TLP : GREEN

라자루스 공격 그룹의 BYOVD를 활용한 루트킷 악성코드 분석 보고서

안랩 시큐리티대응센터(ASEC)

2022. 09. 22

AhnLab

문서 등급에 대한 안내

발간물이나 제공되는 컨텐츠는 아래와 같이 문서 등급 별 허가된 범위 내에서만 사용이 가능합니다.

문서 등급	배포 대상	주의 사항
TLP : RED	특정 고객(사)에 한정하여 제공되는 보고서	**보고서 수신자 혹은 수신 부서 만 접근이 허가된 문서** 수신자 외 복제 및 배포 불가
TLP : AMBER	제한된 고객(사)에 한정하여 제공되는 보고서	**보고서 수신 조직(회사) 내부에서는 복제 및 배포 가능** 다만, 조직 외 교육 목적 등을 위해 사용될 경우에는 안랩의 허락 필수
TLP : GREEN	해당 서비스 내 누구나 이용 가능 보고서	**해당 업종 등에서는 자유로운 사용이 가능하며 출처만 밝히면 내부 교육, 동종 업계, 보안 담당자 교육 자료로 활용 가능** 다만, 일반인 대상 발표자료에는 엄격히 제한
TLP : WHITE	자유 이용 가능 보고서	출처표시 상업적, 비상업적 이용 가능 변형 등 2 차적 저작물 작성 가능

일러두기

보고서에 통계와 지표가 포함되어 있는 경우 일부 데이터는 반올림되어
세부 항목의 합과 전체 합계가 일치하지 않을 수도 있습니다.

본 문서의 버전 이력은 다음과 같다.

버전	날짜	내용
1.0	2022-09-22	라자루스 공격 그룹의 BYOVD 를 활용한 루트킷 악성코드 분석 보고서 작성

목차

CAUTION

'본 보고서에는 현재까지 확인한 내용을 기반으로 분석가 의견이 다수 포함되어 있습니다. 분석가들마다 의견이 다를 수 있으며 새로운 근거가 확인되면, 본 보고서 내용도 사전 고지 없이 변경될 수 있습니다.

1. 개요

북한의 해킹 그룹으로 알려진 라자루스 그룹은 지난 2009년부터 국내를 비롯하여 미국, 아시아, 유럽 등의 다양한 국가를 대상으로 공격을 수행하고 있다. 자사 ASD(AhnLab Smart Defense) 인프라에 따르면 2022년 상반기에 라자루스 그룹은 국내의 방산, 금융, 언론, 제약 산업군에 대해 APT(Advanced Persistent Threat) 공격 활동을 전개하였다.

안랩에서는 이러한 APT 공격을 면밀히 추적하였으며 공격 과정에서 보안 제품을 무력화하는 정황을 확인하였다. 공격 과정을 분석한 결과, 라자루스 그룹은 오래된 버전의 이니텍(INITECH) 프로세스를 악용하여 기업에 초기 침투를 수행한 뒤 공격자 서버로부터 루트킷 악성코드를 다운로드 받아 실행하였다.

이번 제품 무력화 공격에 확인된 루트킷 악성코드는 취약한 드라이버 커널 모듈을 악용하여 직접적으로 커널 메모리 영역에 대해 읽기/쓰기 행위를 수행하였으며, 이에 따라 AV(Anti-Virus)를 포함한 시스템 내 모든 모니터링 시스템이 무력화되었다.

이러한 기법은 "BYOVD(Bring Your Own Vulnerable Driver)"라고 불리며 주로 하드웨어 공급 업체의 취약한 드라이버 모듈을 통해 공격이 이루어지는 것으로 알려져 있다. 최신 윈도우 운영체제에서는 더 이상 서명되지 않은 드라이버 로드는 불가하지만, 공격자는 이처럼 합법적으로 서명된 취약한 드라이버를 이용하여 쉽게 커널 영역을 조작할 수 있다.

이번 사례에서 라자루스 그룹이 사용한 취약한 드라이버 모듈은 "ENE Technology"에서 제작한 하드웨어 관련 모듈이다. 이 모듈은 1999년 Yariv Kaplan이 개발한 "WinIO" 라는 오픈소스 라이브러리를 그대로 사용하였다. 해당 모듈의 문제점은 오래된 오픈소스를 사용한 것뿐만 아니라 모듈을 호출하는 프로세스에 대한 검증 조건이 취약하여 간단한 우회 과정을 통해 커널 임의의 메모리 영역에 대해서 읽기/쓰기가 가능하다는 점이다.

따라서 공격자는 해당 모듈을 통해 임의의 커널 메모리 영역에 대한 읽기/쓰기를 수행하였으며 파일, 프로세스, 스레드, 레지스트리, 이벤트 필터 등 커널 관련 전역 데이터 수정을 통해 AV를 포함한 시스템 내 모든 모니터링 프로그램을 무력화하였다.

2. ene.sys 분석

2.1. 물리 메모리 매핑

"ENE Technology"에서 제작한 ene.sys 드라이버 모듈은 오픈소스인 WinIO 라이브러리[1]로 작성되었으며 유저 영역에서 커널의 물리 메모리와 I/O 포트에 직접 접근을 가능하게 해주는 모듈이다. 해당 드라이버가 사용하는 물리 메모리 접근 방식은 [그림 1]과 같이 "ZwMapViewOfSection" API를 통한 공유 메모리 매핑 방식이다.

따라서 ene.sys와 통신하는 유저 프로세스는 IOCTL 통신을 통해 커널 영역의 물리 메모리를 매핑할 수 있게 된다. 이는 유저 영역에서 임의의 커널 물리 메모리 영역을 조작할 수 있다는 의미이기도 하다.

[1] [1] https://swapcontext.blogspot.com/2020/08/ene-technology-inc-vulnerable-drivers.html

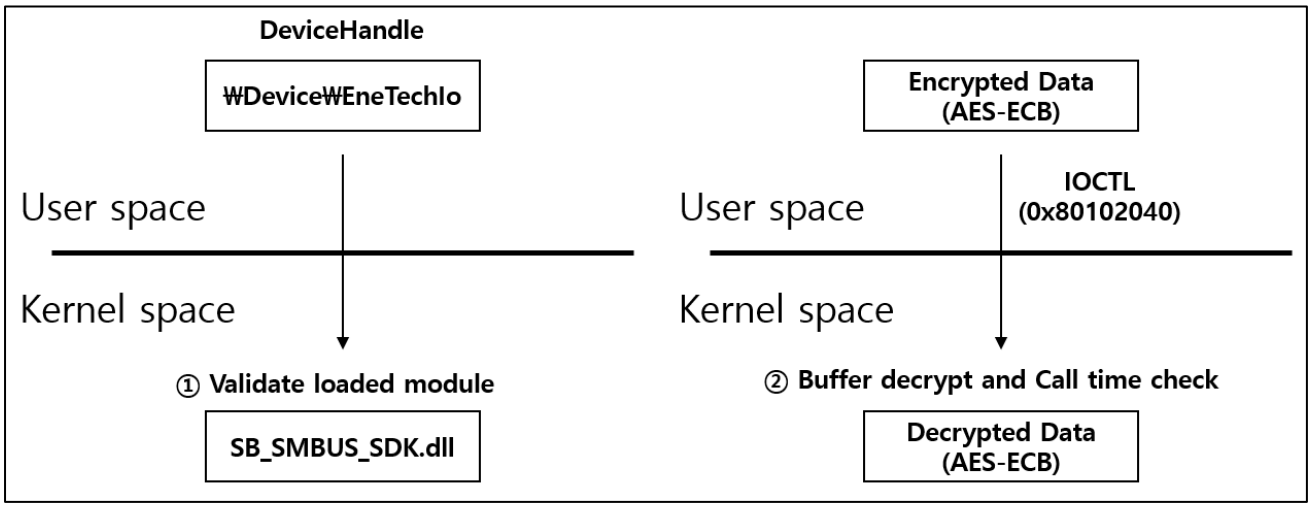
```
RtlInitUnicodeString(&DestinationString, L"\\Device\\PhysicalMemory");
*a4 = 0i64;
Object = a5;
ObjectAttributes.RootDirectory = 0i64;
ObjectAttributes.Length = 48;
ObjectAttributes.Attributes = 576;
*a5 = 0i64;
ObjectAttributes.ObjectName = &DestinationString;
*(_OWORD *)&ObjectAttributes.SecurityDescriptor = 0i64;
v9 = ZwOpenSection(a4, 0xF001Fu, &ObjectAttributes);
if ( v9 < 0 )
  goto LABEL_9;
v9 = ObReferenceObjectByHandle(*a4, 0xF001Fu, 0i64, 0, Object, 0i64);
if ( v9 < 0 )
  goto LABEL_9;
AddressSpace = 0;
TranslatedAddress = BusAddress;
BusAddressa.QuadPart = BusAddress.QuadPart + CommitSize;
v10 = HalTranslateBusAddress(Isa, 0, BusAddress, &AddressSpace, &TranslatedAddress);
AddressSpace = 0;
v11 = v10;
v12 = HalTranslateBusAddress(Isa, 0, BusAddressa, &AddressSpace, &BusAddressa);
if ( v11 && v12 )
{
  SectionOffset = TranslatedAddress;
  CommitSize = BusAddressa.QuadPart - TranslatedAddress.QuadPart;
  v9 = ZwMapViewOfSection(
         *a4,
         (HANDLE)0xFFFFFFFFFFFFFFFFi64,
         &BaseAddress,
         0i64,
         BusAddressa.QuadPart - TranslatedAddress.QuadPart,
         &SectionOffset,
         &CommitSize,
         ViewShare,
         0,
         0x204u);
```

[그림 1] ene.sys의 물리 메모리 매핑 코드 (WinIO 라이브러리)

직접적으로 물리 메모리 영역에 매핑하는 기능은 드라이버의 기능에 따라 필요할 수 있지만, 악용될 경우 큰 위협으로 다가올 수 있으므로 해당 기능을 사용하는 드라이버는 호출자 검증의 작업이 철저하게 이루어져야 한다.

2.2. 호출자 및 유효 데이터 검증

ene.sys의 호출자 검증 절차는 공격자가 쉽게 우회할 수 있도록 설계되어 있다. 드라이버에서 호출자 및 유효한 데이터인지 검증하는 절차는 다음 [그림 2]와 같다.



[그림 2] ene.sys의 호출자 및 유효 데이터 검증 절차

2.2.1. SB_SMBUS_SDK.dll 모듈 로드 검증

ene.sys는 드라이버 실행 시점에 "PsSetLoadImageNotifyRoutine" API를 호출하여 모듈 처리와 관련된 콜백 루틴을 커널에 등록한다. 모듈 콜백이 등록되면 커널은 프로세스에서 모듈 로드 시에 콜백 루틴을 수행할 수 있도록 기능을 제공한다.

ene.sys가 등록한 콜백 루틴은 프로세스에서 로드하는 모듈이 SB_SMBUS_SDK.dll 인지 확인하고, SB_SMBUS_SDK.dll 일 경우 해당 프로세스를 신뢰할 수 있는 프로세스로 인지하여 PID 정보를 ene.sys 드라이버의 전역 변수에 저장한다.

결과적으로 SB_SMBUS_SDK.dll 을 로드한 프로세스는 ene.sys 드라이버와 IOCTL 통신을 할 수 있다.

2.2.2. AES 암호화 IOCTL 통신 및 호출 시간 검증

유저 영역 프로세스가 ene.sys에 물리 메모리 매핑을 요청하기 위해서는 특정 IOCTL 값 (0x80102040)을 전달해야 한다. IOCTL과 함께 드라이버에 전달되는 버퍼는 다음 [표 1] 구조체

정보와 같다.

WinIO 드라이버 메모리 매핑 관련 구조체
typedef struct _WINIO_PHYSICAL_MEMORY_INFO_EX { ULONG_PTR CommitSize; ULONG_PTR BusAddress; HANDLE SectionHandle; PVOID BaseAddress; PVOID ReferencedObject; UCHAR EncryptedKey[16]; } WINIO_PHYSICAL_MEMORY_INFO_EX, * PWINIO_PHYSICAL_MEMORY_INFO_EX;

[표 1] WinIO 드라이버 메모리 매핑 관련 구조체

구조체 멤버 중 BusAddress 변수에는 메모리 매핑을 원하는 물리 메모리 주소가 저장되고, EncryptedKey 변수에는 현재 시간 값에 대한 AES-ECB로 암호화한 값이 저장된다.

ene.sys는 유저 영역에서 요청한 IOCTL에 대한 유효 값을 검증하기 위해 IOCTL 호출 시점의 시각과 해당 IOCTL을 드라이버에서 전달받아 처리한 시각의 차이를 계산한다. 이때 시각의 차이가 2ms 미만일 경우 유효한 것으로 인지하고 요청한 IOCTL을 처리한다.

```
v1 = 0;
if ( !qword_140004010 )
  return 3221225473i64;
sub_140001000(a1);
if ( abs64(sub_140001C54() - *a1) >= 2 )      // Call time check
  return 0xC0000022;                          // STATUS_ACCESS_DENIED
return v1;
```

[그림 3] 호출 시간 검증 루틴

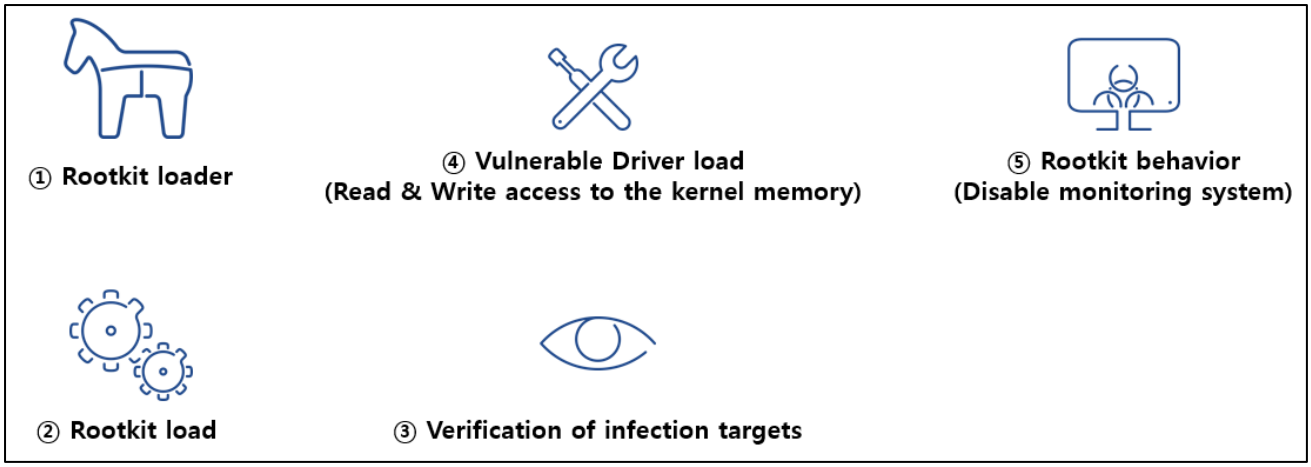
2.3. ene.sys 드라이버(WinIO 라이브러리) 취약성

언급한 내용들을 종합해보면 ene.sys 드라이버는 유저 영역에서 물리 메모리 영역 매핑이 가능한 드라이버이며 호출자 및 데이터에 대한 검증이 미흡한 취약한 드라이버라 할 수 있다.

자사 ASD 인프라를 통해 드라이버의 유포 경로를 분석한 결과 주로 MSI (노트북 제조사)의 RGB 램 모듈 제어 모듈로써 배포되고 있음을 확인하였다. 만일, 사용자 PC 환경에 ene.sys가 설치되어 있을 경우 공격자에 의해 악용될 여지가 있으므로 사용성에 영향이 없다면 제거가 필요하다.

3. 루트킷 악성코드 분석

라자루스 그룹이 보안 제품 무력화를 위해 사용한 루트킷 악성코드의 동작 순서(①~⑤)는 다음 [그림 4]와 같다.



[그림 4] 루트킷 동작 과정 (①~⑤)

루트킷은 루트킷 로더 프로세스 메모리에 DLL 형태로 동작하며 실행 시점에 취약한 드라이버 모듈(ene.sys)을 시스템 드라이브 경로에 생성한다. 그리고 해당 드라이버를 로드하여 커널 메모리 영역의 특정 주소 값을 수정한다.

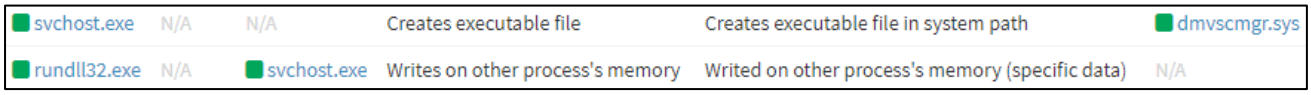
취약한 커널 드라이버에 의해 변조되는 주소 영역은 DLL로 동작 중인 루트킷 스레드 객체의 PreviousMode 주소이며 해당 값을 0으로 변조한다. 유저 스레드 객체의 PreviousMode 값이 0으로 변경될 경우 유저 영역에서 "NtWriteVirtualMemory" API를 통해 커널 영역까지 접근할 수 있게 된다.

이후 공격자는 유저 영역에서 커널 메모리를 조작하여 시스템 내 보안 시스템을 무력화하였다.

3.1. 루트킷 로더 (~BIT353.tmp)

자사 ASD 인프라에 따르면 라자루스 그룹은 루트킷을 DLL(~BIT353.tmp)과 파일리스 형태로 2가지 방식으로 유포하였다. 본 보고서에서는 [그림 5]의 DLL 루트킷을 대상으로 분석한다.

[그림 5] DLL 유포 방식



[그림 6] 파일리스 유포 방식

~BIT353.tmp는 내부에 루트킷 DLL을 XOR 암호화 상태로 저장하여, 이를 실행 시점에 메모리상에서 복호화한다. 즉, 루트킷 자체는 메모리에서 동작하는 구조이다. 다음 [그림 7]은 루트킷 로더가 새로운 메모리를 할당하고 해당 공간에 루트킷을 XOR 복호화하여 루트킷의 익스포트 함수(Create(), Close())를 실행하는 코드이다.

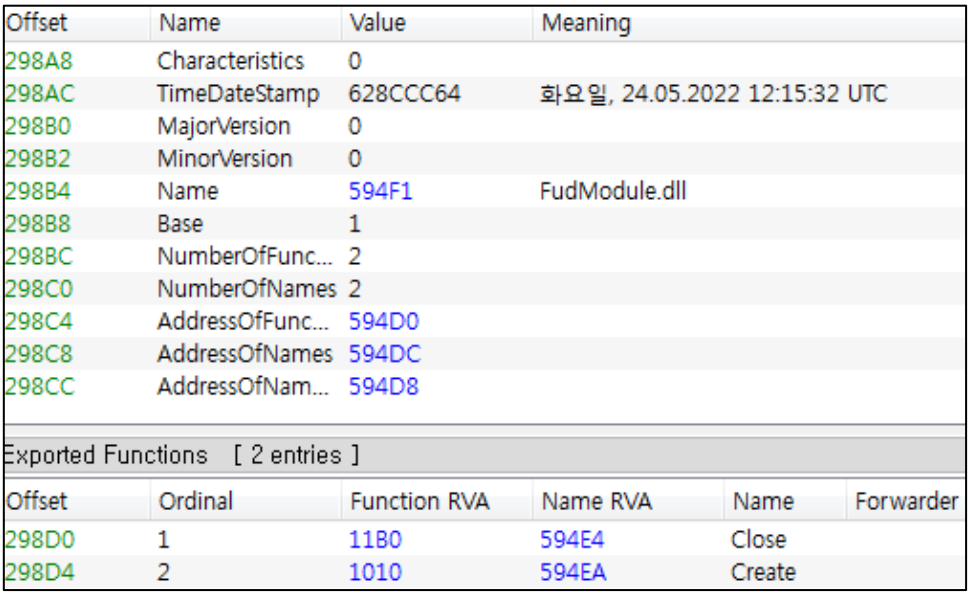
```
v2 = LocalAlloc(0x40u, 0x7FF00ui64);
v3 = v2;
v4 = v2;
v5 = 0x1FFC0i64;
do
{
  v6 = *(_DWORD *)((char *)v4++ + &unk_18000E3C0 - (_UNKNOWN *)v2);
  *(v4 - 1) = v1 ^ v6;
  v1 *= 19;
  --v5;
}                                       // Rootkit Decrypt
while ( v5 );
v7 = (__int64)sub_180001820((char *)v2);
v8 = (void *)v7;
if ( v7 )
{
  v10 = (unsigned int (__fastcall *)(__int64))sub_180001AB0(v7, "Create");
  v11 = (__int64 (*)(void))sub_180001AB0(v8, "Close");
  if ( v10(qword_18008F2F0) )
    v12 = 0xF0000004;
  else
    v12 = v11();                        // Rootkit Load!!
  sub_180001BD0(v8);
```

[그림 7] 루트킷 로더의 루트킷 실행 코드

3.2. 루트킷 (사전 준비 단계)

루트킷은 분석 방해 목적으로 Vmprotect로 실행 압축 되어있으며 다음과 같이 2개의 익스포트 함수가 존재한다.

- 컴파일 시각 : 2022.05.24 12:15:32 (UTC)
- Close() : 루트킷 수행
- Create() : 루트킷 로드 프로세스 메모리 환경 검증
- DLL 이름 : FudModule.dll



Offset	Name	Value	Meaning
298A8	Characteristics	0	
298AC	TimeDateStamp	628CCC64	화요일, 24.05.2022 12:15:32 UTC
298B0	MajorVersion	0	
298B2	MinorVersion	0	
298B4	Name	594F1	FudModule.dll
298B8	Base	1	
298BC	NumberOfFunc...	2	
298C0	NumberOfNames	2	
298C4	AddressOfFunc...	594D0	
298C8	AddressOfNames	594DC	
298CC	AddressOfNam...	594D8	

Exported Functions [2 entries]

Offset	Ordinal	Function RVA	Name RVA	Name	Forwarder
298D0	1	11B0	594E4	Close	
298D4	2	1010	594EA	Create	

[그림 8] 루트킷의 컴파일 시간 및 익스포트 함수 정보

3.2.1. 루트킷 익스포트 함수

a. Close()

Close 함수는 "NtQueryVirtualMemory" API를 통해 루트킷이 로드된 이미지 및 메모리 영역에 대한 검증을 수행한다.

- Type of pages in the region(DLL 베이스 주소) & 0x20000 == 0 (MEM_PRIVATE 검증)
- Type of pages in the region(DLL 베이스 주소) & 0x1000000 == 0 (MEM_IMAGE 검증)
- DLL BASE 주소에 매핑된 맵 파일 이름이 존재하는지 확인
- 동작 중인 OS가 Win 10 RS3 환경보다 최신인지 확인
- ImageSignatureLevel에 다음 값 하나 이상을 포함하는지 확인
  - SE_SIGNING_LEVEL_MICROSOFT
  - SE_SIGNING_LEVEL_WINDOWS
  - SE_SIGNING_LEVEL_WINDOWS_TCB

```
memset(Dst, 0, sizeof(Dst));                    // a1 : DLL Base address
ProcessEnvironmentBlock = NtCurrentTeb()->ProcessEnvironmentBlock;
strcpy(ModuleName, "ntdll");
strcpy(ProcName, "NtQueryVirtualMemory");
if ( !a1 )
  return 0i64;
ModuleHandleA = GetModuleHandleA(ModuleName);
ProcAddress = GetProcAddress(ModuleHandleA, ProcName);
return (ProcAddress)(-1i64, a1, 0i64, v7, 0x30i64, 0i64) < 0// MemoryBasicInformation
    || (v8 & 0x20000) == 0                      // MEM_PRIVATE
    && ((v8 & 0x1000000) == 0                   // MEM_IMAGE
     || ((ProcAddress)(-1i64, a1, 2i64, Dst, 0x7D0i64, 0i64) < 0 || Dst[0])// MemoryMappedFilenameInformation
     && (ProcessEnvironmentBlock->OSBuildNumber < 16299u// Win 10 RS3
      || (ProcAddress)(-1i64, a1, 6i64, v6, 0x18i64, 0i64) < 0// MemoryImageInformation
      || (v6[16] & 0x3Cu) < 4));                // ImageSignatureLevel >= SE_SIGNING_LEVEL_MICROSOFT
                                                // ↘ SE_SIGNING_LEVEL_WINDOWS
                                                // ↘ SE_SIGNING_LEVEL_WINDOWS_TCB
```

[그림 9] Close() 함수

b. Create()

Create의 기능은 ene.sys 드라이버 생성 및 서비스 실행, 보안 제품 무력화 기능을 수행하는 루트킷의 핵심 함수이다.

3.2.2. 감염 대상 검증 루틴

루트킷은 "GetComputerNameW" API 호출의 결과 값을 SHA256으로 계산하여 아래 값과 일치할 경우에만 이후 악성 행위를 수행한다. 이는 감염 대상이 명확하다는 것을 의미하며 APT 공격임을 추정할 수 있다.

- **2022.05.24 컴파일 날짜 파일 :** A1 53 1C 4B FE 51 78 E3 E1 2F 10 35 9D 54 BF 29 42 3C BD 3D 24 F7 71 3D BC 9B D9 0D FA 60 DF C6
- **2022.07.13 컴파일 날짜 파일 :** B4 2D CA BA A0 8D 91 6D F3 B9 66 11 62 24 3F B9 CB 94 DD 08 BD E9 A6 72 30 8D B2 88 AF 73 DA 04

3.2.3. OS 버전 확인

PEB 구조체의 OSBuildNumber 필드를 참고하여 현재 시스템의 OS 정보를 획득한다. 루트킷은 커널 전역 데이터(콜백, 전역 변수) 수정을 통해 보안 제품을 무력화하는 것이 목적이다. 시스템 무력화에 성공하기 위해서는 OS 마다 서로 다른 커널 영역 오프셋의 데이터를 정확하게 수정해야 하므로 획득한 OS 정보를 토대로 커널 전역 데이터 수정을 위한 오프셋 정보를 메모리 공간에 저장한다.

예를 들어 라자루스 그룹의 루트킷이 변조하는 ETHREAD 객체의 PreviousMode 필드의 경우 OS 버전별로 오프셋이 서로 다르다.

- Win7 (7601) PreviousMode 필드 : ETHREAD 구조체의 0x1F6 위치
- Win10 (1809) PreviousMode 필드 : ETHREAD 구조체의 0x232 위치

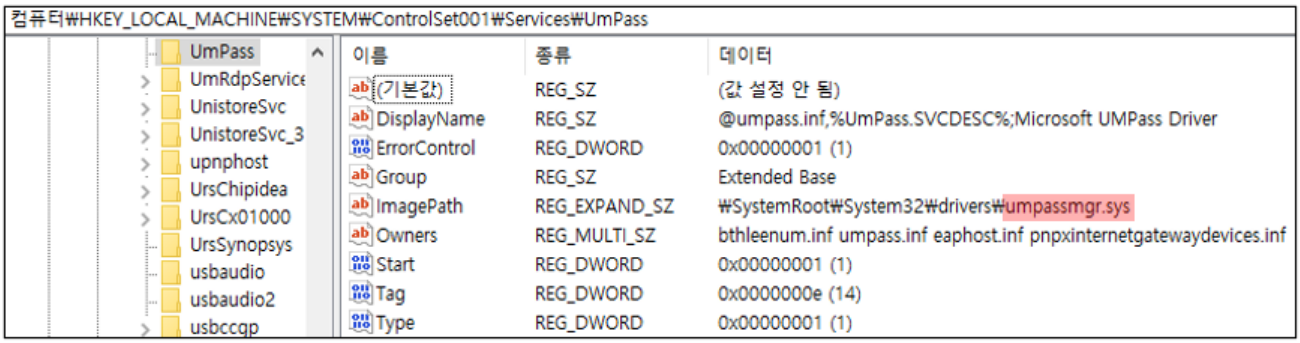
3.2.4. 취약한 드라이버 모듈 로드

커널 메모리 영역에 대한 읽기/쓰기 권한을 획득하기 위해 루트킷은 취약한 커널 드라이버 모듈을 활용한다. 공격에 사용된 커널 드라이버 모듈은 "ENE Technology"에서 제작한 ene.sys라는 모듈을 활용하였다. 앞서 "2. ene.sys 분석" 챕터에서 해당 드라이버에 대한 내용을 자세히 다루었기 때문에 기능 분석은 생략한다.

감염 대상 검증 및 OS 버전 확인 후 루트킷은 ene.sys 드라이버를 시스템 경로에 생성한다. 그리고 해당 드라이버를 실행하기 위해서 기존에 등록된 서비스의 바이너리 경로를 수정했다.

아래 [그림 10]은 루트킷에 의해 기존 윈도우 서비스 레지스트리의 바이너리 경로가 수정된 화면이다.

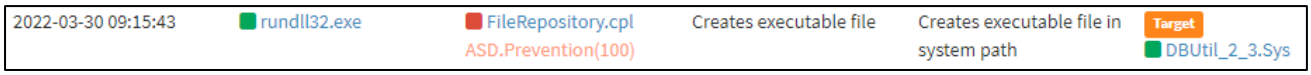
- 변경 전 : ₩SystemRoot₩System32₩drivers₩umpass.sys
- 변경 후 : ₩SystemRoot₩System32₩drivers₩umpassmgr.sys



[그림 10] 기존 윈도우 서비스 레지스트리 키 수정 (umpass.sys -> umpassmgr.sys)

루트킷은 바이너리 경로를 수정한 뒤 "NtLoadDriver" API를 호출하여 해당 서비스를 구동한다.

참고로 라자루스 그룹은 ene.sys 드라이버 이외에도, 루트킷은 변형에 따라 DELL 취약점(CVE-2021-21551)을 악용한 것으로 확인되었다.



[그림 11] DELL 취약점 (CVE-2021-21551) 악용 사례

마찬가지로 CVE-2021-21551 취약점이 악용될 경우 커널 메모리 영역에 대해 읽기/쓰기 권한을 획득할 수 있다. 즉, 현재까지 확인된 드라이버 모듈 악용 사례는 2건이지만 다양한 드라이버가 무력화 공격에 악용될 수 있다.

3.2.5. Kernel DTB (Directory Table Base) 주소 획득

지금까지 과정을 통해 루트킷은 ene.sys 모듈을 이용하여 임의의 커널 메모리 영역에 대한 물리 메모리 매핑으로 읽기/쓰기 권한을 획득하였다. 그러나 물리 메모리 주소에 대해서만 메모리 매핑이 가능하기 때문에 루트킷은 최종적으로 변조하고자 하는 ETHREAD 객체의 PreviousMode 필드의 물리 메모리 주소를 알아야 한다.

```
0: kd> dt _ETHREAD FFFFB88C41945080 +0x232(PreviousMode)
nt!_ETHREAD
   +0x000 Tcb              : _KTHREAD
[+0x232] PreviousMode      : 1 [Type: char]
...
```

[그림 12] ETHREAD 객체의 PreviousMode 필드 주소 정보 (루트킷의 변조 대상 값)

루트킷은 "NtQuerySystemInformation" API에 SystemExtendedHandleInformation을 함수 인자로 전달하여 현재 동작 중인 ETHREAD 객체의 PreviousMode의 가상 주소([그림 12])를 획득하였다. 그리고 해당 주소를 물리 주소로 변환하기 위해서 [그림 13]과 같이 System 프로세스의 DirBase 값을 찾아 직접적으로 Kernel DTB 주소를 획득하는 코드[2]를 구현하였다.

[2] https://public.cnotools.studio/bring-your-own-vulnerable-kernel-driver-byovkd/utilities/loading-device-driver

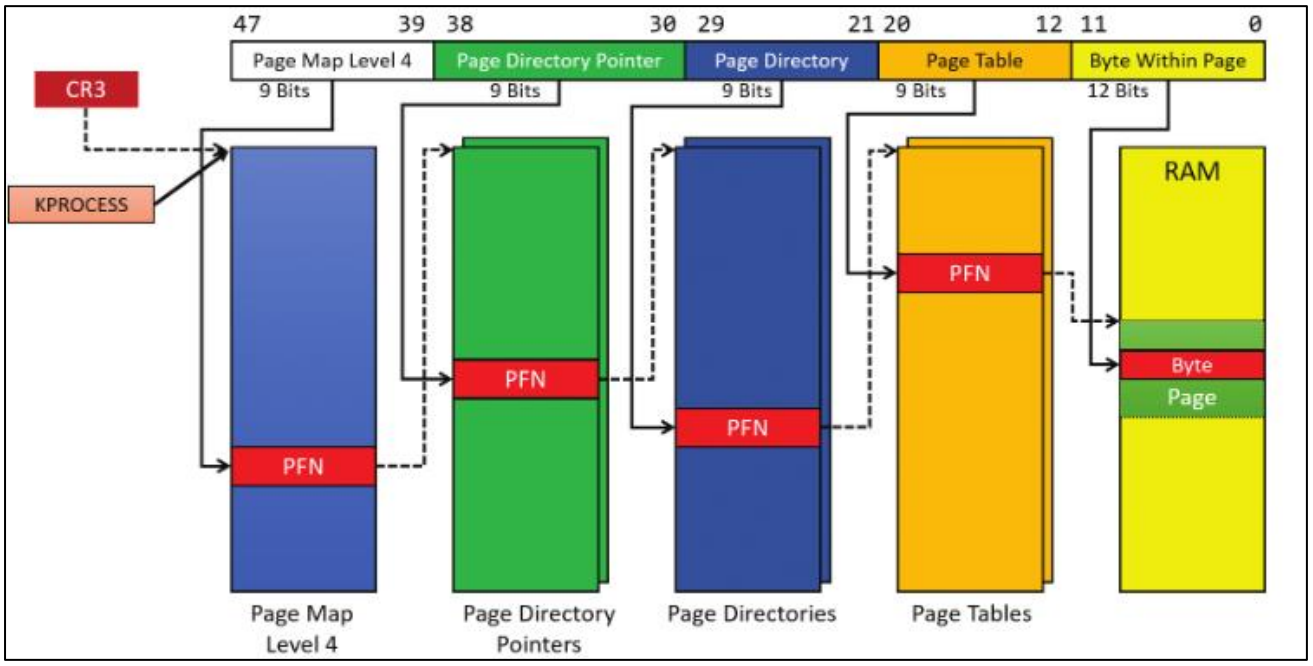
```
v2 = 0;
while ( 2 )
{
  v3 = sub_7FEF3102500(a1, v2, 0x10000u);     // MapPhysicalMemory
  v4 = 0i64;
  v5 = v3;
  v6 = 0;
  do
  {
    if ( (*(v4 + v5) & 0xFFFFFFFFFFFF00FFui64) == 0x1000600E9i64
      && (~*(v4 + v5 + 0x70) & 0xFFFFF80000000000ui64) == 0
      && (*(v4 + v5 + 0xA0) & 0xFFFFFF0000000FFFui64) == 0 )
    {
      v8 = *(v6 + v5 + 0xA0);
      sub_7FEF31025F0(a1);
      return v8;
    }
    v4 += 0x1000i64;
    v6 += 0x1000;
  }
  while ( v4 < 0x10000 );
  *(a1 + 40) = 0i64;
  pcbResult = 0i64;
  *(a1 + 32) = 0i64;
  *(a1 + 72) = time64(0i64);
  BCryptEncrypt(*(a1 + 24), (a1 + 72), 0x10u, 0i64, 0i64, 0, (a1 + 72), 0x10u, &pcbResult, 0);
  DeviceIoControl(                            // UnmapPhysicalMemory
    *(a1 + 16),
    0x80102044,
    (a1 + 32),
    0x38u,
    (a1 + 32),
    0x38u,
    &BytesReturned,
    0i64);
  v2 += 0x10000;
  if ( v2 < 0xA0000 )
    continue;
  break;
}
return 0i64;
```

[그림 13] Kernel DTB 획득 코드

3.2.6. 주소 변환 (가상 주소 -> 물리 주소)

루트킷이 시스템 프로세스의 Kernel DTB (CR3 레지스터) 주소를 획득한 이유는 이를 통해 가상 주소를 물리 주소로 직접 변환할 수 있는 코드를 구현하기 위함이다.

[그림 14]는 Kernel DTB (CR3 레지스터)값을 통해 가상 주소를 물리 주소로 변환하는 과정에 대한 그림이다.



3

[그림 14] 가상 주소 - 물리 주소 변환 과정

루트킷은 [그림 14] 과정을 코드로 구현하였으며, 결과적으로 이 과정을 통해 획득한 PreviousMode 필드 가상 주소의 물리 주소를 계산할 수 있게 되었다.

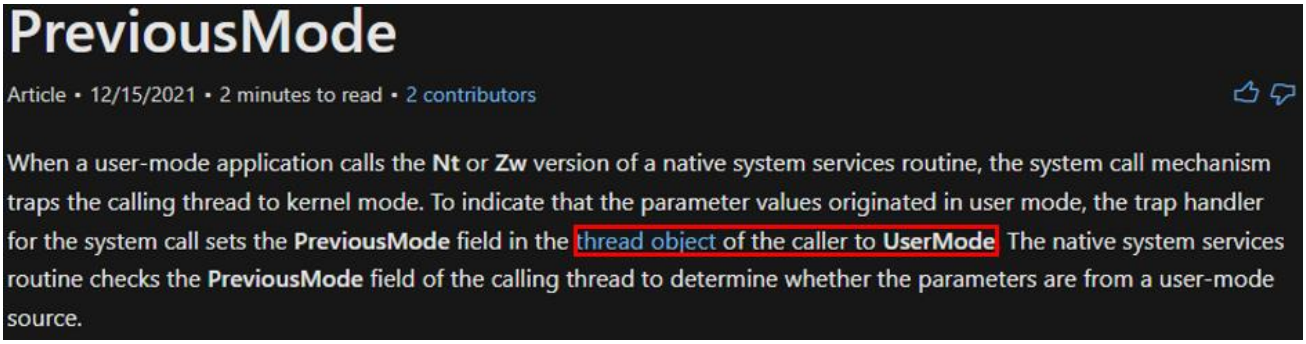
3 Windows Internals: System Architecture, Processes, Threads, Memory Management, and More, Part 1

```
result = sub_7FEF3102500(a1, a2 + 8 * ((a3 >> 39) & 0x1FF), 8u);
if ( result )
{
  v6 = *result;
  sub_7FEF31025F0(a1);
  v7 = sub_7FEF3102500(a1, (v6 & 0xFFFFFFFFFF000i64) + 8 * ((a3 >> 30) & 0x1FF), 8u);
  if ( !v7 )
    return 0i64;
  v8 = *v7;
  sub_7FEF31025F0(a1);
  if ( (v8 & 0x80u) != 0i64 )
  {
    v9 = v8 & 0xFFFFFC0000000i64;
    v10 = a3 & 0x3FFFFFFF;
    return (v10 + v9);
  }
  v11 = sub_7FEF3102500(a1, (v8 & 0xFFFFFFFFFF000i64) + 8 * ((a3 >> 21) & 0x1FF), 8u);
  if ( !v11 )
    return 0i64;
  v12 = *v11;
  sub_7FEF31025F0(a1);
  if ( !v12 )
    return 0i64;
  if ( (v12 & 0x80u) != 0i64 )
  {
    v9 = v12 & 0xFFFFFFFE00000i64;
    v10 = a3 & 0x1FFFFF;
    return (v10 + v9);
  }
  v13 = sub_7FEF3102500(a1, (v12 & 0xFFFFFFFFFF000i64) + 8 * ((a3 >> 12) & 0x1FF), 8u);
  if ( v13 && (v14 = *v13, sub_7FEF31025F0(a1), v14) )
    return ((a3 & 0xFFF) + (v14 & 0xFFFFFFFFFF000i64));
  else
    return 0i64;
}
return result;
```

[그림 15] 가상 주소 – 물리 주소 변환 코드

3.2.7. 스레드 객체의 PreviousMode 필드 변조

PreviousMode 필드는 스레드 객체가 유저 모드에서 호출되었는지 커널에서 검증하는 필드이다. 따라서 해당 멤버 변수가 "0"으로 설정될 경우 유저 영역에서도 커널 영역에 대한 접근이 가능하다.



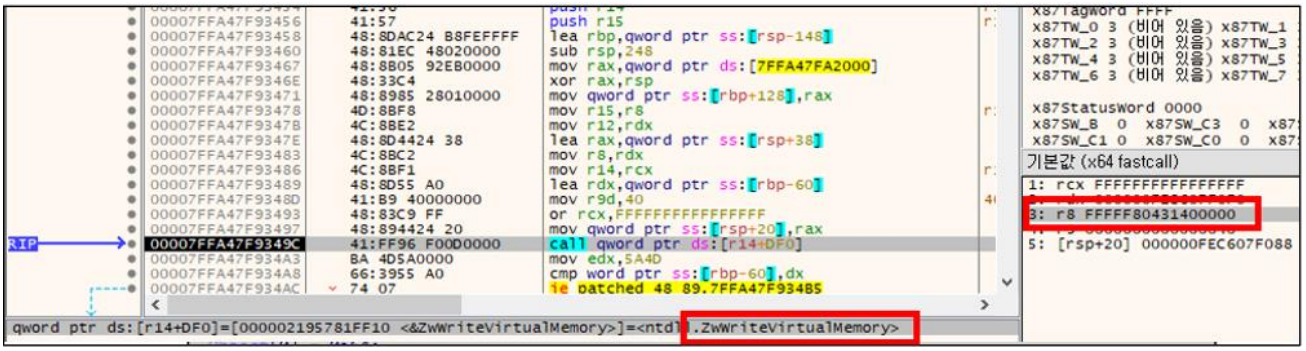
[그림 16] PreviousMode 설명 - msdn

루트킷은 PreviousMode 필드를 수정하기 위해 가상 주소 물리 주소 변환 코드와 커널 물리 메모리 매핑을 활용하여 PreviousMode 필드 값을 기존 "1"에서 "0"으로 변경하였다.

[그림 17] 물리 메모리 매핑을 활용한 PreviousMode 변조

PreviousMode가 "0"으로 설정된 스레드는 "ZwWriteVirtualMemory" API를 통해 유저/커널 영역 모두 접근이 가능한 일종의 "GOD MODE" 상태가 된다.

루트킷은 "ZwWriteVirtualMemory" API를 이용하여 최종 목표인 시스템 무력화를 위해 커널 전역 데이터를 수정한다.



[그림 18] "ZwWriteVirtualMemory" API를 활용하여 유저 영역에서 커널 영역 쓰기 예시 (GOD MODE)

3.3. 루트킷 (보안 제품 무력화 단계)

루트킷의 보안 제품 무력화는 화이트 리스트 기반으로 동작한다. 즉, 시스템 필수 정상 드라이버 파일을 제외한 나머지 모니터링 시스템은 모두 제거한다. 제거 방식은 "ZwWriteVirtualMemory" API를 활용하여 커널 전역 데이터를 조작하여 제거한다.

3.3.1. 미니 파일 필터(fltmgr.sys) 무력화

미니 파일 필터 무력화는 미니 파일 필터(FltMgr)에 등록된 콜백 주소가 아래 표의 화이트 리스트 드라이버에 속하는지 확인한다. 만일, 등록된 드라이버가 해당 드라이버 리스트에 포함되지 않고, 드라이버의 Altitude가 "Anti-virus, Activity-Monitor, Content Screener altitude"에 속할 경우 무력화 대상이 된다.

파일 필터 무력화 검증 정상 드라이버 리스트
scvtrig.sys
fileinfo.sys
afv.sys
cng.sys
filecrypt.sys
storqosflt.sys
bindflt.sys
wcifs.sys

[표 2] 시스템 정상 드라이버 (화이트 리스트)

무력화 방식은 필터 매니저(FltMgr)의 객체 데이터를 조작하여 무력화한다. 이때 조작되는 객체는 "_FLT_FILTER 객체", "_FLT_VOLUME 객체", "_FLT_INSTANCE 객체"가 대상이다.

_FLT_FILTER 객체 무력화
인스턴스리스트(InstanceList) 필드 조작 (무력화 대상 필터의 인스턴스 제거)

[표 3] _FLT_FILTER 객체 무력화 방식

_FLT_VOLUME 객체
인스턴스리스트(InstanceList) 필드 조작 (무력화 대상 필터의 인스턴스 제거)

[표 4] _FLT_VOLUME 객체 무력화 방식

_FLT_INSTANCE 객체
PrimaryLink - 볼륨(_FLT_VOLUME)의 리스트 연결 링크 FilterLink - 볼륨(_FLT_VOLUME)의 리스트 연결 링크
인스턴스 자체는 조작하지 않음

[표 5] _FLT_INSTANCE 객체 무력화 방식

3.3.2. 프로세스/스레드/모듈 탐지 무력화

커널은 프로세스, 스레드, 모듈이 새롭게 생성되거나 로드되는 시점에 모니터링할 수 있는 기능을 제공한다. 일반적으로 보안 제품에서는 커널이 제공하는 이러한 모니터링 기능을 악성코드를 탐지하는데 활용한다.

예를 들어 커널 드라이버는 프로세스 모니터링을 위해 "PsSetCreateProcessNotifyRoutine" 커널 API를 호출하여 "콜백 루틴(callback routine)"을 등록한다. 이후 커널에서는 프로세스 생성 시점에 커널 드라이버가 등록한 콜백 루틴을 수행할 수 있도록 기능을 제공한다.

루트킷은 ntoskrnl.exe 프로세스의 바이너리 패턴 비교를 통해 콜백 루틴이 등록된 아래 전역 변수의 주소 정보를 획득한다. 그리고 해당 주소의 데이터를 조작하여 프로세스/스레드/모듈 콜백을 모두 제거한다. 파일 필터와 마찬가지로 프로세스/스레드/모듈 탐지 무력화는 화이트 리스트 기반으로 동작한다. 루트킷이 검증하는 화이트 리스트는 다음 [표 6]과 같다.

- nt!PspNotifyEnableMask
- nt!PspLoadImageNotifyRoutine (모듈 로드 탐지 무력화)
- nt!PspCreateThreadNotifyRoutine (스레드 생성/종료 탐지 무력화)
- nt!PspCreateProcessNotifyRoutine (프로세스 생성/종료 탐지 무력화)

프로세스 & 이미지 & 스레드 무력화 검증 드라이버 리스트
ntoskrnl.exe
ahcache.sys
mmcss.sys
cng.sys
ksecdd.sys
tcpip.sys
iorate.sys
ci.dll
dxkrnl.sys
peauth.sys

[표 6] 시스템 정상 드라이버 (화이트 리스트)

다음은 스레드 콜백 루틴이 무력화되는 예시이다. nt!PspCreateThreadNotifyRoutine의 전역 데이터가 변경되기 전에는 윈도우 디펜더가 전역 데이터에 저장되어 있었으나 무력화 이후에는 화이트 드라이버(mmcss.sys)를 제외하고 모든 콜백 루틴이 제거되었다.

<스레드 콜백 무력화 전>

```
fffff801`47cec3e0 ffffd60c4804baaf -> WdFilter!MpCreateThreadNotifyRoutine
fffff801`47cec3e8 ffffd60c4804ba7f -> WdFilter!MpCreateThreadNotifyRoutineEx
fffff801`47cec3f0 ffffd60c48de643f -> mmcss!CiThreadNotification
```

<스레드 콜백 무력화 후>

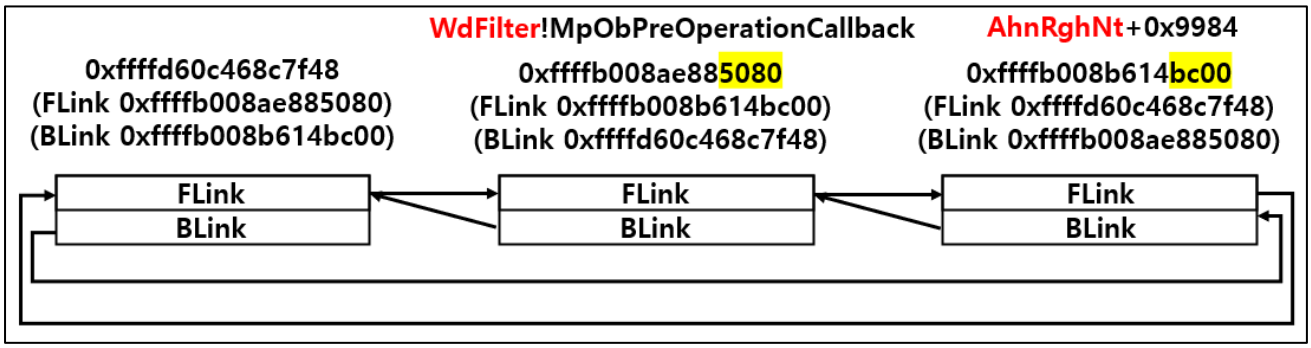
```
fffff801`47cec3e0 ffffd60c48de643f -> mmcss!CiThreadNotification
fffff801`47cec3e8 0000000000000000 -> Remove !!
fffff801`47cec3f0 0000000000000000 -> Remove !!
```

3.3.3. 레지스트리 콜백 무력화

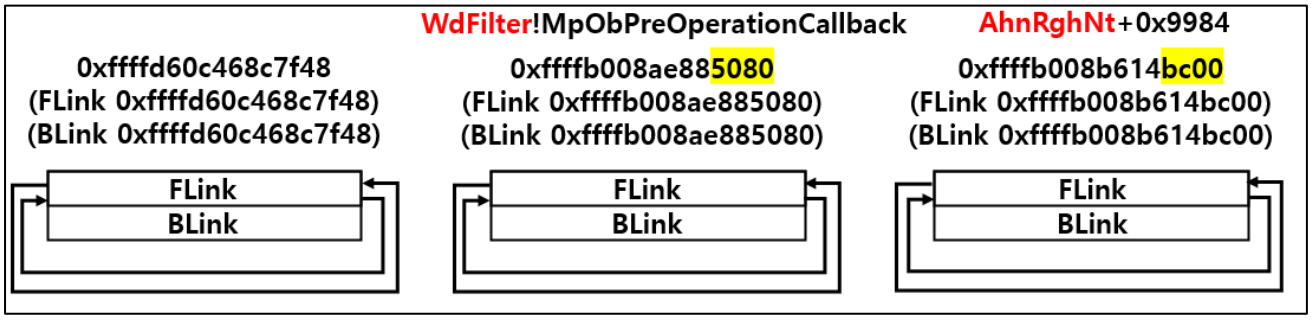
루트킷은 ntoskrnl.exe 프로세스 내 바이너리 패턴 비교를 통해 "CmUnRegisterCallback"의 시작 바이트를 확인한다. 그리고 프로세스/스레드/모듈 무력화와 유사하게 nt!CallbackListHead라는 커널 전역 데이터를 수정하여 시스템에 등록된 레지스트리 콜백을 무력화한다.

레지스트리 콜백이 무력화되면 레지스트리와 관련된 행위 모니터링은 불가능하게 된다.

다음 [그림 19], [그림 20]은 nt!CallbackListHead 데이터가 조작되기 전과 후를 비교한 그림이다.



[그림 19] 레지스트리 콜백 무력화 전



[그림 20] 레지스트리 콜백 무력화 후

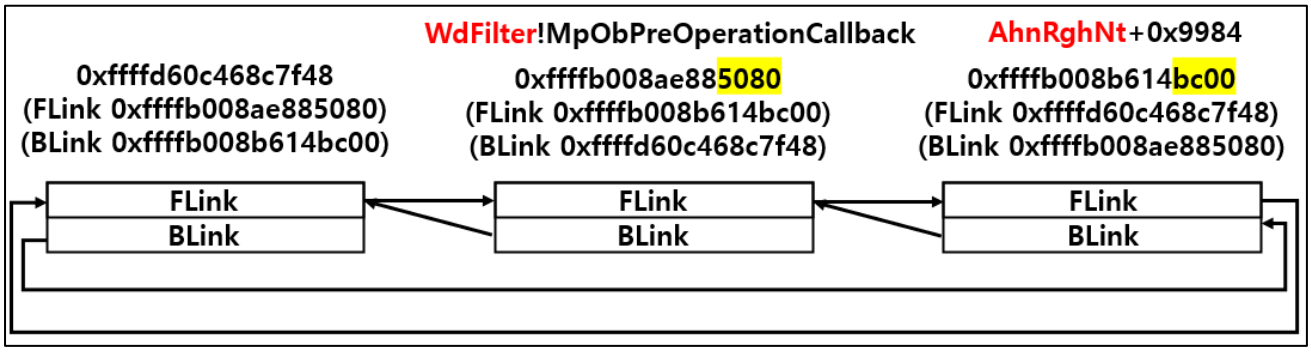
결과적으로 루트킷은 nt!CallbackListHead 변조를 통해 윈도우 디펜더 및 보안 제품의 레지스트리 감시를 무력화 하였다.

3.3.4. 오브젝트 콜백 무력화

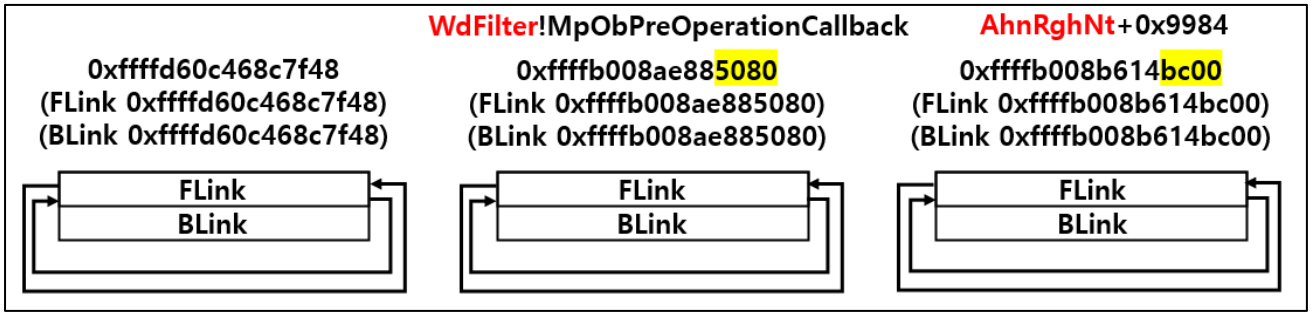
루트킷은 오브젝트 콜백 리스트 조작을 통해 보안 제품의 프로세스 및 스레드 오브젝트에 대한 접근 제어를 무력화하였다.

먼저, ntoskrnl.exe 내부 바이트 패턴 비교를 통해 ObGetObjectType의 시작 바이트 주소를 얻고 "nt!ObTypeIndexTable"라는 전역 데이터 주소 정보를 획득하였다. 그리고 해당 테이블에 저장된 각 요소 중 프로세스 및 스레드 오브젝트에 대한 콜백 리스트를 조작하였다.

다음 [그림 21], [그림 22]는 프로세스 오브젝트의 콜백 리스트가 조작되기 전과 후를 비교한 그림이다.



[그림 21] 오브젝트 콜백 무력화 전



[그림 22] 오브젝트 콜백 무력화 후

콜백 연결 리스트를 제거함으로써 보안 제품의 프로세스 오브젝트에 대한 접근 제어를 무력화하였다.

3.3.5. WFP 네트워크 필터 무력화

Windows Filtering Platform(WFP)은 응용 프로그램에서 네트워크 패킷 제어를 할 수 있도록 커널이 제공하는 플랫폼이다. 따라서 보안 제품은 WFP를 방화벽, 침입 탐지 등의 기능에 활용할 수 있다.

루트킷은 WFP 드라이버의 콜아웃(Callout)을 탐색하여 해당 객체의 플래그 조작을 통해 시스템에 설치된 네트워크 모니터링 시스템을 무력화한다. 미니 파일 필터, 프로세스/이미지/모듈 콜백 조작과 마찬가지로 화이트 리스트 기반으로 윈도우 정상 드라이버 콜아웃을 제외한 나머지 콜아웃 드라이버에 대한 플래그 값만 조작한다.

네트워크 무력화 검증 드라이버 리스트
ndu.exe
tcpip.sys
mpsdrv.sys

[표 7] 시스템 정상 드라이버 (화이트 리스트)

콜아웃 탐색 과정은 이전 방식과 유사하다. 먼저 NETIO.sys 에서 "WfpProcessFlowDelete" 함수의 시작 바이트 주소를 찾는다. 그리고 "NETIO!gwfpGlobal" 주소를 찾아 콜아웃 엔트리를 순회하며 등록된 드라이버 이름을 확인한다.

콜아웃 드라이버가 [표 7]의 화이트 리스트 드라이버가 아닐 경우에는 아래와 같이 특정 바이트에 "1"을 OR 연산하여 무력화를 시도한다.

```
if ( v11[0] && v12 && !sub_7FEF31040D0(a1, v12, 1) )
{
  v4 = v9 + *(a1 + 0xC9C) + i * *(a1 + 0xC98);
  v6 = v11[*(a1 + 0xC9C) >> 3] | 1;
  (*(a1 + 0xDF0))(-1i64, v4, &v6, 8i64, v10);
}
```

[그림 23] 네트워크 필터 무력화 코드

3.3.6. 이벤트 추적 무력화

윈도우는 커널 및 유저 영역에서 발생하는 다양한 이벤트를 기록 및 추적할 수 있도록 기능을 제공한다. 이러한 이벤트는 EDR과 같은 보안 제품에서도 활용하고 있으며 악성코드의 행위를 추적하는데 도움을 준다.

루트킷은 이벤트 추적을 무력화하기 위해 관련 핸들과 변수를 무력화하였다. 무력화 방식은 기존과 유사한 형태이며 ntoskrnl.exe 의 내부 바이너리 패턴 비교를 통해 "EtwRegister" 함수의 시작 주소를 찾아 아래와 같이 등록된 핸들 정보를 획득하였다. 그리고 핸들 값이 저장된 주소들을 모두 "NULL" 로 채워 핸들 정보를 무력화하였다.

<ETW 이벤트 핸들 저장 주소>

```
FFFFF80147C19990 ; nt!EtwpEventTracingProvRegHandle
FFFFF80147C19780 ; nt!EtwKernelProvRegHandle
FFFFF80147C19940 ; nt!EtwpPsProvRegHandle
FFFFF80147C19938 ; nt!EtwpNetProvRegHandle
FFFFF80147C19930 ; nt!EtwpDiskProvRegHandle
FFFFF80147C19928 ; nt!EtwpFileProvRegHandle
...
FFFFF80147C19968 ; nt!EtwSecurityMitigationsRegHandle
```

이외에도, 커널 전역 데이터인 nt!EtwpHostSiloState 의 주소를 찾아 해당 주소에서 0x1080 오프셋 떨어진 4바이트 값을 0x0로 수정하여 이벤트 추적 관련 변수를 무력화하였다.

<nt!EtwpHostSiloState 무력화 전>

```
ffffd60c`469d4080 0001230500000006
ffffd60c`469d4088 0000000000004080
```

<nt!EtwpHostSiloState 무력화 후>

```
ffffd60c`469d4080 0001230500000000 -> Modified !!
ffffd60c`469d4088 0000000000004080
```

"nt!EtwpHostSiloState" 주소에서 루트킷이 변조하는 오프셋은 "3.2.3. OS 버전 확인"에서 언급하였듯이 윈도우 운영체제 별로 정보가 다르다.

- 감염 대상이 Windows Server 2022일 경우 오프셋 : 0x1088
- 감염 대상이 Windows 11일 경우 오프셋 : 0x1098
- 그 외 나머지 OS의 경우 오프셋 : 0x1080

```
v2 = *(v1 + 0x120);                             // OSBuildNumber
if ( v2 < 17763u || v2 > 19044u )               // 17763 : win10 1809, 19044 : win10 21h2
{
  if ( v2 == 20348 )                            // 20348 : windows server 2022
  {
    a1[0x1A2] = 0x1088i64;
  }
  else if ( v2 == 22000 )                       // 22000 : win 11 21h2
  {
    a1[0x1A2] = 0x1098i64;
  }
}
else
{
  a1[0x1A2] = 0x1080i64;
}
```

[그림 24] OS 별 메모리 공간에 저장되는 오프셋 정보

안랩 대응 현황

안랩 제품군의 진단명과 엔진 버전 정보는 다음과 같다.

[파일 진단]
Trojan/Win.Lazardoor.C5157217 (2022.06.04.01)
Rootkit/Win.Agent.C5192169 (2022.07.04.02)
Rootkit/Win.Agent.C5177679 (2022.06.23.00)

[행위 진단]
InitialAccess/MDP.Event.M4419 (2022.09.21.01)
InitialAccess/MDP.Event.M4422 (2022.08.08.02)

이 위협 그룹의 활동이 최근 공개되었다고 해도 일부 악성코드는 안랩 제품에서 진단되고 있었다. ASEC에서 이 그룹의 활동을 추적하며 악성코드를 대응했지만 확인되지 않아 미진단 중인 변형이 존재할 수 있다.

결론

라자루스 그룹은 APT 공격 과정에서 취약한 드라이버를 이용하여 시스템 내부 모니터링 시스템을 모두 무력화하였다. 현재까지 확인된 취약한 드라이버는 2개이지만, 정상적으로 서명된 취약한 드라이버는 이보다 훨씬 많아 악용 사례는 더욱 더 늘어날 것으로 보인다.

루트킷 공격은 윈도우 비스타 이상부터 DSE(Driver Signature Enforcement) 정책이 적용된 이후 감소하는 듯 보였으나, 이처럼 BYOVD(Bring Your Own Vulnerable Driver) 공격 사례는 2014년부터 꾸준히 확인되고 있다. 지금까지의 BYOVD 공격은 주로 권한 상승을 위해 악용한 것으로 알려져 있지만, 이번 공격 사례와 같이 과거 윈도우 7 부터 최신 운영체제인 윈도우 서버 2022까지 모든 시스템을 무력화하도록 정교하게 설계된 루트킷 공격은 라자루스 그룹이 최초인 것으로 보인다.

BYOVD 공격을 대응하기 위해 마이크로 소프트에서는 윈도우 10의 하이퍼바이저 코드 무결성(HVCI) 모드와 S 모드에서 차단 규칙 기반[4]으로 허용되지 않은 드라이버는 로드 되지 않도록 자체 차단하고 있다. 이렇듯 엄격하게 드라이버 로드를 차단하는 것이 현재로서는 이러한 유형의 공격을 예방하기 위한 최선으로 보인다.

따라서 기업 보안 담당자는 일반 사용자 환경에서는 드라이버 로드를 하지 못하도록 통제해야 하고, 보안 S/W 업데이트를 최신으로 유지하여 BYOVD를 활용한 APT 공격을 예방해야 한다.

[4] https://docs.microsoft.com/en-us/windows/security/threat-protection/windows-defender-application-control/microsoft-recommended-driver-block-rules

IoC (Indicators of Compromise)

파일 경로 및 이름

악성코드에서 사용한 파일 경로 및 이름은 다음과 같다. 일부는 정상 파일 이름과 동일할 수 있다.

[루트킷 로더]
- %SystemRoot%₩Comms.bin
- %SystemRoot%₩miblib.bin

[루트킷]
- %SystemRoot%₩miblib.dat
- %SystemRoot%₩temp₩~bit353.tmp
- %SystemRoot%₩bf.dat

[루트킷 관련 모듈] – SB_SMBUS_SDK.dll
- %USERPROFILE%₩appdata₩local₩temp₩1b6955_sb_smbus_sdk.dll
- %SystemRoot%₩temp₩206778_sb_smbus_sdk.dll
- %SystemRoot%₩temp₩4018a066_sb_smbus_sdk.dll
- %SystemRoot%₩temp₩6d0b88f_sb_smbus_sdk.dll
- %SystemRoot%₩temp₩4ab916_sb_smbus_sdk.dll
- %SystemRoot%₩temp₩18532412_sb_smbus_sdk.dll
- %SystemDrive%₩users₩%ASD%₩appdata₩local₩temp₩9e1126_sb_smbus_sdk.dll

파일 Hashes (MD5)

관련 파일의 MD5는 다음과 같다. (단, 민감 샘플이 존재할 경우 제외된다.)

[루트킷 로더]
- 013b4c4e9387d8fe1eab738c42c451da

[루트킷]
- 98e58a39ede26af7980ed4de2873caab
- a6e309f97ffada2d4d0d4aecfb255a91

[루트킷 관련 모듈] – SB_SMBUS_SDK.dll
- c40643751b426dec01bd390e192b4542

참고 문헌

[1] https://swapcontext.blogspot.com/2020/08/ene-technology-inc-vulnerable-drivers.html
[2]https://public.cnotools.studio/bring-your-own-vulnerable-kernel-driver-byovkd/utilities/loading-device-driver
[3] Windows Internals: System Architecture, Processes, Threads, Memory Management, and More, Part 1
[4]https://www.amazon.com/Windows-Internals-Part-architecture-management-ebook-dp-B0711FDMRR/dp/B0711FDMRR/ref=mt_other?_encoding=UTF8&me=&qid=

More security, More freedom

㈜안랩
경기도 성남시 분당구 판교역로 220 (우) 13493
대표전화 : 031-722-8000 | 구매문의 : 1588-3096 | 팩스 : 031-722-8901
www.ahnlab.com

AhnLab