Disclosure: The Mhyprot Vulnerability - Genshin Impact

May 20, 2021

windows mhyprot

Kento Oki
hrn832@protonmail.com

The Mhyprot Vulnerability - Genshin Impact

Almost over a year ago, I found a vulnerability during analysis of the driver, in `mhyprot2.sys`, windows kernel-mode driver, who responsive for protecting game-process, the `Genshin Impact` by Mihoyo.

`mhyprot` is a part of components of the client-sided anti-cheat approach.
As the `kernel-mode` drivers have *system-level privilege*, it's often provoke controversy about user's privacy and its mainly called `rootkit` as Riot's `Vanguard`, `BattlEye` and `EasyAntiCheat` does.

To clarify: I personally **do not think** these are *"Rootkit"* since I am one of the anti-cheat developer who knows what he is doing. It is necessary to have *system-level privilege* to prevent from cheating. but in other hand, it is clear it also necessary to protect user's privacy aswell.

Why

After a while, I submitted this vulnerability to the vendor, Mihoyo. And I thought that this vulnerability will be fixed very early.
Let me get straight to the point, **the vendor does not respond** or even acknowledge it.

PoC - Proof of Concept

Then I published it to the my github repository, as

- `PoC` `evil-mhyprot-cli` - (https://github.com/kkent030315/evil-mhyprot-cli)
- `PoC` `libmhyprot` - (https://github.com/kkent030315/libmhyprot)

repositories were published at *Oct 2020*. after a while, I decided to took down it, for personal reasons.
Also were popular and made a lot of duscussions.

- MiHoYo's anticheat software (mhyprot) used in Genshin Impact has been proven vulnerable, but I do not see it addressed anywhere.

> *Some days ago, a couple of PoC (proof of concept) code was shared on GitHub that takes advantage of the kernel-level anticheat Genshin Impact uses to be able to (edit: further, see this comment, my bad) compromise the system.*

- HackerNews Discussion

> *Genshin Impact's anti-cheat is not completely secure: you can use it to read/write umode memory / read kmode memory with kernel privileges: https://github.com/ScHaTTeNLiLiE/libmhyprot Mirror repo after the original author took the repo down, but still exploitable AFAIK.*

But now, the vendor company still not respond or acknowledge it, I've decided to publish it again (`May 2021`).
(BTW, those were popular from the beginning and there were many forks)

Responsible Disclosure

I, Kento Oki, am not the researcher who expect to be financially compensated.
This vulnerability is being published because the vendor does not respond or fixed it after I noticed them.

Contact: hrn832@protonmail.com

The Vulnerability

The `mhyprot` driver exposes a bunch of IOCTLs that must not be exposed to the user-mode.
For example, the driver could copy kernel virtual memory which could lead to *information-disclosure* (`CWE-200`), *privilege-escalation* (`CWE-269`) and *denial of service* since it could trigger bugcheck intentionally.

As I declared in my PoC repo (https://github.com/kkent030315/libmhyprot#features),

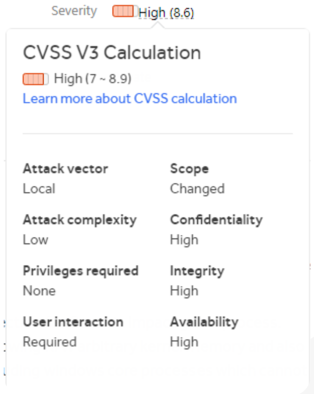
- Read Arbitrary Kernel Memory
- Read Arbitrary Process Memory
- Write Arbitrary Process Memory
- Get Arbitrary Process Modules
- Get Arbitrary Process Threads
- Get System Uptime
- Terminate Arbitrary Process

are possible with **user-privilege**.
Please note that these features is not the all. I belive there are more vulnerable commands.

And the possible impacts:

- Arbitrary Process Information Disclosure - may lead to `CWE-200`
- Arbitrary Process Virtual Memory R/W - may lead to `CWE-200` , `CWE-269` , `CWE-94`
- Arbitrary Kernel Memory R/W - may lead to `CWE-200` , `CWE-269` , `CWE-94`

Does that really makes you think product that take **user privacy** into consideration?
Also it scored as `8.6` by CVSS calculation.



Severity 🟥 High (8.6)

CVSS V3 Calculation

🟥 High (7 ~ 8.9)

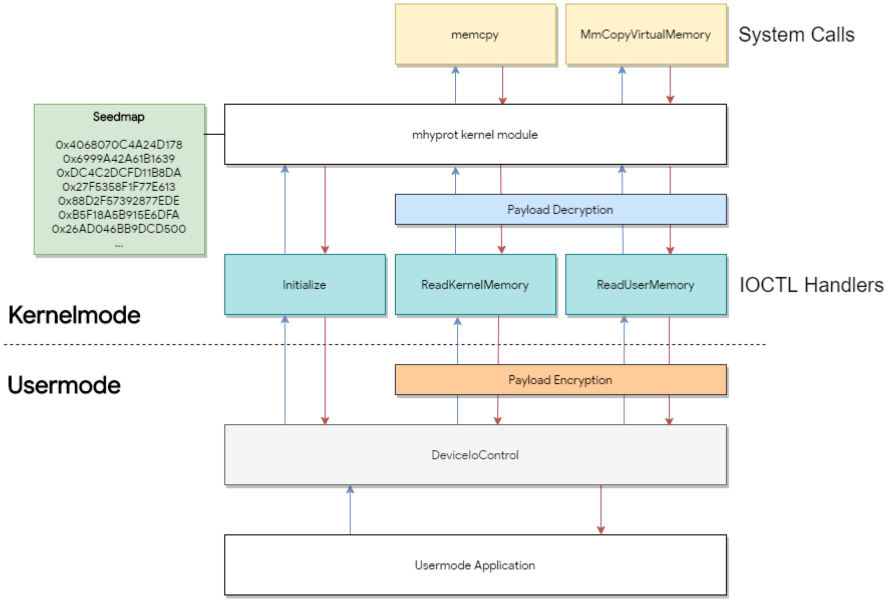
Learn more about CVSS calculation

Attack vector Local	Scope Changed
Attack complexity Low	Confidentiality High
Privileges required None	Integrity High
User interaction Required	Availability High

It feel like the biggest *backdoor* I've ever seen before.

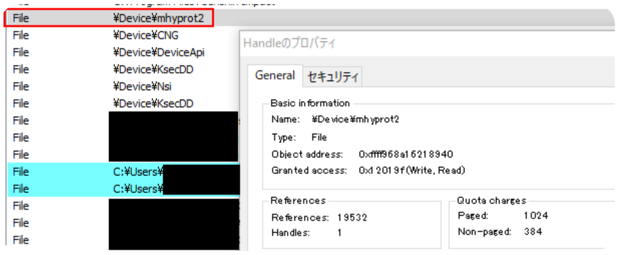
Introduction To The Vulnerable Driver

The `mhyprot` is an anti-cheat kernel mode driver used in [`Genshin Impact`](Genshin Impact) .
The driver has vulnerable `IOCTL` commands that allows attackers to execute improperly from ring-3 (usermode), without privileges that usually needed to be granted by OS system.



# Usermode Module

Driver's device handle is opened by the game process `GenshinImpact.exe` .



File ¥Device¥mhyprot2
File ¥Device¥CNG
File ¥Device¥DeviceApi
File ¥Device¥KsecDD
File ¥Device¥Nsi
File ¥Device¥KsecDD
File [redacted]
File
File
File C:¥Users¥[redacted]
File C:¥Users¥
File [redacted]
File
File

Basic information
Name: ¥Device¥mhyprot2
Type: File
Object address: 0xffff968a16218940
Granted access: 0x12019f(Write, Read)

References
References: 19532
Handles: 1

Quota charges
Paged: 1024
Non-paged: 384

Driver Initialization

The `MHYPROT_IOCTL_INITIALIZE` what I defined in mhyprot.hpp can be found as follows:

```
PAGE:FFFFF800188CD8FD loc_FFFFF800188CD8FD:                    ; CODE XREF: sub_FFFFF800188CD6E0+213↑j
PAGE:FFFFF800188CD8FD                 cmp     ecx, 80034000h  ; MHYPROT_IOCTL_INITIALIZE
PAGE:FFFFF800188CD903                 jnz     short loc_FFFFF800188CD984
PAGE:FFFFF800188CD905                 cmp     r8d, 10h
PAGE:FFFFF800188CD909                 jnz     loc_FFFFF800188CDA4F
PAGE:FFFFF800188CD90F                 mov     rax, 0EBBAAEF4FFF89042h // <- _m_002
PAGE:FFFFF800188CD919                 xor     [rdi+8], rax
PAGE:FFFFF800188CD91D                 mov     rax, [rdi+8]
PAGE:FFFFF800188CD921                 xor     [rdi], rax
PAGE:FFFFF800188CD924                 cmp     dword ptr [rdi+4], 0BAEBAEECh // <- _m_001
PAGE:FFFFF800188CD92B                 jnz     loc_FFFFF800188CDA4F
PAGE:FFFFF800188CD931                 mov     ecx, [rdi]
PAGE:FFFFF800188CD933                 call    sub_FFFFF800188C51A8
PAGE:FFFFF800188CD938                 cmp     dword ptr cs:qword_FFFFF800188CA108, 0
PAGE:FFFFF800188CD93F                 jnz     short loc_FFFFF800188CD97D
PAGE:FFFFF800188CD941                 mov     rdx, [rdi+8]
PAGE:FFFFF800188CD945                 lea     rcx, xmmword_FFFFF800188CA0E8
PAGE:FFFFF800188CD94C                 call    sub_FFFFF800188C301C // <-
PAGE:FFFFF800188CD951                 mov     ebx, 7
```

and the `sub_FFFFF800188C301C` is look like:

```
.text:FFFFF800188C301C ; =============== S U B R O U T I N E =======================================
.text:FFFFF800188C301C
.text:FFFFF800188C301C
.text:FFFFF800188C301C sub_FFFFF800188C301C proc near          ; CODE XREF: sub_FFFFF800188CD6E0+26C↓p
.text:FFFFF800188C301C                                         ; DATA XREF: .upx0:FFFFF800189F2BA8↓o
.text:FFFFF800188C301C
.text:FFFFF800188C301C arg_0           = qword ptr  8
.text:FFFFF800188C301C
.text:FFFFF800188C301C                 test    rcx, rcx
.text:FFFFF800188C301F                 jz      locret_FFFFF800188C30B4
.text:FFFFF800188C3025                 mov     [rsp+arg_0], rbx
.text:FFFFF800188C302A                 push    rdi
.text:FFFFF800188C302B                 sub     rsp, 20h
.text:FFFFF800188C302F                 xor     eax, eax
.text:FFFFF800188C3031                 mov     rdi, rdx
.text:FFFFF800188C3034                 mov     [rcx], rax
.text:FFFFF800188C3037                 mov     rbx, rcx
.text:FFFFF800188C303A                 mov     [rcx+8], rax
.text:FFFFF800188C303E                 mov     edx, 9C0h       ; NumberOfBytes
.text:FFFFF800188C3043                 xor     ecx, ecx        ; PoolType
.text:FFFFF800188C3045                 call    cs:ExAllocatePool
.text:FFFFF800188C304B                 xor     edx, edx
.text:FFFFF800188C304D                 mov     r8d, 9C0h
.text:FFFFF800188C3053                 mov     rcx, rax
.text:FFFFF800188C3056                 mov     [rbx], rax
.text:FFFFF800188C3059                 call    sub_FFFFF800188C7900
.text:FFFFF800188C305E                 mov     rax, [rbx]
.text:FFFFF800188C3061                 mov     r9d, 1
.text:FFFFF800188C3067                 mov     [rbx+0Ch], r9d
.text:FFFFF800188C306B                 mov     [rax], rdi
.text:FFFFF800188C306E                 mov     [rbx+8], r9d
.text:FFFFF800188C3072
.text:FFFFF800188C3072 loc_FFFFF800188C3072:                   ; CODE XREF: sub_FFFFF800188C301C+8C↓j
.text:FFFFF800188C3072                 movsxd  r8, dword ptr [rbx+8]
.text:FFFFF800188C3076                 mov     rdx, [rbx]
.text:FFFFF800188C3079                 mov     rax, [rdx+r8*8-8]
.text:FFFFF800188C307E                 mov     rcx, rax
.text:FFFFF800188C3081                 shr     rcx, 3Eh
.text:FFFFF800188C3085                 xor     rcx, rax
.text:FFFFF800188C3088                 mov     rax, 5851F42D4C957F2Dh
.text:FFFFF800188C3092                 imul    rcx, rax
.text:FFFFF800188C3096                 add     rcx, r8
.text:FFFFF800188C3099                 mov     [rdx+r8*8], rcx
.text:FFFFF800188C309D                 add     [rbx+8], r9d
.text:FFFFF800188C30A1                 cmp     dword ptr [rbx+8], 138h
.text:FFFFF800188C30A8                 jl      short loc_FFFFF800188C3072
.text:FFFFF800188C30AA                 mov     rbx, [rsp+28h+arg_0]
.text:FFFFF800188C30AF                 add     rsp, 20h
.text:FFFFF800188C30B3                 pop     rdi
.text:FFFFF800188C30B4
.text:FFFFF800188C30B4 locret_FFFFF800188C30B4:                ; CODE XREF: sub_FFFFF800188C301C+3↑j
.text:FFFFF800188C30B4                 retn
.text:FFFFF800188C30B4 sub_FFFFF800188C301C endp
```

Copy Arbitrary Kernel Memory

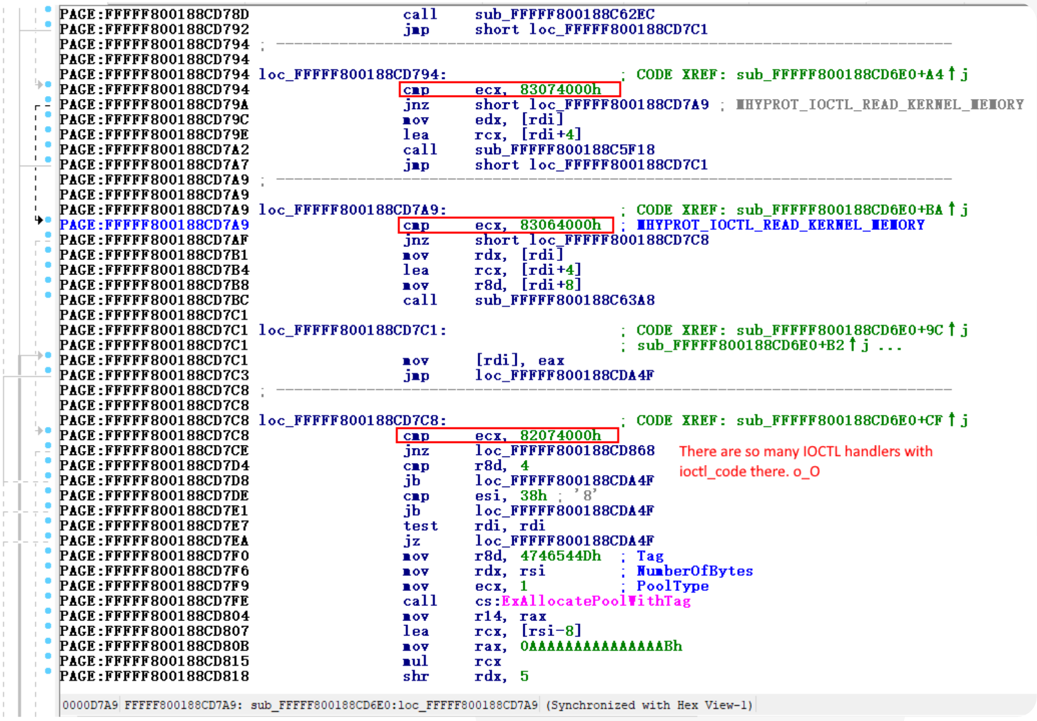
There are so many IOCTL commands and the `MHYPROT_IOCTL_READ_KERNEL_MEMORY` what I defined in mhyprot.hpp can be found as follows:

```
PAGE:FFFFF800188CD7A9 loc_FFFFF800188CD7A9:                    ; CODE XREF: sub_FFFFF800188CD6E0+BA↑j
PAGE:FFFFF800188CD7A9                 cmp     ecx, 83064000h  ; MHYPROT_IOCTL_READ_KERNEL_MEMORY
PAGE:FFFFF800188CD7AF                 jnz     short loc_FFFFF800188CD7C8
PAGE:FFFFF800188CD7B1                 mov     rdx, [rdi]
PAGE:FFFFF800188CD7B4                 lea     rcx, [rdi+4]
PAGE:FFFFF800188CD7B8                 mov     r8d, [rdi+8]
PAGE:FFFFF800188CD7BC                 call    sub_FFFFF800188C63A8 // <-
```

And the `sub_FFFFF800188C63A8` is like:

```
.text:FFFFF800188C63A8 sub_FFFFF800188C63A8 proc near          ; CODE XREF: sub_FFFFF800188CD6E0+DC↓p
.text:FFFFF800188C63A8                                         ; DATA XREF: .upx0:FFFFF800189F2EE4↓o
.text:FFFFF800188C63A8
.text:FFFFF800188C63A8 arg_0           = qword ptr  8
.text:FFFFF800188C63A8 arg_8           = qword ptr  10h
.text:FFFFF800188C63A8
.text:FFFFF800188C63A8                 mov     [rsp+arg_0], rbx
.text:FFFFF800188C63AD                 mov     [rsp+arg_8], rsi
.text:FFFFF800188C63B2                 push    rdi
.text:FFFFF800188C63B3                 sub     rsp, 20h
.text:FFFFF800188C63B7                 mov     edi, r8d
.text:FFFFF800188C63BA                 mov     rbx, rdx
.text:FFFFF800188C63BD                 mov     rsi, rcx
.text:FFFFF800188C63C0                 test    rdx, rdx
.text:FFFFF800188C63C3                 jz      short loc_FFFFF800188C63F2
.text:FFFFF800188C63C5                 test    r8d, r8d
.text:FFFFF800188C63C8                 jz      short loc_FFFFF800188C63F2
.text:FFFFF800188C63CA                 mov     rax, cs:MmHighestUserAddress
.text:FFFFF800188C63D1                 cmp     rdx, [rax]
.text:FFFFF800188C63D4                 jb      short loc_FFFFF800188C63F2
.text:FFFFF800188C63D6                 mov     r8d, edi
.text:FFFFF800188C63D9                 xor     edx, edx
.text:FFFFF800188C63DB                 call    sub_FFFFF800188C7900
.text:FFFFF800188C63E0                 mov     r8d, edi
.text:FFFFF800188C63E3                 mov     rdx, rsi
.text:FFFFF800188C63E6                 mov     rcx, rbx
.text:FFFFF800188C63E9                 call    sub_FFFFF800188C3DD8
.text:FFFFF800188C63EE                 xor     eax, eax
.text:FFFFF800188C63F0                 jmp     short loc_FFFFF800188C63F5
```

Here is the ioctl handlers, found the `0x83064000` (`MHYPROT_IOCTL_READ_KERNEL_MEMORY`) as `cmp ecx, 83064000h` and some another ioctl codes as follows:

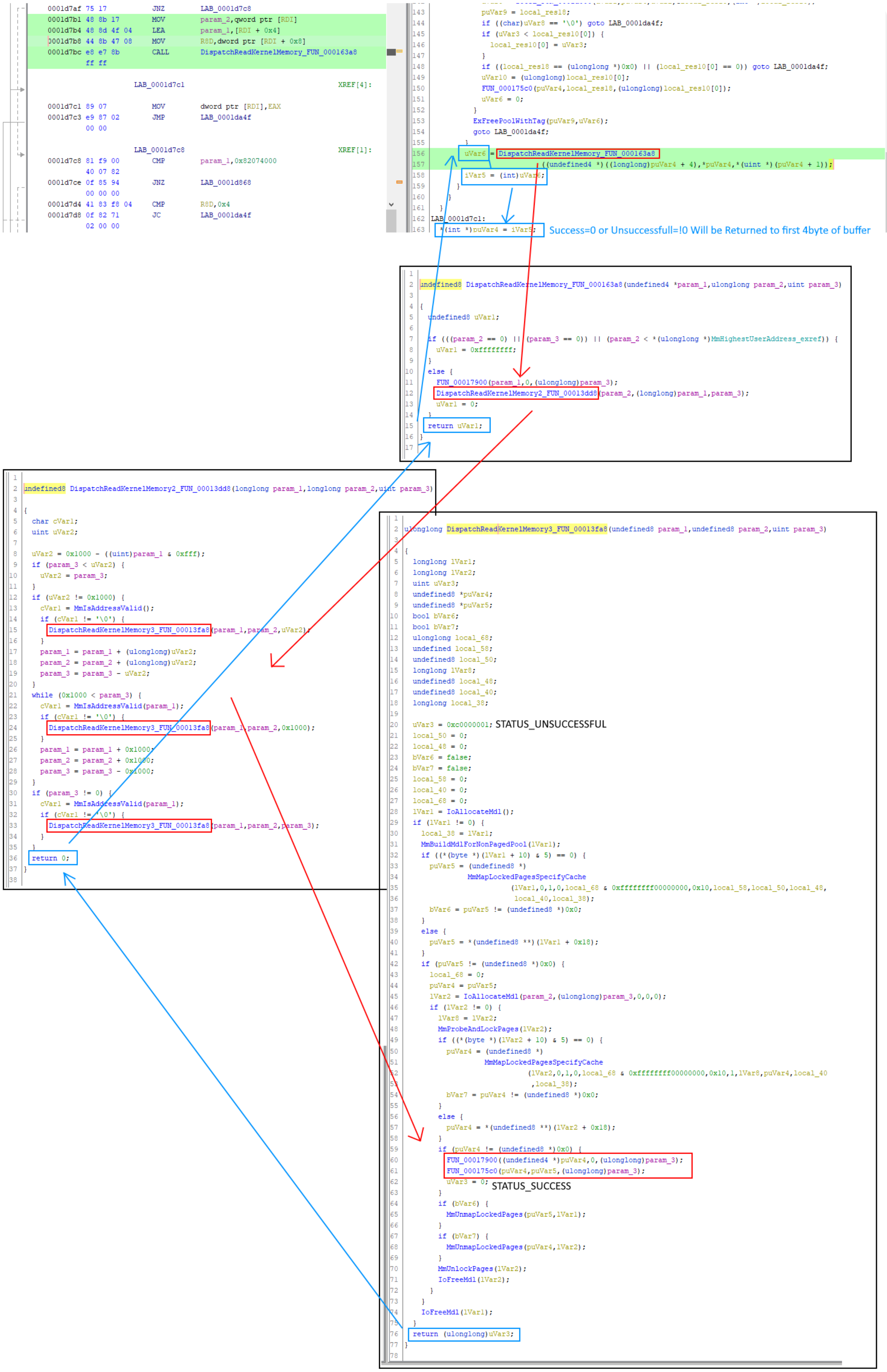


```
PAGE:FFFFF800188CD78D                 call    sub_FFFFF800188C62EC
PAGE:FFFFF800188CD792                 jmp     short loc_FFFFF800188CD7C1
PAGE:FFFFF800188CD794 ; ---------------------------------------------------------------------------
PAGE:FFFFF800188CD794
PAGE:FFFFF800188CD794 loc_FFFFF800188CD794:                   ; CODE XREF: sub_FFFFF800188CD6E0+A4↑j
PAGE:FFFFF800188CD794                 cmp     ecx, 83074000h
PAGE:FFFFF800188CD79A                 jnz     short loc_FFFFF800188CD7A9 ; MHYPROT_IOCTL_READ_KERNEL_MEMORY
PAGE:FFFFF800188CD79C                 mov     edx, [rdi]
PAGE:FFFFF800188CD79E                 lea     rcx, [rdi+4]
PAGE:FFFFF800188CD7A2                 call    sub_FFFFF800188C5F18
PAGE:FFFFF800188CD7A7                 jmp     short loc_FFFFF800188CD7C1
PAGE:FFFFF800188CD7A9 ; ---------------------------------------------------------------------------
PAGE:FFFFF800188CD7A9
PAGE:FFFFF800188CD7A9 loc_FFFFF800188CD7A9:                   ; CODE XREF: sub_FFFFF800188CD6E0+BA↑j
PAGE:FFFFF800188CD7A9                 cmp     ecx, 83064000h  ; MHYPROT_IOCTL_READ_KERNEL_MEMORY
PAGE:FFFFF800188CD7AF                 jnz     short loc_FFFFF800188CD7C8
PAGE:FFFFF800188CD7B1                 mov     rdx, [rdi]
PAGE:FFFFF800188CD7B4                 lea     rcx, [rdi+4]
PAGE:FFFFF800188CD7B8                 mov     r8d, [rdi+8]
PAGE:FFFFF800188CD7BC                 call    sub_FFFFF800188C63A8
PAGE:FFFFF800188CD7C1
PAGE:FFFFF800188CD7C1 loc_FFFFF800188CD7C1:                   ; CODE XREF: sub_FFFFF800188CD6E0+9C↑j
PAGE:FFFFF800188CD7C1                                         ; sub_FFFFF800188CD6E0+B2↑j ...
PAGE:FFFFF800188CD7C1                 mov     [rdi], eax
PAGE:FFFFF800188CD7C3                 jmp     loc_FFFFF800188CDA4F
PAGE:FFFFF800188CD7C8 ; ---------------------------------------------------------------------------
PAGE:FFFFF800188CD7C8
PAGE:FFFFF800188CD7C8 loc_FFFFF800188CD7C8:                   ; CODE XREF: sub_FFFFF800188CD6E0+CF↑j
PAGE:FFFFF800188CD7C8                 cmp     ecx, 82074000h
PAGE:FFFFF800188CD7CE                 jnz     loc_FFFFF800188CD868
PAGE:FFFFF800188CD7D4                 cmp     r8d, 4
PAGE:FFFFF800188CD7D8                 jb      loc_FFFFF800188CDA4F
PAGE:FFFFF800188CD7DE                 cmp     esi, 38h ; '8'
PAGE:FFFFF800188CD7E1                 jb      loc_FFFFF800188CDA4F
PAGE:FFFFF800188CD7E7                 test    rdi, rdi
PAGE:FFFFF800188CD7EA                 jz      loc_FFFFF800188CDA4F
PAGE:FFFFF800188CD7F0                 mov     r8d, 4746544Dh  ; Tag
PAGE:FFFFF800188CD7F6                 mov     rdx, rsi        ; NumberOfBytes
PAGE:FFFFF800188CD7F9                 mov     ecx, 1          ; PoolType
PAGE:FFFFF800188CD7FE                 call    cs:ExAllocatePoolWithTag
PAGE:FFFFF800188CD804                 mov     r14, rax
PAGE:FFFFF800188CD807                 lea     rcx, [rsi-8]
PAGE:FFFFF800188CD80B                 mov     rax, 0AAAAAAAAAAAAAAABh
PAGE:FFFFF800188CD815                 mul     rcx
PAGE:FFFFF800188CD818                 shr     rdx, 5
0000D7A9 FFFFF800188CD7A9: sub_FFFFF800188CD6E0:loc_FFFFF800188CD7A9 (Synchronized with Hex View-1)
```

There are so many IOCTL handlers with ioctl_code there. o_O

# Call map

As I defined as `DWORD result` in mhyprot.hpp the first 4bytes is result.
I can guess it's a `NTSTATUS` as it typedef'ed as `typedef LONG NTSTATUS` natively and the dispathers return types are `NTSTATUS` and the result will directly be got stored from it.



```
0001d7af 75 17          JNZ     LAB_0001d7c8
0001d7b1 48 8b 17       MOV     param_2,qword ptr [RDI]
0001d7b4 48 8d 4f 04    LEA     param_1,[RDI + 0x4]
0001d7b8 44 8b 47 08    MOV     R8D,dword ptr [RDI + 0x8]
0001d7bc e8 e7 8b       CALL    DispatchReadKernelMemory_FUN_000163a8
         ff ff

                  LAB_0001d7c1                              XREF[4]:

0001d7c1 89 07          MOV     dword ptr [RDI],EAX
0001d7c3 e9 87 02       JMP     LAB_0001da4f
         00 00

                  LAB_0001d7c8                              XREF[1]:
0001d7c8 81 f9 00       CMP     param_1,0x82074000
         40 07 82
0001d7ce 0f 85 94       JNZ     LAB_0001d868
         00 00 00
0001d7d4 41 83 f8 04    CMP     R8D,0x4
0001d7d8 0f 82 71       JC      LAB_0001da4f
         02 00 00
```

```
          puVar9 = local_res18;
          if ((char)uVar8 == '\0') goto LAB_0001da4f;
          if (uVar3 < local_res10[0]) {
            local_res10[0] = uVar3;
          }
          if ((local_res18 == (ulonglong *)0x0) || (local_res10[0] == 0)) goto LAB_0001da4f;
          uVar10 = (ulonglong)local_res10[0];
          FUN_000175c0(puVar4,local_res18,(ulonglong)local_res10[0]);
          uVar6 = 0;
        }
        ExFreePoolWithTag(puVar9,uVar6);
        goto LAB_0001da4f;
      }
      uVar6 = DispatchReadKernelMemory_FUN_000163a8
                        ((undefined4 *)((longlong)puVar4 + 4),*puVar4,*(uint *)(puVar4 + 1));
      iVar5 = (int)uVar6;
    }
  }
}
LAB_0001d7c1:
  *(int *)puVar4 = iVar5;
```

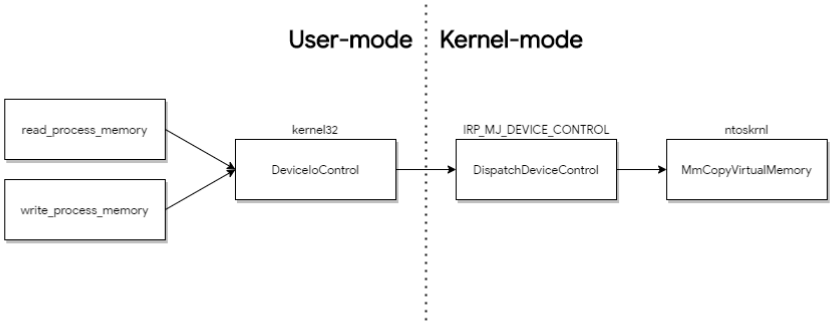
Success=0 or Unsuccessfull=!0 Will be Returned to first 4byte of buffer

```

undefined8 DispatchReadKernelMemory_FUN_000163a8(undefined4 *param_1,ulonglong param_2,uint param_3)

{
  undefined8 uVar1;

  if (((param_2 == 0) || (param_3 == 0)) || (param_2 < *(ulonglong *)MmHighestUserAddress_exref)) {
    uVar1 = 0xffffffff;
  }
  else {
    FUN_00017900(param_1,0,(ulonglong)param_3);
    DispatchReadKernelMemory2_FUN_00013dd8(param_2,(longlong)param_1,param_3);
    uVar1 = 0;
  }
  return uVar1;
}

```

```

undefined8 DispatchReadKernelMemory2_FUN_00013dd8(longlong param_1,longlong param_2,uint param_3)

{
  char cVar1;
  uint uVar2;

  uVar2 = 0x1000 - ((uint)param_1 & 0xfff);
  if (param_3 < uVar2) {
    uVar2 = param_3;
  }
  if (uVar2 != 0x1000) {
    cVar1 = MmIsAddressValid();
    if (cVar1 != '\0') {
      DispatchReadKernelMemory3_FUN_00013fa8(param_1,param_2,uVar2);
    }
    param_1 = param_1 + (ulonglong)uVar2;
    param_2 = param_2 + (ulonglong)uVar2;
    param_3 = param_3 - uVar2;
  }
  while (0x1000 < param_3) {
    cVar1 = MmIsAddressValid(param_1);
    if (cVar1 != '\0') {
      DispatchReadKernelMemory3_FUN_00013fa8(param_1,param_2,0x1000);
    }
    param_1 = param_1 + 0x1000;
    param_2 = param_2 + 0x1000;
    param_3 = param_3 - 0x1000;
  }
  if (param_3 != 0) {
    cVar1 = MmIsAddressValid(param_1);
    if (cVar1 != '\0') {
      DispatchReadKernelMemory3_FUN_00013fa8(param_1,param_2,param_3);
    }
  }
  return 0;
}

```

```

ulonglong DispatchReadKernelMemory3_FUN_00013fa8(undefined8 param_1,undefined8 param_2,uint param_3)

{
  longlong lVar1;
  longlong lVar2;
  uint uVar3;
  undefined8 *puVar4;
  undefined8 *puVar5;
  bool bVar6;
  bool bVar7;
  ulonglong local_68;
  undefined local_58;
  undefined8 local_50;
  longlong lVar8;
  undefined8 local_48;
  undefined8 local_40;
  longlong local_38;

  uVar3 = 0xc0000001; STATUS_UNSUCCESSFUL
  local_50 = 0;
  local_48 = 0;
  bVar6 = false;
  bVar7 = false;
  local_58 = 0;
  local_40 = 0;
  local_68 = 0;
  lVar1 = IoAllocateMdl();
  if (lVar1 != 0) {
    local_38 = lVar1;
    MmBuildMdlForNonPagedPool(lVar1);
    if ((*(byte *)(lVar1 + 10) & 5) == 0) {
      puVar5 = (undefined8 *)
               MmMapLockedPagesSpecifyCache
                         (lVar1,0,1,0,local_68 & 0xffffffff00000000,0x10,local_58,local_50,local_48,
                          local_40,local_38);
      bVar6 = puVar5 != (undefined8 *)0x0;
    }
    else {
      puVar5 = *(undefined8 **)(lVar1 + 0x18);
    }
    if (puVar5 != (undefined8 *)0x0) {
      local_68 = 0;
      puVar4 = puVar5;
      lVar2 = IoAllocateMdl(param_2,(ulonglong)param_3,0,0,0);
      if (lVar2 != 0) {
        lVar8 = lVar2;
        MmProbeAndLockPages(lVar2);
        if ((*(byte *)(lVar2 + 10) & 5) == 0) {
          puVar4 = (undefined8 *)
                   MmMapLockedPagesSpecifyCache
                             (lVar2,0,1,0,local_68 & 0xffffffff00000000,0x10,1,lVar8,puVar4,local_40
                              ,local_38);
          bVar7 = puVar4 != (undefined8 *)0x0;
        }
        else {
          puVar4 = *(undefined8 **)(lVar2 + 0x18);
        }
        if (puVar4 != (undefined8 *)0x0) {
          FUN_00017900((undefined4 *)puVar4,0,(ulonglong)param_3);
          FUN_000175c0(puVar4,puVar5,(ulonglong)param_3);
          uVar3 = 0; STATUS_SUCCESS
        }
        if (bVar6) {
          MmUnmapLockedPages(puVar5,lVar1);
        }
        if (bVar7) {
          MmUnmapLockedPages(puVar4,lVar2);
        }
        MmUnlockPages(lVar2);
        IoFreeMdl(lVar2);
      }
    }
    IoFreeMdl(lVar1);
  }
  return (ulonglong)uVar3;
}

```

Copy Arbitrary Process Memory



The mhyprot calls `MmCopyVirtualMemory` eventually as wrapper defined as follows:

```
__int64 __fastcall sub_FFFFF800188C3EB8(struct _EPROCESS *a1, _DWORD *a2, __int64 a3)
{
  __int64 v3; // rbp
  _DWORD *v4; // rdi
  struct _EPROCESS *v5; // rbx
  PEPROCESS v6; // rsi
  char v8; // [rsp+28h] [rbp-20h]

  v3 = a3;
  v4 = a2;
  v5 = a1;
  if ( *a2 == 1 )
  {
    v6 = IoGetCurrentProcess();
  }
  else
  {
    v6 = a1;
    v5 = IoGetCurrentProcess();
  }
  v8 = 0;
  return MmCopyVirtualMemory(v6, *((_QWORD *)v4 + 3), v5, *((_QWORD *)v4 + 2), (unsigned int)v4[8], v8, v3);
}
```

Called by:

```
__int64 __fastcall sub_FFFFF800188C3F2C(_DWORD *a1_rw_request, __int64 a2_returnsize, __int64 a3)
{
  __int64 v3_returnsize; // rsi
  _DWORD *v4_rw_request; // rbx
  __int64 v5_processid; // rcx
  bool v6_ntstatus_lookup_success_bool; // di
  unsigned int v8_ntstatus; // ebx
  PVOID Object; // [rsp+40h] [rbp+8h]

  v3_returnsize = a2_returnsize;
  v4_rw_request = a1_rw_request;
  v5_processid = (unsigned int)a1_rw_request[2];
  Object = 0i64;
  v6_ntstatus_lookup_success_bool = (int)PsLookupProcessByProcessId(v5_processid, &Object, a3) >= 0;// NT_SUCCESS
  if ( !Object )
    return 3221225473i64;
  v8_ntstatus = sub_FFFFF800188C3EB8((struct _EPROCESS *)Object, v4_rw_request, v3_returnsize);
  if ( v6_ntstatus_lookup_success_bool )
    ObfDereferenceObject(Object);
  return v8_ntstatus;
}
```

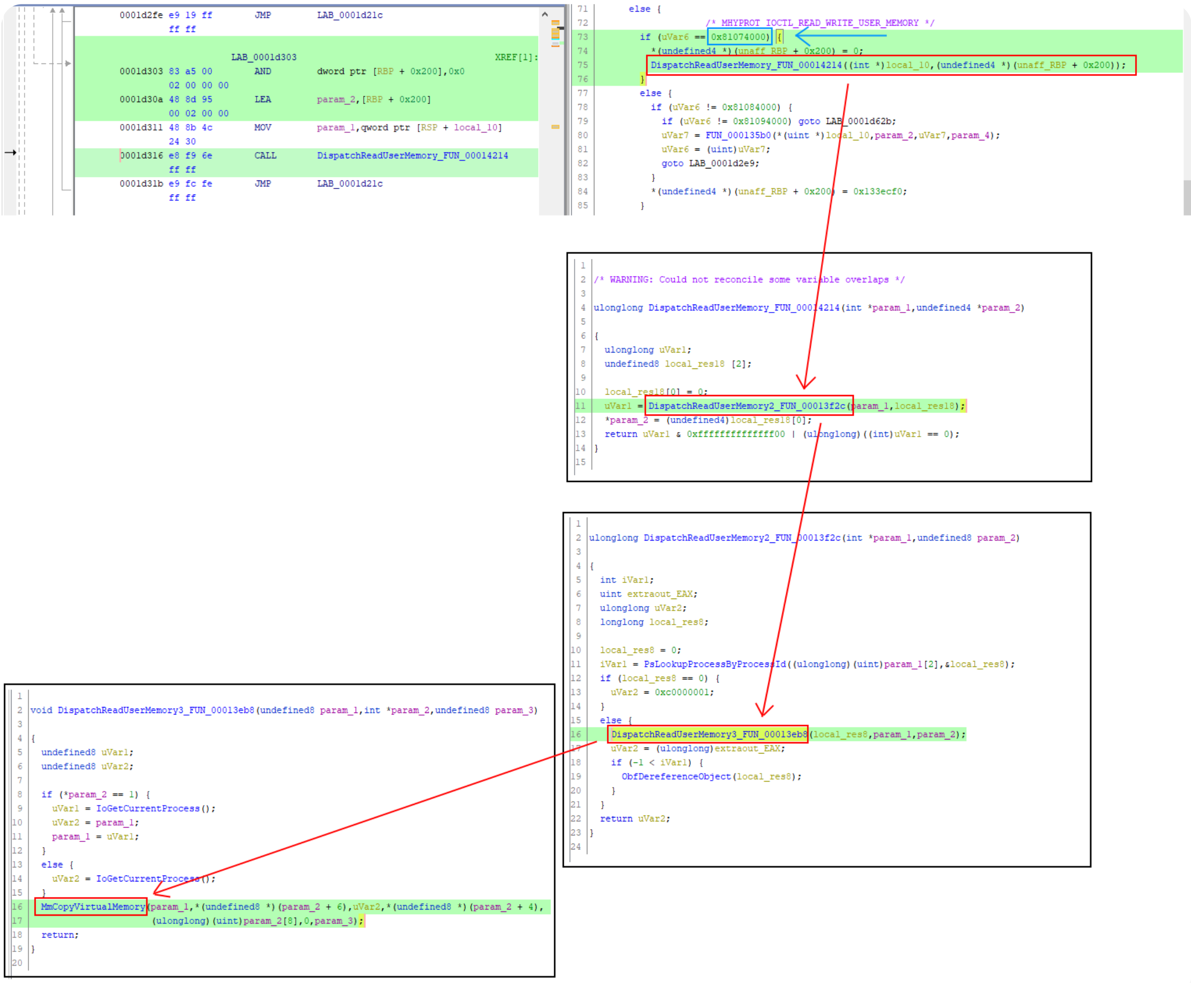
Called by:

```
bool __fastcall sub_FFFFF800188C4214(_DWORD *a1_rw_request, _DWORD *a2_returnsize, __int64 a3)
{
  _DWORD *v3_returnsize; // rbx
  int v5_ntstatus; // [rsp+20h] [rbp-18h]
  __int64 v6_returnsize; // [rsp+50h] [rbp+18h]

  v3_returnsize = a2_returnsize;
  v6_returnsize = 0i64;
  v5_ntstatus = sub_FFFFF800188C3F2C(a1_rw_request, (__int64)&v6_returnsize, a3);
  *v3_returnsize = v6_returnsize;
  return v5_ntstatus == 0;                      // NT_SUCCESS(v5_ntstatus)
}
```

Finally we are at the root of the tree, this is in the packed segment and is in encryption-dedicated IOCTL handler function:

```
PAGE:FFFFF800188CD303 loc_FFFFF800188CD303:                   ; CODE XREF: sub_FFFFF800188CD000+2C7↑j
PAGE:FFFFF800188CD303                 and     dword ptr [rbp+1D0h+arg_20], 0
PAGE:FFFFF800188CD30A                 lea     rdx, [rbp+1D0h+arg_20]
PAGE:FFFFF800188CD311                 mov     rcx, [rsp+30h]
PAGE:FFFFF800188CD316                 call    sub_FFFFF800188C4214 // <- Here
PAGE:FFFFF800188CD31B                 jmp     loc_FFFFF800188CD21C
```

# Call map



```
0001d2fe e9 19 ff        JMP        LAB_0001d21c
         ff ff
                 LAB_0001d303                                XREF[1]:
0001d303 83 a5 00        AND        dword ptr [RBP + 0x200],0x0
         02 00 00 00
0001d30a 48 8d 95        LEA        param_2,[RBP + 0x200]
         00 02 00 00
0001d311 48 8b 4c        MOV        param_1,qword ptr [RSP + local_10]
         24 30
0001d316 e8 f9 6e        CALL       DispatchReadUserMemory_FUN_00014214
         ff ff
0001d31b e9 fc fe        JMP        LAB_0001d21c
         ff ff
```

```
      else {
                    /* MHYPROT_IOCTL_READ_WRITE_USER_MEMORY */
        if (uVar6 == 0x81074000) {
          *(undefined4 *)(unaff_RBP + 0x200) = 0;
          DispatchReadUserMemory_FUN_00014214((int *)local_10,(undefined4 *)(unaff_RBP + 0x200));
        }
        else {
          if (uVar6 != 0x81084000) {
            if (uVar6 != 0x81094000) goto LAB_0001d62b;
            uVar7 = FUN_000135b0(*(uint *)local_10,param_2,uVar7,param_4);
            uVar6 = (uint)uVar7;
            goto LAB_0001d2e9;
          }
          *(undefined4 *)(unaff_RBP + 0x200) = 0x133ecf0;
        }
```

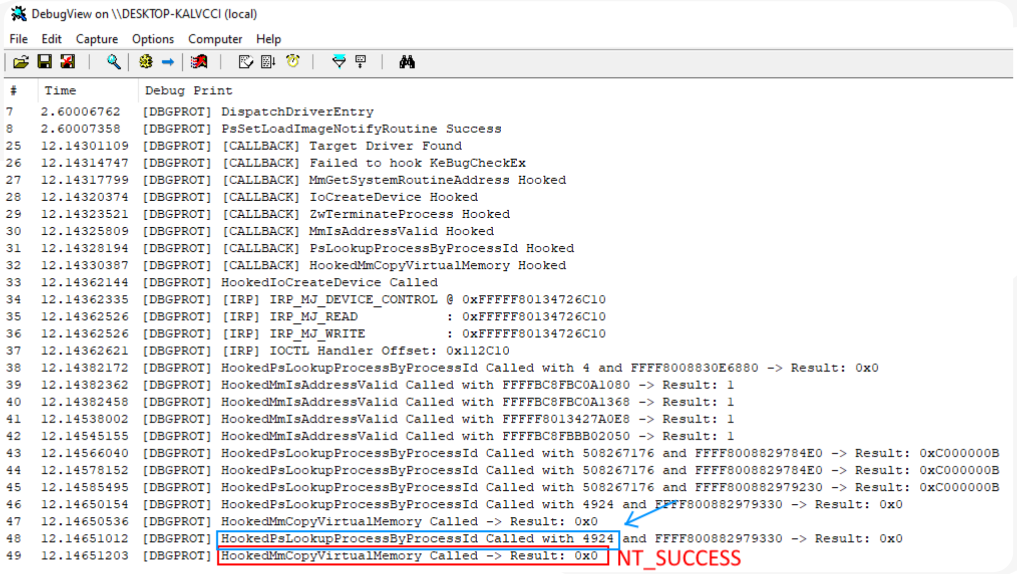
```

/* WARNING: Could not reconcile some variable overlaps */

ulonglong DispatchReadUserMemory_FUN_00014214(int *param_1,undefined4 *param_2)

{
  ulonglong uVar1;
  undefined8 local_res18 [2];

  local_res18[0] = 0;
  uVar1 = DispatchReadUserMemory2_FUN_00013f2c(param_1,local_res18);
  *param_2 = (undefined4)local_res18[0];
  return uVar1 & 0xffffffffffffff00 | (ulonglong)((int)uVar1 == 0);
}

```

```

ulonglong DispatchReadUserMemory2_FUN_00013f2c(int *param_1,undefined8 param_2)

{
  int iVar1;
  uint extraout_EAX;
  ulonglong uVar2;
  longlong local_res8;

  local_res8 = 0;
  iVar1 = PsLookupProcessByProcessId((ulonglong)(uint)param_1[2],&local_res8);
  if (local_res8 == 0) {
    uVar2 = 0xc0000001;
  }
  else {
    DispatchReadUserMemory3_FUN_00013eb8(local_res8,param_1,param_2);
    uVar2 = (ulonglong)extraout_EAX;
    if (-1 < iVar1) {
      ObfDereferenceObject(local_res8);
    }
  }
  return uVar2;
}

```

```

void DispatchReadUserMemory3_FUN_00013eb8(undefined8 param_1,int *param_2,undefined8 param_3)

{
  undefined8 uVar1;
  undefined8 uVar2;

  if (*param_2 == 1) {
    uVar1 = IoGetCurrentProcess();
    uVar2 = param_1;
    param_1 = uVar1;
  }
  else {
    uVar2 = IoGetCurrentProcess();
  }
  MmCopyVirtualMemory(param_1,*(undefined8 *)(param_2 + 6),uVar2,*(undefined8 *)(param_2 + 4),
                      (ulonglong)(uint)param_2[8],0,param_3);
  return;
}

```

# Proof

I have confirmed that by simply hooking mhyprot kernel module:



DebugView on \\DESKTOP-KALVCCI (local)
File Edit Capture Options Computer Help

```
#   Time        Debug Print
7   2.60006762  [DBGPROT] DispatchDriverEntry
8   2.60007358  [DBGPROT] PsSetLoadImageNotifyRoutine Success
25  12.14301109 [DBGPROT] [CALLBACK] Target Driver Found
26  12.14314747 [DBGPROT] [CALLBACK] Failed to hook KeBugCheckEx
27  12.14317799 [DBGPROT] [CALLBACK] MmGetSystemRoutineAddress Hooked
28  12.14320374 [DBGPROT] [CALLBACK] IoCreateDevice Hooked
29  12.14323521 [DBGPROT] [CALLBACK] ZwTerminateProcess Hooked
30  12.14325809 [DBGPROT] [CALLBACK] MmIsAddressValid Hooked
31  12.14328194 [DBGPROT] [CALLBACK] PsLookupProcessByProcessId Hooked
32  12.14330387 [DBGPROT] [CALLBACK] HookedMmCopyVirtualMemory Hooked
33  12.14362144 [DBGPROT] HookedIoCreateDevice Called
34  12.14362335 [DBGPROT] [IRP] IRP_MJ_DEVICE_CONTROL @ 0xFFFFF80134726C10
35  12.14362526 [DBGPROT] [IRP] IRP_MJ_READ           : 0xFFFFF80134726C10
36  12.14362526 [DBGPROT] [IRP] IRP_MJ_WRITE          : 0xFFFFF80134726C10
37  12.14362621 [DBGPROT] [IRP] IOCTL Handler Offset: 0x112C10
38  12.14382172 [DBGPROT] HookedPsLookupProcessByProcessId Called with 4 and FFFF8008830E6880 -> Result: 0x0
39  12.14382362 [DBGPROT] HookedMmIsAddressValid Called with FFFFBC8FBC0A1080 -> Result: 1
40  12.14382458 [DBGPROT] HookedMmIsAddressValid Called with FFFFBC8FBC0A1368 -> Result: 1
41  12.14538002 [DBGPROT] HookedMmIsAddressValid Called with FFFFF8013427A0E8 -> Result: 1
42  12.14545155 [DBGPROT] HookedMmIsAddressValid Called with FFFFBC8FBBB02050 -> Result: 1
43  12.14566040 [DBGPROT] HookedPsLookupProcessByProcessId Called with 508267176 and FFFF8008829784E0 -> Result: 0xC000000B
44  12.14578152 [DBGPROT] HookedPsLookupProcessByProcessId Called with 508267176 and FFFF8008829784E0 -> Result: 0xC000000B
45  12.14585495 [DBGPROT] HookedPsLookupProcessByProcessId Called with 508267176 and FFFF800882979230 -> Result: 0xC000000B
46  12.14650154 [DBGPROT] HookedPsLookupProcessByProcessId Called with 4924 and FFFF800882979330 -> Result: 0x0
47  12.14650536 [DBGPROT] HookedMmCopyVirtualMemory Called -> Result: 0x0
48  12.14651012 [DBGPROT] HookedPsLookupProcessByProcessId Called with 4924 and FFFF800882979330 -> Result: 0x0
49  12.14651203 [DBGPROT] HookedMmCopyVirtualMemory Called -> Result: 0x0
```

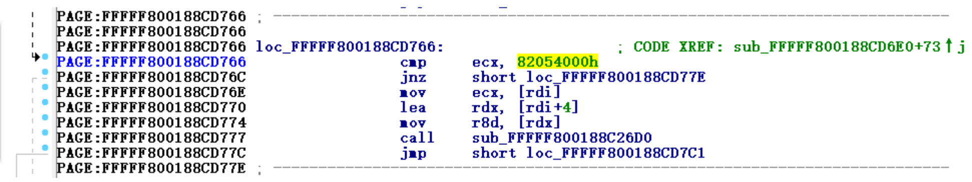
NT_SUCCESS

Enumerate Process Modules

The driver has a lots of commands that make us advantage.
In this case, we are able to enumerate modules that loaded in the target process by process id and a number which specifies we want to get.

I'll explain herewith below how I made it managed to work it with reverse engineering.
The implementation can be found at mhyprot.cpp#L343.

First of all, As you can see there is `cmp ecx, 82054000h` as I defined in mhyprot.hpp as `MHYPROT_IOCTL_ENUM_PROCESS_MODULES`.



```
PAGE:FFFFF800188CD766 ; ---------------------------------------------------------------------------
PAGE:FFFFF800188CD766
PAGE:FFFFF800188CD766 loc_FFFFF800188CD766:                   ; CODE XREF: sub_FFFFF800188CD6E0+73↑j
PAGE:FFFFF800188CD766                 cmp     ecx, 82054000h
PAGE:FFFFF800188CD76C                 jnz     short loc_FFFFF800188CD77E
PAGE:FFFFF800188CD76E                 mov     ecx, [rdi]
PAGE:FFFFF800188CD770                 lea     rdx, [rdi+4]
PAGE:FFFFF800188CD774                 mov     r8d, [rdx]
PAGE:FFFFF800188CD777                 call    sub_FFFFF800188C26D0
PAGE:FFFFF800188CD77C                 jmp     short loc_FFFFF800188CD7C1
PAGE:FFFFF800188CD77E ; ---------------------------------------------------------------------------
```

And it calls:

```
__int64 __fastcall sub_FFFFF800188C26D0(unsigned int a1, __int64 a2, __int64 a3)
{
  __int64 v3; // rsi
  unsigned int v4; // ebx
  bool v5; // di
  unsigned int v7; // ebx
  PVOID Object; // [rsp+58h] [rbp+20h]

  v3 = a2;
  Object = 0i64;
  v4 = a3;
  v5 = (int)PsLookupProcessByProcessId(a1, &Object, a3) >= 0;
  if ( !Object )
    return 0i64;
  v7 = sub_FFFFF800188C27D4(Object, v3, v4);
  if ( Object )
  {
    if ( v5 )
      ObfDereferenceObject(Object);
  }
  return v7;
}
```

As you can see, the function checks is process `32-bit` or `64-bit` by `PsGetProcessWow64Process()` since `PEB` is different between 32 and 64-bit processes.
In this case, I only talk about for 64-bit process.

After that, the function attaches from kernel using `KeStackAttachProcess` . the second parameter is `PKAPC_STATE` .
Then, call `PsGetProcessPeb` and get the PEB belongs to the target process.

`LDR_MODULE` is undocumented structure.

```
typedef struct _LDR_MODULE {
  LIST_ENTRY              InLoadOrderModuleList;
  LIST_ENTRY              InMemoryOrderModuleList;
  LIST_ENTRY              InInitializationOrderModuleList;
  PVOID                   BaseAddress;
  PVOID                   EntryPoint;
  ULONG                   SizeOfImage;
  UNICODE_STRING          FullDllName;
  UNICODE_STRING          BaseDllName;
  ULONG                   Flags;
  SHORT                   LoadCount;
  SHORT                   TlsIndex;
  LIST_ENTRY              HashTableEntry;
  ULONG                   TimeDateStamp;
} LDR_MODULE, *PLDR_MODULE;
```

And the function pseudocode for `sub_FFFFF800188C27D4` is like:

```
__int64 __fastcall sub_FFFFF800188C27D4(
  __int64 a1,       // pEPROCESS
  __int64 a2,       // pointer to the buffer that sent from usermode
  unsigned int a3   // max count to get
)
{
  ...

  if ( !a1 )
    return 0i64;

  v9 = ((__int64 (*)(void))PsGetProcessWow64Process)() != 0;
  KeStackAttachProcess(v5, &v30);

  if ( !v9 ) // the process is 64-bit
  {
    v17 = PsGetProcessPeb(v5); // Lookup PEB
    v18 = v17;
    if ( v17 )
    {
      v19 = *(_QWORD *)(v17 + 24); // PEB->Ldr
      if ( v19 )
      {
        for ( j = *(__int64 **)(v19 + 16);
              j != (__int64 *)(*(_QWORD *)(v18 + 24) + 16i64); // PEB->Ldr->InMemoryOrderModuleList.Flink
              j = (__int64 *)*j )
        {
          if ( v7 < v3 ) // if the counter less than a number what we want to get
          {
            v21 = 928i64 * v7; // [IMPORTANT] we can see output structure is 0x3A0 alignment
            sub_FFFFF800188C7900(v21 + v4 + 12, 0i64, 256i64); // fill memory by 0 sizeof 0x100
            sub_FFFFF800188C7900(v21 + v4 + 268, 0i64, 520i64); // fill memory by 0 sizeof 0x208
            *(_QWORD *)(v21 + v4) = j[6];
            *(_DWORD *)(v21 + v4 + 8) = *((_DWORD *)j + 16);
            v22 = *((_WORD *)j + 44);
            v23 = 127i64;
            if ( v22 <= 0x7Fu )
              v23 = v22;
            sub_FFFFF800188C75C0(v21 + v4 + 12, j[12], v23); // copy BaseDllName to the buffer
            v24 = *((_WORD *)j + 36);
            v25 = v24;
            if ( v24 > 0x103u )
              v25 = 259i64;
            sub_FFFFF800188C75C0(v21 + v4 + 268, j[10], v25); // copy FullDllName to the buffer
            *(_QWORD *)(v21 + v4 + 792) = *((unsigned int *)j + 32);
            v3 = v32;
          }
          ++v6; // counter
          ++v7; // counter
        }
      }
    }
  } else { ... /* 32-bit PEB (Redacted) */ }
  KeUnstackDetachProcess(&v30); // detach
  return v6;
}
```

We got a much information from it as follows:

- We can get `BaseDllName` and `FullDllName` using this ioctl command
- What we need is only `ProcessId` and `MaxCount`
- The output buffer will overrided in the request buffer
- The output buffer also must have `0x3A0` size alignment per module

Definition of structure for the payload be like: (This is defined in mhyprot.hpp as well.)

```
typedef struct _MHYPROT_ENUM_PROCESS_MODULES_REQUEST
{
    uint32_t process_id;
    uint32_t max_count;
} MHYPROT_ENUM_PROCESS_MODULES_REQUEST, * PMHYPROT_ENUM_PROCESS_MODULES_REQUEST;
```

By:

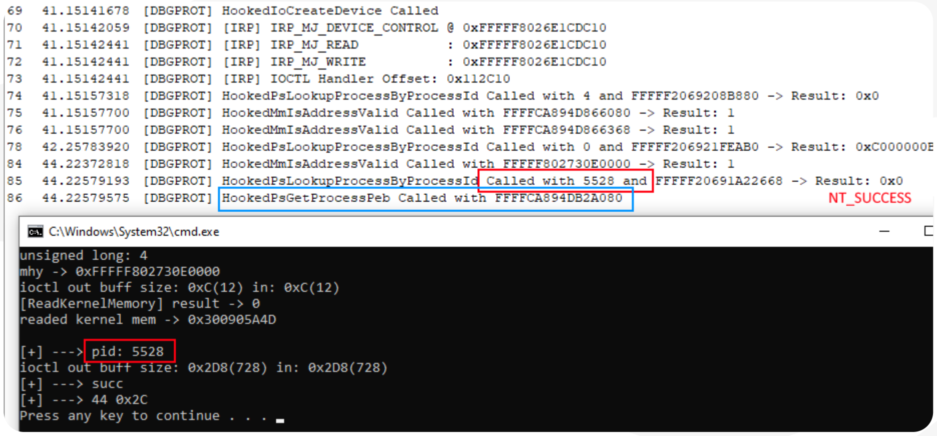
```
if (uVar1_ControlCode == 0x82054000) {
    uVar6 = GetModuleListByProcessId_FUN_000126d0
                // process id
        (*(uint *)puVar3_RequestContext,

                // out buffer, the output will be stored with overriding max count...
        (longlong)(uint *)((longlong)puVar3_RequestContext + 4),

                // max count
        *(uint *)((longlong)puVar3_RequestContext + 4)
        );
    iVar3 = (int)uVar6;
}
...
// mhyprot overrides first 4byte of the payload buffer to identify success or fail
*(int *)puVar3_RequestContext = iVar3;
```

What we need is:

- i. Allocate memory for payload and its result, `0x3A0` * `MaxCount`
- i. Send the payload with the ioctl code `0x82054000`
- i. Check for the first 4byte

# Proof

I've hooked some part of mhyprot kernel module, especially `PsGetProcessPEB` and `PsLookupProcessByProcessId` and confirmed.



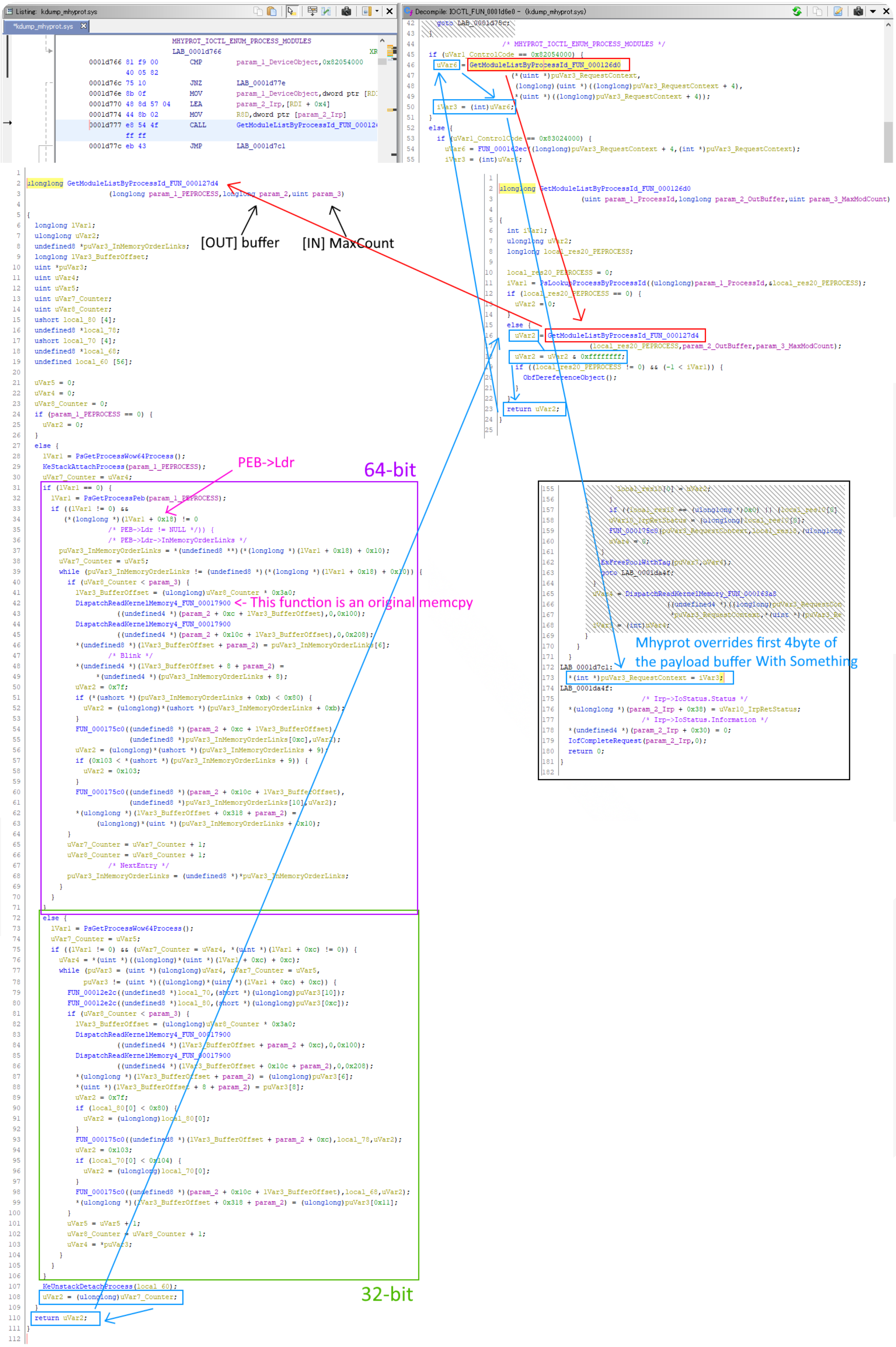
```
69   41.15141678  [DBGPROT] HookedIoCreateDevice Called
70   41.15142059  [DBGPROT] [IRP] IRP_MJ_DEVICE_CONTROL @ 0xFFFFF8026E1CDC10
71   41.15142441  [DBGPROT] [IRP] IRP_MJ_READ           : 0xFFFFF8026E1CDC10
72   41.15142441  [DBGPROT] [IRP] IRP_MJ_WRITE          : 0xFFFFF8026E1CDC10
73   41.15142441  [DBGPROT] [IRP] IOCTL Handler Offset: 0x112C10
74   41.15157318  [DBGPROT] HookedPsLookupProcessByProcessId Called with 4 and FFFFF2069208B880 -> Result: 0x0
75   41.15157700  [DBGPROT] HookedMmIsAddressValid Called with FFFFCA894D866080 -> Result: 1
76   41.15157700  [DBGPROT] HookedMmIsAddressValid Called with FFFFCA894D866368 -> Result: 1
78   42.25783920  [DBGPROT] HookedPsLookupProcessByProcessId Called with 0 and FFFFF206921FEAB0 -> Result: 0xC000000E
84   44.22372818  [DBGPROT] HookedMmIsAddressValid Called with FFFFF802730E0000 -> Result: 1
85   44.22579193  [DBGPROT] HookedPsLookupProcessByProcessId Called with 5528 and FFFFF20691A22668 -> Result: 0x0
86   44.22579575  [DBGPROT] HookedPsGetProcessPeb Called with FFFFCA894DB2A080
```

NT_SUCCESS

C:\Windows\System32\cmd.exe

```
unsigned long: 4
mhy -> 0xFFFFF802730E0000
ioctl out buff size: 0xC(12) in: 0xC(12)
[ReadKernelMemory] result -> 0
readed kernel mem -> 0x300905A4D

[+] ---> pid: 5528
ioctl out buff size: 0x2D8(728) in: 0x2D8(728)
[+] ---> succ
[+] ---> 44 0x2C
Press any key to continue . . .
```

# Call map



Listing: kdump_mhyprot.sys
*kdump_mhyprot.sys
Decompile: IOCTL_FUN_0001d6e0 - (kdump_mhyprot.sys)
[OUT] buffer
[IN] MaxCount
PEB->Ldr
64-bit
<- This function is an original memcpy
32-bit
Mhyprot overrides first 4byte of the payload buffer With Something

```
                         MHYPROT_IOCTL_ENUM_PROCESS_MODULES
                         LAB_0001d766                                    XR
    0001d766 81 f9 00        CMP        param_1_DeviceObject,0x82054000
             40 05 82
    0001d76c 75 10           JNZ        LAB_0001d77e
    0001d76e 8b 0f           MOV        param_1_DeviceObject,dword ptr [RDI
    0001d770 48 8d 57 04     LEA        param_2_Irp,[RDI + 0x4]
    0001d774 44 8b 02        MOV        R8D,dword ptr [param_2_Irp]
    0001d777 e8 54 4f        CALL       GetModuleListByProcessId_FUN_00012
             ff ff
    0001d77c eb 43           JMP        LAB_0001d7c1
```

```
    goto LAB_0001d75c;
  }
                    /* MHYPROT_IOCTL_ENUM_PROCESS_MODULES */
  if (uVar1_ControlCode == 0x82054000) {
    uVar6 = GetModuleListByProcessId_FUN_000126d0
                      (*(uint *)puVar3_RequestContext,
                       (longlong)(uint *)((longlong)puVar3_RequestContext + 4),
                       *(uint *)((longlong)puVar3_RequestContext + 4));
    iVar3 = (int)uVar6;
  }
  else {
    if (uVar1_ControlCode == 0x83024000) {
      uVar6 = FUN_000162ec((longlong)puVar3_RequestContext + 4,(int *)puVar3_RequestContext);
      iVar3 = (int)uVar6;
```

```

ulonglong GetModuleListByProcessId_FUN_000127d4
                    (longlong param_1_PEPROCESS,longlong param_2,uint param_3)

{
  longlong lVar1;
  ulonglong uVar2;
  undefined8 *puVar3_InMemoryOrderLinks;
  longlong lVar3_BufferOffset;
  uint *puVar3;
  uint uVar4;
  uint uVar5;
  uint uVar7_Counter;
  uint uVar8_Counter;
  ushort local_80 [4];
  undefined8 *local_78;
  ushort local_70 [4];
  undefined8 *local_68;
  undefined local_60 [56];

  uVar5 = 0;
  uVar4 = 0;
  uVar8_Counter = 0;
  if (param_1_PEPROCESS == 0) {
    uVar2 = 0;
  }
  else {
    lVar1 = PsGetProcessWow64Process();
    KeStackAttachProcess(param_1_PEPROCESS);
    uVar7_Counter = uVar4;
    if (lVar1 == 0) {
      lVar1 = PsGetProcessPeb(param_1_PEPROCESS);
      if ((lVar1 != 0) &&
         (*(longlong *)(lVar1 + 0x18) != 0
                    /* PEB->Ldr != NULL */)) {
                    /* PEB->Ldr->InMemoryOrderLinks */
        puVar3_InMemoryOrderLinks = *(undefined8 **)(*(longlong *)(lVar1 + 0x18) + 0x10);
        uVar7_Counter = uVar5;
        while (puVar3_InMemoryOrderLinks != (undefined8 *)(*(longlong *)(lVar1 + 0x18) + 0x10)) {
          if (uVar8_Counter < param_3) {
            lVar3_BufferOffset = (ulonglong)uVar8_Counter * 0x3a0;
            DispatchReadKernelMemory4_FUN_00017900
                      ((undefined4 *)(param_2 + 0xc + lVar3_BufferOffset),0,0x100);
            DispatchReadKernelMemory4_FUN_00017900
                      ((undefined4 *)(param_2 + 0x10c + lVar3_BufferOffset),0,0x208);
            *(undefined8 *)(lVar3_BufferOffset + param_2) = puVar3_InMemoryOrderLinks[6];
                    /* Blink */
            *(undefined4 *)(lVar3_BufferOffset + 8 + param_2) =
                 *(undefined4 *)(puVar3_InMemoryOrderLinks + 8);
            uVar2 = 0x7f;
            if (*(ushort *)(puVar3_InMemoryOrderLinks + 0xb) < 0x80) {
              uVar2 = (ulonglong)*(ushort *)(puVar3_InMemoryOrderLinks + 0xb);
            }
            FUN_000175c0((undefined8 *)(param_2 + 0xc + lVar3_BufferOffset),
                         (undefined8 *)puVar3_InMemoryOrderLinks[0xc],uVar2);
            uVar2 = (ulonglong)*(ushort *)(puVar3_InMemoryOrderLinks + 9);
            if (0x103 < *(ushort *)(puVar3_InMemoryOrderLinks + 9)) {
              uVar2 = 0x103;
            }
            FUN_000175c0((undefined8 *)(param_2 + 0x10c + lVar3_BufferOffset),
                         (undefined8 *)puVar3_InMemoryOrderLinks[10],uVar2);
            *(ulonglong *)(lVar3_BufferOffset + 0x318 + param_2) =
                 (ulonglong)*(uint *)(puVar3_InMemoryOrderLinks + 0x10);
          }
          uVar7_Counter = uVar7_Counter + 1;
          uVar8_Counter = uVar8_Counter + 1;
                    /* NextEntry */
          puVar3_InMemoryOrderLinks = (undefined8 *)*puVar3_InMemoryOrderLinks;
        }
      }
    }
    else {
      lVar1 = PsGetProcessWow64Process();
      uVar7_Counter = uVar5;
      if ((lVar1 != 0) && (uVar7_Counter = uVar4, *(uint *)(lVar1 + 0xc) != 0)) {
        uVar4 = *(uint *)((ulonglong)*(uint *)(lVar1 + 0xc) + 0xc);
        while (puVar3 = (uint *)(ulonglong)uVar4, uVar7_Counter = uVar5,
              puVar3 != (uint *)((ulonglong)*(uint *)(lVar1 + 0xc) + 0xc)) {
          FUN_00012e2c((undefined8 *)local_70,(short *)(ulonglong)puVar3[10]);
          FUN_00012e2c((undefined8 *)local_80,(short *)(ulonglong)puVar3[0xc]);
          if (uVar8_Counter < param_3) {
            lVar3_BufferOffset = (ulonglong)uVar8_Counter * 0x3a0;
            DispatchReadKernelMemory4_FUN_00017900
                      ((undefined4 *)(lVar3_BufferOffset + param_2 + 0xc),0,0x100);
            DispatchReadKernelMemory4_FUN_00017900
                      ((undefined4 *)(lVar3_BufferOffset + 0x10c + param_2),0,0x208);
            *(ulonglong *)(lVar3_BufferOffset + param_2) = (ulonglong)puVar3[6];
            *(uint *)(lVar3_BufferOffset + 8 + param_2) = puVar3[8];
            uVar2 = 0x7f;
            if (local_80[0] < 0x80) {
              uVar2 = (ulonglong)local_80[0];
            }
            FUN_000175c0((undefined8 *)(lVar3_BufferOffset + param_2 + 0xc),local_78,uVar2);
            uVar2 = 0x103;
            if (local_70[0] < 0x104) {
              uVar2 = (ulonglong)local_70[0];
            }
            FUN_000175c0((undefined8 *)(param_2 + 0x10c + lVar3_BufferOffset),local_68,uVar2);
            *(ulonglong *)(lVar3_BufferOffset + 0x318 + param_2) = (ulonglong)puVar3[0x11];
          }
          uVar5 = uVar5 + 1;
          uVar8_Counter = uVar8_Counter + 1;
          uVar4 = *puVar3;
        }
      }
    }
    KeUnstackDetachProcess(local_60);
    uVar2 = (ulonglong)uVar7_Counter;
  }
  return uVar2;
}

```

```

ulonglong GetModuleListByProcessId_FUN_000126d0
                    (uint param_1_ProcessId,longlong param_2_OutBuffer,uint param_3_MaxModCount)

{
  int iVar1;
  ulonglong uVar2;
  longlong local_res20_PEPROCESS;

  local_res20_PEPROCESS = 0;
  iVar1 = PsLookupProcessByProcessId((ulonglong)param_1_ProcessId,&local_res20_PEPROCESS);
  if (local_res20_PEPROCESS == 0) {
    uVar2 = 0;
  }
  else {
    uVar2 = GetModuleListByProcessId_FUN_000127d4
                      (local_res20_PEPROCESS,param_2_OutBuffer,param_3_MaxModCount);
    uVar2 = uVar2 & 0xffffffff;
    if ((local_res20_PEPROCESS != 0) && (-1 < iVar1)) {
      ObfDereferenceObject();
    }
  }
  return uVar2;
}

```

```
            local_res10[0] = uVar2;
          }
          if ((local_res18 == (ulonglong *)0x0) || (local_res10[0]
          uVar10_IrpRetStatus = (ulonglong)local_res10[0];
          FUN_000175c0(puVar3_RequestContext,local_res18,(ulonglong
          uVar4 = 0;
        }
        ExFreePoolWithTag(puVar7,uVar4);
        goto LAB_0001da4f;
      }
      uVar4 = DispatchReadKernelMemory_FUN_000163a8
                        ((undefined4 *)((longlong)puVar3_RequestCon
                         *puVar3_RequestContext,*(uint *)(puVar3_Re
      iVar3 = (int)uVar4;
    }
  }
}
LAB_0001d7c1:
  *(int *)puVar3_RequestContext = iVar3;
LAB_0001da4f:
                    /* Irp->IoStatus.Status */
  *(ulonglong *)(param_2_Irp + 0x38) = uVar10_IrpRetStatus;
                    /* Irp->IoStatus.Information */
  *(undefined4 *)(param_2_Irp + 0x30) = 0;
  IofCompleteRequest(param_2_Irp,0);
  return 0;
}

```

Enumerate Process Threads

The driver has a vulnerable ioctl that allows us to enumerate threads in specific process as ring-0 privilege. it also make us able to read kernel structure `PETHREAD` because the ioctl result contains a pointer to it.

to read kernel memory, we are already able to do it through this vulnerable driver as well.

I'll explain how I made managed to work it with reverse engineering.

First of all, the driver has a function that executes `ZwQuerySystemInformation` .
Here is a block found on ioctl handler subroutine (is in the encryption-dedicated IOCTL handler function):

```
PAGE:FFFFF800188CD77E loc_FFFFF800188CD77E:                   ; CODE XREF: sub_FFFFF800188CD6E0+8C↑j
PAGE:FFFFF800188CD77E                 cmp     ecx, 83024000h
PAGE:FFFFF800188CD784                 jnz     short loc_FFFFF800188CD794
PAGE:FFFFF800188CD786                 lea     rcx, [rdi+4]
PAGE:FFFFF800188CD78A                 mov     rdx, rdi
PAGE:FFFFF800188CD78D                 call    sub_FFFFF800188C62EC
PAGE:FFFFF800188CD792                 jmp     short loc_FFFFF800188CD7C1
```

`sub_FFFFF800188C62EC` is:

```
__int64 __fastcall sub_FFFFF800188C62EC(__int64 a1, _DWORD *a2)
{
  __int64 result; // rax

  if ( *a2 == 136 ) // *a2 == 0x88
    result = sub_FFFFF800188C6488(a2[2], a1, a2[1]);
  else
    result = 0xFFFFFFFFi64;
  return result;
}
```

We are seeing an if statement `if ( *a2 == 136 )` , `136` is `0x88` , if the a2(given by context) is not `0x88` , the driver will returns `0xFFFFFFFF` .
I have no idea what is this validation is even I finished looking around it for a while...

Also `sub_FFFFF800188C6488` is:

```
__int64 __fastcall sub_FFFFF800188C6488(int a1, __int64 a2_OutBuffer, unsigned int a3_ProcessId)
{
  v3_OutBuffer = a2_OutBuffer;
  v4 = a1;
  v5 = a3_ProcessId;
  v6 = -1;
  RtlInitUnicodeString(&SystemRoutineName, L"ZwQuerySystemInformation");
  v7_pZwQuerySystemInformation = (int (__fastcall *)(__int64, __m128 *, _QWORD, SIZE_T *))MmGetSystemRoutineAddress(&SystemRoutineName);
  v8_pZwQuerySystemInformation = v7_pZwQuerySystemInformation;
  if ( v7_pZwQuerySystemInformation )
  {
    LODWORD(NumberOfBytes) = 0;
    if ( v7_pZwQuerySystemInformation(5i64, 0i64, 0i64, &NumberOfBytes) == -1073741820 )// SystemProcessInformation
    {
      if ( (_DWORD)NumberOfBytes )
      {
        v9 = (__m128 *)ExAllocatePool(NonPagedPool, (unsigned int)NumberOfBytes);
        v10_ProcInfo = v9;
        if ( v9 )
        {
          RKM_sub_FFFFF800188C7900(v9, 0, (unsigned int)NumberOfBytes);// fill the memory by 0
          if ( v8_pZwQuerySystemInformation(5i64, v10_ProcInfo, (unsigned int)NumberOfBytes, &NumberOfBytes) >= 0 )// SystemProcessInformation
          {
            v11_ProcInfo = v10_ProcInfo;
            while ( (unsigned __int8)MmIsAddressValid(v11_ProcInfo) )
            {
              if ( v11_ProcInfo[5].m128_i32[0] == v4 )
              {
                v6 = HIDWORD(v11_ProcInfo->m128_u64[0]);
                if ( v6 <= v5 )                 // == ProcessId
                {
                  v13 = 0i64;
                  if ( v6 )
                  {
                    v14 = v3_OutBuffer + 8;     // data offset per item
                    v15 = v11_ProcInfo + 19;
                    do
                    {
                      v16 = v15->m128_u64[0];
                      Object_PETHREAD = 0i64;
                      PsLookupThreadByThreadId(v16, &Object_PETHREAD);
                      v17_PETHREAD = Object_PETHREAD;
                      v18_PETHREAD = Object_PETHREAD;
                      *(_DWORD *)(v14 - 8) = v11_ProcInfo[5].m128_i32[0];
                      *(_DWORD *)(v14 - 4) = v16;
                      *(_QWORD *)v14 = v18_PETHREAD;// set a pointer to the this PETHREAD
                      *(_QWORD *)(v14 + 8) = GetThreadStartAddress_sub_FFFFF800188C68C8((__int64)v18_PETHREAD);// set thread start address
                      *(_QWORD *)(v14 + 16) = sub_FFFFF800188C687C((__int64)v17_PETHREAD);// this actually return PETHREAD+0x400
                      *(_DWORD *)(v14 + 24) = sub_FFFFF800188C67F4((__int64)v17_PETHREAD) != 0;// unknown, bool
                      if ( v17_PETHREAD )
                        ObfDereferenceObject(v17_PETHREAD);
                      ++v13;
                      v15 += 5;
                      v14 += 168i64;            // 0xA8 alignment
                    }
                    while ( v13 < HIDWORD(v11_ProcInfo->m128_u64[0]) ); SYSTEM_PROCESS_INFORMATION->Threads
                  }
                }
                break;
              }
              v12 = LODWORD(v11_ProcInfo->m128_u64[0]);
              if ( (_DWORD)v12 )
              {
                v11_ProcInfo = (__m128 *)((char *)v11_ProcInfo + v12);
                if ( v11_ProcInfo )
                  continue;
              }
              break;
            }
            ExFreePoolWithTag(v10_ProcInfo, 0);
          }
        }
      }
    }
  }
  return v6;
}
```

As the pseudocode says, subroutine does:

- i. Get the pointer to the `ZwQuerySystemInformation` by `MmGetSystemRoutineAddress`
- i. Call `ZwQuerySystemInformation` with `SystemProcessInformation` to get pool size what we have to allocate. (bad implementation)
- i. Allocate memory using `ExAllocatePool` with the size
- i. Call `ZwQuerySystemInformation` again to enumerate processes
- i. Enumerate for every single processes and making sure the address is valid by `MmIsAddressValid`
    - If the process id is match, call `PsLookupThreadByThreadId` to get `PETHREAD` by thread id, then write information into the payload buffer, every single threads.

Also:

- The output data structure is `0xA8` alignment
- We can get its thread's start address by `sub_FFFFF800188C68C8`
- We can get its thread's `PETHREAD` address in the kernel

So I don't know what `sub_FFFFF800188C687C` and `sub_FFFFF800188C67F4` does.
only one thing I know is that the first one references `PETHREAD+0x400` as follows:

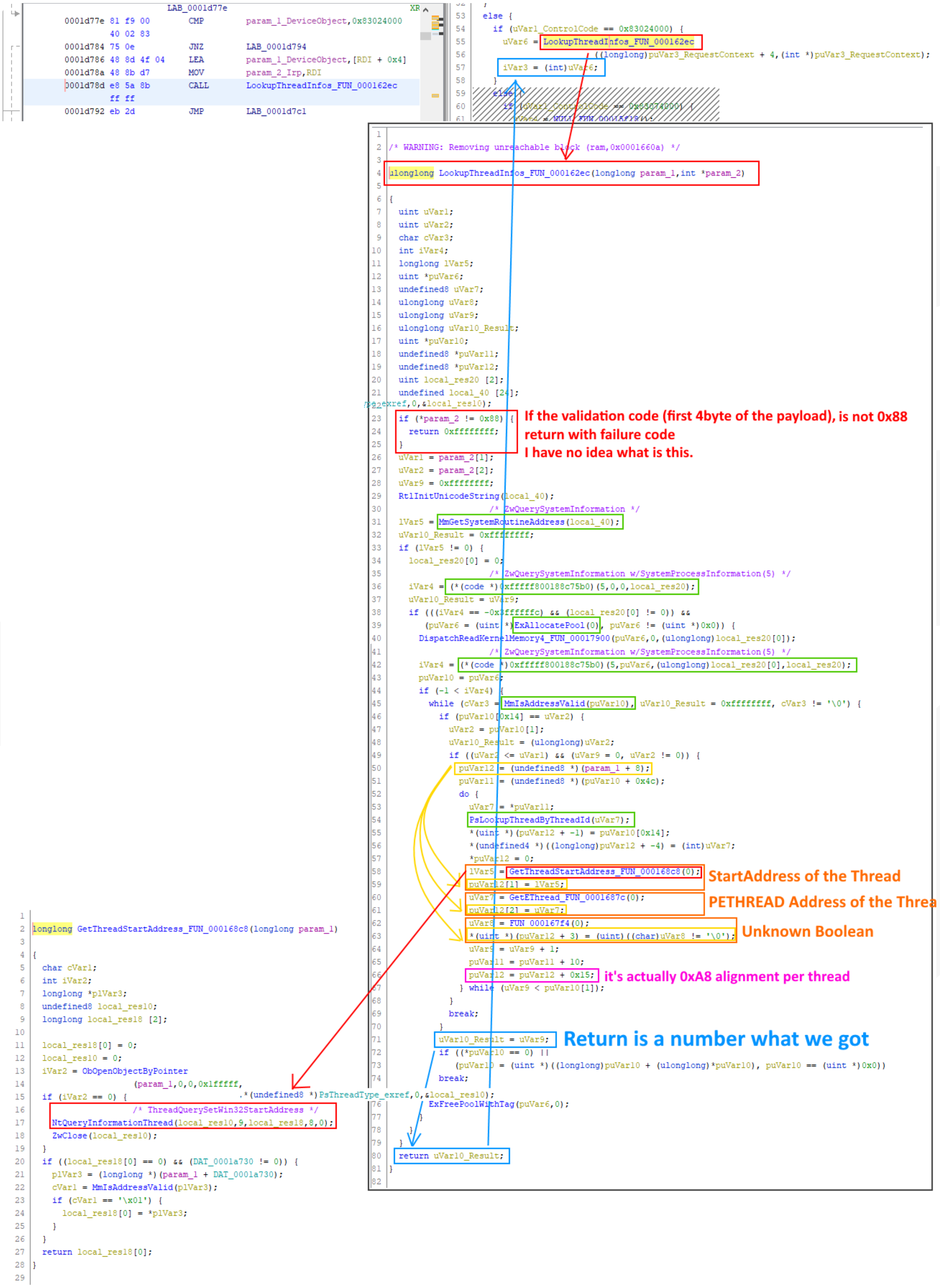
```
__int64 __fastcall sub_FFFFF800188C687C(__int64 a1_PETHREAD)
{
  __int64 v1; // rbx
  __int64 v2_PETHREAD; // rdi
  __int64 *v4; // rdi

  v1 = 0i64;
  v2_PETHREAD = a1_PETHREAD;
  if ( !qword_FFFFF800188CA728 )
    return 0i64;
  if ( (unsigned __int8)MmIsAddressValid(a1_PETHREAD) == 1 )
  {
    v4 = (__int64 *)(qword_FFFFF800188CA728 + v2_PETHREAD);// 1048i64 + v2_PETHREAD, winver depends
    if ( (unsigned __int8)MmIsAddressValid(v4) == 1 )
      v1 = *v4;
  }
  return v1;
}
```

`qword_FFFFF800188CA728` is an static variable which has a winver-depends offset for the struct member.
Confirmed by this subroutine:
As you can see the switch-case is winver.

```
bool __fastcall sub_FFFFF800188C70CC(__int64 a1, __int64 a2, __int64 a3)
{
  char v4; // [rsp+20h] [rbp-128h]
  unsigned int v5; // [rsp+2Ch] [rbp-11Ch]
  __int64 v6; // [rsp+150h] [rbp+8h]

  switch ( dword_FFFFF800188CA748 )
  {
    case 61:
      qword_FFFFF800188CA700 = 384i64;
      qword_FFFFF800188CA708 = 360i64;
      qword_FFFFF800188CA710 = 496i64;
      qword_FFFFF800188CA720 = 512i64;
      qword_FFFFF800188CA718 = 616i64;
      qword_FFFFF800188CA728 = 872i64;
      qword_FFFFF800188CA730 = 1048i64; // <-
      qword_FFFFF800188CA738 = 1104i64;
      qword_FFFFF800188CA740 = 736i64;
      break;
    case 62:
      qword_FFFFF800188CA730 = 1008i64; // <-
      qword_FFFFF800188CA738 = 1068i64;
      goto LABEL_15;
    case 63:
      qword_FFFFF800188CA730 = 1656i64; // <-
      qword_FFFFF800188CA738 = 1716i64;
LABEL_15:
      qword_FFFFF800188CA718 = 936i64;
      qword_FFFFF800188CA720 = 1032i64;
      qword_FFFFF800188CA710 = 1040i64;
      qword_FFFFF800188CA700 = 736i64;
      qword_FFFFF800188CA740 = 768i64;
      break;
    case 100:
      RtlGetVersion(&v4);
      if ( v5 >= 0x4A61 )
      {
        qword_FFFFF800188CA730 = 0i64; // <-
        qword_FFFFF800188CA700 = 1088i64;
        qword_FFFFF800188CA710 = 1400i64;
        qword_FFFFF800188CA718 = 1304i64;
        qword_FFFFF800188CA720 = 1392i64;
        qword_FFFFF800188CA708 = 1128i64;
        qword_FFFFF800188CA738 = 1296i64;
LABEL_9:
        qword_FFFFF800188CA740 = 912i64;
        break;
      }
      qword_FFFFF800188CA710 = 1056i64;
      qword_FFFFF800188CA720 = 1048i64;
      if ( v5 >= 0x47BA )
      {
        qword_FFFFF800188CA700 = 744i64;
        qword_FFFFF800188CA718 = 960i64;
        qword_FFFFF800188CA708 = 784i64;
        qword_FFFFF800188CA730 = 1696i64; // <-
        qword_FFFFF800188CA738 = 1760i64;
        goto LABEL_9;
      }
      qword_FFFFF800188CA718 = 952i64;
      qword_FFFFF800188CA740 = 904i64;
      if ( v5 < 0x3AD7 )
      {
        qword_FFFFF800188CA700 = 744i64;
        qword_FFFFF800188CA730 = 1672i64; // <-
        qword_FFFFF800188CA738 = 1728i64;
```

```
            }
            else
            {
              qword_FFFFF800188CA700 = 736i64;
              qword_FFFFF800188CA708 = 776i64;
              qword_FFFFF800188CA730 = 1680i64; // <-
              qword_FFFFF800188CA738 = 1744i64;
            }
            break;
        }
        v6 = 0i64;
        PsLookupProcessByProcessId(4i64, &v6, a3);
        return sub_FFFFF800188C3D08(v6) == 4;
      }
```

# Call map



```
                             LAB_0001d77e
        0001d77e 81 f9 00        CMP        param_1_DeviceObject,0x83024000
                 40 02 83
        0001d784 75 0e           JNZ        LAB_0001d794
        0001d786 48 8d 4f 04     LEA        param_1_DeviceObject,[RDI + 0x4]
        0001d78a 48 8b d7        MOV        param_2_Irp,RDI
        0001d78d e8 5a 8b        CALL       LookupThreadInfos_FUN_000162ec
                 ff ff
        0001d792 eb 2d           JMP        LAB_0001d7c1
```

```
  else {
    if (uVar1_ControlCode == 0x83024000) {
      uVar6 = LookupThreadInfos_FUN_000162ec
                        ((longlong)puVar3_RequestContext + 4,(int *)puVar3_RequestContext);
      iVar3 = (int)uVar6;
    }
    else {
      if (uVar1_ControlCode == 0x83074000) {
```

```
/* WARNING: Removing unreachable block (ram,0x0001660a) */

ulonglong LookupThreadInfos_FUN_000162ec(longlong param_1,int *param_2)

{
  uint uVar1;
  uint uVar2;
  char cVar3;
  int iVar4;
  longlong lVar5;
  uint *puVar6;
  undefined8 uVar7;
  ulonglong uVar8;
  ulonglong uVar9;
  ulonglong uVar10_Result;
  uint *puVar10;
  undefined8 *puVar11;
  undefined8 *puVar12;
  uint local_res20 [2];
  undefined local_40 [24];
pe_exref,0,&local_res10);
  if (*param_2 != 0x88) {
    return 0xffffffff;
  }
  uVar1 = param_2[1];
  uVar2 = param_2[2];
  uVar9 = 0xffffffff;
  RtlInitUnicodeString(local_40);
                    /* ZwQuerySystemInformation */
  lVar5 = MmGetSystemRoutineAddress(local_40);
  uVar10_Result = 0xffffffff;
  if (lVar5 != 0) {
    local_res20[0] = 0;
                    /* ZwQuerySystemInformation w/SystemProcessInformation(5) */
    iVar4 = (*(code *)0xfffff800188c75b0)(5,0,0,local_res20);
    uVar10_Result = uVar9;
    if (((iVar4 == -0x3ffffffc) && (local_res20[0] != 0)) &&
       (puVar6 = (uint *)ExAllocatePool(0), puVar6 != (uint *)0x0)) {
      DispatchReadKernelMemory4_FUN_00017900(puVar6,0,(ulonglong)local_res20[0]);
                    /* ZwQuerySystemInformation w/SystemProcessInformation(5) */
      iVar4 = (*(code *)0xfffff800188c75b0)(5,puVar6,(ulonglong)local_res20[0],local_res20);
      puVar10 = puVar6;
      if (-1 < iVar4) {
        while (cVar3 = MmIsAddressValid(puVar10), uVar10_Result = 0xffffffff, cVar3 != '\0') {
          if (puVar10[0x14] == uVar2) {
            uVar2 = puVar10[1];
            uVar10_Result = (ulonglong)uVar2;
            if ((uVar2 <= uVar1) && (uVar9 = 0, uVar2 != 0)) {
              puVar12 = (undefined8 *)(param_1 + 8);
              puVar11 = (undefined8 *)(puVar10 + 0x4c);
              do {
                uVar7 = *puVar11;
                PsLookupThreadByThreadId(uVar7);
                *(uint *)(puVar12 + -1) = puVar10[0x14];
                *(undefined4 *)((longlong)puVar12 + -4) = (int)uVar7;
                *puVar12 = 0;
                lVar5 = GetThreadStartAddress_FUN_000168c8(0);
                puVar12[1] = lVar5;
                uVar7 = GetEThread_FUN_0001687c(0);
                puVar12[2] = uVar7;
                uVar8 = FUN_000167f4(0);
                *(uint *)(puVar12 + 3) = (uint)((char)uVar8 != '\0');
                uVar9 = uVar9 + 1;
                puVar11 = puVar11 + 10;
                puVar12 = puVar12 + 0x15;
              } while (uVar9 < puVar10[1]);
            }
            break;
          }
          uVar10_Result = uVar9;
          if ((*puVar10 == 0) ||
             (puVar10 = (uint *)((longlong)puVar10 + (ulonglong)*puVar10), puVar10 == (uint *)0x0))
          break;
        }
      }
      ExFreePoolWithTag(puVar6,0);
    }
  }
  return uVar10_Result;
}
```

If the validation code (first 4byte of the payload), is not 0x88
return with failure code
I have no idea what is this.

StartAddress of the Thread

PETHREAD Address of the Threa

Unknown Boolean

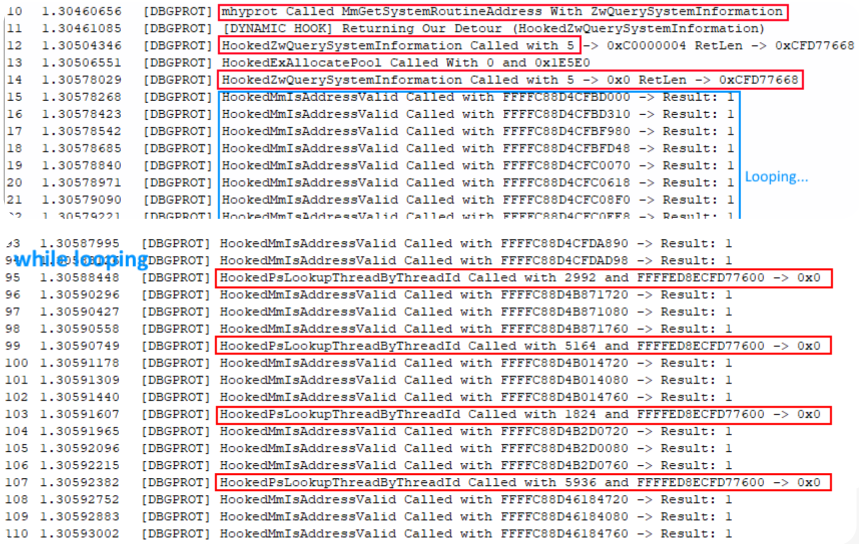
it's actually 0xA8 alignment per thread

Return is a number what we got

```
longlong GetThreadStartAddress_FUN_000168c8(longlong param_1)

{
  char cVar1;
  int iVar2;
  longlong *plVar3;
  undefined8 local_res10;
  longlong local_res18 [2];

  local_res18[0] = 0;
  local_res10 = 0;
  iVar2 = ObOpenObjectByPointer
                    (param_1,0,0,0x1fffff,
                     .*(undefined8 *)PsThreadType_exref,0,&local_res10);
  if (iVar2 == 0) {
                    /* ThreadQuerySetWin32StartAddress */
    NtQueryInformationThread(local_res10,9,local_res18,8,0);
    ZwClose(local_res10);
  }
  if ((local_res18[0] == 0) && (DAT_0001a730 != 0)) {
    plVar3 = (longlong *)(param_1 + DAT_0001a730);
    cVar1 = MmIsAddressValid(plVar3);
    if (cVar1 == '\x01') {
      local_res18[0] = *plVar3;
    }
  }
  return local_res18[0];
}
```

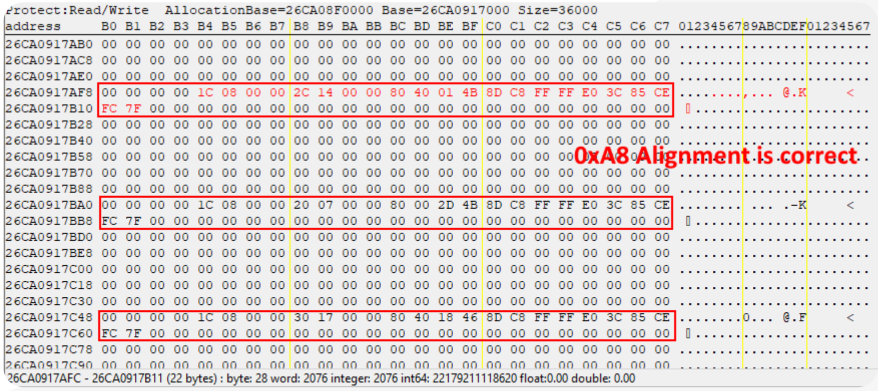
# Proof

Confirmed by hooking mhyprot kernel module.
System-calls are properly called exactly same as the pseudocode:



```
10   1.30460656   [DBGPROT] mhyprot Called MmGetSystemRoutineAddress With ZwQuerySystemInformation
11   1.30461085   [DBGPROT] [DYNAMIC HOOK] Returning Our Detour (HookedZwQuerySystemInformation)
12   1.30504346   [DBGPROT] HookedZwQuerySystemInformation Called with 5 -> 0xC0000004 RetLen -> 0xCFD77668
13   1.30506551   [DBGPROT] HookedExAllocatePool Called With 0 and 0x1E5E0
14   1.30578029   [DBGPROT] HookedZwQuerySystemInformation Called with 5 -> 0x0 RetLen -> 0xCFD77668
15   1.30578268   [DBGPROT] HookedMmIsAddressValid Called with FFFFC88D4CFBD000 -> Result: 1
16   1.30578423   [DBGPROT] HookedMmIsAddressValid Called with FFFFC88D4CFBD310 -> Result: 1
17   1.30578542   [DBGPROT] HookedMmIsAddressValid Called with FFFFC88D4CFBF980 -> Result: 1
18   1.30578685   [DBGPROT] HookedMmIsAddressValid Called with FFFFC88D4CFBFD48 -> Result: 1
19   1.30578840   [DBGPROT] HookedMmIsAddressValid Called with FFFFC88D4CFC0070 -> Result: 1
20   1.30578971   [DBGPROT] HookedMmIsAddressValid Called with FFFFC88D4CFC0618 -> Result: 1
21   1.30579090   [DBGPROT] HookedMmIsAddressValid Called with FFFFC88D4CFC08F0 -> Result: 1
22   1.30579221   [DBGPROT] HookedMmIsAddressValid Called with FFFFC88D4CFC0FE8 -> Result: 1

93   1.30587995   [DBGPROT] HookedMmIsAddressValid Called with FFFFC88D4CFDA890 -> Result: 1
94   [illegible]  [DBGPROT] HookedMmIsAddressValid Called with FFFFC88D4CFDAD98 -> Result: 1
95   1.30588448   [DBGPROT] HookedPsLookupThreadByThreadId Called with 2992 and FFFFED8ECFD77600 -> 0x0
96   1.30590296   [DBGPROT] HookedMmIsAddressValid Called with FFFFC88D4B871720 -> Result: 1
97   1.30590427   [DBGPROT] HookedMmIsAddressValid Called with FFFFC88D4B871080 -> Result: 1
98   1.30590558   [DBGPROT] HookedMmIsAddressValid Called with FFFFC88D4B871760 -> Result: 1
99   1.30590749   [DBGPROT] HookedPsLookupThreadByThreadId Called with 5164 and FFFFED8ECFD77600 -> 0x0
100  1.30591178   [DBGPROT] HookedMmIsAddressValid Called with FFFFC88D4B014720 -> Result: 1
101  1.30591309   [DBGPROT] HookedMmIsAddressValid Called with FFFFC88D4B014080 -> Result: 1
102  1.30591440   [DBGPROT] HookedMmIsAddressValid Called with FFFFC88D4B014760 -> Result: 1
103  1.30591607   [DBGPROT] HookedPsLookupThreadByThreadId Called with 1824 and FFFFED8ECFD77600 -> 0x0
104  1.30591965   [DBGPROT] HookedMmIsAddressValid Called with FFFFC88D4B2D0720 -> Result: 1
105  1.30592096   [DBGPROT] HookedMmIsAddressValid Called with FFFFC88D4B2D0080 -> Result: 1
106  1.30592215   [DBGPROT] HookedMmIsAddressValid Called with FFFFC88D4B2D0760 -> Result: 1
107  1.30592382   [DBGPROT] HookedPsLookupThreadByThreadId Called with 5936 and FFFFED8ECFD77600 -> 0x0
108  1.30592752   [DBGPROT] HookedMmIsAddressValid Called with FFFFC88D46184720 -> Result: 1
109  1.30592883   [DBGPROT] HookedMmIsAddressValid Called with FFFFC88D46184080 -> Result: 1
110  1.30593002   [DBGPROT] HookedMmIsAddressValid Called with FFFFC88D46184760 -> Result: 1
```

Data is set by the driver, this is the memory view of payload buffer:



```
Protect:Read/Write  AllocationBase=26CA08F0000 Base=26CA0917000 Size=36000
address     B0 B1 B2 B3 B4 B5 B6 B7 B8 B9 BA BB BC BD BE BF C0 C1 C2 C3 C4 C5 C6 C7 0123456789ABCDEF01234567
26CA0917AB0 00 00 00 00 00 00 00 00 00 00 00 00 00 00 00 00 00 00 00 00 00 00 00 00 ........................
26CA0917AC8 00 00 00 00 00 00 00 00 00 00 00 00 00 00 00 00 00 00 00 00 00 00 00 00 ........................
26CA0917AE0 00 00 00 00 00 00 00 00 00 00 00 00 00 00 00 00 00 00 00 00 00 00 00 00 ........................
26CA0917AF8 00 00 00 00 1C 08 00 00 2C 14 00 00 80 40 01 4B 8D C8 FF FF E0 3C 85 CE ........,... @.K      <
26CA0917B10 FC 7F 00 00 00 00 00 00 00 00 00 00 00 00 00 00 00 00 00 00 00 00 00 00 ▯.......................
26CA0917B28 00 00 00 00 00 00 00 00 00 00 00 00 00 00 00 00 00 00 00 00 00 00 00 00 ........................
26CA0917B40 00 00 00 00 00 00 00 00 00 00 00 00 00 00 00 00 00 00 00 00 00 00 00 00 ........................
26CA0917B58 00 00 00 00 00 00 00 00 00 00 00 00 00 00 00 00 00 00 00 00 00 00 00 00 ........................
26CA0917B70 00 00 00 00 00 00 00 00 00 00 00 00 00 00 00 00 00 00 00 00 00 00 00 00 ........................
26CA0917B88 00 00 00 00 00 00 00 00 00 00 00 00 00 00 00 00 00 00 00 00 00 00 00 00 ........................
26CA0917BA0 00 00 00 00 1C 08 00 00 20 07 00 00 80 00 2D 4B 8D C8 FF FF E0 3C 85 CE ........ ... .-K      <
26CA0917BB8 FC 7F 00 00 00 00 00 00 00 00 00 00 00 00 00 00 00 00 00 00 00 00 00 00 ▯.......................
26CA0917BD0 00 00 00 00 00 00 00 00 00 00 00 00 00 00 00 00 00 00 00 00 00 00 00 00 ........................
26CA0917BE8 00 00 00 00 00 00 00 00 00 00 00 00 00 00 00 00 00 00 00 00 00 00 00 00 ........................
26CA0917C00 00 00 00 00 00 00 00 00 00 00 00 00 00 00 00 00 00 00 00 00 00 00 00 00 ........................
26CA0917C18 00 00 00 00 00 00 00 00 00 00 00 00 00 00 00 00 00 00 00 00 00 00 00 00 ........................
26CA0917C30 00 00 00 00 00 00 00 00 00 00 00 00 00 00 00 00 00 00 00 00 00 00 00 00 ........................
26CA0917C48 00 00 00 00 1C 08 00 00 30 17 00 00 80 40 18 46 8D C8 FF FF E0 3C 85 CE ........0... @.F      <
26CA0917C60 FC 7F 00 00 00 00 00 00 00 00 00 00 00 00 00 00 00 00 00 00 00 00 00 00 ▯.......................
26CA0917C78 00 00 00 00 00 00 00 00 00 00 00 00 00 00 00 00 00 00 00 00 00 00 00 00 ........................
26CA0917C90 00 00 00 00 00 00 00 00 00 00 00 00 00 00 00 00 00 00 00 00 00 00 00 00 ........................
26CA0917AFC - 26CA0917B11 (22 bytes) : byte: 28 word: 2076 integer: 2076 int64: 22179211118620 float:0.00 double: 0.00
```

Getting System Uptime

The driver ioctl implements getting system uptime as follows:

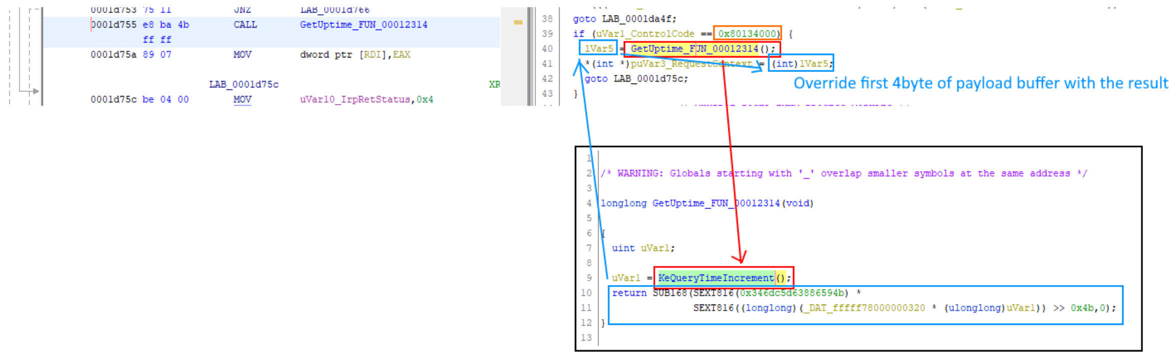
It eventually calls `KeQueryTimeIncrement` which could get system uptime in nanoseconds.

```
PAGE:FFFFF800188CD737 loc_FFFFF800188CD737:                    ; CODE XREF: sub_FFFFF800188CD6E0+38↑j
PAGE:FFFFF800188CD737                 lea     eax, [rcx+7FEEC000h]
PAGE:FFFFF800188CD73D                 mov     edx, 80134000h
PAGE:FFFFF800188CD742                 test    eax, 0FFFCFFFFh
PAGE:FFFFF800188CD747                 jnz     short loc_FFFFF800188CD751
PAGE:FFFFF800188CD749                 cmp     ecx, edx
PAGE:FFFFF800188CD74B                 jnz     loc_FFFFF800188CDA4F
PAGE:FFFFF800188CD751
PAGE:FFFFF800188CD751 loc_FFFFF800188CD751:                    ; CODE XREF: sub_FFFFF800188CD6E0+67↑j
PAGE:FFFFF800188CD751                 cmp     ecx, edx // if (ioctl_code == 0x80134000)
PAGE:FFFFF800188CD753                 jnz     short loc_FFFFF800188CD766
PAGE:FFFFF800188CD755                 call    sub_FFFFF800188C2314 // <-
PAGE:FFFFF800188CD75A                 mov     [rdi], eax // *(unsigned int*)req_ctx = (unsigned int)result
```

and the `sub_FFFFF800188C2314` is:

```
.text:FFFFF800188C2314 sub_FFFFF800188C2314 proc near          ; CODE XREF: sub_FFFFF800188C141C+C↑p
.text:FFFFF800188C2314                                         ; sub_FFFFF800188C5C0C+38↓p ...
.text:FFFFF800188C2314                 sub     rsp, 28h
.text:FFFFF800188C2318                 call    cs:KeQueryTimeIncrement // <-
.text:FFFFF800188C231E                 mov     eax, eax
.text:FFFFF800188C2320                 mov     rcx, 0FFFFF78000000320h
.text:FFFFF800188C232A                 mov     rcx, [rcx]
.text:FFFFF800188C232D                 imul    rcx, rax
.text:FFFFF800188C2331                 mov     rax, 346DC5D63886594Bh
.text:FFFFF800188C233B                 imul    rcx
.text:FFFFF800188C233E                 sar     rdx, 0Bh
.text:FFFFF800188C2342                 mov     rax, rdx
.text:FFFFF800188C2345                 shr     rax, 3Fh
.text:FFFFF800188C2349                 add     rax, rdx
.text:FFFFF800188C234C                 add     rsp, 28h
.text:FFFFF800188C2350                 retn // (unsigned integer) miliseconds
.text:FFFFF800188C2350 sub_FFFFF800188C2314 endp
```

# Call map



```
0001d753 75 11          JNZ        LAB_0001d766
0001d755 e8 ba 4b       CALL       GetUptime_FUN_00012314
         ff ff
0001d75a 89 07          MOV        dword ptr [RDI],EAX

                     LAB_0001d75c                                    XR
0001d75c be 04 00       MOV        uVar10_IrpRetStatus,0x4
```

```
goto LAB_0001da4f;
if (uVar1_ControlCode == 0x80134000) {
  lVar5 = GetUptime_FUN_00012314();
  *(int *)puVar3_RequestContext = (int)lVar5;
  goto LAB_0001d75c;
}
```

Override first 4byte of payload buffer with the result

```

/* WARNING: Globals starting with '_' overlap smaller symbols at the same address */

longlong GetUptime_FUN_00012314(void)

{
  uint uVar1;

  uVar1 = KeQueryTimeIncrement();
  return SUB168(SEXT816(0x346dc5d63886594b) *
                SEXT816((longlong)(_DAT_fffff78000000320 * (ulonglong)uVar1)) >> 0x4b,0);
}

```

Terminate Process

The driver has a vulnerable ioctl code for terminating process, with a specific process id.
It eventually calls `ZwTerminateProcess` in the vulnerable driver context (ring-0).

The ioctl code is `0x81034000` as you can see:

```
PAGE:FFFFF800188CD0F9                 cmp     ebx, 81034000h
PAGE:FFFFF800188CD0FF                 jz      short loc_FFFFF800188CD16C


PAGE:FFFFF800188CD16C loc_FFFFF800188CD16C:                   ; CODE XREF: sub_FFFFF800188CD000+FF↑j
PAGE:FFFFF800188CD16C                 mov     rax, [rsp+30h]
PAGE:FFFFF800188CD171                 mov     ecx, [rax]
PAGE:FFFFF800188CD173                 call    sub_FFFFF800188C36A8
PAGE:FFFFF800188CD178                 and     dword ptr [rbp+1D0h+arg_20], 0
```

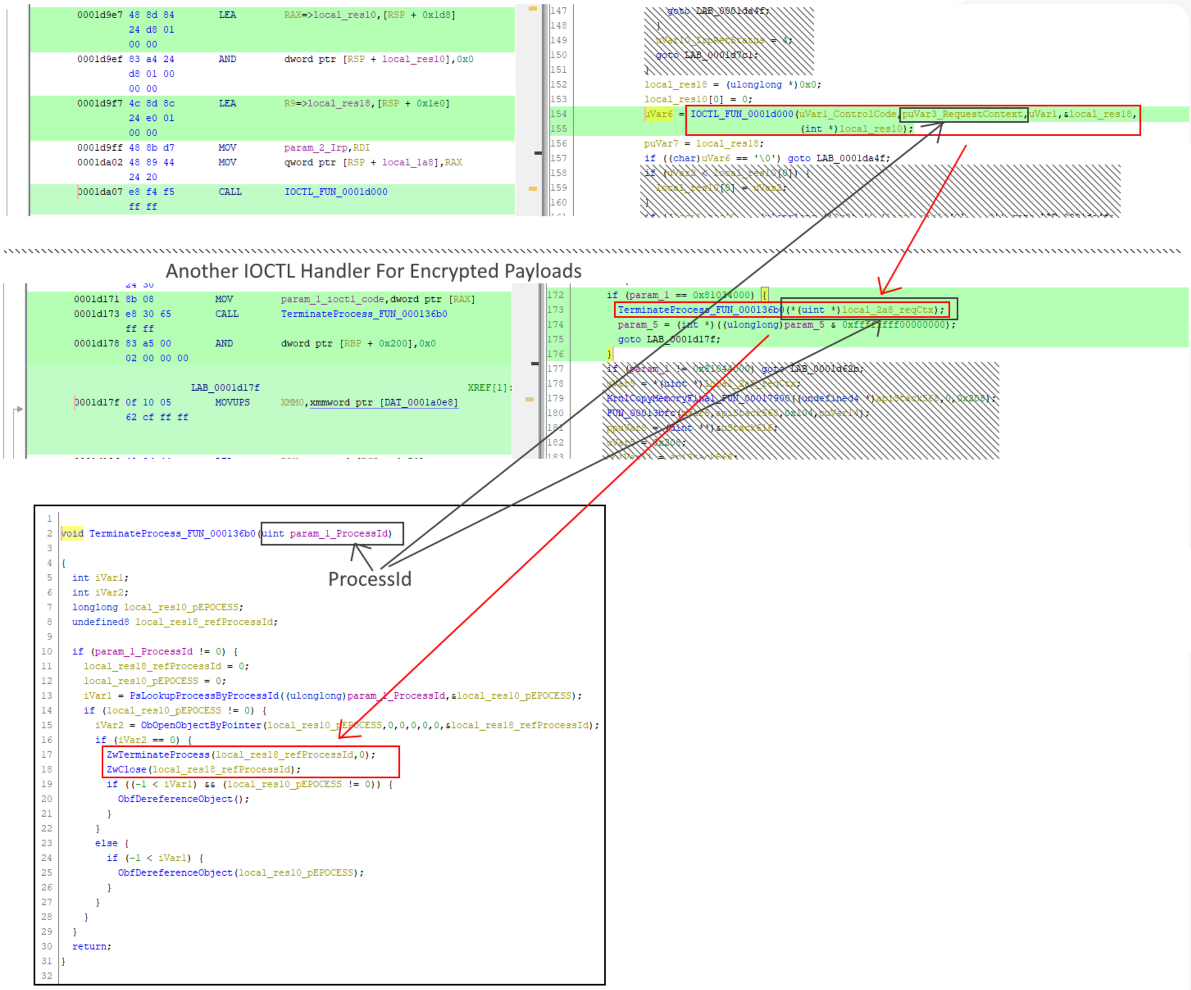
and the `sub_FFFFF800188C36A8` is in `.text` segment:

```
.text:FFFFF800188C36B0 sub_FFFFF800188C36B0 proc near          ; CODE XREF: sub_FFFFF800188C36A8↑j
.text:FFFFF800188C36B0                                         ; sub_FFFFF800188C4600+27↓p
.text:FFFFF800188C36B0                                         ; DATA XREF: ...
.text:FFFFF800188C36B0
.text:FFFFF800188C36B0 var_38          = qword ptr -38h
.text:FFFFF800188C36B0 var_30          = byte ptr -30h
.text:FFFFF800188C36B0 var_28          = qword ptr -28h
.text:FFFFF800188C36B0 var_18          = byte ptr -18h
.text:FFFFF800188C36B0 arg_0           = qword ptr  8
.text:FFFFF800188C36B0 Object          = qword ptr  10h
.text:FFFFF800188C36B0 Handle          = qword ptr  18h
.text:FFFFF800188C36B0 arg_18          = qword ptr  20h
.text:FFFFF800188C36B0
.text:FFFFF800188C36B0 ; __unwind { // __C_specific_handler
.text:FFFFF800188C36B0                 test    ecx, ecx // if (param1_processid != NULL)
.text:FFFFF800188C36B2                 jz      locret_FFFFF800188C3779
.text:FFFFF800188C36B8                 mov     rax, rsp
.text:FFFFF800188C36BB                 mov     [rax+8], rbx
.text:FFFFF800188C36BF                 mov     [rax+20h], rsi
.text:FFFFF800188C36C3                 push    rdi
.text:FFFFF800188C36C4                 sub     rsp, 50h
.text:FFFFF800188C36C8                 xor     ebx, ebx
.text:FFFFF800188C36CA                 mov     dil, 1
.text:FFFFF800188C36CD                 mov     [rax-18h], dil
.text:FFFFF800188C36D1                 mov     [rax+18h], rbx
.text:FFFFF800188C36D5                 mov     [rax+10h], rbx
.text:FFFFF800188C36D9                 mov     ecx, ecx
.text:FFFFF800188C36DB                 lea     rdx, [rax+10h]
.text:FFFFF800188C36DF                 call    PsLookupProcessByProcessId // <- Lookup _EPROCESS
.text:FFFFF800188C36E4                 movzx   esi, dil
.text:FFFFF800188C36E8                 test    eax, eax // if (_EPROCESS != NULL)
.text:FFFFF800188C36EA                 cmovs   esi, ebx
.text:FFFFF800188C36ED                 mov     [rsp+58h+var_18], sil
.text:FFFFF800188C36F2                 mov     rcx, [rsp+58h+Object]
.text:FFFFF800188C36F7                 test    rcx, rcx
.text:FFFFF800188C36FA                 jz      short loc_FFFFF800188C376A
.text:FFFFF800188C36FC                 lea     rax, [rsp+58h+Handle]
.text:FFFFF800188C3701                 mov     [rsp+58h+var_28], rax
.text:FFFFF800188C3706                 mov     [rsp+58h+var_30], bl
.text:FFFFF800188C370A                 mov     [rsp+58h+var_38], rbx
.text:FFFFF800188C370F                 xor     r9d, r9d
.text:FFFFF800188C3712                 xor     r8d, r8d
.text:FFFFF800188C3715                 xor     edx, edx
.text:FFFFF800188C3717                 call    cs:ObOpenObjectByPointer
.text:FFFFF800188C371D                 test    eax, eax
.text:FFFFF800188C371F                 jz      short loc_FFFFF800188C3733
.text:FFFFF800188C3721                 cmp     sil, dil
.text:FFFFF800188C3724                 jnz     short loc_FFFFF800188C3731
.text:FFFFF800188C3726                 mov     rcx, [rsp+58h+Object] ; Object
.text:FFFFF800188C372B                 call    cs:ObfDereferenceObject
.text:FFFFF800188C3731
.text:FFFFF800188C3731 loc_FFFFF800188C3731:                   ; CODE XREF: sub_FFFFF800188C36B0+74↑j
.text:FFFFF800188C3731                 jmp     short loc_FFFFF800188C376A
.text:FFFFF800188C3733 ; ---------------------------------------------------------------------------
.text:FFFFF800188C3733
.text:FFFFF800188C3733 loc_FFFFF800188C3733:                   ; CODE XREF: sub_FFFFF800188C36B0+6F↑j
.text:FFFFF800188C3733                 xor     edx, edx
.text:FFFFF800188C3735                 mov     rcx, [rsp+58h+Handle]
.text:FFFFF800188C373A                 call    cs:ZwTerminateProcess // <- terminate the process
.text:FFFFF800188C3740                 mov     rcx, [rsp+58h+Handle] ; Handle
.text:FFFFF800188C3745                 call    cs:ZwClose // <- close the handle
.text:FFFFF800188C374B                 jmp     short loc_FFFFF800188C3755
.text:FFFFF800188C374D ; ---------------------------------------------------------------------------
.text:FFFFF800188C374D                 mov     dil, 1
.text:FFFFF800188C3750                 mov     sil, [rsp+58h+var_18]
.text:FFFFF800188C3755
.text:FFFFF800188C3755 loc_FFFFF800188C3755:                   ; CODE XREF: sub_FFFFF800188C36B0+9B↑j
.text:FFFFF800188C3755                 cmp     sil, dil
.text:FFFFF800188C3758                 jnz     short loc_FFFFF800188C376A
.text:FFFFF800188C375A                 mov     rcx, [rsp+58h+Object] ; Object
.text:FFFFF800188C375F                 test    rcx, rcx
.text:FFFFF800188C3762                 jz      short loc_FFFFF800188C376A
.text:FFFFF800188C3764                 call    cs:ObfDereferenceObject
.text:FFFFF800188C376A
.text:FFFFF800188C376A loc_FFFFF800188C376A:                   ; CODE XREF: sub_FFFFF800188C36B0+4A↑j
.text:FFFFF800188C376A                                         ; sub_FFFFF800188C36B0:loc_FFFFF800188C3731↑j ...
.text:FFFFF800188C376A                 mov     rbx, [rsp+58h+arg_0]
.text:FFFFF800188C376F                 mov     rsi, [rsp+58h+arg_18]
.text:FFFFF800188C3774                 add     rsp, 50h
.text:FFFFF800188C3778                 pop     rdi
.text:FFFFF800188C3779
.text:FFFFF800188C3779 locret_FFFFF800188C3779:                ; CODE XREF: sub_FFFFF800188C36B0+2↑j
.text:FFFFF800188C3779                 retn
.text:FFFFF800188C3779 ; } // starts at FFFFF800188C36B0
.text:FFFFF800188C3779 sub_FFFFF800188C36B0 endp
```

Since this IOCTL handler has payload encryption measure, we have to encrypt the payload.
And the structure for the request will be like:

```
typedef struct _MHYPROT_TERMINATE_PROCESS_REQUEST
{
    uint64_t response;
    uint32_t process_id;
} MHYPROT_TERMINATE_PROCESS_REQUEST, * PMHYPROT_TERMINATE_PROCESS_REQUEST;
```

# Call Map



```
0001d9e7 48 8d 84      LEA      RAX=>local_res10,[RSP + 0x1d8]
         24 d8 01
         00 00
0001d9ef 83 a4 24      AND      dword ptr [RSP + local_res10],0x0
         d8 01 00
         00 00
0001d9f7 4c 8d 8c      LEA      R9=>local_res18,[RSP + 0x1e0]
         24 e0 01
         00 00
0001d9ff 48 8b d7      MOV      param_2_Irp,RDI
0001da02 48 89 44      MOV      qword ptr [RSP + local_1a8],RAX
         24 20
0001da07 e8 f4 f5      CALL     IOCTL_FUN_0001d000
         ff ff
```

```
      goto LAB_0001da4f;
    }
    uVar10_IrpRetStatus = 4;
    goto LAB_0001d7c1;
  }
  local_res18 = (ulonglong *)0x0;
  local_res10[0] = 0;
  uVar6 = IOCTL_FUN_0001d000(uVar1_ControlCode,puVar3_RequestContext,uVar1,&local_res18,
                             (int *)local_res10);
  puVar7 = local_res18;
  if ((char)uVar6 == '\0') goto LAB_0001da4f;
  if (uVar2 < local_res10[0]) {
    local_res10[0] = uVar2;
  }
```

```
0001d171 8b 08         MOV      param_1_ioctl_code,dword ptr [RAX]
0001d173 e8 30 65      CALL     TerminateProcess_FUN_000136b0
         ff ff
0001d178 83 a5 00      AND      dword ptr [RBP + 0x200],0x0
         02 00 00 00

                 LAB_0001d17f                                   XREF[1]:
0001d17f 0f 10 05      MOVUPS   XMM0,xmmword ptr [DAT_0001a0e8]
         62 cf ff ff
```

```
  if (param_1 == 0x81034000) {
    TerminateProcess_FUN_000136b0(*(uint *)local_2a8_reqCtx);
    param_5 = (int *)((ulonglong)param_5 & 0xffffffff00000000);
    goto LAB_0001d17f;
  }
  if (param_1 != 0x81044000) goto LAB_0001d62b;
  uVar5 = *(uint *)local_2a8_reqCtx;
  KrnlCopyMemoryFinal_FUN_00017900((undefined4 *)apiStack568,0,0x208);
  FUN_00013bfc(uVar5,apiStack568,0x104,puVar14);
  ppuVar8 = (uint **)&uStack616;
  uVar5 = 0x208;
```

```

void TerminateProcess_FUN_000136b0(uint param_1_ProcessId)

{
  int iVar1;
  int iVar2;
  longlong local_res10_pEPOCESS;
  undefined8 local_res18_refProcessId;

  if (param_1_ProcessId != 0) {
    local_res18_refProcessId = 0;
    local_res10_pEPOCESS = 0;
    iVar1 = PsLookupProcessByProcessId((ulonglong)param_1_ProcessId,&local_res10_pEPOCESS);
    if (local_res10_pEPOCESS != 0) {
      iVar2 = ObOpenObjectByPointer(local_res10_pEPOCESS,0,0,0,0,0,&local_res18_refProcessId);
      if (iVar2 == 0) {
        ZwTerminateProcess(local_res18_refProcessId,0);
        ZwClose(local_res18_refProcessId);
        if ((-1 < iVar1) && (local_res10_pEPOCESS != 0)) {
          ObfDereferenceObject();
        }
      }
      else {
        if (-1 < iVar1) {
          ObfDereferenceObject(local_res10_pEPOCESS);
        }
      }
    }
  }
  return;
}

```

→ NEXT ANALYSIS OF PSSETCREATEPROCESSNOTIFYROUTINE

← PREV WINDOWS X64 ユーザーモードから任意のカーネルコードを実行する