
Malware Analysis Report

Sample Information

Sample Name:	Ransomware.wannahusky.exe
MD5 Hash:	0287b38f8240a025b30c0a231ea403fc
SHA1 Hash:	691ac1b4b7b494f7b56eff0b48ba3e31a14e0d7d
Fuzzy Hash:	12288:dusXq0SdXCgSIAIZm/OSJsqrHrLOBHzI5WTvz:Qiq0SdXCgSIAIZm/tJsqrLCBHzI5WTb
Imphash:	a97ffe6ec502dacc4c154f9dc2b58725

Executive Summary

This Nim compiled binary performs a very small set of actions. The sample is ransomware, however, it only targets one file for encryption. The file cosmo.jpeg must be located on the Desktop for all actions to occur. If this file is present, the ransomware will execute properly. The Desktop wallpaper will be replaced with the ransom message, the file cosmo.jpeg will be encrypted, and and the *tree* command will be executed. If the target file is not found, the only action taken is the *tree* command. No network activity was found.

Technical Analysis

This sample appears to be compiled using Nim. This can be confirmed by looking at the symbols within the binary (Figure 1).

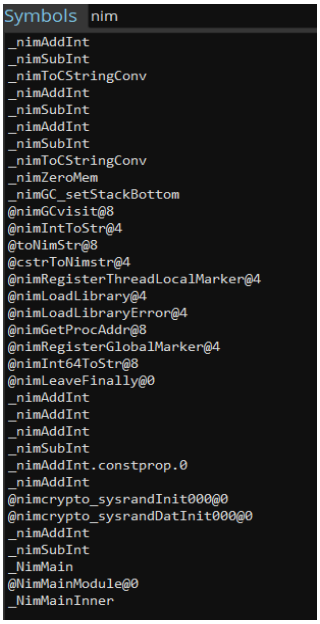

Figure 1 - Symbol names indicating binary was compiled in Nim. The most telling is the presence of NimMain.

On first detonation, a command prompt window was seen running a command (Figure 2) and the Desktop wallpaper was changed to a ransom message (Figure 3). The ransom message appears to specifically mention a picture of cosmo being targeted for ransom. This will be confirmed through additional analysis later.

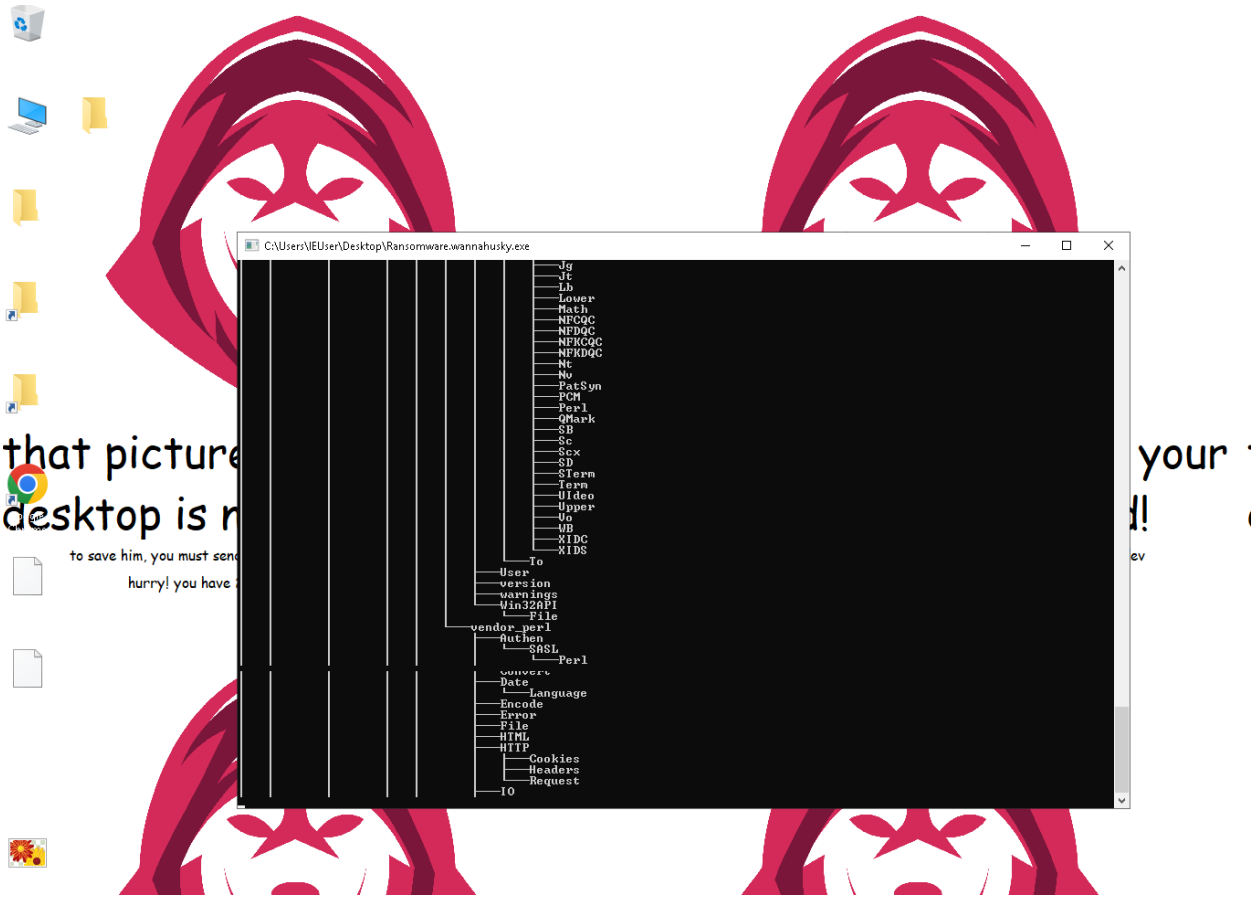

Figure 2 – Command Prompt and modified Desktop wallpaper.

Figure 3 – Ransom message set as the wallpaper to inform the user of their infection.

Utilizing procmon, a historical process tree is generated to understand the number of additional processes spawned by the sample (Figure 4).

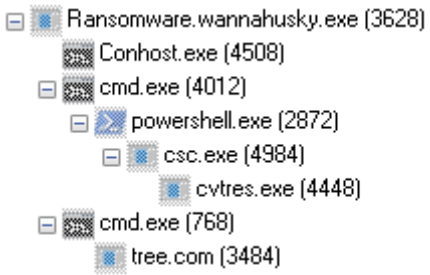

Figure 4 - Historical process tree generated from Procmon logs.

Through sandbox analysis, the basic steps of the ransomware sample were identified. Advanced analysis techniques were utilized to see the advanced details of the operations. First, the sample saves a PNG file to the Desktop called WANNAHUSKY.png (Figure 5). This PNG file is located within the sample's binary at offset 0x412100 (Figure 6). The sample will then write a Powershell script named ps1.ps1 to the Desktop as well (Figure 7). The Powershell script is located at offset 0x411ea0 within the sample's binary (Figure 8). The executable will then proceed to perform the encryption step. The only file it targets is "cosmo.jpeg" which was saved by the analyst to the Desktop before detonation (Figure 9).

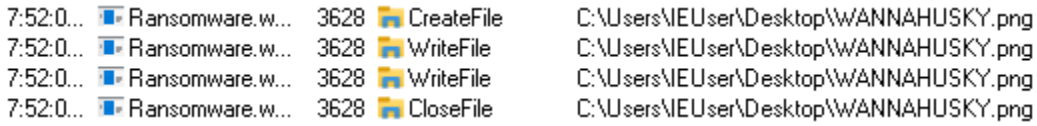

Figure 5 - Sample saving WANNAHUSKY.png to the Desktop.

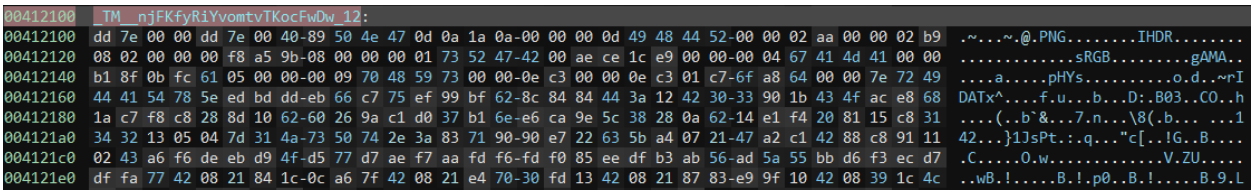

Figure 6 - PNG file header found within sample binary at offset 0x412100.

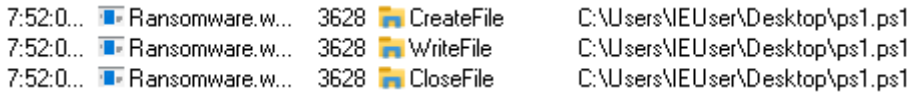

Figure 7 - Sample saving ps1.ps1 to the Desktop.

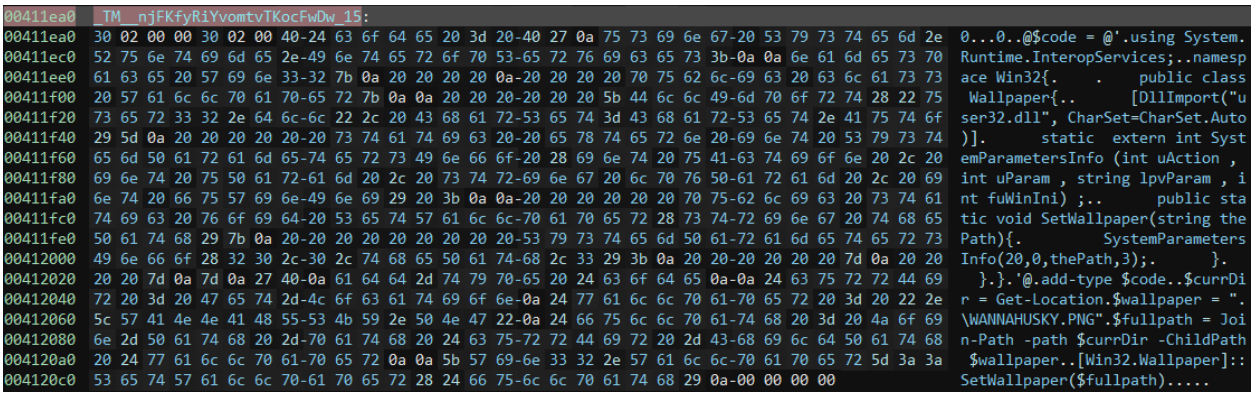

Figure 8 - Powershell script stored within sample binary at offset 0x411ea0.

Figure 9 - Sample reading cosmo.jpeg test file and writing the encrypted bytes to cosmo.WANNAHUSKY on the Desktop.

It can be confirmed that cosmo.jpeg is the only file targeted by this binary by performing static code analysis. Within the strings, the only filename mentioned is cosmo.jpeg and the encrypted version name of cosmo.WANNAHUSKY (Figure 10). The cross references of these strings show the use of them within a function where the encryption function is called. When looking at the function, the variable containing these strings are appended to a variable containing the path of the user's home directory (Figure 11). The appended string is then utilized for read/write operations using CreateFileA, ReadFile, and WriteFile API calls in which the memory buffer of ReadFile will be passed into the encryption function.

```
0041a0a0  1f 00 00 00 1f 00 00 40-44 65 73 6b 74 6f 70 5c-74 61 72 67 65 74 5c 63-6f 73 6d 6f 2e 57 41 4e  .......@Desktop\target\cosmo.WAN
0041a0c0  4e 41 48 55 53 4b 59 00                                                    NAHUSKY.
0041a0c8  _TM__njFKfyRiYvomtvTKocFwDw_5:
0041a0c8                          12 00 00 00 12 00 00 40-44 65 73 6b 74 6f 70 5c-63 6f 73 6d 6f 2e 6a 70        .......@Desktop\cosmo.jp
0041a0e0  65 67 00 00 00 00 00 00-00 00 00 00 00 00 00 00-00 00 00 00 00 00 00 00-00 00 00 00 00 00 00 00  eg..............................
```

Figure 10 - Strings in sample specifically targeting cosmo.jpeg.

```
uint32_t ebx_1 = eax_1
int32_t eax_2 = homeDirectoryPath
int32_t ecx = 0x12
if (eax_2 != 0)
    ecx = *eax_2 + 0x12
void* eax_3
eax_3, ecx = @rawNewString@4(ecx)
_appendString(ecx, homeDirectoryPath)
_appendString.part.0(eax_3, &cosmo.jpeg)
int32_t eax_5 = homeDirectoryPath
int32_t ecx_2 = 0x1f
if (eax_5 != 0)
    ecx_2 = *eax_5 + 0x1f
void* eax_6
eax_6, ecx_2 = @rawNewString@4(ecx_2)
_appendString(ecx_2, homeDirectoryPath)
_appendString.part.0(eax_6, &cosmo.WANNAHUSKY)
int32_t* eax_7 = @readFile__4PGnM9bWmsH0Nu7dnr3XzgA@4(eax_3)
```

Figure 11 - Decompiled function where the sample is appending the user's HOME directory with the /Desktop/cosmo.jpeg string to specifically read the file in preparation for encryption.

An additional confirmation of the cosmo.jpeg file being the only target is that if this file is not present, the sample outputs in error in a console window (Figure 12). The only step that is then taken by the sample is the tree command. No other actions are taken by the sample and the user's cosmo picture remains safe.

Figure 12 - Sample will output an error into console if no cosmo.jpeg exists on the Desktop.

There is also the issue of the encryption process itself and the way it encrypts files. The bytes of cosmo.jpeg is stored in a memory buffer using the ReadFile API call. This buffer is then put through the encryption process. However, the original cosmo.jpeg file is deleted using DeleteFileW instead of being

overwritten (Figure 13). The encrypted bytes are stored in a new file called cosmo.WANNAHUSKY. This allows the original file to be recovered from the MFT via traditional data recovery techniques.

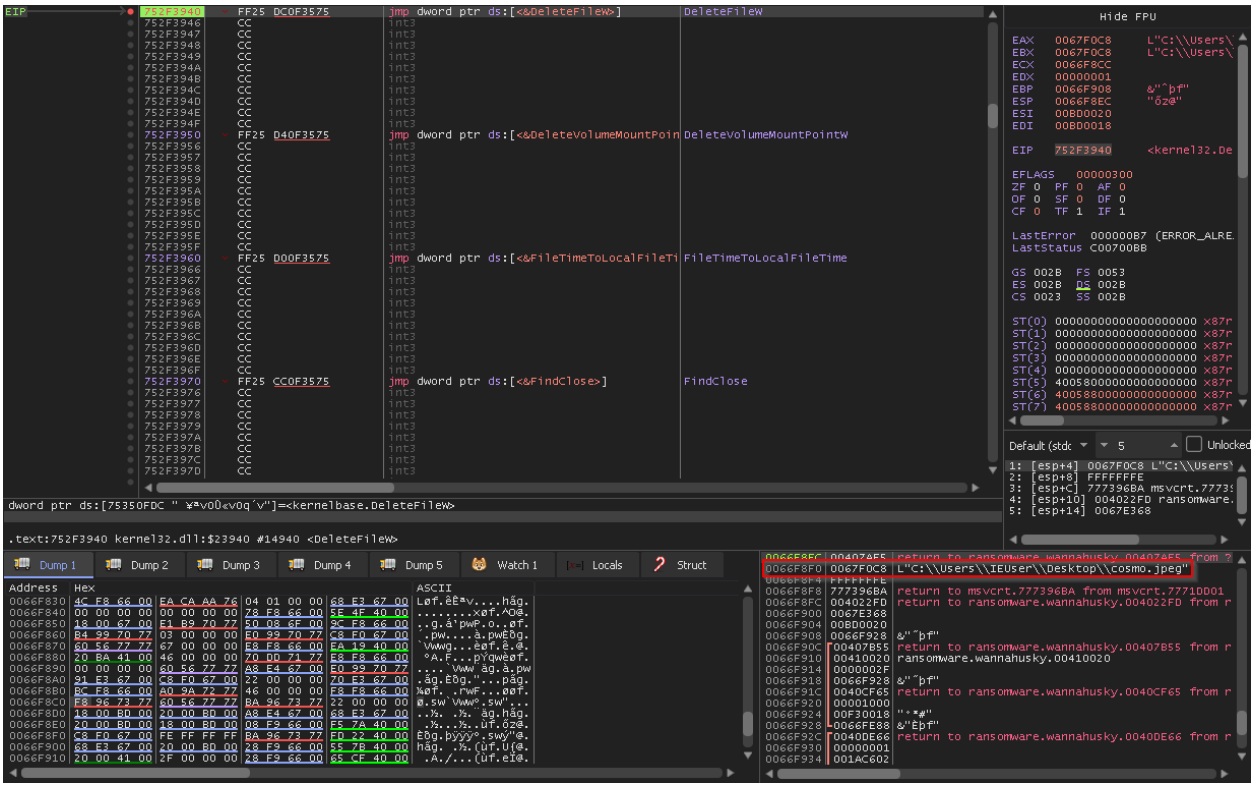

Figure 13 - The original cosmo.jpeg being deleted via DeleteFileW instead of being overwritten.

After the encryption routine is finished, the sample spawns a new cmd.exe process which executes the dropped Powershell script (Figure 14). After execution, the script is deleted from disk using DeleteFileW (Figures 15 and 16).

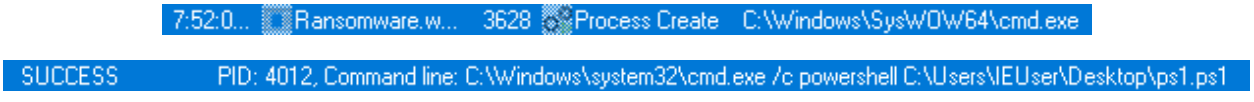

Figure 14 - Sample spawning a new cmd.exe process that calls powershell in order to execute the dropped ps1.ps1 script.

Figure 15 - DeleteFileW API call used to remove Powershell script from disk.

```
0066FE4C  00407AF5  return to ransomware.wannahusky.00407AF5 from ???
0066FE50  0067F2F8  L"C:\\Users\\IEUser\\Desktop\\ps1.ps1"
0066FE54  00000000
0066FE58  0066FE78  &"˜þf"
0066FE5C  00404F88  return to ransomware.wannahusky.00404F88 from ransomware.wannahusky.00404F3A
0066FE60  0067B4A8
0066FE64  00000032
0066FE68  0066FE88  &"Èþf"
```

Figure 16 - Argument on the stack of the file to be deleted. Shows the ps1.ps1 file being targeted for deletion.

By setting a breakpoint on DeleteFileW, the script was able to be viewed by the analyst before being removed by the sample (Figure 17). This script is responsible for setting the ransom message PNG file as the Desktop wallpaper.

```
$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)

```

Figure 17 - The contents of the Powershell script written to the Desktop.

The final step of the sample is the spawning of a new cmd.exe process using CreateProcessA which executes the tree command on disk C:\ (Figures 18, 19, and 20).

```
7:52:1...  Ransomware.w...  3628  Process Create  C:\Windows\SysWOW64\cmd.exe

SUCCESS  PID: 768, Command line: C:\Windows\system32\cmd.exe /c tree C:\
```

Figure 18 - Sample spawning a new cmd.exe process that calls the tree command with the C:\ drive as the argument.

```
EIP  75303D50  8BFF      mov edi,edi                                  CreateProcessA
     75303D52  55        push ebp
     75303D53  8BEC      mov ebp,esp
     75303D55  5D        pop ebp
     75303D56  FF25 88143575  jmp dword ptr ds:[<&CreateProcessA>]    JMP.&CreateProcessA
     75303D5C  CC        int3
     75303D5D  CC        int3
     75303D5E  CC        int3
```

Figure 19 - CreateProcessA call used to spawn a new cmd.exe process.

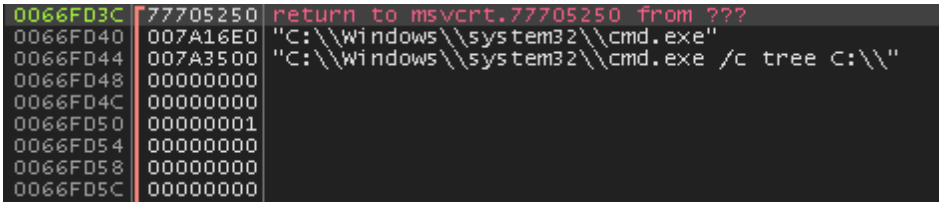

Figure 20 - Stack arguments containing the command argument to be included when spawning the new cmd.exe process.

Finally, once the tree command is finished, the sample calls ExitProcess to terminate.

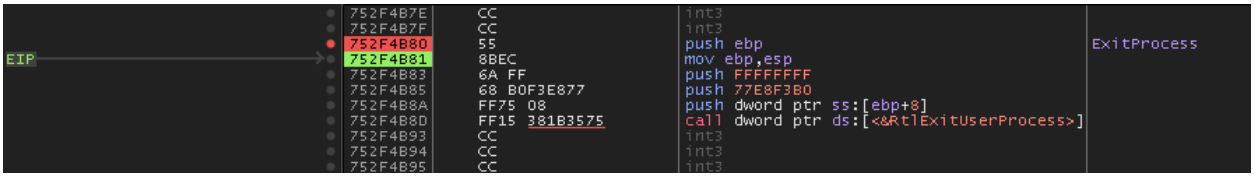

Figure 21 - ExitProcess called after all operations are finished.

MITRE ATT&CK Techniques

Here a list of all MITRE ATT&CK classifications of the malware sample will be included.

ATT&CK ID	Tactic Name	Description of use
T1059.001	Command and Scripting Interpreter: Powershell	Sample saves and executes a Powershell script.
T1059.003	Command and Scripting Interpreter: Windows Command Shell	Sample executes commands via cmd.exe.
T1486	Data Encrypted for Impact	Sample encrypts data on target systems.

YARA Signature

```
rule WANNAHUSKY
{

    meta:
        author = "KrknSec"
        info = "WANNAHUSKY Ransomware"

    strings:
        $s1 = "_NimMain." ascii
        $s2 = "@mwannahusky.nim.c_asgnRef0" ascii
        $s3 = "@Desktop\\ps1.ps1" ascii
        $s4 = "@Desktop\\WANNAHUSKY.png" ascii
        $s5 = "@Desktop\\target\\cosmo.WANNAHUSKY" ascii
        $s6 = "@Desktop\\cosmo.jpeg" ascii

    condition:
        all of them
}
```

Extractor Script

```python
import pefile
import binascii

def getRdataSection(file):
    pe_file = pefile.PE(file)
    for section in pe_file.sections:
        if b".rdata" in section.Name:
            return section.get_data()

def main():
    # Get path of executiable
    sample = input("Enter path of executable: ")

    # Find the .rdata section
    rdata_section = getRdataSection(sample)

    # Extract the embedded PNG file
    pngFile = rdata_section[4360:]
    fullData = pngFile[0:36839]
    f = open("ransomNote.png", "wb")
    f.write(pngFile)
    f.close()

    # Extract the Ransom target
    target = rdata_section[37071:]
    targetData = target[0:49]
    print("\n[+] Targeted file is: \n" + str(targetData) + "\n")

    # Extract the Powershell Script
    pshell = rdata_section[3753:]
    pshellData = pshell[0:559]
    f = open("extractedPowershell.txt", "wb")
    f.write(pshellData)
    f.close()

main()
```

Indicators of Compromise

- WANNAHUSKY.png
  - MD5: 28b6be9ee7d9fc481b51f077a077191e
  - SHA1: 2897ee9611876a8be3111ca9738e2c0942c5b71e
  - Location: %USERPROFILE%\Desktop\
- Ps1.ps1
  - MD5: 7c1bbff5820495dd9f7a294777a49d33
  - SHA1: 23436dcbe4a79b5a6a4930909f0b6a7bff2434b1
  - Location: %USERPROFILE%\Desktop\
  - **Deleted after execution**
- .WANNAHUSKY file extension