Talos Vulnerability Report

TALOS-2022-1648

Callback technologies CBFS Filter handle_ioctl_8314C null pointer dereference vulnerability

NOVEMBER 22, 2022

CVE NUMBER

CVE-2022-43589

SUMMARY

A null pointer dereference vulnerability exists in the handle_ioctl_8314C functionality of Callback technologies CBFS Filter 20.0.8317. A specially-crafted I/O request packet (IRP) can lead to denial of service. An attacker can issue an ioctl to trigger this vulnerability.

CONFIRMED VULNERABLE VERSIONS

The versions below were either tested or verified to be vulnerable by Talos or confirmed to be vulnerable by the vendor.

Callback technologies CBFS Filter 20.0.8317

PRODUCT URLS

CBFS Filter - https://www.callback.com/cbfsfilter/

CVSSV3 SCORE

6.2 - CVSS:3.0/AV:L/AC:L/PR:N/UI:N/S:U/C:N/I:N/A:H

CWE

CWE-476 - NULL Pointer Dereference

DETAILS

A windows device driver is almost like a kernel DLL that, once loaded, provides additional features. In order to communicate with these device drivers, Windows has a major component named Windows I/O Manager. The Windows IO Manager is responsible for the interface between user applications and device drivers. It implements I/O Request Packets (IRP) to enable the communication with the devices drivers, answering to all I/O requests. For more information see the Microsoft website.

The driver is responsible for creating a device interface with different functions to answer to specific codes, named major code function. If the designer wants to implement customized functions into a driver, there is one major function code named `IRP_MJ_DEVICE_CONTROL`. By handling such major code function, device drivers will support specific I/O Control Code (IOCTL) through a dispatch routine.

The Windows I/O Manager provides three different methods to enable the shared memory: Buffered I/O, Direct I/O, Neither I/O.
Without getting into the details of the IO Manager mechanisms, the method Buffered I/O is often the easiest one for handling memory user buffers from a device perspective.
The I/O Manager is providing all features to enable device drivers sharing buffers between userspace and kernelspace. It will be responsible for copying data back and forth.

Let's see some examples of routines (which you should not copy as is) that explain how things work.
When creating a driver, you'll have several functions to implement, and you'll find some dispatcher routines to handle different IRP as follows:

```
extern "C"
NTSTATUS DriverEntry(_In_ PDRIVER_OBJECT pDriverObject, _In_ PUNICODE_STRING RegistryPath)
{
 [...]
        pDriverObject->DriverUnload = DriverUnload;
        pDriverObject->MajorFunction[IRP_MJ_DEVICE_CONTROL] = DriverIOctl;
        pDriverObject->MajorFunction[IRP_MJ_CREATE] = DriverCreate;
        pDriverObject->MajorFunction[IRP_MJ_CLOSE] = DriverClose;
[...]
}
```

The DriverEntry is the function `main` for a driver. This is the place where initializations start.

We can see for example the `pDriverObject` which is a `PDRIVER_OBJECT` object given by the system to associate different routines, to be called against specific codes, into the `Majorfunction` table `IRP_MJ_DEVICE_CONTROL` for `DriverIOctl` etc.

Then later inside the driver you'll see the implementation of the `DriverIOctl` routine responsible for handling the IOCTL code. It can be something like below:

```
NTSTATUS DriverIOctl(PDEVICE_OBJECT pDevObject, PIRP pIrp)
{
   [...]
    auto pIrpSp = IoGetCurrentIrpStackLocation(pIrp);
    switch (pIrpSp->Parameters.DeviceIoControl.IoControlCode)
    {

    case IO_CREATE_EXAMPLE:
            ioctl_inbuffer_data = (ioctl_inbuffer*)pIrp->AssociatedIrp.SystemBuffer;
            auto InputBufferLength = pIrpSp->Parameters.DeviceIoControl.InputBufferLength;
            auto OutputBufferLength = pIrpSp->Parameters.DeviceIoControl.OutputBufferLength;
     [...] some code

        pIrp->IoStatus.Information = 0;
        pIrp->IoStatus.Status = status;

        break;
     }

     pIrp->IoStatus.Information = some value;
     pIrp->IoStatus.Status = status;
     return status;
}
```

First we can see the `pIrp` pointer to an IRP structure (the description would be out of the scope of this document). Keep in mind this pointer will be useful for accessing data.
So here for example we can observe some switch-case implementation depending on the `IoControlCode` IOCTL. When the device driver gets an IRP with code value `IO_CREATE_EXAMPLE`, it performs the operations below the case. To get into the buffer data exchanged between userspace and kernelspace and vice-versa, we'll look into `SystemBuffer` passed as an argument through the `pIrp` pointer.

On the device side, the pointer inside an IRP represents the user buffer, usually a field named `Irp->AssociatedIrp.SystemBuffer`, when the buffered I/O mechanism is chosen. The specification of the method is indicated by the code itself.
On the userspace side, an application would need to gain access to the device driver symbolic link if it exists, then send some ioctl requests as follows:

```
success  = ::DeviceIoControl(
    ghDevice,
    IO_CREATE_EXAMPLE,                  // control code
    &gpIoctl,                           // input buffer
    sizeof(struct ioctl_inbuffer),      // input buffer length
    &gpIoctl,                           // output buffer
    sizeof(struct ioctl_inbuffer),      // output buffer length
    &returned,
    nullptr
);
```

Such a call will result in an IRP with a major code `IRP_MJ_DEVICE_CONTROL` and a control code to `IO_CREATE_EXAMPLE`. The buffer passed from userspace here as input `gpIoctl`, and output will be accessible from the device driver in the kernelspace via `pIrp->AssociatedIrp.SystemBuffer`. The lengths specified on the `DeviceIoControl` parameters will be used to build the IRP, and the device would be able to get them into the `InputBufferLength` and the `OutputBufferLength` respectively.

Now below we'll see one example of sending a correct output buffer length directly without providing the driver some previous context which can lead to different behaviors and more frequently a local denial of service and blue screen of death through the usage of the device driver `cbfilter20`.

After the system has normally booted and the driver is running, sending an IOCTL 0x8314C with a valid buffer output size lead to the following situation

```
*** Fatal System Error: 0x0000003b
                       (0x00000000C0000005,0xFFFFF80483B05A39,0xFFFF8406C39D1700,0x0000000000000000)

Break instruction exception - code 80000003 (first chance)

A fatal system error has occurred.
Debugger entered on first try; Bugcheck callbacks have not been invoked.

A fatal system error has occurred.

nt!DbgBreakPointWithStatus:
fffff804`80000f70 cc              int     3
```

While looking at analysis output we can see an access violation while attempting to dereference a null pointer

```
rax=ffff958db7744630 rbx=ffff958db28f1510 rcx=0000000000000000
rdx=ffff958db28f1510 rsi=ffff958db20b7cd0 rdi=ffff958db28f1510
rip=fffff80483b05a39 rsp=ffff8406c39d2100 rbp=ffff8406c39d2150
 r8=0000000000000493  r9=ffff958db20b7cd0 r10=fffff80483ae5780
r11=0000000000000000 r12=0000000000000000 r13=0000000000000000
r14=ffff958db7744630 r15=ffff958db20b7cd0
iopl=0         nv up ei pl zr na po nc
cs=0010  ss=0018  ds=002b  es=002b  fs=0053  gs=002b             efl=00050246

cbfilter20+0x25a39:
fffff804`83b05a39 488b4908        mov     rcx,qword ptr [rcx+8] ds:002b:00000000`00000008=????????????????
Resetting default scope
```

When looking at pseudo code corresponding to the culprit function we can see the following:

```
LINE1  MACRO_STATUS __fastcall handle_ioctl_8314C(struct _DEVICE_OBJECT *a1, PIRP a2)
LINE2  {
LINE3    _IO_STACK_LOCATION *CurrentStackLocation; // rax
LINE4    struct _LIST_ENTRY **v5; // rax
LINE5
LINE6    CurrentStackLocation = a2->Tail.Overlay.CurrentStackLocation;
LINE7    if ( CurrentStackLocation->Parameters.DeviceIoControl.OutputBufferLength != 4 )
LINE8      return STATUS_INVALID_PARAMETER;
LINE9    v5 = sub_0_FFFFF8006F656E90(*((struct _FILE_OBJECT **)CurrentStackLocation->FileObject->FsContext2 + 1));
LINE10   *(_DWORD *)a2->AssociatedIrp.SystemBuffer = (*((_DWORD *)v5 + 58) >> 7) & 1;
LINE11   a2->IoStatus.Information = 4i64;
LINE12   sub_0_FFFFF8006F656E08((__int64)v5);
LINE13   return 0i64;
LINE14 }
```

The ioctl handler `handle_ioctl_8314C` is directly calling and relying over a `FileContext` provided through the `CurrentStackLocation` of the IRP. The null pointer dereference is at LINE9 while processing the `CurrentStackLocation` structure which is derived from an PIRP passed as parameter `a2` LINE6.

Parsing the call stack below permit us to easily identify the IRP value as the second value in the callstack to the function `IofCallDriver`

```
STACK_TEXT:
    ffff8406`c39d2100 fffff804`83afc60d     : 00000000`00000010 00000000`00050246 ffff8406`c39d2158 00000000`00000018 : cbfilter20+0x25a39
    ffff8406`c39d2130 fffff804`83ae57f3     : 958db774`00000000 ffff958d`b28f1510 00000000`00000001 ffff958d`b20b7cd0 : cbfilter20+0x1c60d
    ffff8406`c39d2160 fffff804`7fe917d5     : 00000000`0000000e 00000000`00000000 ffff958d`b28f1510 00000000`00000001 : cbfilter20+0x57f3
    ffff8406`c39d21c0 fffff804`80277a08     : ffff8406`c39d2540 ffff958d`b28f1510 00000000`00000001 ffff958d`b6bf0080 :
nt!IofCallDriver+0x55
    ffff8406`c39d2200 fffff804`802772d5     : 00000000`0008314c ffff8406`c39d2540 00000000`00000005 ffff8406`c39d2540 :
nt!IopSynchronousServiceTail+0x1a8
    ffff8406`c39d22a0 fffff804`80276cd6     : 00000000`09895008 00000000`00000000 00000000`00000000 00000000`00000000 :
nt!IopXxxControlFile+0x5e5
    ffff8406`c39d23e0 fffff804`8000aab5     : 00000000`00000000 00000000`00000000 00000000`00000000 00000000`0037ddc8 :
nt!NtDeviceIoControlFile+0x56
    ffff8406`c39d2450 00000000`77b41cfc     : 00000000`77b41933 00000023`77bc385c 00007ff8`7a4e0023 00000000`09895000 :
nt!KiSystemServiceCopyEnd+0x25
    00000000`0037e718 00000000`77b41933     : 00000023`77bc385c 00007ff8`7a4e0023 00000000`09895000 00000000`006fea60 :
wow64cpu!CpupSyscallStub+0xc
    00000000`0037e720 00000000`77b411b9     : 00000000`006ff764 00007ff8`7a4e39b4 00000000`0037e7f0 00007ff8`7a4e3aaf :
wow64cpu!DeviceIoctlFileFault+0x31
    00000000`0037e7d0 00007ff8`7a4e38c9     : 00000000`00569000 00000000`002a0080 00000000`00000000 00000000`0037f040 :
wow64cpu!BTCpuSimulate+0x9
    00000000`0037e810 00007ff8`7a4e32bd     : 00000000`00000000 00000000`007c2418 00000000`00000000 00000000`00000000 :
wow64!RunCpuSimulation+0xd
    00000000`0037e840 00007ff8`7c343592     : 00000000`00000010 00000000`00000010 00007ff8`7c3a1a90 00000000`00568000 :
wow64!Wow64LdrpInitialize+0x12d
    00000000`0037eaf0 00007ff8`7c2e4cdb     : 00000000`00000001 00000000`00000000 00000000`00000000 00000000`00000001 :
ntdll!LdrpInitializeProcess+0x1932
    00000000`0037ef20 00007ff8`7c2e4b63     : 00000000`00000000 00007ff8`7c270000 00000000`00000000 00000000`0056a000 :
ntdll!LdrpInitialize+0x15f
    00000000`0037efc0 00007ff8`7c2e4b0e     : 00000000`0037f040 00000000`00000000 00000000`00000000 00000000`00000000 :
ntdll!LdrpInitialize+0x3b
    00000000`0037eff0 00000000`00000000     : 00000000`00000000 00000000`00000000 00000000`00000000 00000000`00000000 :
ntdll!LdrInitializeThunk+0xe
```

Looking into the IRP from analysis give us the following :

```
1: kd> dt nt!_IRP ffff958d`b28f1510
   +0x000 Type             : 0n6
   +0x002 Size             : 0x118
   +0x004 AllocationProcessorNumber : 1
   +0x006 Reserved         : 0
   +0x008 MdlAddress       : (null)
   +0x010 Flags            : 0x60070
   +0x018 AssociatedIrp    : <anonymous-tag>
   +0x020 ThreadListEntry  : _LIST_ENTRY [ 0xffff958d`b60f9530 - 0xffff958d`b60f9530 ]
   +0x030 IoStatus         : _IO_STATUS_BLOCK
   +0x040 RequestorMode    : 1 ''
   +0x041 PendingReturned  : 0 ''
   +0x042 StackCount       : 1 ''
   +0x043 CurrentLocation  : 1 ''
   +0x044 Cancel           : 0 ''
   +0x045 CancelIrql       : 0 ''
   +0x046 ApcEnvironment   : 0 ''
   +0x047 AllocationFlags  : 0x6 ''
   +0x048 UserIosb         : 0x00000000`0037e770 _IO_STATUS_BLOCK
   +0x050 UserEvent        : (null)
   +0x058 Overlay          : <anonymous-tag>
   +0x068 CancelRoutine    : (null)
   +0x070 UserBuffer       : 0x00000000`029d041c Void
   +0x078 Tail             : <anonymous-tag>
```

Windbg gives us more information on the stack location

```
1: kd> dx -id 0,0,ffff958db6bf0080 -r1 (*((ntkrnlmp!_IRP *)0xffff958db28f1510)).Tail
(*((ntkrnlmp!_IRP *)0xffff958db28f1510)).Tail                 [Type: <anonymous-tag>]
    [+0x000] Overlay          [Type: <anonymous-tag>]
    [+0x000] Apc              [Type: _KAPC]
    [+0x000] CompletionKey    : 0x0 [Type: void *]
1: kd> dx -r1 (*((ntkrnlmp!_IRP *)0xffff958db28f1510)).Tail.Overlay
(*((ntkrnlmp!_IRP *)0xffff958db28f1510)).Tail.Overlay                 [Type: <anonymous-tag>]
    [+0x000] DeviceQueueEntry [Type: _KDEVICE_QUEUE_ENTRY]
    [+0x000] DriverContext    [Type: void * [4]]
    [+0x020] Thread           : 0xffff958db60f9080 [Type: _ETHREAD *]
    [+0x028] AuxiliaryBuffer  : 0x0 [Type: char *]
    [+0x030] ListEntry        [Type: _LIST_ENTRY]
    [+0x040] CurrentStackLocation : 0xffff958db28f15e0 : IRP_MJ_DEVICE_CONTROL / 0x0 for Device for "\FileSystem\cbfilter20" [Type: 
_IO_STACK_LOCATION *]
    [+0x040] PacketType       : 0xb28f15e0 [Type: unsigned long]
    [+0x048] OriginalFileObject : 0xffff958db7744630 [Type: _FILE_OBJECT *]
    [+0x050] IrpExtension     : 0x0 [Type: void *]
```

Now we got the `CurrentStackLocation`at offset [+0x040] from the `Overlay`, we can look into it:

```
1: kd> dt nt!_IO_STACK_LOCATION 0xffff958db28f15e0
   +0x000 MajorFunction    : 0xe ''
   +0x001 MinorFunction    : 0 ''
   +0x002 Flags            : 0x5 ''
   +0x003 Control          : 0 ''
   +0x008 Parameters       : <anonymous-tag>
   +0x028 DeviceObject     : 0xffff958d`b20b7cd0 _DEVICE_OBJECT
   +0x030 FileObject       : 0xffff958d`b7744630 _FILE_OBJECT
   +0x038 CompletionRoutine : (null)
   +0x040 Context          : (null)
```

Then we can look into the `FileObject` offset `+0x030` of the `_IO_STACK_LOCATION`

```
1: kd> dx -id 0,0,ffff958db6bf0080 -r1 ((ntkrnlmp!_FILE_OBJECT *)0xffff958db7744630)
((ntkrnlmp!_FILE_OBJECT *)0xffff958db7744630)                 : 0xffff958db7744630 [Type: _FILE_OBJECT *]
    [+0x000] Type             : 5 [Type: short]
    [+0x002] Size             : 216 [Type: short]
    [+0x008] DeviceObject     : 0xffff958db20b7cd0 : Device for "\FileSystem\cbfilter20" [Type: _DEVICE_OBJECT *]
    [+0x010] Vpb              : 0x0 [Type: _VPB *]
    [+0x018] FsContext        : 0x0 [Type: void *]
    [+0x020] FsContext2       : 0x0 [Type: void *]
    [+0x028] SectionObjectPointer : 0x0 [Type: _SECTION_OBJECT_POINTERS *]
    [+0x030] PrivateCacheMap  : 0x0 [Type: void *]
    [+0x038] FinalStatus      : 0 [Type: long]
    [+0x040] RelatedFileObject : 0x0 [Type: _FILE_OBJECT *]
    [+0x048] LockOperation    : 0x0 [Type: unsigned char]
    [+0x049] DeletePending    : 0x0 [Type: unsigned char]
    [+0x04a] ReadAccess       : 0x0 [Type: unsigned char]
    [+0x04b] WriteAccess      : 0x0 [Type: unsigned char]
    [+0x04c] DeleteAccess     : 0x0 [Type: unsigned char]
    [+0x04d] SharedRead       : 0x0 [Type: unsigned char]
    [+0x04e] SharedWrite      : 0x0 [Type: unsigned char]
    [+0x04f] SharedDelete     : 0x0 [Type: unsigned char]
    [+0x050] Flags            : 0x40002 [Type: unsigned long]
    [+0x058] FileName         : "" [Type: _UNICODE_STRING]
    [+0x068] CurrentByteOffset : {0} [Type: _LARGE_INTEGER]
    [+0x070] Waiters          : 0x0 [Type: unsigned long]
    [+0x074] Busy             : 0x1 [Type: unsigned long]
    [+0x078] LastLock         : 0x0 [Type: void *]
    [+0x080] Lock             [Type: _KEVENT]
    [+0x098] Event            [Type: _KEVENT]
    [+0x0b0] CompletionContext : 0x0 [Type: _IO_COMPLETION_CONTEXT *]
    [+0x0b8] IrpListLock      : 0x0 [Type: unsigned __int64]
    [+0x0c0] IrpList          [Type: _LIST_ENTRY]
    [+0x0d0] FileObjectExtension : 0x0 [Type: void *]
```

And we finally can see here the null pointer `FsContext2` at `+0x020` of the `FILEOBJECT` structure. The vulnerability exists as there is no checking for a null pointer into the code which lead to an access violation when there is no file context initialized which then leads to a denial of service.

Crash Information

```
*******************************************************************************
*                                                                             *
*                        Bugcheck Analysis                                    *
*                                                                             *
*******************************************************************************

SYSTEM_SERVICE_EXCEPTION (3b)
An exception happened while executing a system service routine.
Arguments:
Arg1: 00000000c0000005, Exception code that caused the bugcheck
Arg2: fffff80483b05a39, Address of the instruction which caused the bugcheck
Arg3: ffff8406c39d1700, Address of the context record for the exception that caused the bugcheck
Arg4: 0000000000000000, zero.

Debugging Details:
------------------

*** WARNING: Unable to verify checksum for System.Windows.Forms.ni.dll

"C:\WINDOWS\System32\KERNELBASE.dll" was not found in the image list.
Debugger will attempt to load "C:\WINDOWS\System32\KERNELBASE.dll" at given base 00000000`00000000.

Please provide the full image name, including the extension (i.e. kernel32.dll)
for more reliable results.Base address and size overrides can be given as
.reload <image.ext>=<base>,<size>.

KEY_VALUES_STRING: 1

    Key  : Analysis.CPU.mSec
    Value: 2796

    Key  : Analysis.DebugAnalysisManager
    Value: Create

    Key  : Analysis.Elapsed.mSec
    Value: 5375

    Key  : Analysis.Init.CPU.mSec
    Value: 79359

    Key  : Analysis.Init.Elapsed.mSec
    Value: 8247024

    Key  : Analysis.Memory.CommitPeak.Mb
    Value: 82

    Key  : WER.OS.Branch
    Value: vb_release

    Key  : WER.OS.Timestamp
    Value: 2019-12-06T14:06:00Z

    Key  : WER.OS.Version
    Value: 10.0.19041.1


BUGCHECK_CODE:  3b

BUGCHECK_P1: c0000005

BUGCHECK_P2: fffff80483b05a39

BUGCHECK_P3: ffff8406c39d1700

BUGCHECK_P4: 0

CONTEXT:  ffff8406c39d1700 -- (.cxr 0xffff8406c39d1700)
rax=ffff958db7744630 rbx=ffff958db28f1510 rcx=0000000000000000
rdx=ffff958db28f1510 rsi=ffff958db20b7cd0 rdi=ffff958db28f1510
rip=fffff80483b05a39 rsp=ffff8406c39d2100 rbp=ffff8406c39d2150
 r8=0000000000000493  r9=ffff958db20b7cd0 r10=fffff80483ae5780
r11=0000000000000000 r12=0000000000000000 r13=0000000000000000
r14=ffff958db7744630 r15=ffff958db20b7cd0
iopl=0         nv up ei pl zr na po nc
cs=0010  ss=0018  ds=002b  es=002b  fs=0053  gs=002b             efl=00050246
cbfilter20+0x25a39:
fffff804`83b05a39 488b4908        mov     rcx,qword ptr [rcx+8] ds:002b:00000000`00000008=????????????????
Resetting default scope

PROCESS_NAME:  ioctlpus.exe

STACK_TEXT:
ffff8406`c39d2100 fffff804`83afc60d     : 00000000`00000010 00000000`00050246 ffff8406`c39d2158 00000000`00000018 : cbfilter20+0x25a39
ffff8406`c39d2130 fffff804`83ae57f3     : 958db774`00000000 ffff958d`b28f1510 00000000`00000001 ffff958d`b20b7cd0 : cbfilter20+0x1c60d
ffff8406`c39d2160 fffff804`7fe917d5     : 00000000`0000000e 00000000`00000000 ffff958d`b28f1510 00000000`00000001 : cbfilter20+0x57f3
ffff8406`c39d21c0 fffff804`80277a08     : ffff8406`c39d2540 ffff958d`b28f1510 00000000`00000001 ffff958d`b6bf0080 : nt!IofCallDriver+0x55
ffff8406`c39d2200 fffff804`802772d5     : 00000000`0008314c ffff8406`c39d2540 00000000`00000005 ffff8406`c39d2540 :
nt!IopSynchronousServiceTail+0x1a8
ffff8406`c39d22a0 fffff804`80276cd6     : 00000000`09895008 00000000`00000000 00000000`00000000 00000000`00000000 :
nt!IopXxxControlFile+0x5e5
ffff8406`c39d23e0 fffff804`8000aab5     : 00000000`00000000 00000000`00000000 00000000`00000000 00000000`0037ddc8 :
nt!NtDeviceIoControlFile+0x56
ffff8406`c39d2450 00000000`77b41cfc     : 00000000`77b41933 00000023`77bc385c 00007ff8`7a4e0023 00000000`09895000 :
nt!KiSystemServiceCopyEnd+0x25
00000000`0037e718 00000000`77b41933     : 00000023`77bc385c 00007ff8`7a4e0023 00000000`09895000 00000000`006fea60 :
wow64cpu!CpupSyscallStub+0xc
00000000`0037e720 00000000`77b411b9     : 00000000`006ff764 00007ff8`7a4e39b4 00000000`0037e7f0 00007ff8`7a4e3aaf :
wow64cpu!DeviceIoctlFileFault+0x31
00000000`0037e7d0 00007ff8`7a4e38c9     : 00000000`00569000 00000000`002a0080 00000000`00000000 00000000`0037f040 :
wow64cpu!BTCpuSimulate+0x9
00000000`0037e810 00007ff8`7a4e32bd     : 00000000`00000000 00000000`007c2418 00000000`00000000 00000000`00000000 :
wow64!RunCpuSimulation+0xd
00000000`0037e840 00007ff8`7c343592     : 00000000`00000010 00000000`00000010 00007ff8`7c3a1a90 00000000`00568000 :
wow64!Wow64LdrpInitialize+0x12d
00000000`0037eaf0 00007ff8`7c2e4cdb     : 00000000`00000001 00000000`00000000 00000000`00000000 00000000`00000001 :
ntdll!LdrpInitializeProcess+0x1932
00000000`0037ef20 00007ff8`7c2e4b63     : 00000000`00000000 00007ff8`7c270000 00000000`00000000 00000000`0056a000 :
ntdll!LdrpInitialize+0x15f
00000000`0037efc0 00007ff8`7c2e4b0e     : 00000000`0037f040 00000000`00000000 00000000`00000000 00000000`00000000 :
ntdll!LdrpInitialize+0x3b
00000000`0037eff0 00000000`00000000     : 00000000`00000000 00000000`00000000 00000000`00000000 00000000`00000000 :
ntdll!LdrInitializeThunk+0xe


SYMBOL_NAME:  cbfilter20+25a39

MODULE_NAME: cbfilter20
```

```
IMAGE_NAME:  cbfilter20.sys

STACK_COMMAND:  .cxr 0xffff8406c39d1700 ; kb

BUCKET_ID_FUNC_OFFSET:  25a39

FAILURE_BUCKET_ID:  0x3B_c0000005_cbfilter20!unknown_function

OS_VERSION:  10.0.19041.1

BUILDLAB_STR:  vb_release

OSPLATFORM_TYPE:  x64

OSNAME:  Windows 10

FAILURE_ID_HASH:  {d9988c79-0351-22ce-dca4-df9fb9185d5b}

Followup:     MachineOwner
---------
```

TIMELINE

2022-11-04 - Vendor Disclosure
2022-11-04 - Initial Vendor Contact
2022-11-22 - Public Release

CREDIT

Discovered by Emmanuel Tacheau of Cisco Talos.

VULNERABILITY REPORTS | PREVIOUS REPORT | NEXT REPORT

TALOS-2022-1649 | TALOS-2022-1647