

# A technical analysis of the BackMyData ransomware used to attack hospitals in Romania

[cybergeeks.tech/a-technical-analysis-of-the-backmydata-ransomware-used-to-attack-hospitals-in-romania/](https://cybergeeks.tech/a-technical-analysis-of-the-backmydata-ransomware-used-to-attack-hospitals-in-romania/)

## Summary

According to [BleepingComputer](#), a ransomware attack that occurred starting on February 11 forced 100 hospitals across Romania to take their systems offline. BackMyData ransomware, which took credit for it, belongs to the Phobos family. The malware embedded an AES key that is used to decrypt its configuration containing whitelisted extensions, files, and directories, a public RSA key that is used to encrypt AES keys used for files' encryption, and other information. Persistence is achieved by creating an entry under the Run registry key and copying the malware to the Startup folder. The ransomware encrypts the local drives as well as the network shares. It deletes all Volume Shadow Copies and runs commands to disable the firewall.

The files are encrypted using the AES256 algorithm, with the AES key being encrypted using the public RSA key decrypted from the configuration. The malware appends 6 custom bytes at the end of every encrypted file. In the end, the ransomware drops two ransom notes called "info.txt" and "info.hta" that contain information about how to contact the threat actor.

## Technical analysis

SHA256: 396a2f2dd09c936e93d250e8467ac7a9c0a923ea7f9a395e63c375b877a399a6

The ransomware comes with an encrypted configuration that is decrypted using a hard-coded AES key:

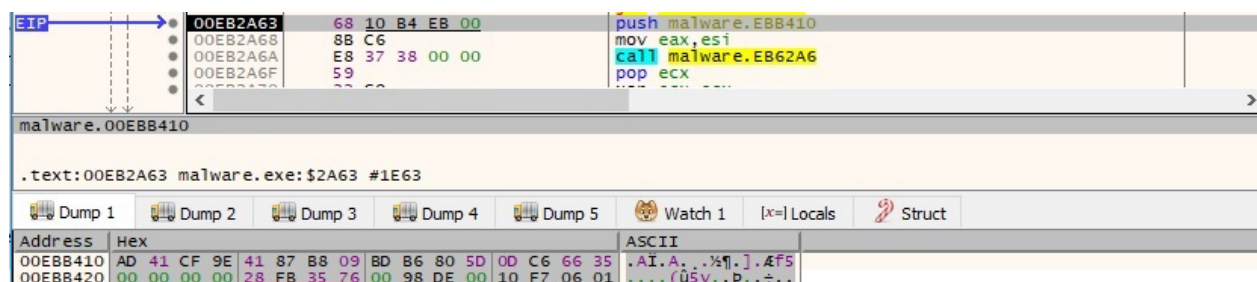


Figure 1

As we can see below, the configuration is stored in an encrypted form:



Figure 2

The malware implements the AES algorithm in its code and doesn't rely on Windows APIs:

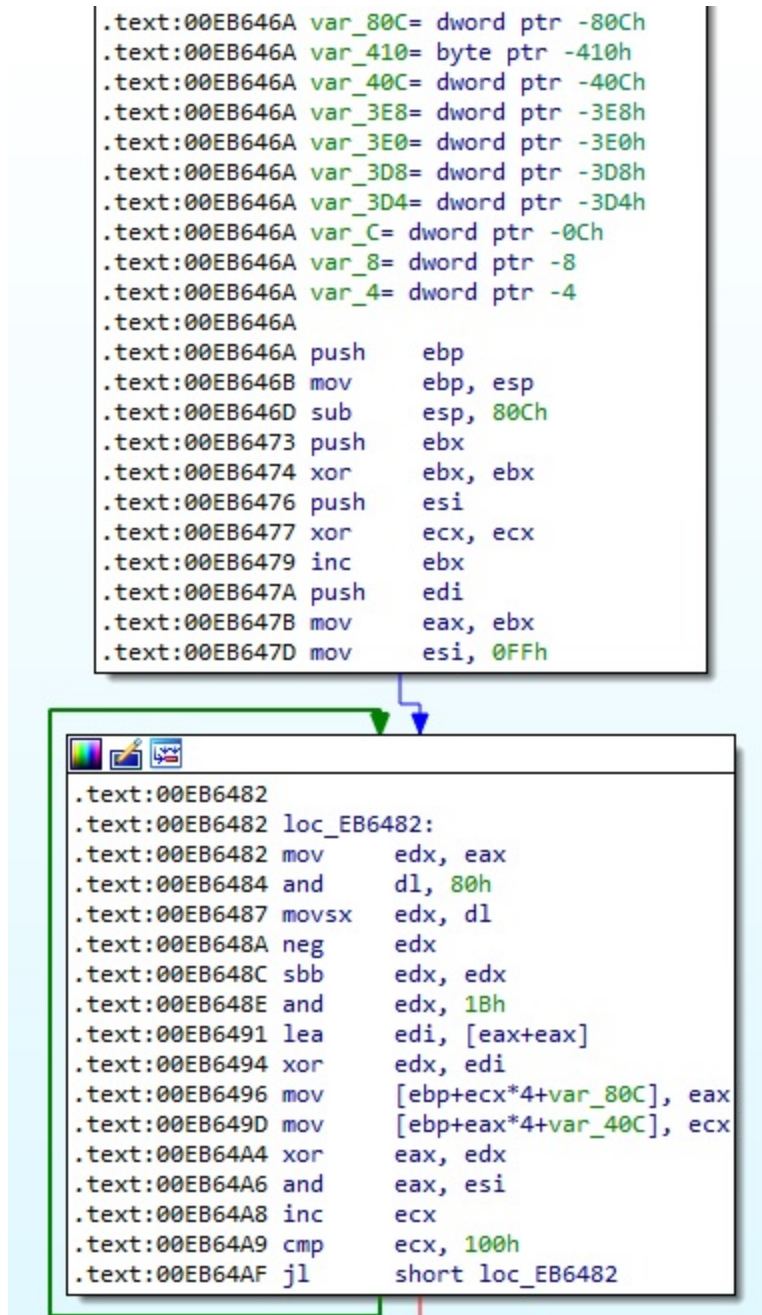


Figure 3

```

EIP → 00EB6715 21 4D FC and dword ptr ss:[ebp-4],ecx
00EB6718 BE FF 00 00 00 mov esi,FF
00EB671D 8B 48 1C mov ecx,dword ptr ds:[eax+1C]
00EB6720 0F B6 50 1F movzx edx,byte ptr ds:[eax+1F]
00EB6724 0F B6 92 48 B4 EB 00 movzx edx,byte ptr ds:[edx+EBB448]
00EB6728 23 CE and ecx,esi
00EB672D 0F B6 89 48 B4 EB 00 movzx ecx,byte ptr ds:[ecx+EBB448]
00EB6734 C1 E1 08 shl ecx,8
00EB6737 33 CA xor ecx,edx
00EB6739 0F B6 50 1E movzx edx,byte ptr ds:[eax+1E]
00EB673D 0F B6 92 48 B4 EB 00 movzx edx,byte ptr ds:[edx+EBB448]
00EB6744 C1 E1 08 shl ecx,8
00EB6747 33 CA xor ecx,edx
00EB6749 0F B6 50 1D movzx edx,byte ptr ds:[eax+1D]
00EB674D 0F B6 92 48 B4 EB 00 movzx edx,byte ptr ds:[edx+EBB448]
00EB6754 C1 E1 08 shl ecx,8
00EB6757 33 CA xor ecx,edx
00EB6759 8B 55 FC mov edx,dword ptr ss:[ebp-4]
00EB675C 33 8A 48 BD EB 00 xor ecx,dword ptr ds:[edx+EBB048]
00EB6762 8B 50 04 mov edx,dword ptr ds:[eax+4]
00EB6765 33 08 xor ecx,dword ptr ds:[eax]
00EB6767 83 45 FC 04 add dword ptr ss:[ebp-4],4
00EB6768 33 D1 xor edx,ecx
00EB676D 89 48 20 mov dword ptr ds:[eax+20],ecx
00EB6770 8B 48 08 mov ecx,dword ptr ds:[eax+8]
00EB6773 33 CA xor ecx,edx
00EB6775 89 48 28 mov dword ptr ds:[eax+28],ecx
00EB6778 89 50 24 mov dword ptr ds:[eax+24],edx
00EB677B 8B 50 0C mov edx,dword ptr ds:[eax+C]
00EB677E 33 D1 xor edx,ecx
00EB6780 89 50 2C mov dword ptr ds:[eax+2C],edx
00EB6783 0F B6 48 2F movzx ecx,byte ptr ds:[eax+2F]
00EB6787 0F B6 89 48 B4 EB 00 movzx ecx,byte ptr ds:[ecx+EBB448]
00EB678E 0F B6 58 2E movzx ebx,byte ptr ds:[eax+2E]
00EB6792 0F B6 9B 48 B4 EB 00 movzx ebx,byte ptr ds:[ebx+EBB448]
00EB6799 C1 E1 08 shl ecx,8
00EB679C 33 CB xor ecx,ebx
00EB679E 0F B6 58 2D movzx ebx,byte ptr ds:[eax+2D]
00EB67A2 0F B6 9B 48 B4 EB 00 movzx ebx,byte ptr ds:[ebx+EBB448]
00EB67A9 C1 E1 08 shl ecx,8
00EB67AC 33 CB xor ecx,ebx
00EB67AE 23 D6 and edx,esi
00EB67B0 0F B6 92 48 B4 EB 00 movzx edx,byte ptr ds:[edx+EBB448]
00EB67B7 C1 E1 08 shl ecx,8
00EB67BA 33 CA xor ecx,edx
00EB67BC 33 48 10 xor ecx,dword ptr ds:[eax+10]
00EB67BF 8B 50 14 mov edx,dword ptr ds:[eax+14]
00EB67C2 33 D1 xor edx,ecx
00EB67C4 89 48 30 mov dword ptr ds:[eax+30],ecx
00EB67C7 8B 48 18 mov ecx,dword ptr ds:[eax+18]

```

dword ptr [ebp-4]=[00D3F8F4]=0  
ecx=0  
.text:00EB6715 malware.exe:\$6715 #5B15

Address	Hex	ASCII
00D3F940	AD 41 CF 9E 41 87 B8 09 BD B6 80 5D 0D C6 66 35	.AI.A. .%].Af5
00D3F950	00 00 00 00 28 FB 35 76 00 98 DE 00 10 F7 06 01	....(05v..P..÷..

Figure 4

The malicious process retrieves the number of milliseconds elapsed since the system was started using GetTickCount:

```

EIP → 00EB2A94 FF 15 84 A0 EB 00 call dword ptr ds:[<&GetTickCount>]

```

dword ptr [00EBA084 <malware.&GetTickCount>]=<kernel32.GetTickCount>

Figure 5

The GetLocaleInfoW function is used to obtain the default locale for the operating system (0x800 = **LOCALE\_SYSTEM\_DEFAULT**, 0x58 = **LOCALE\_FONTSIGNATURE**). The binary verifies whether the 9th bit, which represents Cyrillic alphabets, is cleared. This technique of avoiding systems that have this setting as default was also documented by Malwarebytes in their [article about Phobos ransomware](#).

```

.text:00EB2AA6 push    20h ; ' ' ; cchData
.text:00EB2AA8 lea    eax, [esp+104h+LCDData]
.text:00EB2AAC push    eax ; lpLCDData
.text:00EB2AAD push    58h ; 'X' ; LCTYPE
.text:00EB2AAF push    800h ; Locale
.text:00EB2AB4 call   ds:GetLocaleInfoW
.text:00EB2ABA test   eax, eax
.text:00EB2ABC jz    short loc_EB2ACA

.text:00EB2ABE mov    eax, dword ptr [esp+100h+LCDData]
.text:00EB2AC2 shr    eax, 9
.text:00EB2AC5 and    eax, 1
.text:00EB2AC8 jmp    short loc_EB2ACC

.text:00EB2ACA loc_EB2ACA:
.text:00EB2ACA xor    eax, eax

```

Figure 6

An example of decrypting values from configuration is highlighted below:

```

00EB2ADD 53          push    edx
00EB2AE0 E8 62 38 00 00 call   malware.EB6347

```

Figure 7

```

00EB642B 5F          pop    edi
00EB642C 5E          pop    esi
00EB642D 5B          pop    ebx
00EB642E 8B ES      mov    esp, ebp
00EB6430 5D          pop    ebp
00EB6431 C3          ret

.text:00EB6431 malware.exe:$6431 #5831

```

Figure 8

The binary retrieves the path of the executable file of the current process via a function call to GetModuleFileNameW (see Figure 9).

```

00EB5B0E 68 04 01 00 00 push    104
00EB5B13 FF 74 24 08      push    dword ptr ss:[esp+8]
00EB5B19 FF 15 FC A0 EB 00 call   dword ptr ds:[<&GetModuleFileNameW>]

```

Figure 9

Interestingly, the process is looking for a file called “backm” that wasn’t previously created by the ransomware (0x80000000 = **GENERIC\_READ**, 0x3 = **OPEN\_EXISTING**):

```

00EB278F 57          push    edi
00EB2790 57          push    edi
00EB2791 6A 03      push    3
00EB2793 57          push    edi
00EB2794 57          push    edi
00EB2795 68 00 00 00 80 push    80000000
00EB279A 56          push    esi
00EB279B FF 15 98 A0 EB 00 call   dword ptr ds:[<&CreateFileW>]

```

Figure 10

The malware extracts the major and minor version numbers of the operating system using the GetVersion method:

Figure 11

It opens the access token associated with the current process by calling the OpenProcessToken API (0x8 = **TOKEN\_QUERY**):

Figure 12

The malicious process verifies if the token is elevated using GetTokenInformation (0x14 = **TokenElevation**):

Figure 13

The environment variable “%systemdrive%” is expanded, which reveals the drive that contains the Windows directory:

Figure 14

GetVolumeInformationW is used to obtain the volume serial number:

Figure 15

The ransomware tries to open two mutexes called “Global\\<<BID>><Volume serial number>00000001” and “Global\\<<BID>><Volume serial number>00000000”, and then creates them:

Figure 16

Figure 17

The DLLs and functions necessary to perform some activities are also decrypted from the configuration. The binary obtains the module handle for a DLL using GetModuleHandleA:

Figure 18

The address of the exported functions is retrieved by calling the GetProcAddress API:

Figure 19

The malware disables file system redirection for the calling thread:

Figure 20

It obtains a handle to the Shell's desktop window via a function call to GetShellWindow (Figure 21).

Figure 21

Using the above handle, the process calls the GetWindowThreadProcessId function to retrieve the identifier of the process that created the window (explorer.exe):

Figure 22

The binary opens the “explorer.exe” process using the OpenProcess method (0x400 = **PROCESS\_QUERY\_INFORMATION**):

Figure 23

OpenProcessToken is used to open the access token associated with the above process (0x02000000 = **MAXIMUM\_ALLOWED**):

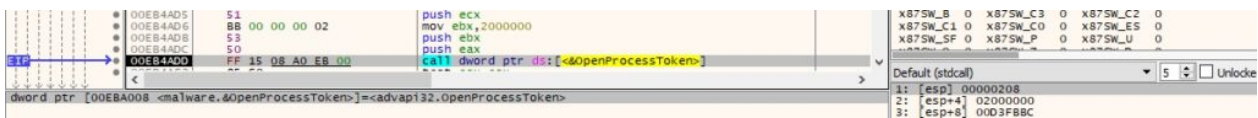


Figure 24

The DuplicateTokenEx API is utilized to create a new access token that duplicates the token mentioned above (0x2 = **SecurityImpersonation**, 0x1 = **TokenPrimary**):



Figure 25

The ransomware spawns itself running in the security context of the newly created token (Figure 26).



Figure 26

It creates a new thread that will run the following commands in the sub\_EB4B85 function:

- vssadmin delete shadows /all /quiet – delete all Volume Shadow Copies
- wmic shadowcopy delete – delete all Volume Shadow Copies
- bcdedit /set {default} bootstatuspolicy ignoreallfailures – ignore errors if there is a failed boot, shutdown, or checkpoint
- bcdedit /set {default} recoveryenabled no – disable automatic repair
- wbadmin delete catalog -quiet – delete the backup catalog on the machine
- netsh advfirewall set currentprofile state off – disable the firewall for the current network profile
- netsh firewall set opmode mode=disable – disable the firewall

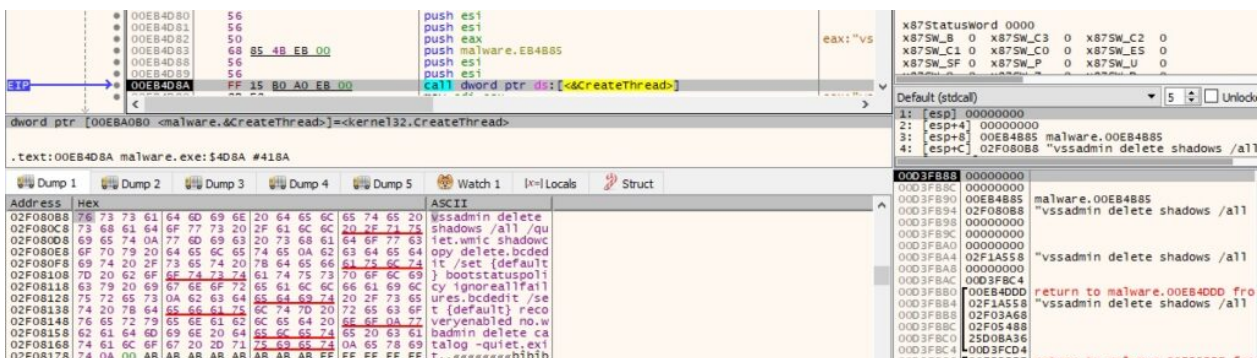


Figure 27

The process copies its executable to the “%AppData%\Local” directory, as highlighted in Figure 28.



Figure 28

RegOpenKeyExW is used to open the Run registry key (0x80000002 = HKEY\_LOCAL\_MACHINE, 0x20106 = KEY\_WRITE | KEY\_WOW64\_64KEY):

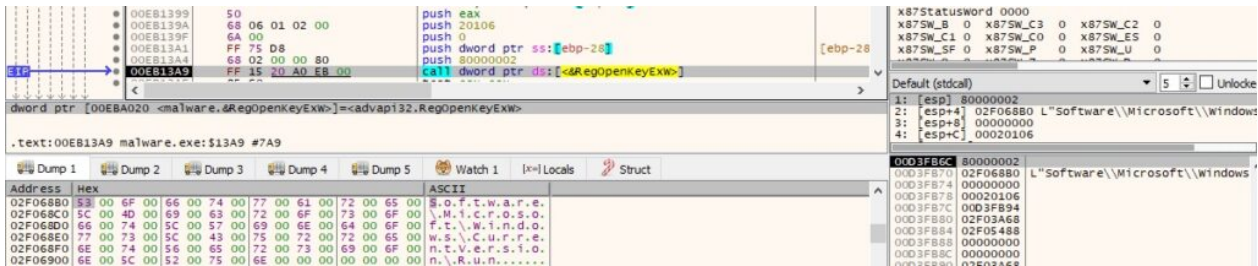


Figure 29

The ransomware establishes persistence by creating an entry named based on the executable name, which points to the newly created executable:

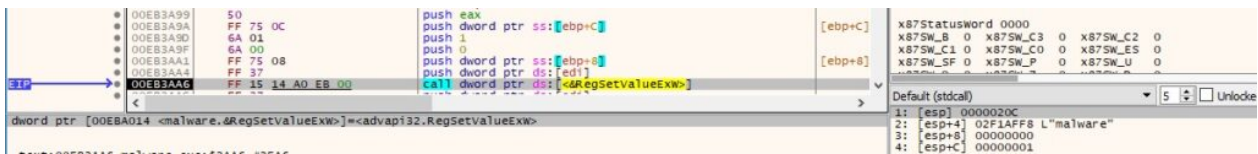


Figure 30

The malicious binary tries to copy the non-existent file called “backm” to the same directory:



Figure 31

The second persistence mechanism consists of copying the executable to the Startup folder.

The following extensions are targeted, but the ransomware will encrypt other extensions as well:

fdb sql 4dd 4dl abs abx accdb accdc accde adb adf ckp db db-journal db-shm db-wal db2 db3 dbc dbf dbs dbt dbv dcb dp1 eco edb epim fcd gdb mdb mdf ldf myd ndf nwdb nyf sqlitedb sqlite3 sqlite

Also, the ransomware doesn't encrypt files that were previously encrypted by other ransomware families:



backmydata actin DIKE Acton actor Acuff FILE Acuna fullz MMXXII GrafGrafel monero n3on jopanaxy 2700 DEVOS kmrox s0m1n qos cg ext rdptest S0va 6y8dghklp SHTORM NURRI GHOST FF6OM6 blue NX BACKJOHN OWN FS23 2QZ3 top blackrock CHCRBO G-STARs faust unknown STEEL worry WIN duck fopra unique acute adage make Adair MLF magic Adame banhu banjo Banks Banta Barak Caleb Cales Caley calix Calle Calum Calvo deuce Dever devil Devoe Devon Devos dewar eight eject eking Elbie elbow elder phobos help blend bqux com mamba KARLOS DDoS phoenix PLUT karma bbc CAPITAL WALLET LKS tech s1g2n3a4l MURK makop ebaka jook LOGAN FIASKO GUCCI decrypt OOH Non grt LIZARD FLSCRYPT SDK 2023 vhdv

The following files and directories will also be skipped during the encryption process:

- info.hta info.txt boot.ini bootfont.bin ntldr ntdetect.com io.sys backm
- C:\WINDOWS C:\ProgramData\microsoft\windows\caches

The process splits further malicious activity into multiple threads that will be described in the following paragraphs. The following functions will be executed: sub\_EB22EE, sub\_EB239A, sub\_EB2161, sub\_EB1A76, and sub\_EB1CC5.

Figure 32

### Thread activity – sub\_EB22EE function

The malware opens the access token associated with the current process (0x20 = **TOKEN\_ADJUST\_PRIVILEGES**):

Figure 33

The LookupPrivilegeValueW method is used to extract the locally unique identifier (LUID) that represents the “SeDebugPrivilege” privilege:

Figure 34

The malicious process enables the above privilege via a call to AdjustTokenPrivileges:

```

00EB3DFA 56          push esi
00EB3DFC 89 45 E4    mov dword ptr ss:[ebp-1C],eax
00EB3DFF 8B 45 F4    mov eax,dword ptr ss:[ebp-C]
00EB3E02 56          push esi
00EB3E03 56          push esi
00EB3E04 89 45 E8    mov dword ptr ss:[ebp-18],eax
00EB3E07 8D 45 E0    lea eax,dword ptr ss:[ebp-20]
00EB3E0A 50          push eax
00EB3E0B 56          push esi
00EB3E0C FF 75 F8    push dword ptr ss:[ebp-8]
00EB3E0F C7 45 E0 01 00 00 00 mov dword ptr ss:[ebp-20],1
00EB3E14 C7 45 EC 02 00 00 00 mov dword ptr ss:[ebp-14],2
00EB3E1D FF 15 1C AD EB 00 call dword ptr ds:[&AdjustTokenPrivileges]
    
```

Figure 35

The following processes will be killed because they could lock files to be encrypted:

- msftesql.exe sqlagent.exe sqlbrowser.exe sqlservr.exe sqlwriter.exe oracle.exe
- ocssd.exe dbsnmp.exe synctime.exe agntsvc.exe mydesktopqos.exe isqlplussvc.exe
- xfssvcon.exe mydesktopservice.exe ocautoupds.exe agntsvc.exe agntsvc.exe
- agntsvc.exe encsvc.exe firefoxconfig.exe tbirdconfig.exe ocomm.exe mysqld.exe
- mysqld-nt.exe mysqld-opt.exe dbeng50.exe sqbcoreservice.exe excel.exe infopath.exe
- msaccess.exe mspub.exe onenote.exe outlook.exe powerpnt.exe steam.exe
- thebat.exe thebat64.exe thunderbird.exe visio.exe winword.exe wordpad.exe

The malware takes a snapshot of all processes in the system, as displayed in the figure below.

```

00EB4DFA 57          push edi
00EB4DFB 6A 02      push 2
00EB4DFD 89 7D FC    mov dword ptr ss:[ebp-4],edi
00EB4E00 FF 15 24 A1 EB 00 call dword ptr ds:[&CreateToolhelp32Snapshot]
    
```

Figure 36

The processes are enumerated using the Process32FirstW and Process32NextW APIs:

```

00EB4E2E 50          push eax
00EB4E2F push dword ptr ss:[ebp-8]
00EB4E33 89 85 CC FD FF FF mov dword ptr ss:[ebp-23],esi
00EB4E38 FF 15 10 A1 EB 00 call dword ptr ds:[&Process32FirstW]
    
```

Figure 37

```

00EB4E8A 50          push eax
00EB4E8B push dword ptr ss:[ebp-8]
00EB4E8E FF 15 1C A1 EB 00 call dword ptr ds:[&Process32NextW]
    
```

Figure 38

Any target process is stopped using the TerminateProcess method:

```

.text:00EB4E5D push [ebp+pe.th32ProcessID] ; dwProcessId
.text:00EB4E63 xor ebx, ebx
.text:00EB4E65 push ebx ; bInheritHandle
.text:00EB4E66 push 1 ; dwDesiredAccess
.text:00EB4E68 call ds:OpenProcess
.text:00EB4E6E mov esi, eax
.text:00EB4E70 test esi, esi
.text:00EB4E72 jz short loc_EB4E81

.text:00EB4E74 push ebx ; uExitCode
.text:00EB4E75 push esi ; hProcess
.text:00EB4E76 call ds:TerminateProcess
.text:00EB4E7C push esi ; hObject
.text:00EB4E7D mov ebx, eax
.text:00EB4E7F call edi ; CloseHandle

```

Figure 39

### Thread activity – sub\_EB239A function

OpenProcessToken is utilized to open the access token associated with the process (0x8 = **TOKEN\_QUERY**):

Figure 40

The binary verifies again if the token is elevated by calling the GetTokenInformation API (0x14 = **TokenElevation**):

Figure 41

It calls again the OpenMutexW and CreateMutexW methods with the “Global\\<<BID>> <Volume serial number>00000000” mutex name:

Figure 42

### Thread activity – sub\_EB2161 function

The ransomware uses events to synchronize threads. It creates two unnamed event objects using CreateEventW:

Figure 43

The NetBIOS name of the local machine is extracted (Figure 44).

Figure 44

WNetOpenEnumW is used to start an enumeration of all currently connected resources (0x1 = RESOURCE\_CONNECTED):

Figure 45

The enumeration continues by calling the WNetEnumResourceW function:

Figure 46

The process obtains the interface-to-IPv4 address mapping table via a function call to GetIpAddrTable, as shown below:

Figure 47

Every IP address extracted above is converted from network order to host byte order using ntohs:

Figure 48

The malware creates a TCP socket (0x2 = AF\_INET, 0x1 = SOCK\_STREAM, 0x6 = IPPROTO\_TCP):

Figure 49

It tries to connect to every host on the network on port 445 in order to encrypt every available network share:



Figure 50

### Thread activity – sub\_EB4B85 function

The process creates two anonymous pipes by calling the CreatePipe method (see Figure 51).

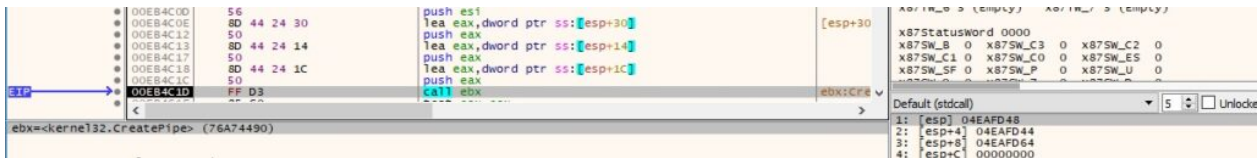


Figure 51

The read handles are made inheritable using SetHandleInformation (0x1 = HANDLE\_FLAG\_INHERIT):

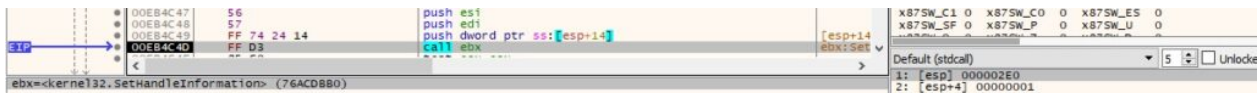


Figure 52

The ransomware creates a “cmd.exe” process that will execute multiple commands:



Figure 53

The commands responsible for disabling the firewall, deleting all Volume Shadow Copies, and so on, are transmitted to the newly created process via pipes:

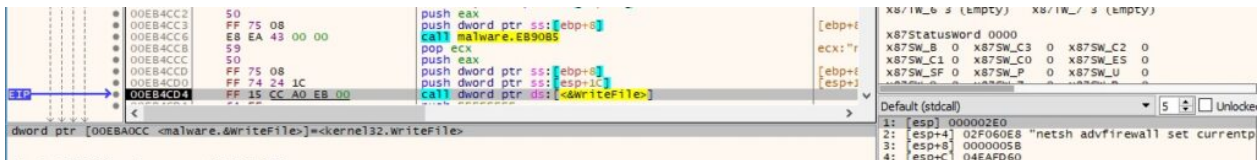


Figure 54

### Thread activity – sub\_EB1CC5 function

This thread keeps extracting a bitmask representing the currently available disk drives using the GetLogicalDrives API:

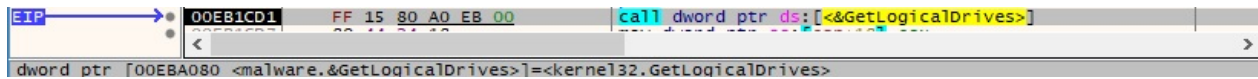


Figure 55

### Thread activity – sub\_EB1A76 function

The malware decrypts the public RSA key that will be used to encrypt the AES256 key used for file’s encryption. The same key was used by Phobos ransomware since 2019 according to [Talos](#).

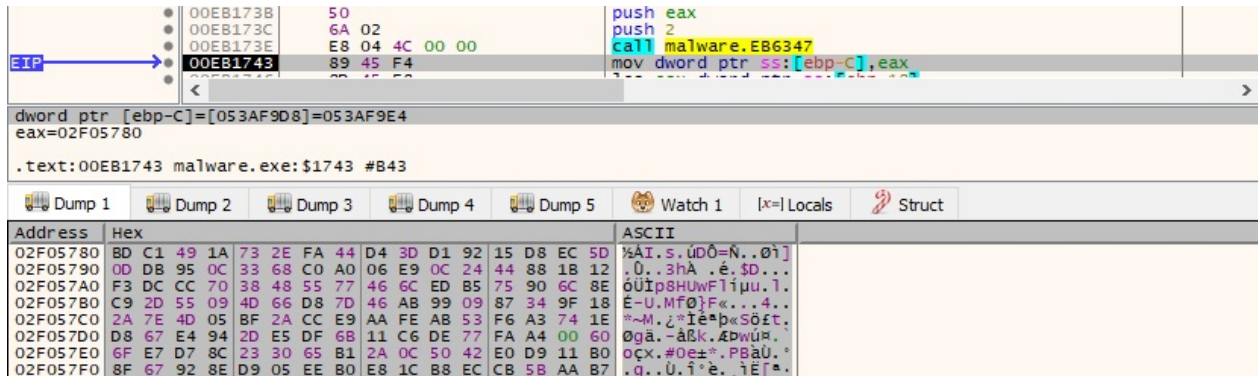


Figure 56

It extracts the current local date and time, the current process and thread IDs, and other information using multiple functions:

```

.text:00EB77FF mov     edi, ds:QueryPerformanceCounter
.text:00EB7805 lea     eax, [ebp+PerformanceCount]
.text:00EB7808 push    eax                ; lpPerformanceCount
.text:00EB7809 call   edi ; QueryPerformanceCounter
.text:00EB780B call   ds:GetTickCount
.text:00EB7811 xor     dword_EBD680, eax
.text:00EB7817 mov     eax, dword ptr [ebp+PerformanceCount+4]
.text:00EB781A xor     dword_EBD684, eax
.text:00EB7820 mov     eax, dword ptr [ebp+PerformanceCount]
.text:00EB7823 xor     dword_EBD688, eax
.text:00EB7829 call   ds:GetCurrentProcessId
.text:00EB782F xor     dword_EBD68C, eax
.text:00EB7835 call   ds:GetCurrentThreadId
.text:00EB783B xor     dword_EBD690, eax
.text:00EB7841 lea     eax, [ebp+SystemTime]
.text:00EB7844 push    eax                ; lpSystemTime
.text:00EB7845 call   ds:GetLocalTime
.text:00EB784B lea     eax, [ebp+FileTime]
.text:00EB784E push    eax                ; lpFileTime
.text:00EB784F lea     eax, [ebp+SystemTime]
.text:00EB7852 push    eax                ; lpSystemTime
.text:00EB7853 call   ds:SystemTimeToFileTime
.text:00EB7859 mov     eax, [ebp+FileTime.dwHighDateTime]
.text:00EB785C xor     dword_EBD694, eax
.text:00EB7862 mov     eax, [ebp+FileTime.dwLowDateTime]
.text:00EB7865 xor     dword_EBD698, eax
.text:00EB786B lea     eax, [ebp+PerformanceCount]
.text:00EB786E push    eax                ; lpPerformanceCount
.text:00EB786F call   edi ; QueryPerformanceCounter
.text:00EB7871 mov     eax, dword ptr [ebp+PerformanceCount+4]
.text:00EB7874 xor     dword_EBD69C, eax
.text:00EB787A mov     eax, dword ptr [ebp+PerformanceCount]
.text:00EB787D xor     dword_EBD680, eax

```

Figure 57

The binary creates a new thread that will traverse the network shares and drives in order to extract files to be encrypted:

```

00EB58E8 53          push ebx
00EB58E9 53          push ebx
00EB58EA 56          push esi
00EB58EB 68 B3 56 EB 00  push malware.EB56B3
00EB58F0 53          push ebx
00EB58F1 53          push ebx
00EB58F2 FF 15 B0 A0 EB 00  call dword ptr ds:[<CreateThread>]

```

Stack (stdcall):

- 1: [esp] 00000000
- 2: [esp+4] 00000000
- 3: [esp+8] 00EB56B3 malware.00EB56B3
- 4: [esp+C] 02F06B58

Figure 58

### Thread activity – sub\_EB56B3 function

Two new threads, which will be responsible for file’s encryption, are created:

```

00EB5737 50          push eax
00EB5738 50          push eax
00EB5739 8D 4C 24 1C    tea ecx,dword ptr ss:[esp+1C]
00EB573D 51          push ecx
00EB573E 68 BF 54 EB 00  push malware.EB54BF
00EB5743 6A 01        push 1
00EB5745 50          push eax
00EB5746 FF 15 B0 A0 EB 00  call dword ptr ds:[<CreateThread>]

```

Stack (stdcall):

- 1: [esp] 00000000
- 2: [esp+4] 00000001
- 3: [esp+8] 00EB54BF malware.00EB54BF
- 4: [esp+C] 0576FB5C

Figure 59

The files are enumerated using the FindFirstFileW and FindNextFileW methods:



Figure 60



Figure 61

The malware sets the event objects to the signaled state via a function call to SetEvent:

```

.text:00EB33C5
.text:00EB33C5
.text:00EB33C5
.text:00EB33C5 sub_EB33C5 proc near
.text:00EB33C5 push ebx
.text:00EB33C6 push edi
.text:00EB33C7 lea ebx, [esi+0Ch]
.text:00EB33CA push ebx ; lpCriticalSection
.text:00EB33CB call ds:EnterCriticalSection
.text:00EB33D1 push dword ptr [esi+4] ; hEvent
.text:00EB33D4 mov edi, ds:SetEvent
.text:00EB33DA or dword ptr [esi+28h], 1
.text:00EB33DE call edi ; SetEvent
.text:00EB33E0 push dword ptr [esi] ; hEvent
.text:00EB33E2 call edi ; SetEvent
.text:00EB33E4 push dword ptr [esi+8] ; hEvent
.text:00EB33E7 call edi ; SetEvent

```

Figure 62

The process waits until the new threads finish their execution using the WaitForMultipleObjects function.

### Thread activity – sub\_EB54BF function

The ransomware opens a file to be encrypted in reading mode (0x80000000 = **GENERIC\_READ**, 0x7 = **FILE\_SHARE\_DELETE** | **FILE\_SHARE\_WRITE** | **FILE\_SHARE\_READ**, 0x3 = **OPEN\_EXISTING**):



Figure 63

It retrieves the size of the file using the GetFileSizeEx API:

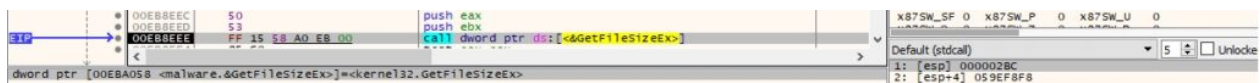


Figure 64



The size is compared with 0x180000 bytes (1.5MB), and the files having more bytes are partially encrypted. The rest of the files are totally encrypted:

```

.text:00EB8F4A cmp     [ebp+var_18], 180000h
.text:00EB8F51 jb     short loc_EB8F67

.text:00EB8F53
.text:00EB8F53 loc_EB8F53:          ; char
.text:00EB8F53 push   dword ptr [ebp+arg_C]
.text:00EB8F56 mov    eax, [ebp+arg_0]
.text:00EB8F59 push  [ebp+lpFileName] ; lpNewFileName
.text:00EB8F5C push  edi               ; lpExistingFileName
.text:00EB8F5D push  [ebp+arg_4]       ; int
.text:00EB8F60 call  sub_EB8C42
.text:00EB8F65 jmp   short loc_EB8F79

.text:00EB8F67
.text:00EB8F67 loc_EB8F67:          ; char
.text:00EB8F67 push   dword ptr [ebp+arg_C]
.text:00EB8F6A mov    eax, [ebp+arg_0]
.text:00EB8F6D push  [ebp+lpFileName] ; LPCWSTR
.text:00EB8F70 push  edi               ; lpFileName
.text:00EB8F71 push  [ebp+arg_4]       ; int
.text:00EB8F74 call  sub_EB8782
  
```

Figure 65

As we can see below, not only the “.backmydata” extension will be added to an encrypted file, but also the volume serial number and the threat actor’s email address:

```

00EB889A 6A 00          push 0
00EB889C FF 75 E4      push dword ptr ss:[ebp-1C]
00EB889F 6A 01          push 1
00EB88A1 6A 00          push 0
00EB88A3 6A 01          push 1
00EB88A5 68 00 00 40   push 40000000
00EB88AA FF 75 10      push dword ptr ss:[ebp+10]
00EB88AD FF 15 98 A0 E8 00 call dword ptr ds:[<<CreateFile>]
  
```

Hex Dump (Address 02F99918):

Address	Hex	ASCII
02F99918	5C 00 5C 00 3F 00 5C 00 43 00 3A 00 5C 00 45 00	.\.?.\.\.C.:.\.E.
02F99928	46 00 53 00 20 00 53 00 6F 00 66 00 74 00 77 00	F.S. .S.o.f.t.w.
02F99938	61 00 72 00 65 00 5C 00 45 00 61 00 73 00 79 00	a.r.e.\.E.a.s.y.
02F99948	20 00 43 00 68 00 61 00 74 00 20 00 53 00 65 00	.C.h.a.t. .S.e.
02F99958	72 00 76 00 65 00 72 00 5C 00 72 00 79 00 2E 00	r.v.e.r.\.r.y..
02F99968	67 00 64 00 62 00 2E 00 69 00 64 00 58 00 41 00	g.d.b...i.d.].A.
02F99978	32 00 43 00 39 00 41 00 44 00 32 00 46 00 2D 00	2.C.9.A.D.2.F..
02F99988	33 00 35 00 31 00 31 00 5D 00 2E 00 58 00 62 00	3.S.1.1.]...[.b.
02F99998	61 00 63 00 6B 00 6D 00 79 00 64 00 61 00 74 00	a.c.k.m.y.d.a.t.
02F999A8	61 00	a
02F999B8	63 00	c.o.m.]...b.a.c.
02F999C8	68 00 6D 00 79 00 64 00 61 00 74 00 61 00 00 00	k.m.y.d.a.t.a...

Figure 66

The file’s content is read using the ReadFile method (see Figure 67).

```

00EB8957 6A 00          push 0
00EB8959 8D 45 F4      lea eax, dword ptr ss:[ebp-C]
00EB895C 50          push eax
00EB895D FF 75 F0      push dword ptr ss:[ebp-10]
00EB8960 FF 76 20      push dword ptr ds:[esp+20]
00EB8963 FF 75 F8      push dword ptr ss:[ebp-8]
00EB8966 FF 15 94 A0 E8 00 call dword ptr ds:[<<ReadFile>]
  
```

Figure 67

There is a custom implementation of the AES256 algorithm, as highlighted in the figure below.

```

.text:00EB6ABB
.text:00EB6ABB loc_EB6ABB:
.text:00EB6ABB shr     edi, 8
.text:00EB6ABE and     edi, eax
.text:00EB6AC0 shr     edx, 10h
.text:00EB6AC3 shr     ebx, 8
.text:00EB6AC6 and     ebx, eax
.text:00EB6AC8 and     edx, eax
.text:00EB6ACA mov     edx, dword_EBC970[edx*4]
.text:00EB6AD1 xor     edx, dword_EBCD70[edi*4]
.text:00EB6AD8 mov     edi, [ebp+var_C]
.text:00EB6ADB shr     edi, 18h
.text:00EB6ADE xor     edx, dword_EBC570[edi*4]
.text:00EB6AE5 mov     edi, [ebp+var_4]
.text:00EB6AE8 and     edi, eax
.text:00EB6AEA xor     edx, dword_EBD170[edi*4]
.text:00EB6AF1 mov     edi, [ebp+var_C]
.text:00EB6AF4 shr     edi, 10h
.text:00EB6AF7 and     edi, eax
.text:00EB6AF9 mov     edi, dword_EBC970[edi*4]
.text:00EB6B00 xor     edi, dword_EBCD70[ebx*4]
.text:00EB6B07 mov     ebx, [ebp+var_4]
.text:00EB6B0A shr     ebx, 18h
.text:00EB6B0D xor     edi, dword_EBC570[ebx*4]
.text:00EB6B14 mov     ebx, [ebp+var_8]
.text:00EB6B17 and     ebx, eax
.text:00EB6B19 xor     edi, dword_EBD170[ebx*4]
.text:00EB6B20 mov     ebx, [ebp+var_4]
.text:00EB6B23 xor     edi, [ecx+4]
.text:00EB6B26 shr     ebx, 10h
.text:00EB6B29 mov     [ebp+arg_0], edi
.text:00EB6B2C mov     edi, [ebp+var_C]
.text:00EB6B2F shr     edi, 8
.text:00EB6B32 and     edi, eax
.text:00EB6B34 mov     edi, dword_EBCD70[edi*4]
.text:00EB6B3B and     ebx, eax
.text:00EB6B3D xor     edi, dword_EBC970[ebx*4]
.text:00EB6B44 mov     ebx, [ebp+var_8]
.text:00EB6B47 shr     ebx, 18h
.text:00EB6B4A xor     edi, dword_EBC570[ebx*4]
.text:00EB6B51 mov     ebx, [ebp+var_14]
.text:00EB6B54 and     ebx, eax

```

```

.text:00EB6CA7 shr     edx, 10h
.text:00EB6CAA and     edx, eax
.text:00EB6CAC shr     edi, 8
.text:00EB6CAF and     edi, eax
.text:00EB6CB1 shr     ebx, 8
.text:00EB6CB4 mov     [ebp+arg_0], edi
.text:00EB6CB7 mov     edi, dword_EBC970[edx*4]
.text:00EB6CBE mov     edx, [ebp+arg_0]
.text:00EB6CC1 xor     edi, dword_EBCD70[edx*4]
.text:00EB6CC8 mov     edx, [ebp+var_C]
.text:00EB6CCB shr     edx, 18h
.text:00EB6CCE xor     edi, dword_EBC570[edx*4]
.text:00EB6CD5 mov     edi, [ebp+var_4]
.text:00EB6CD8 and     edx, eax
.text:00EB6CDA xor     edi, dword_EBD170[edx*4]
.text:00EB6CE1 mov     edx, [ebp+var_C]
.text:00EB6CE4 shr     edx, 10h
.text:00EB6CE7 and     edx, eax
.text:00EB6CE9 mov     edx, dword_EBC970[edx*4]
.text:00EB6CF0 and     ebx, eax
.text:00EB6CF2 xor     edx, dword_EBCD70[ebx*4]
.text:00EB6CF9 mov     ebx, [ebp+var_4]
.text:00EB6CFC shr     ebx, 18h
.text:00EB6CFF xor     edx, dword_EBC570[ebx*4]
.text:00EB6D06 mov     ebx, [ebp+var_8]
.text:00EB6D09 and     ebx, eax
.text:00EB6D0B xor     edx, dword_EBD170[ebx*4]
.text:00EB6D12 mov     ebx, [ebp+var_4]
.text:00EB6D15 xor     edx, [ecx+4]
.text:00EB6D18 shr     ebx, 10h
.text:00EB6D1B mov     [ebp+arg_0], edx
.text:00EB6D1E mov     edx, [ebp+var_C]
.text:00EB6D21 shr     edx, 8
.text:00EB6D24 and     edx, eax
.text:00EB6D26 mov     edx, dword_EBCD70[edx*4]
.text:00EB6D2D and     ebx, eax
.text:00EB6D2F xor     edx, dword_EBC970[ebx*4]
.text:00EB6D36 mov     ebx, [ebp+var_8]
.text:00EB6D39 shr     ebx, 18h
.text:00EB6D3C xor     edx, dword_EBC570[ebx*4]
.text:00EB6D43 mov     ebx, [ebp+var_14]
.text:00EB6D46 and     ebx, eax

```

Figure 68

The content is encrypted using the AES256 algorithm and written to the newly created file:

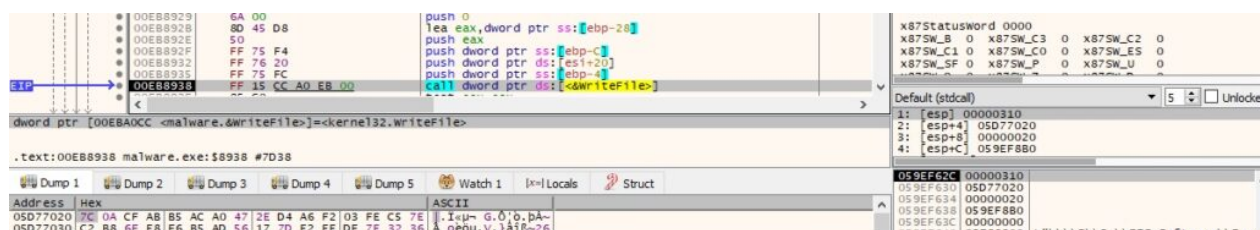


Figure 69

The file's name is encrypted as well and will appear in the encrypted file:

Address	Hex	ASCII
05D77020	00 00 00 00 02 00 00 00 12 5E A7 F0 03 FE C5 7E	.....^şđ.bA~
05D77030	C2 B8 6F E8 F6 B5 AD 56 20 00 00 00 DF 7E 32 36	À.oëöµ.v...B~26
05D77040	72 00 79 00 2E 00 67 00 64 00 62 00 00 00 00 00	r.y...g.d.b.....
05D77050	00 00 00 00 00 00 00 00 00 00 00 00 00 00 00 00	.....
05D77060	00 00 00 00 69 82 53 58 32 36 81 A3 D0 4F AD 35	...i.SX26.£D0.5
05D77070	CB F8 62 ED 08 00 00 00 4F B9 19 7E F4 B1 8B FE	Èöbì...0'.~ô±.p
05D77080	20 78 E8 09 2D 54 DC F4 F0 2E 8A C0 C8 82 17 6B	xè.-TÙöð..AÈ..k
05D77090	B9 7A 98 FE 85 85 D1 4D 6D 88 D9 D2 B5 1A 7E E3	'z.p..Nmm.Ùöµ.~ã
05D770A0	29 EE A2 C4 ED 0C 83 E8 ED 27 44 3C B4 B9 A4 47	)içAi..èi'D<'rçg
05D770B0	7E AC 18 15 1A 2B E9 44 8F 0D 9C DD C4 2C 92 EC	~...+éd...YA..i
05D770C0	85 CD F9 AA 13 0F 36 FC D6 7D 2C 3C 42 6E 35 4C	.Iü*.6üö}.<Bn5L
05D770D0	65 16 BF 96 0F C4 5A 98 15 AE 1E 78 EC 19 CE E4	e.¿.AZ..ø.xi.Iä
05D770E0	F5 F8 85 68 49 EE 98 EF CA 4F 63 88 BD AF EE 12	öø.hIî.iëÖc.½-i.
05D770F0	DA 8B 21 FB 68 C2 8C 3F E2 00 00 00 DD F9 CC F5	Ü.!!üKÄ.?ä...Yüiö
05D77100	B3 44 00 00 00 00 00 00 00 00 00 00 00 00 00 00	*D.....

Figure 70

The following information is also written to the encrypted file: unencrypted 16-byte IV, RSA-encrypted AES256 key, and 6 bytes decrypted from the config that identifies the ransomware “DD F9 CC F5 B3 44”:

Figure 71

The unencrypted file is overwritten with zeros and deleted afterwards:

Figure 72

The structure of an encrypted file is displayed below. We’ve already described the meaning of the buffers.

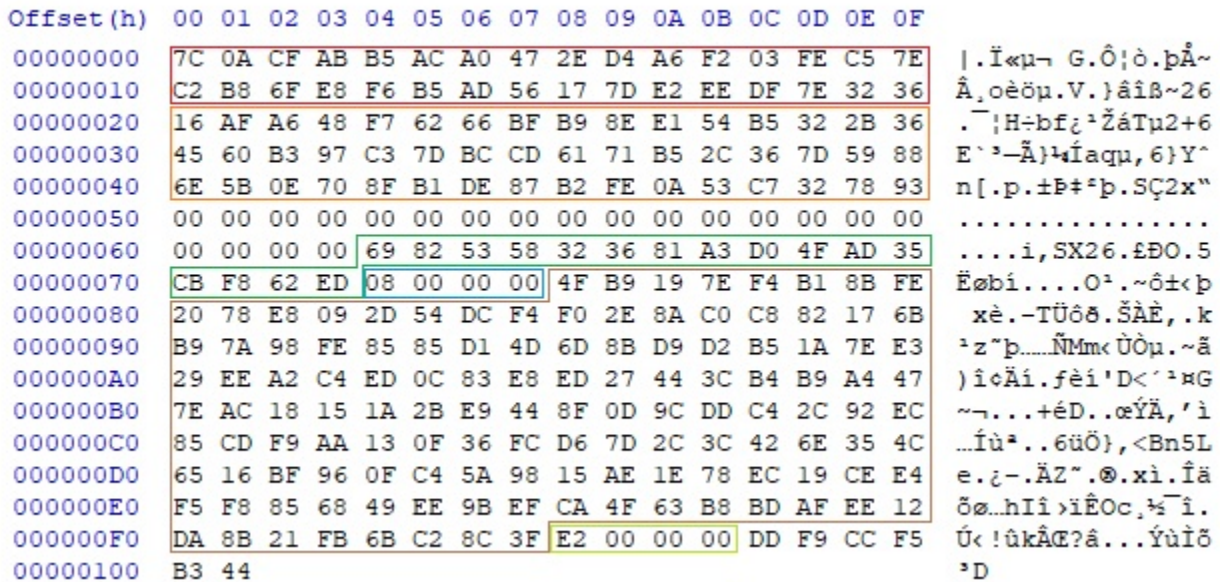


Figure 73

The ransomware drops two ransom notes: "info.txt" and "info.hta". The communication with the threat actor can be done via email or Session messenger.

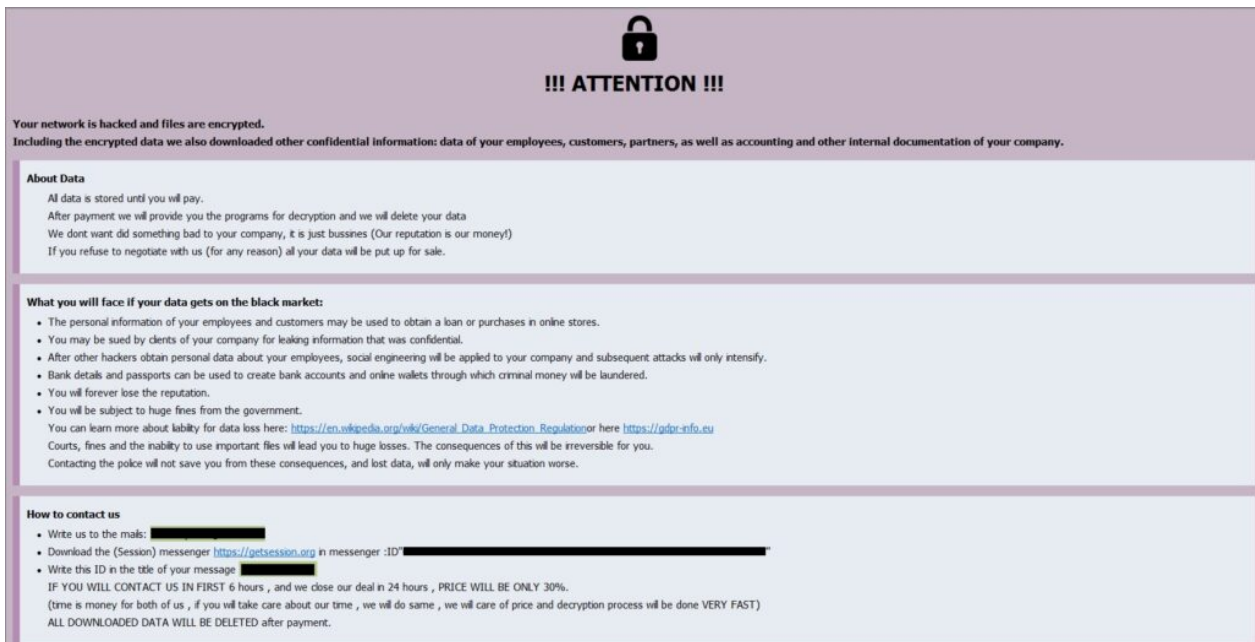


Figure 74

## INDICATORS OF COMPROMISE

### SHA256

396a2f2dd09c936e93d250e8467ac7a9c0a923ea7f9a395e63c375b877a399a6

### BackMyData ransom notes

info.txt, info.hta

### **Files created**

%AppData%\Local\

C:\ProgramData\Microsoft\Windows\Start Menu\Programs\Startup\

### **Registry values**

HKEY\_LOCAL\_MACHINE\Software\Microsoft\Windows\CurrentVersion\Run\

HKEY\_CURRENT\_USER\Software\Microsoft\Windows\CurrentVersion\Run\

### **Processes spawned**

vssadmin delete shadows /all /quiet

wmic shadowcopy delete

bcdedit /set {default} bootstatuspolicy ignoreallfailures

bcdedit /set {default} recoveryenabled no

wbadmin delete catalog -quiet

netsh advfirewall set currentprofile state off

netsh firewall set opmode mode=disable

### **Mutexes**

Global\\<<BID>><Volume serial number>00000000

Global\\<<BID>><Volume serial number>00000001

### **References**

<https://www.bleepingcomputer.com/news/security/ransomware-attack-forces-100-romanian-hospitals-to-go-offline/>

<https://www.malwarebytes.com/blog/news/2019/07/a-deep-dive-into-phobos-ransomware>

<https://blog.talosintelligence.com/deep-dive-into-phobos-ransomware/>

<https://docs.microsoft.com/en-us/windows/win32/api/>