CODE BLUE 2024 @TOKYO

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

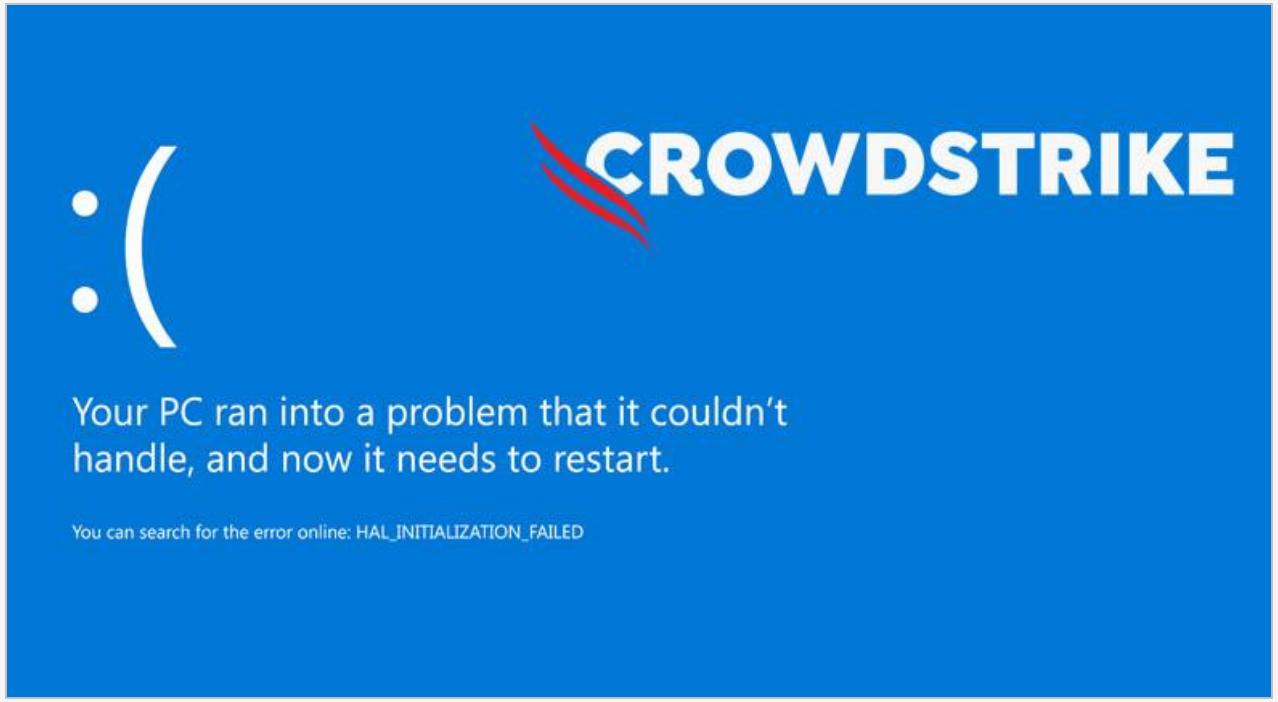
- **Jongseong Kim** ジョンソン・キム
  - ENKI WhiteHat, Security Researcher
  - B.S. Student at Ajou University
  - Linkedin @jongseongkim

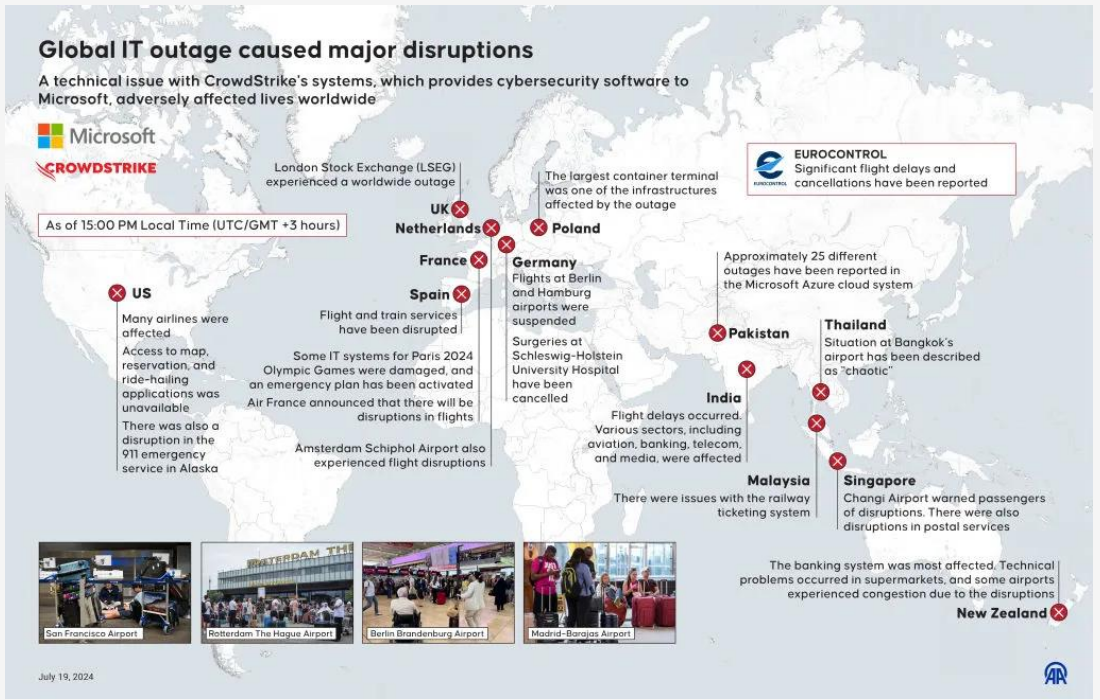
Agenda

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?





[1] UC today, Much of the Digital World Goes Down, with One Faulty Cybersecurity Update Seemingly to Blame (24.07.19)
[2] Getty Anadolu, Global IT outage caused major disruptions

Exploited Windows Kernel Driver

- Hackers also know about it and abuse this!

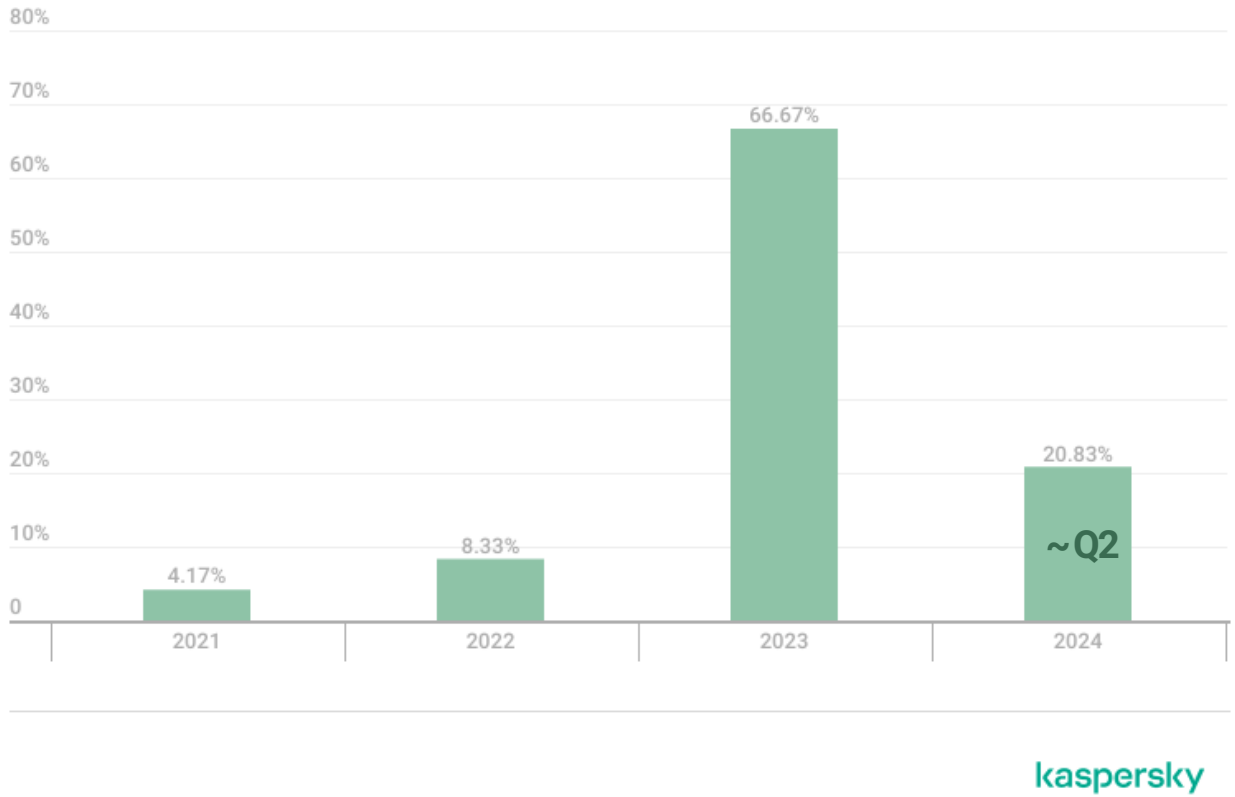
Windows driver zero-day exploited by Lazarus hackers to install rootkit

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.

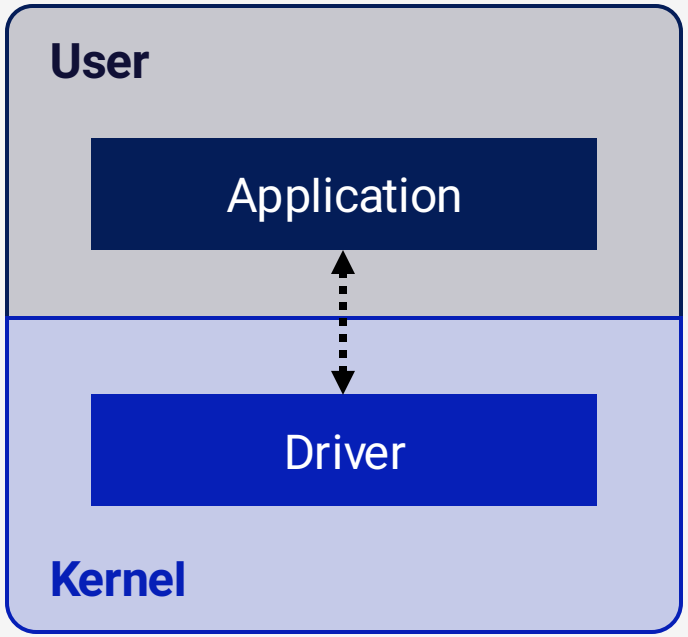
Microsoft fixed the flaw, tracked as CVE-2024-38193 during its August 2024 Patch Tuesday, along with seven other zero-day vulnerabilities.



[1] BleepingComputer, Windows driver zero-day exploited by Lazarus hackers to install rootkit (24.08.19)
[2] Kaspersky, Kaspersky detects 23% increase in attacks targeting vulnerable Windows driver (24.08.21)

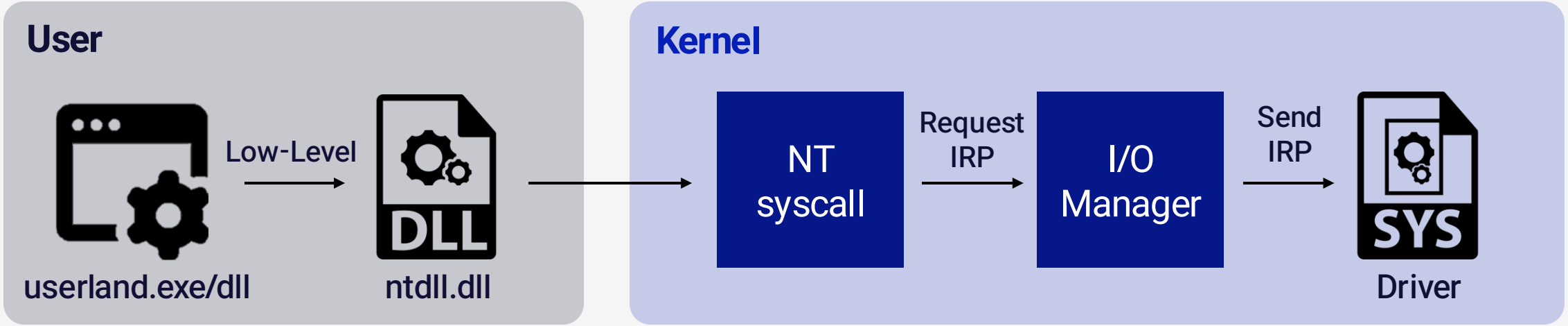
Windows Kernel Driver

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



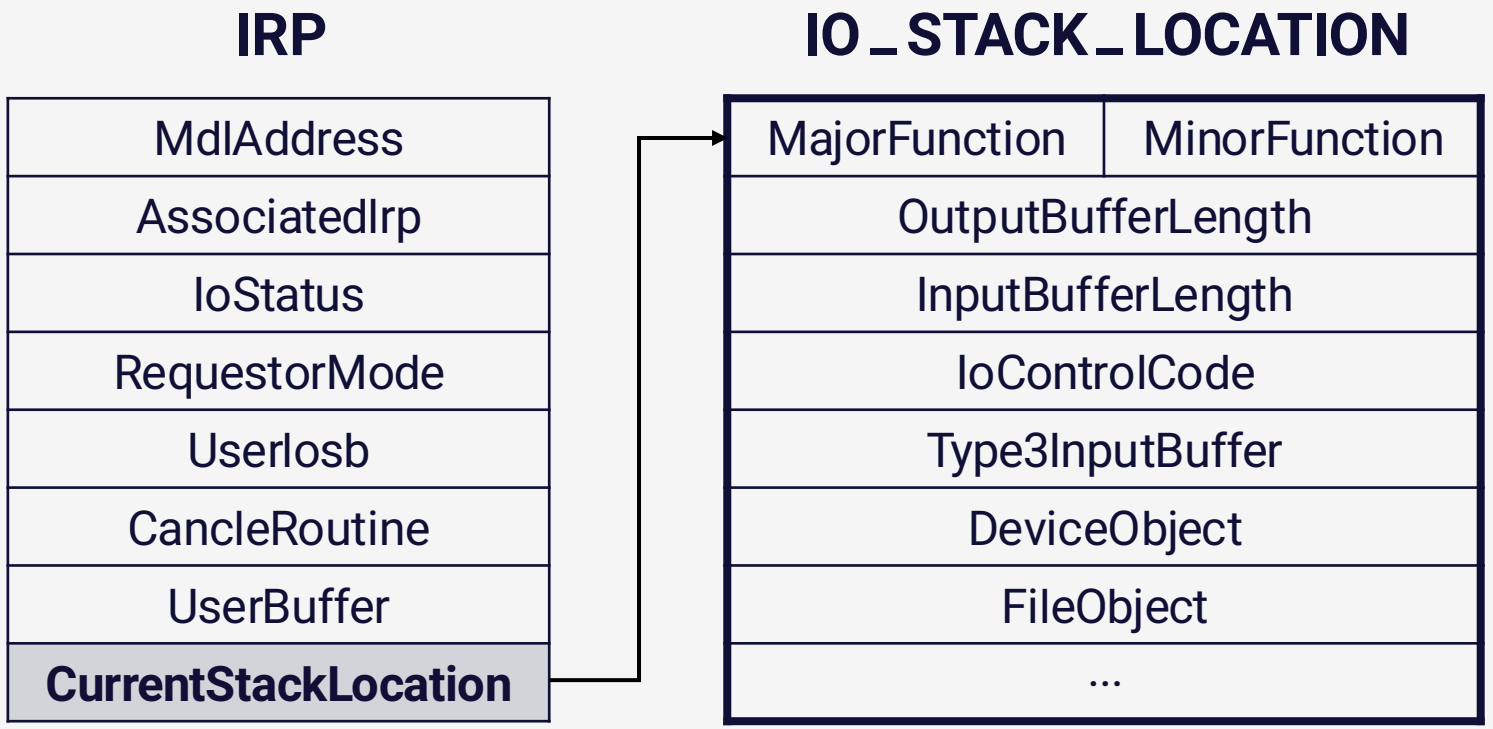
How to interact with driver?

- 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



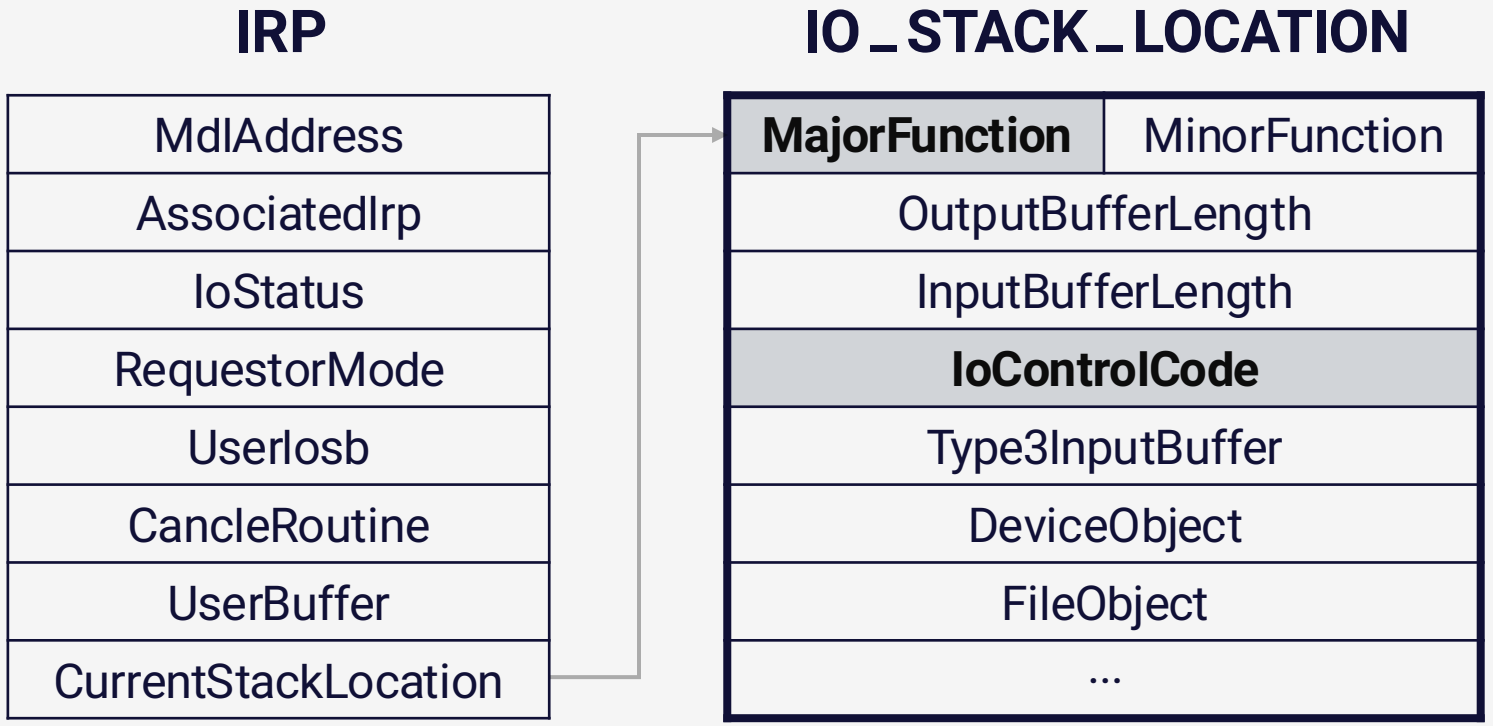
What is IRP?

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



IRP Major Fuction

- **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

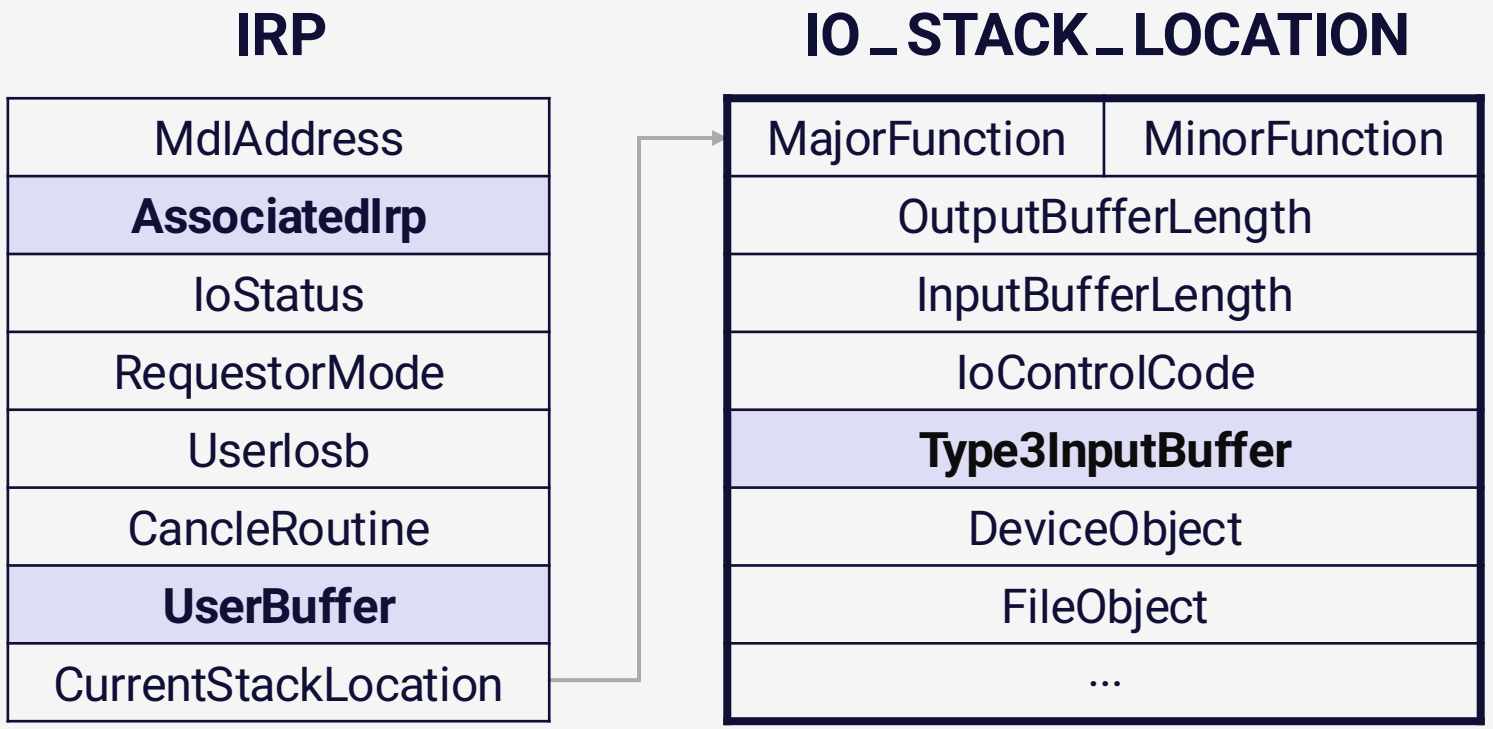
- The user needs to fill in the function arguments to send a request
  ex) IoControlCode, InBuffer, InBufferSize

```
BOOL DeviceIoControl(
  [in]                HANDLE       hDevice,
  [in]                DWORD        dwIoControlCode,
  [in, optional]      LPVOID       lpInBuffer,
  [in]                DWORD        nInBufferSize,
  [out, optional]     LPVOID       lpOutBuffer,
  [in]                DWORD        nOutBufferSize,
  [out, optional]     LPDWORD      lpBytesReturned,
  [in, out, optional] LPOVERLAPPED lpOverlapped
);
```


[1] https://learn.microsoft.com/en-us/windows/win32/api/ioapiset/nf-ioapiset-deviceiocontrol

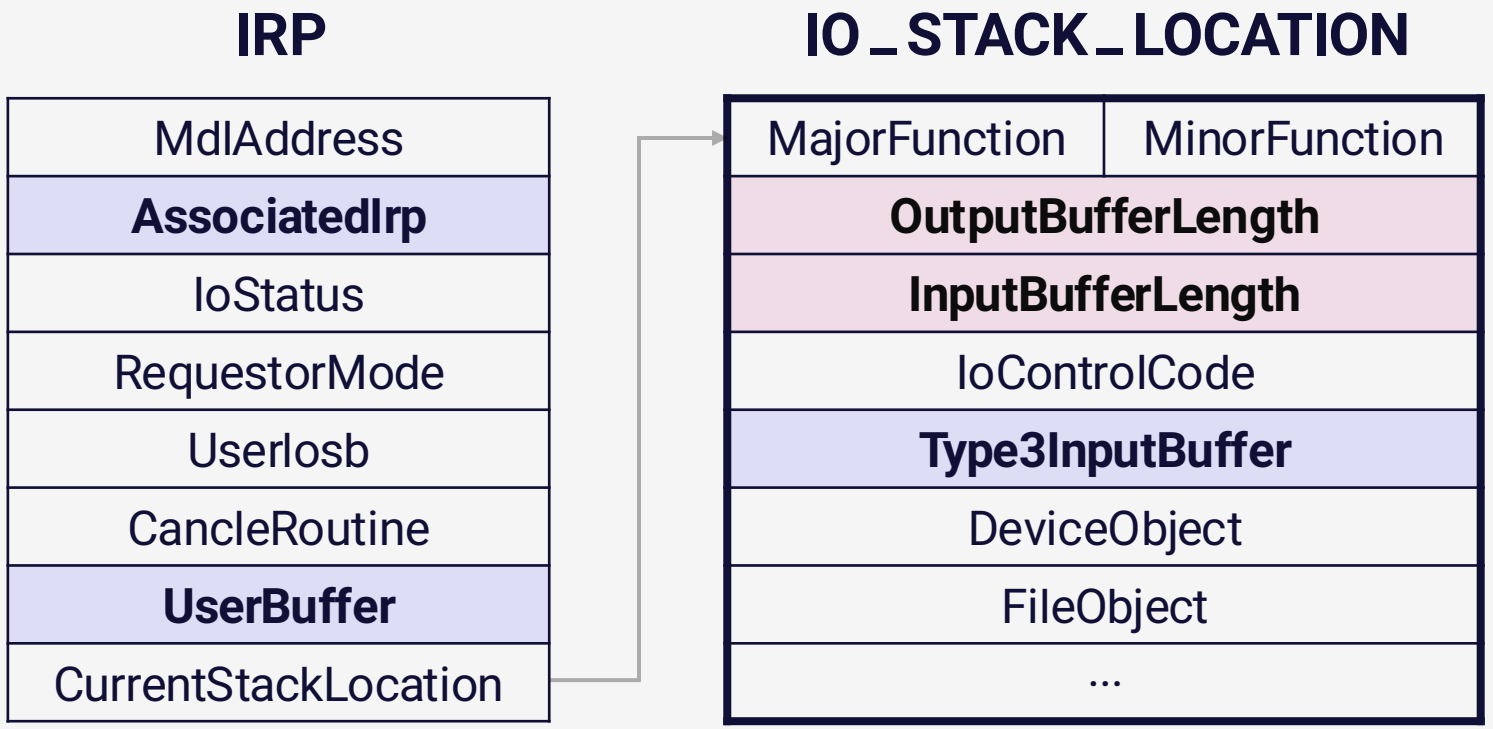
InputBuffer/OutputBuffer

- The **passed data** stored in **IRP** when requesting DeviceIoControl commands



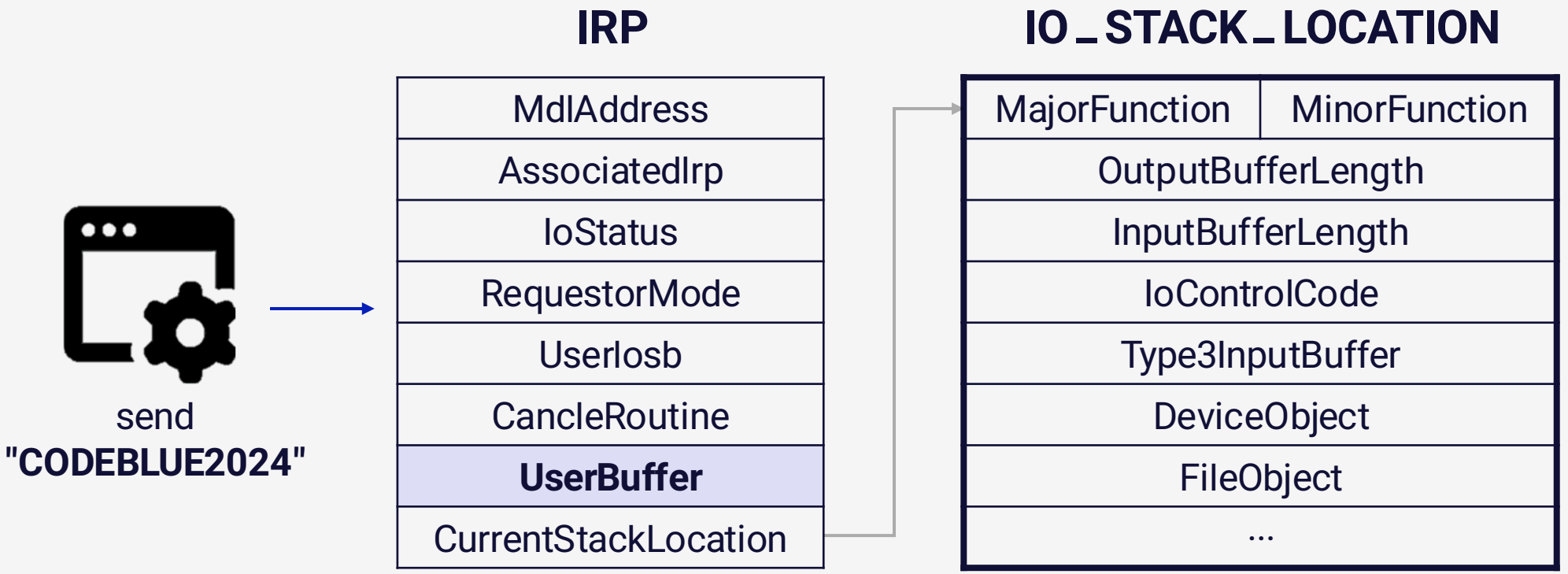
Buffer Length

- IRP contains BufferLength field



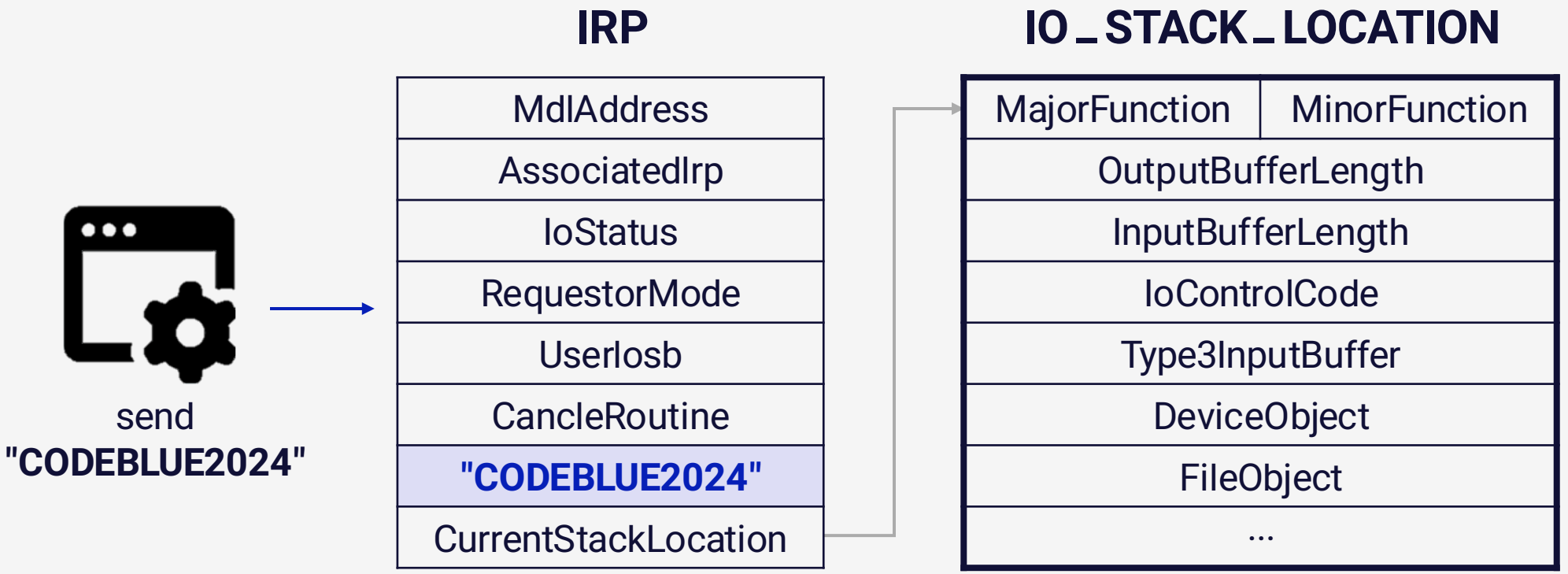
I/O Request Example

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



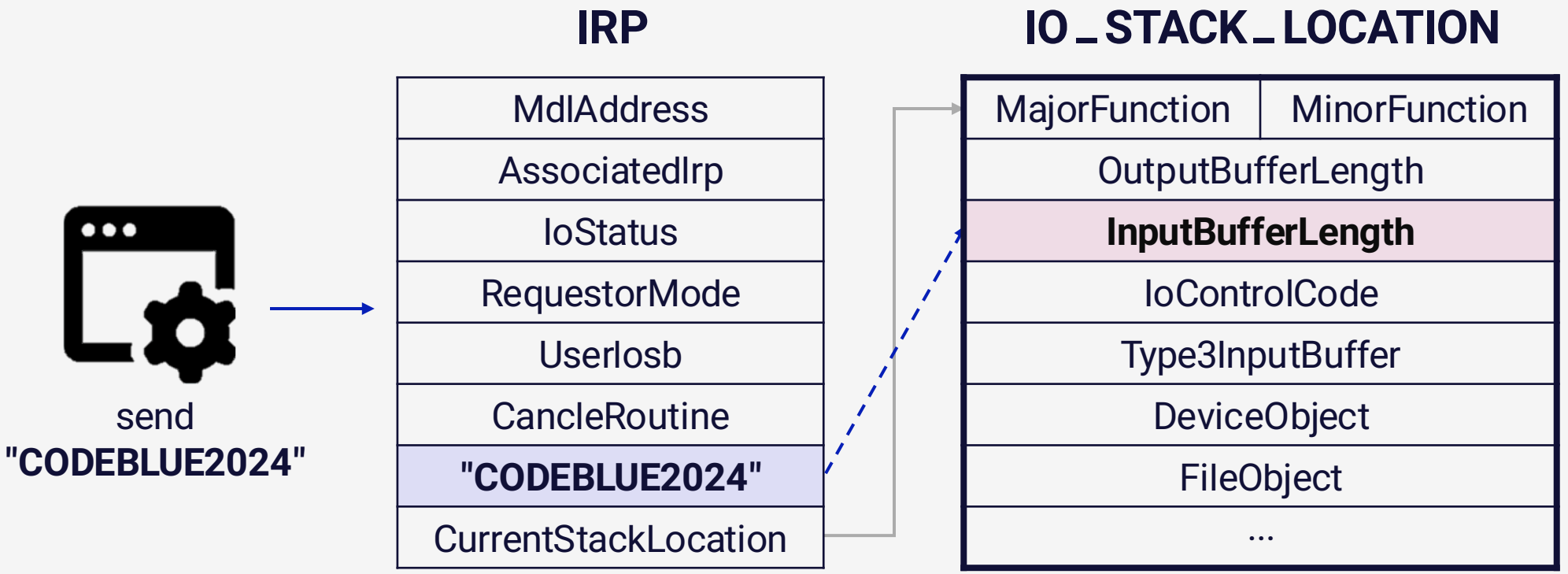
I/O Request Example

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



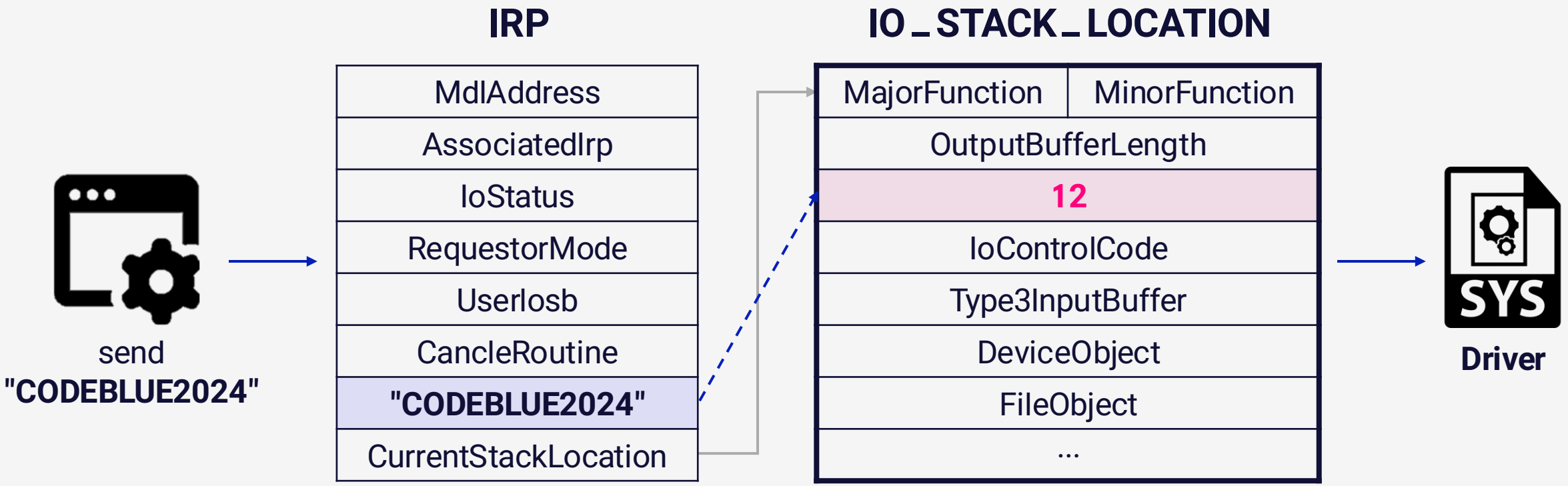
I/O Request Example

- 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;
    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

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

```
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

- 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

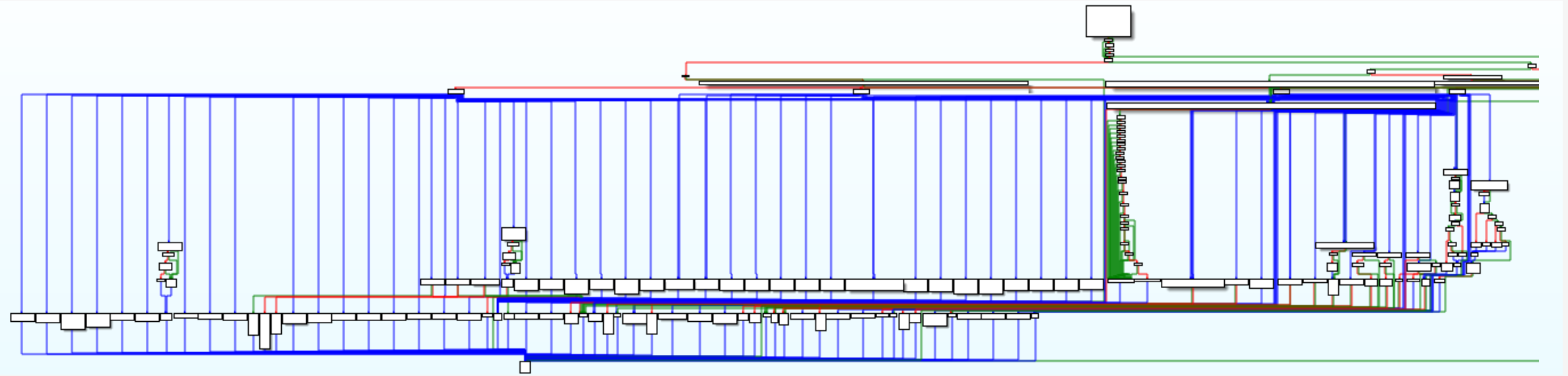
Need too much manual work for making fuzz driver

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 🤣

Ideal Code

```
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;
```

Realistic Code

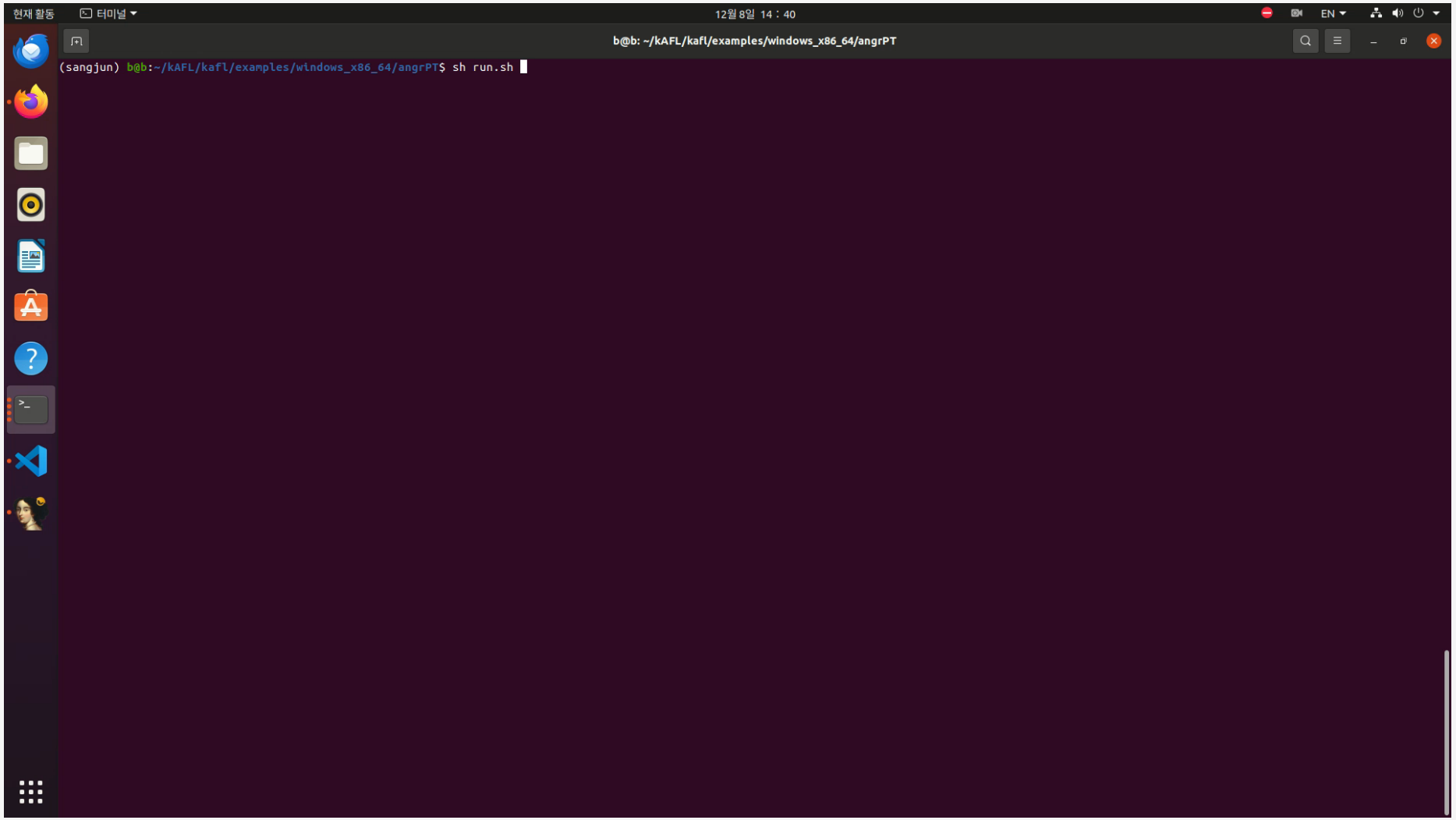
```
if ( v4 > 0x220438 )
{
  if ( v4 > 0x220450 )
  {
    v20 = v4 - 2229332;
    if ( v20 )
    {
      v21 = v20 - 4;
      if ( v21 )
      {
        v22 = v21 - 44;
        if ( v22 )
        {
          v23 = v22 - 4;
          if ( v23 )
          {
            if ( v23 != 4 )
              goto 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;
```

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;
```

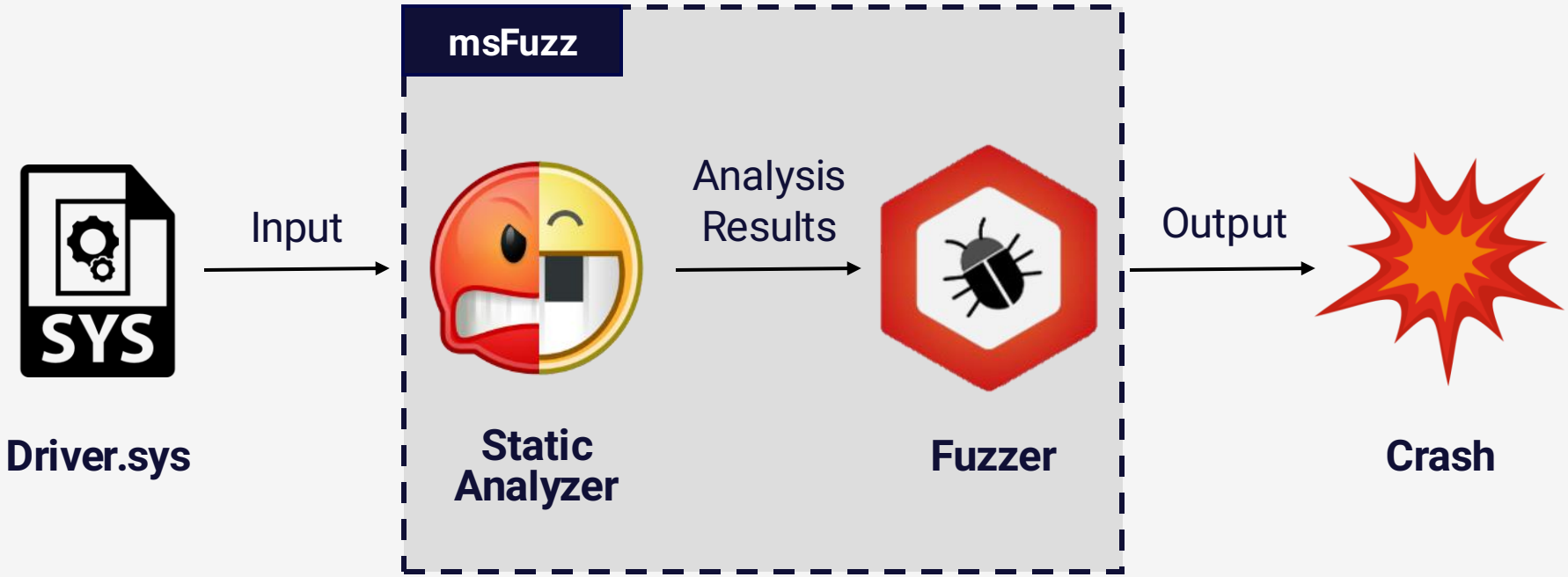
msFuzz Demo Video



Demo Video: https://youtu.be/EvlVc0bY1t4

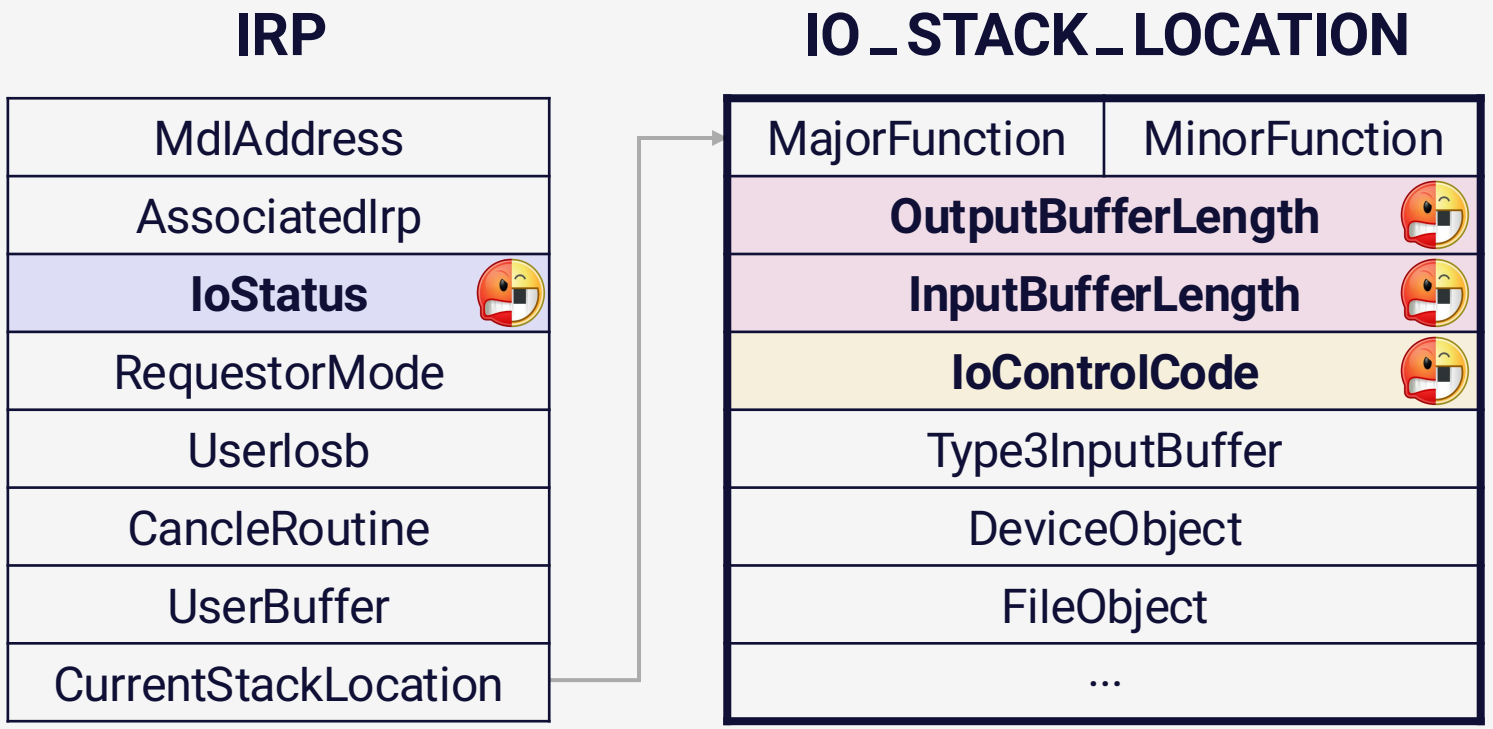
Design of MSFuzz

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



Symbolic Variable

- Trace IoStatus, OutputBufferLength, InputBufferLength and IoControlCode

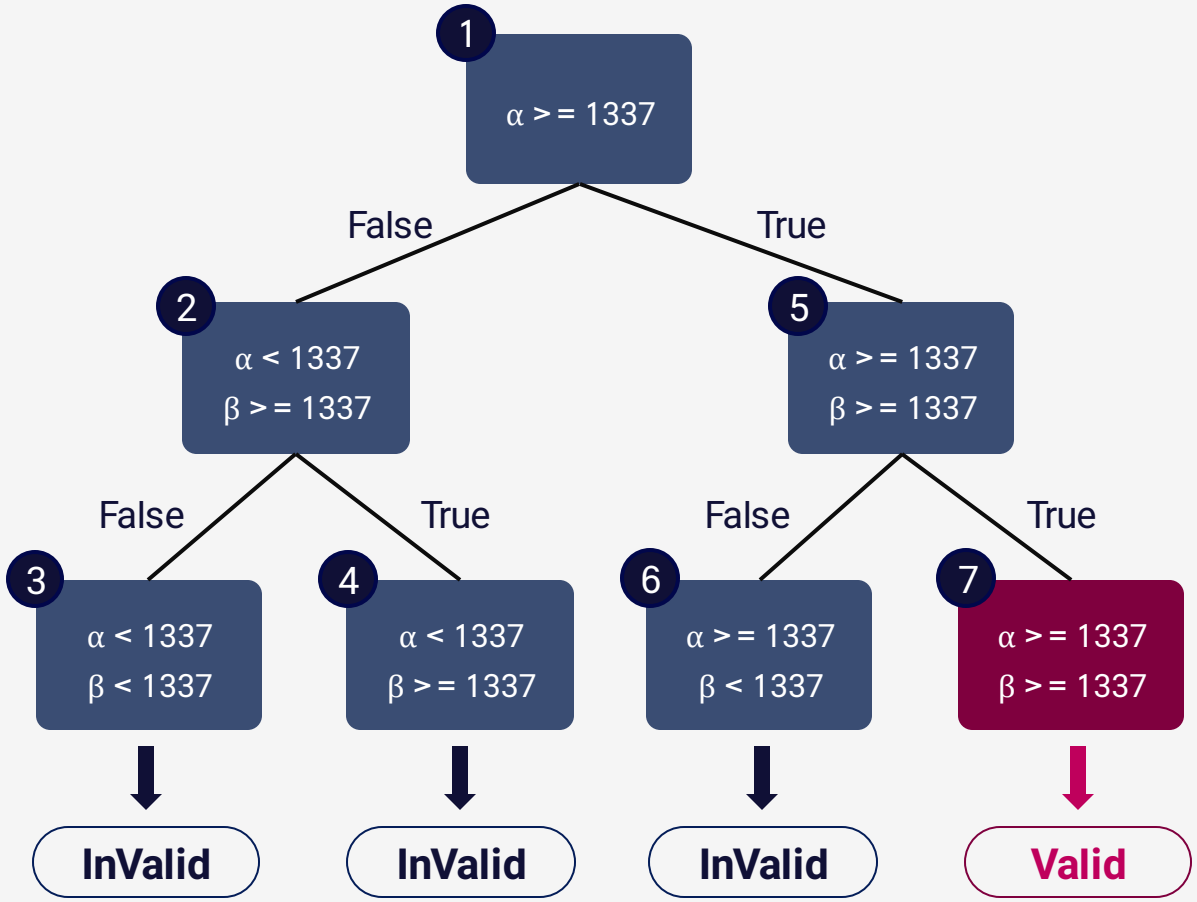


Symbolic Execution Example

- Analyze Path Constraints

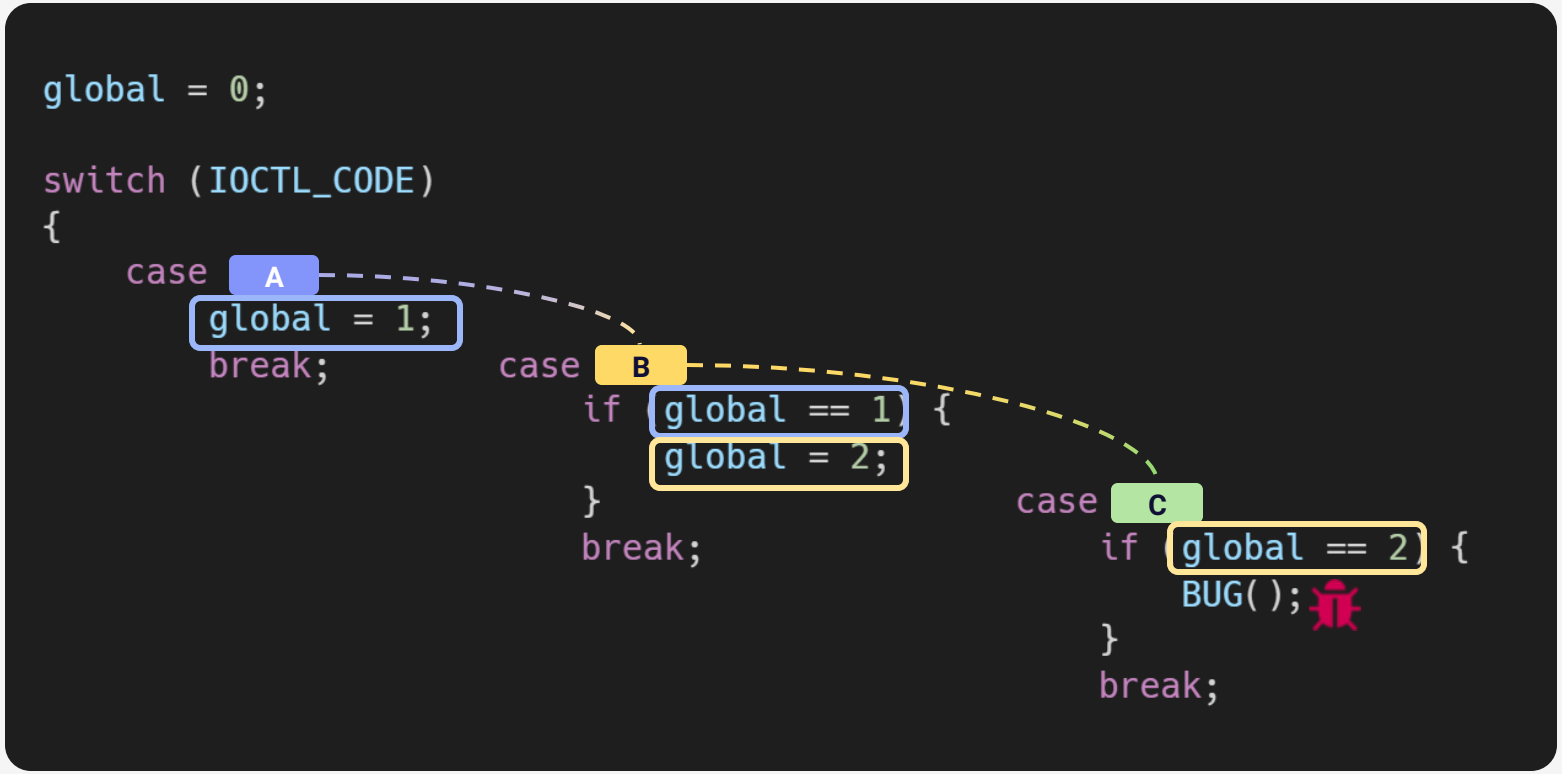
```
case IOCTL_CODE:
    InputBuffer = IRP->AssociatedIrp.SystemBuffer;
    OutputBuffer = IRP->AssociatedIrp.SystemBuffer;

    if (InputBufferLength >= 1337 & OutputBufferLength >= 1337) {
      IOCTL_Handler(InputBuffer, OutputBuffer);
      IRP->IoStatus.Status = STATUS_SUCCESS;
      ...
    } else {
      IRP->IoStatus.Status = STATUS_INVALID_PARAMETER;
    }

    break;
```

α β γ



Dependency

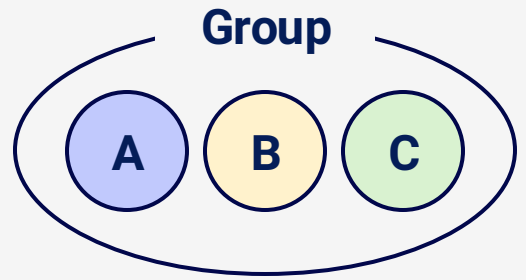
- Drivers are characterized by being stateful like network protocols
- Global variable is the "**Key State Variable**"



Analyze Dependency with Global Variables

- Static analyze operators with global variables as operands

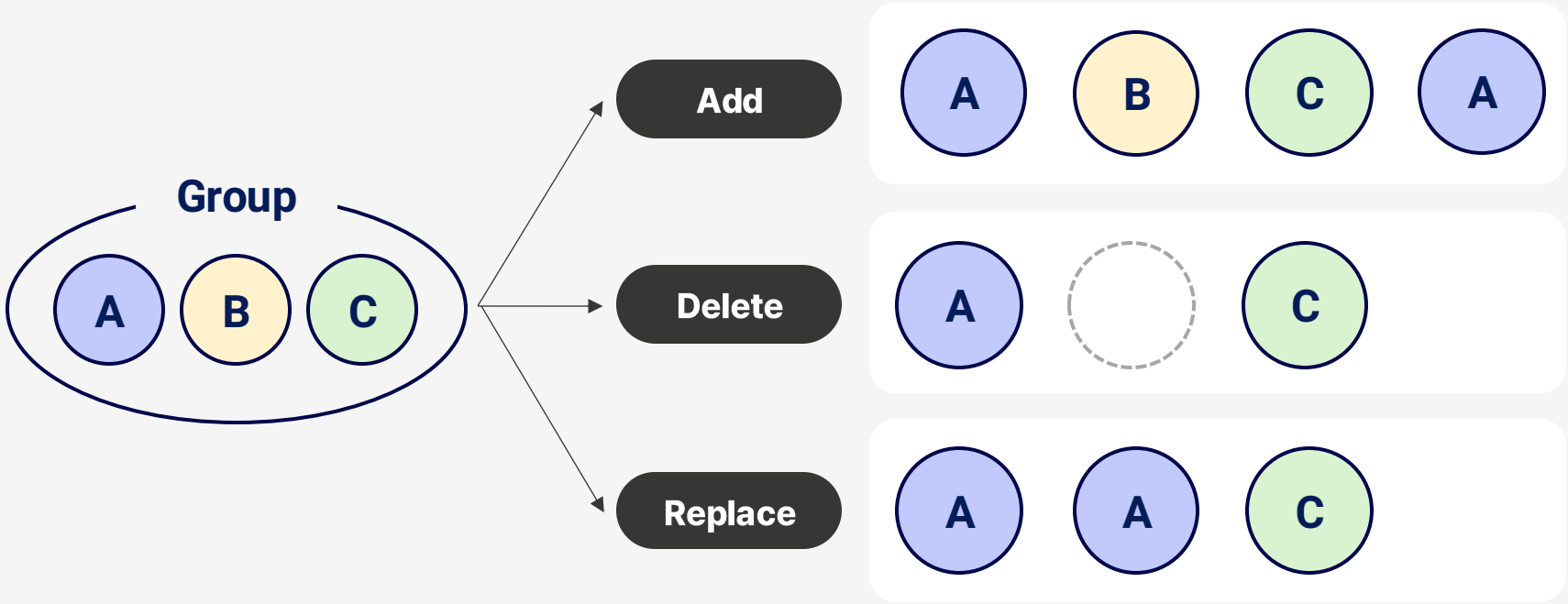
```
global = 0;

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

Bother fuzzing

More Coverage

```
case B:
    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; 💥 // OOB Write From 0x711E0;
        global++;
    } while(LoopCount) // Can Loop Control

    ...
    ...
}
```

IOCTL A Routine

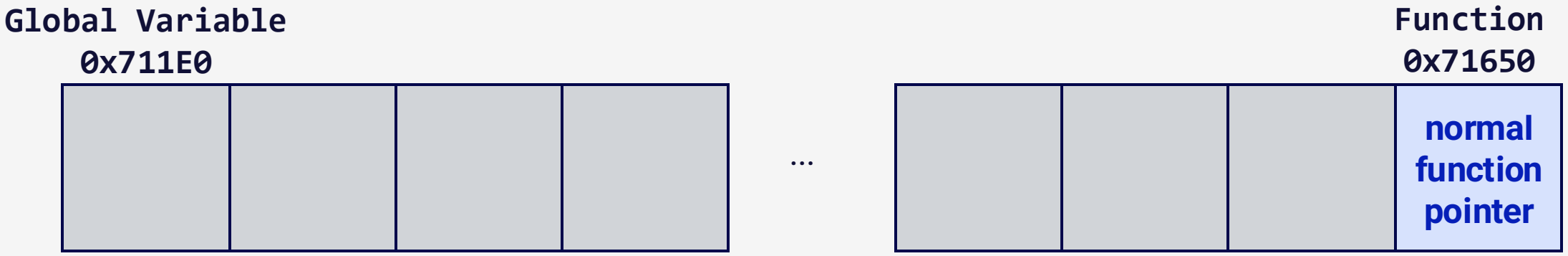
```
void sub_178A8(__int64 a1) {
    __int64 v2;

    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?

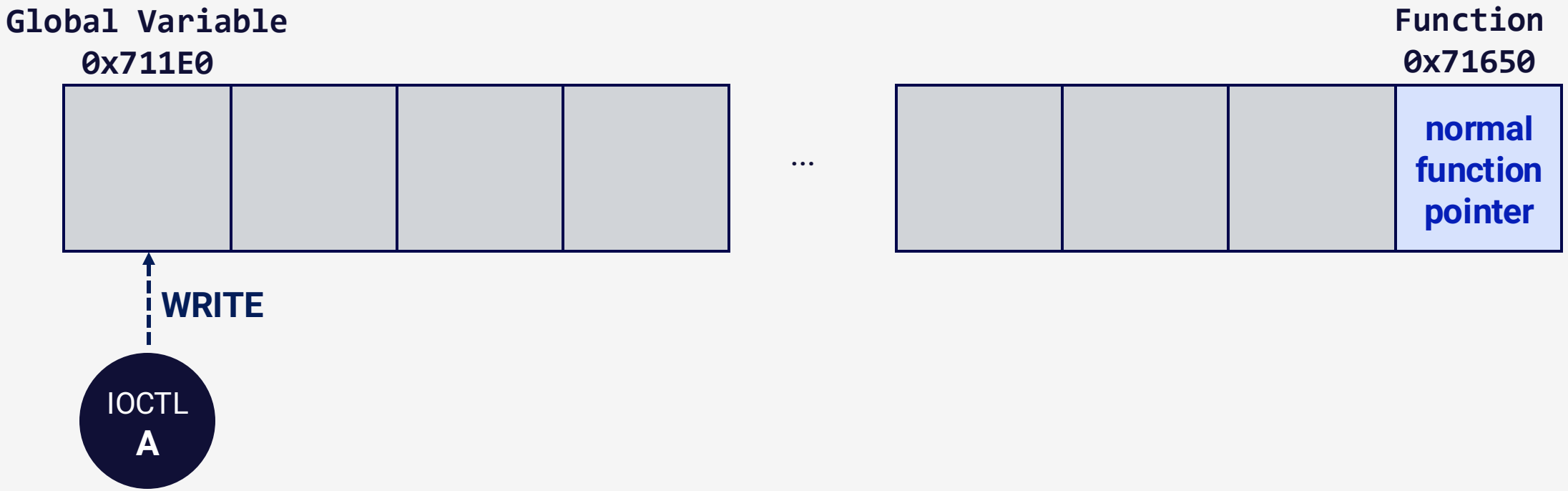
Not consider dependency



Real world case 2)

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

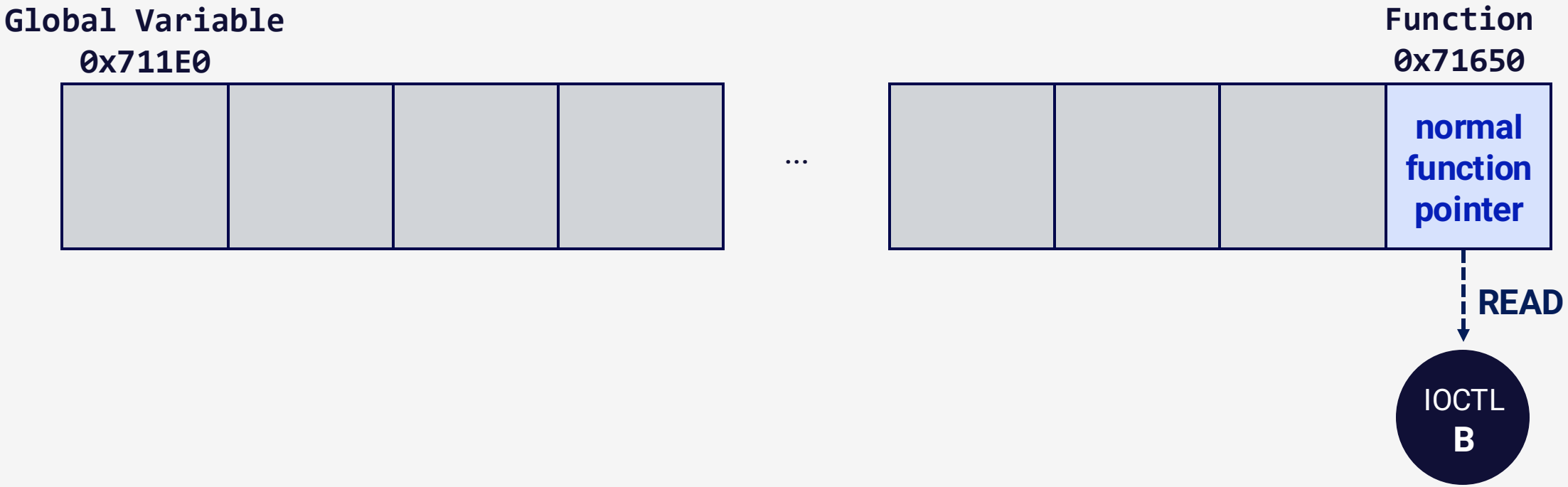
Not consider dependency



Real world case 2)

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

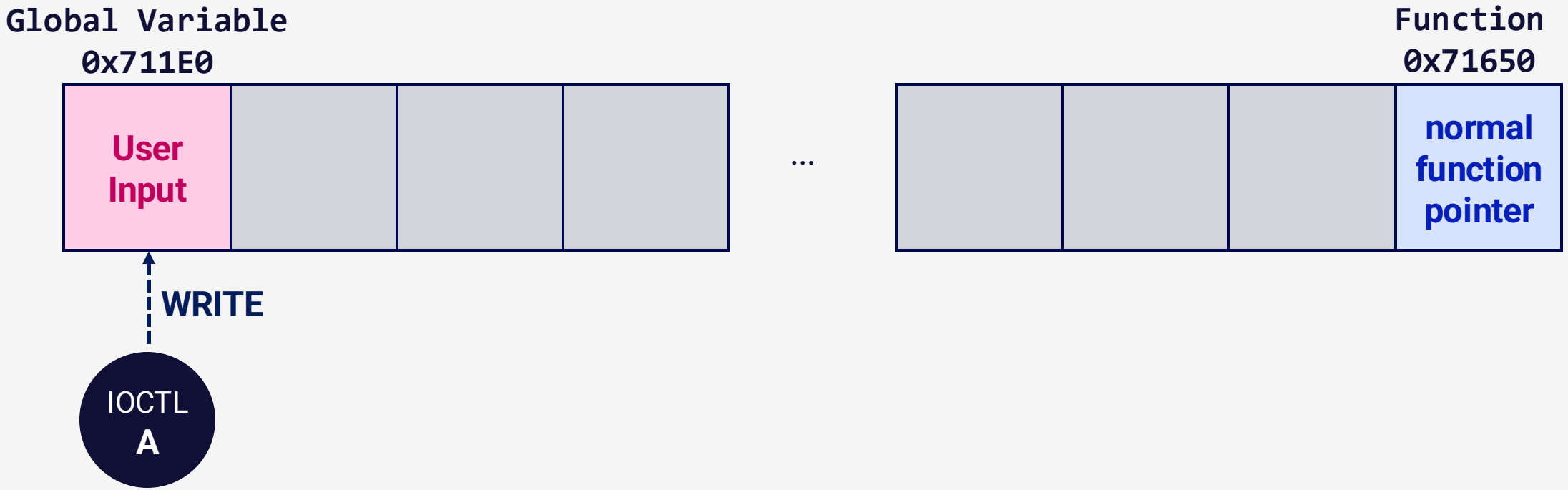
Not consider dependency



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**

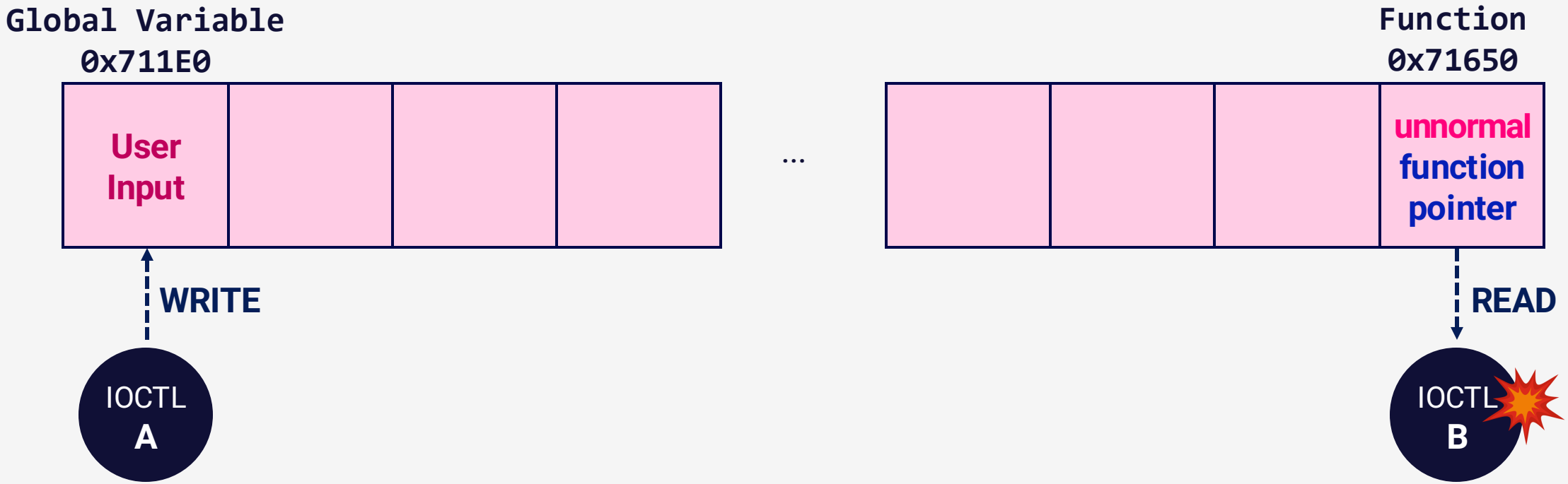
Consider dependency



Real world case 2)

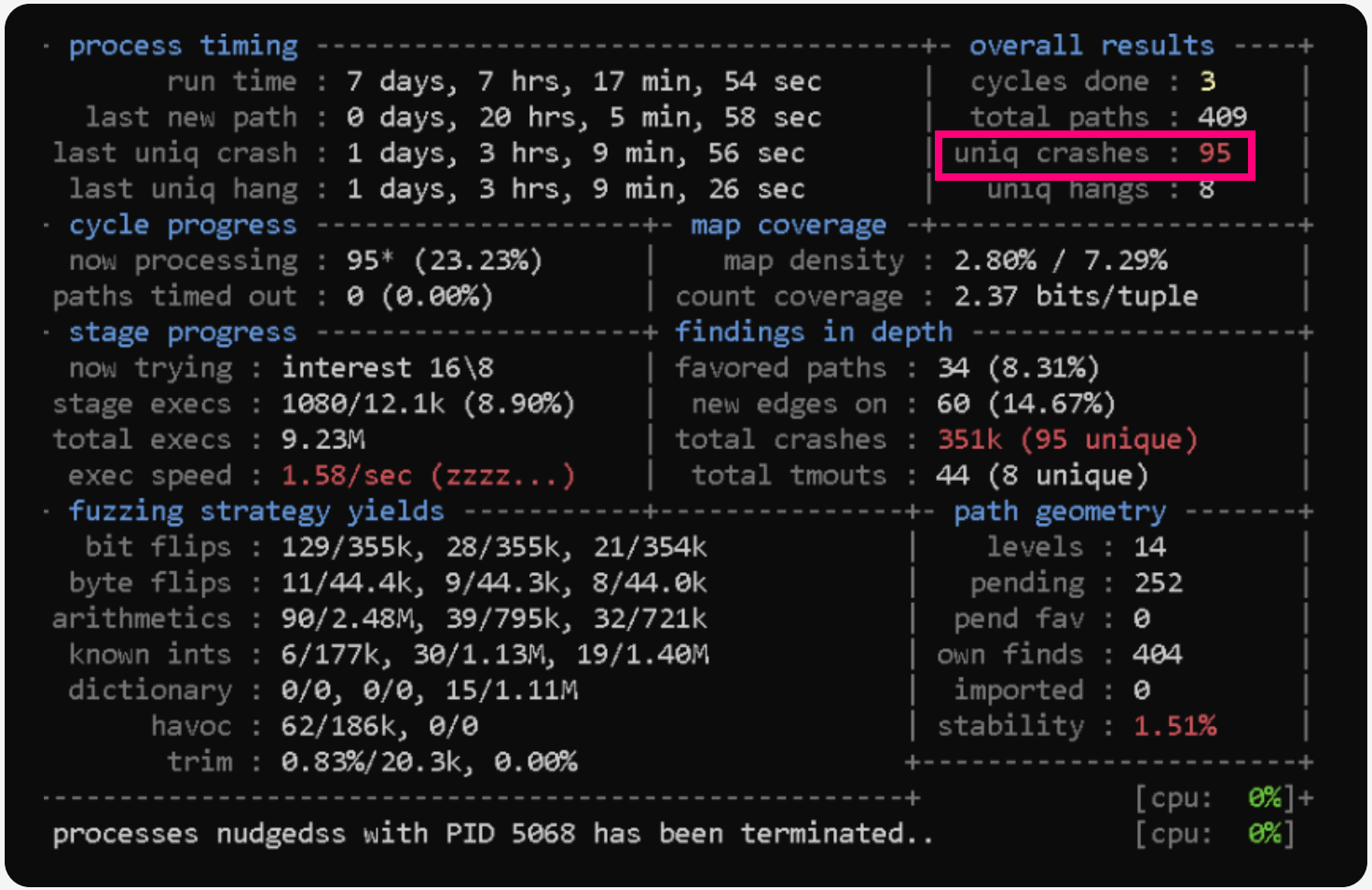
- Through IOCTL A, user can write data from **`0x711E0`**. (Out of Bounds Write)
- It means, **`0x71650`** at IOCTL B also can manipulate so that leads **RIP Control**

Consider dependency



Difficult of Handling duplicate crash

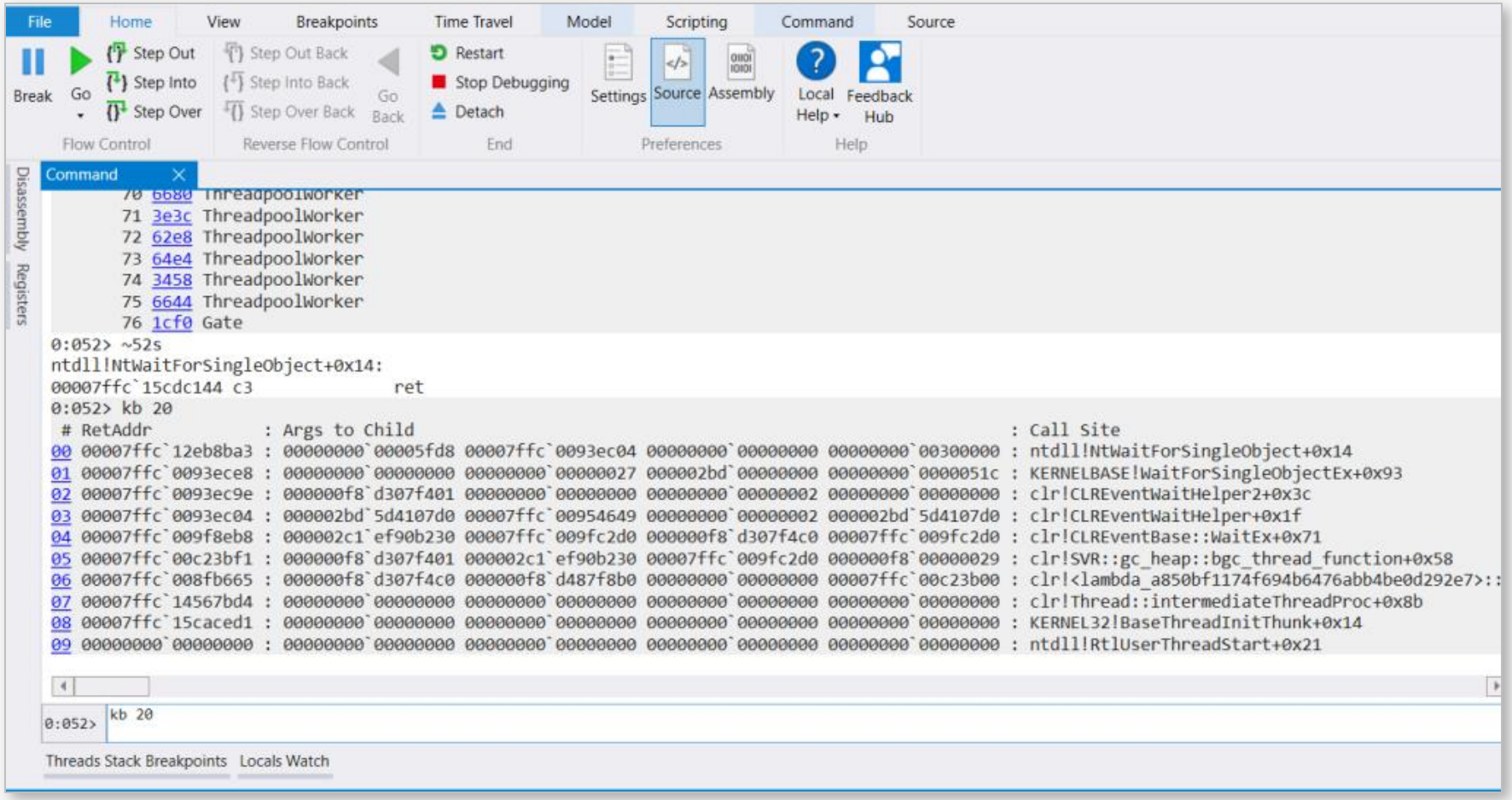
- AFL Fuzzer: "Same root cause? It's not a Unique Crash :D"



[1] https://spaceraccoon.dev/all-your-d-base-are-belong-to-us-part-2-code-execution-in-microsoft-office/

Difficult of Handling duplicate crash

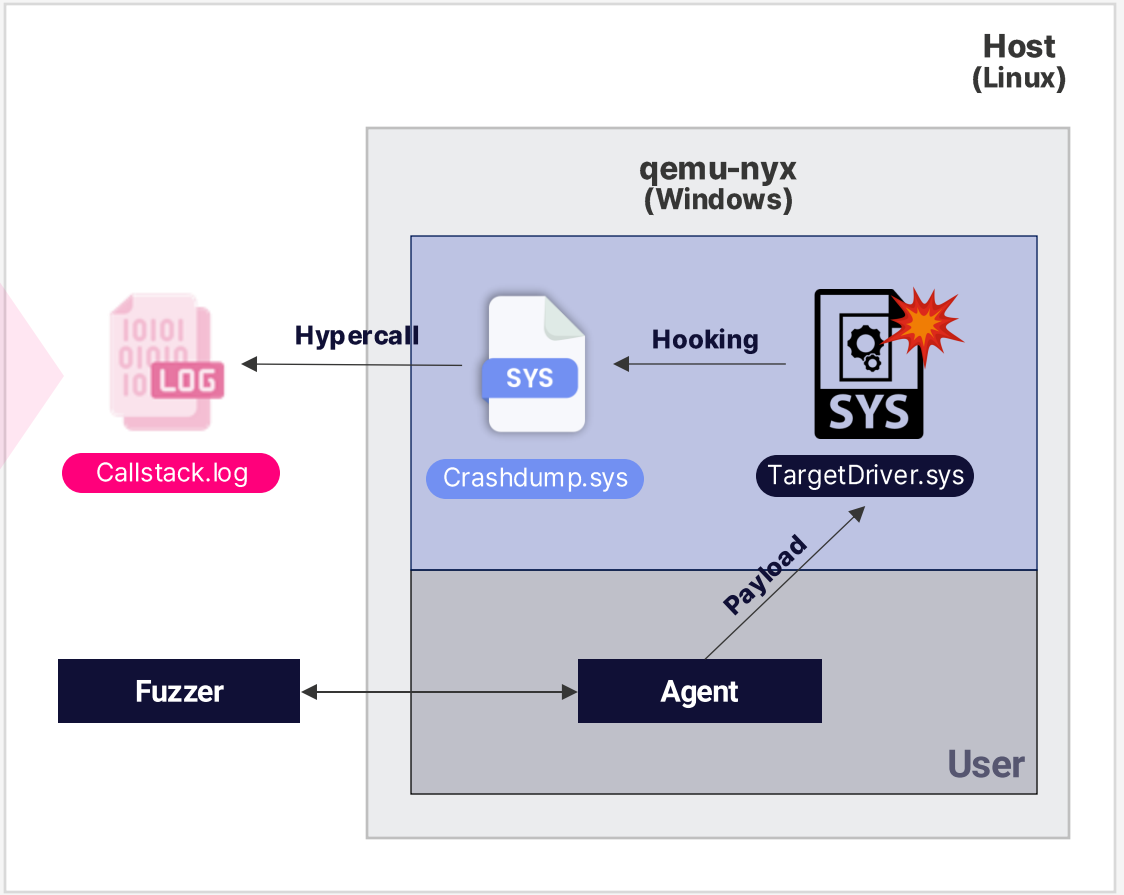
- Exhausting time to validate



Call Stack Parser

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

Register	Callstack
ffff9686`3ab9e4f0 fffff805`5837dec0 fffff805`73270000 00000000`00000000	nt!DbgBreakPointWithStatus
fffff805`00000003 ffff9686`3ab9e4f0 fffff805`58414190 ffff9686`3ab9ea40	nt!KiBugCheckDebugBreak+0x12
00000000`00000000 00000000`00000000 fffff804`9f26f2d8 fffff804`9f26f2d8	nt!KeBugCheck2+0x946
00000000`00000050 fffff804`9f26f2d8 00000000`00000002 ffff9686`3ab9ede0	nt!KeBugCheckEx+0x107
ffff8684`ab979a20 00000000`00000002 ffff9686`3ab9ee60 00000000`00000000	nt!MiSystemFault+0x1dda4f
00000000`00000001 ffff8684`b263da20 ffffffff`ffffffff fffff805`58254d26	nt!MmAccessFault+0x400
fffff805`732785f3 ffff8684`cafebabe 00000000`deadbeef ffffe306`00000002	nt!KiPageFault+0x358
00000000`00000001 ffff8684`b46e8e50 ffff8684`b2c4bd90 ffff8684`aeb65080	nt!IofCallDriver+0x55
ffff8684`b46e8e50 ffff9686`3ab9fb40 00000000`00010000 ffff8684`b46e8e50	nt!IopSynchronousServiceTail+0x34c
00000000`00000001 00000000`00000000 00000000`00000000 00000000`00000000	nt!IopXxxControlFile+0xc71
00000000`00000000 00000000`00000000 00000000`00000000 00000000`00000000	nt!NtDeviceIoControlFile+0x56
00007fff`b8e8591b 00000002`c00000bb 00000000`00000000 00008b50`a144092d	nt!KiSystemServiceCopyEnd+0x25
00000002`c00000bb 00000000`00000000 00008b50`a144092d 00007fff`b8e9b4ee	ntdll!NtDeviceIoControlFile+0x14
00000000`0022209c 00000000`00000000 00000000`00000000 00000000`00000000	KERNELBASE!DeviceIoControl+0x6b
00000000`00000000 0000013c`00000000 00000010`9f9efa80 00000000`00000000	KERNEL32!DeviceIoControlImplementation+0x81
0000013c`00000000 00000010`9f9efa80 00000000`00000000 0000013c`63df0000	Project2+0x11ac6



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

Vendor	CVE
Microsoft (1)	CVE-2024-21442
Jungo (4)	CVE-2023-47569 CVE-2023-47570 CVE-2023-47571 CVE-2023-47572
AMD (1)	CVE-2023-31341
Siemens (1)	CVE-2023-46280
Wibukey (2)	CVE-2024-45181 CVE-2024-45182

Vendor	CVE
Mitsubishi (12)	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

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

CODE BLUE 2024 @ TOKYO

Thank you

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

github.com/0dayResearchLab/msFuzz