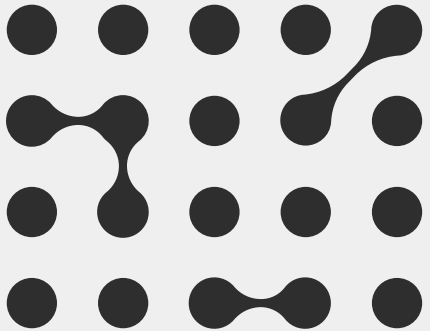
Enhanced Vulnerability Hunting in WDM Drivers Using Symbolic Execution and Taint Analysis



- TeamT5 Engine Team - Zeze

- TeamT5 Security Researcher
- Member of ⚔️TSJ⚔️ and TWN48 (DEFCON CTF 2023 3rd)
- 50+ Windows Kernel CVEs
- HITCON 2023, VXCON 2022, and CYBERSEC 2023 Speaker

Overview

01 **Introduction** to WDM driver and IOCTLance

02 **Design** of IOCTLance

03 **Vulnerability** types and real-world cases

04 **Enhancement** made to get better performance

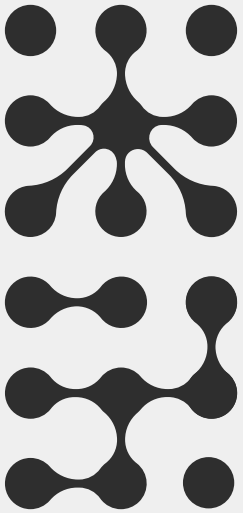
05 **Evaluation** with both known and unknown drivers

06 **Conclusion**

01

Introduction

introduction to WDM driver and IOCTLance



WDM Driver

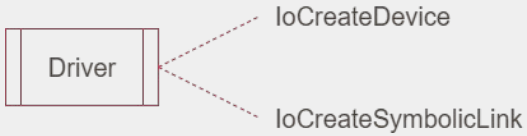
- Since Windows 2000, Microsoft has recommended using **WDM (Windows Driver Model)** drivers to provide support for devices.
- WDM drivers account for **most of the Windows kernel drivers** in the market.
- Many **vulnerabilities** have been discovered in WDM drivers that could be **exploited** by attackers.

WDM Driver

1. Create a device.

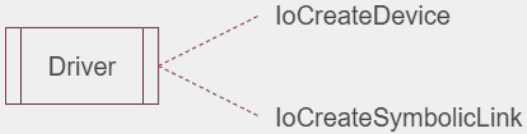
WDM Driver

1. Create a device.
2. Create a symbolic link for the device.



WDM Driver

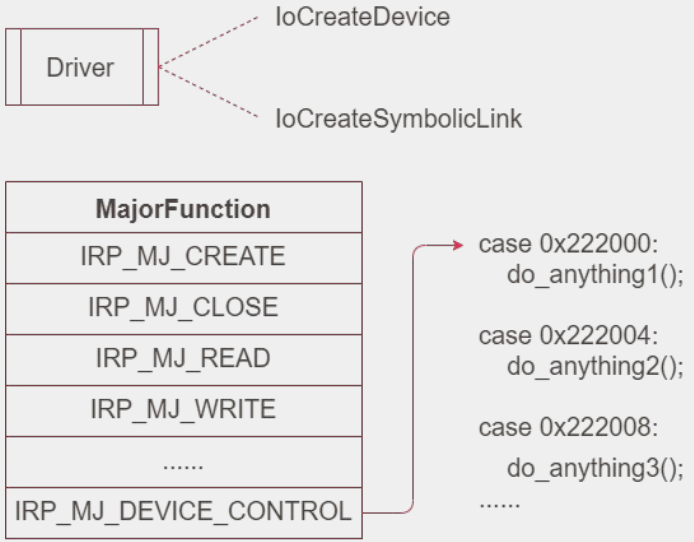
1. Create a device.
2. Create a symbolic link for the device.
3. Define dispatch routines for each IRP.



MajorFunction
IRP_MJ_CREATE
IRP_MJ_CLOSE
IRP_MJ_READ
IRP_MJ_WRITE
.....
IRP_MJ_DEVICE_CONTROL

WDM Driver

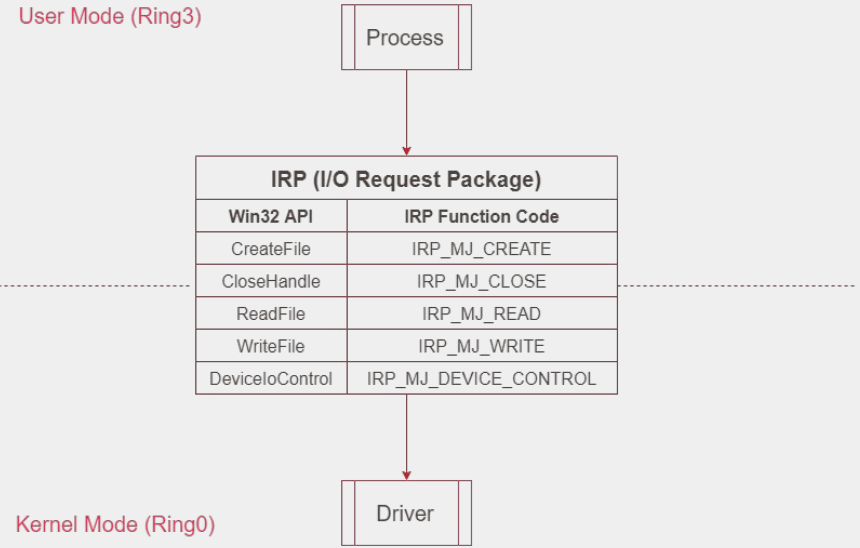
1. Create a device.
2. Create a symbolic link for the device.
3. Define dispatch routines for each IRP.
4. Implement IOCTL handler.



IRP

(I/O Request Packet)

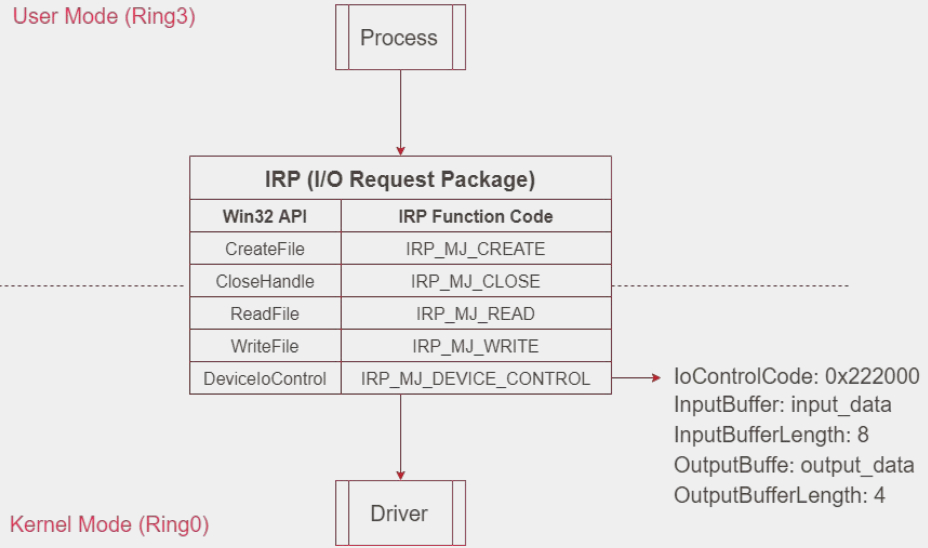
a data structure to communicate between drivers and OS



IOCTL

(Device Input and Output Control)

an interface communicating with a device driver



BYOVD

Bring Your Own Vulnerable Driver

	Abuse Driver	Vulnerability	Reason	Motivation
BlackByte	RTCore.sys	read/write controllable address	bypass antivirus	execute ransomware
Candiru	HW.sys	map physical memory	install rootkit	execute spyware

Symbolic Execution & Taint Analysis

techniques used in software security

Symbolic Execution

- Analyze code without running it on a real system.
- Explore all possible paths through the program.
- Create a mathematical model and assign symbolic variables..

Taint Analysis

- Track potentially unsafe inputs.
- Mark data originated from an untrusted source.
- Identify where the tainted data may be used in an unintended way.

IOCTLance

Enhancement

ability to detect various vulnerability types in WDM drivers

Efficiency

IOCTLance efficiently identify vulnerabilities without the environment.

CVE

Detect 117 vulnerabilities across 26 distinct drivers

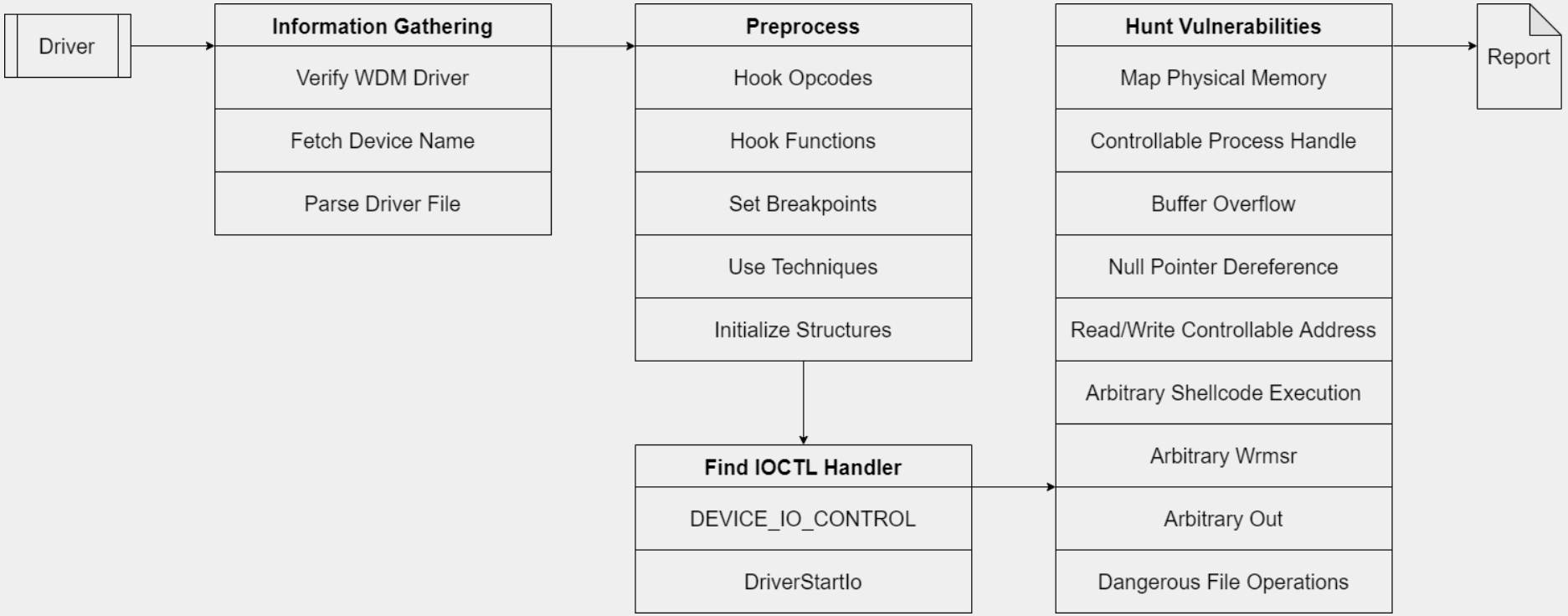
Related Work

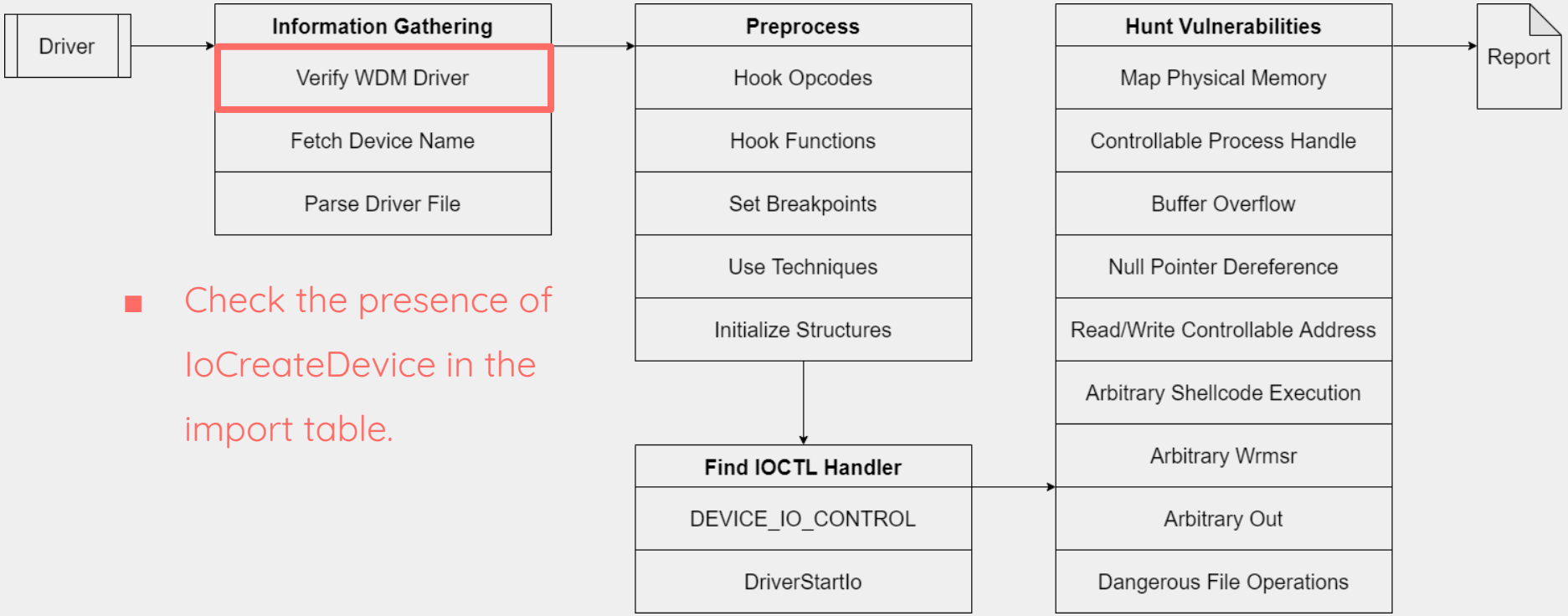
- **Fuzzing**: ioctlfuzzer, ioctlbf, iofuzz, IoAttack, etc.
  - Hard to analyze code in-depth.
- **Fuzzing + Symbolic Execution**: CAB-FUZZ, SmartFuzz, Dowser, DIODE, etc.
  - Require the environment to analyze.
- **Symbolic Execution + Taint Analysis**: Screwed-Drivers, POPKORN, etc.
  - Suffer path explosion and currently few vulnerability types.

02

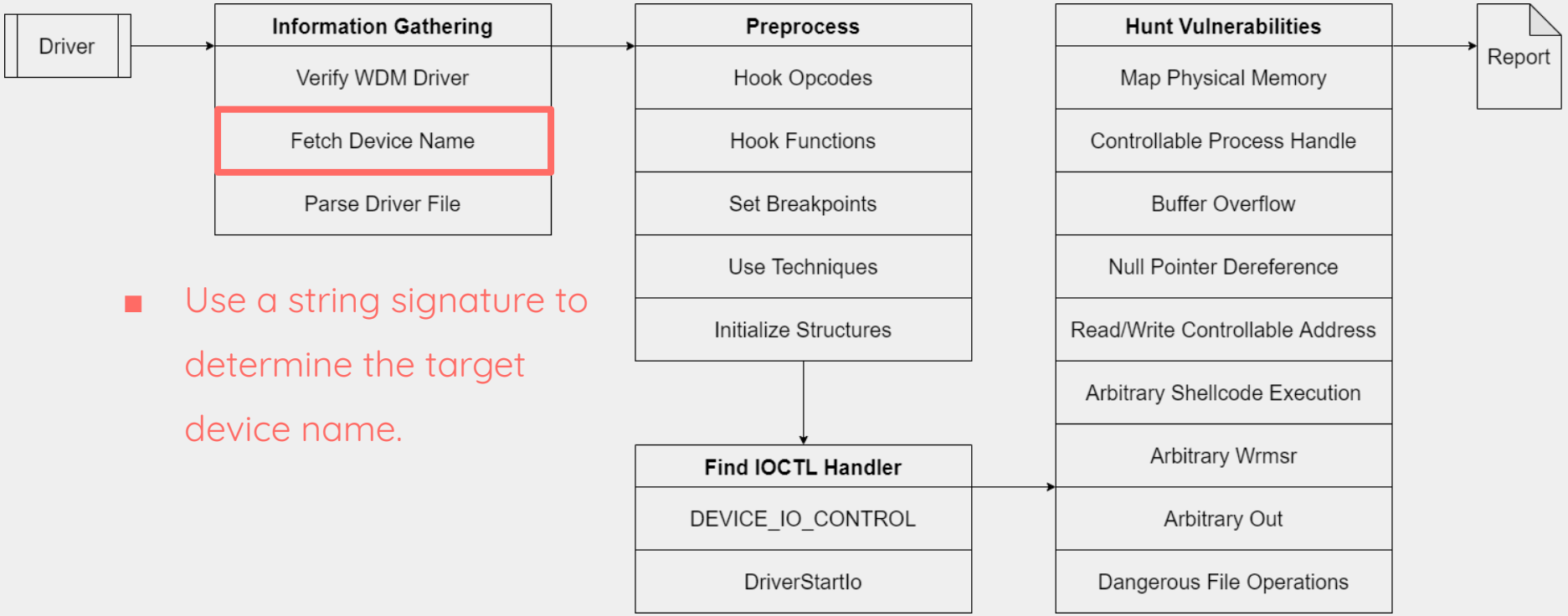
Design

- design of IOCTLance

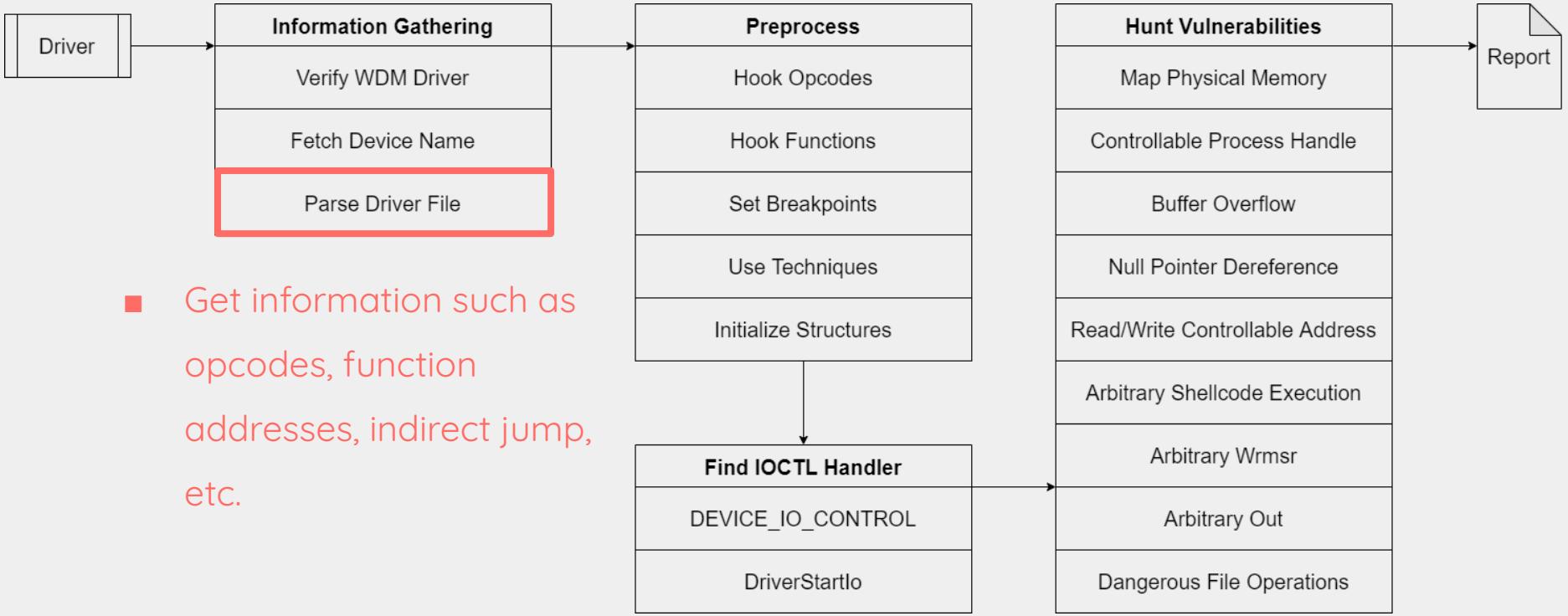

Driver
Information Gathering
Verify WDM Driver
Fetch Device Name
Parse Driver File
Preprocess
Hook Opcodes
Hook Functions
Set Breakpoints
Use Techniques
Initialize Structures
Find IOCTL Handler
DEVICE_IO_CONTROL
DriverStartIo
Hunt Vulnerabilities
Map Physical Memory
Controllable Process Handle
Buffer Overflow
Null Pointer Dereference
Read/Write Controllable Address
Arbitrary Shellcode Execution
Arbitrary Wrmsr
Arbitrary Out
Dangerous File Operations
Report



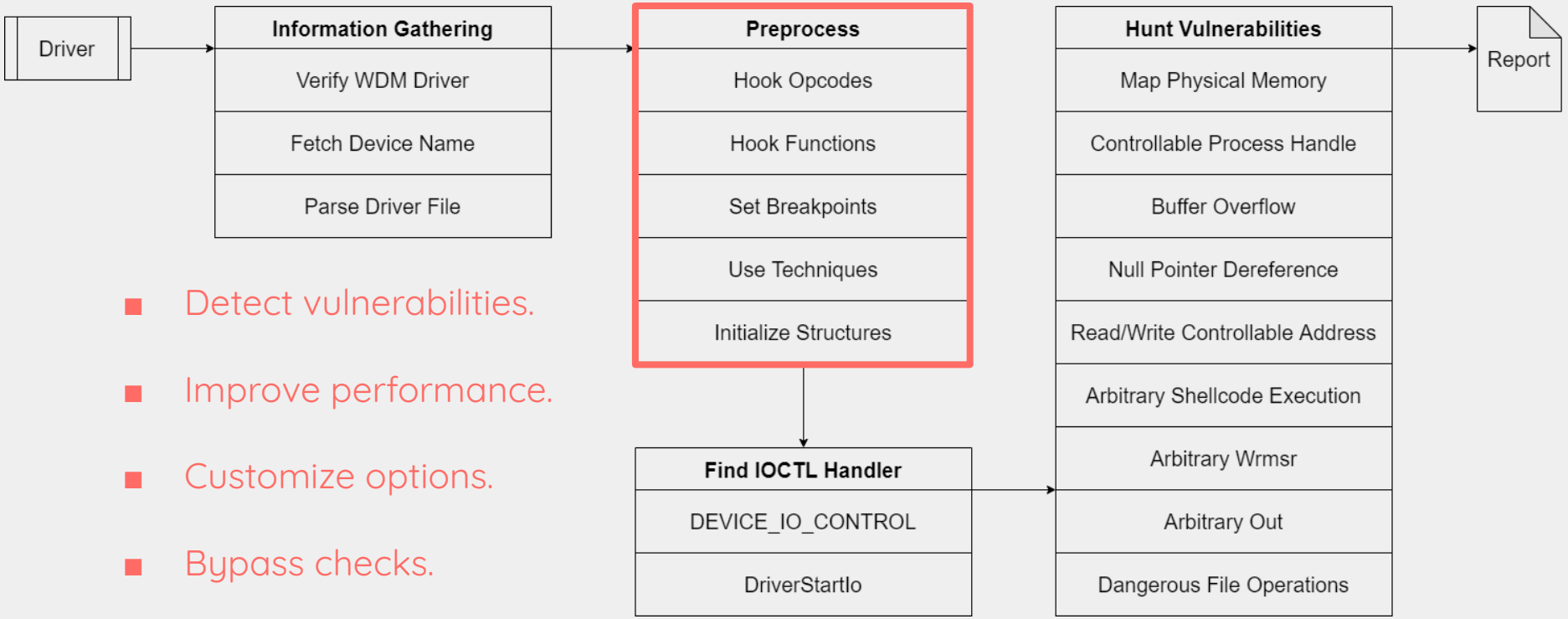
- Check the presence of IoCreateDevice in the import table.



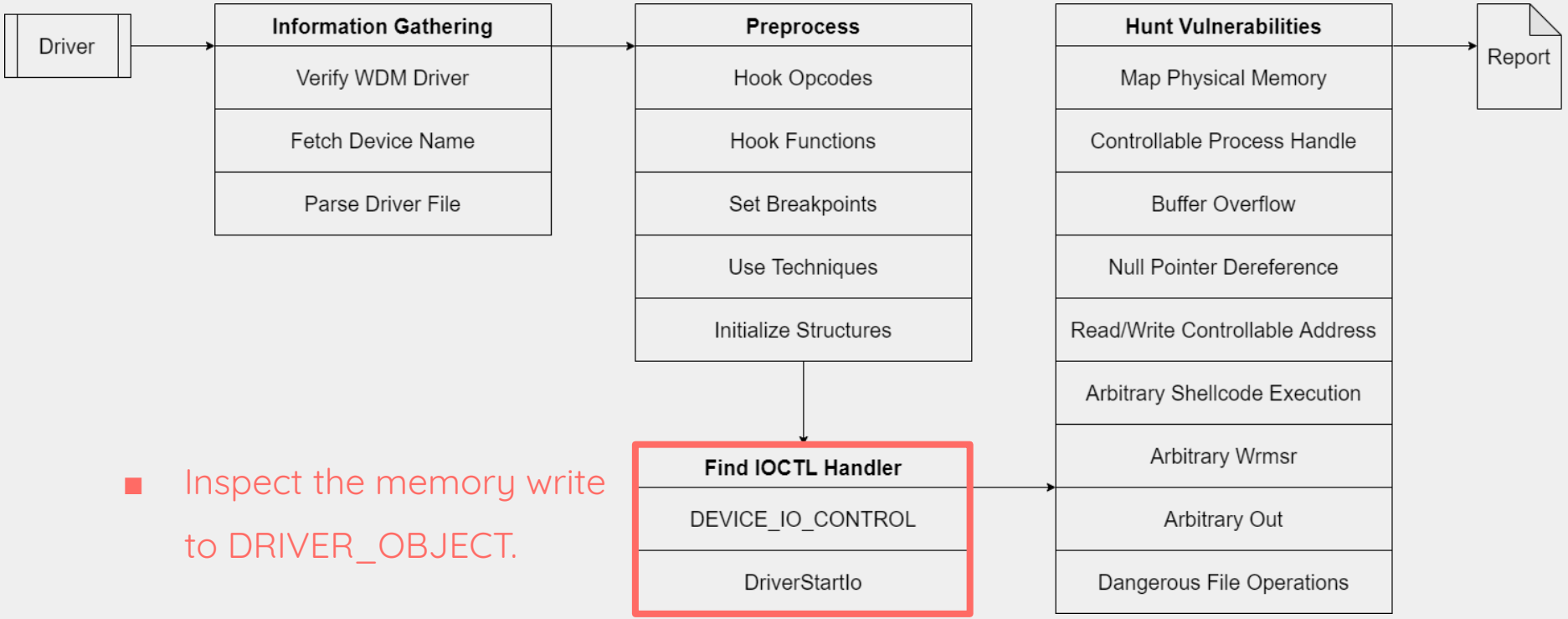
- Use a string signature to determine the target device name.



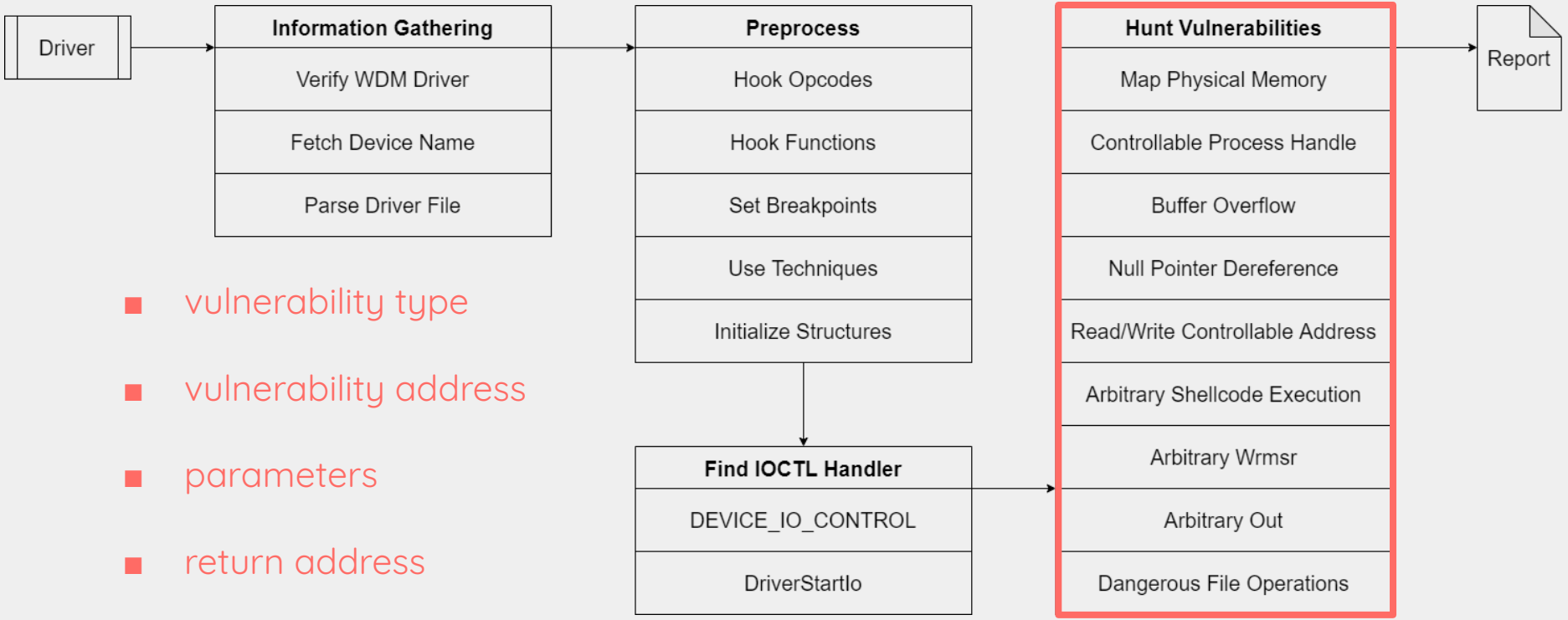
- Get information such as opcodes, function addresses, indirect jump, etc.



- Detect vulnerabilities.
- Improve performance.
- Customize options.
- Bypass checks.



- Inspect the memory write to DRIVER_OBJECT.



- vulnerability type
- vulnerability address
- parameters
- return address
- constraints

Vulnerability

vulnerability types and
real-world cases

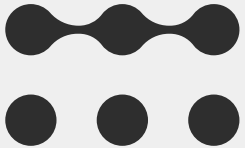
Vulnerability

- map physical memory
- controllable process handle
- buffer overflow
- null pointer dereference
- read/write controllable address
- arbitrary shellcode execution
- arbitrary wrmsr
- arbitrary out
- dangerous file operation

Map Physical Memory

MmMapIoSpace, MmMapIoSpaceEx

```
VOID Vuln_MmMapIoSpace(PVOID inbuf)
{
    MmMapIoSpace(
        *(PHYSICAL_ADDRESS *)inbuf,
        *(SIZE_T *)(inbuf + 8),
        MmNonCached
    );
}
```



controllable PhysicalAddress

controllable NumberOfBytes

Lead to elevation of privilege.

Map Physical Memory

ZwMapViewOfSection

controllable SectionHandle or \Device\PhysicalMemory

controllable BaseAddress

controllable CommitSize

controllable ViewSize

```
VOID Vuln_ZwMapViewOfSection(PVOID inbuf)
{
    ZwMapViewOfSection(
        *(HANDLE *)inbuf,
        ZwCurrentProcess(),
        *(PVOID **)(inbuf + 8),
        0,
        1000,
        0,
        1000,
        (SECTION_INHERIT)1,
        MEM_RESERVE,
        PAGE_READWRITE
    );
}
```

Lead to elevation of privilege.

CVE-2023-1679

DriverGenius, mydrivers64.sys

```
113     case 0x9C40A108:
114       v10 = ArbitraryWritePA(
115               (int *)&pIrp->AssociatedIrp.MasterIrp->Type,
116               IoStack->Parameters.Create.Options,
117               (__int64)pIrp->AssociatedIrp.MasterIrp,
118               IoStack->Parameters.Read.Length,
119               pInfo);
120       break;
```

Find IoControlCode 0x9C40A108

```
19  if ( InputLength < 0x10 )
20    return 0xC000000Di64;
21  NumberOfBytes = (unsigned int)(inbuf[2] * inbuf[3]);
22  if ( InputLength < NumberOfBytes + 16 )
23    return 0xC000000Di64;
24  MappedPA = (int *)MmMapIoSpace(*(PHYSICAL_ADDRESS *)inbuf, NumberOfBytes, MmNonCached);
25  v9 = 0;
26  switch ( inbuf[2] )
27  {
28    case 1:
29      qmemcpy(MappedPA, inbuf + 4, (unsigned int)inbuf[3]);
30      break;
```

tainted NumberOfBytes

controllable PhysicalAddress

arbitrary write

Controllable Process Handle

ZwOpenProcess

```
VOID Vuln_ZwOpenProcess(PVOID inbuf)
{
    OBJECT_ATTRIBUTES objAttr;
    CLIENT_ID clientId;
    HANDLE processHandle;
    InitializeObjectAttributes(
        &objAttr,
        NULL,
        OBJ_KERNEL_HANDLE,
        NULL,
        NULL
    );
    clientId.UniqueProcess = *(HANDLE *)inbuf;
    ZwOpenProcess(
        &processHandle,
        PROCESS_ALL_ACCESS,
        &objAttr,
        clientId
    );
}
```

not OBJ_FORCE_ACCESS_CHECK

controllable pid

tainted CLIENT_ID

Lead to broken access control.

Controllable Process Handle

ObOpenObjectByPointer

```
VOID Vuln_ObOpenObjectByPointer(PVOID inbuf)
{
    PEPROCESS p;
    HANDLE h;
    PsLookupProcessByProcessId(*(HANDLE *)inbuf, &p);
    ObOpenObjectByPointer(
        p,
        OBJ_KERNEL_HANDLE,
        NULL,
        KEY_ALL_ACCESS,
        (POBJECT_TYPE)PsProcessType,
        0,
        &h
    );
}
```

controllable pid

tainted EPROCESS

not OBJ_FORCE_ACCESS_CHECK

Lead to broken access control.

CVE-2023-1445

Twister Antivirus, fildds.sys

```
139     case 0x80112053:
140       inbuf_ = SystemBuffer;
141       inbuf_0 = ReturnSelf(*(_DWORD *)SystemBuffer);
142       *(_DWORD *)outbuf = TerminateProcess(inbuf_0, inbuf[2]);
143       *a5 = 4;
144       break;
```

Find IoControlCode 0x80112053

```
8   if ( !inbuf_0 )
9     return 0xC000000Di64;
10  ClientId.UniqueProcess = inbuf_0;
11  ClientId.UniqueThread = 0i64;
12  ObjectAttributes.Length = 48;
13  ObjectAttributes.RootDirectory = 0i64;
14  ObjectAttributes.Attributes = 512;
15  ObjectAttributes.ObjectName = 0i64;
16  ObjectAttributes.SecurityDescriptor = 0i64;
17  ObjectAttributes.SecurityQualityOfService = 0i64;
18  if ( (unsigned int)GetVersion_filnk_203() <= 0x1F4 )
19    ObjectAttributes.Attributes &= ~0x200u;
20  v6 = ZwOpenProcess(&ProcessHandle, 0x1F0601u, &ObjectAttributes, &ClientId);
21  if ( (v6 & 0x80000000) != 0 )
22    return v6;
23  if ( sub_A1850(ProcessHandle) )
24    sub_A18C0(ProcessHandle, 0);
25  v6 = ZwTerminateProcess(ProcessHandle, inbuf);
```

tainted CLIENT_ID

not OBJ_FORCE_ACCESS_CHECK

controllable process handle

terminate arbitrary process

Buffer Overflow

memcpy

```
VOID Vuln_BufferOverflow(PVOID inbuf)
{
    memcpy(dest, src, *(SIZE_T *)(inbuf));
}
```

controllable size

Lead to denial of service or elevation of privilege.

CVE-2023-1646

IObit Malware Fighter, IMFCameraProtect.sys

```
42    case 0x8018E004:
43      if ( Event )
44      {
45        memcpy(&Dst, inbuf, IoStack->Parameters.Create.Options);
46        byte_15984 = Dst == 1;
47      }
48      else
49      {
50        v5 = 0xC0000001;
51      }
52      break;
```

Find IoControlCode 0x80181E004

controllable length

stack overflow

Null Pointer Dereference

Tainted Buffer

```
VOID Vuln_NullPointerDereference_TaintedBuffer(PVOID inbuf)
{
    *(CHAR *)inbuf = 0;
}
```

Write into tainted buffer without validating the pointer.

Lead to denial of service.

Null Pointer Dereference

ExAllocatePool, ExAllocatePool2, ExAllocatePool3,
MmAllocateNonCachedMemory, ExAllocatePoolWithTag,
MmAllocateContiguousMemorySpecifyCache

```
VOID Vuln_NullPointerDereference_AllocatedMemory()
{
    memory = MmAllocateNonCachedMemory(0x1000);
    *(CHAR *)memory = 0;
}
```

Write into allocated memory without checking return value.

Lead to denial of service.

CVE-2023-1638

IObit Malware Fighter, ImfRegistryFilter.sys

```
31  if ( IoControlCode == 0x8001E024 )
32  {
33    byte_152C0 = *inbuf == 1;
34    goto LABEL_31;
35  }
```

Find IoControlCode 0x8001E024

Write into input buffer without validating the pointer.

Read/Write Controllable Address

Tainted Buffer

```
VOID Vuln_ReadWriteControllableAddress_TaintedBuffer(PVOID inbuf)
{
    **(char **)inbuf = **(char **)(inbuf + 8);
}
```

Write into tainted buffer.

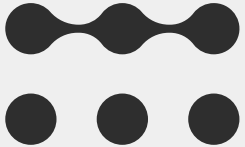
Read from tainted buffer.

Lead to denial of service or elevation of privilege.

Read/Write Controllable Address

memcpy

```
VOID Vuln_ReadWriteControllableAddress_memcpy(PVOID inbuf)
{
    memcpy(*(CHAR **)inbuf, *(CHAR **)(inbuf + 8), 4);
}
```



controllable destination address

controllable source address

Lead to denial of service or elevation of privilege.

CVE-2023-20562

AMD μProf, AMDCpuProfiler.sys

Find IoControlCode 0x222058

```
469     case 0x222058u:
470       DbgPrint("[CpuProf] Processing %s (Function: 0x%03X)...\n", "IOCTL", (IoControlCode >> 2) & 0xFFF);
471       IoStack = pIrp->Tail.Overlay.CurrentStackLocation;
472       pIrp->IoStatus.Information = 0i64;
473       if ( IoStack->Parameters.Create.Options != 0x28 )
474         goto LABEL_24;
475       OutputLength = IoStack->Parameters.Read.Length;
476       v11 = OutputLength < 0x28;
477       if ( OutputLength < 0x28 )
478         goto LABEL_23;
479       inbuf = (DWORD *)pIrp->AssociatedIrp.MasterIrp;
480       status = OutputLength < 0x28 ? 0xC0000023 : 0;
481       inbuf_0 = *inbuf;
482       if ( (unsigned int)inbuf_0 < 8
483         && (item = (__int64)&DeviceExtension_v5[2376 * inbuf_0 + 32]) != 0
484         && *(_QWORD *)(item + 136) )
485       {
486         inbuf_32 = inbuf[8];
487         inbuf_24 = (_IRP *)*((_QWORD *)inbuf + 3);
488         if ( !*(_QWORD *)item
489           || !*(_QWORD *)(item + 104)
490           || !(unsigned int)IOCTL_GET_OUTPUT_FILE((void **)item, *((_OWORD **)inbuf + 1), inbuf[4])
```

Call IOCTL_GET_OUTPUT_FILE with input buffer

CVE-2023-20562

AMD μProf, AMDCpuProfiler.sys

```
11  v3 = *item;
12  if ( !v3 )
13    return 0i64;
14  FileInformation = 0;
15  filepath[0] = 0;
16  if ( ZwQueryInformationFile(v3, &IoStatusBlock, &FileInformation, 0x20Eu, FileNameInformation) )
17    return 0i64;
18  if ( IoStatusBlock.Status )
19    return 0i64;
20  len_v6 = FileInformation;
21  if ( FileInformation <= 2 )
22    return 0i64;
23  if ( FileInformation >= 2 * inbuf_16 )
24    len_v6 = 2 * inbuf_16;
25  v7 = len_v6;
26  memcpy(inbuf_8, filepath, len_v6);
27  result = v7 >> 1;
28  *((_WORD *)inbuf_8 + result) = 0;
29  return result;
```

controllable destination address

Arbitrary Shellcode Execution

Breakpoint: Call

```
VOID Vuln_ArbitraryShellcodeExecution(PVOID inbuf)
{
    ((VOID (*)())inbuf)();
}
```

controllable function pointer

Lead to arbitrary kernel code execution.

CVE-2023-36759

Visual Studio 2022, pgodriver.sys

```
9   if ( CurrentStackLocation->Parameters.Read.ByteOffset.LowPart == 0x156002 )
10  {
11    v5 = IOCTL_0x156002(pIrp, CurrentStackLocation);
12    goto LABEL_6;
13  }
```

Find IoControlCode 0x156002

call IOCTL_0x156002

```
15  pIrp->IoStatus.Information = 0i64;
16  log("IrtSweep\n");
17  if ( IoStackLocaiton_a2->Parameters.Create.Options == 8 )
18  {
19    if ( IoStackLocaiton_a2->Parameters.Read.Length >= 0xC4 )
20    {
21      inbuf_0 = ExecuteCode(pIrp);
```

IRP contains input buffer.

```
1  _DWORD *__fastcall ExecuteCode(PIRP pIrp)
2  {
3    __int64 *inbuf; // rcx
4    _DWORD *inbuf_0; // rax
5    __int64 code; // rax
6    _DWORD *v4; // [rsp+30h] [rbp+8h] BYREF
7
8    inbuf = (__int64 *)&pIrp->AssociatedIrp.MasterIrp->Type;
9    inbuf_0 = (_DWORD *)*inbuf;
10   v4 = inbuf_0;
11   if ( !inbuf_0 )
12     return 0i64;
13   if ( *inbuf_0 != 'POGO' )
14   {
15     code = Parse(*inbuf);
16     if ( !code )
17       return 0i64;
18     v4 = 0i64;
19     ((void (__fastcall *)(_DWORD **))code)(&v4);
20     inbuf_0 = v4;
21     if ( !v4 || *v4 != 'POGO' )
22       return 0i64;
23   }
24   return inbuf_0;
25 }
```

call Parse

arbitrary code execution

return code

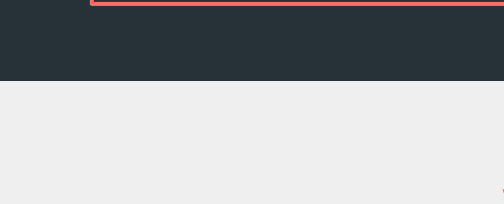
```
1  __int64 __fastcall Parse(__int64 inbuf_0)
2  {
3    __int64 inbuf_0_60; // rdx
4    __int64 v2; // r9
5    __int64 code; // r10
6    int length; // r11d
7    _DWORD *signature; // r8
8
9    inbuf_0_60 = *(int *)(inbuf_0 + 0x3C);
10   v2 = 0i64;
11   code = 0i64;
12   length = *(unsigned __int16 *)(inbuf_0_60 + inbuf_0 + 6);
13   signature = (_DWORD *)(*(unsigned __int16 *)(inbuf_0_60 + inbuf_0 + 20) + inbuf_0_60 + inbuf_0 + 24);
14   if ( *(_WORD *)(inbuf_0_60 + inbuf_0 + 6) )
15   {
16     do
17     {
18       if ( *signature == 'OGOP' && signature[1] == 'EDOC' )
19         code = inbuf_0 + *(unsigned int *)((unsigned int)signature[3] + inbuf_0);
20       signature += 10;
21       --length;
22     }
23     while ( length );
24   }
25   if ( (unsigned __int64)(code - inbuf_0) <= 0x8000000 )
26     return code;
27   return v2;
28 }
```

tainted code

Arbitrary Wrmsr

__writemsr

```
VOID Vuln_ArbitraryWrmsr(PVOID inbuf)
{
    __writemsr(*(int *)inbuf, *(unsigned __int64 *)(inbuf + 8));
}
```



controllable Register and Value

Lead to arbitrary kernel code execution.

CVE-2023-1489

Wise System Monitor, WiseHDInfo64.dll

```
72          else if ( IoControlCode == 0x9C402088 )
73          {
74            Status = ArbitraryWriteMsr((__int64)pIrp->AssociatedIrp.MasterIrp, IoStatus);
75          }
```

Find IoControlCode 0x9C402088

call ArbitraryWriteMsr

```
1 __int64 __fastcall ArbitraryWriteMsr(DWORD *inbuf, _DWORD *IoStatus)
2 {
3   __writemsr(*inbuf, *(_QWORD *)(inbuf + 1));
4   *IoStatus = 0;
5   return 0i64;
6 }
```

controllable Register and Value

Arbitrary Out

__outbyte, __outword, __outdword

```
VOID Vuln_ArbitraryOut(PVOID inbuf)
{
    __outbyte(*(USHORT *)inbuf, *(UCHAR *)(inbuf + 8));
}
```

controllable Port and Data

Lead to denial of service.

CVE-2023-2870

EnTech Taiwan, Se64a.sys

```
86      case 0x80002014:
87        if ( InputLength == 12 )
88        {
89          inbuf_0 = *(_DWORD *)inbuf;
90          if ( *((_DWORD *)inbuf + 2) == 1 )
91            goto LABEL_19;
92          if ( *((_DWORD *)inbuf + 2) == 2 )
93          {
94            __outword(inbuf_0, inbuf[2]);
95            irp->IoStatus.Information = 0i64;
96            break;
97          }
```

Find IoControlCode 0x80002014

controllable Port and Data

Dangerous File Operation

ZwDeleteFile, ZwOpenFile, ZwCreateFile, IoCreateFile,
IoCreateFileEx, IoCreateFileSpecifyDeviceObjectHint

controllable ObjectName

not OBJ_FORCE_ACCESS_CHECK

```
VOID Vuln_DangerousFileOperation(PVOID inbuf)
{
    OBJECT_ATTRIBUTES ObjectAttributes;
    InitializeObjectAttributes(
        &ObjectAttributes,
        (PUNICODE_STRING)inbuf,
        OBJ_CASE_INSENSITIVE | OBJ_KERNEL_HANDLE,
        NULL,
        NULL
    );
    ZwDeleteFile(&ObjectAttributes);
}
```

Lead to broken access control.

CVE-2023-1453

Watchdog Anti-Virus, wsdk-driver.sys

```
23          case 0x80002008:
24            IOCTL_0x80002008(inbuf);
25            break;
```

Find IoControlCode 0x80002008

```
1  NTSTATUS __fastcall IOCTL_0x80002008(const WCHAR *inbuf)
2  {
3    _UNICODE_STRING DestinationString; // [rsp+20h] [rbp-28h] BYREF
4    _UNICODE_STRING inbuf_unicode_string; // [rsp+30h] [rbp-18h] BYREF
5
6    if ( !inbuf )
7      return 0xC0000001;
8    RtlInitUnicodeString(&DestinationString, inbuf + 2);
9    inbuf_unicode_string = DestinationString;
10   return DeleteFile(&inbuf_unicode_string);
11 }
```

tainted unicode string

CVE-2023-1453

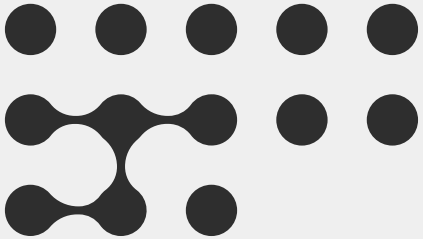
Watchdog Anti-Virus, wsdk-driver.sys

```
1  NTSTATUS __fastcall DeleteFile(_UNICODE_STRING *inbuf_unicode_string)
2  {
3    NTSTATUS result; // eax
4    struct _OBJECT_ATTRIBUTES ObjectAttributes; // [rsp+20h] [rbp-38h] BYREF
5
6    result = 0xC0000001;
7    if ( inbuf_unicode_string->Length )
8    {
9      ObjectAttributes.ObjectName = inbuf_unicode_string;
10     ObjectAttributes.Length = 48;
11     *(_OWORD *)&ObjectAttributes.SecurityDescriptor = 0i64;
12     ObjectAttributes.RootDirectory = 0i64;
13     ObjectAttributes.Attributes = 576;
14     result = ZwDeleteFile(&ObjectAttributes);
15     if ( result >= 0 )
16       return 1;
17   }
18   return result;
19 }
```

tainted ObjectName

not OBJ_FORCE_ACCESS_CHECK

Dangerous File Operation



04

Enhancement

- enhancement made to get better performance

Enhancement

- Improve Opcodes
- Improve Functions
- Constraints
- Bypass Checks
- Customization

Improve Opcodes

rep

Evaluate the **size** for rep and set **thresholds**.

indirect jump

Evaluate the **target address** of indirect jumps.

call

Evaluate the **symbolic function address**.

Improve Functions

memset & memcpy

```
result = Dst;
if ( Size < 8 )
{
  for ( ; Size; --Size )
    *((char *)Dst + Size - 1) = Val;
}
else
{
  v4 = 0x1010101010101010i64 * (unsigned __int8)Val;
  if ( Size >= 0x4F )
  {
    v8 = _mm_movelh_ps((__m128)(unsigned __int64)v4, (__m128)(unsigned __int64)v4);
    *(__m128 *)Dst = v8;
    v9 = (char *)Dst + Size;
    v10 = (__m128 *)(((unsigned __int64)Dst + 16) & 0xFFFFFFFFFFFFFF0ui64);
    v11 = v9 - (char *)v10;
    v12 = v11 >> 7;
    if ( v11 >> 7 )
    {
      do
      {
        *v10 = v8;
        v10[1] = v8;
        v10 += 8;
        v10[-6] = v8;
        v10[-5] = v8;
        --v12;
        v10[-4] = v8;
        v10[-3] = v8;
        v10[-2] = v8;
        v10[-1] = v8;
      }
      while ( v12 );
      v11 &= 0x7Fu;
    }
    for ( i = v11 >> 4; i; --i )
      *v10++ = v8;
    v14 = v11 & 0xF;
    if ( v14 )
      *(__m128 *)((char *)v10 + v14 - 16) = v8;
  }
  else
  {
```

memset

```
v3 = (__m128i *)Dst;
v4 = Src < Dst;
v5 = (_BYTE *)Src - (_BYTE *)Dst;
if ( v4 )
{
  v19 = (__m128i *)((char *)Dst + MaxCount);
  if ( MaxCount >= 0x4F )
  {
    if ( v5 > 0xFFFFFFFFFFFFFF0ui64 )
    {
      for ( ; ((unsigned __int8)v19 & 0xF) != 0; v19->m128i_i8[0] = v19->m128i_i8[v5] )
      {
        v19 = (__m128i *)((char *)v19 - 1);
        --MaxCount;
      }
    }
    else
    {
      v24 = v19;
      v25 = (unsigned __int8)v19 & 0xF;
      if ( (_DWORD)v25 )
      {
        MaxCount -= (unsigned int)v25;
        v25 = -(__int64)(unsigned int)v25;
        v24[-1] = _mm_loadu_si128((__m128i *)((char *)v24 + v5 - 16));
      }
      v19 = (__m128i *)((char *)v24 + v25);
      v26 = MaxCount >> 7;
      if ( MaxCount >> 7 <= 0x2000 || v5 > 0xFFFFFFFFFFFFFE00ui64 )
        goto LABEL_38;
      do
      {
        v29 = 4;
        do
        {
          v19 -= 8;
          _mm_prefetch(&v19->m128i_i8[v5], 0);
          _mm_prefetch(&v19[4].m128i_i8[v5], 0);
          --v29;
        }
        while ( v29 );
        v19 += 32;
```

memcpy

complicated

Contraints

Restricted Address: Tag the restricted tainted buffer.

MmIsAddressValid

Check if a given virtual address will cause a **page fault** during a **read** or **write** operation.

ProbeForRead

Check if a buffer is located in **user mode** and is **readable**.

ProbeForWrite

Check if a buffer is located in **user mode** and is **writable**.

Bypass Checks

PsGetVersion & RtlGetVersion

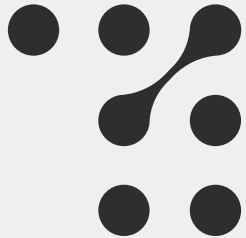
Set symbolic variable to the output value to **bypass conditions** of the **version**.

MmGetSystemRoutineAddress & FltGetRoutineAddress

Resolve the routine name and return a **SimProcedure**.

Customization

Optional



Length Limit

stop old states

IoControlCode Timeout

longer for complex IOCTL

Loop Bound

neglect concrete loops

Recursion

neglect complicated recursion

Total Timeout

longer for complex program

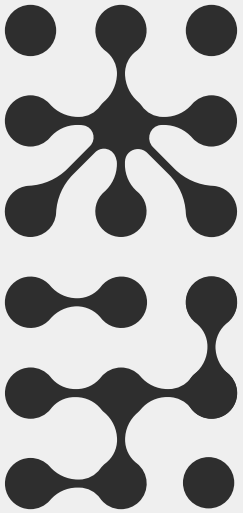
Symbolize Data Section

bypass checks e.g. global var

05

Evaluation

- evaluation with both known and unknown drivers



Known Drivers

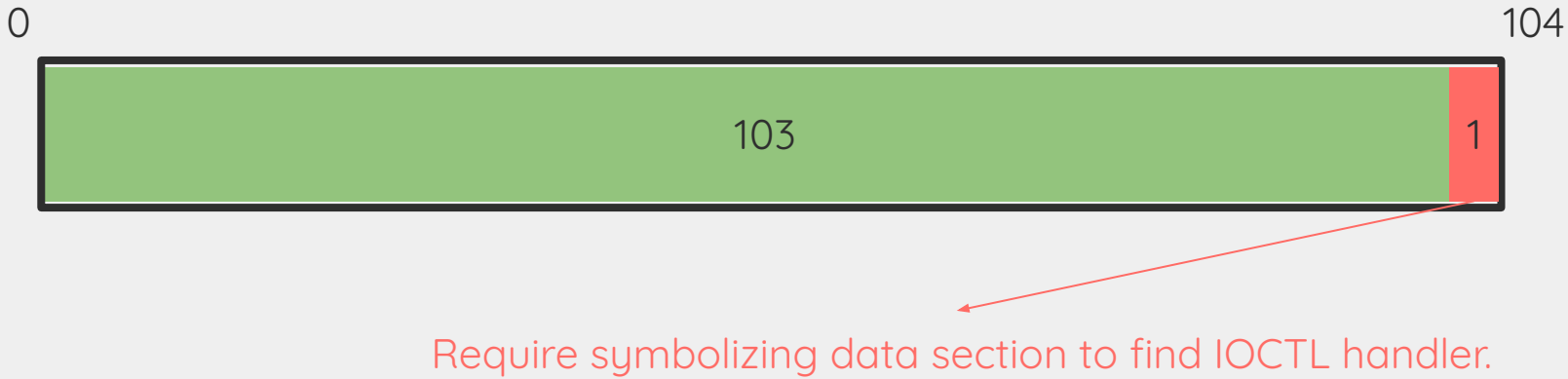
Obtain vulnerable WDM drivers from **namazso/physmem_drivers** and **CaledoniaProject/drivers-binaries**.

104	10.1 KB
unique vulnerable drivers	smallest known driver
1.3 MB	62.4 KB
largest known driver	average size

Known Drivers: Performance

30 minutes Total Timeout

40 seconds IoControlCode Timeout

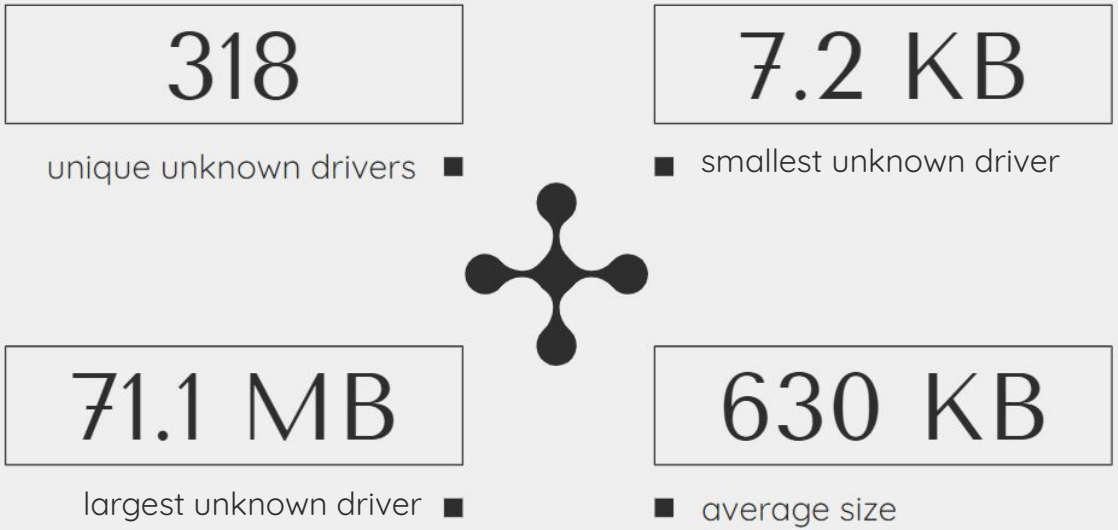


Known Drivers: Vulnerabilities

Vulnerability Types	Ground Truth	Experiment
map physical memory	221	217
controllable process handle	12	12
read/write controllable address	104	103
buffer overflow	65	65
null pointer dereference	818	813
arbitrary shellcode execution	8	8
arbitrary wrmsr	49	49
arbitrary out	179	172
dangerous file operation	6	4

Unknown Drivers

Obtain drivers by manually downloading various software.



Unknown Drivers: Performance

30 minutes Total Timeout

40 seconds IoControlCode Timeout

Unknown Drivers: Vulnerabilities

Vulnerability Types	Result
map physical memory	19
controllable process handle	1
read/write controllable address	18
buffer overflow	5
null pointer dereference	55
arbitrary shellcode execution	1
arbitrary wrmsr	2
arbitrary out	12
dangerous file operation	4

- 117 unknown vulnerabilities
- 26 unique drivers
- 41 CVEs

Few Vulns Found by IOCTLance

CVE-2023-1186, CVE-2023-1188, CVE-2023-1189, CVE-2023-1453, CVE-2023-1486, CVE-2023-1487, CVE-2023-1488, CVE-2023-1489, CVE-2023-1490, CVE-2023-1491, CVE-2023-1492, CVE-2023-1493, CVE-2023-1629, CVE-2023-1630, CVE-2023-20556, CVE-2023-20561, CVE-2023-20562, CVE-2023-20560, CVE-2023-20564, CVE-2023-1638, CVE-2023-1640, CVE-2023-1641, CVE-2023-1643, CVE-2023-1644, CVE-2023-1645, CVE-2023-1646, CVE-2023-1676, CVE-2023-1677, CVE-2023-1678, CVE-2023-1679, CVE-2023-28262, CVE-2023-28263, CVE-2023-2870, CVE-2023-2871, CVE-2023-2872, CVE-2023-1445, CVE-2023-2873, CVE-2023-2874, CVE-2023-2875, CVE-2023-36758, CVE-2023-36759

Manual Investigation

```
void fp1(PVOID inbuf)
{
    __try
    {
        // false positive: NPD
        *inbuf = 0;
    }
    __except(EXCEPTION_EXECUTE_HANDLER)
    {
    }
}
```

try-except

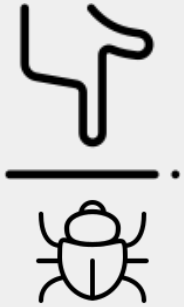
```
void fp2(PVOID inbuf)
{
    DestStr = NULL;
    RtlInitUnicodeString(&DestStr, inbuf);
    if (DestStr)
    {
        // false positive: NPD
        *inbuf = 0;
    }
}
```

indirect check

Manual Investigation

administrator-only

kernel-only



hard to trigger

06

Conclusion

Conclusion

- WDM drivers can pose a significant **security risk** to a system because of their **kernel-level privileges**.
- IOCTLance targets several vulnerability types and makes various enhancement to get better performance.
- IOCTLance has uncovered **117 vulnerabilities** that were previously unknown in **26 unique drivers**, resulting in the assignment of **41 CVEs**.

THANKS

Do you have any questions?

zeze@teamt5.org

+886 989325139