



1-Click-Fuzz

: Systematically Fuzzing
the Windows Kernel Driver with Symbolic Execution

Sangjun Park, Jongseong Kim, Yunjin Park

About us



- **Sangjun Park** サンジュン・パク

- M.S. Student at KAIST (Software Security Lab)
- B.S. at Soongsil University
- Fuzzing Researching



- **Yunjin Park** ユンジン・パク

- B.S. Student at Seoul Women's University
- Interested in Enterprise Security & Red teaming
- Linkedin @yun-jin-park



- **Jongseong Kim** ジョンソン・キム

- ENKI WhiteHat, Security Researcher
- B.S. Student at Ajou University
- Linkedin @jongseongkim

Agenda

1

Introduction to **Windows Kernel Driver**

2

Backgrounds

3

Design of **msFuzz**

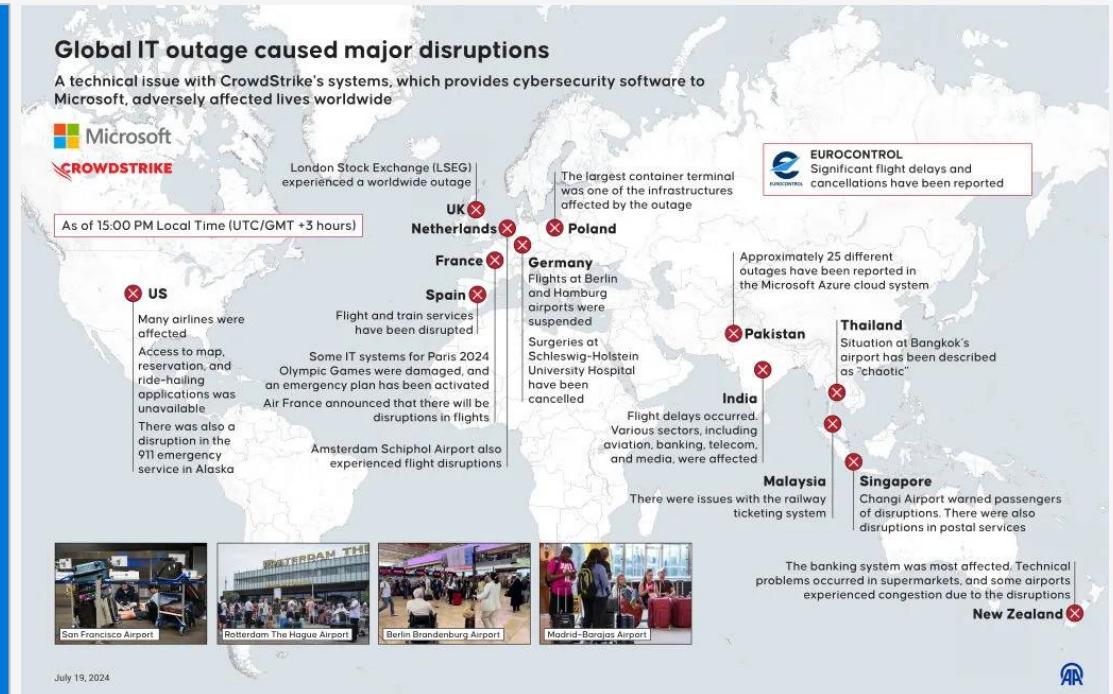
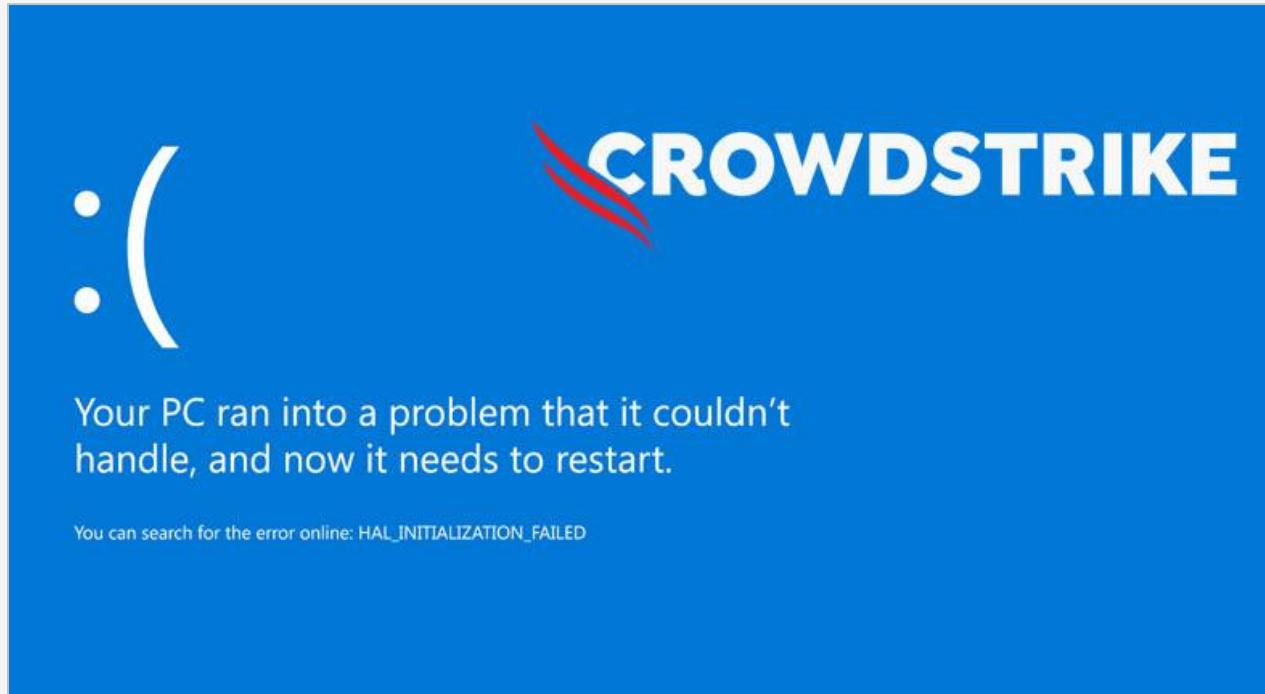
4

Conclusion



Crowdstrike Global IT Outages

- On 19 July, CrowdStrike faculty update caused BSOD all over the world
- What if LPE/RCE attack was possible, not just a BSOD?



Exploited Windows Kernel Driver

- Hackers also know about it and abuse this!

Windows driver zero-day exploited by Lazarus hackers to install rootkit

By Lawrence Abrams

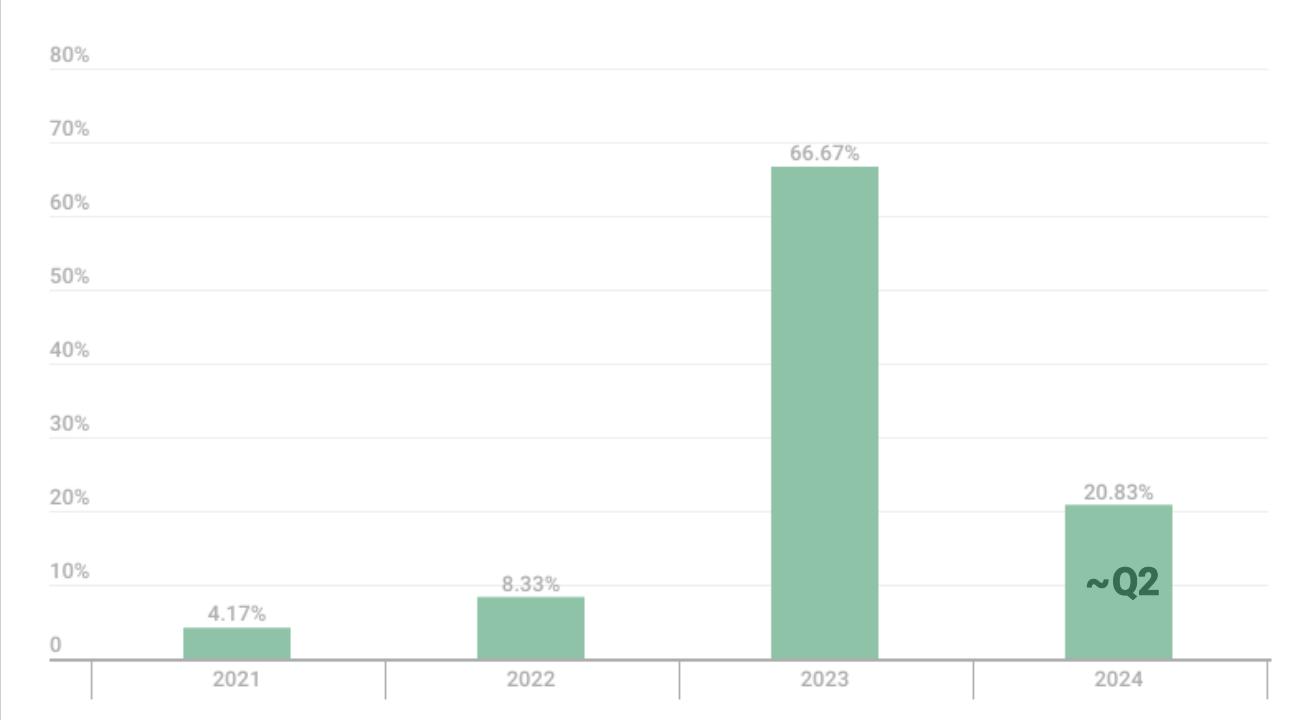
August 19, 2024 11:32 PM 2



Image: Midjourney

The notorious North Korean Lazarus hacking group exploited a zero-day flaw in the Windows AFD.sys driver to elevate privileges and install the FUDModule rootkit on targeted systems.

Microsoft fixed the flaw, tracked as CVE-2024-38193 during its [August 2024 Patch Tuesday](#), along with seven other zero-day vulnerabilities.

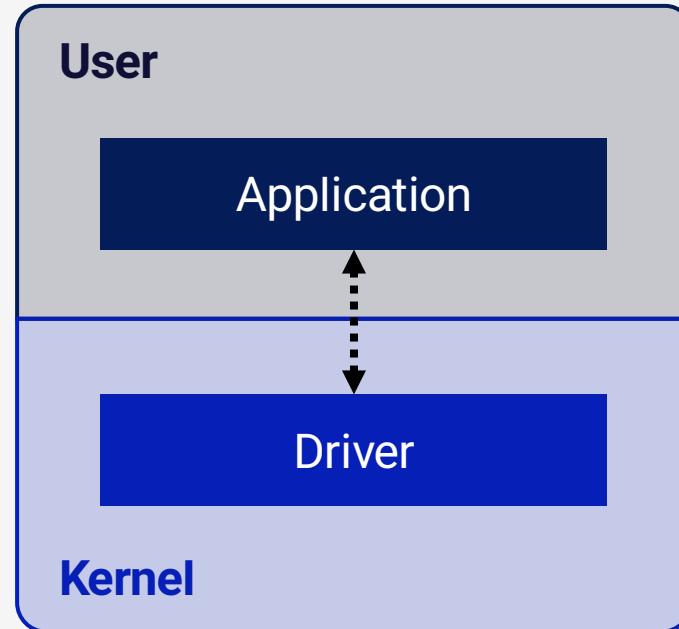


Year	Percentage
2021	4.17%
2022	8.33%
2023	66.67%
2024 (~Q2)	20.83%

kaspersky

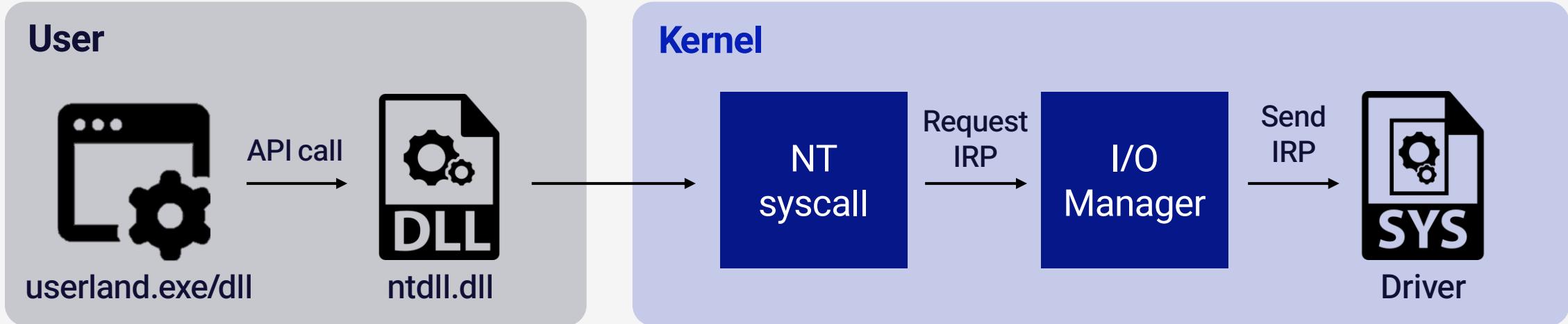
Windows Kernel Driver

- User Application interacts with the kernel Driver
 - allowing applications to indirectly use high-privilege functions



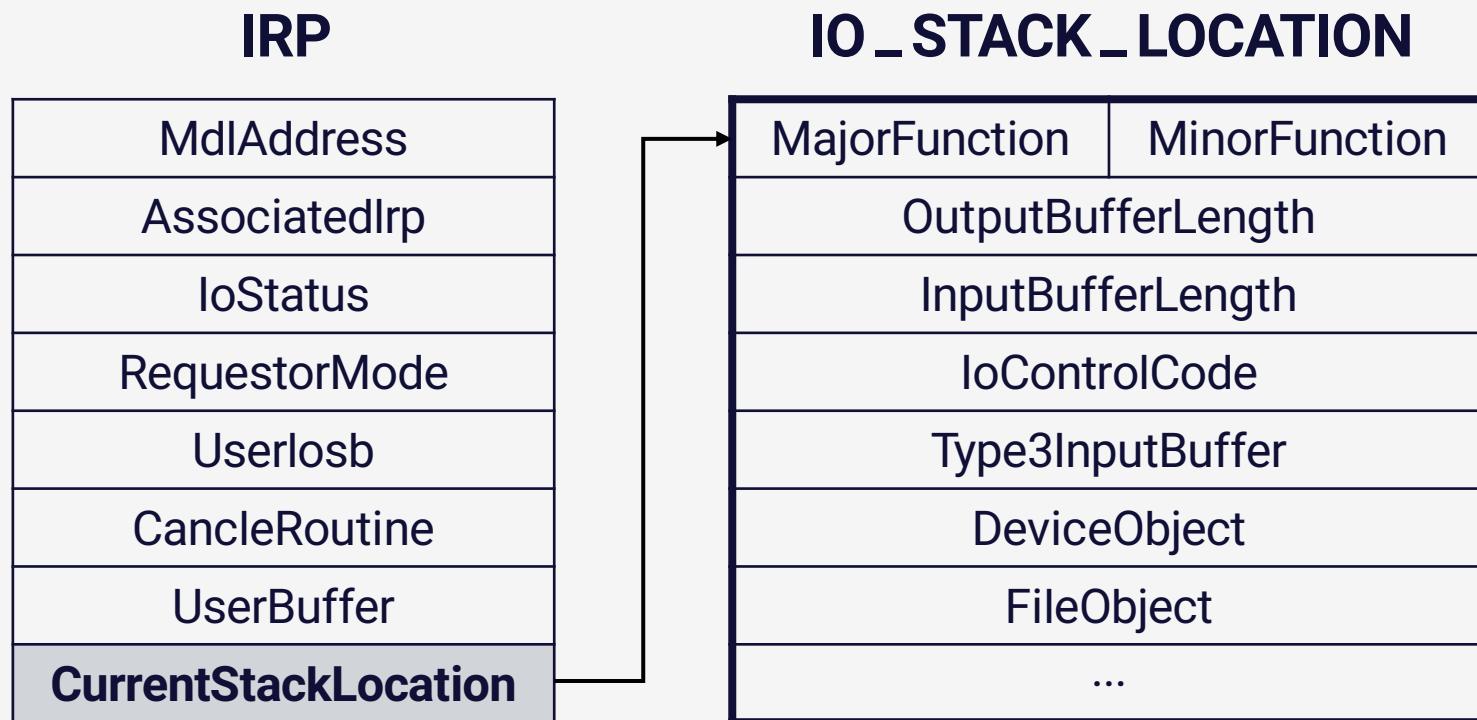
How to interact with driver?

- Obtain a handle to the driver
- Sends command to the driver using a handle
- The I/O manager creates an IRP and passes it to the driver for processing



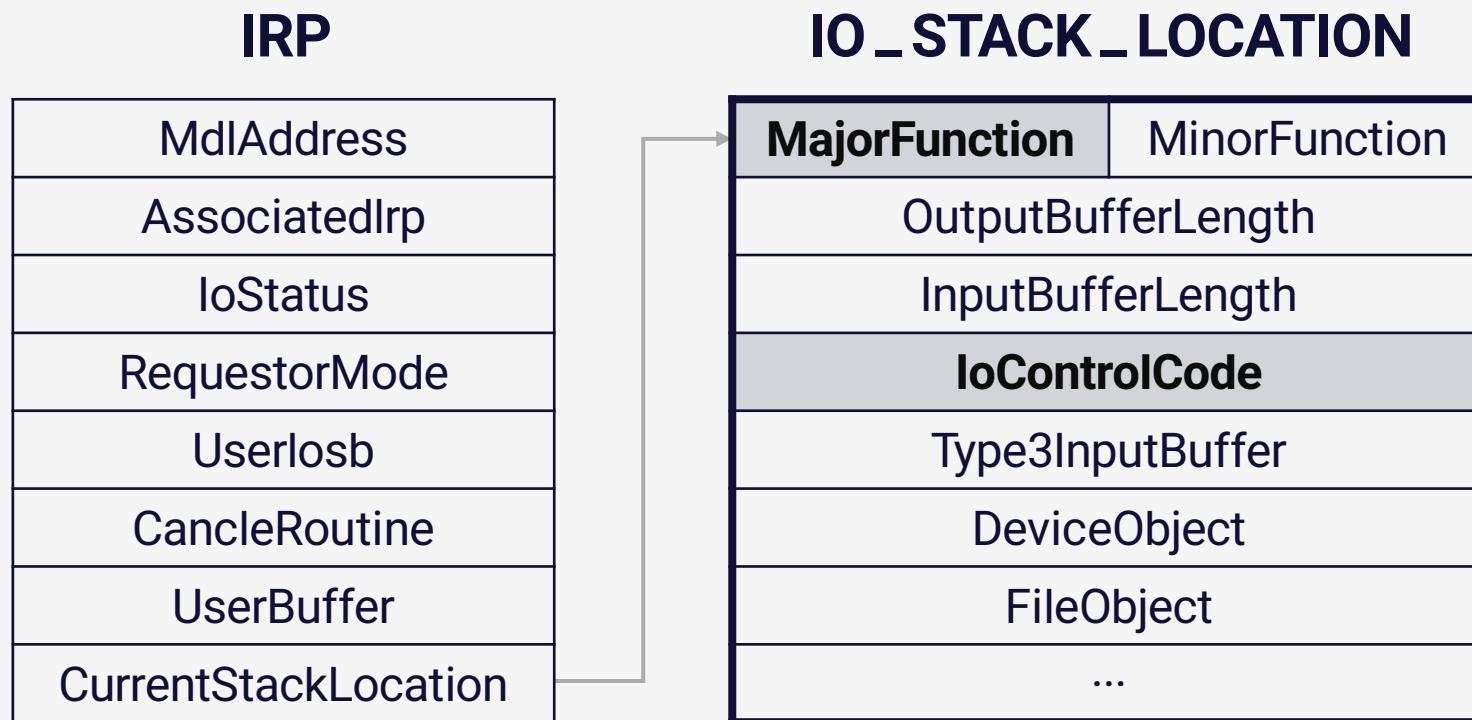
What is IRP?

- IRP is a **kernel structure** used to process I/O requests between the OS and drivers



IRP Major Function

- Major function code tells the driver on the operation to perform for the I/O request
ex) open, read, write, Device Control
- The complex Device Control routine has led to many vulnerabilities



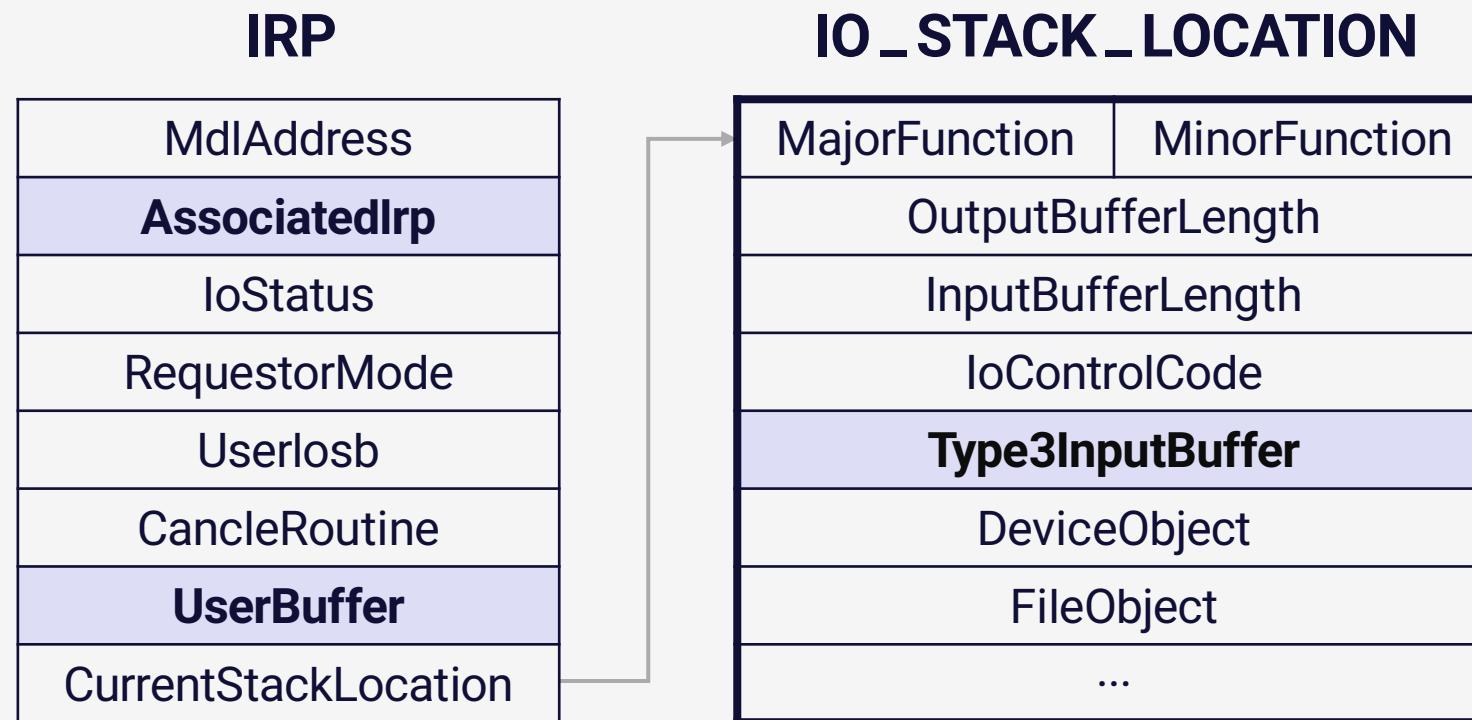
DeviceIoControl

- The user needs to fill in the function arguments to send a request
ex) IoControlCode, InputBuffer, InputBufferLength

```
BOOL DeviceControl(
    [in]                 HANDLE     hDevice,
    [in]                 DWORD      IoControlCode,
    [in]                 LPVOID     InputBuffer,
    [in]                 DWORD      InputBufferLength,
    [out]                LPVOID     OutputBuffer,
    [in]                 DWORD      OutputBufferLength,
    [out]                LPDWORD    BytesReturned,
    [in, out, optional] LPOVERLAPPED OverLapped
)
```

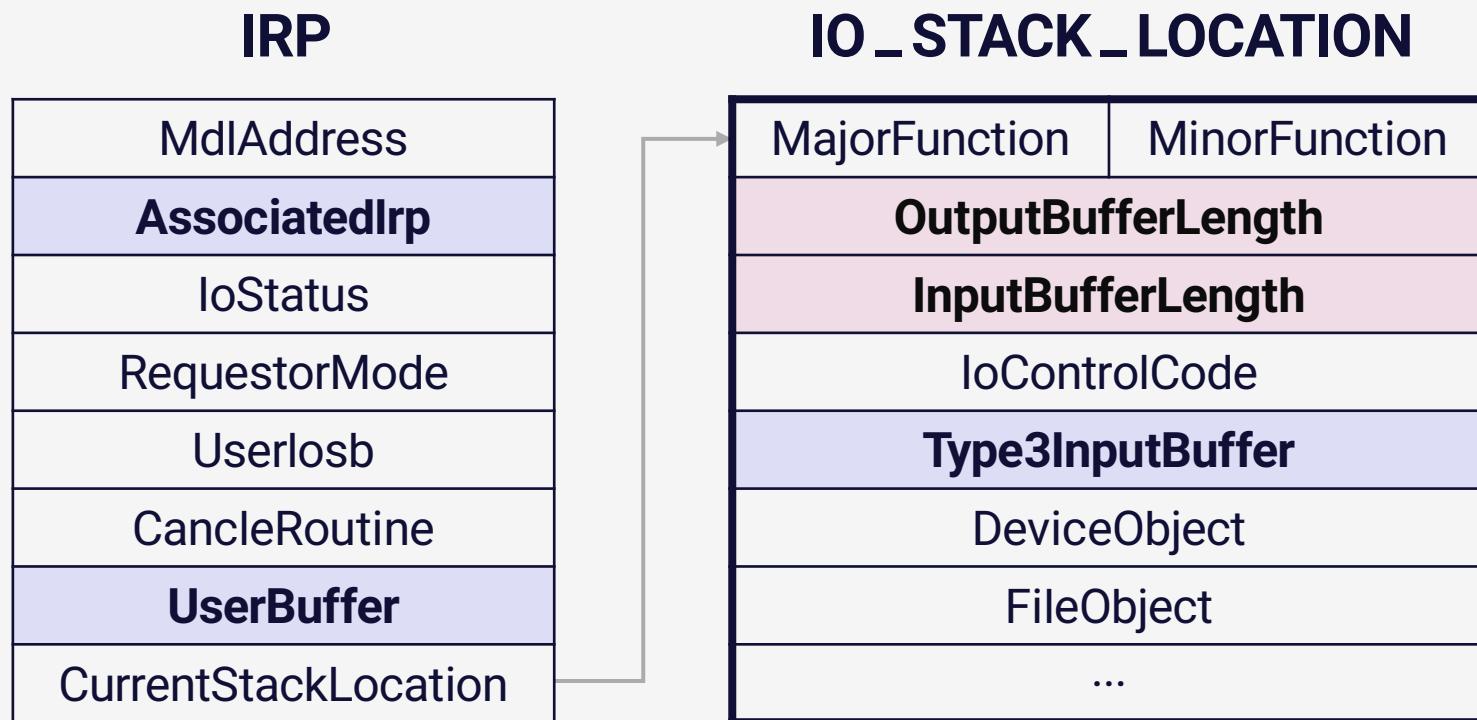
InputBuffer/OutputBuffer

- The passed data stored in IRP when requesting DeviceIoControl commands



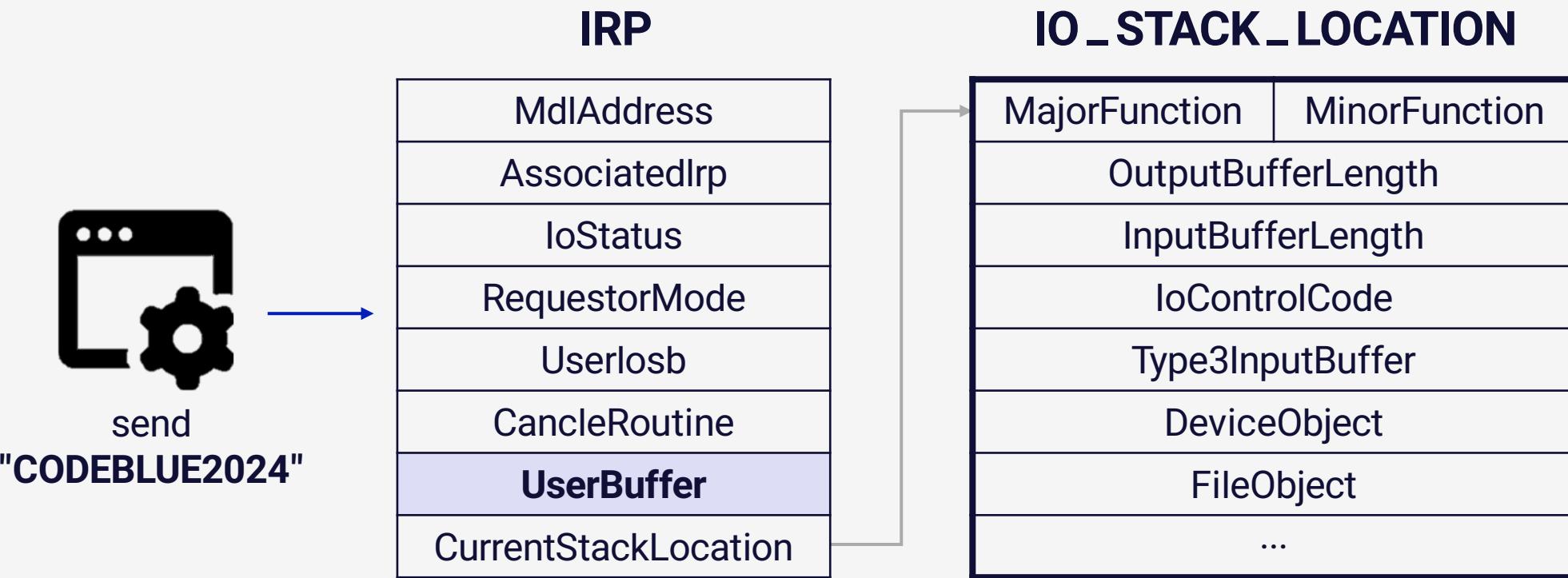
Buffer Length

- IRP contains BufferLength field



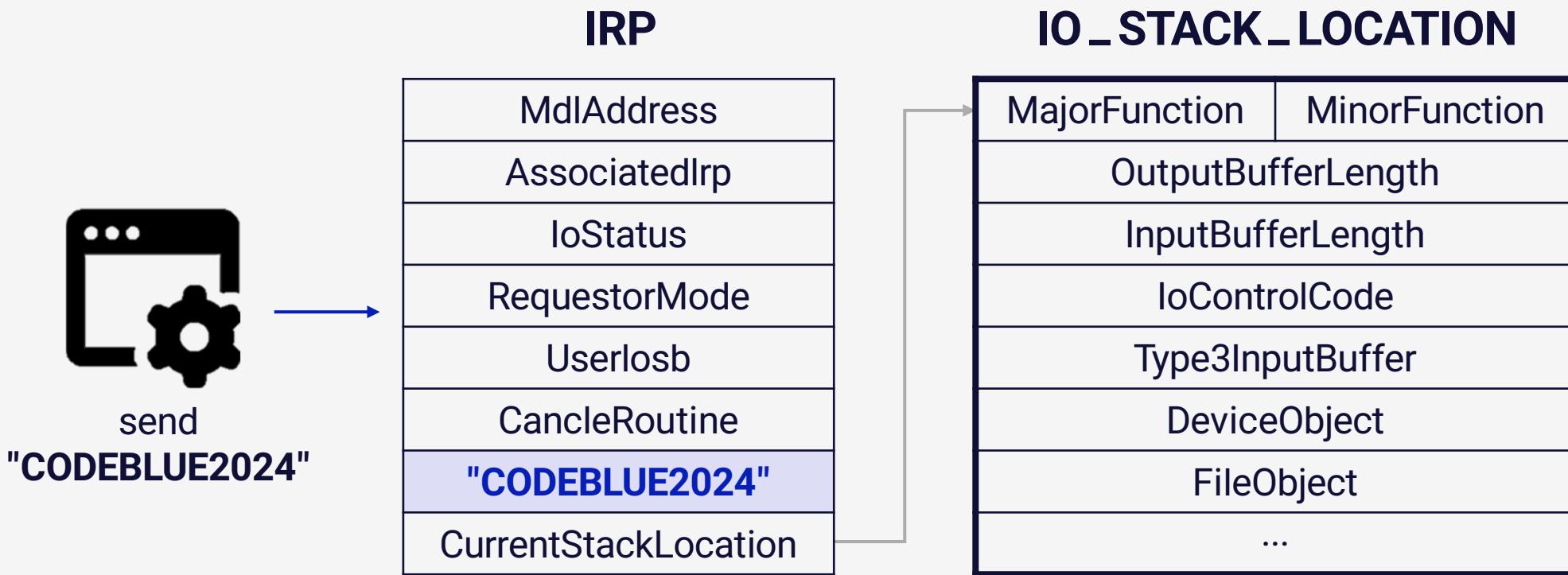
I/O Request Example

- When I/O command is requested with "CODEBLUE2024"



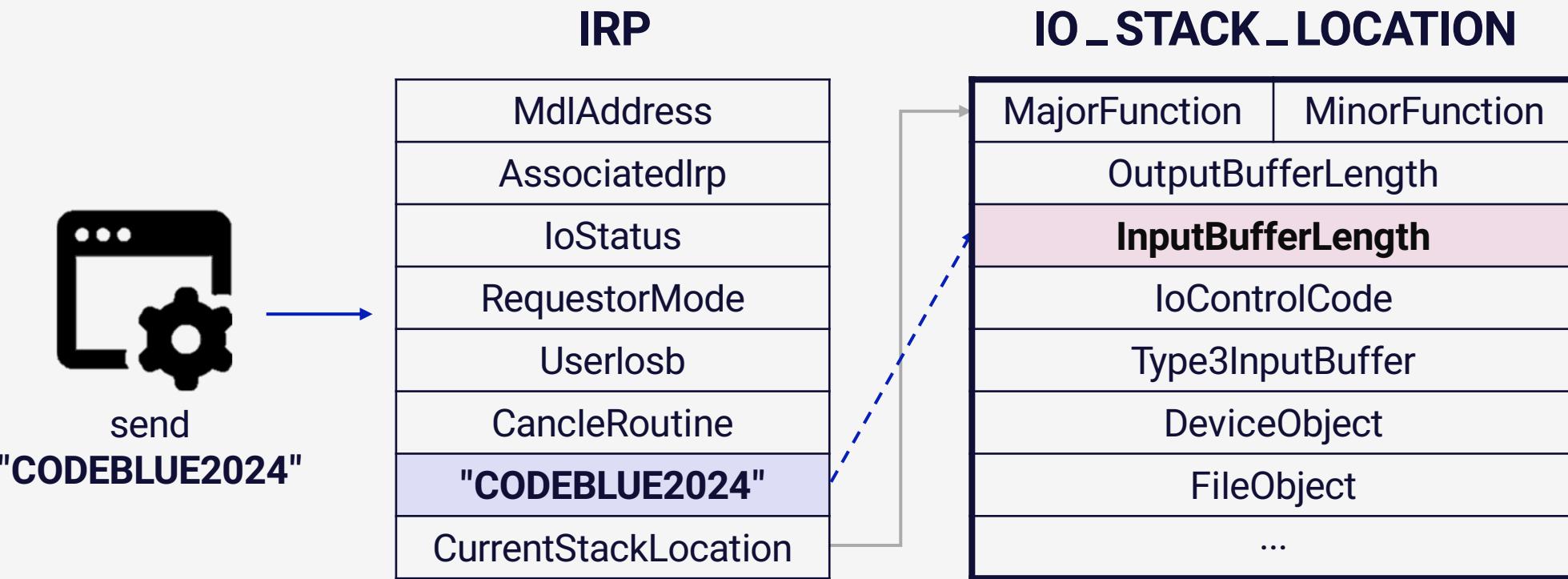
I/O Request Example

- When I/O command is requested with "CODEBLUE2024"



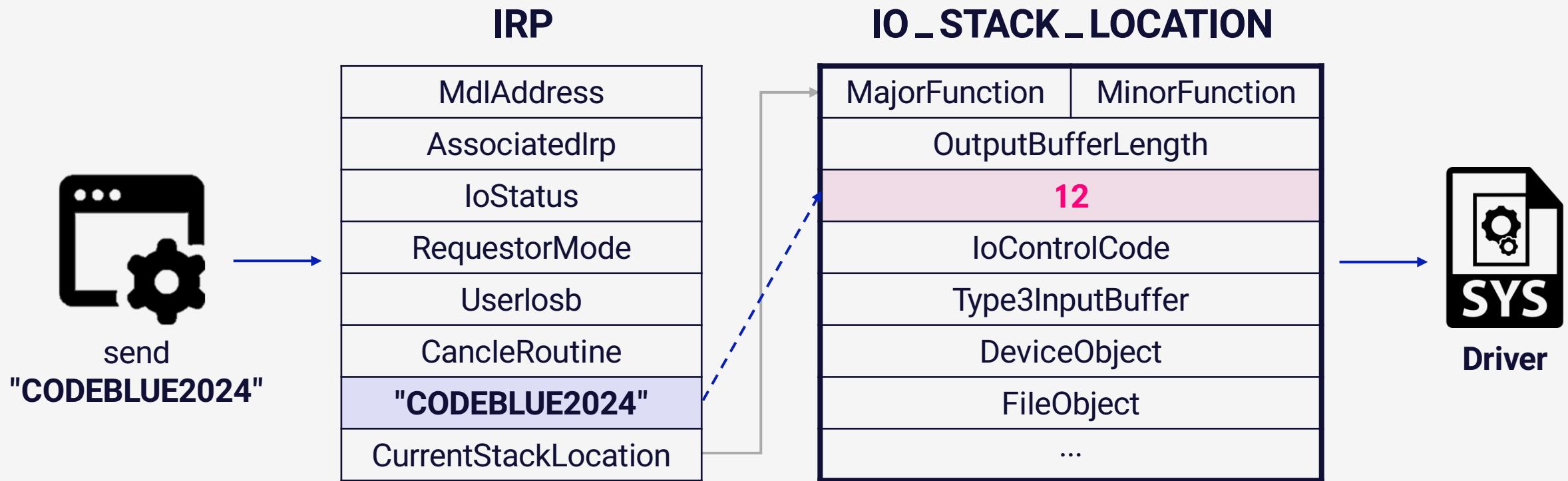
I/O Request Example

- When I/O command is requested with "CODEBLUE2024"



I/O Request Example

- When I/O command is requested with "CODEBLUE2024"



IRP Processing in a Driver

- Most driver validates the buffer length

```
case 0x222004u:  
    v5 = sub_1400075F0(IRP, CurrentStackLocation);  
    IRP->IoStatus.Status = v5;  
    IofCompleteRequest(a2, 0);  
  
    return;
```

sub_1400075F0

```
__int64 __fastcall sub_1400075F0(_IRP *a1, _IO_STACK_LOCATION *a2)  
{  
    unsigned int v7;  
  
    a1->IoStatus.Information = 0i64;  
    v7 = 0xC0000004;  
    if ( a2->Parameters.DeviceIoControl1.InputBufferLength == 4 )  
    {  
        if ( a2->Parameters.DeviceIoControl2.OutputBufferLength == 4 )  
            ProbeForWrite(a1->UserBuffer, 4ui64, 1u);  
        else  
            v7 = 0xC0000004;  
    }  
    ...  
    ...  
    return v7;  
}
```

IRP Processing in a Driver

- If length is not correct, driver returns an Error using NTSTATUS

```
case 0x222004u:  
    v5 = sub_1400075F0(IRP, CurrentStackLocation);  
    IRP->IoStatus.Status = v5;  
    IoCompleteRequest(a2, 0);  
  
    return;
```

sub_1400075F0

```
__int64 __fastcall sub_1400075F0(_IRP *a1, _IO_STACK_LOCATION *a2)  
{  
    unsigned int v7;  
  
    a1->IoStatus.Information = 0i64;  
    v7 = 0xC0000004;  
    if ( a2->Parameters.DeviceIoControl1.InputBufferLength == 4 )  
    {  
        if ( a2->Parameters.DeviceIoControl2.OutputBufferLength == 4 )  
            ProbeForWrite(a1->UserBuffer, 4ui64, 1u);  
        else  
            v7 = 0xC0000004;  
    }  
    ...  
    ...  
    return v7;  
}
```

IRP Processing in a Driver

- Return value is stored in **IoStatus** field in IRP

```
case 0x222004u:  
    v5 = sub_1400075F0(IRP, CurrentStackLocation);  
    IRP->IoStatus.Status = v5;  
    IoCompleteRequest(a2, 0);  
  
    return;
```

sub_1400075F0

```
__int64 __fastcall sub_1400075F0(_IRP *a1, _IO_STACK_LOCATION *a2)  
{  
    unsigned int v7;  
  
    a1->IoStatus.Information = 0i64;  
    v7 = 0xC0000004;  
    if ( a2->Parameters.DeviceIoControl1.InputBufferLength == 4 )  
    {  
        if ( a2->Parameters.DeviceIoControl2.OutputBufferLength == 4 )  
            ProbeForWrite(a1->UserBuffer, 4ui64, 1u);  
        else  
            v7 = 0xC0000004;  
    }  
    ...  
    ...  
    return v7;  
}
```

IRP Processing in a Driver

- Calling **IofCompleteRequest**, Driver complete I/O request

```
case 0x222004u:  
    v5 = sub_1400075F0(IRP, CurrentStackLocation);  
    IRP->IoStatus.Status = v5;  
    IofCompleteRequest(a2, 0);  
  
    return;
```

sub_1400075F0

```
__int64 __fastcall sub_1400075F0(_IRP *a1, _IO_STACK_LOCATION *a2)  
{  
    unsigned int v7;  
  
    a1->IoStatus.Information = 0i64;  
    v7 = 0xC0000004;  
    if ( a2->Parameters.DeviceIoControl1.InputBufferLength == 4 )  
    {  
        if ( a2->Parameters.DeviceIoControl2.OutputBufferLength == 4 )  
            ProbeForWrite(a1->UserBuffer, 4ui64, 1u);  
        else  
            v7 = 0xC0000004;  
    }  
    ...  
    ...  
    return v7;  
}
```

NTSTATUS Value

- NTSTATUS represents the status of the Valid/Invalid routine
- It helps us find constraints of each IOCTL

NTSTATUS	DESCRIPTION
0xC0000001	STATUS_UNSUCCESSFUL
0xC0000002	STATUS_NOT_IMPLEMENTED
0xC0000004	STATUS_INFO_LENGTH_MISMATCH
0xC000000D	STATUS_INVALID_PARAMETER
0xC0000023	STATUS_BUFFER_TO_SMALL
⋮	⋮

The Necessity of Automation

- Modern software is too large to manually find vulnerabilities
- Fuzzing is a practical and powerful method of automation
- OSS-Fuzz's fully-automated approach has found **30,000+** bugs!

Previous Work

Static Analysis ----- POPKORN, IOCTLance

False positive, Focuses too much on known bug cases

Fuzzing ----- WTF, Xen-Fuzzer, kAFL

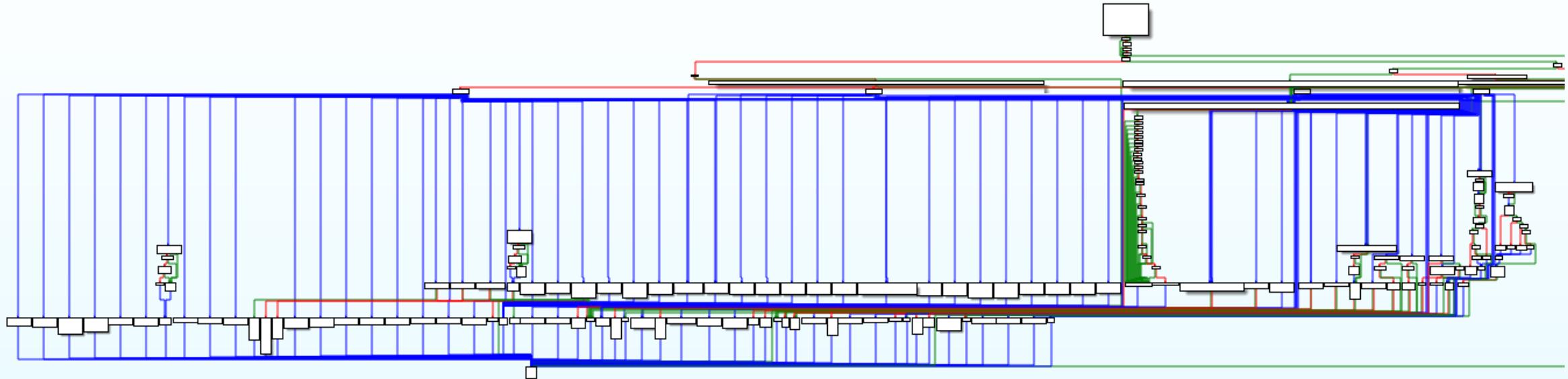
Need too much manual work for making fuzz driver

Static Analysis + Fuzzing ----- Kronl, CAB-Fuzz

Insufficient performance, Requiring environment to analyze

Challenges

- Real-world driver example: **30+** I/O control code..
- A lot of routines, A lot of **constraints** :(



Challenges

- Extremely Challenging Math Problem 

```
switch ( IOCTL )
{
    case 0x22E004u:
        if ( !(unsigned __int8)sub_180005030() )
            goto LABEL_36;
        v5 = sub_180003980(a2);
        break;
    case 0x22E008u:
        if ( !(unsigned __int8)sub_180005030() )
            goto LABEL_36;
        v5 = sub_180003800(a2, CurrentStackLocation);
        break;
    case 0x22E00Cu:
        if ( !(unsigned __int8)sub_180005030() )
            goto LABEL_36;
        v5 = sub_1800039D0(a2, CurrentStackLocation);
        break;
    case 0x22E010u:
        if ( !(unsigned __int8)sub_180005030() )
            goto LABEL_36;
        v5 = sub_1800038B0(a2, CurrentStackLocation);
        break;
    case 0x22E014u:
        if ( !(unsigned __int8)sub_180005030() )
            goto LABEL_36;
        v5 = sub_180003AB0(a2, CurrentStackLocation);
        break;
    case 0x22E040u:
        v5 = sub_180003B90(a2, CurrentStackLocation);
        break;
```

Ideal Code

```
if ( v4 > 0x220438 )
{
    if ( v4 > 0x220450 )
    {
        v20 = v4 - 2229332;
        if ( v20 )
        {
            v21 = v20 - 4;
            if ( v21 )
            {
                v22 = v21 - 44;
                if ( v22 )
                {
                    v23 = v22 - 4;
                    if ( v23 )
                    {
                        if ( v23 != 4 )
                            goto LABEL_50;
                        else if ( v4 > 0x220420 )
                        {
                            v12 = v4 - 2229284;
                            if ( v12 )
                            {
                                v13 = v12 - 4;
                                if ( v13 )
                                {
                                    v14 = v13 - 4;
                                    if ( v14 )
                                    {
                                        v15 = v14 - 4;
                                        if ( v15 )
                                        {
                                            if ( v15 != 4 )
                                                goto LABEL_50;
                                        }
                                    }
                                }
                            }
                        }
                    }
                }
            }
        }
    }
}
```

Realistic Code

Stateful

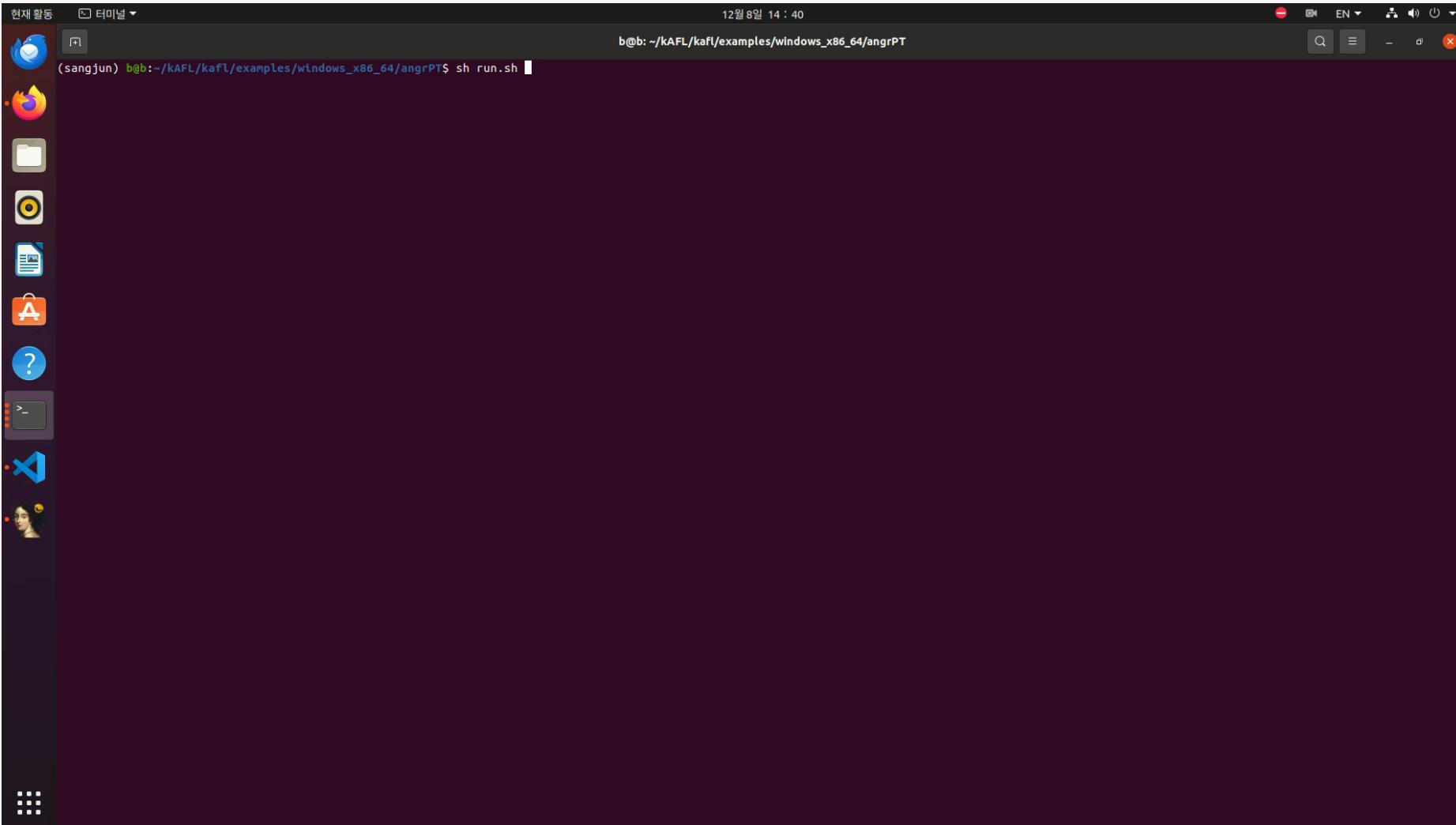
- There are **dependencies** on global/struct variables
- That's why manual work is complicated

```
DeviceObject->Field.called = False;

case 0x20241113:
    if(CurrentStackLocation.InBufferLength == 0x10) {
        DeviceObject->Field.called = True;
    }
    break;

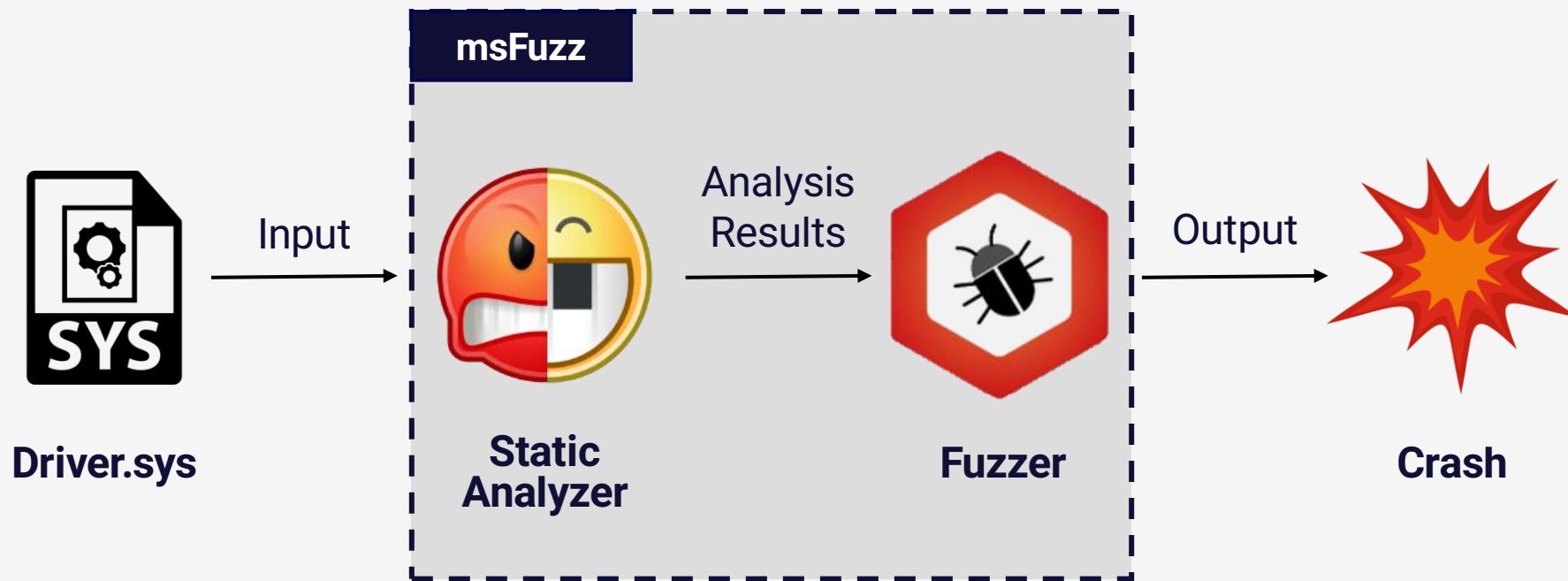
case 0x20241115:
    if(DeviceObject->Field.called == True) {
        /*
            Deep Routine
        */
    }
    break;
```

msFuzz Demo Video



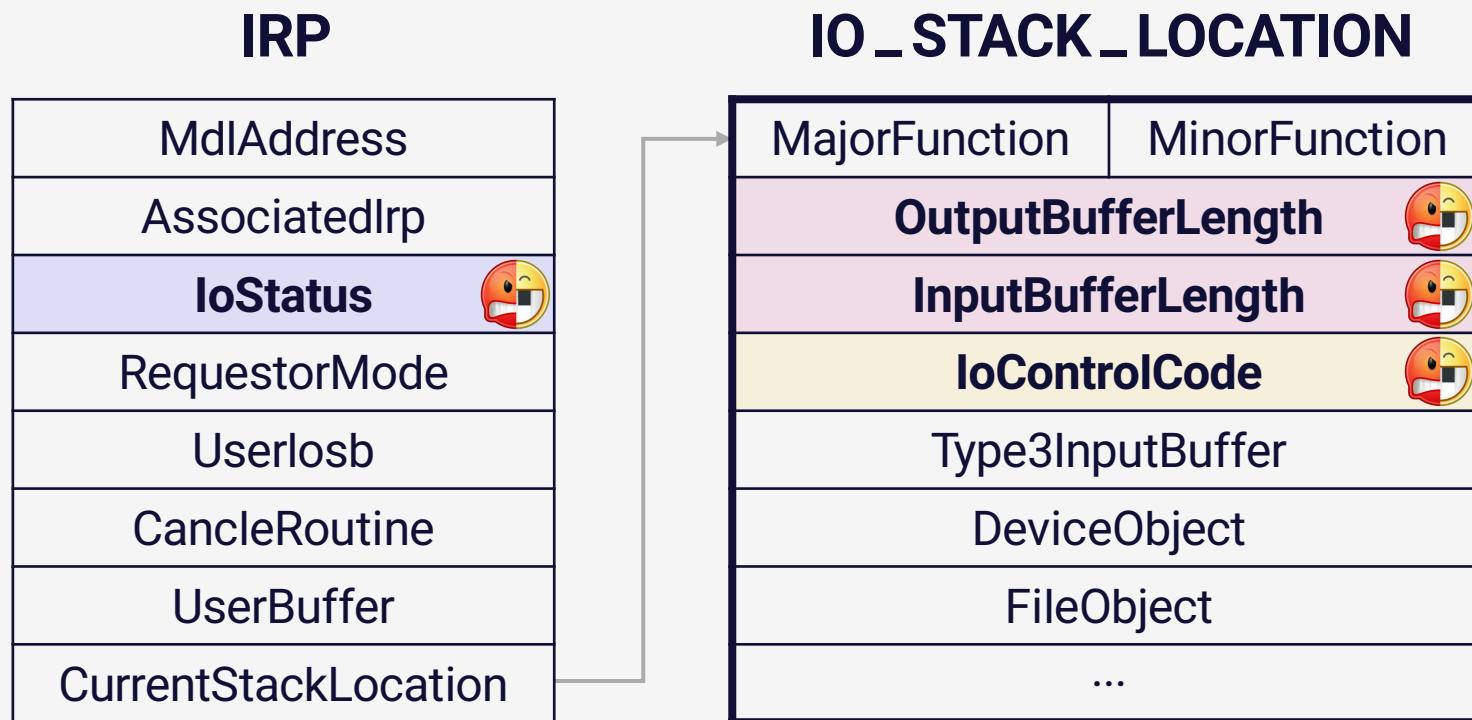
Design of MSFuzz

- It consists of **Static Analysis** and **Fuzzing**



Symbolic Variable

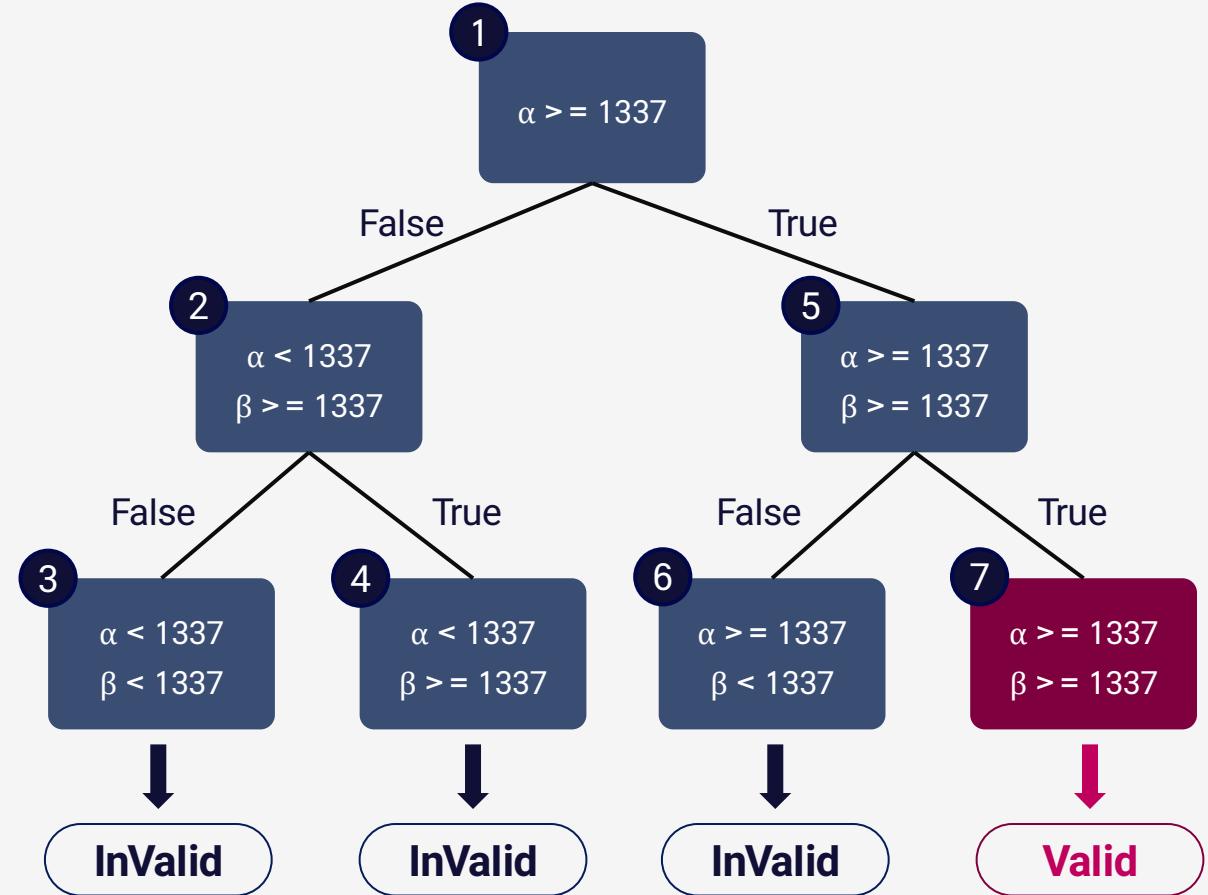
- Trace IoStatus, OutputBufferLength, InputBufferLength and IoControlCode



Symbolic Execution Example

- Analyze Path Constraints

```
case IOCTL_CODE:  
    InputBuffer = IRP->AssociatedIrp.SystemBuffer;  
    OutputBuffer = TSP->AssociatedIrp.SystemBuffer;  
     $\alpha$   $\beta$   
    if [InputBufferLength >= 1337] & [OutputBufferLength >= 1337] {  
        IOCTL_Handler(InputBuffer, OutputBuffer);  
        IRP->IoStatus.Status = STATUS_SUCCESS;  
        ...  
    } else {  $\gamma$   
        IRP->IoStatus.Status = STATUS_INVALID_PARAMETER;  
    }  
  
    break;
```



Dependency

- Drivers are characterized by being stateful like network protocols
- Global variable is the "**Key State Variable**"

```
global = 0;

switch (IOCTL_CODE)
{
    case A:
        global = 1;
        break;
    case B:
        if (global == 1) {
            global = 2;
        }
        break;
    case C:
        if (global == 2) {
            BUG(); ⚡
        }
        break;
}
```

The diagram illustrates the flow of dependencies between three code blocks based on the value of the global variable:

- Case A:** Sets global to 1 and exits.
- Case B:** Checks if global is 1. If true, it sets global to 2 and exits.
- Case C:** Checks if global is 2. If true, it calls the `BUG()` function and exits.

Dashed arrows indicate dependencies:

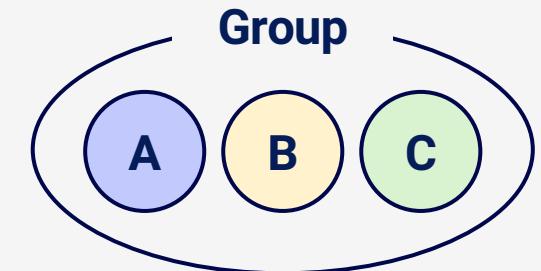
- A dashed arrow points from the initial assignment of `global = 0;` to the assignment in **Case A**.
- A dashed arrow points from the assignment in **Case A** to the condition in **Case B**.
- A dashed arrow points from the condition in **Case B** to the assignment in **Case C**.

Analyze Dependency with Global Variables

- Static analyze operators with global variables as operands

```
global = 0;

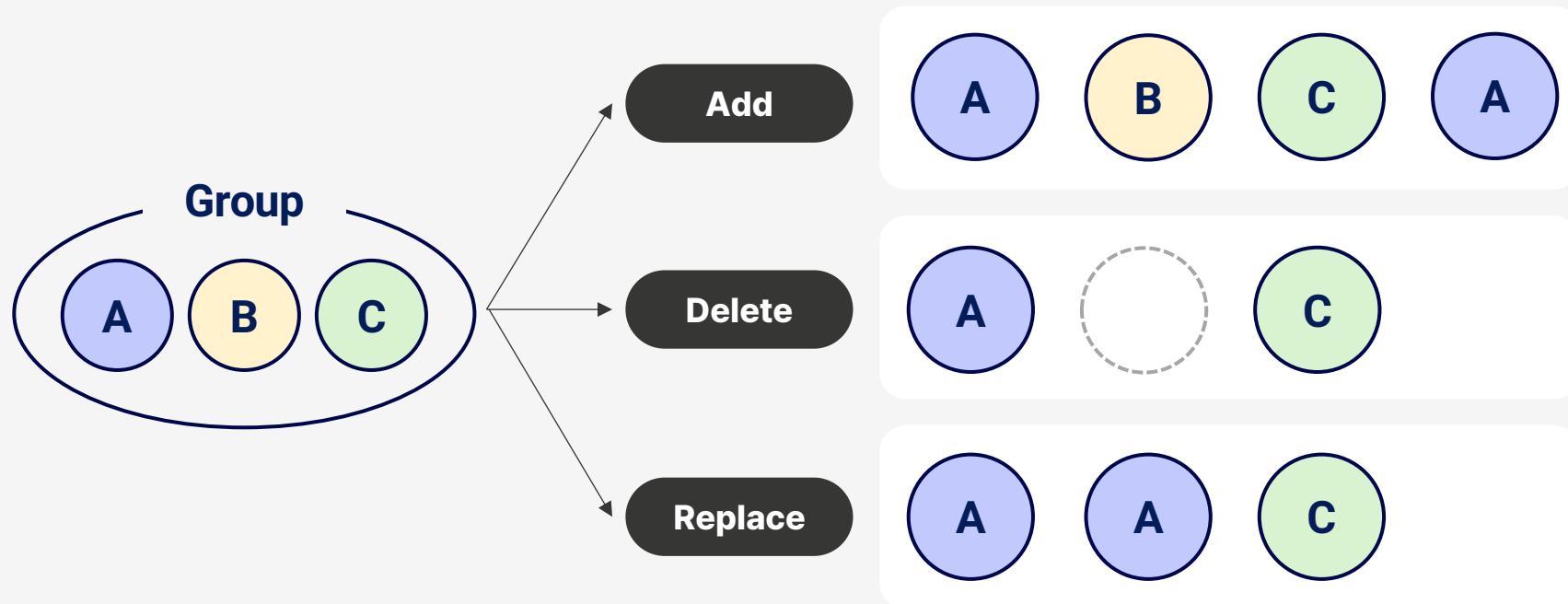
switch (IOCTL_CODE)
{
    case A
        Write
        break;
    case B
        if Read
            Write
        }
        break;
    case c
        if Read
            BUG(); ⚡
        }
        break;
}
```



- 1) Using same global variable
- 2) Having global variable

Consider dependency

- IRP Sequence level mutation



Real world case 1)

- This case shows expand code coverage with considering dependency
- IOCTL B is explored only called after IOCTL A

```
DWORD signature = 0;

case A:
    UserInput = IRP->AssociatedIrp.MasterIrp;
    if(UserInput == 0x4D53492E) {
        signature = 0x4D53492E;
    }
    break;
```

IOCTL A Routine

```
case B:          Bother fuzzing
if(signature != 0x4D53492E)
    goto FAIL;

UserInput = IRP->AssociatedIrp.MasterIrp;
v16 = (DWORD)UserInput[0];

switch (AnotherIOCTL) {
    case X:
        ...
        ...
    case Y:
        ...
        ...
}
```

IOCTL B Routine

Real world case 2)

- This case shows find vulnerability with considering dependency

```
UserInput = IRP->AssociatedIrp.MasterIrp;
if(UserInput) {
    global = &word_711E0;

    do {
        LoopCount = UserInput[0];
        *global = UserInput[0]; // Arbitrary Write
        UserInput = (char *)UserInput + 2; * // 00B Write From 0x711E0;
        global++;
    } while(LoopCount) // Can Loop Control

    ...
}

}
```

IOCTL A Routine

```
void sub_178A8(__int64 a1) {
    __int64 v2;

    if (qword_71650) { // Overwrite is possible!
        v2 = *(__int64 *)(a1 + 8);
        if (v2)
            qword_71650(v2); * RIP Control
    }
}
```

IOCTL B Routine

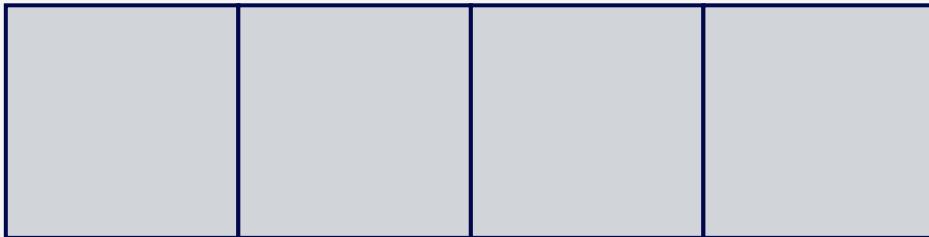
Real world case 2)

- IOCTL B read function pointer
- But when IOCTL B called after IOCTL A?

Not consider dependency

Global Variable

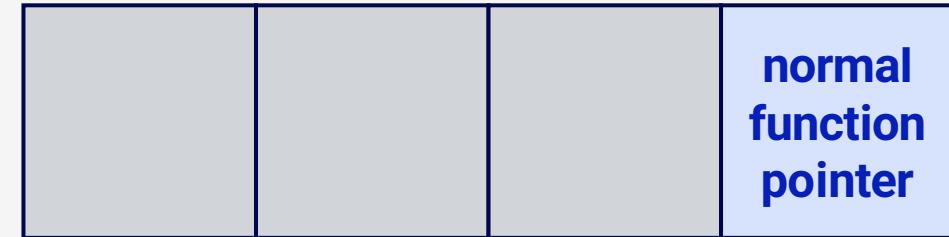
0x711E0



...

Function

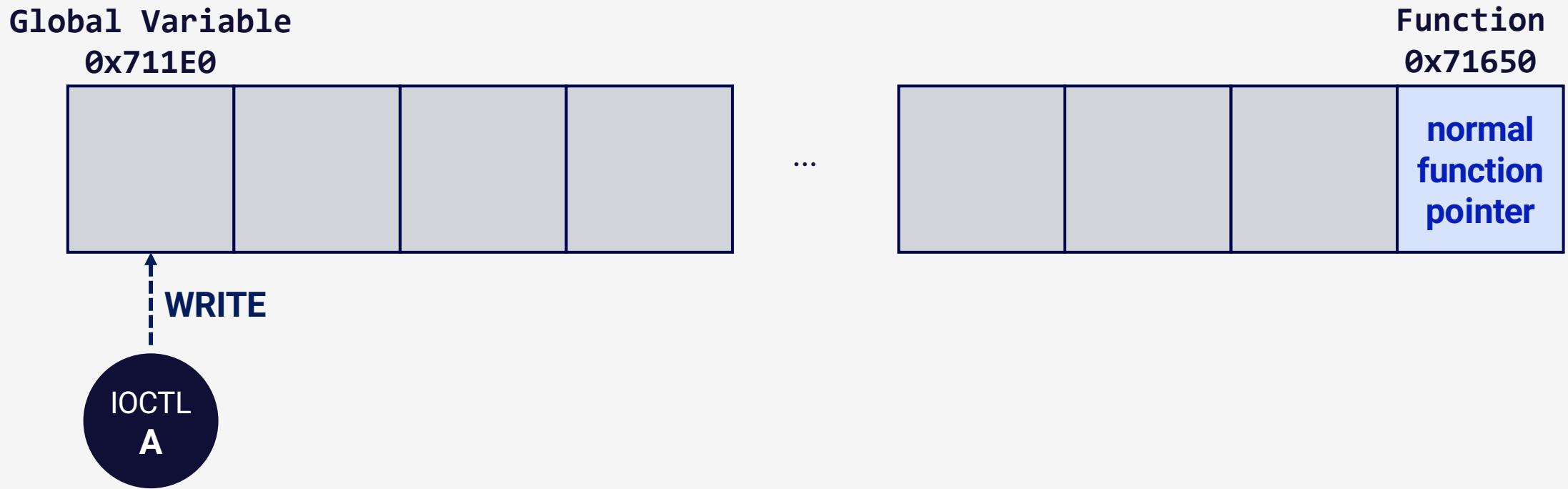
0x71650



Real world case 2)

- IOCTL B read function pointer
- But when IOCTL B called after IOCTL A?

Not consider dependency



Real world case 2)

- IOCTL B read function pointer
- But when IOCTL B called after IOCTL A?

Not consider dependency

Global Variable

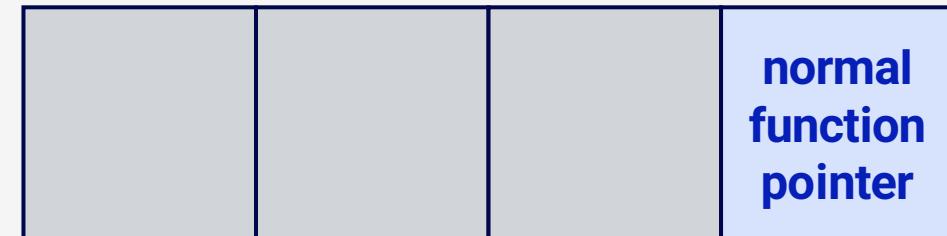
0x711E0



...

Function
0x71650

normal
function
pointer



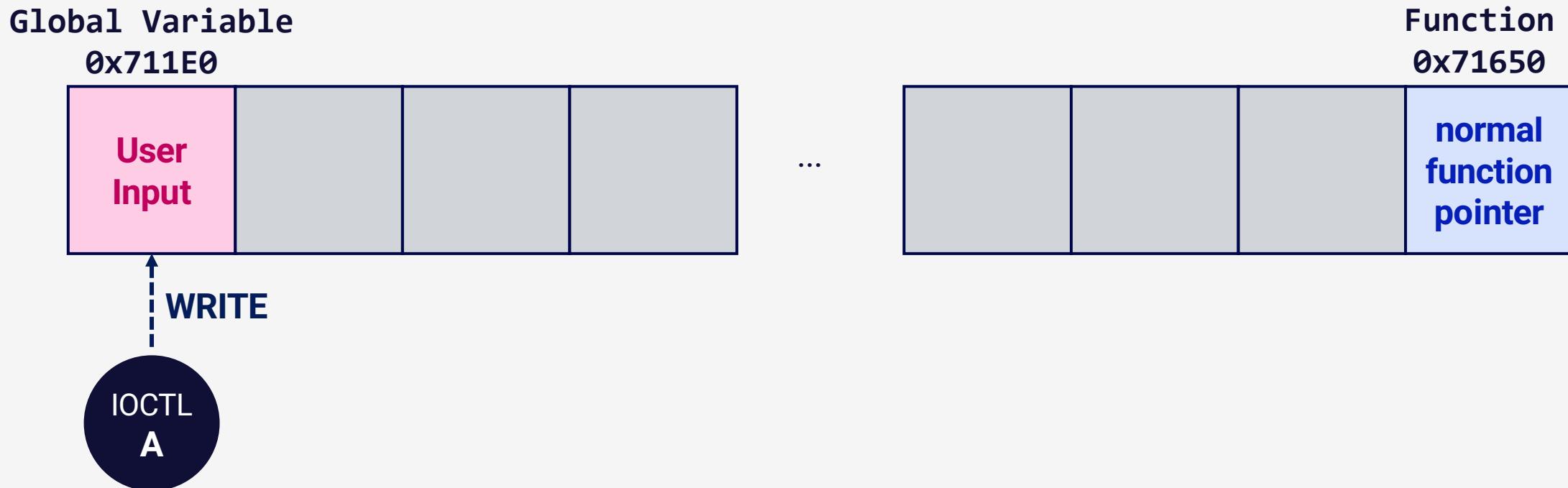
READ

IOCTL
B

Real world case 2)

- Through IOCTL A, user can write data from **0x711E0**. (Out of Bounds Write)
- It means, **0x71650** at IOCTL B also can manipulate so that leads RIP Control

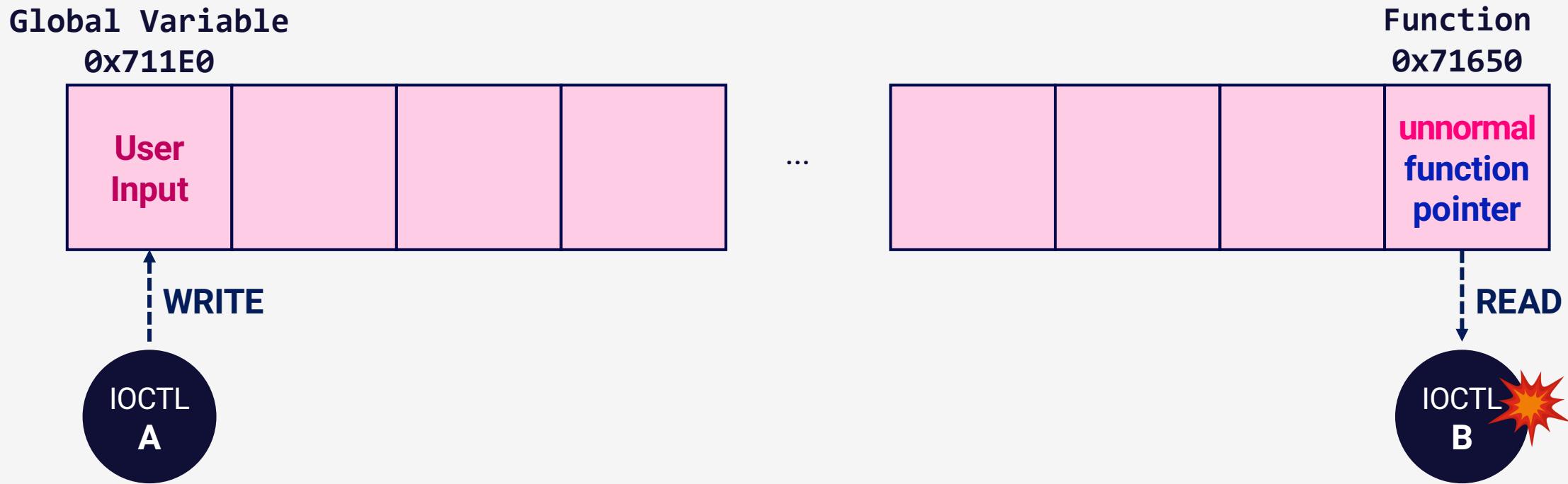
Consider dependency



Real world case 2)

- Through IOCTL A, user can write data from **0x711E0**. (Out of Bounds Write)
- It means, **0x71650** at IOCTL B also can manipulate so that leads RIP Control

Consider dependency



Difficult of Handling duplicate crash

- AFL Fuzzer: "Same root cause? It's not a Unique Crash :D"

```
process timing -----+-----+ overall results -----+
    run time : 7 days, 7 hrs, 17 min, 54 sec | cycles done : 3
    last new path : 0 days, 20 hrs, 5 min, 58 sec | total paths : 409
    last uniq crash : 1 days, 3 hrs, 9 min, 56 sec | uniq crashes : 95
    last uniq hang : 1 days, 3 hrs, 9 min, 26 sec | uniq hangs : 8
cycle progress -----+-----+ map coverage -----+
    now processing : 95* (23.23%) | map density : 2.80% / 7.29%
    paths timed out : 0 (0.00%) | count coverage : 2.37 bits/tuple
stage progress -----+-----+ findings in depth -----+
    now trying : interest 16\8 | favored paths : 34 (8.31%)
    stage execs : 1080/12.1k (8.90%) | new edges on : 60 (14.67%)
    total execs : 9.23M | total crashes : 351k (95 unique)
    exec speed : 1.58/sec (zzzz...) | total tmouts : 44 (8 unique)
fuzzing strategy yields -----+-----+ path geometry -----+
    bit flips : 129/355k, 28/355k, 21/354k | levels : 14
    byte flips : 11/44.4k, 9/44.3k, 8/44.0k | pending : 252
    arithmetics : 90/2.48M, 39/795k, 32/721k | pend fav : 0
    known ints : 6/177k, 30/1.13M, 19/1.40M | own finds : 404
    dictionary : 0/0, 0/0, 15/1.11M | imported : 0
    havoc : 62/186k, 0/0 | stability : 1.51%
    trim : 0.83%/20.3k, 0.00% -----+
-----+-----+ [cpu: 0%]+
processes nudgedss with PID 5068 has been terminated.. [cpu: 0%]
```

Difficult of Handling duplicate crash

- Exhausting time to validate

The screenshot shows a debugger's command window with the following content:

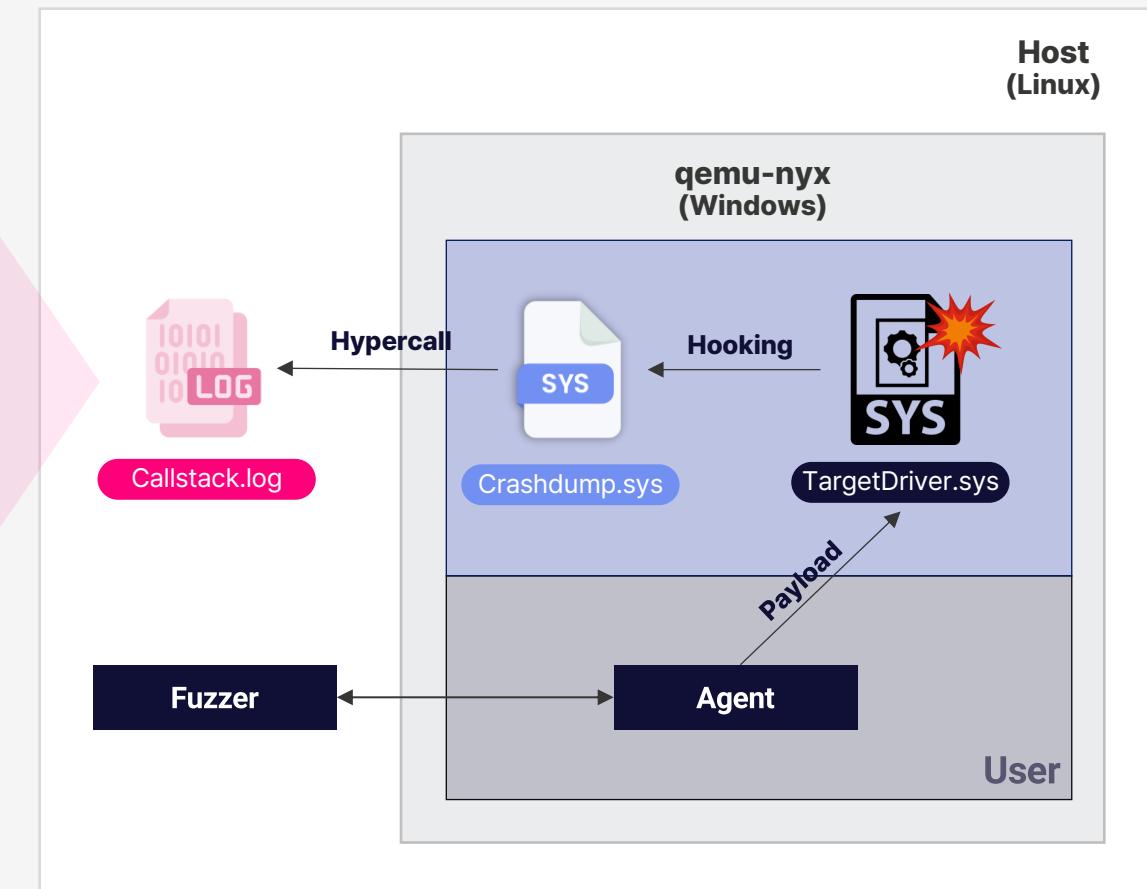
```
0:052> ~52s
ntdll!NtWaitForSingleObject+0x14:
00007ffc`15cdc144 c3          ret
0:052> kb 20
# RetAddr      : Args to Child
00 00007ffc`12eb8ba3 : 00000000`00005fd8 00007ffc`0093ec04 00000000`00000000 00000000`00300000 : ntdll!NtWaitForSingleObject+0x14
01 00007ffc`0093ece8 : 00000000`00000000 00000000`00000027 000002bd`00000000 00000000`0000051c : KERNELBASE!WaitForSingleObjectEx+0x93
02 00007ffc`0093ec9e : 000000f8`d307f401 00000000`00000000 00000000`00000002 00000000`00000000 : clr!CLREventWaitHelper2+0x3c
03 00007ffc`0093ec04 : 000002bd`5d4107d0 00007ffc`00954649 00000000`00000002 000002bd`5d4107d0 : clr!CLREventWaitHelper+0x1f
04 00007ffc`009f8eb8 : 000002c1`ef90b230 00007ffc`009fc2d0 000000f8`d307f4c0 00007ffc`009fc2d0 : clr!CLREventBase::WaitEx+0x71
05 00007ffc`00c23bf1 : 000000f8`d307f401 000002c1`ef90b230 00007ffc`009fc2d0 000000f8`00000029 : clr!SVR::gc_heap::bgc_thread_function+0x58
06 00007ffc`008fb665 : 000000f8`d307f4c0 000000f8`d487f8b0 00000000`00000000 00007ffc`00c23b00 : clr!<lambda_a850bf1174f694b6476abb4be0d292e7>::operator()
07 00007ffc`14567bd4 : 00000000`00000000 00000000`00000000 00000000`00000000 00000000`00000000 : clr!Thread::intermediateThreadProc+0x8b
08 00007ffc`15caced1 : 00000000`00000000 00000000`00000000 00000000`00000000 00000000`00000000 : KERNEL32!BaseThreadInitThunk+0x14
09 00000000`00000000 : 00000000`00000000 00000000`00000000 00000000`00000000 00000000`00000000 : ntdll!RtlUserThreadStart+0x21
```

The bottom status bar shows "0:052> kb 20".

Call Stack Parser

- Using Hypervisors to get the guest VM's callstack from the host OS

Register	Callstack
<code>fffff9686`3ab9e4f0 ffffff805`5837dec0 ffffff805`73270000 00000000`00000000 fffff805`00000003 fffff9686`3ab9e4f0 ffffff805`58414190 fffff9686`3ab9ea40 00000000`00000000 00000000`00000000 ffffff804`9f26f2d8 ffffff804`9f26f2d8 00000000`00000050 ffffff804`9f26f2d8 00000000`0000002 fffff9686`3ab9ede0 fffff8684`ab979a20 00000000`0000002 fffff9686`3ab9ee68 00000000`00000000 00000000`00000001 fffff8684`b263da20 ffffffff`ffffffff ffffff805`58254d26 fffff805`732785f3 fffff8684`cafebabe 00000000`deadbeef fffffe306`00000002 00000000`00000001 fffff8684`b46e8e50 fffff8684`b2c4bd90 fffff8684`ae650808 fffff8684`b46e8e50 fffff9686`3ab9fb40 00000000`00010000 fffff8684`b46e8e50 00000000`00000001 00000000`00000000 00000000`00000000 00000000`00000000 00000000`00000000 00000000`00000000 00000000`00000000 00000000`00000000 000007fff`b8e8591b 00000002`c00000bb 00000000`00000000 00008b50`a144092d 00000002`c00000bb 00000000`00000000 00008b50`a144092d 00007fff`b8e9b4ee 00000000`0022209c 00000000`00000000 00000000`00000000 00000000`00000000 00000000`00000000 0000013c`00000000 00000010`9f9efa80 00000000`00000000 0000013c`00000000 00000010`9f9efa80 00000000`00000000 0000013c`63df0000</code>	<code>nt!DbgBreakPointWithStatus nt!KiBugCheckDebugBreak+0x12 nt!KeBugCheck2+0x946 nt!KeBugCheckEx+0x107 nt!MiSystemFault+0x1dda4f nt!MmAccessFault+0x400 nt!KiPageFault+0x358 nt!IofCallDriver+0x55 nt!IoPynchronousServiceTail+0x34c nt!IoPxxxControlFile+0xc71 nt!NtDeviceIoControlFile+0x56 nt!KiSystemServiceCopyEnd+0x25 ntdll!NtDeviceIoControlFile+0x14 KERNELBASE!DeviceIoControl+0x6b KERNEL32!DeviceIoControlImplementation+0x81 Project2+0x11ac6</code>



As a result

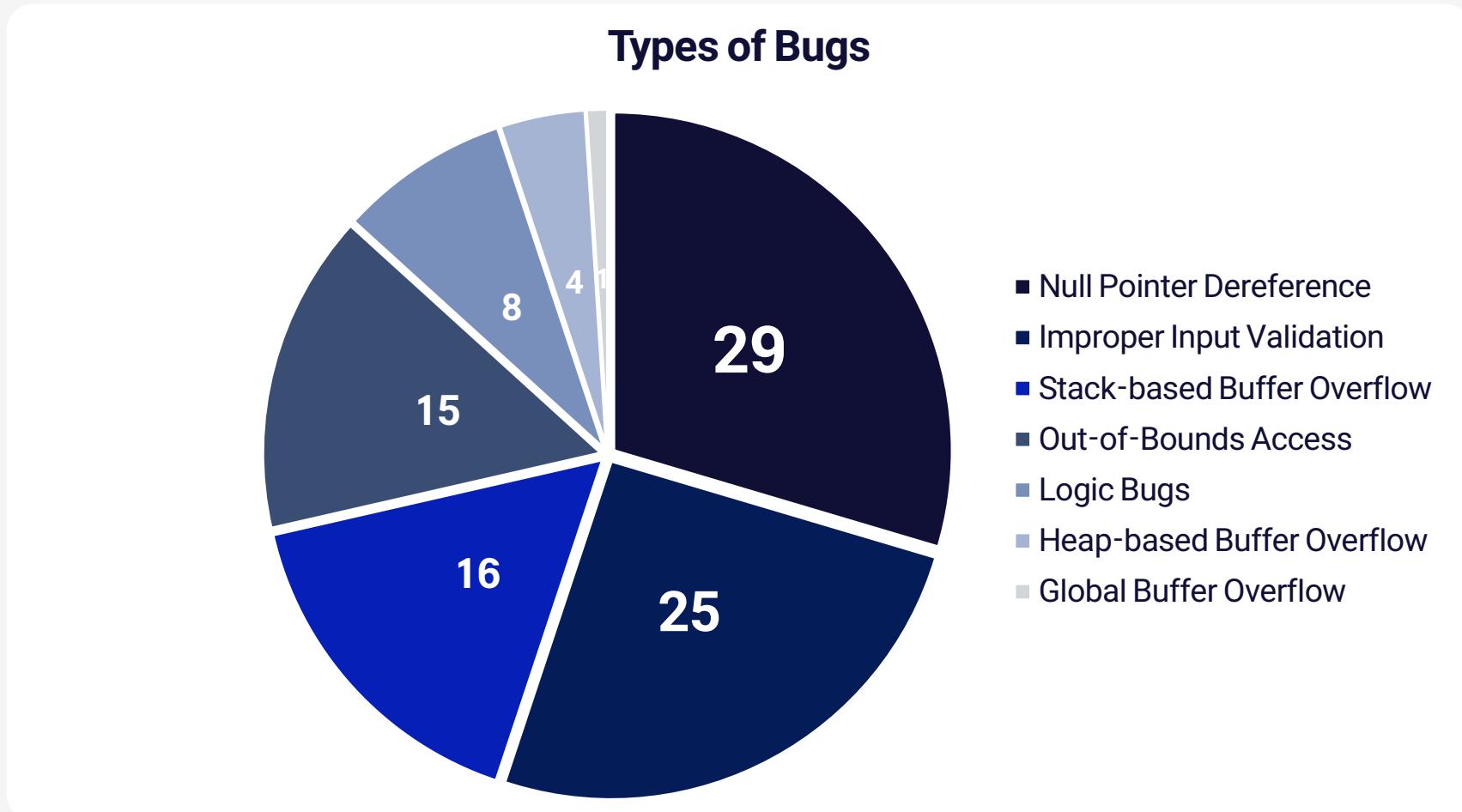
- Leveraging high **code coverage** with constraint solver
- Consider **dependency** with global variable
- Fully-automated from finding **bugs** to **analyze root cause**
- Allows anyone to easily validate windows kernel drivers

Implementation

- **Static Analyzer**
 - Used **Angr** for Path Constraints and dependency
- **Fuzzer**
 - Used **Nyx Fuzzer** to keep clean VM State and intel PT
 - Extended **Nyx-Qemu** for the Call Stack Log

Results

- We reported **100+ reports** and got 21 CVEs



Results

- We reported 100+ reports and got 21 CVEs

Vendor	CVE	Vendor	CVE
Microsoft (1)	CVE-2024-21442		CVE-2023-51776 CVE-2023-51777 CVE-2023-51778 CVE-2024-22102 CVE-2024-22103 CVE-2024-22104 CVE-2024-22105 CVE-2024-22106 CVE-2024-25086 CVE-2024-25087 CVE-2024-25088 CVE-2024-26314
Jungo (4)	CVE-2023-47569 CVE-2023-47570 CVE-2023-47571 CVE-2023-47572	Mitsubishi (12)	
AMD (1)	CVE-2023-31341		
Siemens (1)	CVE-2023-46280		
Wibukey (2)	CVE-2024-45181 CVE-2024-45182		

Limitation & Future Work

- Improving the precision of static analysis
- Emulating real devices expands the attack surface
- Absolutely need to fuzz drivers that only load in specific situations
- To find more Stateful bugs, it need the incremental snapshot mechanism used by Agamotto

Conclusion

- Developed an automated framework for **fuzzing WDM drivers**, leading to the discovery of **100+** vulnerabilities in 100 days
- Our fuzzer is published as open source on Github
 - github.com/0dayResearchLab/msFuzz

In Collaborations with

- **Byunghyun Kang**
 - B.S. Student at Sejong University
- **Seungchan Kim**
 - Student at Korea Digital Media High School
- **Yul Kwon**
 - Student at Sunrin Internet High School
- **Jinho Jung, Joosaeng Kim, Yohan Won**
 - Advisors



Thank you

1-Click-Fuzz: Systematically Fuzzing the Windows Kernel Driver with Symbolic Execution



github.com/0dayResearchLab/msFuzz