Daax R
(https://revers.engineering/)

(https://twitter.com/daaximus)

Nick Peterson
(https://revers.engineering/)

(https://twitter.com/4d4ndy)

Aidan Khoury
(https://revers.engineering/)

(https://www.linkedin.com/in/aidan-khoury)

On **February 16, 2024** By **daax (https://revers.engineering/author/daax/)**

Beyond Process And Object Callbacks: An Unconventional Method

Overview

In this article, I wanted to introduce a fun approach to performing functions similar to those enabled by **Windows Object Callbacks** (https://learn.microsoft.com/en-us/windows-hardware/drivers/kernel/callback-objects) but through an alternative means. It's well known that anti-malware, anti-cheat, and generic monitoring tools on Windows systems often use these callbacks. However, their usability is limited to parties with signed modules, and the callbacks come with some risks, mainly the ease with which these callbacks can be tampered with if not adequately validated. I'll be showcasing a simple example of leveraging this undocumented method. We'll explore how the proposed method could achieve comparable outcomes and more flexibility depending on the object type. I won't spend much time on high-level details of Windows objects – I highly recommend **Windows Internals** (https://learn.microsoft.com/en-us/sysinternals/resources/windows-internals) or **Windows Kernel Programming** (https://leanpub.com/windowskernelprogrammingsecondedition) for more details. In no particular order, we'll cover object construction, the various types, notification routines, and use cases, especially in anti-malware and anti-cheat software, before examining a few issues and then detailing the implementation of alt-process notifications and an anti-debugging method.

> 📢 **Disclaimer**
>
> This implementation was tested on **Windows 11 23H2 (OS Build 22631.3085)**. It will work for any prior versions of Windows 10 as they leverage the same mechanism described in this article. Future deployments of Windows 11 are subject to change these mechanisms and their organization or protections. If you experience issues, please validate the version you are testing on.

Building Blocks

Objects within the Windows kernel are fundamental to the operation and bookkeeping of the OS. I'm assuming mild familiarity with Windows objects, but if you need a refresher, some examples are Process, Thread, File, Mutant, Semaphore, IoRing, etc. They're all constructed by their respective component during

OS initialization and managed by the Object Manager (routines are prefixed `Ob` in ntoskrnl). We'll stick with a familiar object for the following subsections: the Process.

Process Creation and Notification

Process Notification Callbacks in Windows are a cornerstone of system monitoring and security. These callbacks, primarily utilized by anti-malware and anti-cheat systems, offer real-time notifications about process creation and termination events. They'll initialize the appropriate structures and then call `PsSetCreateProcessNotifyRoutine` to register the callback. It may be obvious why security products utilize this mechanism. Still, it enables a wide range of actions for those unfamiliar, from general logging to first-chance validations or process termination based on the information provided within the callback.

When software registers this notification routine, it will be appended to a list of callbacks managed in the kernel labeled `PspCreateProcessNotifyRoutine`. Whenever a process is created by an API such as **NtCreateUserProcess** (https://captmeelo.com/redteam/maldev/2022/05/10/ntcreateuserprocess.html) or **NtCreateProcess** (http://undocumented.ntinternals.net/index.html?page=UserMode%2FUndocumented%20Functions%2FNT%20Objects%2FProcess%2FNtCreateProcess.html) the result will always include the enumeration of this list and subsequent execution of any added callbacks. The general flow from invocation to notification is given below:

```
1.  |- ntdll.dll!NtCreateUserProcess
2.  |    |- ntoskrnl.exe!NtCreateUserProcess
3.  |    |    |- ntoskrnl.exe!PspInsertThread
4.  |    |    |    |- ntoskrnl.exe!PspCallProcessNotifyRoutines
5.  |    |    |    |    |- <N_module>!NmHandleProcessNotification
6.  |    |    |    |    |    |- etc...
```

If we look at the internals of `PspCallProcessNotifyRoutines`, we'll see the enumeration and execution of each callback as they were added.

```
102  if ( (PspNotifyEnableMask & 2) != 0 || ProcessNotify )
103  {
104    PspNotificationIndex = 0;
105    while ( 1 )
106    {
107      PspCallbackBlk = ExReferenceCallBackBlock(&PspCreateProcessNotifyRoutine[PspNotificationIndex]);
108      if ( PspCallbackBlk )
109      {
110        CbContext = PspCallbackBlk->Context;
111        if ( !PicoCtx || (CbContext & 4) != 0 )
112        {
113          if ( (CbContext & 2) != 0 )
114          {
115            if ( ProcessNotify )
116              (PspCallbackBlk->Function)(Process, Process->UniqueProcessId, CreateNotifyInfo);
117          }
118          else
119          {
120            (PspCallbackBlk->Function)(Process->InheritedFromUniqueProcessId, Process->UniqueProcessId, Create);
121          }
122        }
123        ExDereferenceCallBackBlock(&PspCreateProcessNotifyRoutine[PspNotificationIndex], PspCallbackBlk);
124        if ( CreateNotifyInfo )
125        {
126          CreationStatus = CreateNotifyInfo->CreationStatus;
127          if ( CreationStatus < 0 )
128            break;
129        }
130      }
131      if ( ++PspNotificationIndex >= 0x40 )
132        goto use_extension_table;
133    }
134    v9 = CreateNotifyInfo->CreationStatus;
135    PsTerminateProcess(Process, CreationStatus);
```

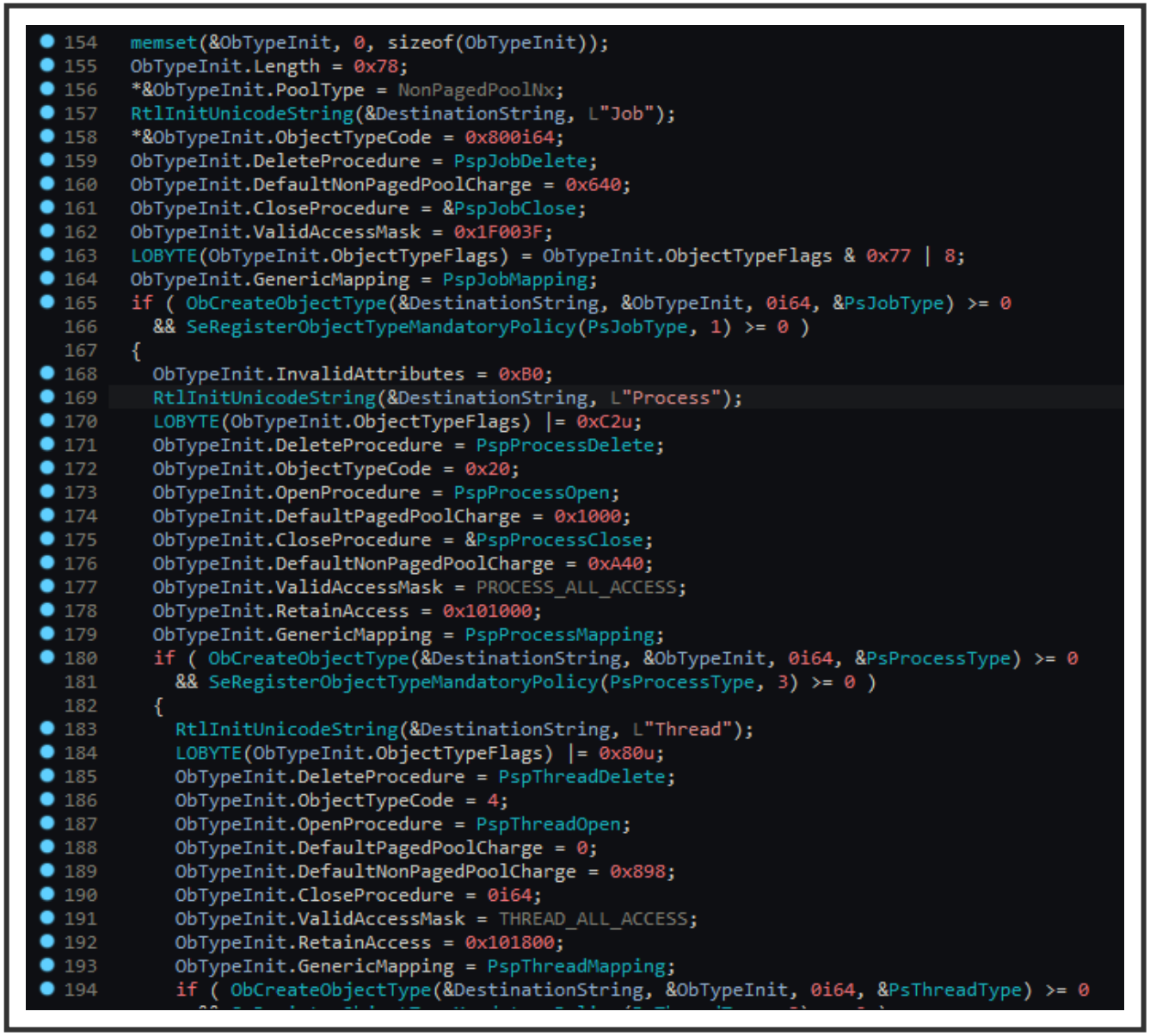
Several methods have been documented for attackers to prevent this first-chance access to process creation. An **older article on this blog** (https://revers.engineering/hiding-drivers-on-windows-10/) addresses one potential method, and the next logical step from seeing the above is to locate the callback entry of interest and remove it from the `PspCreateProcessNotifyRoutine` list. There is **an article that details this approach** (https://br-sn.github.io/Removing-Kernel-Callbacks-Using-Signed-Drivers/) thoroughly. The takeaway is that anti-malware/anti-cheat/general security products typically rely on these

callbacks and may assume they're untampered with; however, as mentioned — attacking the reliability and usability of these mechanisms is somewhat trivial through the abuse of the never-ending number of WHQL-signed drivers that hardware and/or security vendors push out.

Now, let's consider the less legitimate perspective. In years prior, you could register object callbacks and process notification callbacks with an unsigned driver (i.e., using one of those WHQL-signed drivers that allowed unrestricted access to system resources to map your own driver). One method was to perform a little wizardry on the `DriverObject->DriverSection` that is **documented here** (https://revers.engineering/superseding-driver-altitude-checks-on-windows/). However, nowadays, you'll be met with the `STATUS_ACCESS_DENIED` result upon attempting to register object notifications when Windows is not in test-signing mode or without a signed module. This method bypasses the need to modify driver section attributes, sign your driver, or run in test-signing mode.

Function Pointer Rebinding

Alright, no more snooze fest explanation. Let's dive right into how to implement **Process Notification callbacks** (https://learn.microsoft.com/en-us/windows-hardware/drivers/ddi/ntddk/nf-ntddk-pssetcreateprocessnotifyroutine) by avoiding the object callback lists altogether. I will present a single image; I'm sure you will see how this works immediately. If not, fear not... it'll make sense when the first proof-of-concept is presented. Ready?



```
154   memset(&ObTypeInit, 0, sizeof(ObTypeInit));
155   ObTypeInit.Length = 0x78;
156   *&ObTypeInit.PoolType = NonPagedPoolNx;
157   RtlInitUnicodeString(&DestinationString, L"Job");
158   *&ObTypeInit.ObjectTypeCode = 0x800i64;
159   ObTypeInit.DeleteProcedure = PspJobDelete;
160   ObTypeInit.DefaultNonPagedPoolCharge = 0x640;
161   ObTypeInit.CloseProcedure = &PspJobClose;
162   ObTypeInit.ValidAccessMask = 0x1F003F;
163   LOBYTE(ObTypeInit.ObjectTypeFlags) = ObTypeInit.ObjectTypeFlags & 0x77 | 8;
164   ObTypeInit.GenericMapping = PspJobMapping;
165   if ( ObCreateObjectType(&DestinationString, &ObTypeInit, 0i64, &PsJobType) >= 0
166     && SeRegisterObjectTypeMandatoryPolicy(PsJobType, 1) >= 0 )
167   {
168     ObTypeInit.InvalidAttributes = 0xB0;
169     RtlInitUnicodeString(&DestinationString, L"Process");
170     LOBYTE(ObTypeInit.ObjectTypeFlags) |= 0xC2u;
171     ObTypeInit.DeleteProcedure = PspProcessDelete;
172     ObTypeInit.ObjectTypeCode = 0x20;
173     ObTypeInit.OpenProcedure = PspProcessOpen;
174     ObTypeInit.DefaultPagedPoolCharge = 0x1000;
175     ObTypeInit.CloseProcedure = &PspProcessClose;
176     ObTypeInit.DefaultNonPagedPoolCharge = 0xA40;
177     ObTypeInit.ValidAccessMask = PROCESS_ALL_ACCESS;
178     ObTypeInit.RetainAccess = 0x101000;
179     ObTypeInit.GenericMapping = PspProcessMapping;
180     if ( ObCreateObjectType(&DestinationString, &ObTypeInit, 0i64, &PsProcessType) >= 0
181       && SeRegisterObjectTypeMandatoryPolicy(PsProcessType, 3) >= 0 )
182     {
183       RtlInitUnicodeString(&DestinationString, L"Thread");
184       LOBYTE(ObTypeInit.ObjectTypeFlags) |= 0x80u;
185       ObTypeInit.DeleteProcedure = PspThreadDelete;
186       ObTypeInit.ObjectTypeCode = 4;
187       ObTypeInit.OpenProcedure = PspThreadOpen;
188       ObTypeInit.DefaultPagedPoolCharge = 0;
189       ObTypeInit.DefaultNonPagedPoolCharge = 0x898;
190       ObTypeInit.CloseProcedure = 0i64;
191       ObTypeInit.ValidAccessMask = THREAD_ALL_ACCESS;
192       ObTypeInit.RetainAccess = 0x101800;
193       ObTypeInit.GenericMapping = PspThreadMapping;
194       if ( ObCreateObjectType(&DestinationString, &ObTypeInit, 0i64, &PsThreadType) >= 0
```

Ah... nice.

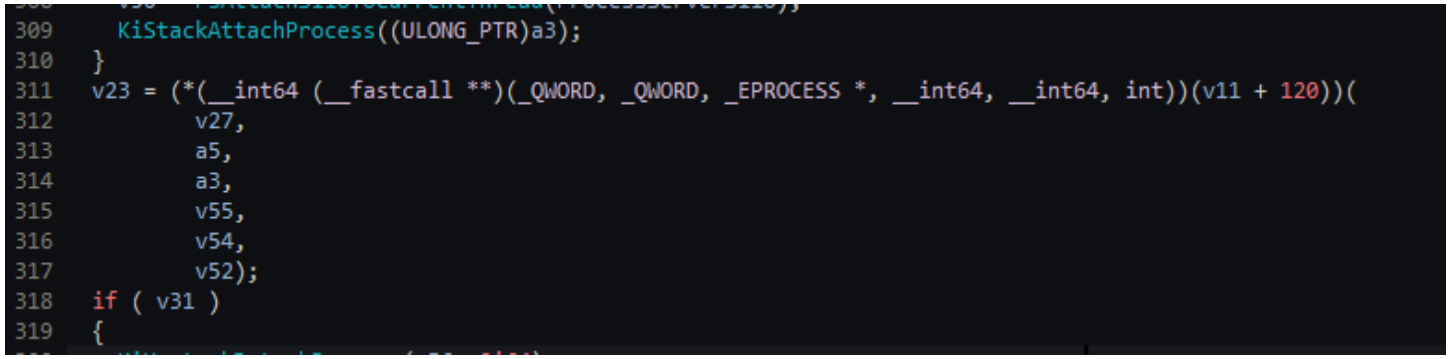
Immediately after applying the appropriate types to variables within `PspInitPhase0` the function, pointers to several methods stand out. Great, yeah, so how do we find the invocations of these? I'm glad you didn't ask, let me show you. I slapped an IDA Python script together to find references to functions at N depth from a starting point. It's terrific for pinpointing opportunities within a target module (yeah, I could've set a breakpoint on `PspProcessOpen` , but I was curious about all indirect invocations in the call graph).

Let's look at a handful of results from the thousands dumped:

```
[1] ------------------------------------------------------------------------------
|    |- ntoskrnl.exe!NtCreateUserProcess
|    |    |- ntoskrnl.exe!PspInsertProcess
|    |    |    |- ntoskrnl.exe!ObInsertObjectEx
|    |    |    |    |- ntoskrnl.exe!ObpCreateHandle
|    |    |    |    |    |- ntoskrnl.exe!ObpIncrementHandleCountEx
|    |    |    |    |    |    |- ntoskrnl.exe!PspChargeQuota
|    |    |    |    |    |    |    |- ntoskrnl.exe!PspExpandQuota @ 0x14048494E
[2] ------------------------------------------------------------------------------
|    |- ntoskrnl.exe!NtCreateUserProcess
|    |    |- ntoskrnl.exe!PspInsertProcess
|    |    |    |- ntoskrnl.exe!ObInsertObjectEx
|    |    |    |    |- ntoskrnl.exe!ObpCreateHandle
|    |    |    |    |    |- ntoskrnl.exe!ObpIncrementHandleCountEx @ 0x14064B733
[3] ------------------------------------------------------------------------------
|    |- ntoskrnl.exe!NtCreateUserProcess
|    |    |- ntoskrnl.exe!PspInsertProcess
|    |    |    |- ntoskrnl.exe!ObInsertObjectEx
|    |    |    |    |- ntoskrnl.exe!ObpCreateHandle
|    |    |    |    |    |- ntoskrnl.exe!ObpIncrementHandleCountEx
|    |    |    |    |    |    |- ntoskrnl.exe!KiUnstackDetachProcess
|    |    |    |    |    |    |    |- ntoskrnl.exe!HalRequestSoftwareInterrupt @
0x140308C63
[4] ------------------------------------------------------------------------------
|    |- ntoskrnl.exe!NtCreateUserProcess
|    |    |- ntoskrnl.exe!PspInsertProcess
|    |    |    |- ntoskrnl.exe!ObInsertObjectEx
|    |    |    |    |- ntoskrnl.exe!ObpCreateHandle
|    |    |    |    |    |- ntoskrnl.exe!ObpInsertOrLocateNamedObject
|    |    |    |    |    |    |- ntoskrnl.exe!ObpLookupObjectName @ 0x14064A502
[5] ------------------------------------------------------------------------------
```

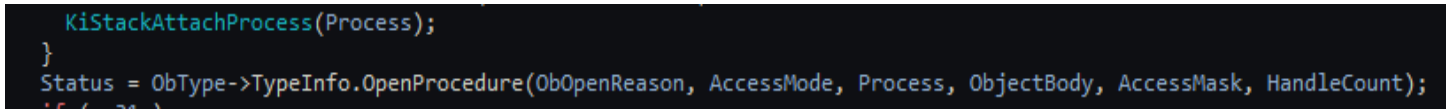
```
|    |- ntoskrnl.exe!NtCreateUserProcess
|    |    |- ntoskrnl.exe!PspInsertProcess
|    |    |    |- ntoskrnl.exe!ObInsertObjectEx
|    |    |    |    |- ntoskrnl.exe!ObpCreateHandle
|    |    |    |    |    |- ntoskrnl.exe!ObpInsertOrLocateNamedObject
|    |    |    |    |    |    |- ntoskrnl.exe!ObpGetObjectSecurity @ 0x140625CF3
[6] ------------------------------------------------------------------------------
|    |- ntoskrnl.exe!NtCreateUserProcess
|    |    |- ntoskrnl.exe!PspInsertProcess
|    |    |    |- ntoskrnl.exe!ObInsertObjectEx
|    |    |    |    |- ntoskrnl.exe!ObpCreateHandle
|    |    |    |    |    |- ntoskrnl.exe!ObpInsertOrLocateNamedObject
|    |    |    |    |    |    |- ntoskrnl.exe!ObpGetObjectSecurity @ 0x140625D97
[7] ------------------------------------------------------------------------------
|    |- ntoskrnl.exe!NtCreateUserProcess
|    |    |- ntoskrnl.exe!PspInsertProcess
|    |    |    |- ntoskrnl.exe!ObInsertObjectEx
|    |    |    |    |- ntoskrnl.exe!ObpCreateHandle
|    |    |    |    |    |- ntoskrnl.exe!ObpInsertOrLocateNamedObject
|    |    |    |    |    |    |- ntoskrnl.exe!ObpDecrementHandleCount @ 0x140674E28
[8] ------------------------------------------------------------------------------
|    |- ntoskrnl.exe!NtCreateUserProcess
|    |    |- ntoskrnl.exe!PspInsertThread
|    |    |    |- ntoskrnl.exe!ObInsertObjectEx
|    |    |    |    |- ntoskrnl.exe!ObpCreateHandle
|    |    |    |    |    |- ntoskrnl.exe!ObpIncrementHandleCountEx
|    |    |    |    |    |    |- ntoskrnl.exe!PspChargeQuota
|    |    |    |    |    |    |    |- ntoskrnl.exe!PspExpandQuota @ 0x14048494E
[9] ------------------------------------------------------------------------------
|    |- ntoskrnl.exe!NtCreateUserProcess
|    |    |- ntoskrnl.exe!PspInsertThread
|    |    |    |- ntoskrnl.exe!ObInsertObjectEx
|    |    |    |    |- ntoskrnl.exe!ObpCreateHandle
|    |    |    |    |    |- ntoskrnl.exe!ObpIncrementHandleCountEx @ 0x14064B733
```

The items at **[2]** and **[9]** were immediately interesting, as I'm unfamiliar with the indirect calls performed within these routines. Upon inspecting the address `0x14064B733` further...



```
309     KiStackAttachProcess((ULONG_PTR)a3);
310   }
311   v23 = (*(__int64 (__fastcall **)(_QWORD, _QWORD, _EPROCESS *, __int64, __int64, int))(v11 + 120))(
312           v27,
313           a5,
314           a3,
315           v55,
316           v54,
317           v52);
318   if ( v31 )
319   {
```

Let's symbolize this a bit.



```
    KiStackAttachProcess(Process);
  }
  Status = ObType->TypeInfo.OpenProcedure(ObOpenReason, AccessMode, Process, ObjectBody, AccessMask, HandleCount);
```

Who needs to open WinDbag when you have DFS? We do... if we're gonna be thorough and verify this gets hit. If we look back at the initial image, we'll see the `ObTypeInit.OpenProcedure` for `PsProcessType` points to `PspProcessOpen`. I'll set a breakpoint in WinDbg to confirm my assumptions: `bp nt!PspProcessOpen "kb;g"`. The results are numerous, but one confirms:

```
 1.  00 fffff806`62b432c3     : 00000000`00000001 ffffc309`606ff040 ffffc309`606b3e60
     ffffc309`00000000 : nt!PspProcessOpen
 2.  01 fffff806`62b404ba     : 00000000`00000200 00000000`00000401 ffffe480`633b0da0
     00000000`00000000 : nt!ObpIncrementHandleCountEx+0x4d3
 3.  02 fffff806`62afef42     : 00000000`00000000 00000000`00000200 ffffc309`67538080
     ffffc309`606b3e60 : nt!ObpCreateHandle+0x21a
 4.  03 fffff806`675d9eb8     : ffffaf06`01719570 ffffbf85`2ea9ea68 ffffbf85`2ea9ea20
     ffffaf06`01719570 : nt!ObOpenObjectByPointer+0x152
 5.  04 fffff806`675ee472     : 00000000`00001558 ffffc309`67538080 00000000`00000424
     ffffbf85`2ea9ea20 : WdFilter!MpCreateProcessContext+0x208
 6.  05 fffff806`675ee04a     : ffffbf85`2ea9ebc0 ffffbf85`2ea9eb00 ffffbf85`2ea9f538
     ffffbf85`2ea9ebc0 : WdFilter!MpHandleProcessNotification+0xe6
 7.  06 fffff806`62b24ab8     : ffffbf85`2ea9ebc0 ffffbf85`2ea9ebc0 00000000`00000000
     ffffbf85`2ea9f538 : WdFilter!MpCreateProcessNotifyRoutineEx+0xaa
 8.  07 fffff806`62b235db     : 00000000`00000000 00000000`00000000 00000000`00000001
     ffffc309`713050c0 : nt!PspCallProcessNotifyRoutines+0x204
 9.  08 fffff806`62b742ce     : ffffc309`707d9080 ffffc309`67538080 ffffbf85`2ea9f400
     ffffbf85`2ea9f2b8 : nt!PspInsertThread+0x72f
```

```
 09 fffff806`6282bbe5     : 00000000`00000000 00000000`00000000 ffffc309`702ca080
fffff806`62aee7f6 : nt!NtCreateUserProcess+0xa2e
 0a 00007fff`3b130d44     : 00000000`00000000 00000000`00000000 00000000`00000000
00000000`00000000 : nt!KiSystemServiceCopyEnd+0x25
```

That's a hit on process creation and was all I needed to justify wasting time messing with this. Alright, so now, how the hell do we utilize this? Well, let's lay out a few things that we know.

- Object Types are created at kernel initialization.
  - REF: `PspInitPhase0`
- Every Object Type has a name associated with it.
  - REF: `ObCreateObjectType(&ObjectTypeName, ...)`
- Object Type objects are stored in the ObTypeIndexTable at their respective indexes.
  - REF: `ObCreateObjectTypeEx[Index] = ObTypeObjectN`
- The procedures in the initial image are stored in the `TypeInfo` field of the **_OBJECT_TYPE** (https://www.vergiliusproject.com/kernels/x86/Windows%208/RTM/_OBJECT_TYPE) structure, which is the type of every entry in the `ObTypeIndexTable`.
- These procedures are not checked to point to the correct internal function by the OS.
- `ObGetObjectType` can be acquired via `MmGetSystemRoutineAddress`.
- **Zydis** (https://github.com/zyantific/zydis) exists.
- lock xchg go brrr.
- ???
- Profit.

Knowing the above, we can instrument these functions to achieve our objective. First, here are some structure definitions you'll want if you want to replicate:

```
typedef struct __declspec( align( 8 ) ) _object_dump_control
{
    void* Stream;
    unsigned int Detail;
} object_dump_control, object_dump_control;

enum e_ob_open_reason : int
{
    ob_create_handle = 0x0,
    ob_open_handle = 0x1,
```

```
11.      ob_duplicate_handle = 0x2,
12.      ob_inherit_handle = 0x3,
13.      ob_max_reason = 0x4,
14.  };
15.
16.  typedef struct _ob_extended_parse_paramters
17.  {
18.      unsigned short length;
19.      unsigned int restricted_access_mask;
20.      _EJOB* silo;
21.  } ob_extended_parse_parameters, * pob_extended_parse_parameters;
22.
23.  typedef struct _object_name_information
24.  {
25.      UNICODE_STRING Name;
26.  } object_name_information, * pobject_name_information;
27.
28.  using dump_procedure_ty = void( __fastcall* )( void*, object_dump_control* );
29.  using open_procedure_ty = int( __fastcall* )( e_ob_open_reason, char, PEPROCESS, void*,
     unsigned int*, unsigned int );
30.  using close_procedure_ty = void( __fastcall* )( PEPROCESS, void*, unsigned long long,
     unsigned long long );
31.  using delete_procedure_ty = void( __fastcall* )( void* );
32.  using parse_procedure_ty = int( __fastcall* )( void*, void*, ACCESS_STATE*, char, unsigned
     int, UNICODE_STRING*, UNICODE_STRING*, void*, SECURITY_QUALITY_OF_SERVICE*, void** );
33.  using parse_procedure_ex_ty = int( __fastcall* )( void*, void*, ACCESS_STATE*, char,
     unsigned int, UNICODE_STRING*, UNICODE_STRING*, void*, SECURITY_QUALITY_OF_SERVICE*,
     ob_extended_parse_parameters*, void** );
34.  using security_procedure_ty = int( __fastcall* )( void*, SECURITY_OPERATION_CODE, unsigned
     int*, void*, unsigned int*, void**, POOL_TYPE, GENERIC_MAPPING*, char );
35.  using query_name_procedure_ty = int( __fastcall* )( void*, unsigned char,
     object_name_information*, unsigned int, unsigned int*, char );
36.  using okay_to_close_procedure_ty = unsigned char( __fastcall* )( PEPROCESS, void*, void*,
     char );
37.
38.  union parse_procedure_detail_ty
39.  {
40.      parse_procedure_ty parse_procedure;
41.      parse_procedure_ex_ty parse_procedure_ex;
42.  };
43.
44.  struct object_type_initializer
45.  {
46.      unsigned short length;
47.
```

```
48.      union
49.      {
50.          unsigned short flags;
51.          unsigned char case_insensitive : 1;
52.          unsigned char unnamed_objects_only : 1;
53.          unsigned char use_default_object : 1;
54.          unsigned char security_required : 1;
55.          unsigned char maintain_handle_count : 1;
56.          unsigned char maintain_type_list : 1;
57.          unsigned char supports_object_callbacks : 1;
58.          unsigned char cache_aligned : 1;
59.          unsigned char use_extended_parameters : 1;
60.          unsigned char reserved : 7;
61.      } object_type_flags;
62.      unsigned int object_type_code;
63.      unsigned int invalid_attributes;
64.      GENERIC_MAPPING generic_mapping;
65.      unsigned int valid_access_mask;
66.      unsigned int retain_access;
67.      POOL_TYPE pool_type;
68.      unsigned int default_paged_pool_charge;
69.      unsigned int default_non_paged_pool_charge;
70.      void( __fastcall* dump_procedure )( void*, object_dump_control* );
71.      int( __fastcall* open_procedure )( e_ob_open_reason, char, PEPROCESS, void*, unsigned
     int*, unsigned int );
72.      void( __fastcall* close_procedure )( PEPROCESS, void*, unsigned long long, unsigned
     long long );
73.      void( __fastcall* delete_procedure )( void* );
74.      union
75.      {
76.          int( __fastcall* parse_procedure )( void*, void*, ACCESS_STATE*, char, unsigned
     int, UNICODE_STRING*, UNICODE_STRING*, void*, SECURITY_QUALITY_OF_SERVICE*, void** );
77.          int( __fastcall* parse_procedure_ex )( void*, void*, ACCESS_STATE*, char, unsigned
     int, UNICODE_STRING*, UNICODE_STRING*, void*, SECURITY_QUALITY_OF_SERVICE*,
     ob_extended_parse_parameters*, void** );
78.      } parse_procedure_detail;
79.      int( __fastcall* security_procedure )( void*, SECURITY_OPERATION_CODE, unsigned int*,
     void*, unsigned int*, void**, POOL_TYPE, GENERIC_MAPPING*, char );
80.      int( __fastcall* query_name_procedure )( void*, unsigned char,
     object_name_information*, unsigned int, unsigned int*, char );
81.      unsigned char( __fastcall* okay_to_close_procedure )( PEPROCESS, void*, void*, char );
82.      unsigned int wait_object_flag_mask;
83.      unsigned short wait_object_flag_offset;
84.      unsigned short wait_object_pointer_offset;
85.  };
```

```
86.
87.  typedef struct _ex_push_lock_flags
88.  {
89.      unsigned long long Locked : 1;
90.      unsigned long long Waiting : 1;
91.      unsigned long long Waking : 1;
92.      unsigned long long MultipleShared : 1;
93.      unsigned long long Shared : 60;
94.  } ex_push_lock_flags;
95.
96.  typedef struct _ex_push_lock
97.  {
98.      union
99.      {
100.         ex_push_lock_flags flags;
101.         unsigned long long value;
102.         void* ptr;
103.     } u;
104. } ex_push_lock, * pex_push_lock;
105.
106. typedef struct object_type
107. {
108.     LIST_ENTRY type_list;
109.     UNICODE_STRING name;
110.     void* default_object;
111.     unsigned char index;
112.     unsigned int total_number_of_objects;
113.     unsigned int total_number_of_handles;
114.     unsigned int high_water_number_of_objects;
115.     unsigned int high_water_number_of_handles;
116.     object_type_initializer type_info;
117.     ex_push_lock type_lock;
118.     unsigned int key;
119.     LIST_ENTRY callback_list;
120. } object_type, *p_object_type;
121.
122. struct ob_type_hook_pair
123. {
124.     object_type* target_object;
125.
126.     dump_procedure_ty               o_dump_procedure;
127.     open_procedure_ty               o_open_procedure;
128.     close_procedure_ty              o_close_procedure;
129.     delete_procedure_ty             o_delete_procedure;
130.     parse_procedure_detail_ty       o_parse_procedure_detail;
```

```
    security_procedure_ty           o_security_procedure;
    query_name_procedure_ty         o_query_name_procedure;
    okay_to_close_procedure_ty      o_okay_to_close_procedure;
};
```

As we noted in our list of “things we know”... we can find the `ObGetObjectType` function, and within it we find the `ObTypeIndexTable`. We’ll do this using Zydis:

```
bool find_ob_type_index_table( void** fn )
{
    auto ob_get_object_type = utils::nt::get_kernel_function( "ObGetObjectType"_w );

    if ( ob_get_object_type == nullptr )
        return false;

    ZydisDecoder zydis_decoder;
    ZydisStatus zydis_status = ZydisDecoderInit(
        &zydis_decoder,
        ZYDIS_MACHINE_MODE_LONG_64,
        ZYDIS_ADDRESS_WIDTH_64
    );

    if ( !ZYDIS_SUCCESS( zydis_status ) )
        return false;

    void* p_ob_type_index_table = nullptr;
    for ( unsigned long it = 0, len = 0; it < 64; it++ )
    {
        ZydisDecodedInstruction inst;
        zydis_status = ZydisDecoderDecodeBuffer(
            &zydis_decoder,
            MAKE_PTR( PVOID, ob_get_object_type, len ),
            16,
            MAKE_PTR( ZydisU64, ob_get_object_type, len ),
            &inst
        );

        if ( !ZYDIS_SUCCESS( zydis_status ) )
            break;

        len += inst.length;

        if ( inst.mnemonic != ZYDIS_MNEMONIC_LEA &&
             inst.operands[ 0 ].type != ZYDIS_OPERAND_TYPE_REGISTER &&
```

```
                inst.operands[ 0 ].size != 64 &&
                inst.operands[ 0 ].reg.value != ZYDIS_REGISTER_RCX ||
                inst.mnemonic == ZYDIS_MNEMONIC_MOVZX
                )
            {
                continue;
            }

            zydis_status = ZydisCalcAbsoluteAddress(
                &inst,
                &inst.operands[ 1 ],
                reinterpret_cast< ZydisU64* >( fn )
            );

            if ( !ZYDIS_SUCCESS( zydis_status ) )
                continue;

            return true;
        }

        return false;
    }
```

Let's tie it in with our `DriverEntry` and verify the result.

```
    extern "C" NTSTATUS
    DriverEntry(
        const PDRIVER_OBJECT driver_object,
        const PUNICODE_STRING registry_path
    )
    {
        __do_global_ctors_aux();

        UNREFERENCED_PARAMETER( registry_path );

        driver_object->DriverUnload = driver_unload;

    #ifdef _SERIAL_LOGGING
        io_initialize_serial_port();
    #endif

        void* ob_type_index_table = nullptr;
        if ( !find_ob_type_index_table( &ob_type_index_table ) )
        {
```

```
20.           OUT_ERR(
21.               "Unable to locate ObTypeIndexTable."
22.           );
23.       }
24.       else
25.       {
26.           OUT_INF(
27.               "ObTypeIndexTable located @ %p",
28.               ob_type_index_table
29.           );
30.       }
31.
32.       if ( ob_type_index_table )
33.       {
34.           // Process the type index table, and rebind the function pointers for our target
    object.
35.           //
36.       }
37.
38.       return STATUS_SUCCESS;
39.   }
```

Processing the ObTypeIndexTable

The name might be self-explanatory to some of the readers already. Still, for completeness, the `ObTypeIndexTable` is an array of pointers to **_OBJECT_TYPE** (https://www.vergiliusproject.com/kernels/x86/Windows%208/RTM/_OBJECT_TYPE) structures that describe the various **Windows Kernel Objects** (https://learn.microsoft.com/en-us/windows-hardware/drivers/kernel/windows-kernel-mode-object-manager) created/registered at OS initialization. If we dump the first few entries and then cast the 3rd element of the array to **_OBJECT_TYPE** (https://www.vergiliusproject.com/kernels/x86/Windows%208/RTM/_OBJECT_TYPE), we'll see the data below.

```
12: kd> dq nt!ObTypeIndexTable
fffff806`6311e630  00000000`00000000 ffffe480`62f8a000
fffff806`6311e640  ffffc309`60676ef0 ffffc309`60689ef0
fffff806`6311e650  ffffc309`60662ef0 ffffc309`6069aef0
fffff806`6311e660  ffffc309`606ade60 ffffc309`606b3e60
fffff806`6311e670  ffffc309`606e5d40 ffffc309`606e5eb0
fffff806`6311e680  ffffc309`606a8c40 ffffc309`606a8db0
fffff806`6311e690  ffffc309`606afa40 ffffc309`606afbb0
fffff806`6311e6a0  ffffc309`606afd20 ffffc309`606afe90
12: kd> dt nt!_OBJECT_TYPE ffffc309`60676ef0
   +0x000 TypeList         : _LIST_ENTRY [ 0xffffc309`60676ea0 - 0xffffc309`697fb330 ]
   +0x010 Name             : _UNICODE_STRING "Type"
   +0x020 DefaultObject    : 0xfffff806`6303f7c0 Void
   +0x028 Index            : 0x2 ''
   +0x02c TotalNumberOfObjects : 0x46
   +0x030 TotalNumberOfHandles : 0
   +0x034 HighWaterNumberOfObjects : 0x46
   +0x038 HighWaterNumberOfHandles : 0
   +0x040 TypeInfo         : _OBJECT_TYPE_INITIALIZER
   +0x0b8 TypeLock         : _EX_PUSH_LOCK
   +0x0c0 Key              : 0x546a624f
   +0x0c8 CallbackList     : _LIST_ENTRY [ 0xffffc309`60676fb8 - 0xffffc309`60676fb8 ]
```

This array's 0th and 1st index are invalid entries, so we will skip these when enumerating the table. If we consider that this is an array of **_OBJECT_TYPE*** (https://www.nirsoft.net/kernel_struct/vista/OBJECT_TYPE.html) and we want to start at a specific index (2, the first valid entry), we can write a helper function like so:

```
1.  auto get_object = [ ob_type_index_table ] ( size_t idx ) -> object_type*
2.  {
3.      return *reinterpret_cast< object_type** >(
4.          static_cast< uint8_t* >( ob_type_index_table ) + idx * sizeof( object_type* )
5.          );
6.  };
```

❗ Weird Legacy

The requirement to index into the **ObTypeIndexTable** (https://codemachine.com/articles/object_headers.html) past the first two entries is a bit odd. It appears that these are placeholder entries. The reasoning for their invalidity is likely historical; my best guess is that they used a different structure for the object type list. The second entry points to **MmBadPointer** (https://learn.microsoft.com/en-us/windows-hardware/drivers/kernel/mm-bad-pointer). However, this is more recent. In late 2018, they used some other magic value `0x0bad0b0b` as described **here** (https://www.sentinelone.com/blog/skream-kernel-mode-exploits-mitigations-rest-us/). All of the current initialization code sets the starting index for the `ObTypeIndexTable` to `2`. This can be verified by analyzing `ObInitSystem` and `ObCreateObjectTypeEx`. I verified that these indexes were not used when Hyper-V and the Windows Sandbox were enabled; only two new object types were introduced: `CrossVmMutant` and `CrossVmEvent`.

If someone knows why the first two entries are invalid, I'd like to learn why.

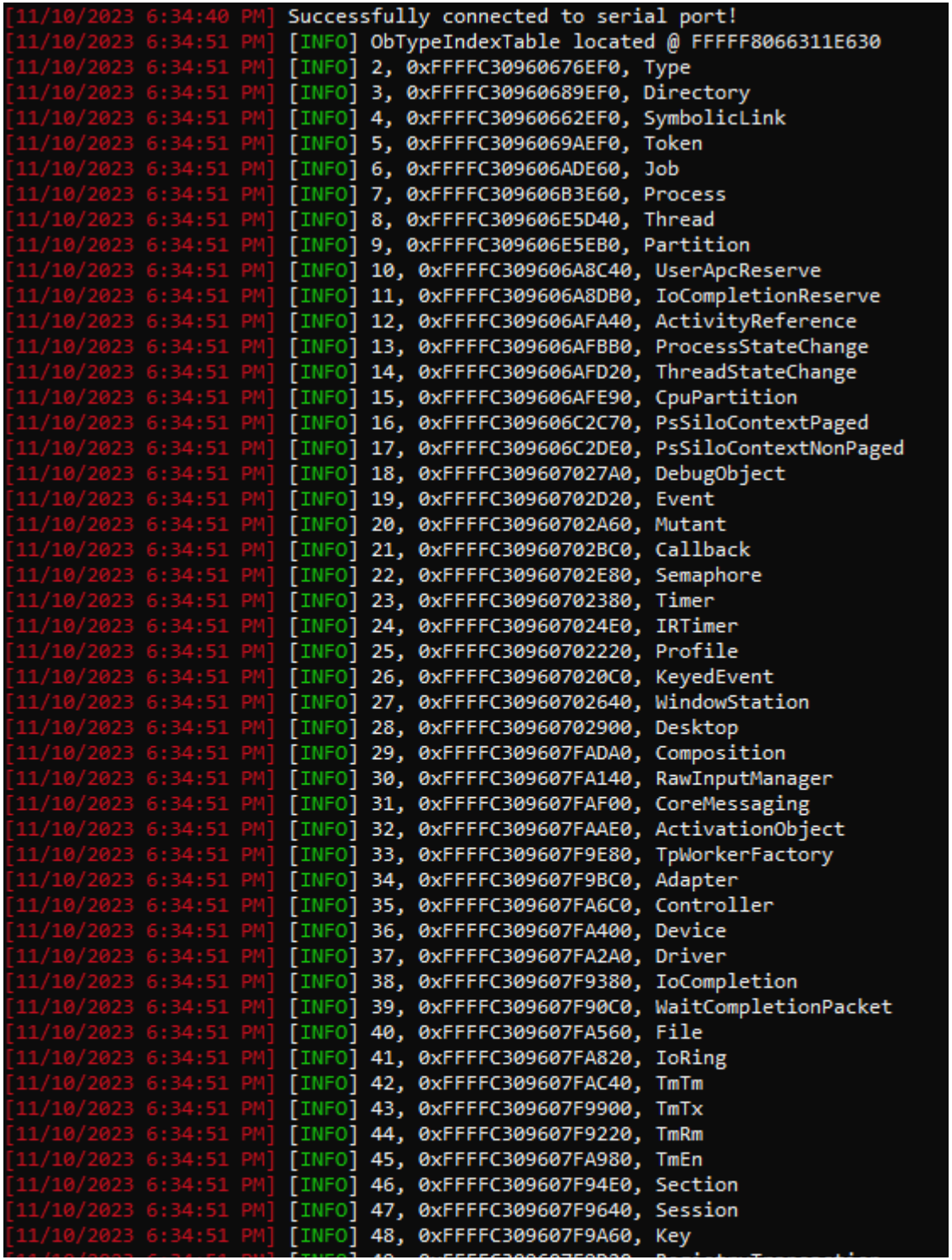
Updating DriverEntry

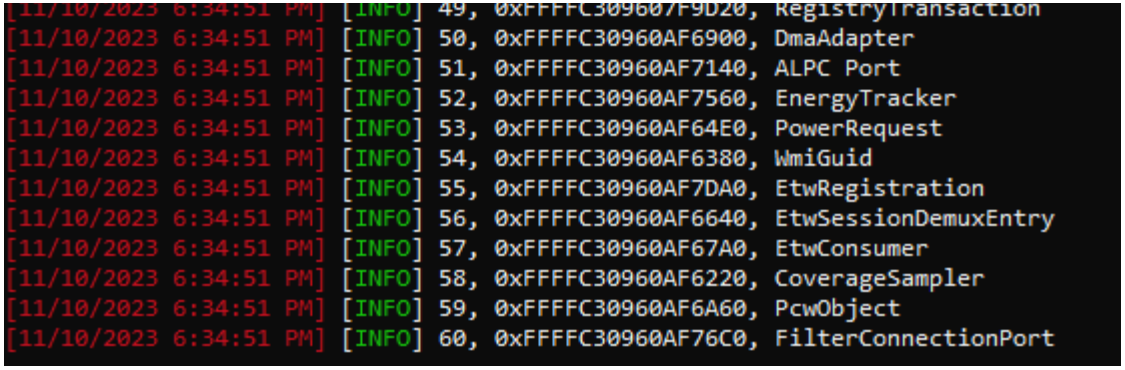
All that's left is to add some logic to our `DriverEntry` and verify we have the correct logic for enumerating and singling out our target type (`PsProcessType`).

```
1.  extern "C" NTSTATUS
2.  DriverEntry(
3.      const PDRIVER_OBJECT driver_object,
4.      const PUNICODE_STRING registry_path
5.  )
6.  {
7.      __do_global_ctors_aux();
8.
9.      UNREFERENCED_PARAMETER( registry_path );
10.
11.     driver_object->DriverUnload = driver_unload;
12.
13. #ifdef _SERIAL_LOGGING
14.     io_initialize_serial_port();
15. #endif
16.
17.     void* ob_type_index_table = nullptr;
```

```
18.      if ( !find_ob_type_index_table( &ob_type_index_table ) )
19.      {
20.          OUT_ERR(
21.              "Unable to locate ObTypeIndexTable."
22.          );
23.      }
24.      else
25.      {
26.          OUT_INF(
27.              "ObTypeIndexTable located @ %p",
28.              ob_type_index_table
29.          );
30.      }
31.
32.      if ( ob_type_index_table )
33.      {
34.          void* fnc = nullptr;
35.
36.          auto get_object = [ ob_type_index_table ] ( size_t idx ) -> object_type*
37.          {
38.              return *reinterpret_cast< object_type** >(
39.                  static_cast< uint8_t* >( ob_type_index_table ) + idx * sizeof(
    object_type* )
40.                  );
41.          };
42.
43.          // Start at the first valid object table index.
44.          //
45.          uint64_t index = 2;
46.          for ( object_type* obj = get_object( index ); obj != nullptr; obj = get_object(
    ++index ) )
47.          {
48.              ANSI_STRING ob_type_name{};
49.              RtlUnicodeStringToAnsiString( &ob_type_name, &obj->name, TRUE );
50.              OUT_INF( "%llu, 0x%p, %s", index, obj, ob_type_name.Buffer );
51.          }
52.      }
53.
54.      return STATUS_SUCCESS;
55.  }
```

Object Type Dump Verification





Looks good; these are all the object types in the object table. I verified against other references from **previous dumps via WinDbg** (https://gist.github.com/daaximus/2cd107e3c5077d569ca88f15ad287371), and all looked good. All that is left is for us to write our function to replace the original function pointer in the **_OBJECT_TYPE_INITIALIZER** (https://www.vergiliusproject.com/kernels/x86/Windows%207/RTM/_OBJECT_TYPE_INITIALIZER) structure for the `PsProcessType` object entry. However, to do this, we need to reverse `PspProcessOpen` to understand how it works. All we know is that it's called at some point during process initialization based on initial analysis. The PspProcessOpen function prototype looks like this:

```
NTSTATUS
__fastcall
PspProcessOpen(
        _OB_OPEN_REASON OpenReason,
        INT8 AccessMode,
        _EPROCESS *TargetProcess,
        _EPROCESS *Object,
        UINT64 *GrantedAccess,
        UINT64 HandleCount);
```

PsProcessType.OpenProcedure (PspProcessOpen) Hook

```
NTSTATUS
process_open_procedure(
    e_ob_open_reason open_reason,
    uint8_t access_mode,
    PEPROCESS process,
    PEPROCESS object_body,
```

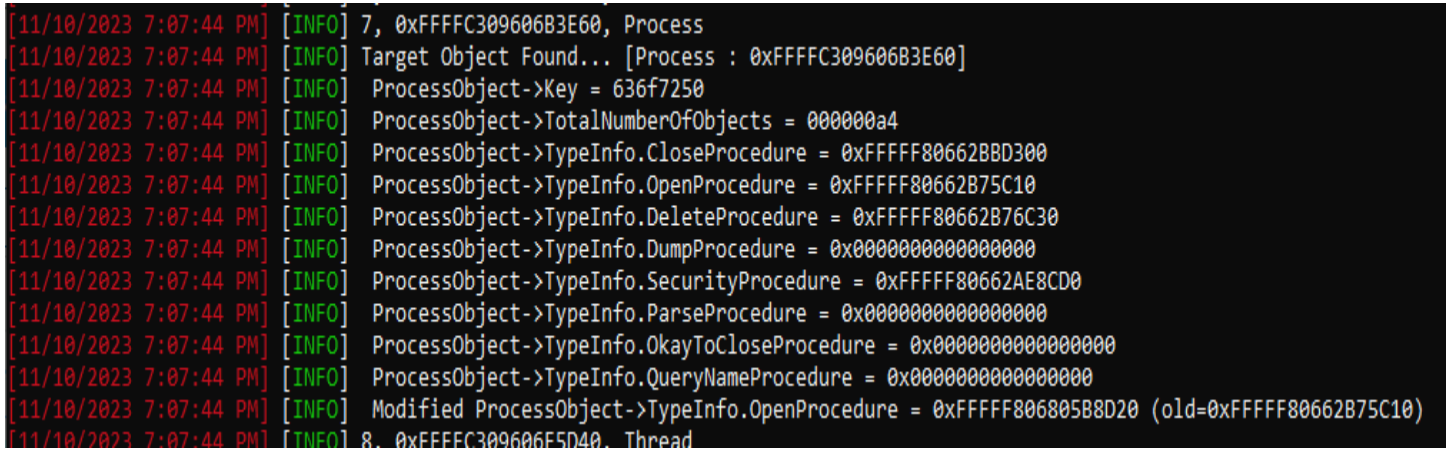
```
7.      unsigned int* granted_access,
8.      unsigned long handle_count)
9.  {
10.     NTSTATUS status = STATUS_SUCCESS;
11.
12.     if (open_reason == e_ob_open_reason::ob_open_handle && process && access_mode == 0)
13.     {
14.         auto allocate_unicode_string = [](size_t size) -> xstd::anyptr<UNICODE_STRING> {
15.             auto ptr = static_cast<PUNICODE_STRING>(ExAllocatePool2(NonPagedPool, size,
    0));
16.             return xstd::anyptr<UNICODE_STRING>(ptr, [](PUNICODE_STRING p) {
17.                 if (p) {
18.                     ExFreePool2(p, 0, nullptr, 0);
19.                 }
20.             });
21.         };
22.
23.         auto primary_name = allocate_unicode_string(0x400);
24.         auto secondary_name = allocate_unicode_string(0x400);
25.
26.         if (!primary_name || !secondary_name)
27.             return STATUS_INSUFFICIENT_RESOURCES;
28.
29.         SeLocateProcessImageName(process, &primary_name);
30.         SeLocateProcessImageName(object_body, &secondary_name);
31.
32.         if (primary_name->Length > 0 && secondary_name->Length > 0)
33.         {
34.             ANSI_STRING aname_parent{};
35.             RtlUnicodeStringToAnsiString(&aname_parent, primary_name.get(), TRUE);
36.
37.             ANSI_STRING aname_child{};
38.             RtlUnicodeStringToAnsiString(&aname_child, secondary_name.get(), TRUE);
39.
40.             if (aname_parent.Length > 0 && aname_child.Length > 0)
41.                 OUT_INF("[PROCESS CREATED] => %s", aname_child.Buffer);
42.         }
43.     }
44.
45.     return g_ob_type_hook_pair.o_open_procedure(open_reason, access_mode, process,
    object_body, granted_access, handle_count);
46. }
```

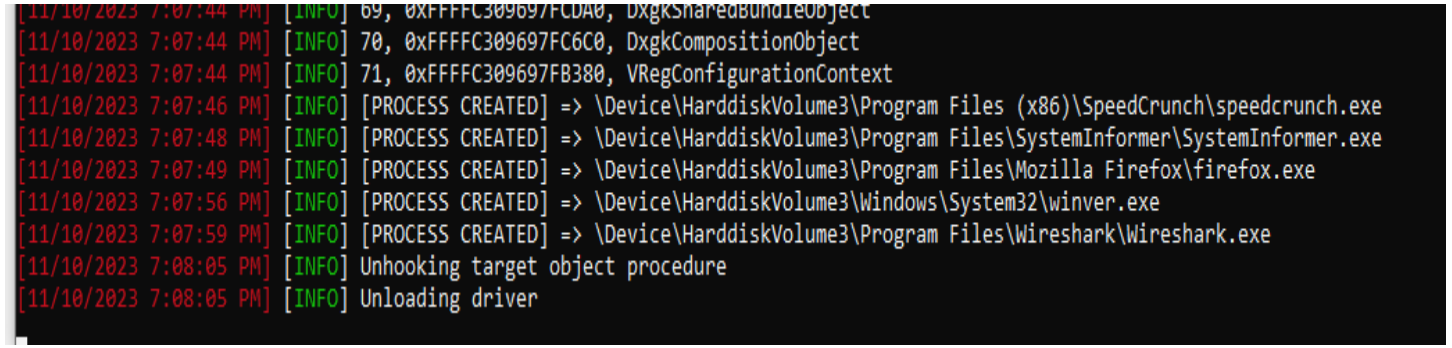
Finalized DriverEntry and Results

```
1.  extern "C" NTSTATUS
2.  DriverEntry(
3.      const PDRIVER_OBJECT driver_object,
4.      const PUNICODE_STRING registry_path
5.  )
6.  {
7.      __do_global_ctors_aux();
8.
9.      UNREFERENCED_PARAMETER( registry_path );
10.
11.     driver_object->DriverUnload = driver_unload;
12.
13. #ifdef _SERIAL_LOGGING
14.     io_initialize_serial_port();
15. #endif
16.
17.     void* ob_type_index_table = nullptr;
18.     if ( !find_ob_type_index_table( &amp;ob_type_index_table ) )
19.     {
20.         OUT_ERR(
21.             "Unable to locate ObTypeIndexTable."
22.         );
23.     }
24.     else
25.     {
26.         OUT_INF(
27.             "ObTypeIndexTable located @ %p",
28.             ob_type_index_table
29.         );
30.     }
31.
32.     if ( ob_type_index_table )
33.     {
34.         void* fnc = nullptr;
35.
36.         auto get_object = [ ob_type_index_table ] ( size_t idx ) -> object_type*
37.         {
38.             return *reinterpret_cast< object_type** >(
39.                 static_cast< uint8_t* >( ob_type_index_table ) + idx * sizeof(
    object_type* )
40.                 );
41.         };
42.
```

```
43.          // Start at the first valid object table index.
44.          //
45.          uint64_t index = 2;
46.          for ( object_type* obj = get_object( index ); obj != nullptr; obj = get_object(
    ++index ) )
47.          {
48.              ANSI_STRING ob_type_name{};
49.              RtlUnicodeStringToAnsiString( &ob_type_name, &obj->name, TRUE );
50.              OUT_INF( "%llu, 0x%p, %s", index, obj, ob_type_name.Buffer );
51.
52.              UNICODE_STRING type_name = RTL_CONSTANT_STRING( L"Process" );
53.
54.              if ( RtlCompareUnicodeString( &obj->name, &type_name, TRUE ) == 0 )
55.              {
56.                  OUT_TYPE_INF("%s", object_type, obj,
57.                      key,
58.                      total_number_of_objects,
59.                      close_procedure,
60.                      open_procedure,
61.                      delete_procedure,
62.                      dump_procedure,
63.                      security_procedure,
64.                      parse_procedure_detail,
65.                      okay_to_close_procedure,
66.                      query_name_procedure
67.                  );
68.
69.                  g_ob_type_hook_pair.target_object = obj;
70.                  g_ob_type_hook_pair.o_close_procedure = obj->type_info.close_procedure;
71.                  g_ob_type_hook_pair.o_open_procedure = obj->type_info.open_procedure;
72.                  g_ob_type_hook_pair.o_delete_procedure = obj->type_info.delete_procedure;
73.                  g_ob_type_hook_pair.o_dump_procedure = obj->type_info.dump_procedure;
74.                  g_ob_type_hook_pair.o_security_procedure = obj-
    >type_info.security_procedure;
75.                  g_ob_type_hook_pair.o_okay_to_close_procedure = obj-
    >type_info.okay_to_close_procedure;
76.                  g_ob_type_hook_pair.o_query_name_procedure = obj-
    >type_info.query_name_procedure;
77.
78.                  _InterlockedExchangePointer(
79.                      reinterpret_cast< void** >( &obj->type_info.open_procedure ),
80.                      reinterpret_cast< void* >( process_open_procedure )
81.                  );
82.              }
83.          }
```

```
84.         }
85.
86.         return STATUS_SUCCESS;
87.     }
```





The result is that only processes created following the rebinding of this function pointer will be logged. You must verify the `access_mode` is 0 and validate the name of the secondary object (not the primary process object) passed to the function, as this is the application being created. The primary process object (3rd argument) is the "parent." The primary process object for new processes will be `System` when `PspProcessOpen` is called. If you log the objects regardless of `access_mode` and `open_reason` you'll be spammed with irrelevant information.

Leverage the SecurityProcedure

An alternative is to leverage the security procedure within the object type initializers structure. During process initialization, it's invoked with the operation code `AssignSecurityDescriptor`, and on process termination, you can catch the `DeleteSecurityDescriptor` case and check for the `PsProcessType` — during normal operations, these two will always indicate process startup/termination.

To determine where it occurred, I initially just traced with the aforementioned script to see where the method was invoked and... unsurprisingly, it was within `ObInsertObjectEx`.

```
[311] ------------------------------------------------------------------------------
|    |- NtCreateUserProcess
|    |    |- PspInsertProcess
|    |    |    |- ObInsertObjectEx @ 0x14062018F
```

❗ For Future Reference

If you're uncertain where to find references to these functions outside of just breaking on the function in WinDbg, think about the operation performed on the object. When an object, such as a mutant, section, semaphore, process, thread, etc., is created, it has to be inserted into the appropriate list. Logically, you may find calls to the respective object procedures in relevant functions like `ObInsertObjectEx` — which is typically the highest Ob-related entry on the call stack, and subsequent calls will invoke one of the procedures for `OpenProcedure` / `SecurityProcedure` at some point, sometimes more than once. Of course, if the object type doesn't initialize those procedures, you won't wind up down that path in practice.

The `OkayToClose` / `Close` / `Delete` procedures will be seen referenced in functions that deal with the release/closing of object handles, ex: `ObCloseHandleTableEntry`. You'll also see the invocation of the `SecurityProcedure` in closing operations. It makes sense because when an object has a security descriptor assigned on construction, it must also have the SD released on destruction. As a thought exercise, consider where you might see the `QueryName` or `DumpProcedure` referenced. I'll provide the answer later on.

After double-checking with WinDbg, the resulting sequence was:

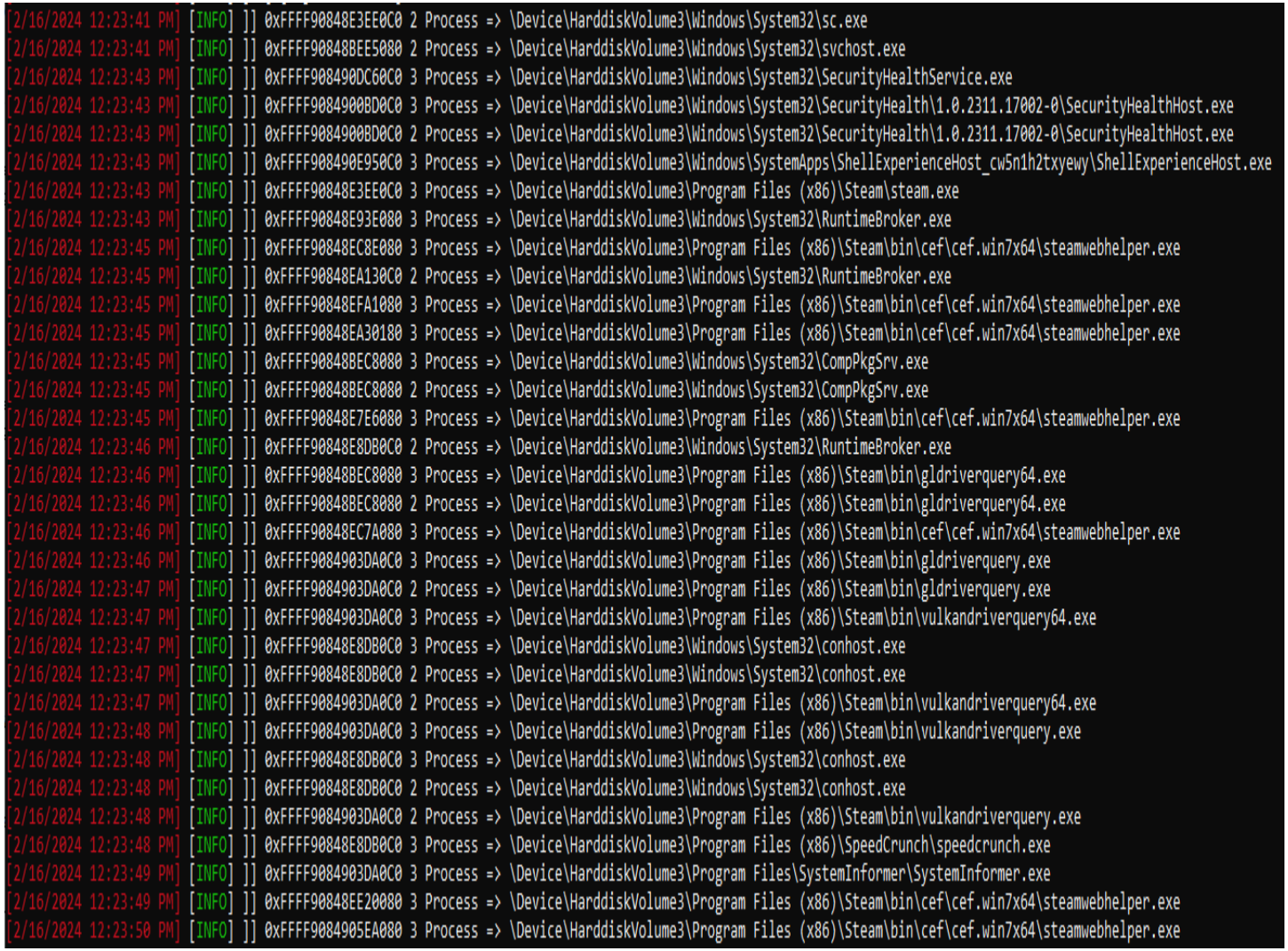
```
|    |- ntoskrnl.exe!KiSystemServiceCopyEnd @ 0x15e2bce5
|    |    |- ntoskrnl.exe!NtCreateUserProcess @ 0x16174275
|    |    |    |- ntoskrnl.exe!PspInsertProcess @ 0x161756f3
|    |    |    |    |- ntoskrnl.exe!ObInsertObjectEx @ 0x160fd663
|    |    |    |    |    |- ntoskrnl.exe!SeDefaultObjectMethod
```

Given this information, we can implement similar to the `OpenProcedure` .

```
 NTSTATUS generic_security_procedure(
     void* object,
     SECURITY_OPERATION_CODE operation_code,
     unsigned* security_information,
     void* security_descriptor,
     unsigned* captured_length,
     void** objects_security_descriptor,
     POOL_TYPE pool_type,
     GENERIC_MAPPING* generic_mapping,
     char access_mode
 ) {
     if (object == nullptr) {
         return g_ob_type_tracking_data[0]->o_security_procedure(
             object, operation_code, security_information, security_descriptor,
             captured_length, objects_security_descriptor, pool_type, generic_mapping,
access_mode
         );
     }

     auto ob_type = ob_get_object_type(object);

     if (!ob_type) {
         return g_ob_type_tracking_data[0]->o_security_procedure(
             object, operation_code, security_information, security_descriptor,
             captured_length, objects_security_descriptor, pool_type, generic_mapping,
access_mode
         );
     }

     if (operation_code == AssignSecurityDescriptor ||
         operation_code == DeleteSecurityDescriptor ||
         operation_code == SetSecurityDescriptor) {

```

```
32.          if (!ob_type->name.Buffer || ob_type->name.Length <= 0)
33.              return STATUS_INVALID_PARAMETER;
34.
35.          auto ob_type_name = xstd::anystr( 0x100, 0x00 ).to_ansi(&ob_type->name);
36.
37.          if (ob_type == *PsProcessType) {
38.              auto primary_name = xstd::anystr( 0x100, 0x00 );
39.              SeLocateProcessImageName(static_cast<PEPROCESS>(object), &primary_name);
40.              auto process_name = primary_name.to_ansi();
41.
42.              OUT_INF("]] 0x%p %d %s => %s", object, operation_code, ob_type_name.c_str(),
process_name.c_str());
43.          }
44.      }
45.
46.      return g_ob_type_tracking_data[ob_type->index - 2]->o_security_procedure(
47.          object, operation_code, security_information, security_descriptor,
48.          captured_length, objects_security_descriptor, pool_type, generic_mapping,
access_mode
49.      );
50.  }
```

Below are the results after logging in and starting a handful of processes.



With a few other checks to determine if the process is in the initial startup phases, you can have your own process creation callbacks without registering with the object manager.

Simple System-wide Anti-debug

We used the process object because it's likely the most familiar to those reading this. However, we don't have to stop there. In the previous section, I noted that all object types with these procedures setup will invoke them at one point or another. What do we know about debugging? Well, if you're not familiar with the internals of debugging then that's something for another post. The only relevant thing to know for this part is that debuggers call **DbgUiConnectToDbg** (https://doxygen.reactos.org/dd/ddc/dbgui_8c.html) API, which is called when a debugger attempts to attach to a process, or, for some, they directly implement their own **DbgUiConnectToDbg** (https://github.dev/x64dbg/TitanEngine/blob/49f59781da9ef9ed8b14963a0ecf499695971b5f/TitanEngine/Global.Debugge The same goes for **DebugActiveProcess** (https://learn.microsoft.com/en-us/windows/win32/api/debugapi/nf-debugapi-debugactiveprocess).

```
1.  // Reference:
    https://github.dev/x64dbg/TitanEngine/blob/49f59781da9ef9ed8b14963a0ecf499695971b5f/TitanEng
2.  //
3.  static NTSTATUS NTAPI DbgUiConnectToDbg_()
4.  {
5.      if(NtCurrentTeb()->DbgSsReserved[1] != NULL)
6.          return STATUS_SUCCESS;
7.
8.      OBJECT_ATTRIBUTES ObjectAttributes;
9.      InitializeObjectAttributes(&ObjectAttributes, NULL, 0, NULL, NULL);
10.     return NtCreateDebugObject(&NtCurrentTeb()->DbgSsReserved[1], DEBUG_ALL_ACCESS, &Object
11. }
```

The name is self-explanatory: **NtCreateDebugObject** (https://ntdoc.m417z.com/ntcreatedebugobject); it will attempt to create and insert a `DebugObject` and the newly created handles into the appropriate object tables. I mentioned earlier that anything that will call into `ObInsertObjectEx` will wind up invoking one of these procedures if they exist and if we check on the "constructor" of `DbgkDebugObjectType` we'll note that there is no overwriting of the `SecurityProcedure`, which means that it will be set to the `SeDefaultObjectMethod`.

```
DbgkObjectType.InvalidAttributes = 0;
DbgkObjectType.DefaultPagedPoolCharge = 0;
DbgkObjectType.DefaultNonPagedPoolCharge = 0;
DbgkObjectType.DeleteProcedure = AlpcMessageDeleteProcedure;
DbgkObjectType.Length = 0x78;
DbgkObjectType.CloseProcedure = &DbgkpCloseObject;
v1 = 8i64;
LOBYTE(DbgkObjectType.ObjectTypeFlags) |= 8u;
DbgkObjectType.ValidAccessMask = 0x1F000F;
DbgkObjectType.GenericMapping.GenericAll = 0x1F000F;
DbgkObjectType.PoolType = NonPagedPoolNx;
DbgkObjectType.GenericMapping.GenericRead = 0x20001;
DbgkObjectType.GenericMapping.GenericWrite = 0x20002;
DbgkObjectType.GenericMapping.GenericExecute = 0x120000;
if ( ObCreateObjectType(&v4, &DbgkObjectType, 0i64, &DbgkDebugObjectType) >= 0 )
```

And a quick trace to confirm this:

```
1.  |    |- ntdll.exe!RtlUserThreadStart @ 0xd346aa58
2.  |    |    |- KERNEL32.dll!BaseThreadInitThunk @ 0xd26f257d
3.  |    |    |    |- x64dbg.exe!UnknownFunction @ 0x609cbd3e
4.  |    |    |    |    |- x64dbg.exe!UnknownFunction @ 0x609d8207
5.  |    |    |    |    |    |- TitanEngine.dll!AttachDebugger @ 0x8f7e5462
6.  |    |    |    |    |    |    |- TitanEngine.dll!UnknownFunction @ 0x8f7d5b55
7.  |    |    |    |    |    |    |    |- ntdll.dll!NtCreateDebugObject @ 0xd34b0884
8.  |    |    |    |    |    |    |    |    |- ntoskrnl.exe!KiSystemServiceCopyEnd @ 0x15e2bce5
9.  |    |    |    |    |    |    |    |    |    |- ntoskrnl.exe!NtCreateDebugObject @
    0x163366e2
10. |    |    |    |    |    |    |    |    |    |    |- ntoskrnl.exe!ObInsertObjectEx @
    0x160fd663
11. |    |    |    |    |    |    |    |    |    |    |    |-
    ntoskrnl.exe!SeDefaultObjectMethod
```

So, with this in mind, we can modify our function rebinding code to target the `SecurityProcedure` for **PsProcessType** (https://learn.microsoft.com/en-us/windows-hardware/drivers/ddi/wdm/nf-wdm-obreferenceobjectbyhandle) and **DbgkDebugObjectType (https://github.com/mic101/windows/blob/master/WRK-v1.2/base/ntos/dbgk/dbgkobj.c)**; or the shorthand by comparing the type names to `Process` and `DebugObject`.

```
1.  uint64_t index = 2;
2.  for ( object_type* obj = get_object( index ); obj != nullptr; obj = get_object( ++index )
    )
```

```
 {
     auto ob_name = xstd::anystr( 0x100, 0x00 ).to_ansi( &obj->name );
     OUT_INF( "%llu, 0x%p, %s", index, obj, ob_name.c_str() );

     const auto ob_type_tracker = reinterpret_cast< ob_type_tracking_data* >( allocator->alloc( sizeof( ob_type_tracking_data ) ) );

     if ( !ob_type_tracker )
         return false;

     ob_type_tracker->ob_type_name = ob_name;
     ob_type_tracker->target_object = obj;
     ob_type_tracker->o_open_procedure = obj->type_info.open_procedure;
     ob_type_tracker->o_close_procedure = obj->type_info.close_procedure;
     ob_type_tracker->o_delete_procedure = obj->type_info.delete_procedure;
     ob_type_tracker->o_security_procedure = obj->type_info.security_procedure;
     ob_type_tracker->o_okay_to_close_procedure = obj->type_info.okay_to_close_procedure;
     ob_type_tracker->o_query_name_procedure = obj->type_info.query_name_procedure;
     ob_type_tracker->o_parse_procedure_detail.parse_procedure = obj->type_info.parse_procedure_detail.parse_procedure;

     g_ob_type_tracking_data.push_back( ob_type_tracker );

     if ( ob_name.compare( "Process" ) || ob_name.compare( "DebugObject" ) )
     {
         _InterlockedExchangePointer(
             reinterpret_cast< void** >( &obj->type_info.security_procedure ),
             reinterpret_cast< void* >( generic_security_procedure )
         );
     }
 }
```

The last thing to be done is to modify our `generic_security_procedure` shared earlier to handle the case for `DebugObject` type. If you want to disable the ability to debug system-wide then you can simply return **STATUS_DEBUG_ATTACH_FAILED** (https://learn.microsoft.com/en-us/openspecs/windows_protocols/ms-erref/596a1078-e883-4972-9bbc-49e60bebca55) in the case for `DebugObject` . If you want to deny debugging *of* a specific process or *by* a specific application you'll have to do a little extra. The results of denying **x64dbg** (https://x64dbg.com/) and **WinDbg** (https://learn.microsoft.com/en-us/windows-hardware/drivers/debugger/) specifically are given below.

Parting Notes

The methods and use cases described in this article are not the end-all-be-all. There are **71 object types** (https://gist.github.com/daaximus/2cd107e3c5077d569ca88f15ad287371), all of which will have various configurations and flexibility. Some have been documented more than others, but putting something to practice for fun is always entertaining. This article was not intended to be comprehensive concerning the details of all the internals, as you could write a book about those. All the examples are somewhat trivial

but, when extended, have some very interesting use cases, especially if you consider the contents of some of my other articles involving manipulating the stack to get control of other things that may otherwise be protected.

If you're interested in the details of the various subsystems, I strongly recommend checking out **Windows Internals 7th Edition Part 1 & 2** (https://www.microsoftpressstore.com/store/windows-internals-part-2-9780135462331) and **Pavel Yosifovich's** (https://twitter.com/zodiacon) **Windows Kernel Programming** (https://leanpub.com/windowskernelprogrammingsecondedition).

As always, I hope you enjoyed the article despite the chaos and disorganization. If you have questions, comments, or feedback or just want to chat, feel free to reach out to me **@daaximus** (https://twitter.com/daaximus).

> 📢 **Callback to earlier...**
>
> For those who might've still had it in the back of their mind, you'll find references to `QueryName` procedures in pretty much any function that tries to capture object information/name information. These include `EtwpEnumerateAddressSpace`, `MmGetFileNameForAddress`, `NtQueryObject` (naturally), `ObQueryNameString`, etc. For `Parse` procedures, you might've already guessed that anything performing a lookup or trying to open some object by name will reference it. The primary caller for these procedures is `ObpLookupObjectByName`.

All Object Types and Procedures Dumped

```
1.  [2/16/2024 5:59:58 PM] [INFO] [2] Type
2.  [2/16/2024 5:59:58 PM] [INFO]   TypeObject->OpenProcedure =
    0000000000000000
3.  [2/16/2024 5:59:58 PM] [INFO]   TypeObject->CloseProcedure =
    0000000000000000
4.  [2/16/2024 5:59:58 PM] [INFO]   TypeObject->DeleteProcedure =
    0000000000000000
```

```
[2/16/2024 5:59:58 PM] [INFO]   TypeObject->DumpProcedure =
0000000000000000
[2/16/2024 5:59:58 PM] [INFO]   TypeObject->OkayToCloseProcedure =
0000000000000000
[2/16/2024 5:59:58 PM] [INFO]   TypeObject->ParseProcedure =
0000000000000000
[2/16/2024 5:59:58 PM] [INFO]   TypeObject->SecurityProcedure =
SeDefaultObjectMethod
[2/16/2024 5:59:58 PM] [INFO] [3] Directory
[2/16/2024 5:59:58 PM] [INFO]   DirectoryObject->OpenProcedure =
0000000000000000
[2/16/2024 5:59:58 PM] [INFO]   DirectoryObject->CloseProcedure =
ObpCloseDirectoryObject
[2/16/2024 5:59:58 PM] [INFO]   DirectoryObject->DeleteProcedure =
ObpDeleteDirectoryObject
[2/16/2024 5:59:59 PM] [INFO]   DirectoryObject->DumpProcedure =
0000000000000000
[2/16/2024 5:59:59 PM] [INFO]   DirectoryObject->OkayToCloseProcedure =
0000000000000000
[2/16/2024 5:59:59 PM] [INFO]   DirectoryObject->ParseProcedure =
0000000000000000
[2/16/2024 5:59:59 PM] [INFO]   DirectoryObject->SecurityProcedure =
SeDefaultObjectMethod
[2/16/2024 5:59:59 PM] [INFO] [4] SymbolicLink
[2/16/2024 5:59:59 PM] [INFO]   SymbolicLinkObject->OpenProcedure =
0000000000000000
[2/16/2024 5:59:59 PM] [INFO]   SymbolicLinkObject->CloseProcedure =
0000000000000000
[2/16/2024 5:59:59 PM] [INFO]   SymbolicLinkObject->DeleteProcedure =
ObpDeleteSymbolicLink
[2/16/2024 5:59:59 PM] [INFO]   SymbolicLinkObject->DumpProcedure =
0000000000000000
[2/16/2024 5:59:59 PM] [INFO]   SymbolicLinkObject->OkayToCloseProcedure
= 0000000000000000
[2/16/2024 5:59:59 PM] [INFO]   SymbolicLinkObject->ParseProcedure =
ObpParseSymbolicLinkEx
```

```
[2/16/2024 5:59:59 PM] [INFO]   SymbolicLinkObject->SecurityProcedure =
SeDefaultObjectMethod
[2/16/2024 5:59:59 PM] [INFO] [5] Token
[2/16/2024 5:59:59 PM] [INFO]   TokenObject->OpenProcedure =
0000000000000000
[2/16/2024 5:59:59 PM] [INFO]   TokenObject->CloseProcedure =
0000000000000000
[2/16/2024 5:59:59 PM] [INFO]   TokenObject->DeleteProcedure =
SepTokenDeleteMethod
[2/16/2024 5:59:59 PM] [INFO]   TokenObject->DumpProcedure =
0000000000000000
[2/16/2024 5:59:59 PM] [INFO]   TokenObject->OkayToCloseProcedure =
0000000000000000
[2/16/2024 5:59:59 PM] [INFO]   TokenObject->ParseProcedure =
0000000000000000
[2/16/2024 5:59:59 PM] [INFO]   TokenObject->SecurityProcedure =
SeDefaultObjectMethod
[2/16/2024 5:59:59 PM] [INFO] [6] Job
[2/16/2024 5:59:59 PM] [INFO]   JobObject->OpenProcedure =
0000000000000000
[2/16/2024 5:59:59 PM] [INFO]   JobObject->CloseProcedure = PspJobClose
[2/16/2024 5:59:59 PM] [INFO]   JobObject->DeleteProcedure =
PspJobDelete
[2/16/2024 5:59:59 PM] [INFO]   JobObject->DumpProcedure =
0000000000000000
[2/16/2024 5:59:59 PM] [INFO]   JobObject->OkayToCloseProcedure =
0000000000000000
[2/16/2024 5:59:59 PM] [INFO]   JobObject->ParseProcedure =
0000000000000000
[2/16/2024 5:59:59 PM] [INFO]   JobObject->SecurityProcedure =
SeDefaultObjectMethod
[2/16/2024 5:59:59 PM] [INFO] [7] Process
[2/16/2024 5:59:59 PM] [INFO]   ProcessObject->OpenProcedure =
PspProcessOpen
[2/16/2024 5:59:59 PM] [INFO]   ProcessObject->CloseProcedure =
PspProcessClose
```

```
[2/16/2024 5:59:59 PM] [INFO]   ProcessObject->DeleteProcedure =
PspProcessDelete
[2/16/2024 5:59:59 PM] [INFO]   ProcessObject->DumpProcedure =
0000000000000000
[2/16/2024 5:59:59 PM] [INFO]   ProcessObject->OkayToCloseProcedure =
0000000000000000
[2/16/2024 5:59:59 PM] [INFO]   ProcessObject->ParseProcedure =
0000000000000000
[2/16/2024 5:59:59 PM] [INFO]   ProcessObject->SecurityProcedure =
SeDefaultObjectMethod
[2/16/2024 5:59:59 PM] [INFO] [8] Thread
[2/16/2024 5:59:59 PM] [INFO]   ThreadObject->OpenProcedure =
PspThreadOpen
[2/16/2024 5:59:59 PM] [INFO]   ThreadObject->CloseProcedure =
0000000000000000
[2/16/2024 5:59:59 PM] [INFO]   ThreadObject->DeleteProcedure =
PspThreadDelete
[2/16/2024 5:59:59 PM] [INFO]   ThreadObject->DumpProcedure =
0000000000000000
[2/16/2024 5:59:59 PM] [INFO]   ThreadObject->OkayToCloseProcedure =
0000000000000000
[2/16/2024 5:59:59 PM] [INFO]   ThreadObject->ParseProcedure =
0000000000000000
[2/16/2024 5:59:59 PM] [INFO]   ThreadObject->SecurityProcedure =
SeDefaultObjectMethod
[2/16/2024 5:59:59 PM] [INFO] [9] Partition
[2/16/2024 5:59:59 PM] [INFO]   PartitionObject->OpenProcedure =
PspOpenPartitionHandle
[2/16/2024 5:59:59 PM] [INFO]   PartitionObject->CloseProcedure =
PspClosePartitionHandle
[2/16/2024 5:59:59 PM] [INFO]   PartitionObject->DeleteProcedure =
PspDeletePartition
[2/16/2024 5:59:59 PM] [INFO]   PartitionObject->DumpProcedure =
0000000000000000
[2/16/2024 5:59:59 PM] [INFO]   PartitionObject->OkayToCloseProcedure =
0000000000000000
```

```
[2/16/2024 5:59:59 PM] [INFO]   PartitionObject->ParseProcedure =
0000000000000000
[2/16/2024 5:59:59 PM] [INFO]   PartitionObject->SecurityProcedure =
SeDefaultObjectMethod
[2/16/2024 5:59:59 PM] [INFO] [10] UserApcReserve
[2/16/2024 5:59:59 PM] [INFO]   UserApcReserveObject->OpenProcedure =
0000000000000000
[2/16/2024 5:59:59 PM] [INFO]   UserApcReserveObject->CloseProcedure =
0000000000000000
[2/16/2024 5:59:59 PM] [INFO]   UserApcReserveObject->DeleteProcedure =
0000000000000000
[2/16/2024 5:59:59 PM] [INFO]   UserApcReserveObject->DumpProcedure =
0000000000000000
[2/16/2024 5:59:59 PM] [INFO]   UserApcReserveObject-
>OkayToCloseProcedure = 0000000000000000
[2/16/2024 5:59:59 PM] [INFO]   UserApcReserveObject->ParseProcedure =
0000000000000000
[2/16/2024 5:59:59 PM] [INFO]   UserApcReserveObject->SecurityProcedure
= SeDefaultObjectMethod
[2/16/2024 5:59:59 PM] [INFO] [11] IoCompletionReserve
[2/16/2024 5:59:59 PM] [INFO]   IoCompletionReserveObject->OpenProcedure
= 0000000000000000
[2/16/2024 5:59:59 PM] [INFO]   IoCompletionReserveObject-
>CloseProcedure = 0000000000000000
[2/16/2024 5:59:59 PM] [INFO]   IoCompletionReserveObject-
>DeleteProcedure = 0000000000000000
[2/16/2024 5:59:59 PM] [INFO]   IoCompletionReserveObject->DumpProcedure
= 0000000000000000
[2/16/2024 5:59:59 PM] [INFO]   IoCompletionReserveObject-
>OkayToCloseProcedure = 0000000000000000
[2/16/2024 5:59:59 PM] [INFO]   IoCompletionReserveObject-
>ParseProcedure = 0000000000000000
[2/16/2024 5:59:59 PM] [INFO]   IoCompletionReserveObject-
>SecurityProcedure = SeDefaultObjectMethod
[2/16/2024 5:59:59 PM] [INFO] [12] ActivityReference
```

```
[2/16/2024 5:59:59 PM] [INFO]   ActivityReferenceObject->OpenProcedure =
0000000000000000
[2/16/2024 5:59:59 PM] [INFO]   ActivityReferenceObject->CloseProcedure
= PspCloseActivityReference
[2/16/2024 5:59:59 PM] [INFO]   ActivityReferenceObject->DeleteProcedure
= 0000000000000000
[2/16/2024 5:59:59 PM] [INFO]   ActivityReferenceObject->DumpProcedure =
0000000000000000
[2/16/2024 5:59:59 PM] [INFO]   ActivityReferenceObject-
>OkayToCloseProcedure = 0000000000000000
[2/16/2024 5:59:59 PM] [INFO]   ActivityReferenceObject->ParseProcedure
= 0000000000000000
[2/16/2024 5:59:59 PM] [INFO]   ActivityReferenceObject-
>SecurityProcedure = SeDefaultObjectMethod
[2/16/2024 5:59:59 PM] [INFO] [13] ProcessStateChange
[2/16/2024 5:59:59 PM] [INFO]   ProcessStateChangeObject->OpenProcedure
= 0000000000000000
[2/16/2024 5:59:59 PM] [INFO]   ProcessStateChangeObject->CloseProcedure
= 0000000000000000
[2/16/2024 5:59:59 PM] [INFO]   ProcessStateChangeObject-
>DeleteProcedure = PspDeleteProcessStateChange
[2/16/2024 5:59:59 PM] [INFO]   ProcessStateChangeObject->DumpProcedure
= 0000000000000000
[2/16/2024 5:59:59 PM] [INFO]   ProcessStateChangeObject-
>OkayToCloseProcedure = 0000000000000000
[2/16/2024 5:59:59 PM] [INFO]   ProcessStateChangeObject->ParseProcedure
= 0000000000000000
[2/16/2024 5:59:59 PM] [INFO]   ProcessStateChangeObject-
>SecurityProcedure = SeDefaultObjectMethod
[2/16/2024 5:59:59 PM] [INFO] [14] ThreadStateChange
[2/16/2024 5:59:59 PM] [INFO]   ThreadStateChangeObject->OpenProcedure =
0000000000000000
[2/16/2024 5:59:59 PM] [INFO]   ThreadStateChangeObject->CloseProcedure
= 0000000000000000
[2/16/2024 5:59:59 PM] [INFO]   ThreadStateChangeObject->DeleteProcedure
= PspDeleteThreadStateChange
```

```
101.  [2/16/2024 5:59:59 PM] [INFO]   ThreadStateChangeObject->DumpProcedure =
      0000000000000000
102.  [2/16/2024 5:59:59 PM] [INFO]   ThreadStateChangeObject-
      >OkayToCloseProcedure = 0000000000000000
103.  [2/16/2024 5:59:59 PM] [INFO]   ThreadStateChangeObject->ParseProcedure
      = 0000000000000000
104.  [2/16/2024 5:59:59 PM] [INFO]   ThreadStateChangeObject-
      >SecurityProcedure = SeDefaultObjectMethod
105.  [2/16/2024 5:59:59 PM] [INFO] [15] CpuPartition
106.  [2/16/2024 5:59:59 PM] [INFO]   CpuPartitionObject->OpenProcedure =
      0000000000000000
107.  [2/16/2024 5:59:59 PM] [INFO]   CpuPartitionObject->CloseProcedure =
      0000000000000000
108.  [2/16/2024 5:59:59 PM] [INFO]   CpuPartitionObject->DeleteProcedure =
      PspDeleteCpuPartition
109.  [2/16/2024 5:59:59 PM] [INFO]   CpuPartitionObject->DumpProcedure =
      0000000000000000
110.  [2/16/2024 5:59:59 PM] [INFO]   CpuPartitionObject->OkayToCloseProcedure
      = 0000000000000000
111.  [2/16/2024 5:59:59 PM] [INFO]   CpuPartitionObject->ParseProcedure =
      0000000000000000
112.  [2/16/2024 5:59:59 PM] [INFO]   CpuPartitionObject->SecurityProcedure =
      SeDefaultObjectMethod
113.  [2/16/2024 5:59:59 PM] [INFO] [16] PsSiloContextPaged
114.  [2/16/2024 5:59:59 PM] [INFO]   PsSiloContextPagedObject->OpenProcedure
      = 0000000000000000
115.  [2/16/2024 5:59:59 PM] [INFO]   PsSiloContextPagedObject->CloseProcedure
      = 0000000000000000
116.  [2/16/2024 5:59:59 PM] [INFO]   PsSiloContextPagedObject-
      >DeleteProcedure = PspDeleteSiloContext
117.  [2/16/2024 5:59:59 PM] [INFO]   PsSiloContextPagedObject->DumpProcedure
      = 0000000000000000
118.  [2/16/2024 5:59:59 PM] [INFO]   PsSiloContextPagedObject-
      >OkayToCloseProcedure = 0000000000000000
119.  [2/16/2024 5:59:59 PM] [INFO]   PsSiloContextPagedObject->ParseProcedure
      = 0000000000000000
```

```
120.  [2/16/2024 5:59:59 PM] [INFO]   PsSiloContextPagedObject->SecurityProcedure = SeDefaultObjectMethod
121.  [2/16/2024 5:59:59 PM] [INFO] [17] PsSiloContextNonPaged
122.  [2/16/2024 5:59:59 PM] [INFO]   PsSiloContextNonPagedObject->OpenProcedure = 0000000000000000
123.  [2/16/2024 5:59:59 PM] [INFO]   PsSiloContextNonPagedObject->CloseProcedure = 0000000000000000
124.  [2/16/2024 5:59:59 PM] [INFO]   PsSiloContextNonPagedObject->DeleteProcedure = PspDeleteSiloContext
125.  [2/16/2024 5:59:59 PM] [INFO]   PsSiloContextNonPagedObject->DumpProcedure = 0000000000000000
126.  [2/16/2024 5:59:59 PM] [INFO]   PsSiloContextNonPagedObject->OkayToCloseProcedure = 0000000000000000
127.  [2/16/2024 5:59:59 PM] [INFO]   PsSiloContextNonPagedObject->ParseProcedure = 0000000000000000
128.  [2/16/2024 5:59:59 PM] [INFO]   PsSiloContextNonPagedObject->SecurityProcedure = SeDefaultObjectMethod
129.  [2/16/2024 5:59:59 PM] [INFO] [18] DebugObject
130.  [2/16/2024 5:59:59 PM] [INFO]   DebugObjectObject->OpenProcedure = 0000000000000000
131.  [2/16/2024 5:59:59 PM] [INFO]   DebugObjectObject->CloseProcedure = DbgkpCloseObject
132.  [2/16/2024 5:59:59 PM] [INFO]   DebugObjectObject->DeleteProcedure = AlpcConnectionCleanupProcedure
133.  [2/16/2024 5:59:59 PM] [INFO]   DebugObjectObject->DumpProcedure = 0000000000000000
134.  [2/16/2024 5:59:59 PM] [INFO]   DebugObjectObject->OkayToCloseProcedure = 0000000000000000
135.  [2/16/2024 5:59:59 PM] [INFO]   DebugObjectObject->ParseProcedure = 0000000000000000
136.  [2/16/2024 5:59:59 PM] [INFO]   DebugObjectObject->SecurityProcedure = SeDefaultObjectMethod
137.  [2/16/2024 5:59:59 PM] [INFO] [19] Event
138.  [2/16/2024 5:59:59 PM] [INFO]   EventObject->OpenProcedure = 0000000000000000
```

```
[2/16/2024 5:59:59 PM] [INFO]   EventObject->CloseProcedure =
0000000000000000
[2/16/2024 5:59:59 PM] [INFO]   EventObject->DeleteProcedure =
0000000000000000
[2/16/2024 5:59:59 PM] [INFO]   EventObject->DumpProcedure =
0000000000000000
[2/16/2024 5:59:59 PM] [INFO]   EventObject->OkayToCloseProcedure =
0000000000000000
[2/16/2024 5:59:59 PM] [INFO]   EventObject->ParseProcedure =
0000000000000000
[2/16/2024 5:59:59 PM] [INFO]   EventObject->SecurityProcedure =
SeDefaultObjectMethod
[2/16/2024 5:59:59 PM] [INFO] [20] Mutant
[2/16/2024 5:59:59 PM] [INFO]   MutantObject->OpenProcedure =
0000000000000000
[2/16/2024 5:59:59 PM] [INFO]   MutantObject->CloseProcedure =
0000000000000000
[2/16/2024 5:59:59 PM] [INFO]   MutantObject->DeleteProcedure =
ExpDeleteMutant
[2/16/2024 5:59:59 PM] [INFO]   MutantObject->DumpProcedure =
0000000000000000
[2/16/2024 5:59:59 PM] [INFO]   MutantObject->OkayToCloseProcedure =
0000000000000000
[2/16/2024 5:59:59 PM] [INFO]   MutantObject->ParseProcedure =
0000000000000000
[2/16/2024 5:59:59 PM] [INFO]   MutantObject->SecurityProcedure =
SeDefaultObjectMethod
[2/16/2024 5:59:59 PM] [INFO] [21] Callback
[2/16/2024 5:59:59 PM] [INFO]   CallbackObject->OpenProcedure =
0000000000000000
[2/16/2024 5:59:59 PM] [INFO]   CallbackObject->CloseProcedure =
0000000000000000
[2/16/2024 6:00:00 PM] [INFO]   CallbackObject->DeleteProcedure =
ExpDeleteCallback
[2/16/2024 6:00:00 PM] [INFO]   CallbackObject->DumpProcedure =
0000000000000000
```

```
158.  [2/16/2024 6:00:00 PM] [INFO]   CallbackObject->OkayToCloseProcedure =
      0000000000000000
159.  [2/16/2024 6:00:00 PM] [INFO]   CallbackObject->ParseProcedure =
      0000000000000000
160.  [2/16/2024 6:00:00 PM] [INFO]   CallbackObject->SecurityProcedure =
      SeDefaultObjectMethod
161.  [2/16/2024 6:00:00 PM] [INFO] [22] Semaphore
162.  [2/16/2024 6:00:00 PM] [INFO]   SemaphoreObject->OpenProcedure =
      0000000000000000
163.  [2/16/2024 6:00:00 PM] [INFO]   SemaphoreObject->CloseProcedure =
      0000000000000000
164.  [2/16/2024 6:00:00 PM] [INFO]   SemaphoreObject->DeleteProcedure =
      0000000000000000
165.  [2/16/2024 6:00:00 PM] [INFO]   SemaphoreObject->DumpProcedure =
      0000000000000000
166.  [2/16/2024 6:00:00 PM] [INFO]   SemaphoreObject->OkayToCloseProcedure =
      0000000000000000
167.  [2/16/2024 6:00:00 PM] [INFO]   SemaphoreObject->ParseProcedure =
      0000000000000000
168.  [2/16/2024 6:00:00 PM] [INFO]   SemaphoreObject->SecurityProcedure =
      SeDefaultObjectMethod
169.  [2/16/2024 6:00:00 PM] [INFO] [23] Timer
170.  [2/16/2024 6:00:00 PM] [INFO]   TimerObject->OpenProcedure =
      0000000000000000
171.  [2/16/2024 6:00:00 PM] [INFO]   TimerObject->CloseProcedure =
      0000000000000000
172.  [2/16/2024 6:00:00 PM] [INFO]   TimerObject->DeleteProcedure =
      ExpDeleteTimer
173.  [2/16/2024 6:00:00 PM] [INFO]   TimerObject->DumpProcedure =
      0000000000000000
174.  [2/16/2024 6:00:00 PM] [INFO]   TimerObject->OkayToCloseProcedure =
      0000000000000000
175.  [2/16/2024 6:00:00 PM] [INFO]   TimerObject->ParseProcedure =
      0000000000000000
176.  [2/16/2024 6:00:00 PM] [INFO]   TimerObject->SecurityProcedure =
      SeDefaultObjectMethod
```

```
[2/16/2024 6:00:00 PM] [INFO] [24] IRTimer
[2/16/2024 6:00:00 PM] [INFO]   IRTimerObject->OpenProcedure =
0000000000000000
[2/16/2024 6:00:00 PM] [INFO]   IRTimerObject->CloseProcedure =
0000000000000000
[2/16/2024 6:00:00 PM] [INFO]   IRTimerObject->DeleteProcedure =
ExpDeleteTimer2
[2/16/2024 6:00:00 PM] [INFO]   IRTimerObject->DumpProcedure =
0000000000000000
[2/16/2024 6:00:00 PM] [INFO]   IRTimerObject->OkayToCloseProcedure =
0000000000000000
[2/16/2024 6:00:00 PM] [INFO]   IRTimerObject->ParseProcedure =
0000000000000000
[2/16/2024 6:00:00 PM] [INFO]   IRTimerObject->SecurityProcedure =
SeDefaultObjectMethod
[2/16/2024 6:00:00 PM] [INFO] [25] Profile
[2/16/2024 6:00:00 PM] [INFO]   ProfileObject->OpenProcedure =
0000000000000000
[2/16/2024 6:00:00 PM] [INFO]   ProfileObject->CloseProcedure =
0000000000000000
[2/16/2024 6:00:00 PM] [INFO]   ProfileObject->DeleteProcedure =
ExpProfileDelete
[2/16/2024 6:00:00 PM] [INFO]   ProfileObject->DumpProcedure =
0000000000000000
[2/16/2024 6:00:00 PM] [INFO]   ProfileObject->OkayToCloseProcedure =
0000000000000000
[2/16/2024 6:00:00 PM] [INFO]   ProfileObject->ParseProcedure =
0000000000000000
[2/16/2024 6:00:00 PM] [INFO]   ProfileObject->SecurityProcedure =
SeDefaultObjectMethod
[2/16/2024 6:00:00 PM] [INFO] [26] KeyedEvent
[2/16/2024 6:00:00 PM] [INFO]   KeyedEventObject->OpenProcedure =
0000000000000000
[2/16/2024 6:00:00 PM] [INFO]   KeyedEventObject->CloseProcedure =
0000000000000000
```

```
196.  [2/16/2024 6:00:00 PM] [INFO]   KeyedEventObject->DeleteProcedure =
      0000000000000000
197.  [2/16/2024 6:00:00 PM] [INFO]   KeyedEventObject->DumpProcedure =
      0000000000000000
198.  [2/16/2024 6:00:00 PM] [INFO]   KeyedEventObject->OkayToCloseProcedure =
      0000000000000000
199.  [2/16/2024 6:00:00 PM] [INFO]   KeyedEventObject->ParseProcedure =
      0000000000000000
200.  [2/16/2024 6:00:00 PM] [INFO]   KeyedEventObject->SecurityProcedure =
      SeDefaultObjectMethod
201.  [2/16/2024 6:00:00 PM] [INFO] [27] WindowStation
202.  [2/16/2024 6:00:00 PM] [INFO]   WindowStationObject->OpenProcedure =
      ExpWin32OpenProcedure
203.  [2/16/2024 6:00:00 PM] [INFO]   WindowStationObject->CloseProcedure =
      ExpWin32CloseProcedure
204.  [2/16/2024 6:00:00 PM] [INFO]   WindowStationObject->DeleteProcedure =
      ExpWin32DeleteProcedure
205.  [2/16/2024 6:00:00 PM] [INFO]   WindowStationObject->DumpProcedure =
      0000000000000000
206.  [2/16/2024 6:00:00 PM] [INFO]   WindowStationObject-
      >OkayToCloseProcedure = ExpWin32OkayToCloseProcedure
207.  [2/16/2024 6:00:00 PM] [INFO]   WindowStationObject->ParseProcedure =
      ExpWin32ParseProcedure
208.  [2/16/2024 6:00:00 PM] [INFO]   WindowStationObject->SecurityProcedure =
      SeDefaultObjectMethod
209.  [2/16/2024 6:00:00 PM] [INFO] [28] Desktop
210.  [2/16/2024 6:00:00 PM] [INFO]   DesktopObject->OpenProcedure =
      ExpWin32OpenProcedure
211.  [2/16/2024 6:00:00 PM] [INFO]   DesktopObject->CloseProcedure =
      ExpWin32CloseProcedure
212.  [2/16/2024 6:00:00 PM] [INFO]   DesktopObject->DeleteProcedure =
      ExpWin32DeleteProcedure
213.  [2/16/2024 6:00:00 PM] [INFO]   DesktopObject->DumpProcedure =
      0000000000000000
214.  [2/16/2024 6:00:00 PM] [INFO]   DesktopObject->OkayToCloseProcedure =
      ExpWin32OkayToCloseProcedure
```

```
215.  [2/16/2024 6:00:00 PM] [INFO]   DesktopObject->ParseProcedure =
      0000000000000000
216.  [2/16/2024 6:00:00 PM] [INFO]   DesktopObject->SecurityProcedure =
      SeDefaultObjectMethod
217.  [2/16/2024 6:00:00 PM] [INFO] [29] Composition
218.  [2/16/2024 6:00:00 PM] [INFO]   CompositionObject->OpenProcedure =
      ExpWin32OpenProcedure
219.  [2/16/2024 6:00:00 PM] [INFO]   CompositionObject->CloseProcedure =
      ExpWin32CloseProcedure
220.  [2/16/2024 6:00:00 PM] [INFO]   CompositionObject->DeleteProcedure =
      ExpWin32DeleteProcedure
221.  [2/16/2024 6:00:00 PM] [INFO]   CompositionObject->DumpProcedure =
      0000000000000000
222.  [2/16/2024 6:00:00 PM] [INFO]   CompositionObject->OkayToCloseProcedure
      = ExpWin32OkayToCloseProcedure
223.  [2/16/2024 6:00:00 PM] [INFO]   CompositionObject->ParseProcedure =
      0000000000000000
224.  [2/16/2024 6:00:00 PM] [INFO]   CompositionObject->SecurityProcedure =
      SeDefaultObjectMethod
225.  [2/16/2024 6:00:00 PM] [INFO] [30] RawInputManager
226.  [2/16/2024 6:00:00 PM] [INFO]   RawInputManagerObject->OpenProcedure =
      ExpWin32OpenProcedure
227.  [2/16/2024 6:00:00 PM] [INFO]   RawInputManagerObject->CloseProcedure =
      ExpWin32CloseProcedure
228.  [2/16/2024 6:00:00 PM] [INFO]   RawInputManagerObject->DeleteProcedure =
      ExpWin32DeleteProcedure
229.  [2/16/2024 6:00:00 PM] [INFO]   RawInputManagerObject->DumpProcedure =
      0000000000000000
230.  [2/16/2024 6:00:00 PM] [INFO]   RawInputManagerObject-
      >OkayToCloseProcedure = ExpWin32OkayToCloseProcedure
231.  [2/16/2024 6:00:00 PM] [INFO]   RawInputManagerObject->ParseProcedure =
      0000000000000000
232.  [2/16/2024 6:00:00 PM] [INFO]   RawInputManagerObject->SecurityProcedure
      = SeDefaultObjectMethod
233.  [2/16/2024 6:00:00 PM] [INFO] [31] CoreMessaging
```

```
[2/16/2024 6:00:00 PM] [INFO]   CoreMessagingObject->OpenProcedure =
ExpWin32OpenProcedure
[2/16/2024 6:00:00 PM] [INFO]   CoreMessagingObject->CloseProcedure =
ExpWin32CloseProcedure
[2/16/2024 6:00:00 PM] [INFO]   CoreMessagingObject->DeleteProcedure =
ExpWin32DeleteProcedure
[2/16/2024 6:00:00 PM] [INFO]   CoreMessagingObject->DumpProcedure =
0000000000000000
[2/16/2024 6:00:00 PM] [INFO]   CoreMessagingObject-
>OkayToCloseProcedure = ExpWin32OkayToCloseProcedure
[2/16/2024 6:00:00 PM] [INFO]   CoreMessagingObject->ParseProcedure =
0000000000000000
[2/16/2024 6:00:00 PM] [INFO]   CoreMessagingObject->SecurityProcedure =
SeDefaultObjectMethod
[2/16/2024 6:00:00 PM] [INFO] [32] ActivationObject
[2/16/2024 6:00:00 PM] [INFO]   ActivationObjectObject->OpenProcedure =
ExpWin32OpenProcedure
[2/16/2024 6:00:00 PM] [INFO]   ActivationObjectObject->CloseProcedure =
ExpWin32CloseProcedure
[2/16/2024 6:00:00 PM] [INFO]   ActivationObjectObject->DeleteProcedure
= ExpWin32DeleteProcedure
[2/16/2024 6:00:00 PM] [INFO]   ActivationObjectObject->DumpProcedure =
0000000000000000
[2/16/2024 6:00:00 PM] [INFO]   ActivationObjectObject-
>OkayToCloseProcedure = ExpWin32OkayToCloseProcedure
[2/16/2024 6:00:00 PM] [INFO]   ActivationObjectObject->ParseProcedure =
0000000000000000
[2/16/2024 6:00:00 PM] [INFO]   ActivationObjectObject-
>SecurityProcedure = SeDefaultObjectMethod
[2/16/2024 6:00:00 PM] [INFO] [33] TpWorkerFactory
[2/16/2024 6:00:00 PM] [INFO]   TpWorkerFactoryObject->OpenProcedure =
0000000000000000
[2/16/2024 6:00:00 PM] [INFO]   TpWorkerFactoryObject->CloseProcedure =
ExpCloseWorkerFactory
[2/16/2024 6:00:00 PM] [INFO]   TpWorkerFactoryObject->DeleteProcedure =
ExpDeleteWorkerFactory
```

```
[2/16/2024 6:00:00 PM] [INFO]   TpWorkerFactoryObject->DumpProcedure =
0000000000000000
[2/16/2024 6:00:00 PM] [INFO]   TpWorkerFactoryObject-
>OkayToCloseProcedure = 0000000000000000
[2/16/2024 6:00:00 PM] [INFO]   TpWorkerFactoryObject->ParseProcedure =
0000000000000000
[2/16/2024 6:00:00 PM] [INFO]   TpWorkerFactoryObject->SecurityProcedure
= SeDefaultObjectMethod
[2/16/2024 6:00:00 PM] [INFO] [34] Adapter
[2/16/2024 6:00:00 PM] [INFO]   AdapterObject->OpenProcedure =
0000000000000000
[2/16/2024 6:00:00 PM] [INFO]   AdapterObject->CloseProcedure =
0000000000000000
[2/16/2024 6:00:00 PM] [INFO]   AdapterObject->DeleteProcedure =
0000000000000000
[2/16/2024 6:00:00 PM] [INFO]   AdapterObject->DumpProcedure =
0000000000000000
[2/16/2024 6:00:00 PM] [INFO]   AdapterObject->OkayToCloseProcedure =
0000000000000000
[2/16/2024 6:00:00 PM] [INFO]   AdapterObject->ParseProcedure =
0000000000000000
[2/16/2024 6:00:00 PM] [INFO]   AdapterObject->SecurityProcedure =
SeDefaultObjectMethod
[2/16/2024 6:00:00 PM] [INFO] [35] Controller
[2/16/2024 6:00:00 PM] [INFO]   ControllerObject->OpenProcedure =
0000000000000000
[2/16/2024 6:00:00 PM] [INFO]   ControllerObject->CloseProcedure =
0000000000000000
[2/16/2024 6:00:00 PM] [INFO]   ControllerObject->DeleteProcedure =
0000000000000000
[2/16/2024 6:00:00 PM] [INFO]   ControllerObject->DumpProcedure =
0000000000000000
[2/16/2024 6:00:00 PM] [INFO]   ControllerObject->OkayToCloseProcedure =
0000000000000000
[2/16/2024 6:00:00 PM] [INFO]   ControllerObject->ParseProcedure =
0000000000000000
```

```
[2/16/2024 6:00:00 PM] [INFO]   ControllerObject->SecurityProcedure =
SeDefaultObjectMethod
[2/16/2024 6:00:00 PM] [INFO] [36] Device
[2/16/2024 6:00:00 PM] [INFO]   DeviceObject->OpenProcedure =
0000000000000000
[2/16/2024 6:00:00 PM] [INFO]   DeviceObject->CloseProcedure =
0000000000000000
[2/16/2024 6:00:00 PM] [INFO]   DeviceObject->DeleteProcedure =
IopDeleteDevice
[2/16/2024 6:00:00 PM] [INFO]   DeviceObject->DumpProcedure =
0000000000000000
[2/16/2024 6:00:00 PM] [INFO]   DeviceObject->OkayToCloseProcedure =
0000000000000000
[2/16/2024 6:00:00 PM] [INFO]   DeviceObject->ParseProcedure =
IopParseDevice
[2/16/2024 6:00:00 PM] [INFO]   DeviceObject->SecurityProcedure =
IopGetSetSecurityObject
[2/16/2024 6:00:00 PM] [INFO] [37] Driver
[2/16/2024 6:00:00 PM] [INFO]   DriverObject->OpenProcedure =
0000000000000000
[2/16/2024 6:00:00 PM] [INFO]   DriverObject->CloseProcedure =
0000000000000000
[2/16/2024 6:00:00 PM] [INFO]   DriverObject->DeleteProcedure =
IopDeleteDriver
[2/16/2024 6:00:00 PM] [INFO]   DriverObject->DumpProcedure =
0000000000000000
[2/16/2024 6:00:00 PM] [INFO]   DriverObject->OkayToCloseProcedure =
0000000000000000
[2/16/2024 6:00:00 PM] [INFO]   DriverObject->ParseProcedure =
0000000000000000
[2/16/2024 6:00:00 PM] [INFO]   DriverObject->SecurityProcedure =
SeDefaultObjectMethod
[2/16/2024 6:00:00 PM] [INFO] [38] IoCompletion
[2/16/2024 6:00:00 PM] [INFO]   IoCompletionObject->OpenProcedure =
0000000000000000
```

```
[2/16/2024 6:00:00 PM] [INFO]   IoCompletionObject->CloseProcedure =
IopCloseIoCompletion
[2/16/2024 6:00:00 PM] [INFO]   IoCompletionObject->DeleteProcedure =
IopDeleteIoCompletion
[2/16/2024 6:00:00 PM] [INFO]   IoCompletionObject->DumpProcedure =
0000000000000000
[2/16/2024 6:00:00 PM] [INFO]   IoCompletionObject->OkayToCloseProcedure
= 0000000000000000
[2/16/2024 6:00:00 PM] [INFO]   IoCompletionObject->ParseProcedure =
0000000000000000
[2/16/2024 6:00:00 PM] [INFO]   IoCompletionObject->SecurityProcedure =
SeDefaultObjectMethod
[2/16/2024 6:00:00 PM] [INFO] [39] WaitCompletionPacket
[2/16/2024 6:00:00 PM] [INFO]   WaitCompletionPacketObject-
>OpenProcedure = 0000000000000000
[2/16/2024 6:00:00 PM] [INFO]   WaitCompletionPacketObject-
>CloseProcedure = IopCloseWaitCompletionPacket
[2/16/2024 6:00:01 PM] [INFO]   WaitCompletionPacketObject-
>DeleteProcedure = 0000000000000000
[2/16/2024 6:00:01 PM] [INFO]   WaitCompletionPacketObject-
>DumpProcedure = 0000000000000000
[2/16/2024 6:00:01 PM] [INFO]   WaitCompletionPacketObject-
>OkayToCloseProcedure = 0000000000000000
[2/16/2024 6:00:01 PM] [INFO]   WaitCompletionPacketObject-
>ParseProcedure = 0000000000000000
[2/16/2024 6:00:01 PM] [INFO]   WaitCompletionPacketObject-
>SecurityProcedure = SeDefaultObjectMethod
[2/16/2024 6:00:01 PM] [INFO] [40] File
[2/16/2024 6:00:01 PM] [INFO]   FileObject->OpenProcedure =
0000000000000000
[2/16/2024 6:00:01 PM] [INFO]   FileObject->CloseProcedure =
IopCloseFile
[2/16/2024 6:00:01 PM] [INFO]   FileObject->DeleteProcedure =
IopDeleteFile
[2/16/2024 6:00:01 PM] [INFO]   FileObject->DumpProcedure =
0000000000000000
```

```
[2/16/2024 6:00:01 PM] [INFO]   FileObject->OkayToCloseProcedure =
0000000000000000
[2/16/2024 6:00:01 PM] [INFO]   FileObject->ParseProcedure =
IopParseFile
[2/16/2024 6:00:01 PM] [INFO]   FileObject->SecurityProcedure =
FFFFF800487A78D0
[2/16/2024 6:00:01 PM] [INFO] [41] IoRing
[2/16/2024 6:00:01 PM] [INFO]   IoRingObject->OpenProcedure =
EtwpOpenRealTimeConnectionObject
[2/16/2024 6:00:01 PM] [INFO]   IoRingObject->CloseProcedure =
IopCloseIoRing
[2/16/2024 6:00:01 PM] [INFO]   IoRingObject->DeleteProcedure =
IopDeleteIoRing
[2/16/2024 6:00:01 PM] [INFO]   IoRingObject->DumpProcedure =
0000000000000000
[2/16/2024 6:00:01 PM] [INFO]   IoRingObject->OkayToCloseProcedure =
0000000000000000
[2/16/2024 6:00:01 PM] [INFO]   IoRingObject->ParseProcedure =
0000000000000000
[2/16/2024 6:00:01 PM] [INFO]   IoRingObject->SecurityProcedure =
SeDefaultObjectMethod
[2/16/2024 6:00:01 PM] [INFO] [42] TmTm
[2/16/2024 6:00:01 PM] [INFO]   TmTmObject->OpenProcedure = DllUnload
[2/16/2024 6:00:01 PM] [INFO]   TmTmObject->CloseProcedure =
TmpCloseTransactionManager
[2/16/2024 6:00:01 PM] [INFO]   TmTmObject->DeleteProcedure =
TmpDeleteTransactionManager
[2/16/2024 6:00:01 PM] [INFO]   TmTmObject->DumpProcedure =
0000000000000000
[2/16/2024 6:00:01 PM] [INFO]   TmTmObject->OkayToCloseProcedure =
0000000000000000
[2/16/2024 6:00:01 PM] [INFO]   TmTmObject->ParseProcedure =
0000000000000000
[2/16/2024 6:00:01 PM] [INFO]   TmTmObject->SecurityProcedure =
SeDefaultObjectMethod
[2/16/2024 6:00:01 PM] [INFO] [43] TmTx
```

```
330.  [2/16/2024 6:00:01 PM] [INFO]   TmTxObject->OpenProcedure =
      0000000000000000
331.  [2/16/2024 6:00:01 PM] [INFO]   TmTxObject->CloseProcedure =
      TmpCloseTransaction
332.  [2/16/2024 6:00:01 PM] [INFO]   TmTxObject->DeleteProcedure =
      TmpDeleteTransaction
333.  [2/16/2024 6:00:01 PM] [INFO]   TmTxObject->DumpProcedure =
      0000000000000000
334.  [2/16/2024 6:00:01 PM] [INFO]   TmTxObject->OkayToCloseProcedure =
      0000000000000000
335.  [2/16/2024 6:00:01 PM] [INFO]   TmTxObject->ParseProcedure =
      0000000000000000
336.  [2/16/2024 6:00:01 PM] [INFO]   TmTxObject->SecurityProcedure =
      SeDefaultObjectMethod
337.  [2/16/2024 6:00:01 PM] [INFO] [44] TmRm
338.  [2/16/2024 6:00:01 PM] [INFO]   TmRmObject->OpenProcedure =
      TmpOpenResourceManager
339.  [2/16/2024 6:00:01 PM] [INFO]   TmRmObject->CloseProcedure =
      TmpCloseResourceManager
340.  [2/16/2024 6:00:01 PM] [INFO]   TmRmObject->DeleteProcedure =
      TmpDeleteResourceManager
341.  [2/16/2024 6:00:01 PM] [INFO]   TmRmObject->DumpProcedure =
      0000000000000000
342.  [2/16/2024 6:00:01 PM] [INFO]   TmRmObject->OkayToCloseProcedure =
      0000000000000000
343.  [2/16/2024 6:00:01 PM] [INFO]   TmRmObject->ParseProcedure =
      0000000000000000
344.  [2/16/2024 6:00:01 PM] [INFO]   TmRmObject->SecurityProcedure =
      SeDefaultObjectMethod
345.  [2/16/2024 6:00:01 PM] [INFO] [45] TmEn
346.  [2/16/2024 6:00:01 PM] [INFO]   TmEnObject->OpenProcedure =
      0000000000000000
347.  [2/16/2024 6:00:01 PM] [INFO]   TmEnObject->CloseProcedure =
      TmpCloseEnlistment
348.  [2/16/2024 6:00:01 PM] [INFO]   TmEnObject->DeleteProcedure =
      TmpDeleteEnlistment
```

```
[2/16/2024 6:00:01 PM] [INFO]   TmEnObject->DumpProcedure =
0000000000000000
[2/16/2024 6:00:01 PM] [INFO]   TmEnObject->OkayToCloseProcedure =
0000000000000000
[2/16/2024 6:00:01 PM] [INFO]   TmEnObject->ParseProcedure =
0000000000000000
[2/16/2024 6:00:01 PM] [INFO]   TmEnObject->SecurityProcedure =
SeDefaultObjectMethod
[2/16/2024 6:00:01 PM] [INFO] [46] Section
[2/16/2024 6:00:01 PM] [INFO]   SectionObject->OpenProcedure =
MiSectionOpen
[2/16/2024 6:00:01 PM] [INFO]   SectionObject->CloseProcedure =
MiSectionClose
[2/16/2024 6:00:01 PM] [INFO]   SectionObject->DeleteProcedure =
MiSectionDelete
[2/16/2024 6:00:01 PM] [INFO]   SectionObject->DumpProcedure =
0000000000000000
[2/16/2024 6:00:01 PM] [INFO]   SectionObject->OkayToCloseProcedure =
0000000000000000
[2/16/2024 6:00:01 PM] [INFO]   SectionObject->ParseProcedure =
0000000000000000
[2/16/2024 6:00:01 PM] [INFO]   SectionObject->SecurityProcedure =
SeDefaultObjectMethod
[2/16/2024 6:00:01 PM] [INFO] [47] Session
[2/16/2024 6:00:01 PM] [INFO]   SessionObject->OpenProcedure =
0000000000000000
[2/16/2024 6:00:01 PM] [INFO]   SessionObject->CloseProcedure =
0000000000000000
[2/16/2024 6:00:01 PM] [INFO]   SessionObject->DeleteProcedure =
MiSessionObjectDelete
[2/16/2024 6:00:01 PM] [INFO]   SessionObject->DumpProcedure =
0000000000000000
[2/16/2024 6:00:01 PM] [INFO]   SessionObject->OkayToCloseProcedure =
0000000000000000
[2/16/2024 6:00:01 PM] [INFO]   SessionObject->ParseProcedure =
0000000000000000
```

```
368.  [2/16/2024 6:00:01 PM] [INFO]   SessionObject->SecurityProcedure =
      SeDefaultObjectMethod
369.  [2/16/2024 6:00:01 PM] [INFO] [48] Key
370.  [2/16/2024 6:00:01 PM] [INFO]   KeyObject->OpenProcedure =
      0000000000000000
371.  [2/16/2024 6:00:01 PM] [INFO]   KeyObject->CloseProcedure =
      CmpCloseKeyObject
372.  [2/16/2024 6:00:01 PM] [INFO]   KeyObject->DeleteProcedure =
      CmpDeleteKeyObject
373.  [2/16/2024 6:00:01 PM] [INFO]   KeyObject->DumpProcedure =
      0000000000000000
374.  [2/16/2024 6:00:01 PM] [INFO]   KeyObject->OkayToCloseProcedure =
      0000000000000000
375.  [2/16/2024 6:00:01 PM] [INFO]   KeyObject->ParseProcedure = CmpParseKey
376.  [2/16/2024 6:00:01 PM] [INFO]   KeyObject->SecurityProcedure =
      FFFFF800486B0000
377.  [2/16/2024 6:00:01 PM] [INFO] [49] RegistryTransaction
378.  [2/16/2024 6:00:01 PM] [INFO]   RegistryTransactionObject->OpenProcedure
      = 0000000000000000
379.  [2/16/2024 6:00:01 PM] [INFO]   RegistryTransactionObject-
      >CloseProcedure = CmpCloseLightWeightTransaction
380.  [2/16/2024 6:00:01 PM] [INFO]   RegistryTransactionObject-
      >DeleteProcedure = CmpDeleteLightWeightTransaction
381.  [2/16/2024 6:00:01 PM] [INFO]   RegistryTransactionObject->DumpProcedure
      = 0000000000000000
382.  [2/16/2024 6:00:01 PM] [INFO]   RegistryTransactionObject-
      >OkayToCloseProcedure = 0000000000000000
383.  [2/16/2024 6:00:01 PM] [INFO]   RegistryTransactionObject-
      >ParseProcedure = 0000000000000000
384.  [2/16/2024 6:00:01 PM] [INFO]   RegistryTransactionObject-
      >SecurityProcedure = SeDefaultObjectMethod
385.  [2/16/2024 6:00:01 PM] [INFO] [50] DmaAdapter
386.  [2/16/2024 6:00:01 PM] [INFO]   DmaAdapterObject->OpenProcedure =
      0000000000000000
387.  [2/16/2024 6:00:01 PM] [INFO]   DmaAdapterObject->CloseProcedure =
      0000000000000000
```

```
388.  [2/16/2024 6:00:01 PM] [INFO]   DmaAdapterObject->DeleteProcedure =
      HalpDmaFreeChildAdapter
389.  [2/16/2024 6:00:01 PM] [INFO]   DmaAdapterObject->DumpProcedure =
      0000000000000000
390.  [2/16/2024 6:00:01 PM] [INFO]   DmaAdapterObject->OkayToCloseProcedure =
      0000000000000000
391.  [2/16/2024 6:00:01 PM] [INFO]   DmaAdapterObject->ParseProcedure =
      0000000000000000
392.  [2/16/2024 6:00:01 PM] [INFO]   DmaAdapterObject->SecurityProcedure =
      SeDefaultObjectMethod
393.  [2/16/2024 6:00:01 PM] [INFO] [51] ALPC Port
394.  [2/16/2024 6:00:01 PM] [INFO]   ALPC PortObject->OpenProcedure =
      AlpcpOpenPort
395.  [2/16/2024 6:00:01 PM] [INFO]   ALPC PortObject->CloseProcedure =
      AlpcpClosePort
396.  [2/16/2024 6:00:01 PM] [INFO]   ALPC PortObject->DeleteProcedure =
      AlpcpDeletePort
397.  [2/16/2024 6:00:01 PM] [INFO]   ALPC PortObject->DumpProcedure =
      0000000000000000
398.  [2/16/2024 6:00:01 PM] [INFO]   ALPC PortObject->OkayToCloseProcedure =
      0000000000000000
399.  [2/16/2024 6:00:01 PM] [INFO]   ALPC PortObject->ParseProcedure =
      0000000000000000
400.  [2/16/2024 6:00:01 PM] [INFO]   ALPC PortObject->SecurityProcedure =
      SeDefaultObjectMethod
401.  [2/16/2024 6:00:01 PM] [INFO] [52] EnergyTracker
402.  [2/16/2024 6:00:01 PM] [INFO]   EnergyTrackerObject->OpenProcedure =
      0000000000000000
403.  [2/16/2024 6:00:01 PM] [INFO]   EnergyTrackerObject->CloseProcedure =
      PopEtEnergyTrackerClose
404.  [2/16/2024 6:00:01 PM] [INFO]   EnergyTrackerObject->DeleteProcedure =
      PopEtEnergyTrackerDelete
405.  [2/16/2024 6:00:01 PM] [INFO]   EnergyTrackerObject->DumpProcedure =
      0000000000000000
406.  [2/16/2024 6:00:01 PM] [INFO]   EnergyTrackerObject-
      >OkayToCloseProcedure = 0000000000000000
```

```
[2/16/2024 6:00:01 PM] [INFO]   EnergyTrackerObject->ParseProcedure =
0000000000000000
[2/16/2024 6:00:01 PM] [INFO]   EnergyTrackerObject->SecurityProcedure =
SeDefaultObjectMethod
[2/16/2024 6:00:01 PM] [INFO] [53] PowerRequest
[2/16/2024 6:00:01 PM] [INFO]   PowerRequestObject->OpenProcedure =
0000000000000000
[2/16/2024 6:00:01 PM] [INFO]   PowerRequestObject->CloseProcedure =
PopPowerRequestClose
[2/16/2024 6:00:01 PM] [INFO]   PowerRequestObject->DeleteProcedure =
PopPowerRequestDelete
[2/16/2024 6:00:01 PM] [INFO]   PowerRequestObject->DumpProcedure =
0000000000000000
[2/16/2024 6:00:01 PM] [INFO]   PowerRequestObject->OkayToCloseProcedure
= 0000000000000000
[2/16/2024 6:00:01 PM] [INFO]   PowerRequestObject->ParseProcedure =
0000000000000000
[2/16/2024 6:00:01 PM] [INFO]   PowerRequestObject->SecurityProcedure =
SeDefaultObjectMethod
[2/16/2024 6:00:01 PM] [INFO] [54] WmiGuid
[2/16/2024 6:00:01 PM] [INFO]   WmiGuidObject->OpenProcedure =
0000000000000000
[2/16/2024 6:00:01 PM] [INFO]   WmiGuidObject->CloseProcedure =
0000000000000000
[2/16/2024 6:00:01 PM] [INFO]   WmiGuidObject->DeleteProcedure =
WmipDeleteMethod
[2/16/2024 6:00:01 PM] [INFO]   WmiGuidObject->DumpProcedure =
0000000000000000
[2/16/2024 6:00:01 PM] [INFO]   WmiGuidObject->OkayToCloseProcedure =
0000000000000000
[2/16/2024 6:00:01 PM] [INFO]   WmiGuidObject->ParseProcedure =
0000000000000000
[2/16/2024 6:00:01 PM] [INFO]   WmiGuidObject->SecurityProcedure =
FFFFF800487A73D0
[2/16/2024 6:00:01 PM] [INFO] [55] EtwRegistration
```

```
426.  [2/16/2024 6:00:01 PM] [INFO]   EtwRegistrationObject->OpenProcedure =
      EtwpOpenRealTimeConnectionObject
427.  [2/16/2024 6:00:01 PM] [INFO]   EtwRegistrationObject->CloseProcedure =
      EtwpCloseRegistrationObject
428.  [2/16/2024 6:00:01 PM] [INFO]   EtwRegistrationObject->DeleteProcedure =
      EtwpDeleteRegistrationObject
429.  [2/16/2024 6:00:01 PM] [INFO]   EtwRegistrationObject->DumpProcedure =
      0000000000000000
430.  [2/16/2024 6:00:01 PM] [INFO]   EtwRegistrationObject-
      >OkayToCloseProcedure = 0000000000000000
431.  [2/16/2024 6:00:01 PM] [INFO]   EtwRegistrationObject->ParseProcedure =
      0000000000000000
432.  [2/16/2024 6:00:01 PM] [INFO]   EtwRegistrationObject->SecurityProcedure
      = SeDefaultObjectMethod
433.  [2/16/2024 6:00:01 PM] [INFO] [56] EtwSessionDemuxEntry
434.  [2/16/2024 6:00:01 PM] [INFO]   EtwSessionDemuxEntryObject-
      >OpenProcedure = EtwpOpenRealTimeConnectionObject
435.  [2/16/2024 6:00:01 PM] [INFO]   EtwSessionDemuxEntryObject-
      >CloseProcedure = 0000000000000000
436.  [2/16/2024 6:00:01 PM] [INFO]   EtwSessionDemuxEntryObject-
      >DeleteProcedure = EtwpDeleteSessionDemuxObject
437.  [2/16/2024 6:00:01 PM] [INFO]   EtwSessionDemuxEntryObject-
      >DumpProcedure = 0000000000000000
438.  [2/16/2024 6:00:01 PM] [INFO]   EtwSessionDemuxEntryObject-
      >OkayToCloseProcedure = 0000000000000000
439.  [2/16/2024 6:00:01 PM] [INFO]   EtwSessionDemuxEntryObject-
      >ParseProcedure = 0000000000000000
440.  [2/16/2024 6:00:01 PM] [INFO]   EtwSessionDemuxEntryObject-
      >SecurityProcedure = SeDefaultObjectMethod
441.  [2/16/2024 6:00:01 PM] [INFO] [57] EtwConsumer
442.  [2/16/2024 6:00:01 PM] [INFO]   EtwConsumerObject->OpenProcedure =
      EtwpOpenRealTimeConnectionObject
443.  [2/16/2024 6:00:01 PM] [INFO]   EtwConsumerObject->CloseProcedure =
      EtwpCloseRealTimeConnectionObject
444.  [2/16/2024 6:00:01 PM] [INFO]   EtwConsumerObject->DeleteProcedure =
      EtwpDeleteRealTimeConnectionObject
```

```
445.  [2/16/2024 6:00:01 PM] [INFO]   EtwConsumerObject->DumpProcedure =
      0000000000000000
446.  [2/16/2024 6:00:01 PM] [INFO]   EtwConsumerObject->OkayToCloseProcedure
      = 0000000000000000
447.  [2/16/2024 6:00:02 PM] [INFO]   EtwConsumerObject->ParseProcedure =
      0000000000000000
448.  [2/16/2024 6:00:02 PM] [INFO]   EtwConsumerObject->SecurityProcedure =
      SeDefaultObjectMethod
449.  [2/16/2024 6:00:02 PM] [INFO] [58] CoverageSampler
450.  [2/16/2024 6:00:02 PM] [INFO]   CoverageSamplerObject->OpenProcedure =
      0000000000000000
451.  [2/16/2024 6:00:02 PM] [INFO]   CoverageSamplerObject->CloseProcedure =
      EtwpCoverageSamplerClose
452.  [2/16/2024 6:00:02 PM] [INFO]   CoverageSamplerObject->DeleteProcedure =
      EtwpCoverageSamplerDelete
453.  [2/16/2024 6:00:02 PM] [INFO]   CoverageSamplerObject->DumpProcedure =
      0000000000000000
454.  [2/16/2024 6:00:02 PM] [INFO]   CoverageSamplerObject-
      >OkayToCloseProcedure = 0000000000000000
455.  [2/16/2024 6:00:02 PM] [INFO]   CoverageSamplerObject->ParseProcedure =
      0000000000000000
456.  [2/16/2024 6:00:02 PM] [INFO]   CoverageSamplerObject->SecurityProcedure
      = SeDefaultObjectMethod
457.  [2/16/2024 6:00:02 PM] [INFO] [59] PcwObject
458.  [2/16/2024 6:00:02 PM] [INFO]   PcwObjectObject->OpenProcedure =
      PcwpOpenObject
459.  [2/16/2024 6:00:02 PM] [INFO]   PcwObjectObject->CloseProcedure =
      PcwpCloseObjectHandle
460.  [2/16/2024 6:00:02 PM] [INFO]   PcwObjectObject->DeleteProcedure =
      PcwpDeleteObject
461.  [2/16/2024 6:00:02 PM] [INFO]   PcwObjectObject->DumpProcedure =
      0000000000000000
462.  [2/16/2024 6:00:02 PM] [INFO]   PcwObjectObject->OkayToCloseProcedure =
      0000000000000000
463.  [2/16/2024 6:00:02 PM] [INFO]   PcwObjectObject->ParseProcedure =
      0000000000000000
```

```
464.  [2/16/2024 6:00:02 PM] [INFO]   PcwObjectObject->SecurityProcedure =
      SeDefaultObjectMethod
465.  [2/16/2024 6:00:02 PM] [INFO] [60] FilterConnectionPort
466.  [2/16/2024 6:00:02 PM] [INFO]   FilterConnectionPortObject-
      >OpenProcedure = 0000000000000000
467.  [2/16/2024 6:00:02 PM] [INFO]   FilterConnectionPortObject-
      >CloseProcedure = FltpServerPortClose
468.  [2/16/2024 6:00:02 PM] [INFO]   FilterConnectionPortObject-
      >DeleteProcedure = FltpServerPortDelete
469.  [2/16/2024 6:00:02 PM] [INFO]   FilterConnectionPortObject-
      >DumpProcedure = 0000000000000000
470.  [2/16/2024 6:00:02 PM] [INFO]   FilterConnectionPortObject-
      >OkayToCloseProcedure = 0000000000000000
471.  [2/16/2024 6:00:02 PM] [INFO]   FilterConnectionPortObject-
      >ParseProcedure = 0000000000000000
472.  [2/16/2024 6:00:02 PM] [INFO]   FilterConnectionPortObject-
      >SecurityProcedure = SeDefaultObjectMethod
473.  [2/16/2024 6:00:02 PM] [INFO] [61] FilterCommunicationPort
474.  [2/16/2024 6:00:02 PM] [INFO]   FilterCommunicationPortObject-
      >OpenProcedure = 0000000000000000
475.  [2/16/2024 6:00:02 PM] [INFO]   FilterCommunicationPortObject-
      >CloseProcedure = FltpClientPortClose
476.  [2/16/2024 6:00:02 PM] [INFO]   FilterCommunicationPortObject-
      >DeleteProcedure = FltpClientPortDelete
477.  [2/16/2024 6:00:02 PM] [INFO]   FilterCommunicationPortObject-
      >DumpProcedure = 0000000000000000
478.  [2/16/2024 6:00:02 PM] [INFO]   FilterCommunicationPortObject-
      >OkayToCloseProcedure = 0000000000000000
479.  [2/16/2024 6:00:02 PM] [INFO]   FilterCommunicationPortObject-
      >ParseProcedure = 0000000000000000
480.  [2/16/2024 6:00:02 PM] [INFO]   FilterCommunicationPortObject-
      >SecurityProcedure = SeDefaultObjectMethod
481.  [2/16/2024 6:00:02 PM] [INFO] [62] NdisCmState
482.  [2/16/2024 6:00:02 PM] [INFO]   NdisCmStateObject->OpenProcedure =
      0000000000000000
```

```
483.  [2/16/2024 6:00:02 PM] [INFO]   NdisCmStateObject->CloseProcedure =
      0000000000000000
484.  [2/16/2024 6:00:02 PM] [INFO]   NdisCmStateObject->DeleteProcedure =
      ndisCmDeleteStateObject
485.  [2/16/2024 6:00:02 PM] [INFO]   NdisCmStateObject->DumpProcedure =
      0000000000000000
486.  [2/16/2024 6:00:02 PM] [INFO]   NdisCmStateObject->OkayToCloseProcedure
      = 0000000000000000
487.  [2/16/2024 6:00:02 PM] [INFO]   NdisCmStateObject->ParseProcedure =
      0000000000000000
488.  [2/16/2024 6:00:02 PM] [INFO]   NdisCmStateObject->SecurityProcedure =
      SeDefaultObjectMethod
489.  [2/16/2024 6:00:02 PM] [INFO] [63] DxgkSharedResource
490.  [2/16/2024 6:00:02 PM] [INFO]   DxgkSharedResourceObject->OpenProcedure
      = DxgkObOpenProcedureStub
491.  [2/16/2024 6:00:02 PM] [INFO]   DxgkSharedResourceObject->CloseProcedure
      = 0000000000000000
492.  [2/16/2024 6:00:02 PM] [INFO]   DxgkSharedResourceObject-
      >DeleteProcedure = DxgkSharedAllocationObDeleteProcedure
493.  [2/16/2024 6:00:02 PM] [INFO]   DxgkSharedResourceObject->DumpProcedure
      = 0000000000000000
494.  [2/16/2024 6:00:02 PM] [INFO]   DxgkSharedResourceObject-
      >OkayToCloseProcedure = 0000000000000000
495.  [2/16/2024 6:00:02 PM] [INFO]   DxgkSharedResourceObject->ParseProcedure
      = 0000000000000000
496.  [2/16/2024 6:00:02 PM] [INFO]   DxgkSharedResourceObject-
      >SecurityProcedure = SeDefaultObjectMethod
497.  [2/16/2024 6:00:02 PM] [INFO] [64] DxgkSharedKeyedMutexObject
498.  [2/16/2024 6:00:02 PM] [INFO]   DxgkSharedKeyedMutexObjectObject-
      >OpenProcedure = DxgkObOpenProcedureStub
499.  [2/16/2024 6:00:02 PM] [INFO]   DxgkSharedKeyedMutexObjectObject-
      >CloseProcedure = 0000000000000000
500.  [2/16/2024 6:00:02 PM] [INFO]   DxgkSharedKeyedMutexObjectObject-
      >DeleteProcedure = DxgkSharedKeyedMutexObjectObDeleteProcedure
501.  [2/16/2024 6:00:02 PM] [INFO]   DxgkSharedKeyedMutexObjectObject-
      >DumpProcedure = 0000000000000000
```

```
[2/16/2024 6:00:02 PM] [INFO]   DxgkSharedKeyedMutexObjectObject->OkayToCloseProcedure = 0000000000000000
[2/16/2024 6:00:02 PM] [INFO]   DxgkSharedKeyedMutexObjectObject->ParseProcedure = 0000000000000000
[2/16/2024 6:00:02 PM] [INFO]   DxgkSharedKeyedMutexObjectObject->SecurityProcedure = SeDefaultObjectMethod
[2/16/2024 6:00:02 PM] [INFO] [65] DxgkSharedSyncObject
[2/16/2024 6:00:02 PM] [INFO]   DxgkSharedSyncObjectObject->OpenProcedure = DxgkObOpenProcedureStub
[2/16/2024 6:00:02 PM] [INFO]   DxgkSharedSyncObjectObject->CloseProcedure = 0000000000000000
[2/16/2024 6:00:02 PM] [INFO]   DxgkSharedSyncObjectObject->DeleteProcedure = DxgkSharedSyncObjectObDeleteProcedure
[2/16/2024 6:00:02 PM] [INFO]   DxgkSharedSyncObjectObject->DumpProcedure = 0000000000000000
[2/16/2024 6:00:02 PM] [INFO]   DxgkSharedSyncObjectObject->OkayToCloseProcedure = 0000000000000000
[2/16/2024 6:00:02 PM] [INFO]   DxgkSharedSyncObjectObject->ParseProcedure = 0000000000000000
[2/16/2024 6:00:02 PM] [INFO]   DxgkSharedSyncObjectObject->SecurityProcedure = SeDefaultObjectMethod
[2/16/2024 6:00:02 PM] [INFO] [66] DxgkSharedSwapChainObject ;)
[2/16/2024 6:00:02 PM] [INFO]   DxgkSharedSwapChainObjectObject->OpenProcedure = DxgkObOpenProcedureStub
[2/16/2024 6:00:02 PM] [INFO]   DxgkSharedSwapChainObjectObject->CloseProcedure = SwapChainObCloseProcedure
[2/16/2024 6:00:02 PM] [INFO]   DxgkSharedSwapChainObjectObject->DeleteProcedure = SwapChainObDeleteProcedure
[2/16/2024 6:00:02 PM] [INFO]   DxgkSharedSwapChainObjectObject->DumpProcedure = 0000000000000000
[2/16/2024 6:00:02 PM] [INFO]   DxgkSharedSwapChainObjectObject->OkayToCloseProcedure = 0000000000000000
[2/16/2024 6:00:02 PM] [INFO]   DxgkSharedSwapChainObjectObject->ParseProcedure = 0000000000000000
[2/16/2024 6:00:02 PM] [INFO]   DxgkSharedSwapChainObjectObject->SecurityProcedure = SeDefaultObjectMethod
```

```
521.  [2/16/2024 6:00:02 PM] [INFO] [67] DxgkDisplayManagerObject
522.  [2/16/2024 6:00:02 PM] [INFO]   DxgkDisplayManagerObjectObject-
     >OpenProcedure = DxgkObOpenProcedureStub
523.  [2/16/2024 6:00:02 PM] [INFO]   DxgkDisplayManagerObjectObject-
     >CloseProcedure = 0000000000000000
524.  [2/16/2024 6:00:02 PM] [INFO]   DxgkDisplayManagerObjectObject-
     >DeleteProcedure = DxgkDisplayManagerDeleteProcedure
525.  [2/16/2024 6:00:02 PM] [INFO]   DxgkDisplayManagerObjectObject-
     >DumpProcedure = 0000000000000000
526.  [2/16/2024 6:00:02 PM] [INFO]   DxgkDisplayManagerObjectObject-
     >OkayToCloseProcedure = 0000000000000000
527.  [2/16/2024 6:00:02 PM] [INFO]   DxgkDisplayManagerObjectObject-
     >ParseProcedure = 0000000000000000
528.  [2/16/2024 6:00:02 PM] [INFO]   DxgkDisplayManagerObjectObject-
     >SecurityProcedure = SeDefaultObjectMethod
529.  [2/16/2024 6:00:02 PM] [INFO] [68] DxgkSharedProtectedSessionObject
530.  [2/16/2024 6:00:02 PM] [INFO]   DxgkSharedProtectedSessionObjectObject-
     >OpenProcedure = DxgkObOpenProcedureStub
531.  [2/16/2024 6:00:02 PM] [INFO]   DxgkSharedProtectedSessionObjectObject-
     >CloseProcedure = 0000000000000000
532.  [2/16/2024 6:00:02 PM] [INFO]   DxgkSharedProtectedSessionObjectObject-
     >DeleteProcedure = DxgkSharedProtectedSessionObDeleteProcedure
533.  [2/16/2024 6:00:02 PM] [INFO]   DxgkSharedProtectedSessionObjectObject-
     >DumpProcedure = 0000000000000000
534.  [2/16/2024 6:00:02 PM] [INFO]   DxgkSharedProtectedSessionObjectObject-
     >OkayToCloseProcedure = 0000000000000000
535.  [2/16/2024 6:00:02 PM] [INFO]   DxgkSharedProtectedSessionObjectObject-
     >ParseProcedure = 0000000000000000
536.  [2/16/2024 6:00:02 PM] [INFO]   DxgkSharedProtectedSessionObjectObject-
     >SecurityProcedure = SeDefaultObjectMethod
537.  [2/16/2024 6:00:02 PM] [INFO] [69] DxgkSharedBundleObject
538.  [2/16/2024 6:00:02 PM] [INFO]   DxgkSharedBundleObjectObject-
     >OpenProcedure = DxgkObOpenProcedureStub
539.  [2/16/2024 6:00:02 PM] [INFO]   DxgkSharedBundleObjectObject-
     >CloseProcedure = 0000000000000000
```

```
[2/16/2024 6:00:02 PM] [INFO]   DxgkSharedBundleObjectObject->DeleteProcedure = DxgkSharedBundleObjectObDeleteProcedure
[2/16/2024 6:00:02 PM] [INFO]   DxgkSharedBundleObjectObject->DumpProcedure = 0000000000000000
[2/16/2024 6:00:02 PM] [INFO]   DxgkSharedBundleObjectObject->OkayToCloseProcedure = 0000000000000000
[2/16/2024 6:00:02 PM] [INFO]   DxgkSharedBundleObjectObject->ParseProcedure = 0000000000000000
[2/16/2024 6:00:02 PM] [INFO]   DxgkSharedBundleObjectObject->SecurityProcedure = SeDefaultObjectMethod
[2/16/2024 6:00:02 PM] [INFO] [70] DxgkCompositionObject
[2/16/2024 6:00:02 PM] [INFO]   DxgkCompositionObjectObject->OpenProcedure = DxgkCompositionObject::Open
[2/16/2024 6:00:02 PM] [INFO]   DxgkCompositionObjectObject->CloseProcedure = DxgkCompositionObject::Close
[2/16/2024 6:00:02 PM] [INFO]   DxgkCompositionObjectObject->DeleteProcedure = DxgkCompositionObject::Delete
[2/16/2024 6:00:02 PM] [INFO]   DxgkCompositionObjectObject->DumpProcedure = 0000000000000000
[2/16/2024 6:00:02 PM] [INFO]   DxgkCompositionObjectObject->OkayToCloseProcedure = DxgkCompositionObject::OkToClose
[2/16/2024 6:00:02 PM] [INFO]   DxgkCompositionObjectObject->ParseProcedure = 0000000000000000
[2/16/2024 6:00:02 PM] [INFO]   DxgkCompositionObjectObject->SecurityProcedure = SeDefaultObjectMethod
[2/16/2024 6:00:02 PM] [INFO] [71] VRegConfigurationContext
[2/16/2024 6:00:02 PM] [INFO]   VRegConfigurationContextObject->OpenProcedure = 0000000000000000
[2/16/2024 6:00:02 PM] [INFO]   VRegConfigurationContextObject->CloseProcedure = 0000000000000000
[2/16/2024 6:00:02 PM] [INFO]   VRegConfigurationContextObject->DeleteProcedure = VrpJobContextDelete
[2/16/2024 6:00:02 PM] [INFO]   VRegConfigurationContextObject->DumpProcedure = 0000000000000000
[2/16/2024 6:00:02 PM] [INFO]   VRegConfigurationContextObject->OkayToCloseProcedure = 0000000000000000
```

```
559.  [2/16/2024 6:00:02 PM] [INFO]   VRegConfigurationContextObject-
      >ParseProcedure = 0000000000000000
560.  [2/16/2024 6:00:02 PM] [INFO]   VRegConfigurationContextObject-
      >SecurityProcedure = SeDefaultObjectMethod
561.  [2/16/2024 6:00:02 PM] [INFO] [72] CrossVmEvent
562.  [2/16/2024 6:00:02 PM] [INFO]   CrossVmEventObject->OpenProcedure =
      0000000000000000
563.  [2/16/2024 6:00:02 PM] [INFO]   CrossVmEventObject->CloseProcedure =
      ExpObCloseCrossVmEvent
564.  [2/16/2024 6:00:02 PM] [INFO]   CrossVmEventObject->DeleteProcedure =
      ExpObDeleteCrossVmEvent
565.  [2/16/2024 6:00:02 PM] [INFO]   CrossVmEventObject->DumpProcedure =
      0000000000000000
566.  [2/16/2024 6:00:02 PM] [INFO]   CrossVmEventObject->OkayToCloseProcedure
      = 0000000000000000
567.  [2/16/2024 6:00:02 PM] [INFO]   CrossVmEventObject->ParseProcedure =
      0000000000000000
568.  [2/16/2024 6:00:02 PM] [INFO]   CrossVmEventObject->SecurityProcedure =
      SeDefaultObjectMethod
569.  [2/16/2024 6:00:02 PM] [INFO] [73] CrossVmMutant
570.  [2/16/2024 6:00:02 PM] [INFO]   CrossVmMutantObject->OpenProcedure =
      0000000000000000
571.  [2/16/2024 6:00:02 PM] [INFO]   CrossVmMutantObject->CloseProcedure =
      ExpObCloseCrossVmMutant
572.  [2/16/2024 6:00:02 PM] [INFO]   CrossVmMutantObject->DeleteProcedure =
      ExpObDeleteCrossVmMutant
573.  [2/16/2024 6:00:02 PM] [INFO]   CrossVmMutantObject->DumpProcedure =
      0000000000000000
574.  [2/16/2024 6:00:02 PM] [INFO]   CrossVmMutantObject-
      >OkayToCloseProcedure = 0000000000000000
575.  [2/16/2024 6:00:02 PM] [INFO]   CrossVmMutantObject->ParseProcedure =
      0000000000000000
576.  [2/16/2024 6:00:03 PM] [INFO]   CrossVmMutantObject->SecurityProcedure =
      SeDefaultObjectMethod
```

Author

daax (https://revers.engineering/author/daax/)

View all posts (https://revers.engineering/author/daax/)

(mailto:admin@revers.engineering) (https://revers.engineering)

Leave a Reply

You must be logged in (https://revers.engineering/wp-login.php?redirect_to=https%3A%2F%2Frevers.engineering%2Fbeyond-process-and-object-callbacks-an-unconventional-method%2F) to post a comment.