Practical Malware Analysis & Triage

Malware Analysis Report

Ransomware.WannaCry

Feb 2023 | thisIsBaggio | v1.0

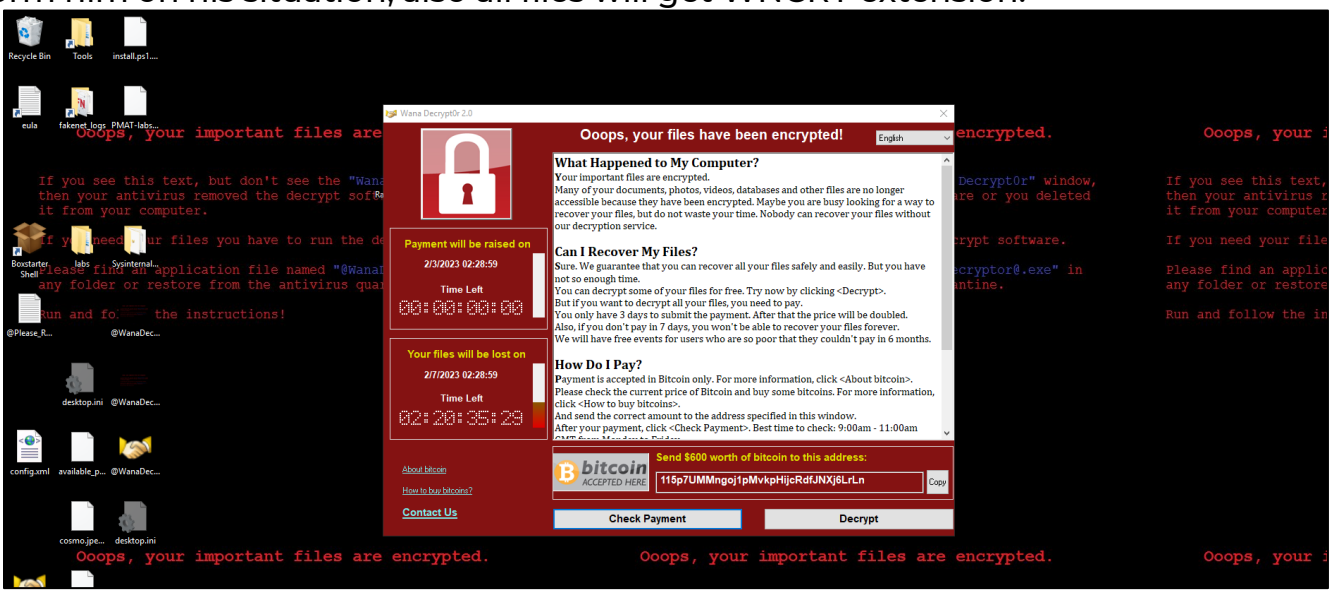
Table of Contents

Table of Contents 2
Executive Summary 3
High-Level Technical Summary 4
Malware Composition 6
Ransomware.wannacry.exe 6
tasksche.exe 6
t.wnry 6
Basic Static Analysis 7
Ransomware.wannacry.exe 7
tasksche.exe 10
t.wnry 11
Basic Dynamic Analysis 13
Advanced Static Analysis 17
Ransomware.wannacry.exe 17
tasksche.exe 21
Advanced Dynamic Analysis 32
Indicators of Compromise 33
Network Indicators 33
Host-based Indicators 33
Rules & Signatures 35

Executive Summary

SHA256 hash	24d004a104d4d54034dbcffc2a4b19a11f39008a575aa614ea04703480b1022c

WannaCry is a crypto ransomware that had initial outbreak between 12 and 15 of May 2017. It is a binary file that consists of multiple embedded files that will be dropped to various directories after detonation and used later for persistence and spreading purposes. Main functionality of this malware is to encrypt user files and demand ransom for decrypting them. After infection user will be presented by popup and background change that will inform him on his situation, also all files will get WNCRY extension:



Goal of this analysis was to get understanding of infection process and create rules applicable for detection. There are parts of the malware that were not analysed (tor executables, other wnry files, process of communicating with hackers).

High-Level Technical Summary

WannaCry have multiple stages of operation.

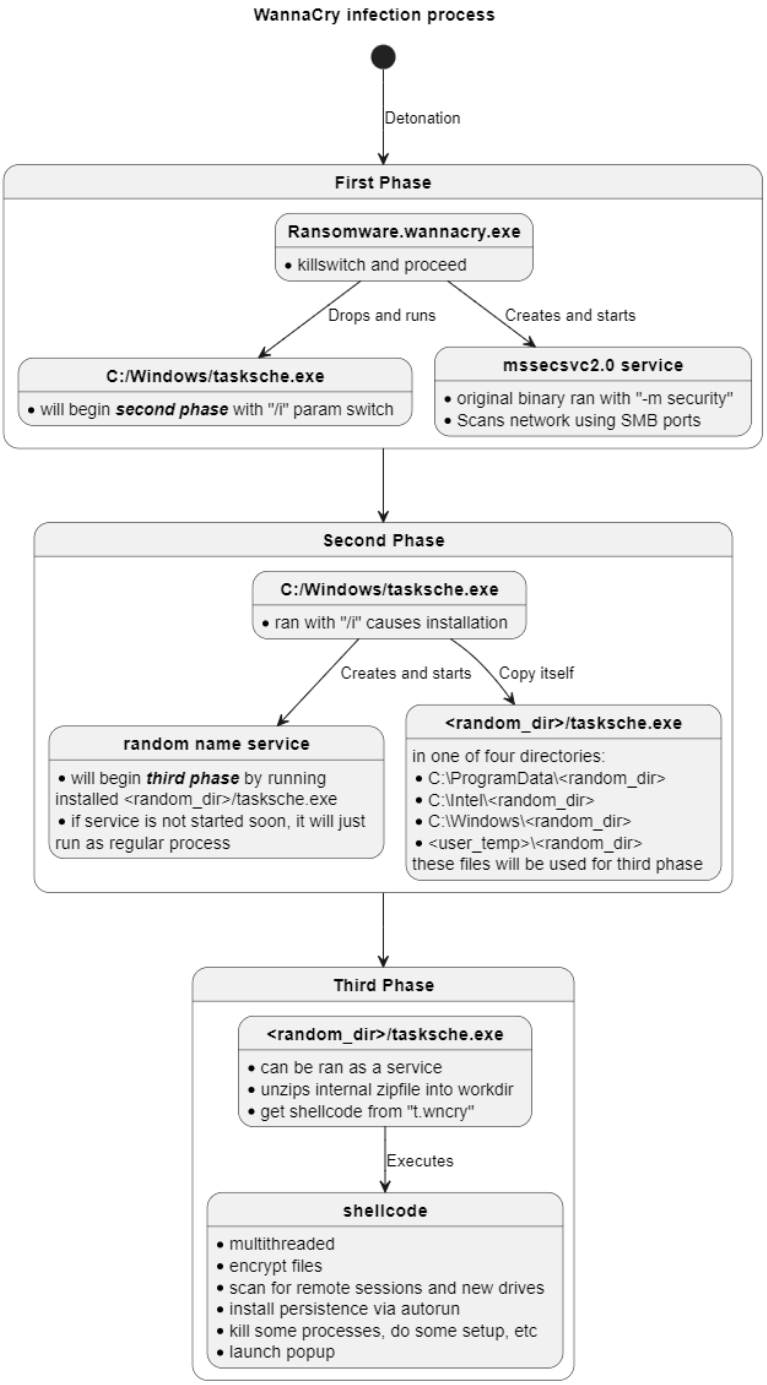


Fig 1: High-level infection chain overview

WannaCry Crypto Malware
Feb 2023
v1.0

Malware Composition

Main WannaCry modules related to infection chain are as following:

File Name	SHA256 Hash
Ransomware.wannacry.exe	24d004a104d4d54034dbcffc2a4b19a11f39008a575aa614ea04703480b1022c
tasksche.exe	ed01ebfbc9eb5bbea545af4d01bf5f1071661840480439c6e5babe8e080e41aa
t.wnry	97ebce49b14c46bebc9ec2448d00e1e397123b256e2be9eba5140688e7bc0ae6

Ransomware.wannacry.exe

The initial module that is ran by user, it will probably try to spread via SMB ports to other machines and will drop tasksche.exe file that will continue infection.

tasksche.exe

Dropped by initial module, contains most of the logic that finalizes setup of malware, it will copy itself into final randomly named directory where it will also put all embedded malware files. It will also create service with the name same as the directory and this service will be a launcher for the shellcode.

t.wnry

File containing encrypted shellcode that will do the heavy-lifting like actual encryption of the files and final setup before and after encryption. It is loaded to memory space of tasksche.exe process.

Basic Static Analysis

Ransomware.wannacry.exe

Interesting strings from floss:
\\172.16.99.5\IPC$
\\192.168.56.20\IPC$
WanaCrypt0r
diskpart.exe
lhdfrgui.exe
SMBr
NT LM 0.12
SMBs
SMB2
http://www.iuqerfsodp9ifjaposdfjhgosurijfaewrwergwea.com

Also, many strings of file extensions.

CAPA output:

```
+------------------------+-------------------------------------------------------------------------------------+
| md5                    | db349b97c37d22f5ea1d1841e3c89eb4                                                    |
| sha1                   | e889544aff85ffaf8b0d0da705105dee7c97fe26                                            |
| sha256                 | 24d004a104d4d54034dbcffc2a4b19a11f39008a575aa614ea04703480b1022c                    |
| os                     | windows                                                                             |
| format                 | pe                                                                                  |
| arch                   | i386                                                                                |
| path                   | Ransomware.wannacry.exe                                                             |
+------------------------+-------------------------------------------------------------------------------------+

+------------------------+-------------------------------------------------------------------------------------+
| ATT&CK Tactic          | ATT&CK Technique                                                                    |
|------------------------+-------------------------------------------------------------------------------------|
| DEFENSE EVASION        | Obfuscated Files or Information::Indicator Removal from Tools T1027.005             |
| DISCOVERY              | File and Directory Discovery T1083                                                  |
|                        | System Information Discovery T1082                                                 |
|                        | System Network Configuration Discovery T1016                                        |
| EXECUTION              | Shared Modules T1129                                                                |
|                        | System Services::Service Execution T1569.002                                        |
| PERSISTENCE            | Create or Modify System Process::Windows Service T1543.003                          |
+------------------------+-------------------------------------------------------------------------------------+
```

Fig 2a: CAPA output for main module

MBC Objective	MBC Behavior
ANTI-BEHAVIORAL ANALYSIS	Conditional Execution::Runs as Service [B0025.007]
	Debugger Detection::Timing/Delay Check QueryPerformanceCounter [B0001.033]
ANTI-STATIC ANALYSIS	Disassembler Evasion::Argument Obfuscation [B0012.001]
COMMAND AND CONTROL	C2 Communication::Receive Data [B0030.002]
	C2 Communication::Send Data [B0030.001]
COMMUNICATION	HTTP Communication::Create Request [C0002.012]
	HTTP Communication::Open URL [C0002.004]
	Socket Communication::Connect Socket [C0001.004]
	Socket Communication::Create TCP Socket [C0001.011]
	Socket Communication::Create UDP Socket [C0001.010]
	Socket Communication::Get Socket Status [C0001.012]
	Socket Communication::Initialize Winsock Library [C0001.009]
	Socket Communication::Receive Data [C0001.006]
	Socket Communication::Send Data [C0001.007]
	Socket Communication::Set Socket Config [C0001.001]
	Socket Communication::TCP Client [C0001.008]
CRYPTOGRAPHY	Generate Pseudo-random Sequence::Use API [C0021.003]
DATA	Compression Library [C0060]
DISCOVERY	Code Discovery::Inspect Section Memory Permissions [B0046.002]
EXECUTION	Install Additional Program [B0023]
FILE SYSTEM	Move File [C0063]
	Read File [C0051]
PROCESS	Create Thread [C0038]
	Terminate Process [C0018]
	Terminate Thread [C0039]

Fig 2b: CAPA output for main module

CAPABILITY	NAMESPACE
check for time delay via QueryPerformanceCounter	anti-analysis/anti-debugging/debugger-detection
contain obfuscated stackstrings	anti-analysis/obfuscation/string/stackstring
receive data (5 matches)	communication
send data (5 matches)	communication
connect to URL	communication/http/client
get socket status	communication/socket
initialize Winsock library	communication/socket
set socket configuration	communication/socket
create UDP socket (4 matches)	communication/socket/udp/send
act as TCP client	communication/tcp/client
generate random numbers via WinAPI	data-manipulation/prng
contain a resource (.rsrc) section	executable/pe/section/rsrc
extract resource via kernel32 functions	executable/resource
contain an embedded PE file	executable/subfile/pe
get file size	host-interaction/file-system/meta
move file	host-interaction/file-system/move
read file on Windows	host-interaction/file-system/read
get number of processors	host-interaction/hardware/cpu
terminate process	host-interaction/process/terminate
run as service	host-interaction/service
create service	host-interaction/service/create
modify service	host-interaction/service/modify
start service	host-interaction/service/start
create thread (4 matches)	host-interaction/thread/create
terminate thread	host-interaction/thread/terminate
link function at runtime on Windows	linking/runtime-linking
linked against ZLIB	linking/static/zlib
inspect section memory permissions	load-code/pe
persist via Windows service	persistence/service

Fig 2c: CAPA output for main module

PEStudio main indicators:

indicator (73)	detail	level
file > extensions (Ransomware | Wiper) > count	159	1
imports > flag	28	1
strings > flag	61	1
library > flag	IP Helper API	1
library > flag	Internet Extensions for Win32 Library	1
resource > size > suspicious	R.1831, 3514368 bytes	1
library > flag	Windows Socket Library	1
URL > pattern	http://www.iuqerfsodp9ifjaposdfjhgosurijfaewrwergwea.com	1
file > embedded	signature: executable, location: .data, offset: 0x0000B020, size: 5263716	1
file > embedded	signature: executable, location: .data, offset: 0x0000F080, size: 159744	1
file > embedded	signature: executable, location: .rsrc, offset: 0x000320A4, size: 3514368	1
imports > anonymous	13	2
string > size > suspicious	1403 bytes	2
string > size > suspicious	1403 bytes	2
string > size > suspicious	1430 bytes	2
string > size > suspicious	1554 bytes	2
string > size > suspicious	2693 bytes	2
string > size > suspicious	2693 bytes	2
string > size > suspicious	2988 bytes	2
resources > file-ratio	94.41%	2
overlay > signature > name	executable	2

Fig 3: PEStudio indictaros for main module

This file is well known to virustotal as well.

Fig 4: VirusTotal result for main module

We can see that this piece of malware will probably try to contact URL found in the strings (this a a killswitch actually). Also we see strings related to SMB shares which are probably used to spread the malware, probably via some exploit. It also contains embedded executables according to PEStudio. Apart from that we see that this malware probably can contact C2 server and also have service creation as well as encryption abilities. List of encrypted extensions sems to be emedded in the string as well.

tasksche.exe

Looks like strings that occur in this file also occurred in the previous list, so it is stored inside the main executable without encryption. This file is also known to virustotal:

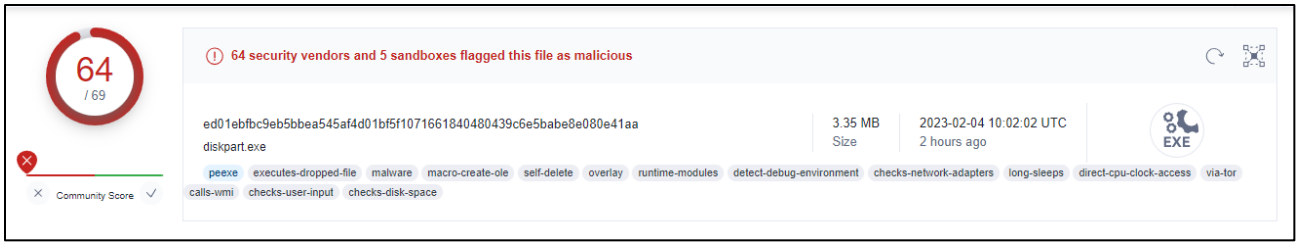


Fig 5: VirusTotal result for dropped tasksche

t.wnry

There is only single string at offset 0 that seems to be magic of the file:

WANACRY!

Apart from that file seems to be encrypted, its very high entropy over whole file confirms that:

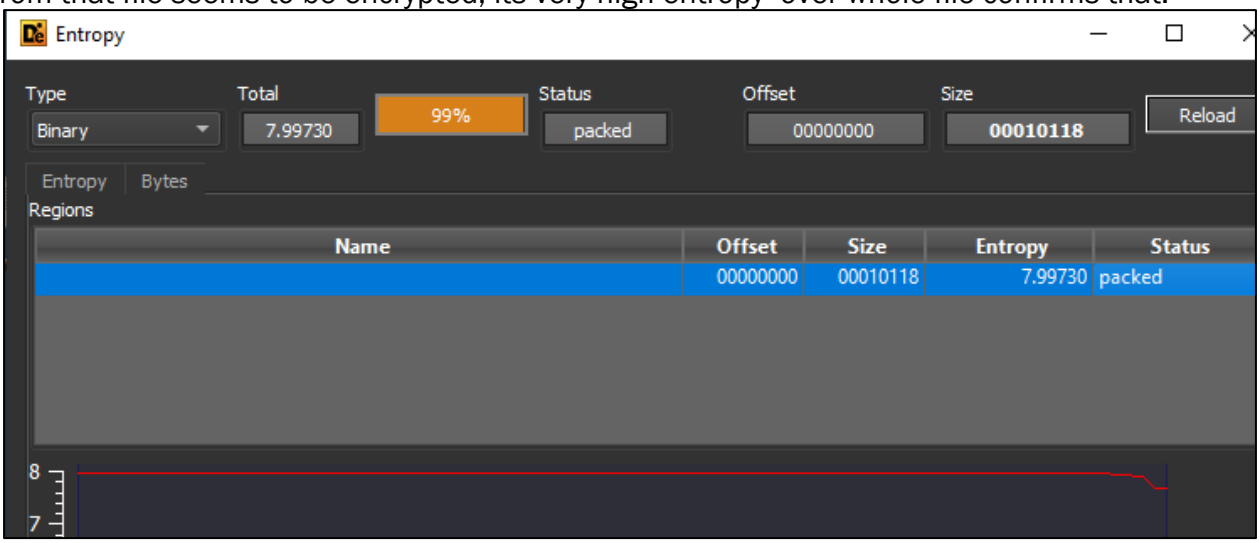


Fig 6: Entropy of shellcode file using DIE tool

Some vendors recognize this file signature as malicious even though it is encrypted.

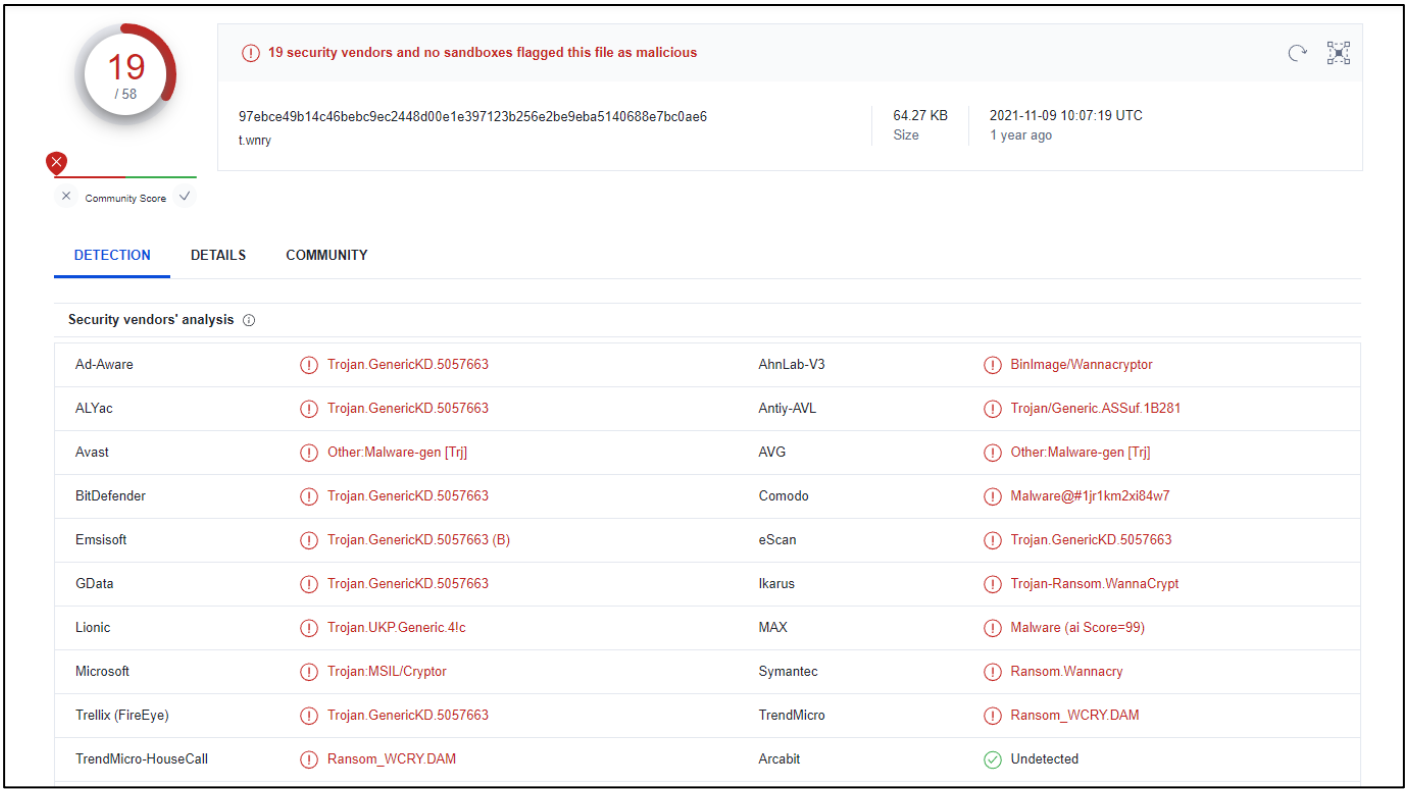


Fig 7: VirusTotal result for t.wnry

Basic Dynamic Analysis

It seems that malware tries to contact http://www.iuqerfsodp9ifjaposdfjhgosurijfaewrwergwea.com URL and if it is found it just exits.

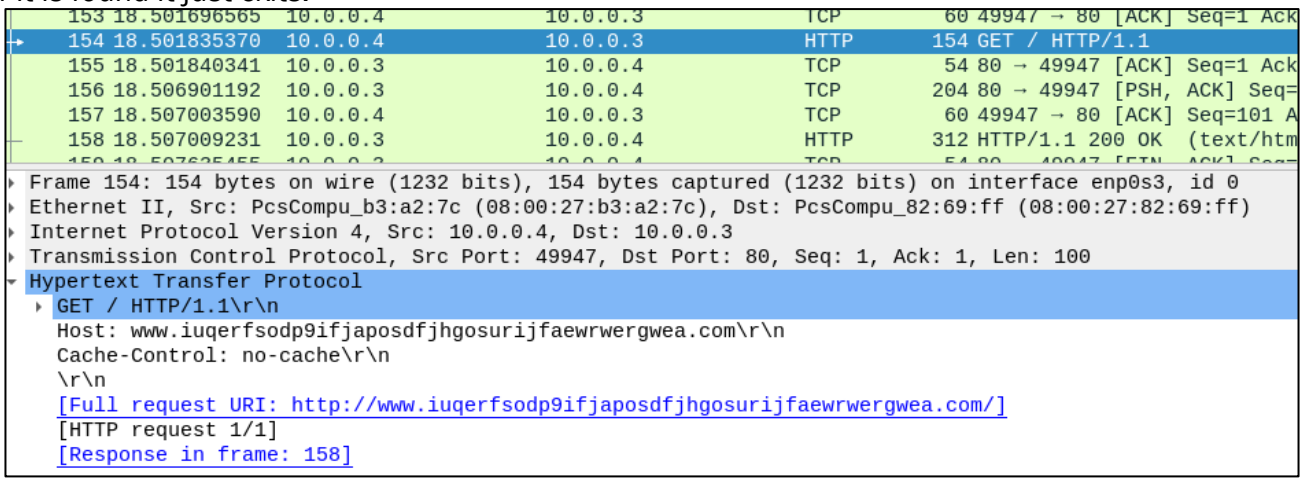


Fig 8: HTTP request to killswitch URL

If malware is ran without access to above URL and with administrative privileges, it starts infection chain. It drops C:\Windows\tasksche.exe and runs it:

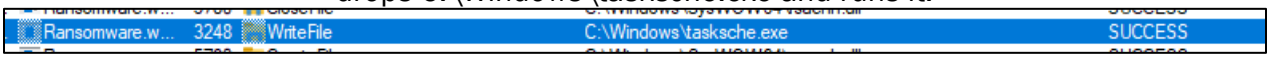


Fig 9: Dropped executable

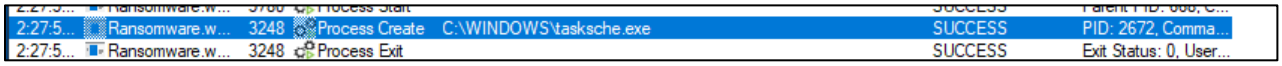


Fig 10: Creation of process based on dropped executable

Tcpview will show that process of scanning various IPs began, and is done by service that seems to be created by running the malware (mssecsvc2.0):

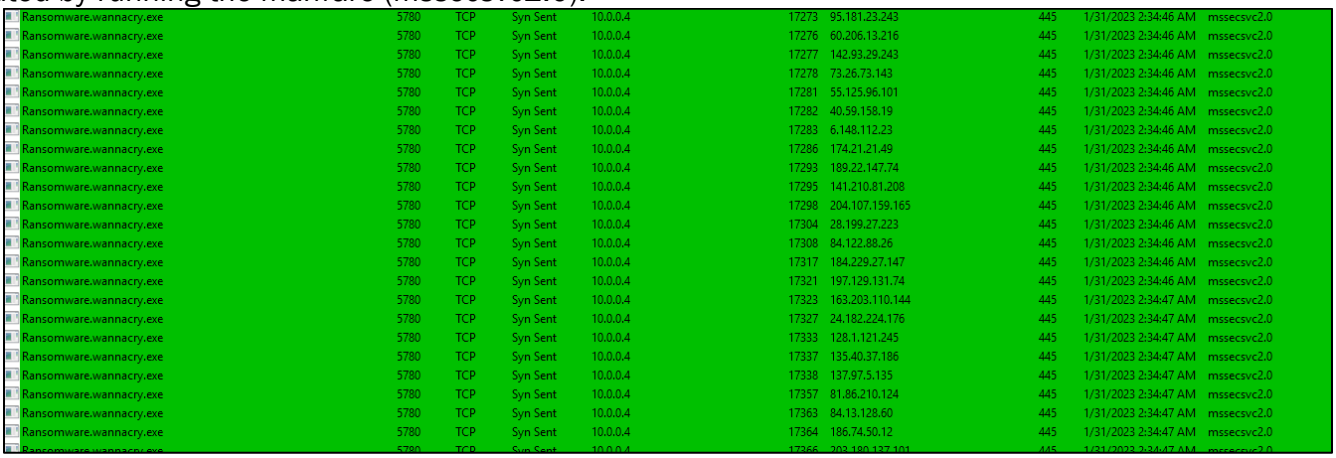


Fig 11: TCPView showing many SMB connections being created to semi-random IPs

C:\Windows\tasksche.exe will create hidden randomly named directory with these files:

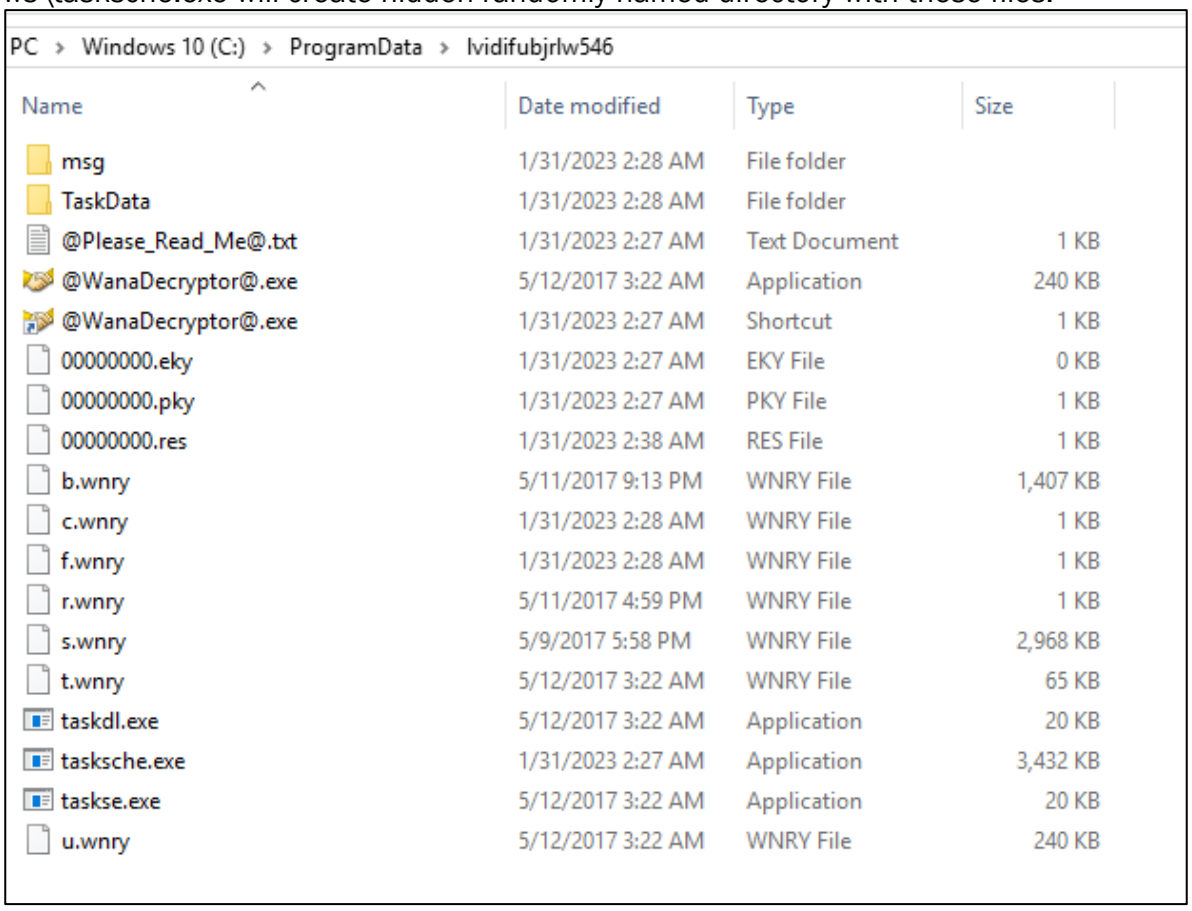


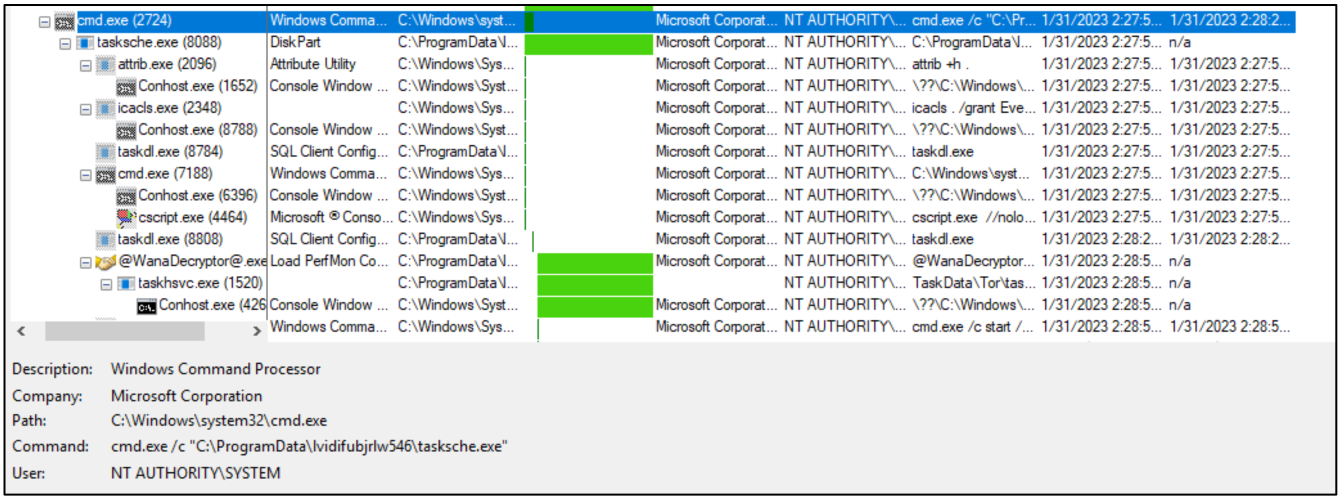
Fig 12: Contents of final drop in randomly named directory

C:\Windows\tasksche.exe is the same binary as newly created one in randomly named directory:

```
C:\Users\IEUser>comp C:\ProgramData\lvidifubjrlw546\tasksche.exe C:\Windows\tasksche.exe
Comparing C:\ProgramData\lvidifubjrlw546\tasksche.exe and C:\Windows\tasksche.exe...
Files compare OK
```

Fig 13: Binary comparison of two tasksche.exe dropped in different places

tasksche.exe from this randomly named directory is then ran by services.exe (randomly named service created by original C:\Windows\tasksche.exe, this random name is the same as random directory name):

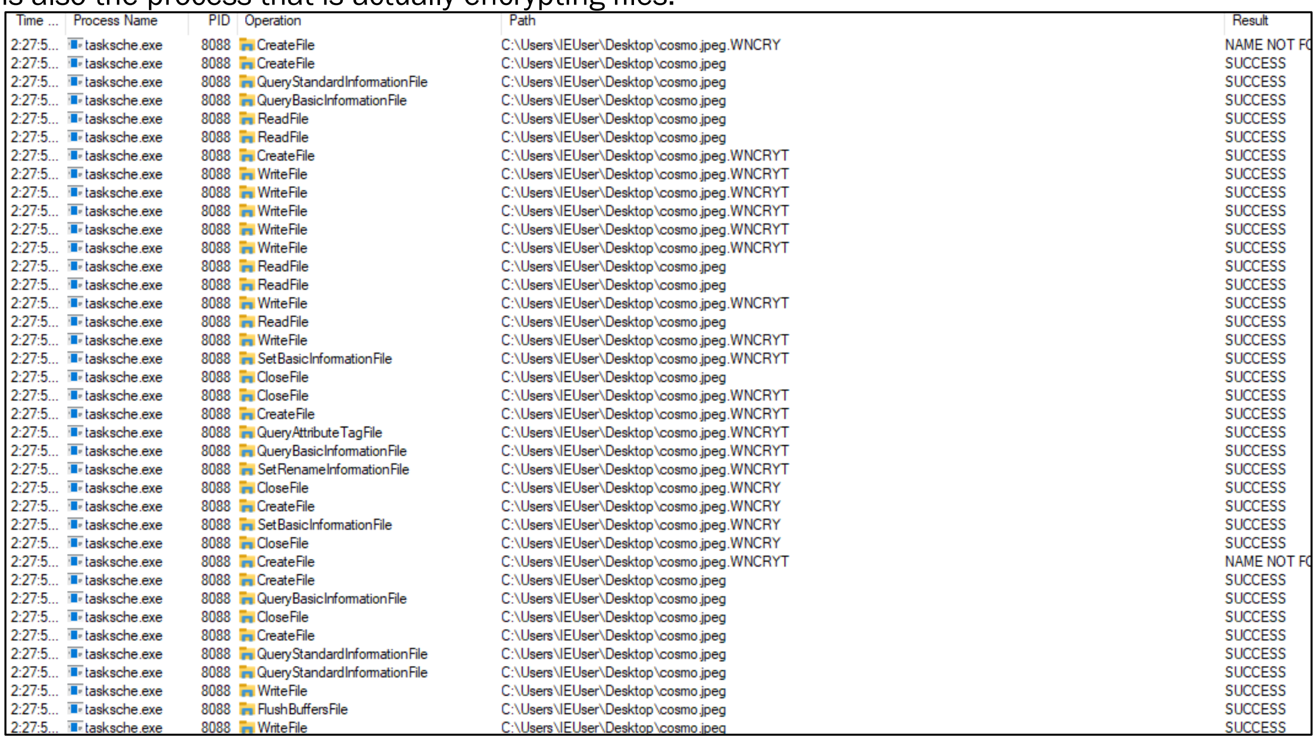


cmd.exe (2724)	Windows Comma...	C:\Windows\syst...	Microsoft Corporat...	NT AUTHORITY\...	cmd.exe /c "C:\Pr...	1/31/2023 2:27:5...	1/31/2023 2:28:2...
taskche.exe (8088)	DiskPart	C:\ProgramData\l...	Microsoft Corporat...	NT AUTHORITY\...	C:\ProgramData\l...	1/31/2023 2:27:5...	n/a
attrib.exe (2096)	Attribute Utility	C:\Windows\Sys...	Microsoft Corporat...	NT AUTHORITY\...	attrib +h .	1/31/2023 2:27:5...	1/31/2023 2:27:5...
Conhost.exe (1652)	Console Window ...	C:\Windows\Syst...	Microsoft Corporat...	NT AUTHORITY\...	\??\C:\Windows\...	1/31/2023 2:27:5...	1/31/2023 2:27:5...
icacls.exe (2348)		C:\Windows\Sys...	Microsoft Corporat...	NT AUTHORITY\...	icacls . /grant Eve...	1/31/2023 2:27:5...	1/31/2023 2:27:5...
Conhost.exe (8788)	Console Window ...	C:\Windows\Syst...	Microsoft Corporat...	NT AUTHORITY\...	\??\C:\Windows\...	1/31/2023 2:27:5...	1/31/2023 2:27:5...
taskdl.exe (8784)	SQL Client Config...	C:\ProgramData\l...	Microsoft Corporat...	NT AUTHORITY\...	taskdl.exe	1/31/2023 2:27:5...	1/31/2023 2:27:5...
cmd.exe (7188)	Windows Comma...	C:\Windows\Sys...	Microsoft Corporat...	NT AUTHORITY\...	C:\Windows\syst...	1/31/2023 2:27:5...	1/31/2023 2:27:5...
Conhost.exe (6396)	Console Window ...	C:\Windows\Syst...	Microsoft Corporat...	NT AUTHORITY\...	\??\C:\Windows\...	1/31/2023 2:27:5...	1/31/2023 2:27:5...
cscript.exe (4464)	Microsoft ® Conso...	C:\Windows\Sys...	Microsoft Corporat...	NT AUTHORITY\...	cscript.exe //nolo...	1/31/2023 2:27:5...	1/31/2023 2:27:5...
taskdl.exe (8808)	SQL Client Config...	C:\ProgramData\l...	Microsoft Corporat...	NT AUTHORITY\...	taskdl.exe	1/31/2023 2:28:2...	1/31/2023 2:28:2...
@WanaDecryptor@.exe	Load PerfMon Co...	C:\ProgramData\l...	Microsoft Corporat...	NT AUTHORITY\...	@WanaDecryptor...	1/31/2023 2:28:5...	n/a
taskhsvc.exe (1520)		C:\ProgramData\l...		NT AUTHORITY\...	TaskData\Tor\tas...	1/31/2023 2:28:5...	n/a
Conhost.exe (426	Console Window ...	C:\Windows\Syst...	Microsoft Corporat...	NT AUTHORITY\...	\??\C:\Windows\...	1/31/2023 2:28:5...	n/a
	Windows Comma...	C:\Windows\Sys...	Microsoft Corporat...	NT AUTHORITY\...	cmd.exe /c start /...	1/31/2023 2:28:5...	1/31/2023 2:28:5...

Description: Windows Command Processor
Company: Microsoft Corporation
Path: C:\Windows\system32\cmd.exe
Command: cmd.exe /c "C:\ProgramData\lvidifubjrlw546\tasksche.exe"
User: NT AUTHORITY\SYSTEM

Fig 14: Process tree for final service created

This is also the process that is actually encrypting files:



Time ...	Process Name	PID	Operation	Path	Result
2:27:5...	tasksche.exe	8088	CreateFile	C:\Users\IEUser\Desktop\cosmo.jpeg.WNCRY	NAME NOT F
2:27:5...	tasksche.exe	8088	CreateFile	C:\Users\IEUser\Desktop\cosmo.jpeg	SUCCESS
2:27:5...	tasksche.exe	8088	QueryStandardInformationFile	C:\Users\IEUser\Desktop\cosmo.jpeg	SUCCESS
2:27:5...	tasksche.exe	8088	QueryBasicInformationFile	C:\Users\IEUser\Desktop\cosmo.jpeg	SUCCESS
2:27:5...	tasksche.exe	8088	ReadFile	C:\Users\IEUser\Desktop\cosmo.jpeg	SUCCESS
2:27:5...	tasksche.exe	8088	ReadFile	C:\Users\IEUser\Desktop\cosmo.jpeg	SUCCESS
2:27:5...	tasksche.exe	8088	CreateFile	C:\Users\IEUser\Desktop\cosmo.jpeg.WNCRYT	SUCCESS
2:27:5...	tasksche.exe	8088	WriteFile	C:\Users\IEUser\Desktop\cosmo.jpeg.WNCRYT	SUCCESS
2:27:5...	tasksche.exe	8088	WriteFile	C:\Users\IEUser\Desktop\cosmo.jpeg.WNCRYT	SUCCESS
2:27:5...	tasksche.exe	8088	WriteFile	C:\Users\IEUser\Desktop\cosmo.jpeg.WNCRYT	SUCCESS
2:27:5...	tasksche.exe	8088	WriteFile	C:\Users\IEUser\Desktop\cosmo.jpeg.WNCRYT	SUCCESS
2:27:5...	tasksche.exe	8088	WriteFile	C:\Users\IEUser\Desktop\cosmo.jpeg.WNCRYT	SUCCESS
2:27:5...	tasksche.exe	8088	ReadFile	C:\Users\IEUser\Desktop\cosmo.jpeg	SUCCESS
2:27:5...	tasksche.exe	8088	ReadFile	C:\Users\IEUser\Desktop\cosmo.jpeg	SUCCESS
2:27:5...	tasksche.exe	8088	WriteFile	C:\Users\IEUser\Desktop\cosmo.jpeg.WNCRYT	SUCCESS
2:27:5...	tasksche.exe	8088	ReadFile	C:\Users\IEUser\Desktop\cosmo.jpeg	SUCCESS
2:27:5...	tasksche.exe	8088	WriteFile	C:\Users\IEUser\Desktop\cosmo.jpeg.WNCRYT	SUCCESS
2:27:5...	tasksche.exe	8088	SetBasicInformationFile	C:\Users\IEUser\Desktop\cosmo.jpeg.WNCRYT	SUCCESS
2:27:5...	tasksche.exe	8088	CloseFile	C:\Users\IEUser\Desktop\cosmo.jpeg	SUCCESS
2:27:5...	tasksche.exe	8088	CloseFile	C:\Users\IEUser\Desktop\cosmo.jpeg.WNCRYT	SUCCESS
2:27:5...	tasksche.exe	8088	CreateFile	C:\Users\IEUser\Desktop\cosmo.jpeg.WNCRYT	SUCCESS
2:27:5...	tasksche.exe	8088	QueryAttributeTagFile	C:\Users\IEUser\Desktop\cosmo.jpeg.WNCRYT	SUCCESS
2:27:5...	tasksche.exe	8088	QueryBasicInformationFile	C:\Users\IEUser\Desktop\cosmo.jpeg.WNCRYT	SUCCESS
2:27:5...	tasksche.exe	8088	SetRenameInformationFile	C:\Users\IEUser\Desktop\cosmo.jpeg.WNCRYT	SUCCESS
2:27:5...	tasksche.exe	8088	CloseFile	C:\Users\IEUser\Desktop\cosmo.jpeg.WNCRY	SUCCESS
2:27:5...	tasksche.exe	8088	CreateFile	C:\Users\IEUser\Desktop\cosmo.jpeg.WNCRY	SUCCESS
2:27:5...	tasksche.exe	8088	SetBasicInformationFile	C:\Users\IEUser\Desktop\cosmo.jpeg.WNCRY	SUCCESS
2:27:5...	tasksche.exe	8088	CloseFile	C:\Users\IEUser\Desktop\cosmo.jpeg.WNCRY	SUCCESS
2:27:5...	tasksche.exe	8088	CreateFile	C:\Users\IEUser\Desktop\cosmo.jpeg.WNCRYT	NAME NOT F
2:27:5...	tasksche.exe	8088	CreateFile	C:\Users\IEUser\Desktop\cosmo.jpeg	SUCCESS
2:27:5...	tasksche.exe	8088	QueryBasicInformationFile	C:\Users\IEUser\Desktop\cosmo.jpeg	SUCCESS
2:27:5...	tasksche.exe	8088	CloseFile	C:\Users\IEUser\Desktop\cosmo.jpeg	SUCCESS
2:27:5...	tasksche.exe	8088	CreateFile	C:\Users\IEUser\Desktop\cosmo.jpeg	SUCCESS
2:27:5...	tasksche.exe	8088	QueryStandardInformationFile	C:\Users\IEUser\Desktop\cosmo.jpeg	SUCCESS
2:27:5...	tasksche.exe	8088	QueryStandardInformationFile	C:\Users\IEUser\Desktop\cosmo.jpeg	SUCCESS
2:27:5...	tasksche.exe	8088	WriteFile	C:\Users\IEUser\Desktop\cosmo.jpeg	SUCCESS
2:27:5...	tasksche.exe	8088	FlushBuffersFile	C:\Users\IEUser\Desktop\cosmo.jpeg	SUCCESS
2:27:5...	tasksche.exe	8088	WriteFile	C:\Users\IEUser\Desktop\cosmo.jpeg	SUCCESS

Fig 15: ProcMon showing user files' encryption process

It also installs persistence via registry:

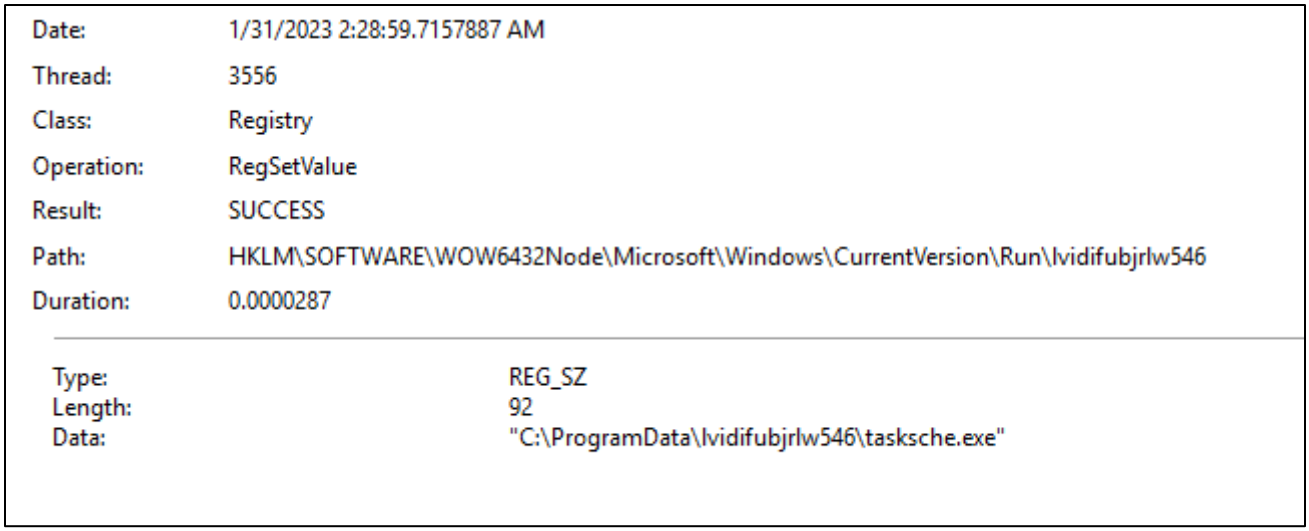


Fig 16: Persistence installation

Advanced Static Analysis

Ransomware.wannacry.exe

First thing that main module is doing is checking the killswitch URL:

```
puVar3 = s_http://www.iuqerfsodp9ifjaposdfj_004313d0;
puVar4 = (char *)local_50;
for (iVar2 = 0xe; iVar2 != 0; iVar2 = iVar2 + -1) {
  *(undefined4 *)puVar4 = *(undefined4 *)puVar3;
  puVar3 = puVar3 + 4;
  puVar4 = puVar4 + 4;
}
*puVar4 = *puVar3;
local_17 = 0;
local_13 = 0;
local_f = 0;
local_b = 0;
local_7 = 0;
local_3 = 0;
local_1 = 0;
uVar1 = InternetOpenA(0,1,0,0,0);
iVar2 = InternetOpenUrlA(uVar1,local_50,0,0,0x84000000,0);
if (iVar2 == 0) {
  InternetCloseHandle(uVar1);
  InternetCloseHandle(0);
  main_function();
  return 0;
}
InternetCloseHandle(uVar1);
InternetCloseHandle(iVar2);
return 0;
```

Fig 17: Killswitch code

Then malware proceeds and if ran without arguments it is going to create “mssecsvc2.0” service that will run the malware itself but with “-m security” param effectively causing it to run the second part of this function (will go back to it later):

```
void main_function(void)

{
  int *piVar1;
  SC_HANDLE hSCManager;
  SC_HANDLE hSCObject;
  SERVICE_TABLE_ENTRYA local_10;
  undefined4 local_8;
  undefined4 local_4;

  GetModuleFileNameA((HMODULE)0x0,&DAT_this_module_filename,0x104);
  piVar1 = (int *)__p___argc();
  if (*piVar1 < 2) {
    create_service_and_run_dropped();
    return;
  }
  hSCManager = OpenSCManagerA((LPCSTR)0x0,(LPCSTR)0x0,0xf003f);
  if (hSCManager != (SC_HANDLE)0x0) {
    hSCObject = OpenServiceA(hSCManager,s_mssecsvc2.0_004312fc,0xf01ff);
    if (hSCObject != (SC_HANDLE)0x0) {
      nullify_svc_failure_action(hSCObject,0x3c);
      CloseServiceHandle(hSCObject);
    }
    CloseServiceHandle(hSCManager);
  }
  local_10.lpServiceName = s_mssecsvc2.0_004312fc;
  local_10.lpServiceProc = MSSECServiceMain;
  local_8 = 0;
  local_4 = 0;
  StartServiceCtrlDispatcherA(&local_10);
```

Fig 18: Two paths of main module

creation of the service:

```
undefined4 create_service(void)

{
  SC_HANDLE hSCManager;
  SC_HANDLE hService;
  char local_104 [260];

  sprintf(local_104,s_%s_-m_security_00431330,&DAT_this_module_filename
  hSCManager = OpenSCManagerA((LPCSTR)0x0,(LPCSTR)0x0,0xf003f);
  if (hSCManager != (SC_HANDLE)0x0) {
    hService = CreateServiceA(hSCManager,s_mssecsvc2.0_004312fc,
                              s_Microsoft_Security_Center_(2.0)_S_00431
                              local_104,(LPCSTR)0x0,(LPDWORD)0x0,(LPCST
                             );
    if (hService != (SC_HANDLE)0x0) {
      StartServiceA(hService,0,(LPCSTR *)0x0);
      CloseServiceHandle(hService);
    }
    CloseServiceHandle(hSCManager);
    return 0;
  }
  return 0;
}
```

Fig 19: Code creating mssecsvc2.0 service

drop (C:\Windows\tasksche.exe) preparation and running with "/i" command argument:

```
  hModule = GetModuleHandleW(u_kernel32.dll_004313b4);
  if (hModule != (HMODULE)0x0) {
    DAT_CreateProcessA_dynamic = GetProcAddress(hModule,s_CreateProcessA_004313a4)
    DAT_CreateFileA_dynamic = GetProcAddress(hModule,s_CreateFileA_00431398);
    DAT_WriteFile_dynamic = GetProcAddress(hModule,s_WriteFile_0043138c);
    DAT_CloseHandle_dynamic = GetProcAddress(hModule,s_CloseHandle_00431380);
    if ((((DAT_CreateProcessA_dynamic != (FARPROC)0x0) && (DAT_CreateFileA_dynamic
        && (DAT_WriteFile_dynamic != (FARPROC)0x0)) && (DAT_CloseHandle_dynamic !=
      hResInfo = FindResourceA((HMODULE)0x0,(LPCSTR)0x727,&DAT_0043137c);
      if (hResInfo != (HRSRC)0x0) {
        hResData = LoadResource((HMODULE)0x0,hResInfo);
```

Fig 20a: Preparing APIs for process drop and creation

```
sprintf(&local_104,s_C:\%s\qeriuwjhrf_00431344);
MoveFileExA(&local_208,&local_104,1);
uVar14 = 2;
uStack644 = 0;
pcStack652 = &local_208;
uStack648 = 0x40000000;
uStack656 = 0x407e49;
iVar4 = (*DAT_CreateFileA_dynamic)();
if (iVar4 != -1) {
  uStack656 = 0;
  (*DAT_WriteFile_dynamic)(iVar4,uVar14,DVar3,&stack0xfffffd84);
  (*DAT_CloseHandle_dynamic)(iVar4);
  pcStack652 = (char *)0x0;
```

Fig 20b: Dropping executable

```
uVar14 = 0;
iVar4 = (*DAT_CreateProcessA_dynamic)
                  (0,acStack572,0,0,0,0x8000000,0,0,&stack0xfffffd8
if (iVar4 != 0) {
  (*DAT_CloseHandle_dynamic)(uVar13);
```

Fig 20c: Running new process

Command: C:\WINDOWS\tasksche.exe /i

Fig 20d: Commandline and location of new process

Going back to the service created, it looks like it will just analyse the network and scan it for SMB port vulnerabilities:

```
void MSSECServiceMain(void)

{
  DAT_svc_status.dwServiceType = SERVICE_WIN32_SHARE_PROCESS;
  DAT_svc_status.dwCurrentState = SERVICE_START_PENDING;
  DAT_svc_status.dwControlsAccepted = SERVICE_ACCEPT_STOP;
  DAT_svc_status.dwWin32ExitCode = 0;
  DAT_svc_status.dwServiceSpecificExitCode = 0;
  DAT_svc_status.dwCheckPoint = 0;
  DAT_svc_status.dwWaitHint = 0;
  DAT_svc_status_handle = RegisterServiceCtrlHandlerA(s_mssecsvc2.
  if (DAT_svc_status_handle != (SERVICE_STATUS_HANDLE)0x0) {
    DAT_svc_status.dwCurrentState = SERVICE_RUNNING;
    DAT_svc_status.dwCheckPoint = 0;
    DAT_svc_status.dwWaitHint = 0;
    SetServiceStatus(DAT_svc_status_handle,&DAT_svc_status);
    run_many_threads();
    Sleep(86400000);
                    /* WARNING: Subroutine does not return */
    ExitProcess(1);
  }
  return;
}
```

Fig 21: Entry function for mssecsvc2.0 service

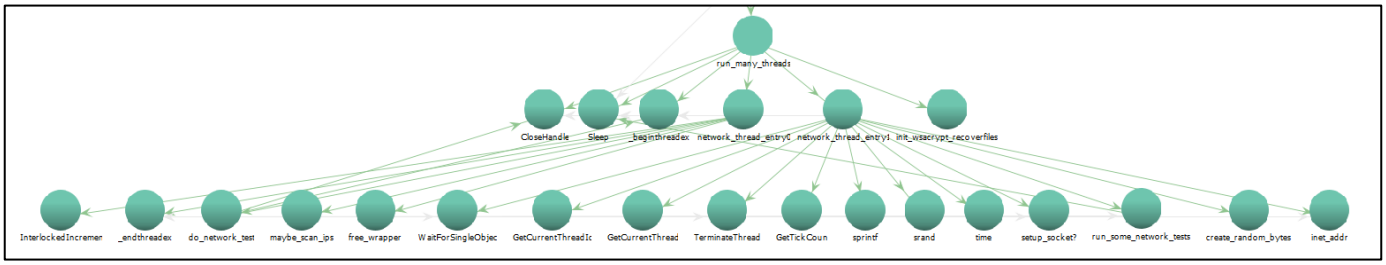


Fig 22: Calltree for the mssecsvc2.0 service

tasksche.exe

When ran with "/i" arg, it will compute pseudorandom hash (<random_dir>) based on computer and install itself in one of those directories (sorted by priority):
C:\ProgramData\<random_dir>
C:\Intel\<random_dir>

C:\Windows\<random_dir>
<user's temp directory>\<random_dir>

```
*(undefined2 *)puVar5 = 0;
*(undefined *)((int)puVar5 + 2) = 0;
GetModuleFileNameA((HMODULE)0x0,&local_210,0x208);
create_pseudorandom_name(&DAT_pseudorandom_dirname);
piVar1 = (int *)__p___argc();
if (*piVar1 == 2) {
  pcVar6 = &DAT_0040f538;
  piVar1 = (int *)__p___argv();
  iVar4 = strcmp(*(char **)(*piVar1 + 4),pcVar6);
  if ((iVar4 == 0) && (iVar4 = install_itself_in_pseudorandom_dir(0), iVar4 != 0))
    CopyFileA(&local_210,s_tasksche.exe_0040f4d8,0);
    DVar2 = GetFileAttributesA(s_tasksche.exe_0040f4d8);
```

Fig 23: Code computing random name and installing malware in final directory

After that, it will create <random_dir> -named service that runs <random_dir>\tasksche.exe without arguments and also it will run tasksche.exe process without arguments if the service won't start within some time (probably as a backup plan):

```
cmd.exe /c "C:\P
rogramData\lvidi
fubjrlw546\tasks
che.exe".......
```

Fig 24: Commandline for second service

```
undefined4 create_svc_and_run_process(void)

{
  int iVar1;
  undefined4 *puVar2;
  CHAR local_20c;
  undefined4 local_20b;

  local_20c = DAT_0040f910;
  puVar2 = &local_20b;
  for (iVar1 = 0x81; iVar1 != 0; iVar1 = iVar1 + -1) {
    *puVar2 = 0;
    puVar2 = puVar2 + 1;
  }
  *(undefined2 *)puVar2 = 0;
  *(undefined *)((int)puVar2 + 2) = 0;
  GetFullPathNameA(s_tasksche.exe_0040f4d8,0x208,&local_20c,(LPSTR *)0
  iVar1 = create_pseudorandom_svc(&local_20c);
  if ((iVar1 != 0) && (iVar1 = wait_for_mutex(0x3c), iVar1 != 0)) {
    return 1;
  }
  iVar1 = startprocess(&local_20c,0,0);
  if ((iVar1 != 0) && (iVar1 = wait_for_mutex(0x3c), iVar1 != 0)) {
    return 1;
  }
  return 0;
```

Fig 24: Code creating new service and running normal process

This process is coordinated by a mutex:

```
undefined4 wait_for_mutex(int param_1)

{
  HANDLE hObject;
  int iVar1;
  char local_68 [100];

  sprintf(local_68,&DAT_0040f4ac,s_Global\MsWinZonesCacheCounterMut_0040f4b4,0);
  iVar1 = 0;
  if (0 < param_1) {
    do {
      hObject = OpenMutexA(0x100000,1,local_68);
      if (hObject != (HANDLE)0x0) {
        CloseHandle(hObject);
        return 1;
      }
      Sleep(1000);
      iVar1 = iVar1 + 1;
    } while (iVar1 < param_1);
  }
  return 0;
}
```

Fig 25: Mutex check

```
push eax                                    eax:"Global\\MsWinZonesCacheCounterMutexA0"
push 1
push 100000
call dword ptr ds:[<&OpenMutexA>]
```

Fig 26: Mutex name

Now analyzing the next phase, service/process without argument:

It will first install/check registry setting for working directory:

Path:	HKLM\SOFTWARE\WOW6432Node\WanaCrypt0r\wd
Duration:	0.0000040
Type:	REG_SZ
Length:	22
Data:	C:\Windows

Fig 27: Regkey used for malware working directory setup

```
RegCreateKeyW(hKey,(LPCWSTR)local_d8,&local_8);
if (local_8 != (HKEY)0x0) {
  if (param_1 == 0) {
    local_10 = 0x207;
    LVar2 = RegQueryValueExA(local_8,&DAT_0040e030,(LPDWORD)0x0,(LPDWORD)0x0,&lo
                             &local_10);
    bVar6 = LVar2 == 0;
    if (bVar6) {
      SetCurrentDirectoryA((LPCSTR)&local_2e0);
    }
  }
  else {
    GetCurrentDirectoryA(0x207,(LPSTR)&local_2e0);
    sVar1 = strlen((char *)&local_2e0);
    LVar2 = RegSetValueExA(local_8,&DAT_0040e030,0,1,&local_2e0,sVar1 + 1);
    bVar6 = LVar2 == 0;
  }
  RegCloseKey(local_8);
  if (bVar6) {
    return 1;
  }
}
```

Fig 28: Code for the check

Then, it will populate <random_dir> with remaining files (dropped from its resources) hide it using attrib and change persmission via icacls:

```
SetCurrentDirectoryA(&local_210);
workingdir_regkey_check(1);
unpack_rsrcs(0,s_WNcry@2ol7_0040f52c);
populate_cwnry();
startprocess(s_attrib_+h_._0040f520,0,0);
startprocess(s_icacls_._/grant_Everyone:F_/T_/C_0040f4fc,0,0);
iVar4 = get_dynamic_procaddresses();
if (iVar4 != 0) {
```

Fig 29: Final drop code

To check whether these packed files are using some known archiving scheme, it was opened by 7zip (also, we have been lucky as string referenced in call to the “unpack_rsrcs” turned out to be a valid a password):

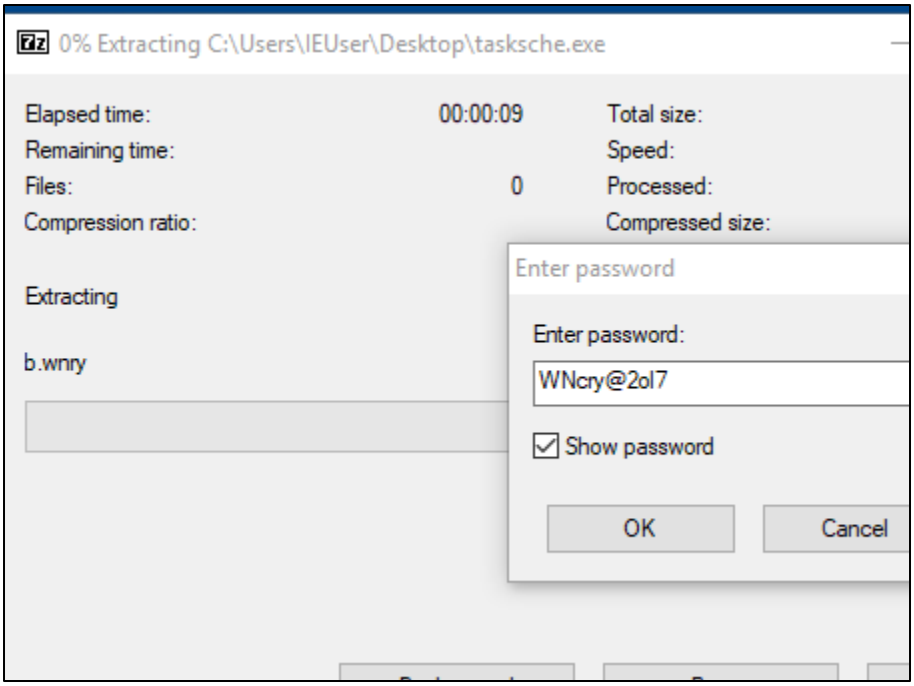


Fig 30: Unzipping the files manually using password used in the code

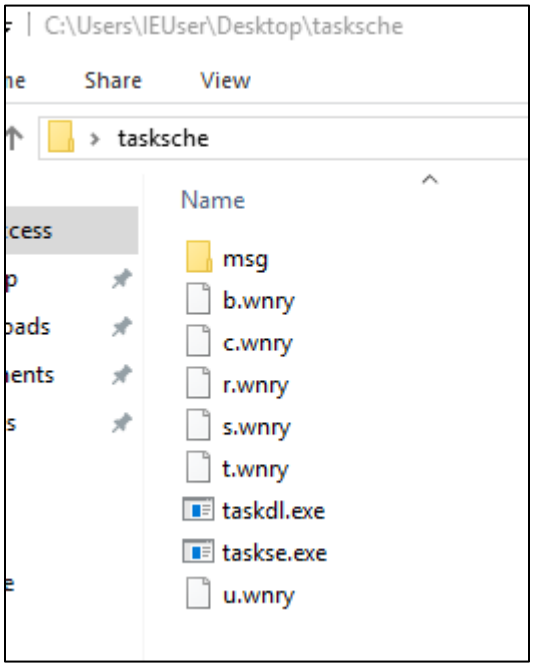


Fig 31: Unzipped files

Then it will load some dynamic functions like below (FUN_00401a45 here does similar things for Crypto library):

```
undefined4 get_dynamic_procaddresses(void)

{
  int iVar1;
  HMODULE hModule;

  iVar1 = FUN_00401a45();
  if (iVar1 != 0) {
    if (_DAT_CreateFileW_dyn != (FARPROC)0x0) {
      return 1;
    }
    hModule = LoadLibraryA(s_kernel32.dll_0040ebe8);
    if (hModule != (HMODULE)0x0) {
      _DAT_CreateFileW_dyn = GetProcAddress(hModule,s_CreateFileW_0040ebdc);
      _DAT_WriteFile_dyn = GetProcAddress(hModule,s_WriteFile_0040ebd0);
      DAT_ReadFile_dyn = GetProcAddress(hModule,s_ReadFile_0040ebc4);
      _DAT_MoveFileW_dyn = GetProcAddress(hModule,s_MoveFileW_0040ebb8);
      _DAT_MoveFileExW = GetProcAddress(hModule,s_MoveFileExW_0040ebac);
      _DAT_DeleteFileW_dyn = GetProcAddress(hModule,s_DeleteFileW_0040eba0);
      _DAT_CloseHandle_dyn = GetProcAddress(hModule,s_CloseHandle_0040eb94);
      if ((((_DAT_CreateFileW_dyn != (FARPROC)0x0) && (_DAT_WriteFile_dyn != (FARPR
          (DAT_ReadFile_dyn != (FARPROC)0x0)) &&
         (((_DAT_MoveFileW_dyn != (FARPROC)0x0 && (_DAT_MoveFileExW != (FARPROC)0x0
          ((_DAT_DeleteFileW_dyn != (FARPROC)0x0 && (_DAT_CloseHandle_dyn != (FARPR
        return 1;
      }
    }
  }
  return 0;
}
```

Fig 32: File and Crypto APIs dynamic preparation

After that it proceed with setting up crypt libraries etc, interesting part is dynamically loaded shellcode (decrypted from t.wnry) being ran:

```
iVar4 = get_dynamic_libraries();
if (iVar4 != 0) {
  FUN_004012fd();
  iVar4 = FUN_00401437(0,0,0);
  if (iVar4 != 0) {
    local_8 = 0;
    iVar4 = FUN_004014a6(s_t.wnry_0040f4f4,&local_8);
    if (((iVar4 != 0) && (iVar4 = FUN_00402lbd(iVar4,loc
        (pcVar3 = (code *)FUN_00402924(iVar4,s_TaskStart_
      (*pcVar3)(0,0);
    }
```

Fig 33: Loading and decrypting shellcode from t.wnry and running it

Running this shellcode will cause process to create many threads doing various things, like encrypting files, scanning for new logical devices or remote sessions, installing persistence via autorun regkey, killing processes, starting message popup etc, API calls used there were analyzed by using debugger:

```
create_process(s_taskkill.exe_/f_/im_Microsoft.Ex_1000d874,0,0);
create_process(s_taskkill.exe_/f_/im_MSExchange*_1000d854,0,0);
create_process(s_taskkill.exe_/f_/im_sqlserver.ex_1000d830,0,0);
create_process(s_taskkill.exe_/f_/im_sqlwriter.ex_1000d80c,0,0);
create_process(s_taskkill.exe_/f_/im_mysqld.exe_1000d7ec,0,0);
```

Fig 34: Shellcode killing various processes

```
pcVar2 = DAT_CreateThread;
iVar4 = (*DAT_CreateThread)(0,0,thread_func0,0,0,0);
pcVar3 = DAT_CloseHandle;
if (iVar4 != 0) {
  (*DAT_CloseHandle)(iVar4);
}
pcVar1 = DAT_Sleep;
(*DAT_Sleep)(100);
iVar4 = (*pcVar2)(0,0,thread_func1,0,0,0);
if (iVar4 != 0) {
  (*pcVar3)(iVar4);
}
(*pcVar1)(100);
iVar4 = (*pcVar2)(0,0,thread_func2,0,0,0);
(*pcVar1)(100);
iVar7 = (*pcVar2)(0,0,run_taskdl_and_sleep,0,0,0);
if (iVar7 != 0) {
  (*pcVar3)(iVar7);
}
(*pcVar1)(100);
iVar7 = (*pcVar2)(0,0,FUN_10004990,0,0,0);
if (iVar7 != 0) {
  (*pcVar3)(iVar7);
}
(*pcVar1)(100);
FUN_100057c0();
```

Fig 35: Shellcode creating various threads

```
void add_run_regkey(undefined4 param_1)

{
  int iVar1;
  undefined4 *puVar2;
  undefined4 *puVar3;
  undefined4 local_498;
  undefined local_464;
  undefined4 local_463;
  undefined local_400 [1024];

  puVar2 = (undefined4 *)s_HKCU\SOFTWARE\Microsoft\Windows\_1000d57c;
  puVar3 = &local_498;
  for (iVar1 = 0xc; iVar1 != 0; iVar1 = iVar1 + -1) {
    *puVar3 = *puVar2;
    puVar2 = puVar2 + 1;
    puVar3 = puVar3 + 1;
  }
  *(undefined2 *)puVar3 = *(undefined2 *)puVar2;
  *(undefined *)((int)puVar3 + 2) = *(undefined *)((int)puVar2 + 2);
  iVar1 = FUN_10001360();
  if (iVar1 != 0) {
    local_498._2_1_ = 0x4c;
    local_498._3_1_ = 0x4d;
  }
  local_464 = DAT_1000dd98;
  puVar2 = &local_463;
  for (iVar1 = 0x18; iVar1 != 0; iVar1 = iVar1 + -1) {
    *puVar2 = 0;
    puVar2 = puVar2 + 1;
  }
  *(undefined2 *)puVar2 = 0;
  *(undefined *)((int)puVar2 + 2) = 0;
  FUN_100014a0(&local_464);
  (*DAT_sprintf)(local_400,s_cmd.exe_/c_reg_add_%s_/v_"%s"_/t_1000d544,&local_
                 );
  create_process(local_400,10000,0);
  return;
```

Fig 36: Shellcode installing persistence

```
if (bVar6) {
  (*DAT_sprintf)(auStack3392,s_%s_co_1000d7e4,s_@WanaDecryptor@.exe_1000d5c4);
  create_process(auStack3392,0,0);
}
DAT_1000dce0 = (*DAT_time)(0);
FUN_10004730();
if (unaff_EDI == 1) {
  (*DAT_sprintf)(auStack3392,s_cmd.exe_/c_start_/b_%s_vs_1000d7c8,
                 s_@WanaDecryptor@.exe_1000d5c4);
  create_process(auStack3392,0,0);
}
```

Fig 37: Shellcode showing the user popup after encryption

WannaCry Crypto Malware
Feb 2023
v1.0

Advanced Dynamic Analysis

Advanced dynamic analysis was used over the course of whole analysis, mostly in analysis of dynamically loaded shellcode to easily get dynamically loaded APIs and to forward that info into Ghidra for further static analysis. It was also used as aid to step over some functions and see what they done as visible by basic dynamic analysis tools. X64dbg project was used for this purpose.

Indicators of Compromise

Network Indicators

- URL killswitch(Fig 8 above): http://www.iuqerfsodp9ifjaposdfjhgosurijfaewrwergwea.com
- SMB port scanning (Fig 11 above)

Host-based Indicators

Unique services:

- "mssecsvc2.0" service
- Randomly named service with "cmd.exe /c "<dir>/tasksche.exe" path

Unique files:

File Name	SHA256 Hash
Ransomware.wannacry.exe	24d004a104d4d54034dbcffc2a4b19a11f39008a575aa614ea04703480b1022c
tasksche.exe	ed01ebfbc9eb5bbea545af4d01bf5f1071661840480439c6e5babe8e080e41aa
b.wnry @WanaDecryptor@.bmp	d5e0e8694ddc0548d8e6b87c83d50f4ab85c1debadb106d6a6a794c3e746f4fa
c.wnry	74a13bd4f2d3819f8b8c3e3321c2473da9414a7a4a732ad8e17a74ab9863898b
f.wnry	667faae492855455ea0d13902b6a597b9ef882d22988d5fa98b9753311834bb9
r.wnry	402751fa49e0cb68fe052cb3db87b05e71c1d950984d339940cf6b29409f2a7c
s.wnry	e18fdd912dfe5b45776e68d578c3af3547886cf1353d7086c8bee037436dff4b
t.wnry	97ebce49b14c46bebc9ec2448d00e1e397123b256e2be9eba5140688e7bc0ae6
u.wnry @WanaDecryptor@.exe	b9c5d4339809e0ad9a00d4d3dd26fdf44a32819a54abf846bb9b560d81391c25
taskdl.exe	4a468603fdcb7a2eb5770705898cf9ef37aade532a7964642ecd705a74794b79
taskse.exe	2ca2d550e603d74dedda03156023135b38da3630cb014e3d00b1263358c5f00d

Unique mutexes:

- Global\MsWinZonesCacheCounterMutexA0

Rules & Signatures

```
rule Yara_WCRY {
    meta:
        last_updated = "2023-02-06"
        author = "thisIsBaggio"
        description = "WannaCry dropper"
        hash = "24d004a104d4d54034dbcffc2a4b19a11f39008a575aa614ea04703480b1022c"

    strings:
        // Fill out identifying strings and other criteria
        $string0 = "mssecsvc2.0" ascii
        $string1 = "WanaCrypt0r" wide
        $string2 = "\\\\172.16.99.5\\IPC$" wide
        $string3 = "\\\\192.168.56.20\\IPC$" wide
        $string4 = "www.iuqerfsodp9ifjaposdfjhgosurijfaewrwergwea.com" ascii
        $string5 = "icacls . /grant Everyone:F /T /C /Q" ascii
        $string6 = "attrib +h ." ascii
        $string7 = "WNcry@2ol7" ascii
        $string8 = ".wnry" ascii
        $string9 = "Global\\MsWinZonesCacheCounterMutexA" ascii
        $PE_magic_byte = "MZ"

    condition:
        // Fill out the conditions that must be met to identify the binary
        $PE_magic_byte at 0
            and $string0 and $string1
            and $string2 and $string3
            and $string4 and $string5
            and $string6 and $string7
            and $string8 and $string9
}
```