 05 je 0040D938

[ebp-20]: "This is a test file for exfil"

0090B480 54 68 69 73 20 69 73 20 61 20 74 65 73 74 20 66 This is a test f
0090B490 69 6C 65 20 66 6F 72 20 65 78 66 69 6C 2E 00 00 ile for exfil.
0090B4A0 04 00 00 00 E0 BB 41 00 LE 00 00 00 LE 00 00 00 ..A...
0090B4B0 88 FD A0 08 82 29 FA A9 C6 68 D6 71 E0 F0 00 E1 .g..JueInhqao.a
0090B4C0 00 00 00 00 E0 BB 41 00 LE 00 00 00 LE 00 00 00 ..A...
0090B4D0 00 00 00 00 00 00 00 00 00 00 00 00 00 00 00 00 ..A...
0090B4E0 00 00 00 00 00 00 00 00 00 00 00 00 00 00 00 00 ..A...
0090B4F0 00 00 00 00 00 00 00 00 00 00 00 00 00 00 00 00 ..A...
0090B500 00 00 00 00 00 00 00 00 00 00 00 00 00 00 00 00 ..A...
0090B510 00 00 00 00 00 00 00 00 00 00 00 00 00 00 00 00 ..A...

```

Encrypted data is stored in this location.
First 16 bytes of encrypted data.

Key Generation

First xor key generation :

The screenshot shows the assembly code for the first xor key generation. The assembly code is as follows:

```
EIP 0040D938
BA 20000000
8D8E E4010000
lea ecx,dword ptr ds:[esi+1E4]
sub esp,C
call ransomware.wannahusky - copy.encrypt_!1>
[ebp-20]:"This is a test file for exfi"
83EC 0C
8845 0C
893C24
8D50 FF
895424 04
E8 7686FFFF
3B7D 0C
72 12
jb ransomware.wannahusky - copy.40D967
mov eax,dword ptr ss:[ebp+C]
mov dword ptr ss:[esp],edi
lea edx,dword ptr ds:[eax-1]
mov dword ptr ss:[esp+4],edx
call ransomware.wannahusky - copy.405FDD
cmp edi,dword ptr ss:[ebp-1C]
jb ransomware.wannahusky - copy.40D97E
mov eax,dword ptr ss:[ebp-1C]
mov dword ptr ss:[esp],edi
lea edx,dword ptr ds:[eax-1]
mov dword ptr ss:[esp+4],edx
call ransomware.wannahusky - copy.405FDD
cmp ebx,1F
jbe ransomware.wannahusky - copy.40D993
c74424 04 1F00
mov dword ptr ss:[esp+4],1F
891C24
mov dword ptr ss:[esp],ebx
call ransomware.wannahusky - copy.405FDD
8845 E0
8A1438
8845 08
32941E 04020000
xor dl,byte ptr ds:[esi+ebx+204]
881438
8047 01
31C7
79 0F
jns ransomware.wannahusky - copy.40D9BC
test eax,eax
79 0B
jns ransomware.wannahusky - copy.40D9BC
```

The assembly code is annotated with several red boxes:

- A red box highlights the instruction `call <ransomware.wannahusky - copy.encrypt_!1>` with the label "Key generator function call".
- A red box highlights the string "[ebp-20]:"This is a test file for exfi" with the label "First xor key for the first 16 bytes".

The memory dump window below shows the first 16 bytes of the xor key:

Address	Hex	ASCII
0066FC04	00 CF FC 30 FC F3 FC 30 C0 3F 00 FC	Iu000A?.
0066FC14	00 00 00 00 00 00 00 00 00 00 00 00 00 00 00 01
0066FC24	DC 95 C0 78 A2 40 89 89 AD 48 A2 14 92 84 20 87	Ü.Ax... H...
0066FC34	00 00 00 00 00 00 00 00 00 00 00 00 00 00 00 00
0066FC44	00 00 00 00 00 00 00 00 00 00 00 00 00 00 00 00
0066FC54	00 00 00 00 00 00 00 00 00 00 00 00 00 00 00 00

Second xor key generation :

The screenshot shows the assembly code for the second xor key generation. The assembly code is identical to the first one, except for the offset in the `xor dl,byte ptr ds:[esi+ebx+204]` instruction.

```
EIP 0040D938
BA 20000000
8D8E E4010000
lea ecx,dword ptr ds:[esi+1E4]
sub esp,C
call ransomware.wannahusky - copy.40C4C1
[ebp-20]:"This is a test file for exfi"
83EC 0C
8845 0C
893C24
8D50 FF
895424 04
E8 7686FFFF
3B7D 0C
72 12
jb ransomware.wannahusky - copy.40D967
mov eax,dword ptr ss:[ebp+C]
mov dword ptr ss:[esp],edi
lea edx,dword ptr ds:[eax-1]
mov dword ptr ss:[esp+4],edx
call ransomware.wannahusky - copy.405FDD
cmp edi,dword ptr ss:[ebp-1C]
jb ransomware.wannahusky - copy.40D97E
mov eax,dword ptr ss:[ebp-1C]
mov dword ptr ss:[esp],edi
lea edx,dword ptr ds:[eax-1]
mov dword ptr ss:[esp+4],edx
call ransomware.wannahusky - copy.405FDD
cmp ebx,1F
jbe ransomware.wannahusky - copy.40D993
c74424 04 1F00
mov dword ptr ss:[esp+4],1F
891C24
mov dword ptr ss:[esp],ebx
call ransomware.wannahusky - copy.405FDD
8845 E0
8A1438
8845 08
32941E 04020000
xor dl,byte ptr ds:[esi+ebx+204]
881438
8047 01
31C7
79 0F
jns ransomware.wannahusky - copy.40D9BC
test eax,eax
79 0B
jns ransomware.wannahusky - copy.40D9BC
```

The assembly code is annotated with several red boxes:

- A red box highlights the instruction `call <ransomware.wannahusky - copy.encrypt_!1>` with the label "Key generator function".
- A red box highlights the string "[ebp-20]:"This is a test file for exfi" with the label "Second xor key for the next 16 bytes".

The memory dump window below shows the next 16 bytes of the xor key:

Address	Hex	ASCII
0066FBF4	00 CF FC 30 FC F3 FC 30 C0 3F 00 FC	Iu000A?.
0066FC04	00 00 00 00 00 00 00 00 00 00 00 00 00 00 01
0066FC14	53 0F 8A FB C7 45 36 B9 A9 63 B4 F1 C4 CB 73 B8	S..üCE6*8c RAES.
0066FC24	00 00 00 00 00 00 00 00 00 00 00 00 00 00 00 00
0066FC34	00 00 00 00 00 00 00 00 00 00 00 00 00 00 00 00
0066FC44	00 00 00 00 00 00 00 00 00 00 00 00 00 00 00 00
0066FC54	00 00 00 00 00 00 00 00 00 00 00 00 00 00 00 00

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.

The screenshot shows the OllyDbg debugger interface. The assembly pane displays the encryption routine, with the instruction at address 0040D910 highlighted in green. A tooltip for this instruction states: "Returns after completing the encryption routine." The memory dump pane below shows two blocks of data: plain text from cosmo.jpeg and encrypted data. The plain text is labeled "plain text data from cosmo.jpeg" and the encrypted data is labeled "Encrypted data".

Assembly code (partial):

```
0040D8FE 3B7D E4 cmp edi,dword ptr ss:[ebp-1C]
0040D901 7C 0E j1 ransomware.wannahusky - copy.40D911
0040D903 89E 24020000 mov dword ptr ds:[esi+224],ebx
0040D905 80D5 F4 lea esp,dword ptr ss:[ebp-C]
0040D907 5B pop ebx
0040D90D 5E pop esi
0040D90E 5F pop edi
0040D90F 5D pop ebp
0040D910 C3 ret Returns after completing the encryption routine.

0040D911 85DB test ebx,ebx
0040D913 75 3B jne ransomware.wannahusky - copy.40D950
0040D915 896E 04020000 lea edx,dword ptr ds:[esi+204]
0040D91B C74424 08 20000 mov dword ptr ss:[esp+8],20
0040D923 8086 E4010000 lea eax,dword ptr ds:[esi+1E4]
0040D929 89F1 mov ecx,esi
0040D92B 895424 04 mov dword ptr ss:[esp+4],edx
0040D92F 89C2 mov edx,eax
0040D931 C70424 20000000 mov dword ptr ss:[esp],20
0040D938 E8 0DE3FFFF call <ransomware.wannahusky - copy.encrypt__!1>
0040D93D BA 20000000 mov edx,20
0040D942 808E E4010000 lea ecx,dword ptr ds:[esi+1E4]
0040D948 83EC 0C sub esp,C
0040D94B E8 71EBFFFF call ransomware.wannahusky - copy.40C4C1
0040D950 3B7D 0C cmp edi,dword ptr ss:[ebp+C]
0040D953 72 12 jb ransomware.wannahusky - copy.40D967
0040D955 8B45 0C mov eax,dword ptr ss:[ebp+C]
0040D958 893C24 mov dword ptr ss:[esp],edi
0040D95B 8D50 FF lea edx,dword ptr ds:[eax-1]
0040D95E 895424 04 mov dword ptr ss:[esp+4],edx
0040D962 E8 7686FFFF call ransomware.wannahusky - copy.405FDD
0040D967 3B7D E4 cmp edi,dword ptr ss:[ebp-1C]
0040D96A 72 12 jb ransomware.wannahusky - copy.40D97E
0040D96C 8B45 E4 mov eax,dword ptr ss:[ebp-1C]
0040D96F 893C24 mov dword ptr ss:[esp],edi
```

Memory dump (partial):

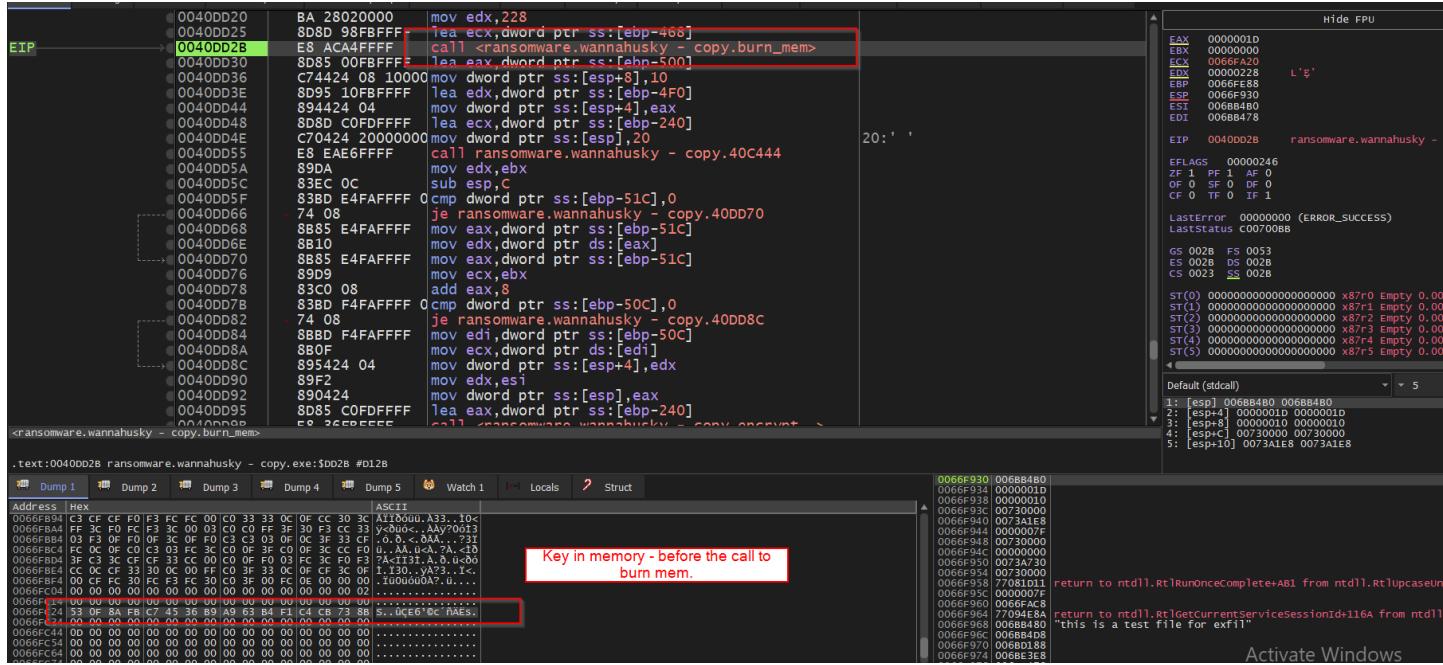
Address	Hex	ASCII
009DB460	69 6C 65 20 66 6F 72 20 65 78 66 69 6C 2E 00 00	file for exfil...
009DB470	04 00 00 00 E0 BB 41 00 1E 00 00 00 1E 00 00 00	This is a test f
009DB480	54 68 69 73 20 69 73 20 61 20 74 65 73 74 20 66	ile for exfil... This is a test f
009DB490	69 6C 65 20 66 6F 72 20 65 78 66 69 6C 2E 00 00	ile for exfil...
009DB4A0	04 00 00 00 E0 BB 41 00 1E 00 00 00 1E 00 00 00	This is a test f
009DB4B0	88 FD A9 08 82 29 FA A9 CC 68 D6 71 E1 F0 00 E1	ile for exfil... .yE.)üEihöqad.ä
009DB4C0	3A 63 EF DB A1 2A 44 99 CC 1B D2 98 A8 E5 00 00	ic10;^D.i.o. ä..
009DB4D0	04 00 00 00 E0 BB 41 00 1E 00 00 00 1E 00 00 00ä.A.....
009DB4E0	00 00 00 00 00 00 00 00 00 00 00 00 00 00 00 00
009DB4F0	00 00 00 00 00 00 00 00 00 00 00 00 00 00 00 00
009DB500	00 00 00 00 00 00 00 00 00 00 00 00 00 00 00 00
009DB510	00 00 00 00 00 00 00 00 00 00 00 00 00 00 00 00

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.

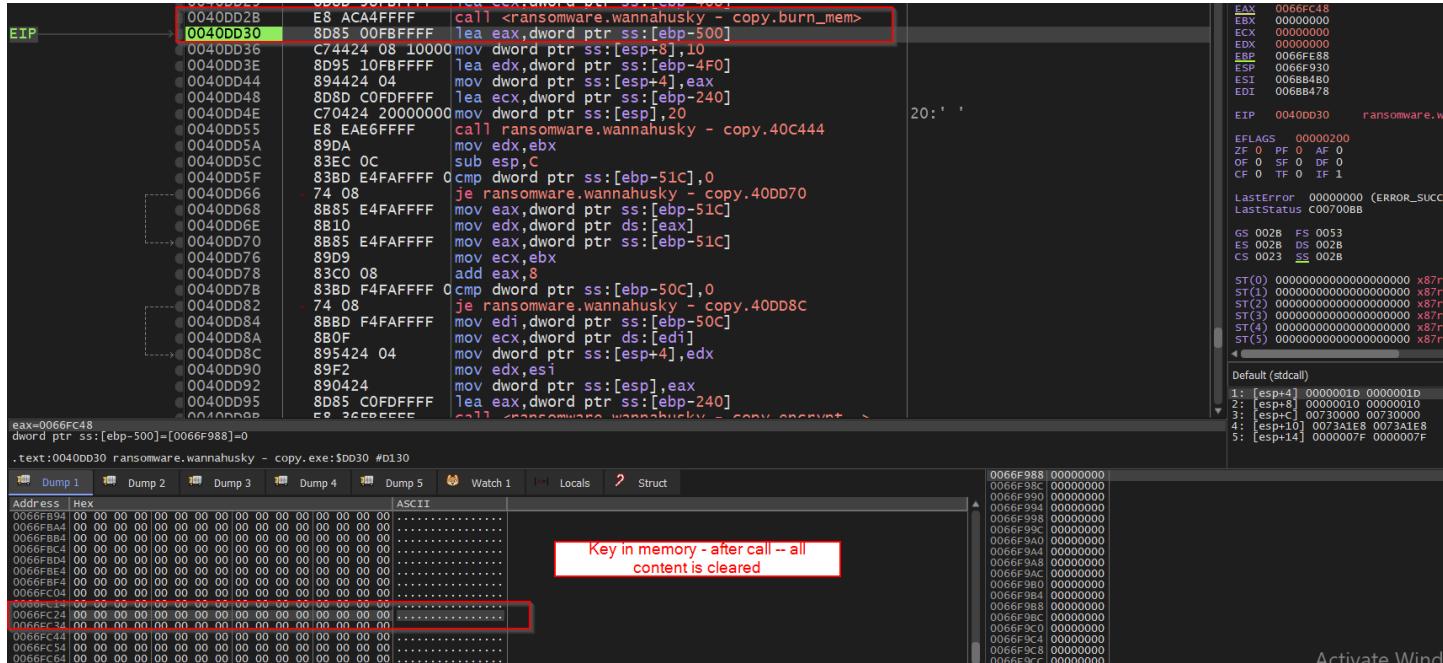
So I saved both the keys from the memory in a file on my desktop so as to decrypt the encrypted data found in the wannahusky file.

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.

The screenshot shows two windows from a debugger. The top window displays assembly code for the `ransomware.wannahusky - copy.405FDD` routine. The bottom window shows a memory dump of the file being processed.

Assembly Code (Top Window):

```

0040D993 891C24 mov dword ptr ss:[esp],ebx
0040D998 E8 4A86FFFF call ransomware.wannahusky - copy.405FDD
0040D999 8845 E0 mov eax,dword ptr ss:[ebp-20]
0040D99A 8A1438 mov dl,byte ptr ds:[eax+edi]
0040D99B 8845 08 mov eax,dword ptr ss:[ebp+8]
0040D99C 32941E 04020000 xor dl,byte ptr ds:[esi+eax+204]
0040D99D 881438 mov byte ptr ds:[eax+edi+1],dl
0040D99E 8047 01 lea eax,dword ptr ds:[ev+1]
0040D99F 31C7 xor edi,eax
0040D9A0 79 0F jns ransomware.wannahusky - copy.40D9BC
0040D9A1 85C0 test eax,eax
0040D9A2 79 0B jns ransomware.wannahusky - copy.40D9BC
0040D9A3 8945 DC mov dword ptr ss:[ebp-24],eax
0040D9A4 04020000 call ransomware.wannahusky - copy.4046B1
0040D9A5 8845 DC mov eax,dword ptr ss:[ebp-24]
0040D9A6 43 inc ebx
0040D9A7 89C7 mov edi,eax
0040D9A8 83E3 OF and ebx,F
0040D9A9 E9 37FFFFFF jmp ransomware.wannahusky - copy.40D8FE
0040D9C7 55 push ebp
0040D9C8 89E5 mov ebp,esp
0040D9C9 57 push edi
0040D9CB 56 push esi
0040D9CC 53 push ebx
0040D9CD 81EC 10050000 sub esp,1C

```

Memory Dump (Bottom Window):

The dump shows memory addresses from `0040D993` to `0040D99D`. A red box highlights the encrypted data at `0040D994` with the label "Encrypted...". Another red box highlights the decrypted plain text at `0040D994` with the label "[ebp+8]:'this is a test file'".

Annotations:

- A red box highlights the assembly instruction `call ransomware.wannahusky - copy.405FDD` with the text "Same encryption routine, but this time the data passed to eax/edx contains the encrypted text from previous call."
- A red box highlights the memory dump entry at `0040D994` with the text "Encrypted...".
- A red box highlights the memory dump entry at `0040D994` with the text "[ebp+8]:'this is a test file'".
- A red box highlights the memory dump entry at `0040D994` with the text "We can see plain text msg being generated by performing xor operation on encrypted data with same key".

Encode function

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

The screenshot shows the debugger's assembly and dump panes across three windows, illustrating the flow of data from the initial encrypted state to its final base64-encoded form.

Window 1 (Top Left): Shows assembly code for the "Encryption call". The instruction at address `00400D8` is `call <ransomware.wannahusky - copy.encrypt_>`. The assembly continues with various memory operations involving `eax`, `ecx`, `edx`, and `esi`. Red boxes highlight the `call` instruction and the parameters `ss:[esp+4]` and `ss:[esp]`.

Window 2 (Top Right): Shows the register state. `ECX` is labeled as the address of the encrypted data, and `EDX` is labeled as the length of the data (30 in decimal).

Window 3 (Bottom Left): Shows the memory dump of the encrypted data. It is labeled "Encrypted data". Red boxes highlight the `call` instruction and the parameters `ss:[esp+4]` and `ss:[esp]`.

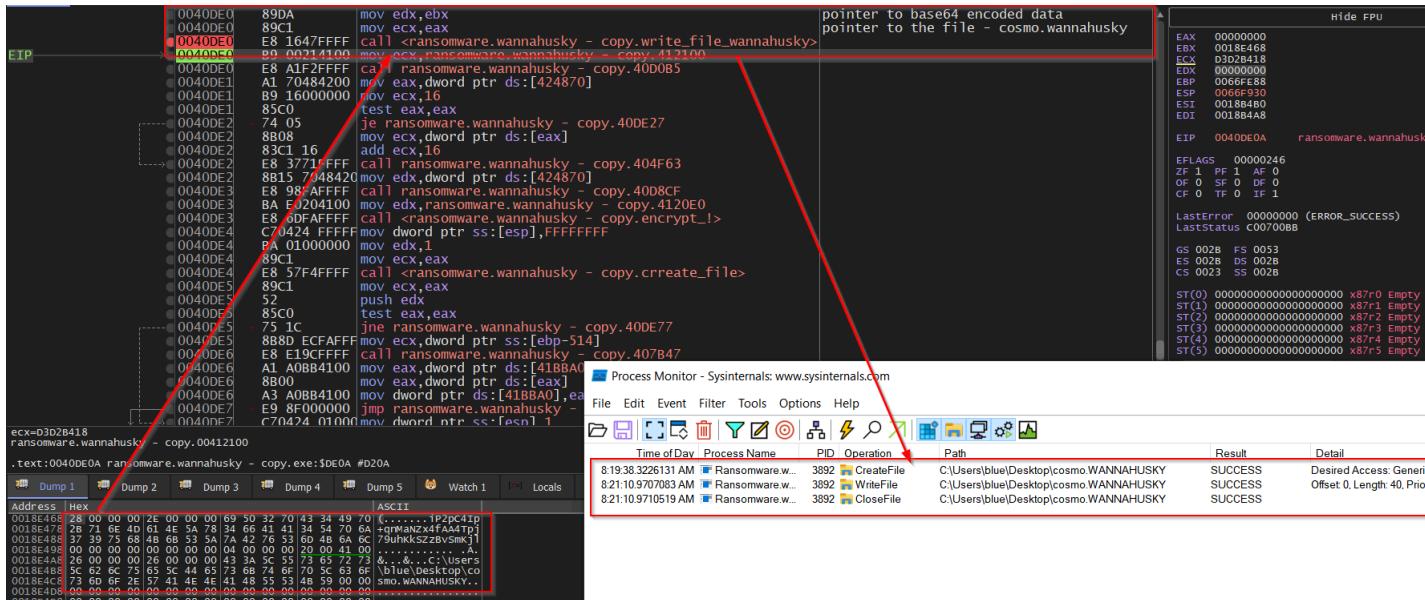
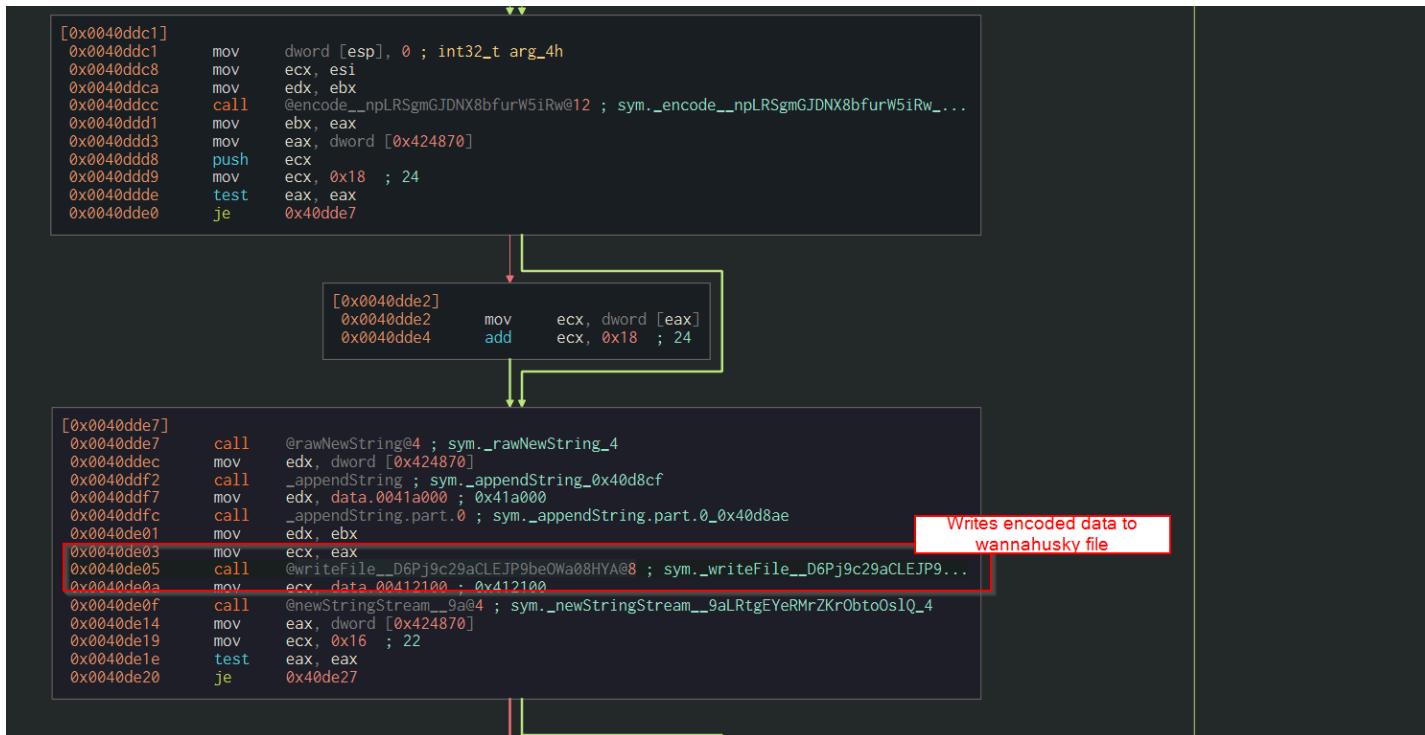
Window 4 (Bottom Right): Shows the register state. `ECX` is labeled as the address of the encrypted data, and `EDX` is labeled as the length of the data (30 in decimal). The assembly code continues with various memory operations.

Window 5 (Bottom Left): Shows the memory dump of the base64-encoded data. It is labeled "Encoded data after the call". Red boxes highlight the `call` instruction and the parameters `ss:[esp+4]` and `ss:[esp]`.

Assembly Labels:

- Encryption call
- Base64 encoding function
- Encrypted data
- Encoded data after the call

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.

The screenshot shows a debugger interface with assembly code and memory dump sections. Red annotations highlight specific assembly instructions and memory dump sections, which are then linked to entries in a Process Monitor log.

Assembly Code Annotations:

- A red box highlights the instruction `E8 BD45FFFF call <ransomware.wannahusky - copy.write_file>`. A callout points to it from the text "Write file function that writes Wannahusky.png and ps1.ps1".
- A red box highlights the instruction `E8 E86FFFFF call <ransomware.wannahusky - copy.execute_shell_cmd>`. A callout points to it from the text "ECX value here points to the powershell command to be executed".
- A red box highlights the instruction `E8 A06FFFFF call <ransomware.wannahusky - copy.append_str>`. A callout points to it from the text "Executes the powershell script which tries to change the background to wannahusky.png".

Memory Dump Annotations:

- A red box highlights a memory dump section containing the string "...powershell <...>". A callout points to it from the text "Executes the powershell script which tries to change the background to wannahusky.png".

Process Monitor Log:

Time of Day	Process Name	PID	Operation	Path	Result	Detail
8:19:46 3226131 AM	Ransomware.w...	3892	CreateFile	C:\Users\blue\Desktop\cosmo WANNAHUSKY	SUCCESS	Des...
8:21:03 9/7/083 AM	Ransomware.w...	3892	WriteFile	C:\Users\blue\Desktop\WANNAHUSKY.png	SUCCESS	Des...
8:25:38 7176614 AM	Ransomware.w...	3892	WriteFile	C:\Users\blue\Desktop\WANNAHUSKY.png	SUCCESS	Des...
8:26:15 8120580 AM	Ransomware.w...	3892	CloseFile	C:\Users\blue\Desktop\WANNAHUSKY.png	SUCCESS	Des...
8:26:21 8121941 AM	Ransomware.w...	3892	CreateFile	C:\Users\blue\Desktop\cosmo.jpeg	SUCCESS	Des...
8:26:33 68337381 AM	Ransomware.w...	3892	QueryAttribute...	C:\Users\blue\Desktop\cosmo.jpeg	SUCCESS	Des...
8:26:33 6838592 AM	Ransomware.w...	3892	SetDisposition...	C:\Users\blue\Desktop\cosmo.jpeg	SUCCESS	Des...
8:26:33 68338962 AM	Ransomware.w...	3892	CloseFile	C:\Users\blue\Desktop\cosmo.jpeg	SUCCESS	Des...
8:27:28 0984758 AM	Ransomware.w...	3892	CreateFile	C:\Users\blue\Desktop\ps1.ps1	SUCCESS	Des...
8:27:58 0100141 AM	Ransomware.w...	3892	WriteFile	C:\Users\blue\Desktop\ps1.ps1	SUCCESS	Des...
8:27:58 0114:391 AM	Ransomware.w...	3892	CloseFile	C:\Users\blue\Desktop\ps1.ps1	SUCCESS	Des...

Second Debugger Session:

The second debugger session shows similar assembly code and memory dump sections. Red annotations highlight the same assembly instructions and memory dump sections as the first session, linking them to the Process Monitor log.

Dropped files

Powershell script : Ps1.ps1

```
1 $code = @'
2     using System.Runtime.InteropServices;
3
4     namespace Win32{
5
6         public class Wallpaper{
7
8             [DllImport("user32.dll", CharSet=CharSet.Auto)]
9             static extern int SystemParametersInfo (int uAction , int uParam , string lpvParam , int fuWinIni) ;
10
11             public static void SetWallpaper(string thePath){
12                 | SystemParametersInfo(20,0,thePath,3);
13             }
14         }
15     }
16     '@
17 add-type $code
18
19 $currDir = Get-Location
20 $wallpaper = ".\WANNAHUSKY.PNG"
21 $fullpath = Join-Path -path $currDir -ChildPath $wallpaper
22
23 [Win32.Wallpaper]::SetWallpaper($fullpath)
24
```

WannaHusky.PNG :



that picture of cosmo on your
desktop is now encrypted!

to save him, you must send 100 Huskycoin to <https://huskyhacks.dev>

hurry! you have 24 hours before we delete cosmo

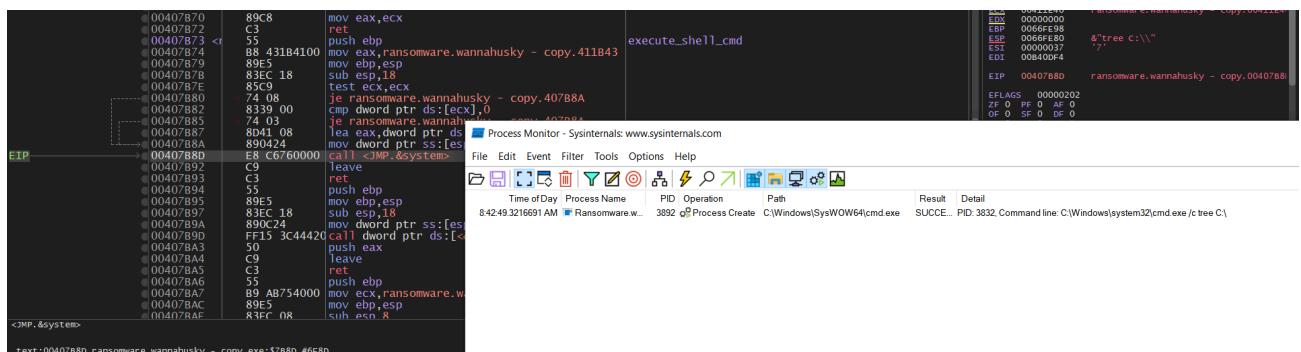
WannaHusky_RansomWare

May 2024

v1.0

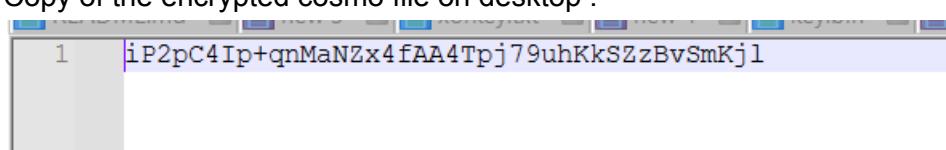
Command Execution

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.



```
00407B70 89C8 mov eax,ecx
00407B72 C3 ret
00407B73 <f>
00407B74 55 push ebp
00407B75 B8 43184100 mov esp, rransomware.wannahusky - copy.411843
00407B79 89E5 sub esp,ss
00407B7B 83EC 18 sub esp,18
00407B7D 74 08 test ecx,ecx
00407B80 89E5 00 cmp dword ptr ds:[eax],0
00407B83 74 03 je rransomware.wannahu...-copy.407B8A
00407B87 8D41 08 lea eax,dword PTR ss:[es...
00407B8A 890424 mov byte PTR ss:[es...
00407B8E E8 CG760000 call <MP.&System>
00407B90 E8 00000000 lea eax,[CG760000]
00407B93 C9 leave
00407B94 55 push ebp
00407B95 89E5 mov esp,esp
00407B97 83EC 18 sub esp,18
00407B99 89C424 mov dword PTR ss:[es...
00407B9B FF15 3C4442 mov dword PTR ds:[es...
00407B9D 83E3 00 push eax
00407B9F C9 leave
00407B A3 C3 ret
00407B9B 89E5 push esp
00407B9D 83EC 00 sub esp,8
00407B9E 83C8 OR sub esp,R
text:00407B8D rransomware.wannahusky - copy.exe+1780 #680
<MP.&System>
Administrator: C:\Users\blue\Desktop\Ransomware.wannahusky - Copy.exe
;x64
;system_enterservices.dll!03f5f7f11d50a3a_4_0_15805_0_none_6b8061d4f8a842e1
;x64 netfx4-enterprise.dll!03f5f7f11d50a3a_4_0_15805_0_none_32c05b2d11fsfa4
;x64 netfx4-enterprise.dll!03f5f7f11d50a3a_4_0_15805_0_none_d768e524291719da
;x64 netfx4-system.web.dll!03f5f7f11d50a3a_4_0_15805_0_none_d768e524291719da
;x64 netfx4-windows.forms.dll!03f5f7f11d50a3a_4_0_15805_0_none_a1559af21b95819
;x64 netfx4-thinclient.dll!03f5f7f11d50a3a_4_0_15805_0_none_0f4664a855d446c9
;x64 netfx4-uninstallperlistssqlstate.sq1!03f5f7f11d50a3a_4_0_15805_0_none_291a4002be92215b
;x64 netfx4-uninstallsslstatetemplate.sql!03f5f7f11d50a3a_4_0_15805_0_none_231ddc33015c6bd
;x64 netfx4-uninstallsslsstate.sq1!03f5f7f11d50a3a_4_0_15805_0_none_chf771a6dad84bf
;x64 netfx4-xmlhttp.dll!03f5f7f11d50a3a_4_0_15805_0_none_e8c80247227c74317fce
;x64 netfx4-vbcr.exe!03f5f7f11d50a3a_4_0_15805_0_none_d946e519e580fe
;x64 netfx4-vcruntime140_c1r.dll!31bf3856ad364e35 4_0_15805_0_none_2efdfd3759de32d3
;x64 netfx4-webengine.dll!03f5f7f11d50a3a_4_0_15805_0_none_21a667a70f19fe6e
;x64 netfx4-webengine.dll!03f5f7f11d50a3a_4_0_15805_0_none_6343bf1ec49231a
;x64 netfx4-webengine.dll!03f5f7f11d50a3a_4_0_15805_445_none_5e05831a74b7hd9
;x64 netfx4-web_config.b603f5f7f11d50a3a_4_0_15805_0_nones_4397eb093850b0d0
;x64 netfx4-web_lowtrust.config.default.b603f5f7f11d50a3a_4_0_15805_0_nones_52232eca5
;x64 netfx4-web_lowtrust.config.default.b603f5f7f11d50a3a_4_0_15805_0_nones_95be3feaa25a13
;x64 netfx4-web_lowtrust.config.default.b603f5f7f11d50a3a_4_0_15805_0_nones_d545e5e52e62a30f
;x64 netfx4-web_lowtrust.config.default.b603f5f7f11d50a3a_4_0_15805_0_nones_642c2d3120c2fb1b3
;x64 netfx4-web_lowtrust.config.default.b603f5f7f11d50a3a_4_0_15805_0_nones_d545e5e52e62a30f
;x64 netfx4-web_minimaltrust.config.default.b603f5f7f11d50a3a_4_0_15805_0_nones_90c8556a1960374
;x64 netfx4-web_minimaltrust.config.b603f5f7f11d50a3a_4_0_15805_0_nones_2a06fa1b329hd70
;x64 netfx4-xweb_minimaltrust.config.default.b603f5f7f11d50a3a_4_0_15805_0_nones_f228cb15a3d9a2
;x64 netfx4-xweb_minimaltrust.config.default.b603f5f7f11d50a3a_4_0_15805_0_nones_63098863d7a63cb0d8
;x64 netfx4-worklowserv..ornamecounters.dll!03f5f7f11d50a3a_4_0_15805_0_nones_ff994c85892f2df
;x64 netfx4-worklowserv..ornamecounters.dll!03f5f7f11d50a3a_4_0_15805_0_nones_f765d5a01a30bda7
;x64 netfx4-worklowserv..ornamecounters.man!03f5f7f11d50a3a_4_0_15805_0_nones_f82830658ac60db
;x64 netfx4-worklowserv..targets.dll!03f5f7f11d50a3a_4_0_15805_0_nones_f498273f6a759d065
;x64 netfx4-xptgx..b603f5f7f11d50a3a_4_0_15805_0_nones_3680a22d62a2a2
;x64 netfx4-xptgx..b603f5f7f11d50a3a_4_0_15805_0_nones_2d12855136597213
;x64 netfx4-xptgx..b603f5f7f11d50a3a_4_0_15805_48nones_24de557af156ecc
;x64 netfx4-xptmem_manifest!03f5f7f11d50a3a_4_0_15805_0_nones_49580d93631cc015
;x64 netfx4-xptmem_manifest!03f5f7f11d50a3a_4_0_15805_0_nones_d5a05e009d980893
;x64 policy.1.0.microcosm..op.security.azureles..31bf3856ad364e35_10_0_19041_1_none_f7572160dc4eabe
;x64 policy.1.2.microcosm..op.security.azureles..31bf3856ad364e35_10_0_19041_1_none_468c57481260dea0
;x64 presentationcore..31bf3856ad364e35_10_0_19041_1_none_7b936f09f28abc
;x64 regsvr..b603f5f7f11d50a3a_4_0_15805_0_nones_498273f6a759d065
;x64 regsvr..b603f5f7f11d50a3a_4_0_15805_0_nones_5012855136597213
;x64 regsvr..b603f5f7f11d50a3a_4_0_15805_0_nones_8cf1f3b4629d3a7
;x64 smlib..31bf3856ad364e35_10_0_19041_1_none_5hbdhee4d17ae1
;x64 system..data..b7a5c561934e89f..10_0_19200_236_nones_h2deae32434c099
;x64 system..net..b603f5f7f11d50a3a_4_0_15805_0_nones_a01546f51b0f0c
;x64 system..net..b603f5f7f11d50a3a_4_0_15805_0_nones_e8c80247227c74317fce
;x64 wcf..msnssvchost..exe!31bf3856ad364e35_10_0_19041_1_none_4866ff4000fd40
;x64 wcf..msnssvchost..perfc..reg!31bf3856ad364e35_10_0_19041_1_none_a8201546f51b0f0c
;x64 wcf..msnssvchost..perfc..reg!31bf3856ad364e35_10_0_19041_1_none_d668098a5c3f32
;x64 wcf..msn..sm..ins..r.dll!31bf3856ad364e35_10_0_19041_1_none_d23ca2c47f61b5d8
;x64 wcf..msvc..mod..end..perf..reg!31bf3856ad364e35_10_0_19041_1_none_b2ce75736c6b6b
;x64 wcf..msvc..mod..end..perf..reg!31bf3856ad364e35_10_0_19041_1_none_20a3e5756df6e5
;x64 wcf..msvc..mod..end..perf..reg!31bf3856ad364e35_10_0_19041_1_none_344304695208844
;x64 wcf..msvc..mod..op..perf..reg!31bf3856ad364e35_10_0_19041_1_none_2191fb2138118f3
;x64 wcf..msvc..mod..op..perf..reg!31bf3856ad364e35_10_0_19041_1_none_22cb892f126c694c
;x64 wcf..msvc..mod..svc..perfc..reg!31bf3856ad364e35_10_0_19041_1_none_d08d8427635608
;x64 wcf..msvc..mod..svc..perfc..reg!31bf3856ad364e35_10_0_19041_1_none_d668098a5c3f32
;x64 wcf..msvc..mod..svc..perfc..reg!31bf3856ad364e35_10_0_19041_1_none_d756ba685b5c3f32
;x64 wcf..msvc..mod..svc..perfc..reg!31bf3856ad364e35_10_0_19041_1_none_f668c1fb7c5bd4
;x64 wcf..msvc..mod..svc..perfc..reg!31bf3856ad364e35_10_0_19041_1_none_f668c1fb7c5bd4
;x64 wcf..system..identitymodel..b603f5f7f11d50a3a_4_0_15805_0_nones_4949d4a2211ecde
;x64 wcf..system..identitymodel..b603f5f7f11d50a3a_4_0_15805_0_nones_e6980dc7d17e3f2b
;x64 wcf..system..identitymodel..b603f5f7f11d50a3a_4_0_15805_0_nones_43f364d80834e0
;x64 wcf..system..identitymodel..b603f5f7f11d50a3a_4_0_15805_0_nones_7894a1c41e0985
;x64 wcf..system..identitymodel..b603f5f7f11d50a3a_4_0_15805_0_nones_6782945101b14d5d7
;x64 wcf..system..runtime..serialization..b603f5f7f11d50a3a_10_0_19041_1_none_d5e154fc7c748389
;x64 wcf..system..servicemode..b603f5f7f11d50a3a_10_0_19200_118_nones_33f18e125b7a18f9
;x64 wcf..system..servicemode..b603f5f7f11d50a3a_10_0_19200_118_nones_d395cd0a88523a81
;x64 wcf..pening..31bf3856ad364e35_10_0_19200_250_nones_7b137039149c1529
;x64 wcf..presentationframework..31bf3856ad364e35_10_0_19200_250_nones_3f66ca8a2fed2d02
;x64 wnf..presentatiionhost..11_31bf3856ad364e35_10_0_19200_250_nones_3f66ca8a2fed2d02
```

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 :

The image shows a terminal window titled 'Cmder' with a black background and white text. The command 'cd Desktop\' is entered, followed by 'python decrypt.py'. The output shows the decrypted data: 'Decrypted Data: b'This is a test file for exfil.'

```
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+qnMaNZx4fAA4Tpj79uhKkSzzBvSmKj1'

    # 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

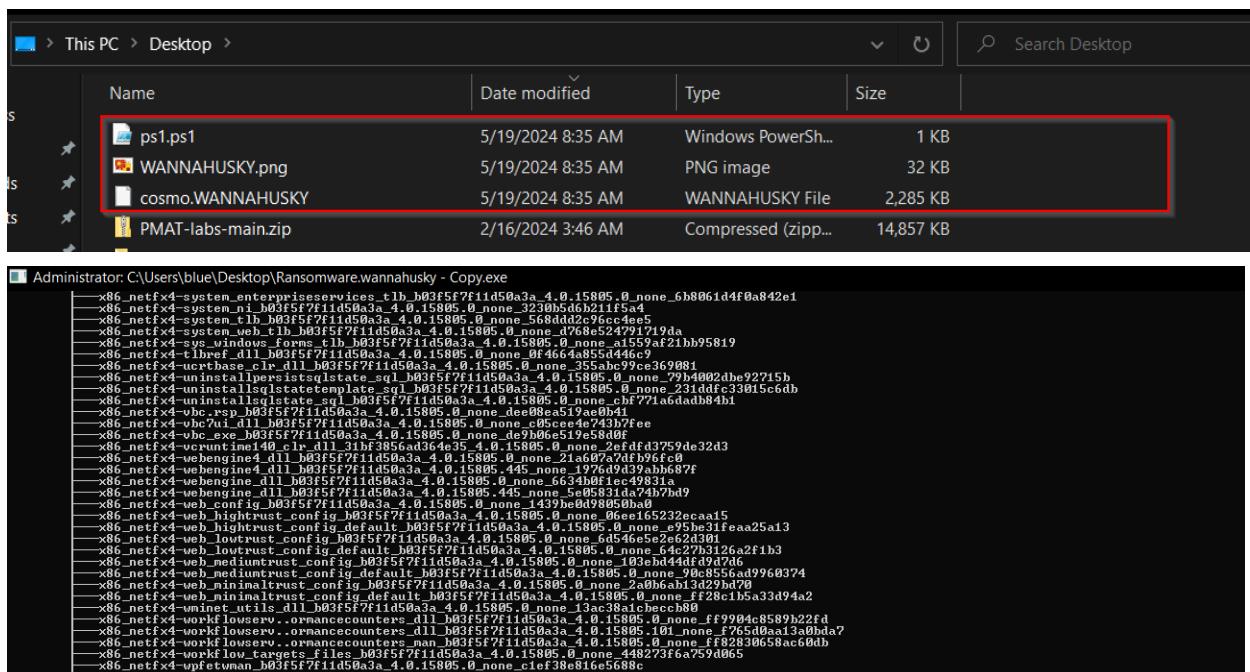
{Description of host-based indicators}

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



WannaHusky_RansomWare

May 2024

v1.0

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-_ABCDEFH
IJKLMNOPQRSTUVWXYZabcdefghijklmnopqrstuvwxyz0123456789+/" 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)

}
```