The Art, Science, and Engineering of Windows driver fuzzing

윈도우 드라이버 퍼징 프레임워크 개발 이야기

조남준, 권현경

Index

I 개요 및 배경

II 윈도우 드라이버 퍼징 프레임워크 개발

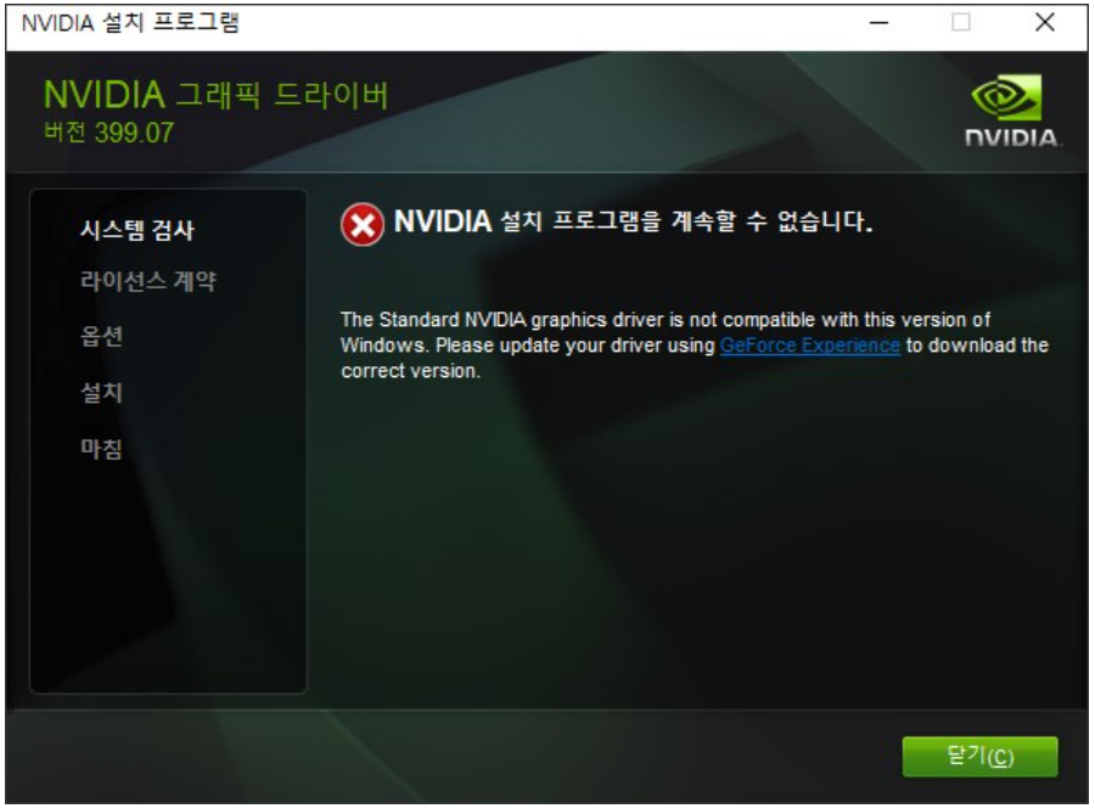
- Symbolic Execution을 활용한 드라이버 자동 분석 툴 **IREC**
- 드라이버 입력 Seed값을 수집하기 위한 **IRCAP**
- IRP 의존성을 해결한 Coverage-guided 퍼저 **IRPT**

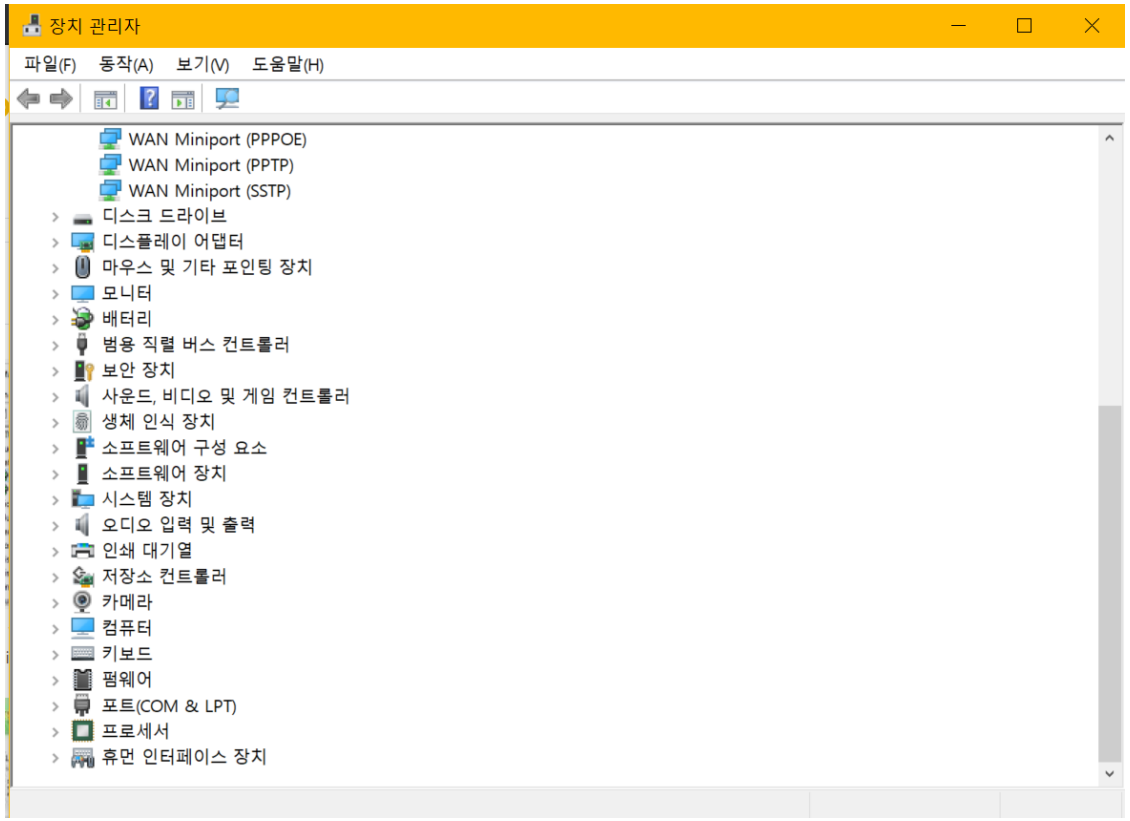
III 결론

개요 및 배경

드라이버란?

장치 드라이버, 백신 드라이버

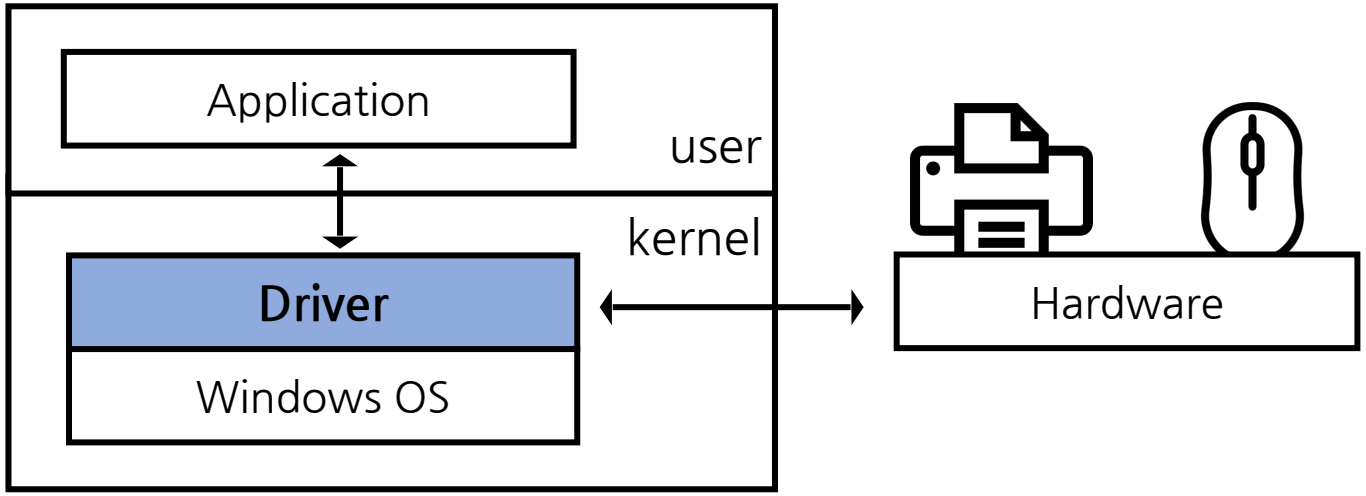




드라이버란?

드라이버의 정의

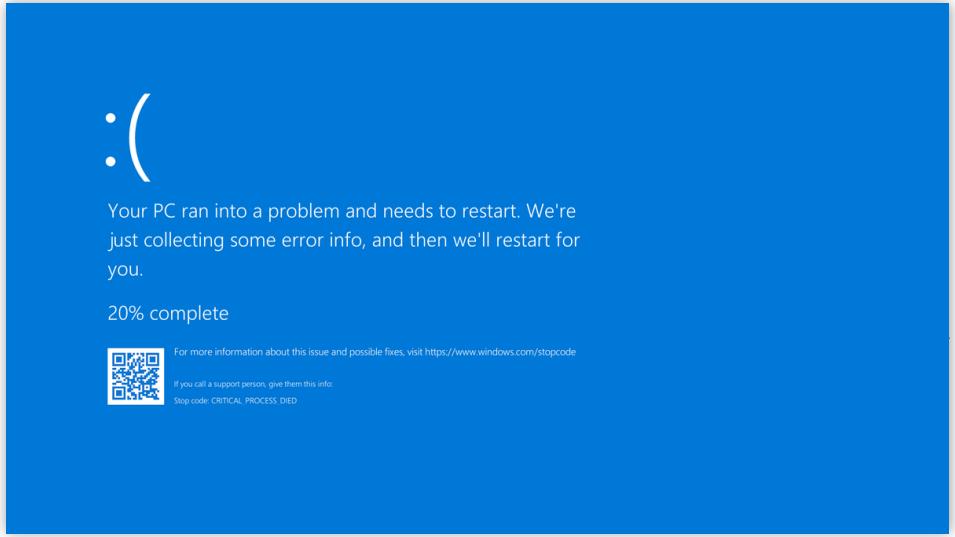
운영체제(OS)와 장치(Hardware)의 통신을 도와주는 소프트웨어

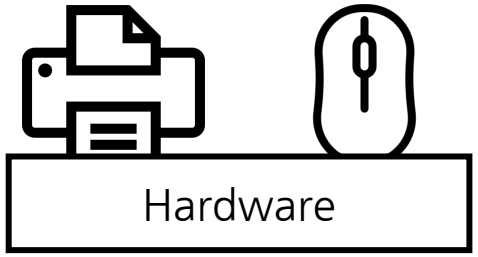


드라이버 취약점 분석의 필요성

커널에서 동작하는 드라이버

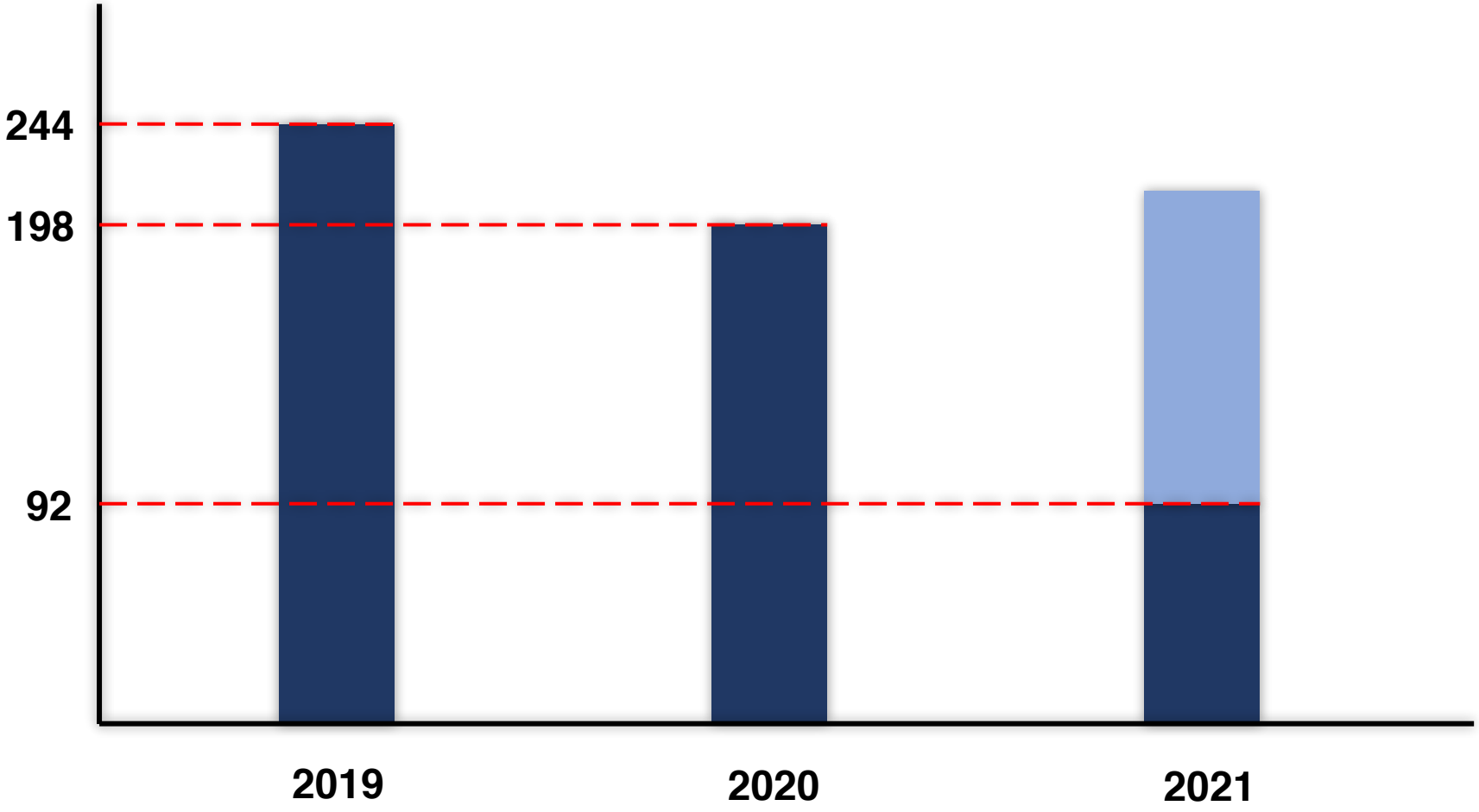
운영체제(OS)와 장치(Hardware)의 통신을 도와주는 소프트웨어





드라이버 취약점 분석의 필요성

취약점 발생 현황



드라이버 취약점 분석의 필요성

취약점 발생원인

1. 대부분의 장치 드라이버, 백신 드라이버는 **Third-party Application**
2. **오픈소스 드라이버 코드**(Winio.sys 등)를 검증없이 사용하여 드라이버 개발

```
case IOCTL_WINIO_WRITEPORT:

        KdPrint(("IOCTL_WINIO_WRITEPORT"));

        if (dwInputBufferLength)
        {
                memcpy (&PortStruct, pvIOBuffer, dwInputBufferLength)
```

오픈소스 코드

```
case 0x80102054:
  DbgPrint("IOCTL_MSIO_WRITEPORT");
  if ( !(_DWORD)v6 )
    goto LABEL_9;
  memmove(&Dst, v4, v6);
  switch ( v18 )
  {
```

M사 상용 드라이버 코드

드라이버 취약점 분석의 필요성

퍼징의 필요

1. 대부분의 장치 드라이버, 백신 드라이버는 **Third-party Application**
2. **오픈소스 드라이버 코드**(Winio.sys 등)를 검증없이 사용하여 드라이버 개발

개발코드에 대한 취약점 분석 자동화(퍼징, fuzzing)이 필요

```
case IOCTL_WINIO_WRITEPORT:

        KdPrint(("IOCTL_WINIO_WRITEPORT"));

        if (dwInputBufferLength)
        {
                memcpy (&PortStruct, pvIOBuffer, dwInputBufferLength)
```

오픈소스 코드

```
case 0x80102054:
  DbgPrint("IOCTL_MSIO_WRITEPORT");
  if ( !(_DWORD)v6 )
    goto LABEL_9;
  memmove(&Dst, v4, v6);
  switch ( v18 )
  {
```

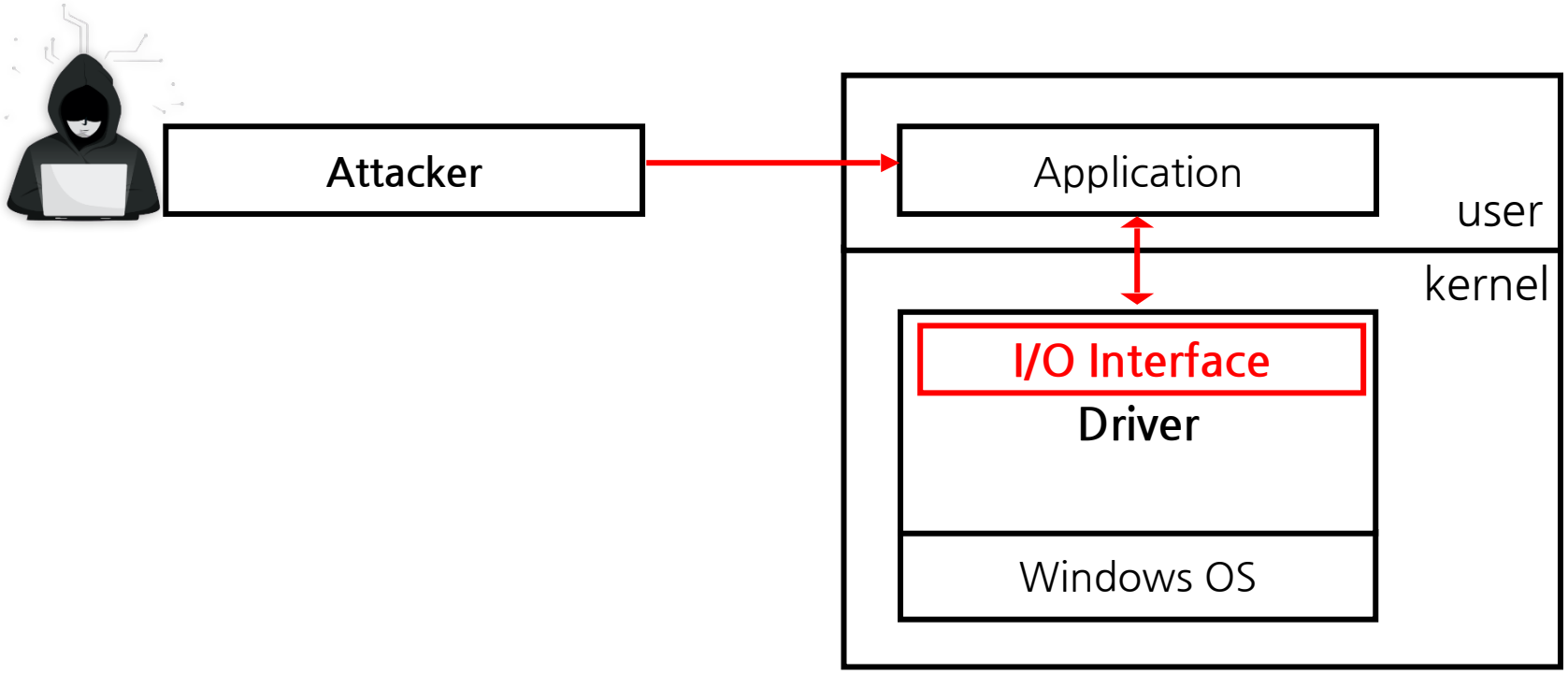
M사 상용 드라이버 코드

드라이버 취약점 분석의 필요성

Attack Vector

통신 인터페이스(I/O Interface)

- 유저모드 어플리케이션이 보낸 데이터를 처리하는 곳



윈도우 드라이버 퍼징 프레임워크

프레임워크 구성도

Kronl

IREC	IRCAP	IRPT
• Driver analysis • Interface Recovery	• Seed selection	• Optimized Fuzzing • Reproduction

윈도우 드라이버 퍼징 프레임워크 개발

IREC

통신 인터페이스 작동방식

Application

I/O request packet

Driver

DriverDispatch

```
switch ( IoControlCode )
{
  case 0xA0006810:
    if ( InputBufferLength < 8 )
      goto INVAILD_ARGUMENT;
    v17 = sub_140009750(&InputBuff
er);
    break;
  case 0xA0006814:
    if ( InputBufferLength )
      goto INVAILD_ARGUMENT;
    v17 = sub_1400247F8(0);
```

IREC

통신 인터페이스 작동방식

Application

I/O request packet

IoControlCode

Driver

DriverDispatch

```
switch ( IoControlCode )
{
  case 0xA0006810:
    if ( InputBufferLength < 8 )
      goto INVAILD_ARGUMENT;
    v17 = sub_140009750(&InputBuff
er);
    break;
  case 0xA0006814:
    if ( InputBufferLength )
      goto INVAILD_ARGUMENT;
    v17 = sub_1400247F8(0);
```

IREC

통신 인터페이스 작동방식

Application

I/O request packet

IoControlCode
InputBufferLength
OutputBufferLength

Driver

DriverDispatch

```
switch ( IoControlCode )
{
  case 0xA0006810:
    if ( InputBufferLength < 8 )
      goto INVAILD_ARGUMENT;
    v17 = sub_140009750(&InputBuff
er);
    break;
  case 0xA0006814:
    if ( InputBufferLength )
      goto INVAILD_ARGUMENT;
    v17 = sub_1400247F8(0);
```

IREC

분석을 위해 필요한 정보

1. **DriverDispatch 함수 주소** 찾기

2. 모든 **IoControlCode** 알아내기

3. **제약조건**(Constraint) 알아내기

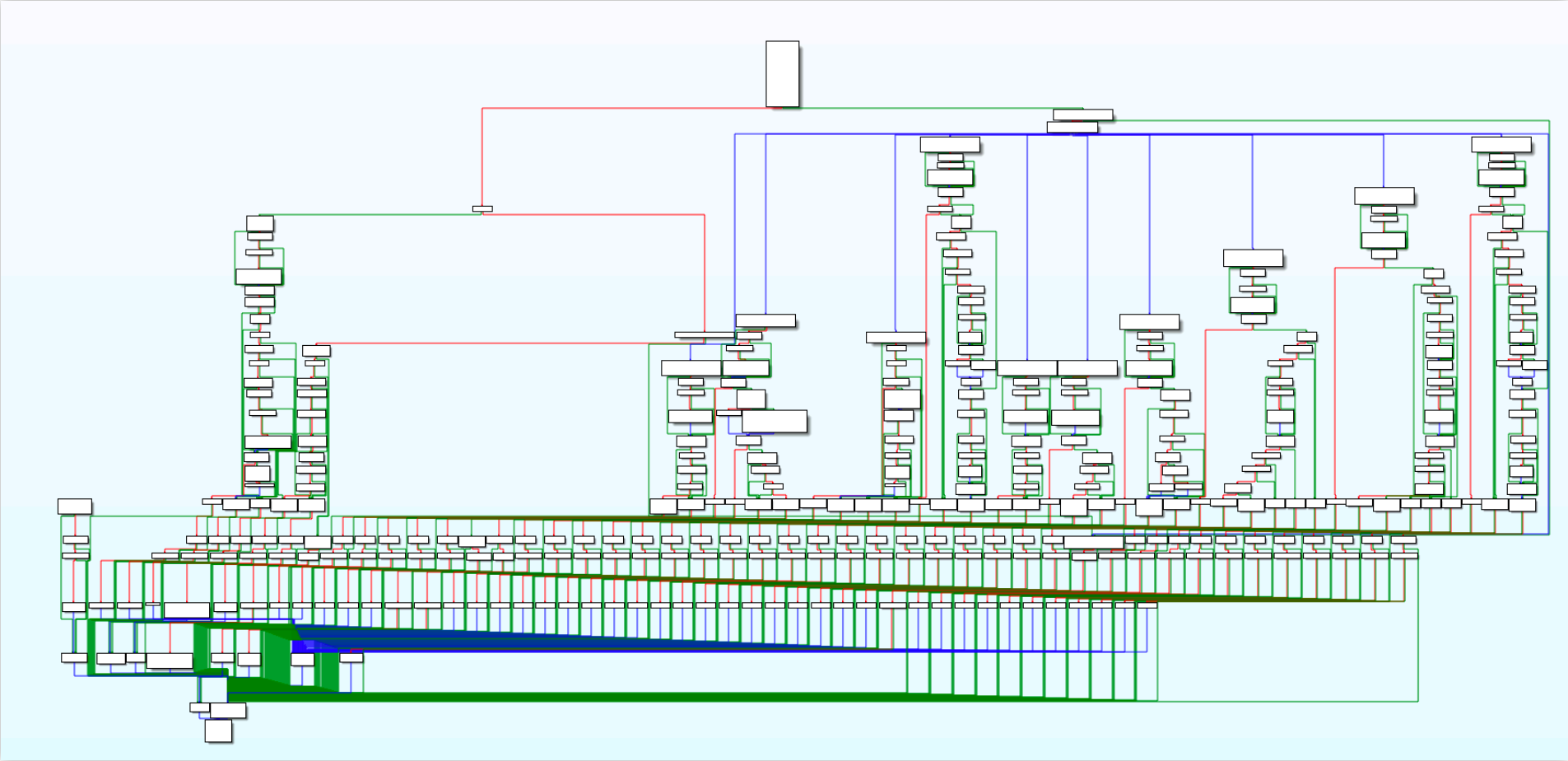
Driver

DriverDispatch

```
switch ( IoControlCode )
{
  case 0xA0006810:
    if ( InputBufferLength < 8 )
      goto INVAILD_ARGUMENT;
    v17 = sub_140009750(&InputBuff
er);
    break;
  case 0xA0006814:
    if ( InputBufferLength )
      goto INVAILD_ARGUMENT;
    v17 = sub_1400247F8(0);
```

IREC

실제 드라이버 통신 코드 그래프

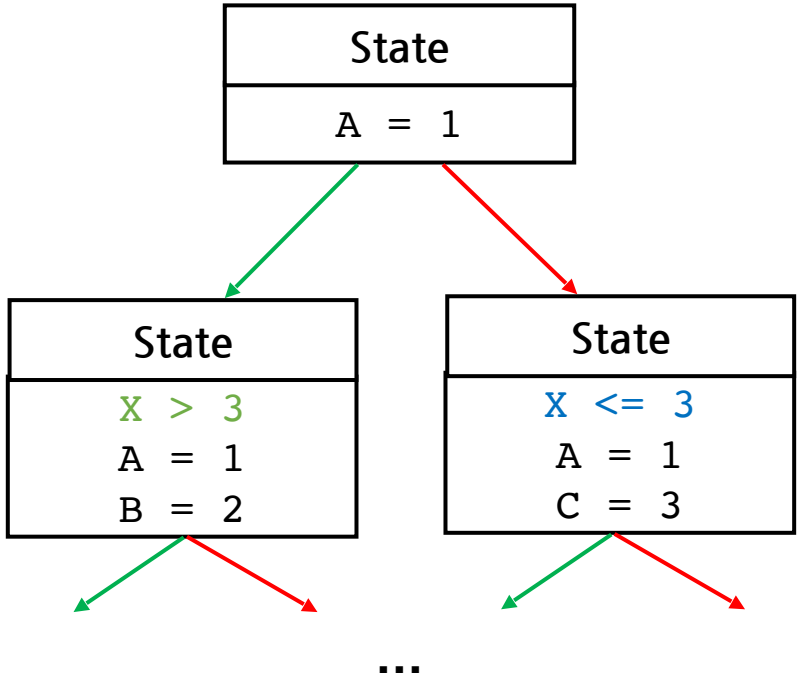


IREC

Symbolic Execution

- 프로그램의 실행 경로를 탐색하는 기법

```
A = 1;
if ( x > 3 ) {
        B = 2;
        if ( y < 4) {
                ...
        }
}
else {
        C = 3;
        if ( z < 4) {
                ...
        }
}
```



IREC

DriverDispatch 함수 주소 찾기

- 메모리 BreakPoint를 사용

```
NTSTATUS
DriverEntry(
        _In_ PDRIVER_OBJECT DriverObject,
        _In_ PUNICODE_STRING RegistryPath
)
{
        ...
        DriverObject->MajorFunction[IRP_MJ_DEVICE_CONTROL]
        ...
}
```

DriverDispatch

IREC

모든 IoControlCode 알아내기

- IoControlCode에 심볼릭 변수 설정

```
switch (IoControlCode)
{
        case 0xa0001:
                func1();
        case 0xa0002:
                func2();
        case 0xa0003:
                func3();
        case 0xa0004:
                func4();
}
```

SymValue

IREC

모든 IoControlCode 알아내기

- State가 SymValue를 하나의 값으로 특정 할때 IoControlCode를 가져옴

```
switch (IoControlCode)
{
        case 0xa0001:
                func1();
        case 0xa0002:
                func2();
        case 0xa0003:
                func3();
        case 0xa0004:
                func4();
}
```

SymValue

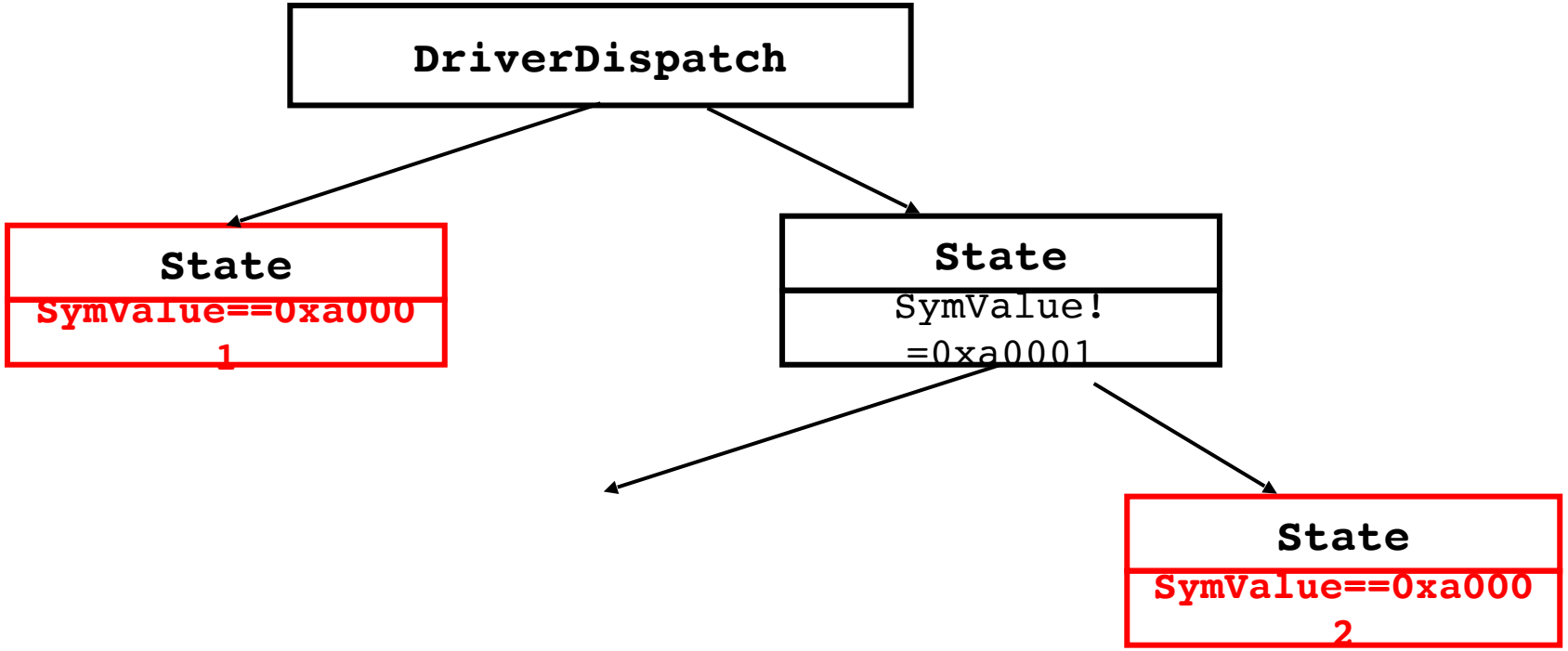
State

SymValue==0xa0001

IREC

모든 IoControlCode 알아내기

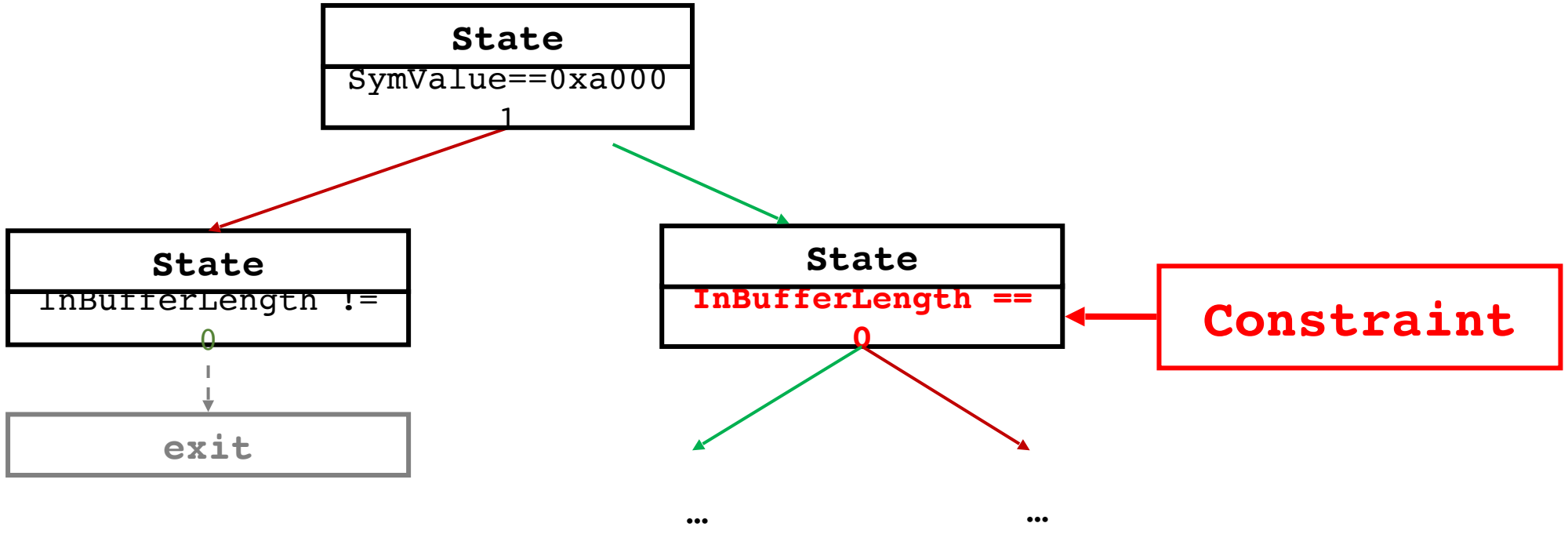
- IoControlCode를 특정하고 나면 더 이상 해당 경로를 탐색하지 않음



IREC

제약조건(Constraint) 알아내기

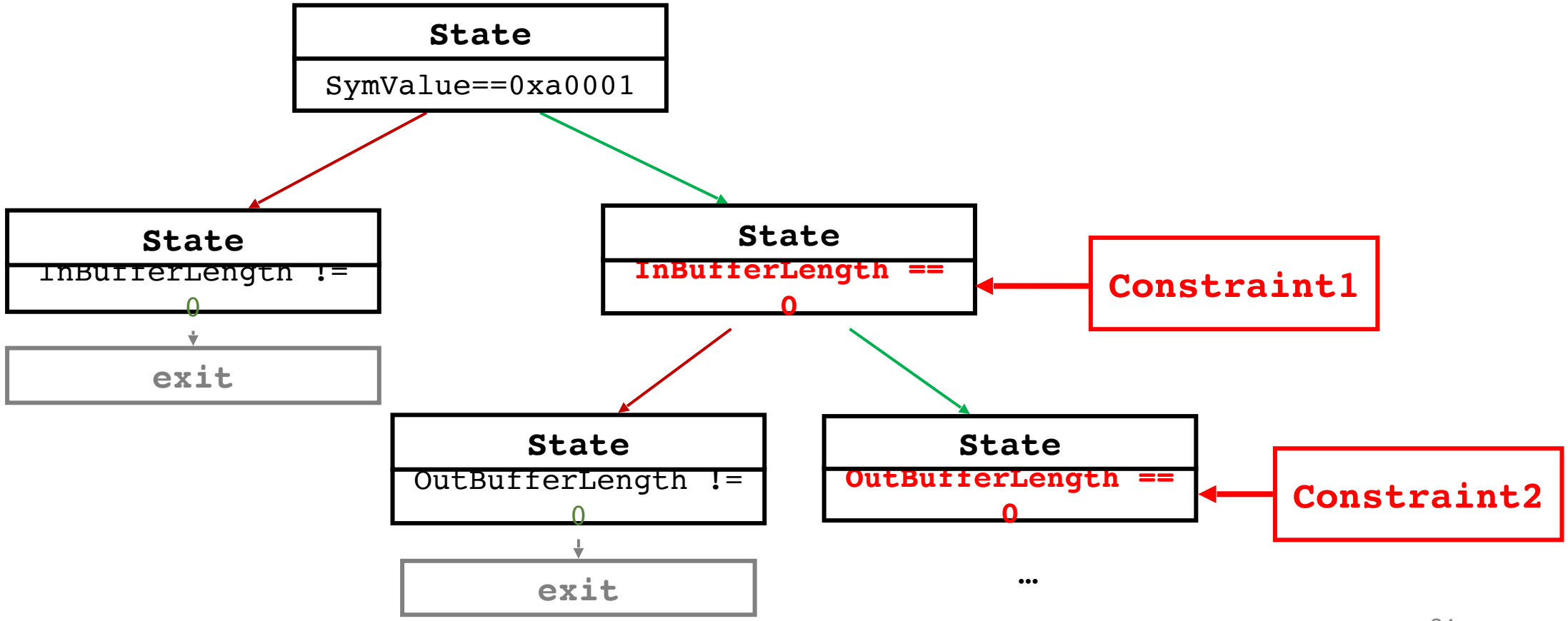
- 조건을 만족하지 못하면 바로 종료되는 특성을 이용



IREC

제약조건(Constraint) 알아내기

- 모든 조건을 찾아야 하므로 exit이 아닌 경로가 하나가 나올때까지 실행



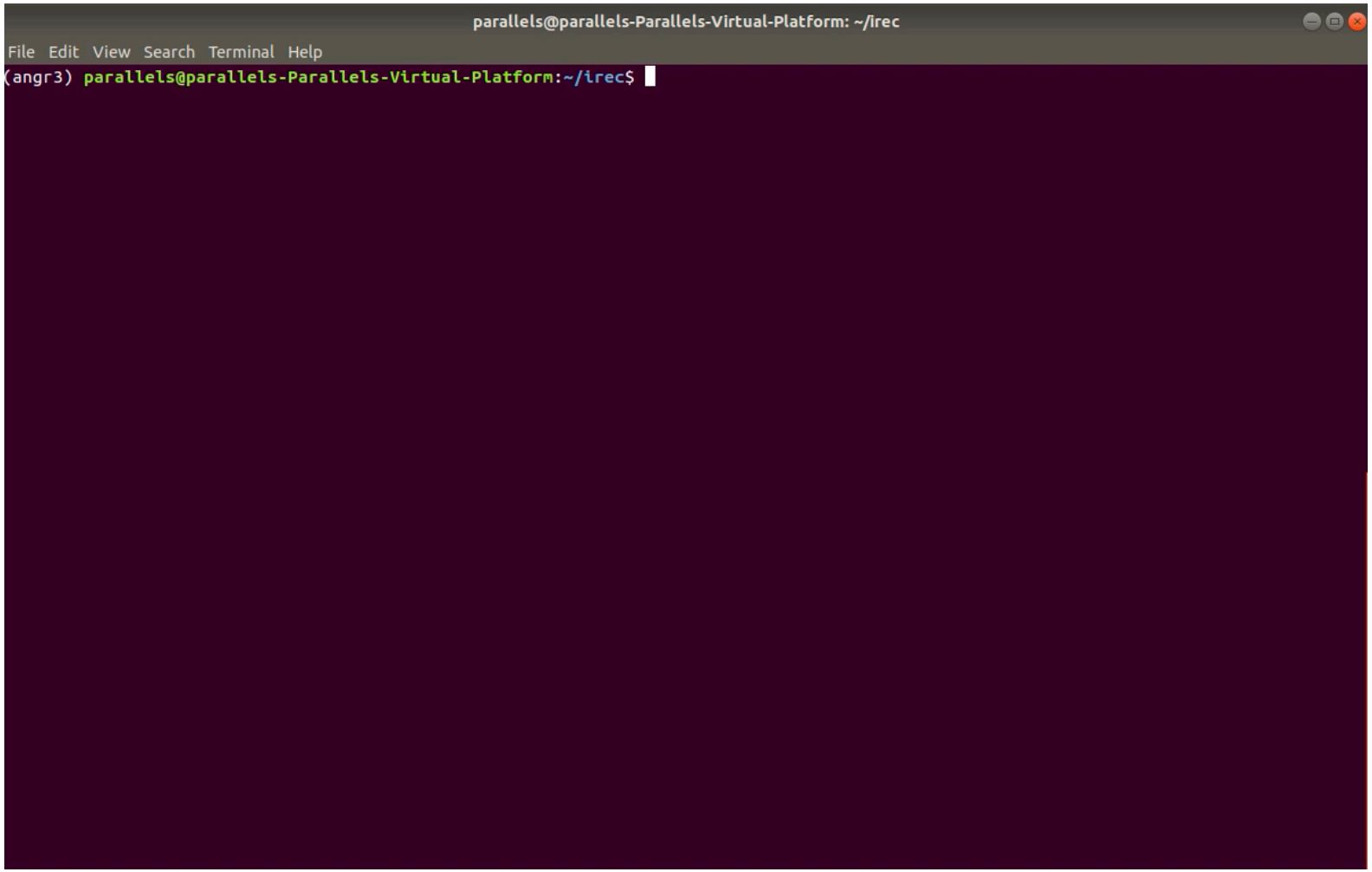
IREC

분석 결과

Driver	Product	Codes	Constraints	Time
medcored.sys	Ahnlab V3 Lite	43/43	43/43	48 seconds
amdfendr.sys	AMD Adrenalin 2020	2/2	2/2	15 seconds
vmnetuserif.sys	Vmware workstation	29/29	29/29	22 seconds
VSPerfDrv10.sys	Visual Studio 2019	25/25	25/25	29 seconds
acpi.sys	Windows Kernel	2/2	2/2	20 seconds
TKIdsVt64.sys	nProtect Online Security	43/43	43/43	2 minutes
Iqvm64e.sys	Intel Ethernet Adapters 700	4/4	4/4	15 seconds
Npcap.sys	Wireshark	16/16	16/16	17 seconds

IREC

Demo


parallels@parallels-Parallels-Virtual-Platform: ~/irec
File Edit View Search Terminal Help
(angr3) parallels@parallels-Parallels-Virtual-Platform:~/irec$

IRCAP

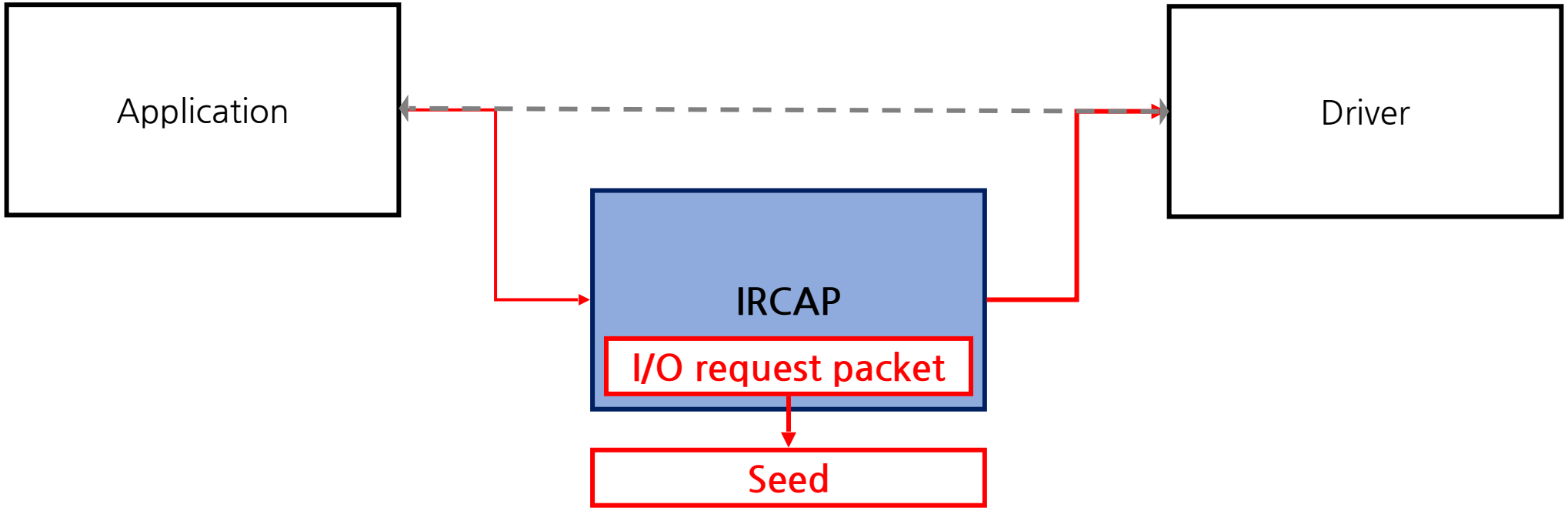
IRCAP 동작방식

Seed 선택을 위해 실제 드라이버에 대한 DeviceIoControl 요청을 Hooking

IRCAP

IRCAP 동작방식

Seed 선택을 위해 실제 드라이버에 대한 DeviceIoControl 요청을 Hooking



IRPT

IRPT Overview

- 효율적인 드라이버 퍼징을 위해

1. 좋은 Input을 생성 및 뮤테이션
2. 퍼저와 통신 과정에서 병목 현상이 없어야 함
3. 크래시를 정확히 판별 및 재현(reproduce) 가능
4. 오버헤드가 적은 Coverage Feedback 퍼징을 구현

kAFL: Hardware-Assisted Feedback Fuzzing for OS Kernels

Sergej Schumilo
Ruhr-Universität Bochum

Cornelius Aschermann
Ruhr-Universität Bochum

Robert Gawlik
Ruhr-Universität Bochum

Sebastian Schinzel
Münster University of Applied Sciences

Thorsten Holz
Ruhr-Universität Bochum

Abstract

Many kinds of memory safety vulnerabilities have been endangering software systems for decades. Amongst other approaches, fuzzing is a promising technique to unveil various software faults. Recently free vulnerabilities, are known threats for programs running in user mode as well as for the operating system (OS) core itself. Past experience has shown that attackers typically focus on user mode applications. This is likely because vulnerabilities in user mode programs are

IRPT

Part 1 - Input Generation

- 드라이버 퍼징에서 Input이란?

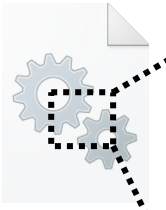
=> 드라이버에 보내는 IOCTL 요청(IRP)을 퍼징 Input으로 정함

$ driver_ioctl.exe

```
1.  h = OpenDriver();
2.  DeviceIoControl(h,
        IoControlCode 0xA001234,
        InputBuffer "aaaa",
        InputBufferLength 4);
```

IOCTL 요청

Target.sys



```
switch ( IoControlCode )
{
  case 0xA001234:
    if ( InputBufferLength != 8 )
      goto INVAILD_ARGUMENT;
    v17 = sub_140009750(&InputBuffer)
;
    break;
  default:
      goto INVAILD_ARGUMENT;
}
```

IOCTL 처리 함수

IRPT

Part 1 - Input Generation

- 드라이버 퍼징에서 좋은 Input이란?

```
switch ( IoControlCode )
{
  case 0xA001234:
    if ( InputBufferLength != 8 )
      goto INVAILD_ARGUMENT;
    v17 = sub_140009750(&InputBuffer);
    break;
  default:
      goto INVAILD_ARGUMENT;
}
```

=> 정상적인 컨트롤 코드와 제약 조건을 만족하는 IOCTL 요청

```
DeviceIoControl(0xA001234, “aaaaaaaa”, 8);
DeviceIoControl(0xA001234, “aaaaaaab”, 8);
DeviceIoControl(0xA001234, “aaaaaaac”, 8);
DeviceIoControl(0xA001234, “aaaaaaad”, 8);
```

IRPT

Part 1 - Input Generation

- 뮤테이션할 데이터나 범위를 줄일수록 효율적

Good Inputs

```
DeviceIoControl(0xA001234, “aaaaaaaa”, 8);
DeviceIoControl(0xA001234, “aaaaaaab”, 8);
DeviceIoControl(0xA001234, “aaaaacab”, 8);
DeviceIoControl(0xA001234, “aaaaaddd”, 8);
```

Bad Inputs

```
DeviceIoControl(0xA001235, “aaaaaaaa”, 8);
DeviceIoControl(0xA001234, “aaaaaaabc”, 9);
DeviceIoControl(0xA001236, “aaaaacab”, 8);
DeviceIoControl(0xA001234, “aaaaaddd”, 10);
```

→

```
switch ( IoControlCode )
{
  case 0xA001234:
    if ( InputBufferLength != 8 )
      goto INVAILD_ARGUMENT;
    v17 = sub_140009750(&InputBuffer);
    break;
  default:
      goto INVAILD_ARGUMENT;
}
```

→

```
switch ( IoControlCode )
{
  case 0xA001234:
    if ( InputBufferLength != 8 )
      goto INVAILD_ARGUMENT;
    v17 = sub_140009750(&InputBuffer);
    break;
  default:
      goto INVAILD_ARGUMENT;
}
```

IRPT

Part 1 - Input Generation

- 좋은 Input을 생성하기 위해선?

=> IREC에서 복구한 IOCTL 인터페이스 정보를 이용

```
{
   "IoControlCode":"0xA0001",
   "InBufferLength":[
      "4-4"
   ],
   "OutBufferLength":[
      "20-20"
   ],
}
```

IOCTL Interface

Generation

Mutation

Good Inputs

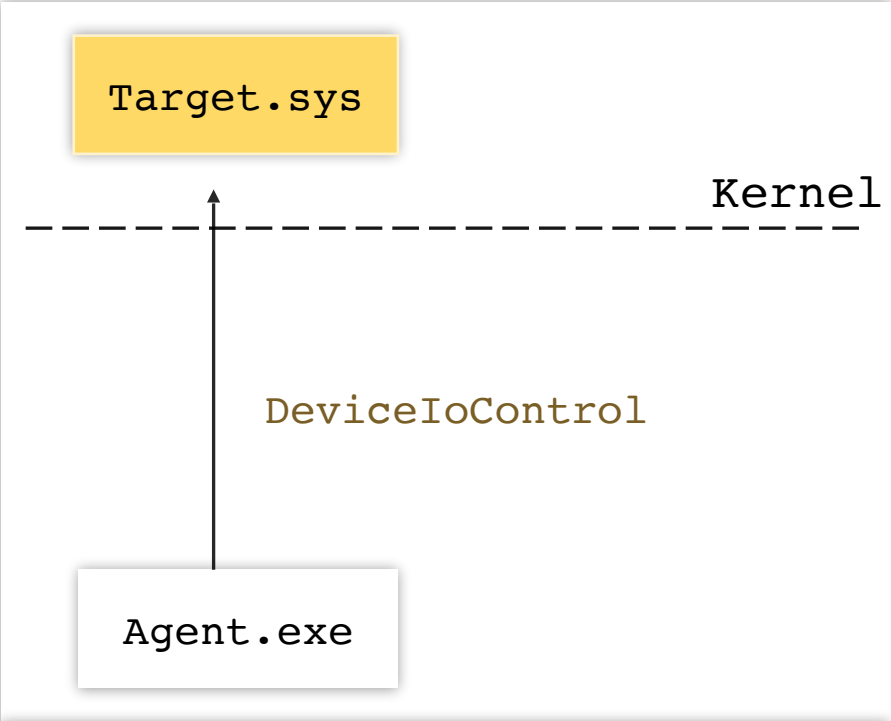
```
DeviceIoControl(0xA0001, “aaaa”, 4);
DeviceIoControl(0xA0001, “aaab”, 4);
DeviceIoControl(0xA0001, “acab”, 4);
DeviceIoControl(0xA0001, “addd”, 4);
```

```
switch ( IoControlCode )
{
  case 0xA0001:
    if ( InputBufferLength != 4 )
      goto INVAILD_ARGUMENT;
    v17 = sub_140009750(&InputBuffer);
    break;
  default:
      goto INVAILD_ARGUMENT;
}
```

IRPT

Part 2 - Fuzzing environment

- 윈도우 드라이버 퍼징 환경 구현



Windows 로컬 환경에서 퍼징

=>

윈도우 로컬 환경에서 퍼징을 돌리면

크래시 및 오류에 대응하기 매우 어려움

:(

Your PC ran into a problem and needs to restart. We're just collecting some error info, and then we'll restart for you.

20% complete

For more information about this issue and possible fixes, visit https://www.windows.com/stopcode

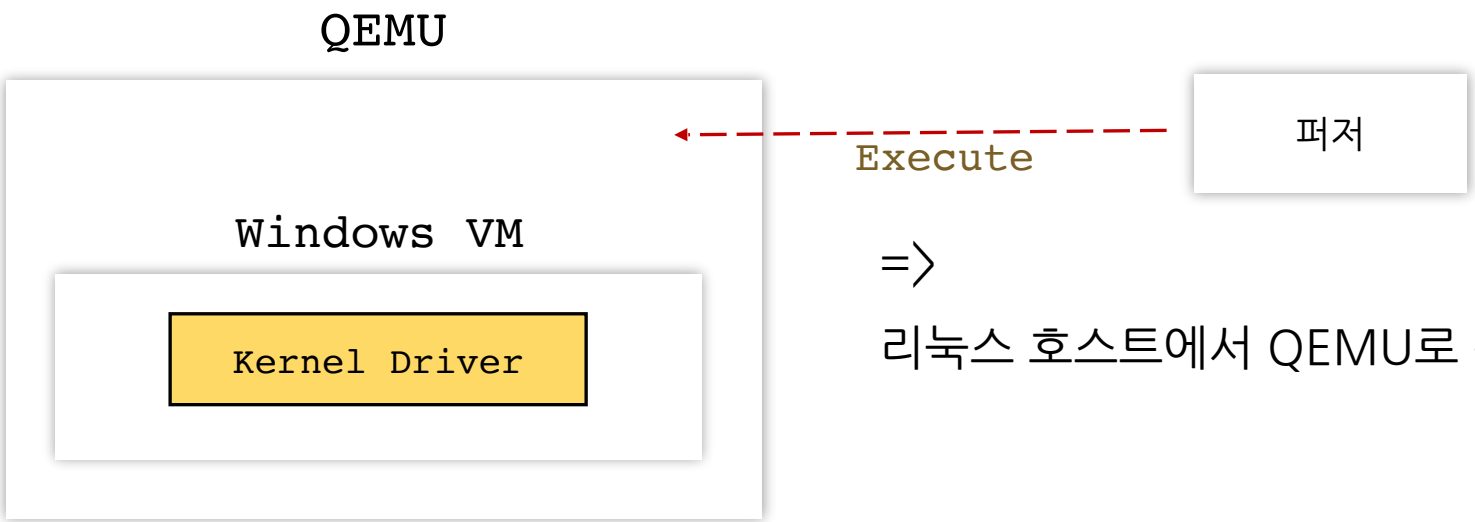
If you call a support person, give them this info:

Stop code: CRITICAL_PROCESS_DIED

IRPT

Part 2 - Fuzzing environment

- QEMU로 윈도우 커널 퍼징 환경 구성



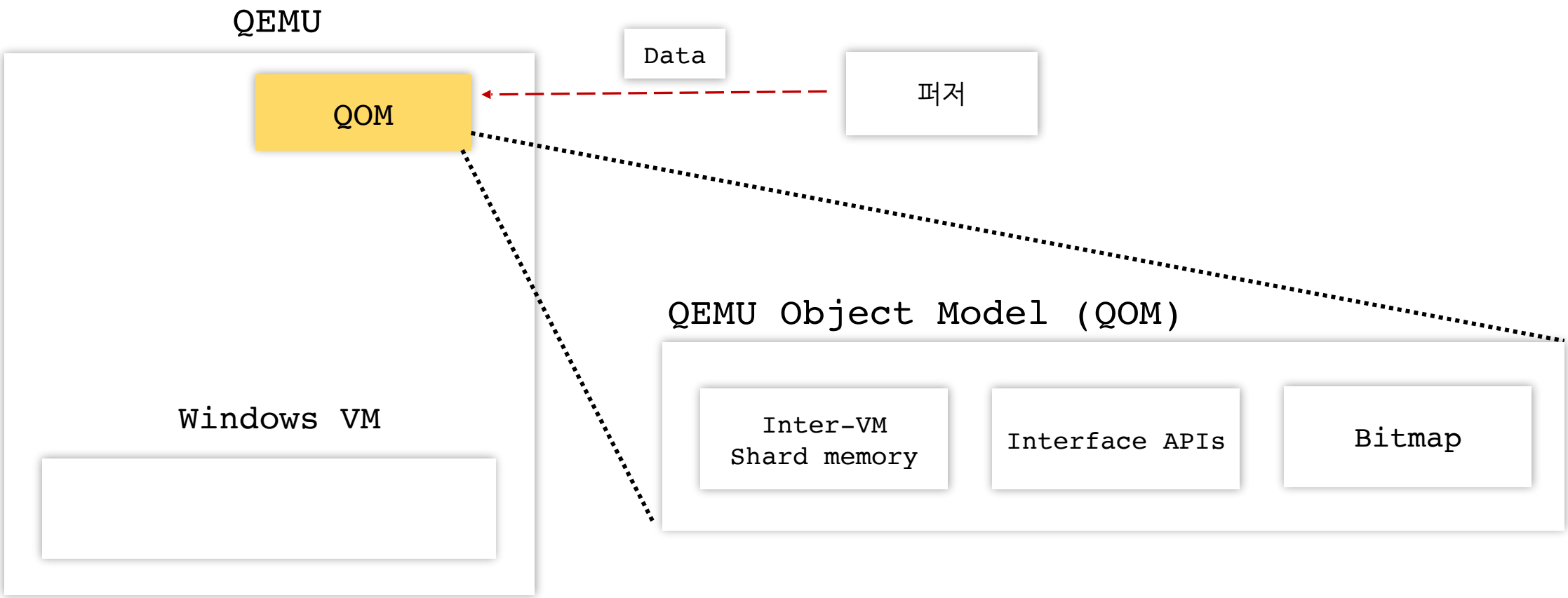
=>

리눅스 호스트에서 QEMU로 윈도우 VM 구동

IRPT

Part 2 - Fuzzing environment

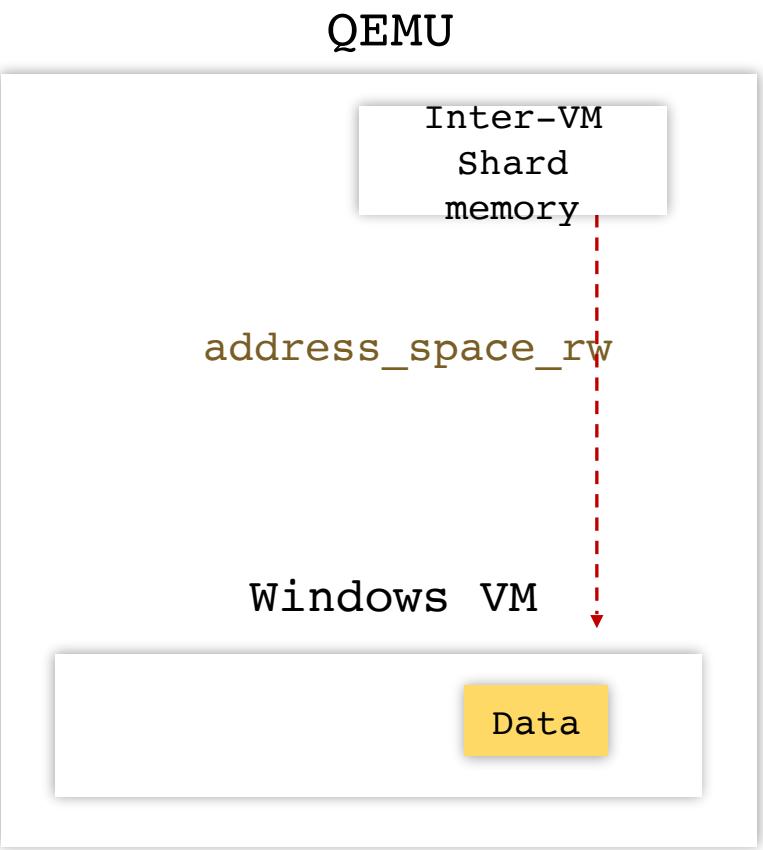
- QOM으로 QEMU와 퍼저가 통신



IRPT

Part 2 - Fuzzing environment

- QEMU API로 윈도우 VM 메모리에 직접 접근



Load and Store APIs

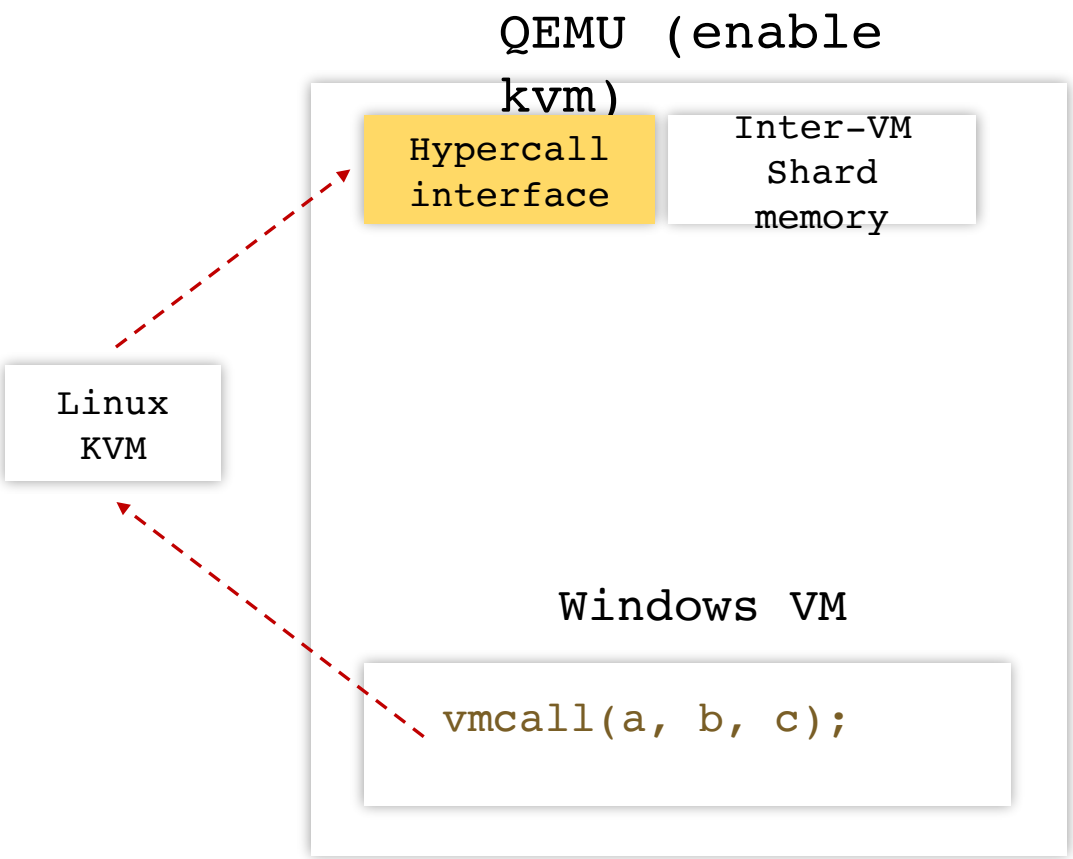
```
address_space_read(address_space, addr, attrs, buf, len)

address_space_write(address_space, addr, attrs, buf, len)

address_space_rw(address_space, addr, attrs, buf, len, is_write)

address_space_ld{sign}{size}_{endian}(address_space, addr, attrs, txresult)

address_space_st{size}_{endian}(address_space, addr, val, attrs, txresult)
```

=> 오버헤드 최소화

IRPT

Part 2 - Fuzzing environment

- 윈도우 VM에서 QEMU로 보내는 데이터는 하이퍼 콜(hypercall)을 이용



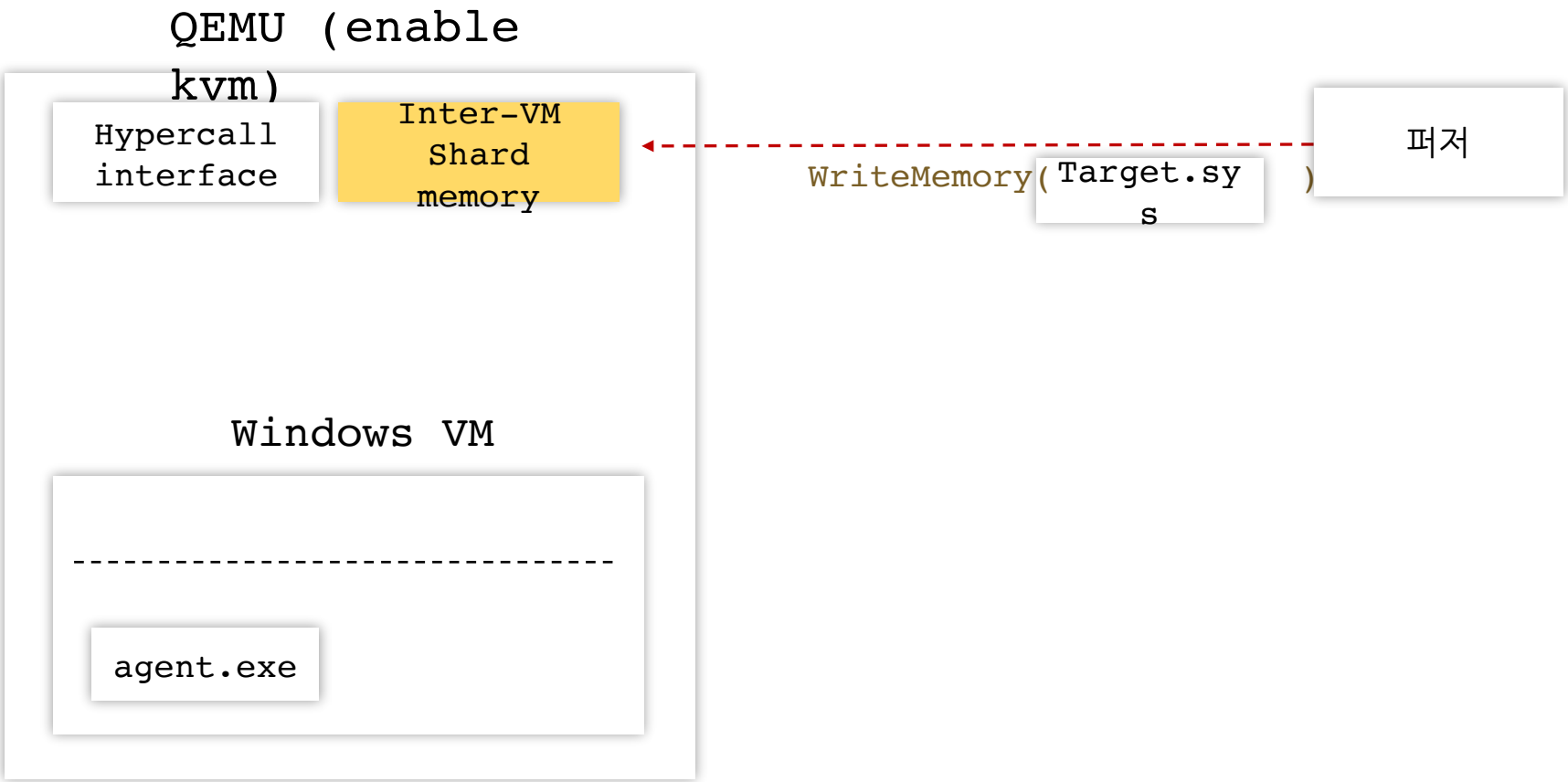
=>

1. QEMU을 kvm 모드로 구동
2. vmcall 명령어로 전달한 인자는 호스트 머신으로 바로 이동
3. 인자가 주솟값이면 address_rw_space로 데이터 읽기

IRPT

Part 2 - Fuzzing environment

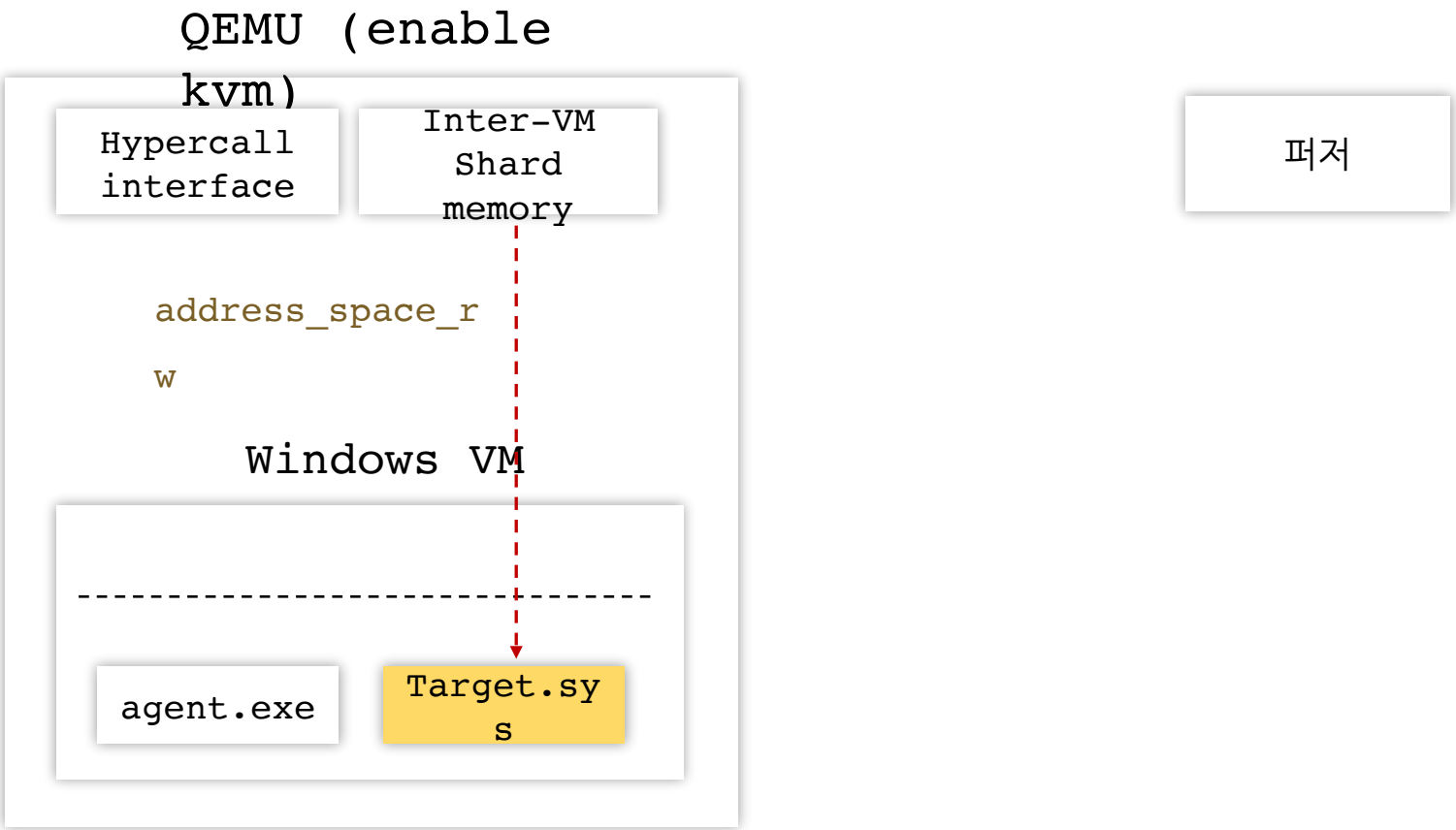
1. 퍼징할 드라이버를 공유 메모리로 전송



IRPT

Part 2 - Fuzzing environment

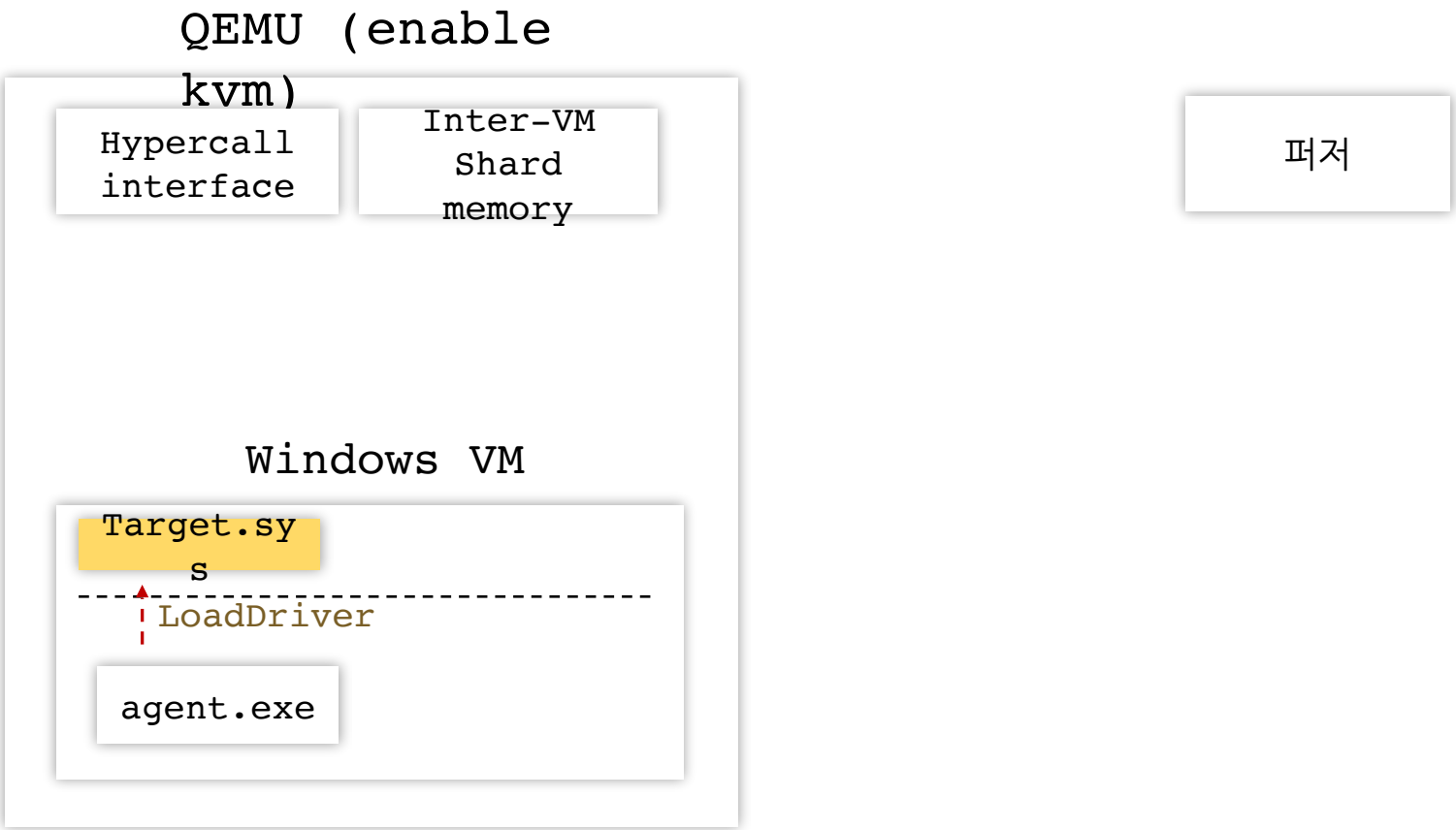
2. address_space_rw 함수로 윈도우 VM 안으로 삽입



IRPT

Part 2 - Fuzzing environment

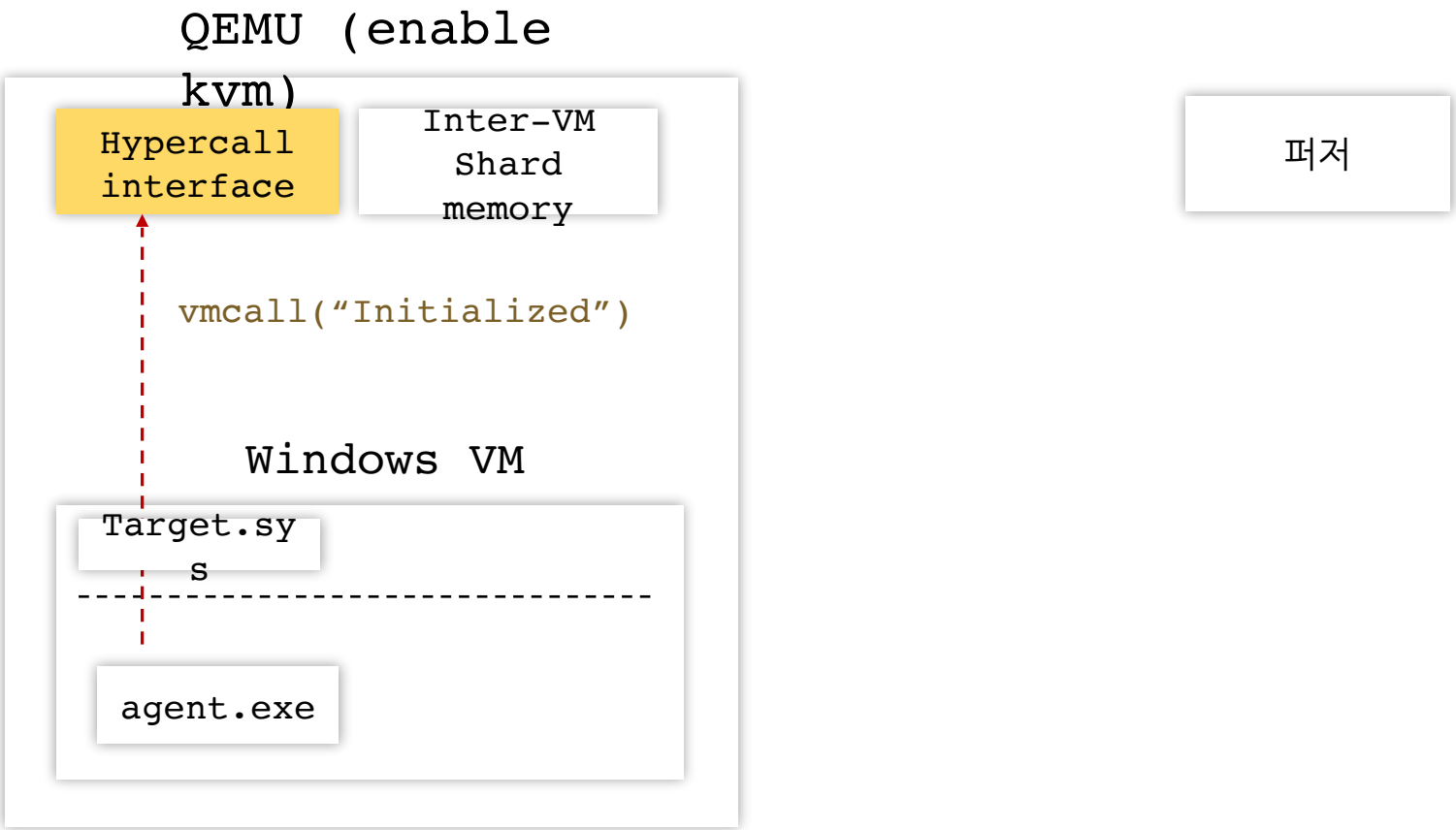
3. Agent.exe에서 커널 드라이버를 로드



IRPT

Part 2 - Fuzzing environment

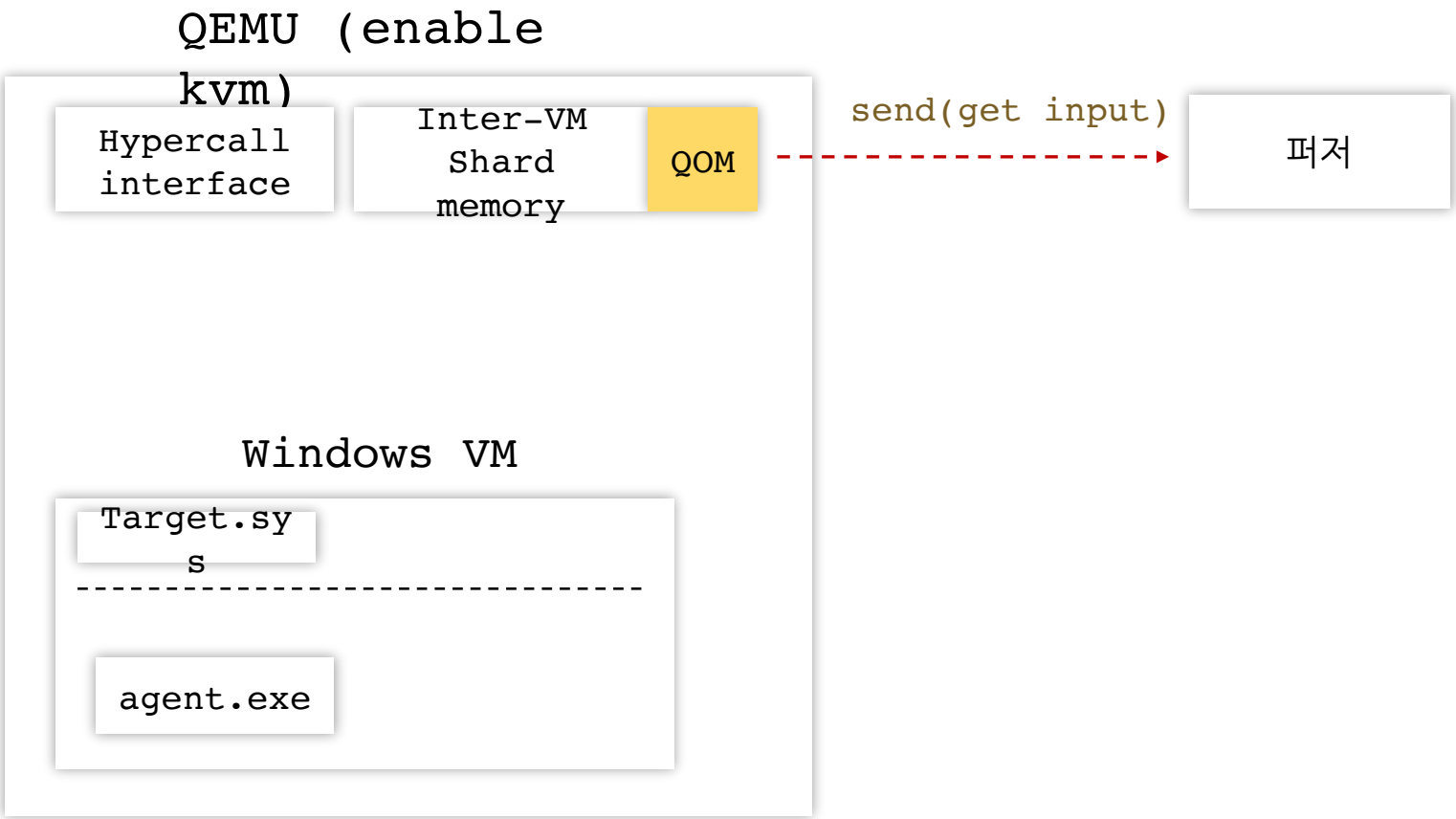
4. 드라이버 세팅이 완료됐음을 hypercall로 알림



IRPT

Part 2 - Fuzzing environment

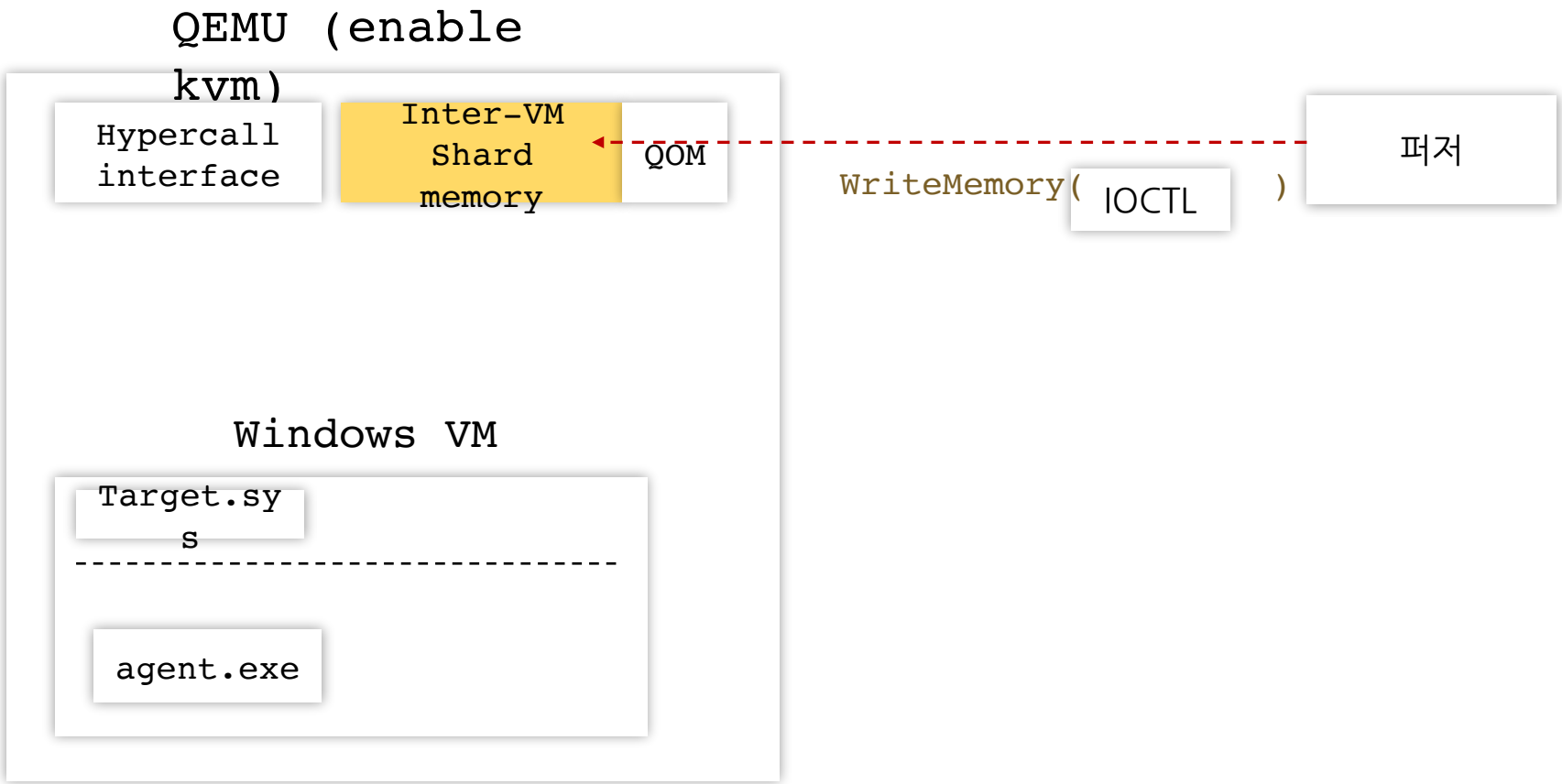
5. chardev로 생성한 소켓 인터페이스가 퍼저에 Input을 요청



IRPT

Part 2 - Fuzzing environment

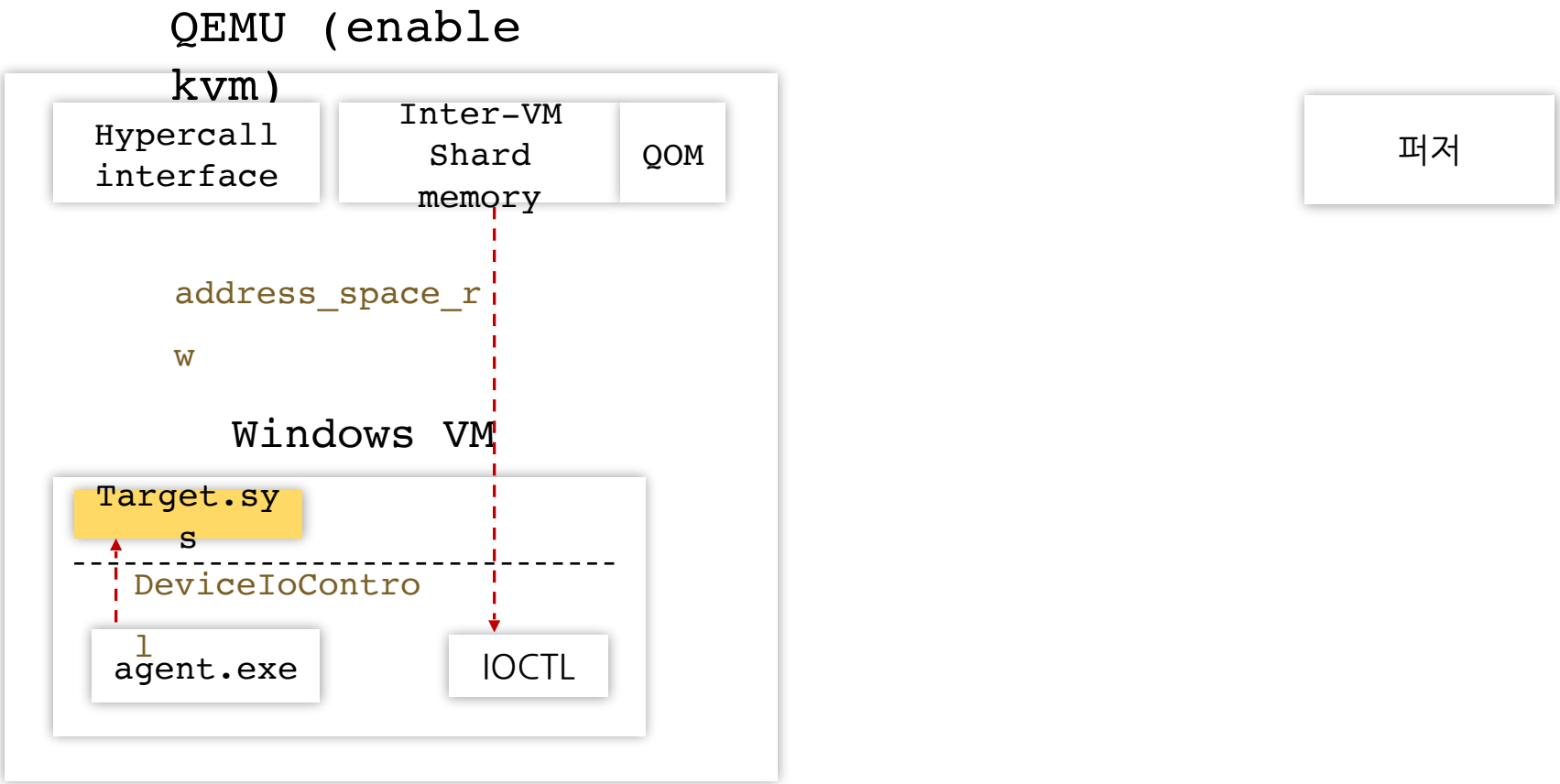
6. 드라이버 대신 IOCTL을 직렬화한 파일을 보냄



IRPT

Part 2 - Fuzzing environment

7. agent.exe는 전달받은 IOCTL 파일을 파싱해서 타겟 드라이버로 요청



IRPT

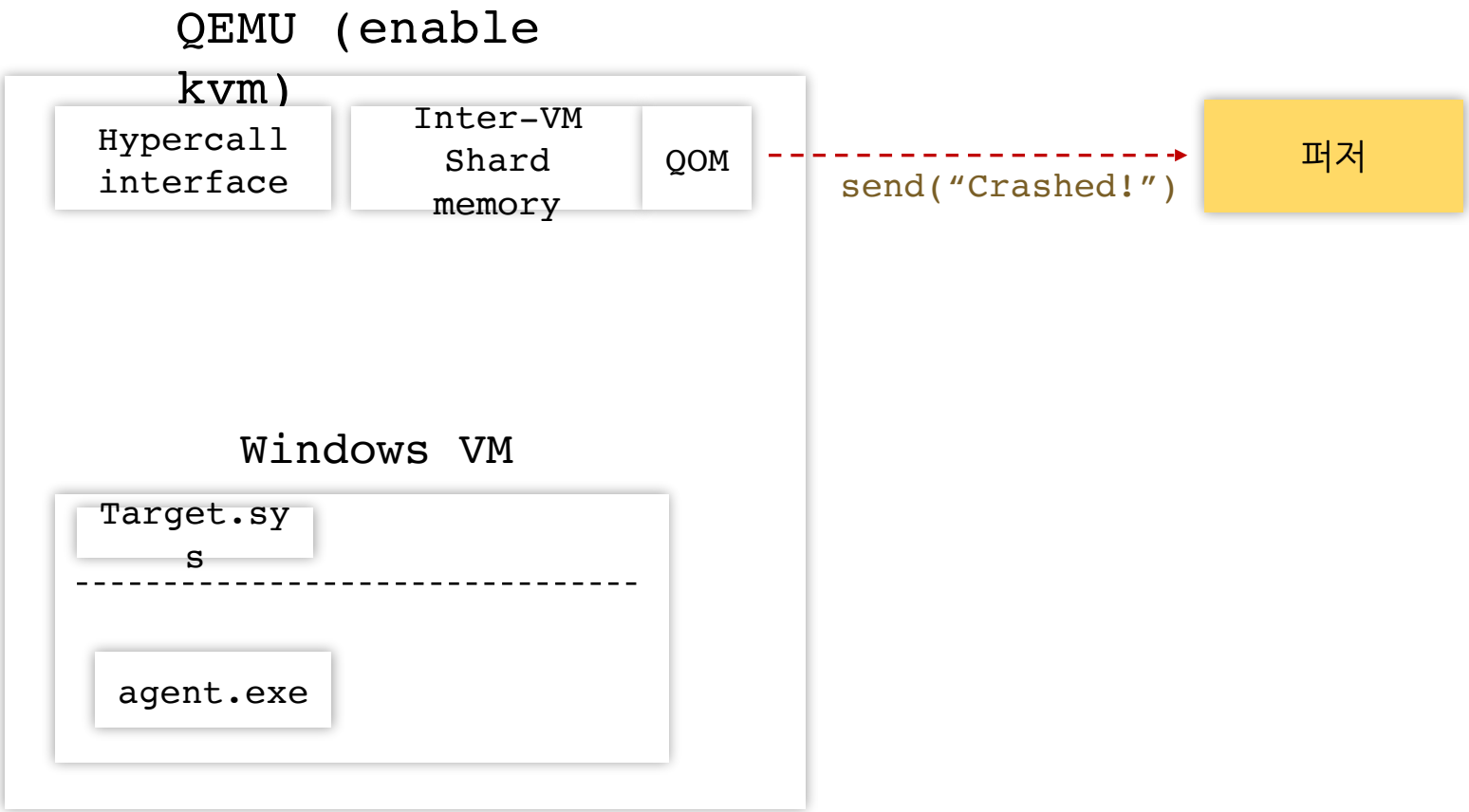
Part 2 - Fuzzing environment

8. 크래시가 발생했는지 여부를 하이퍼 콜로 전송 (크래시 판별은 뒤에서 자세히 다룸)

IRPT

Part 2 - Fuzzing environment

9. 크래시가 발생했으면 해당 인풋을 저장하고 다시 반복



IRPT

Part 3 - Coverage feedback fuzzing

- Instrumentation을 위해 고려할 점

1. 컴파일 타임 불가 (Closed source)
2. 부작용이 조금이라도 없어야 함 (커널 드라이버)

⇒ **Intel PT(Processor Tracing)**를 이용한 Hardware 기반 instrumentation

1. 드라이버 바이너리에 어떠한 수정이 필요 X
2. kAFL에서 많은 부분을 가져올 수 있음
3. 오버헤드도 비교적 적음

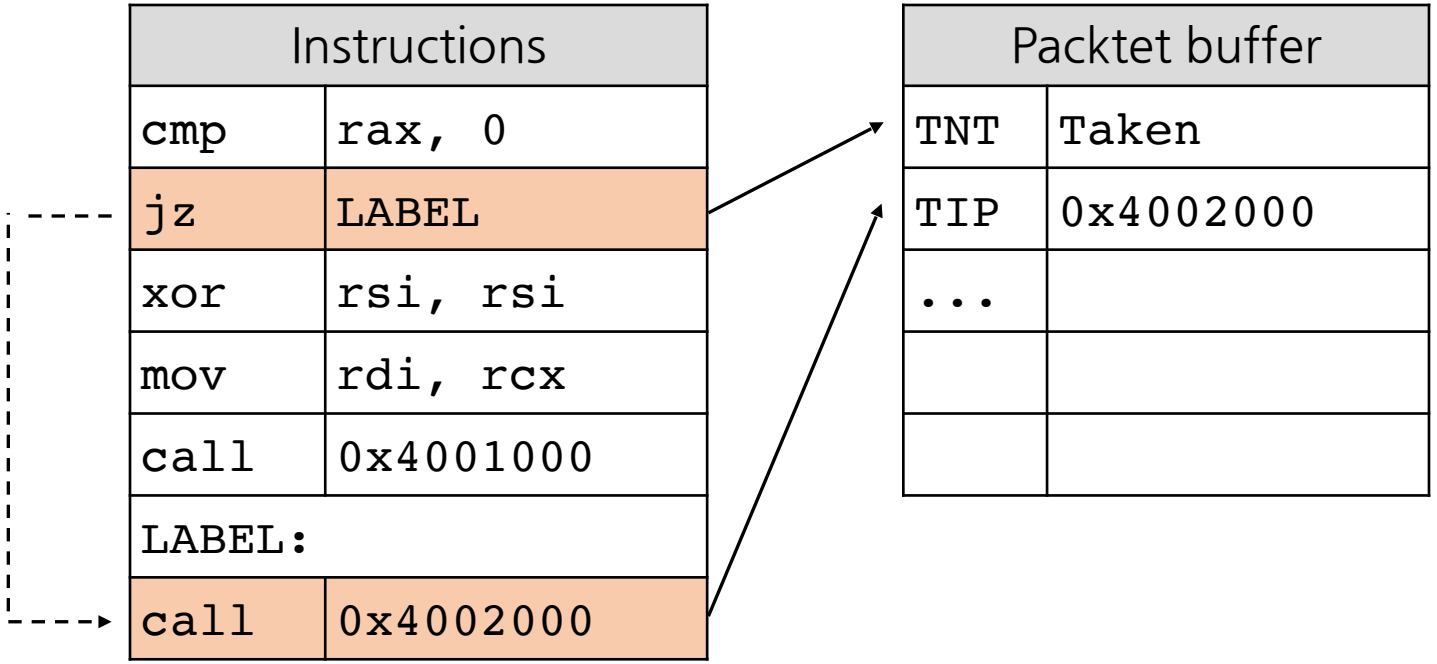
IRPT

Part 3 - Coverage feedback fuzzing

- Intel PT란?

1. 인텔 CPU 5세대 이후부터 가능해진 실행 흐름 추적 기능

2. CPU가 jmp, call, ret 등 실행 흐름이 바뀌는 명령어가 실행되면 패킷 발생



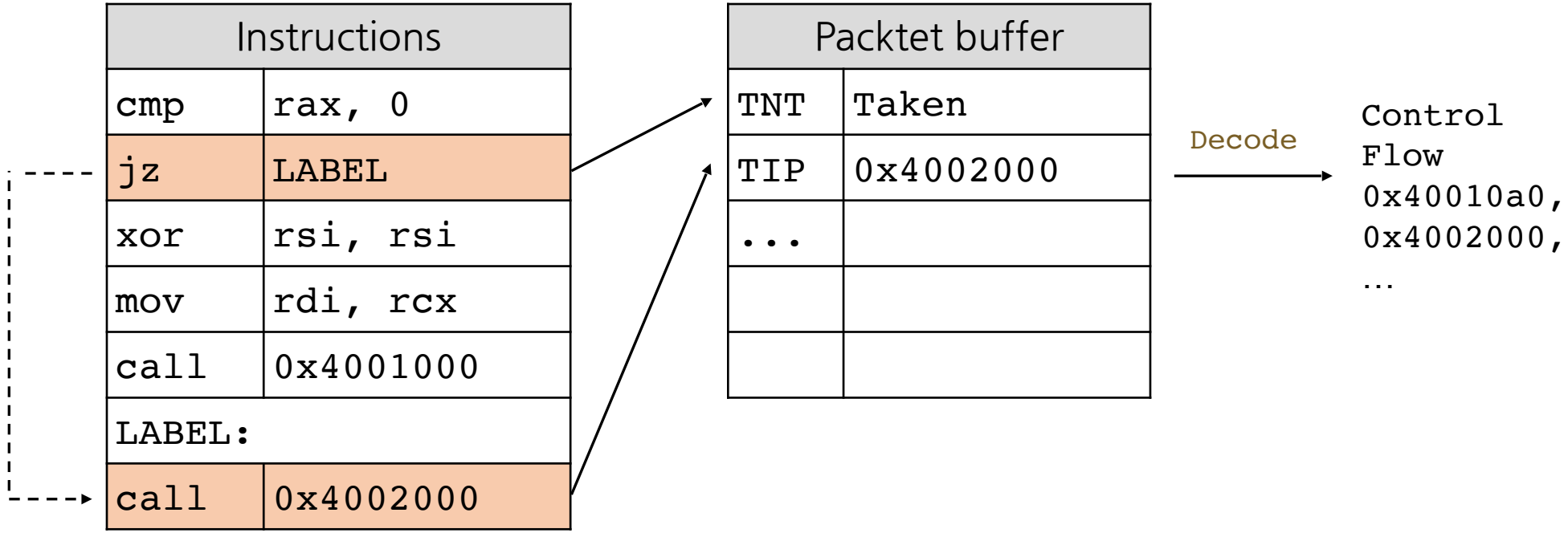
IRPT

Part 3 - Coverage feedback fuzzing

- Intel PT란?

1. 인텔 CPU 5세대 이후부터 가능해진 실행 흐름 추적 기능

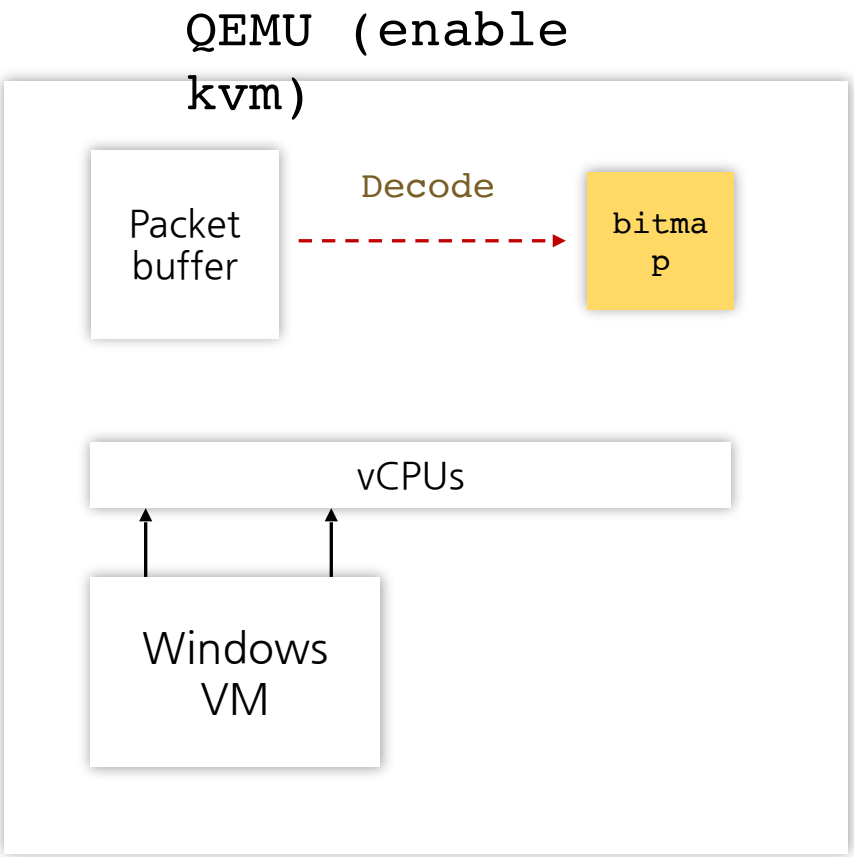
2. CPU가 jmp, call, ret 등 실행 흐름이 바뀌는 명령어가 실행되면 패킷 발생



IRPT

Part 3 - Coverage feedback fuzzing

- 윈도우 커널 Coverage를 측정하기 위해 Intel PT 이용



Packet buffer	
TNT	Taken
TIP	0x4002000
TIP	0x4002030
TIP	0x4002048
...	

Decode

AFL's bitmap[]

=>

Intel PT를 적용한 vCPU가 커널 코드를 실행하면

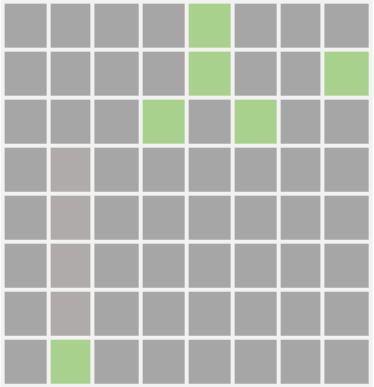
패킷 버퍼에 Trace 패킷 데이터가 저장

IRPT

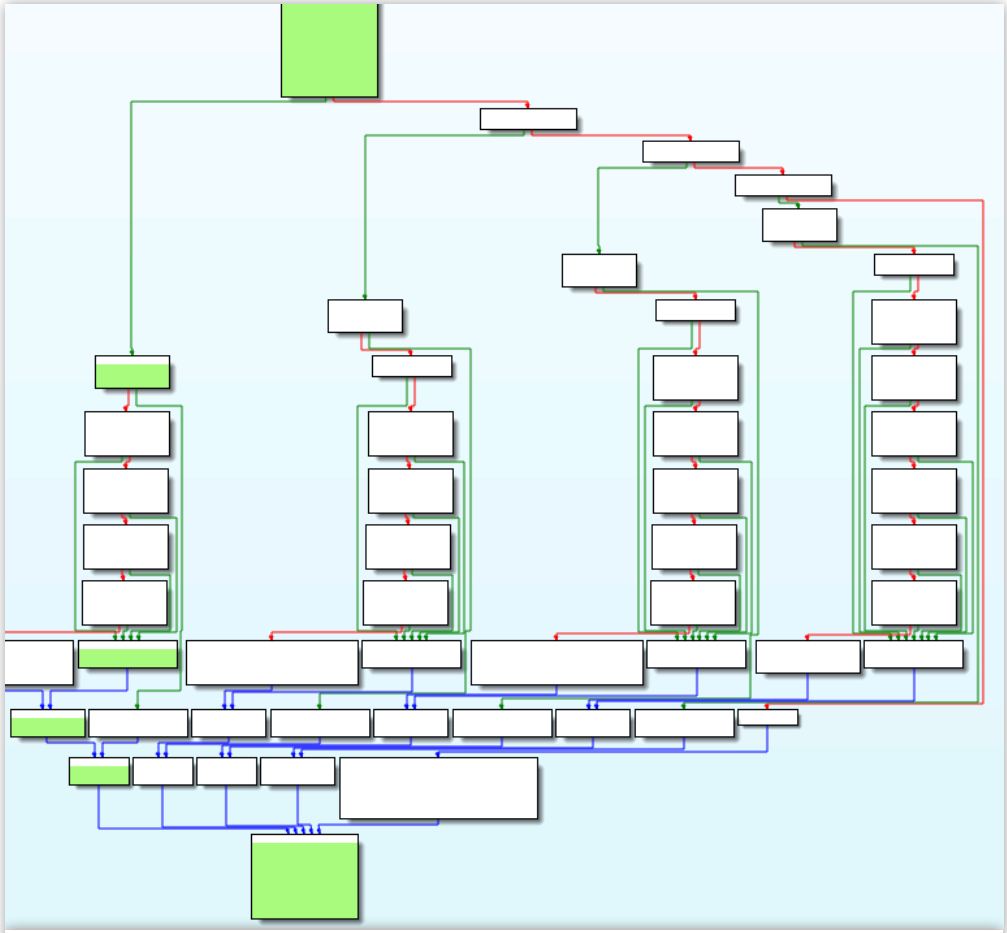
Part 3 - Coverage feedback fuzzing

- AFL과 비슷한 방식으로 Coverage를 넓혀감

1. **비트맵에 변화가 있는지 확인**



AFL's bitmap[]



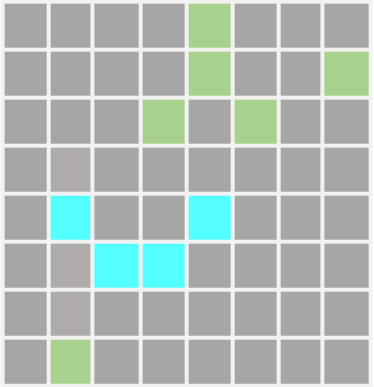
드라이버 Code coverage

IRPT

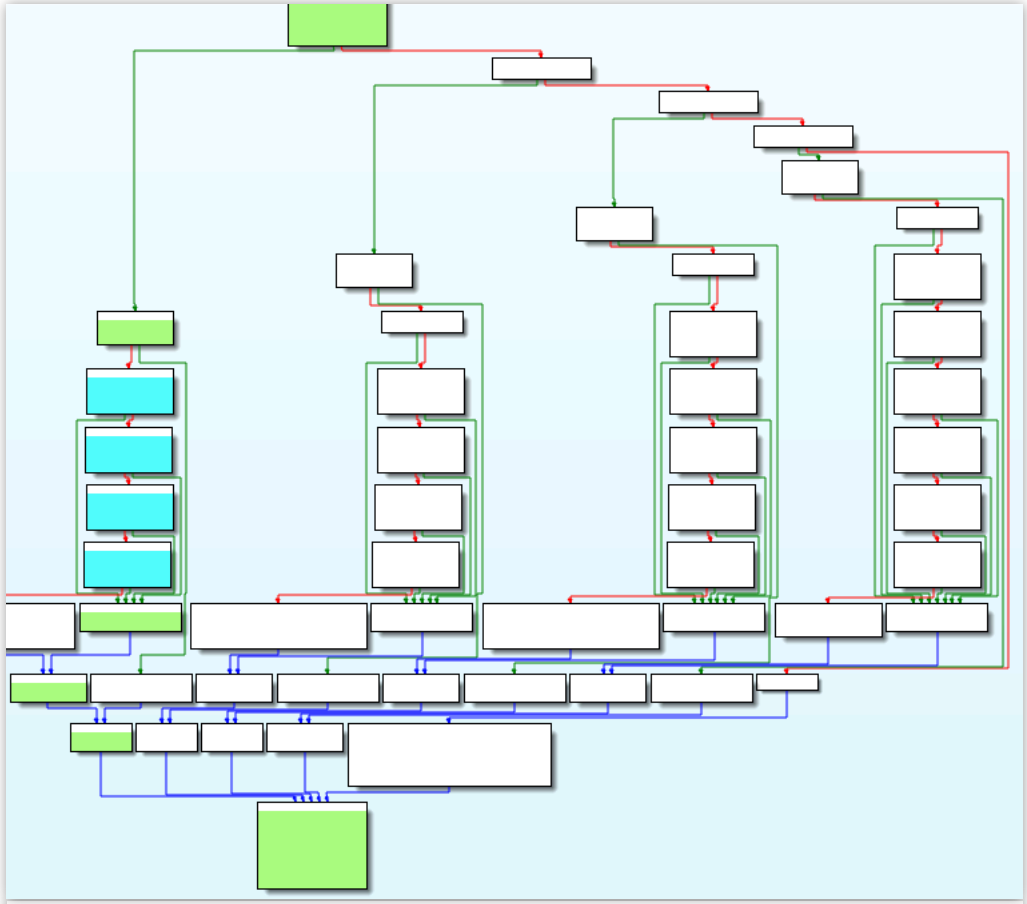
Part 3 - Coverage feedback fuzzing

- AFL과 비슷한 방식으로 Coverage를 넓혀감

1. 비트맵에 변화가 있는지 확인
2. **새로운 코드를 실행한 input 발견**



AFL's bitmap[]



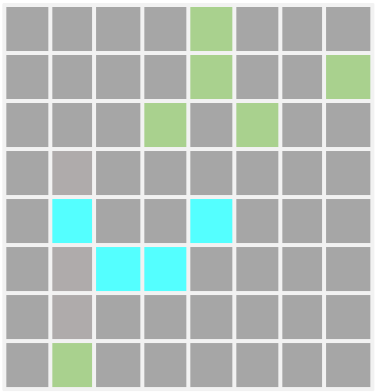
드라이버 Code coverage

IRPT

Part 3 - Coverage feedback fuzzing

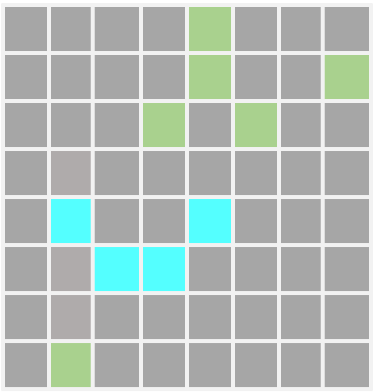
- AFL과 비슷한 방식으로 Coverage를 넓혀감

1. 비트맵에 변화가 있는지 확인
2. 새로운 코드를 실행한 Input 발견
3. **Input 검증 & 최적화**



AFL's bitmap[]

==



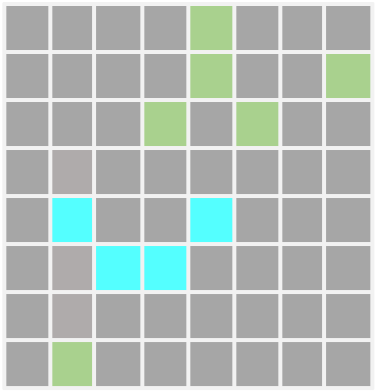
AFL's bitmap2[]

IRPT

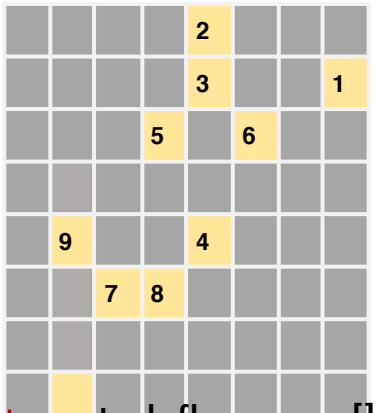
Part 3 - Coverage feedback fuzzing

- AFL과 비슷한 방식으로 Coverage를 넓혀감

1. 비트맵에 변화가 있는지 확인
2. 새로운 코드를 실행한 Input 발견
3. Input 검증 & 최적화
4. **Control Flow 복구**



AFL's bitmap[]



control_flow_map[]

```
Control Flow:

0x140005A0 -> 0x140000600 -> …
```

IRPT

Part 3 - Coverage feedback fuzzing

- AFL과 비슷한 방식으로 Coverage를 넓혀감

1. 비트맵에 변화가 있는지 확인
2. 새로운 코드를 실행한 Input 발견
3. Input 검증 & 최적화
4. Control Flow 복구
5. **Coverage가 가장 넓은 Input에 우선순위 적용**

```
Input 1:

A -> B -> C                    (Score :
20)


Input 2:
```

Corpus 데이터베이스

60)

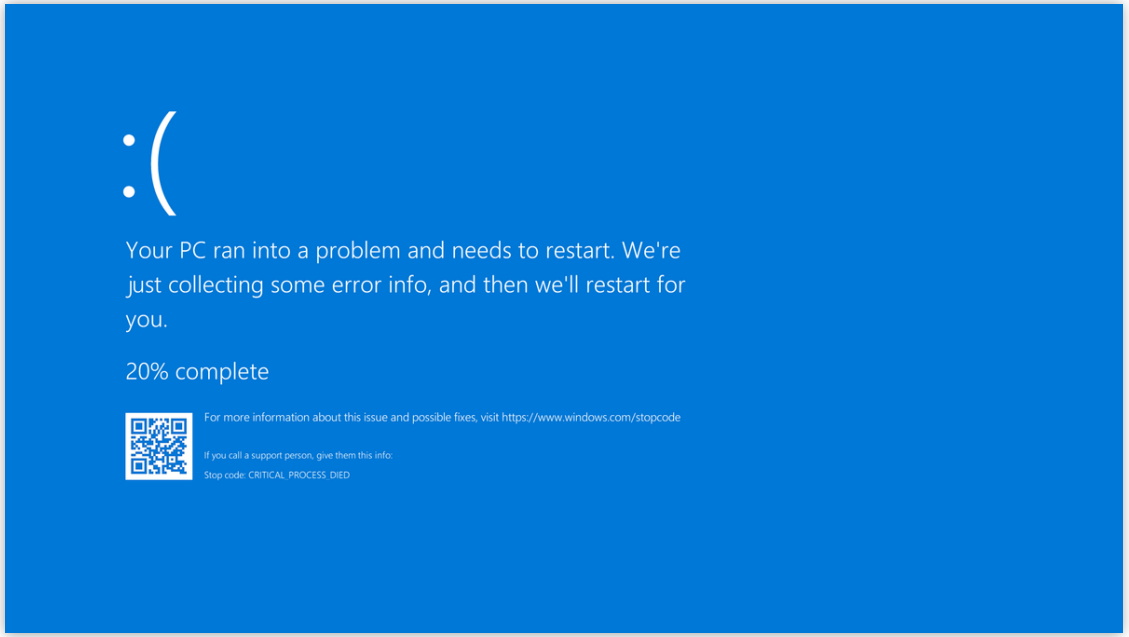
Input 2가 가장 우선적으로 퍼징할 Input으로 선정

IRPT

Part 4 - Crash detection

- 퍼저는 크래시가 발생하면 정확히 잡아낼 수 있어야 함

=> 커널 크래시는 BSOD로 이어짐



IRPT

Part 4 - Crash detection

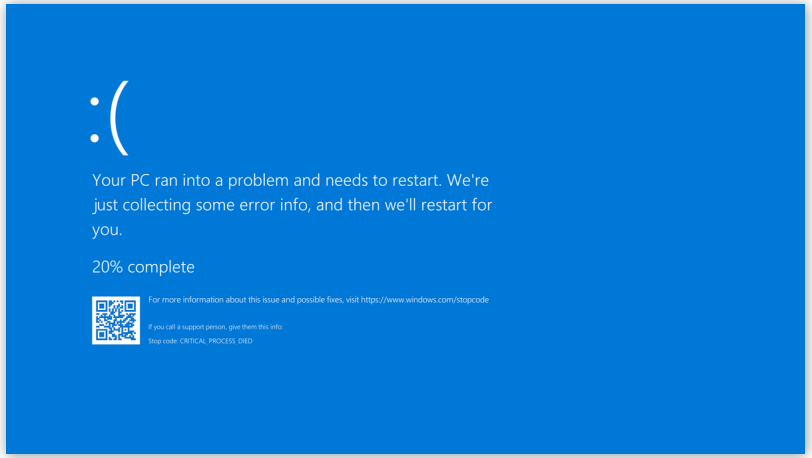
- BSOD 호출 과정

crashed!

↓

call KeBugCheckEx

↓



KeBugCheckEx function (wdm.h)

04/30/2018 • 2 minutes to read

The **KeBugCheckEx** routine brings down the system in a controlled manner when the caller discovers an unrecoverable inconsistency that would corrupt the system if the caller continued to run.

=>

블루스크린이 뜨기 전에 KeBugCheckEx 함수를 호출

IRPT

Part 4 - Crash detection

- KeBugCheckEx 후킹

crashed!

↓

```
KeBugCheckEx:
vmcall(crashed)
...
```

=>

address_space_rw 함수로 KeBugCheckEx 함수가 하이퍼 콜을 호출하도록 패치.

IRPT

Part 4 - Crash de-duplication

- 크래시가 발생한 Input은 실행 경로를 바탕으로 중복 제거

Unique crash

IOCTL 1	A->B->C->D
IOCTL 2	A->B->D
IOCTL 3	B->F->D

Duplicated crash

IOCTL 4	A->B->C->D

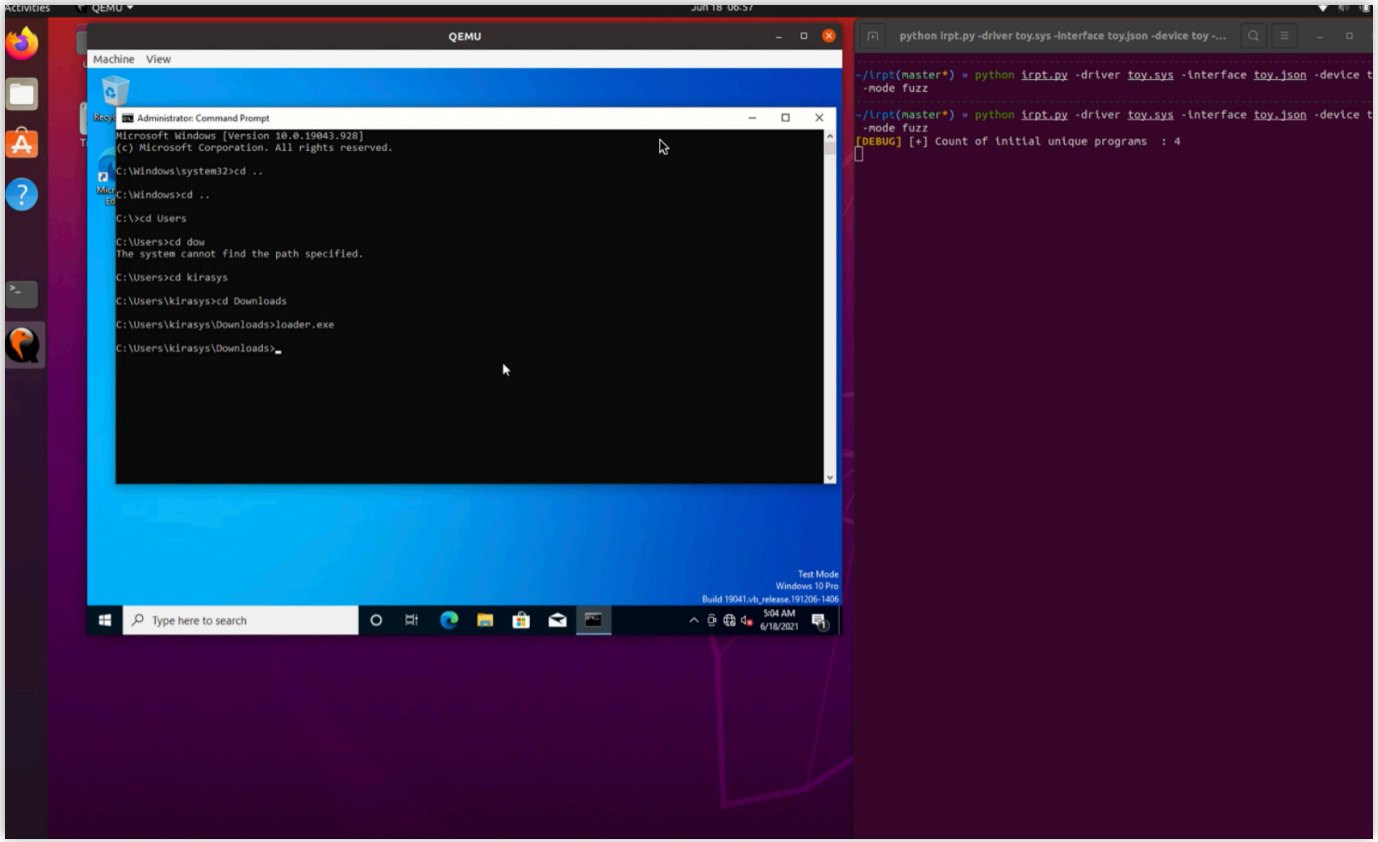
=>

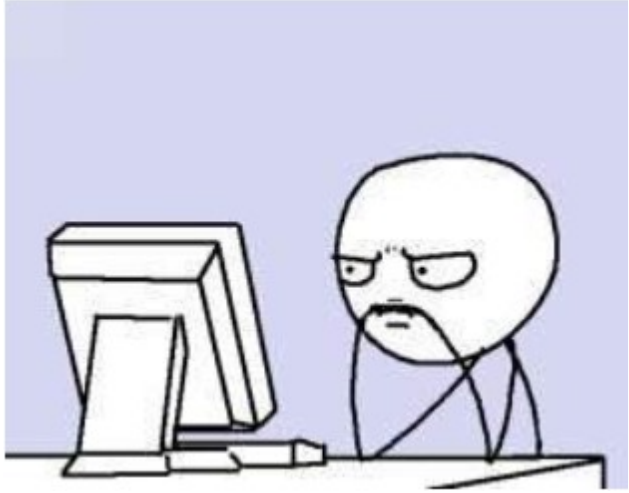
발견된 크래시의 실행 경로는 이미 데이터베이스에 등록된 경로와 달라야 함.

IRPT

Complication 1 - Dependency problem

- 퍼징은 잘 돌아가지만 가끔 크래시가 발생해도 재현이 안되는 이슈 발생





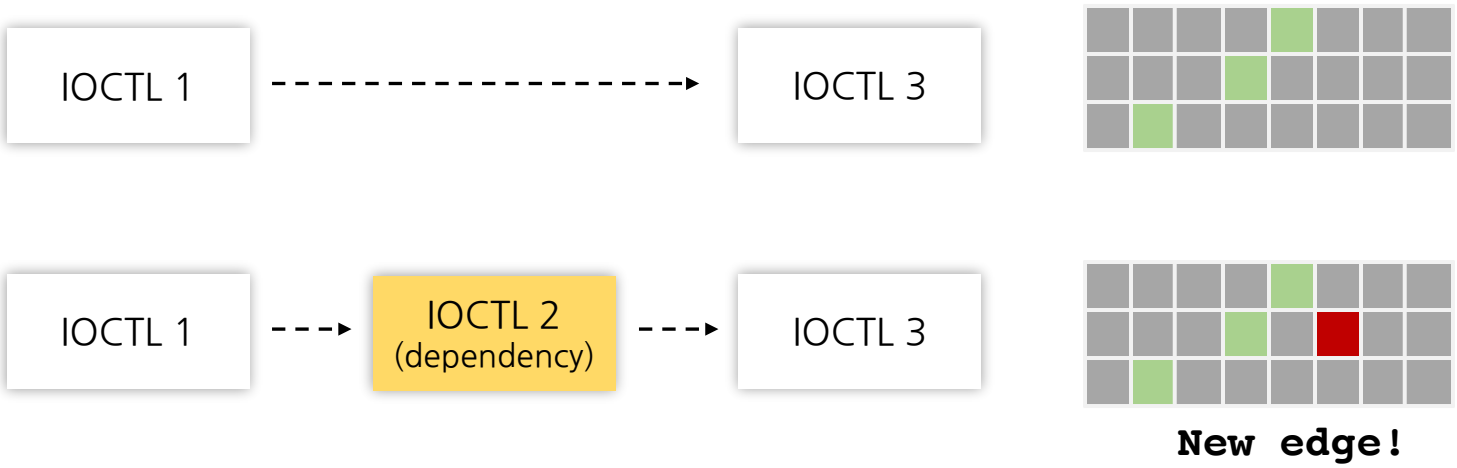
IRPT

Complication 1 - Dependency problem

- 전역 변수 등으로 IOCTL 요청 순서에 따라 커버리지가 늘어날 수 있음

```
133      case 4:
134        if ( !AhcEnabled )
135          goto LABEL_42;
136        ++GlobalData0;
137        v14 = AhcApiSnapStatistics(&AhcS
138        break;
```

전역 변수 사용으로 의존성 발생



IRPT

Complication 1 - Dependency problem

- 뮤테이션 단위를 재정의

IRP Group

1
```
DeviceIoControl(0xA0001, “aaa”, 4);
```

2
```
DeviceIoControl(0xA0002, “bbbb”, 4);
```

3
```
DeviceIoControl(0xA0003, “cccc”, 4);
```

=>

여러 개의 IOCTL 요청을 하나의 구조체로 관리

IRPT

Complication 1 - Dependency problem

- IOCTL 순서 등의 구조를 변경하는 뮤테이션 기법 추가

IRP Group

```
DeviceIoControl(0x001, “AAAA”, 4);
```

```
DeviceIoControl(0x002, “BBBB”, 4);
```

```
DeviceIoControl(0x004, “DDDD”, 4);
```

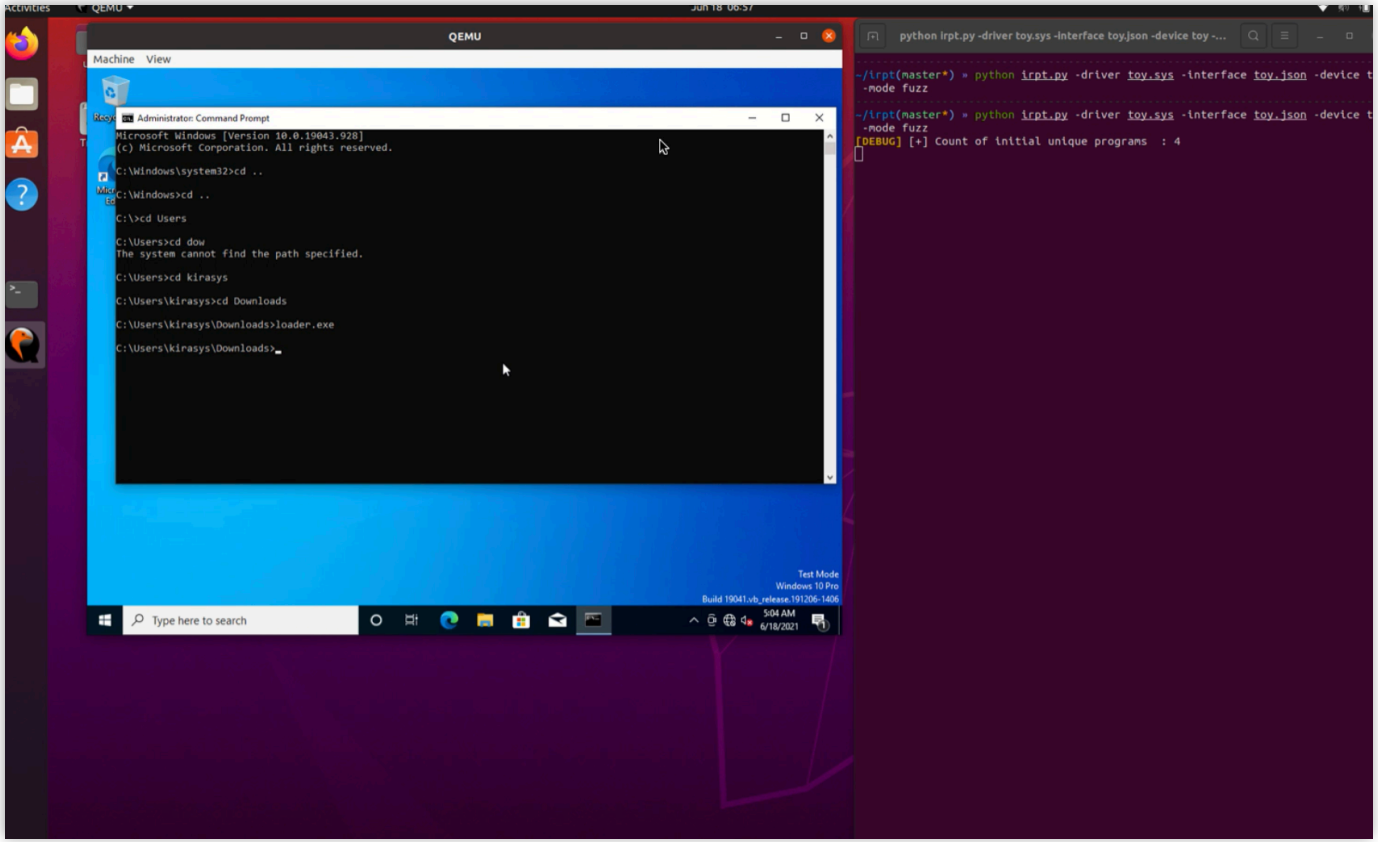
```python
# wdm/program.py
class Program:
    ...
    def mutate(self, corpus_programs):
      ...
      if rand.oneOf(5) and self.__splice(corpus_programs):
          method = "splice"
      elif rand.oneOf(10) and self.__swapIRP():
          method = "swapIRP"
      elif rand.oneOf(10) and self.__insertIRP(corpus_programs):
          method = "insertIRP"
      elif rand.oneOf(20) and self.__removeIRP():
          method = "removeIRP"

      self.set_state(method)
      return method
```

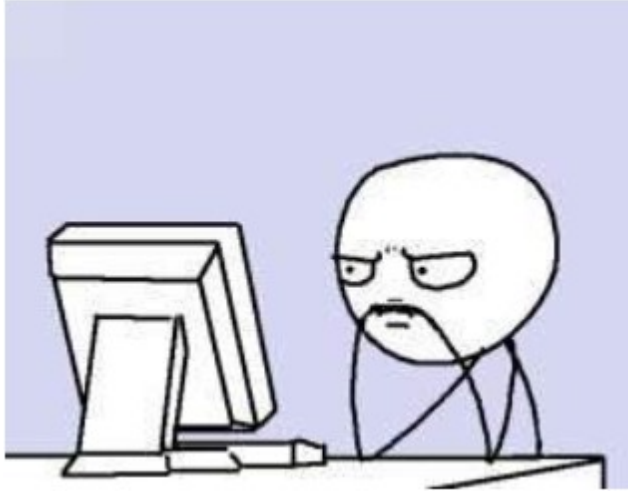
다양한 뮤테이션 기법 적용

IRPT

Complication 2 - Anti fuzzing

- 몇몇 프로그램에선 퍼징 자체가 안되는 이슈 발생





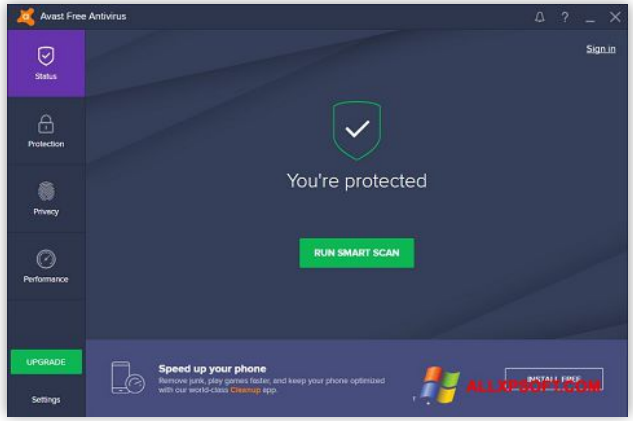
IRPT

Complication 2 - Anti fuzzing

- 일부 드라이버의 퍼징은 실제 프로그램이 실행되고 있는 상태에서만 가능

=> 프로그램과 데이터를 주고 받으면서 초기화 작업을 하기 때문

avast.sys



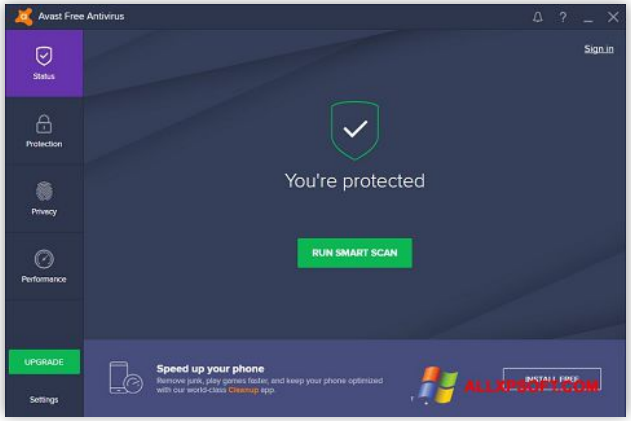
avast

IRPT

Complication 2 - Anti fuzzing

- 일부 드라이버의 퍼징은 실제 프로그램이 실행되고 있는 상태에서만 가능

=> 프로그램과 데이터를 주고 받으면서 초기화 작업을 하기 때문



avast

avast.sys

```
149     case 0x222813:
150       v25 = v4->Parameters.SetFile.DeleteHandle;
151       if ( v25 && (_DWORD)v5 && !v6 )
152       {
153         if ( ExGetPreviousMode() == 1 )
154         {
155           ProbeForRead(v25, v5, 1u);
156           if ( (unsigned int)PsGetCurrentProcessId(v27, v26) != PID
157           {
```

=> PID 검사로 에이전트의 요청을 차단

agent.exe

IRPT

Complication 2 - Anti fuzzing

- 간단하게 드라이버 및 커널 코드 패치를 할 수 있는 API 구현

```
1: kd> u PsGetCurrentProcessId
nt!PsGetCurrentProcessId:
fffff803`34a1ec20 65488b042588010000 mov   rax,qword ptr gs:[188h]
fffff803`34a1ec29 488b8078040000  mov     rax,qword ptr [rax+478h]
fffff803`34a1ec30 3d61616161      cmp     eax,61616161h
fffff803`34a1ec35 7503            jne     nt!PsGetCurrentProcessId+0x1a (fffff803`34a1ec3a)
fffff803`34a1ec37 4831c0          xor     rax,rax
fffff803`34a1ec3a c3              ret
fffff803`34a1ec3b cc              int     3
fffff803`34a1ec3c cc              int     3
```

Insert shellcode

PsGetCurrentProcessId 커널 함수 패치

=> 커널 함수 자체를 변경했을 때는 부작용에 주의해야 함.

IRPT

Demo

- 예제 타겟 드라이버의 크래시 발동 조건

Crash Inputs

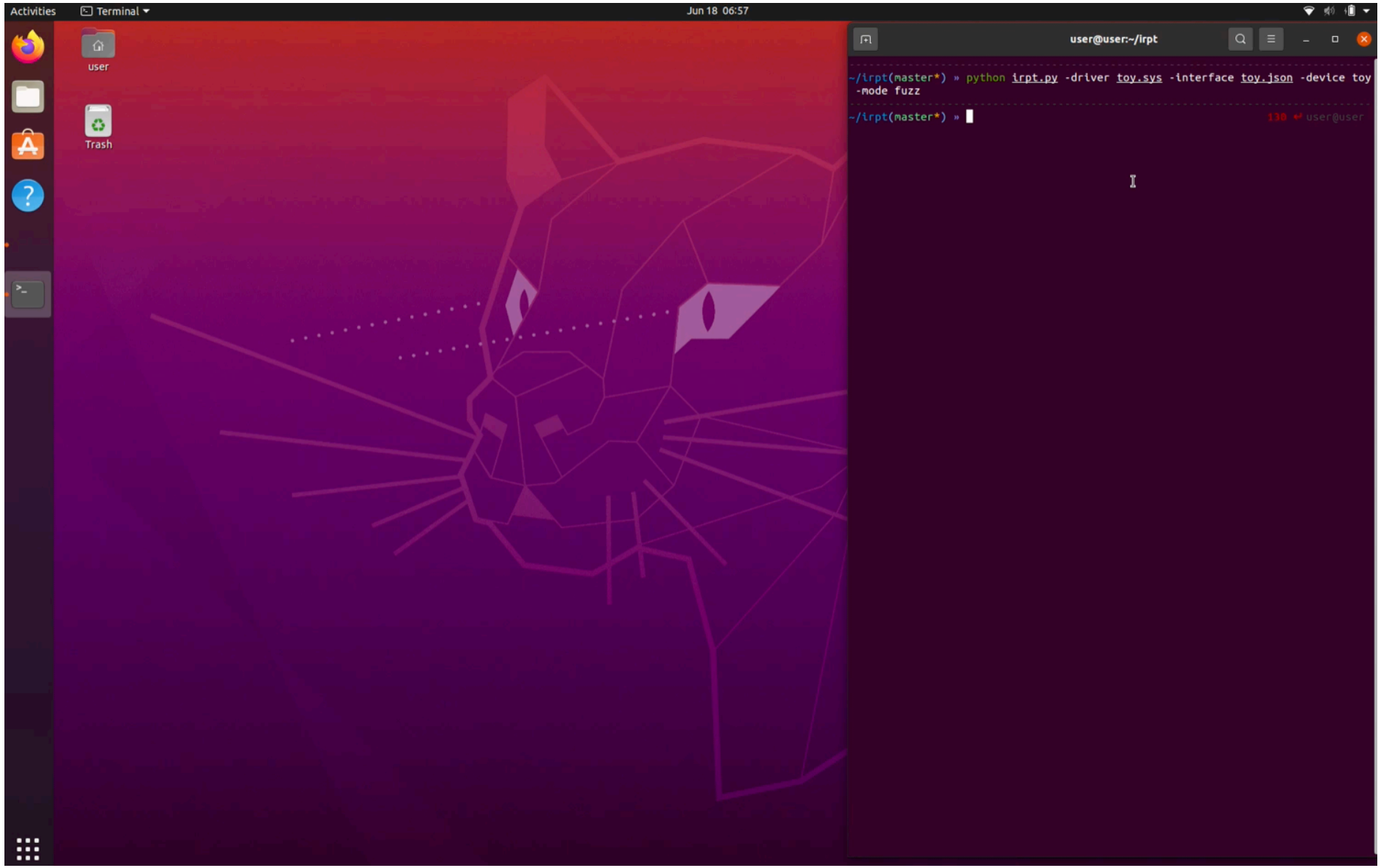
1. `DeviceIoControl(0x9C402000, “IREC”, 4);`
2. `DeviceIoControl(0x9C402004, “IRPT”, 4);`
3. `DeviceIoControl(0x9C402008, “IRCAP”, 5);`
4. `DeviceIoControl(0x9C40200C, “KRONOL”, 6);`

=>

해당 문자열과 IOCTL 순서가 정확히 일치해야 크래시가 터짐

IRPT

Demo


Activities
Terminal
Jun 18 06:57
user
Trash
user@user:~/irpt
~/irpt(master*) » python irpt.py -driver toy.sys -interface toy.json -device toy
-mode fuzz
~/irpt(master*) »
130 ↵ user@user

결론

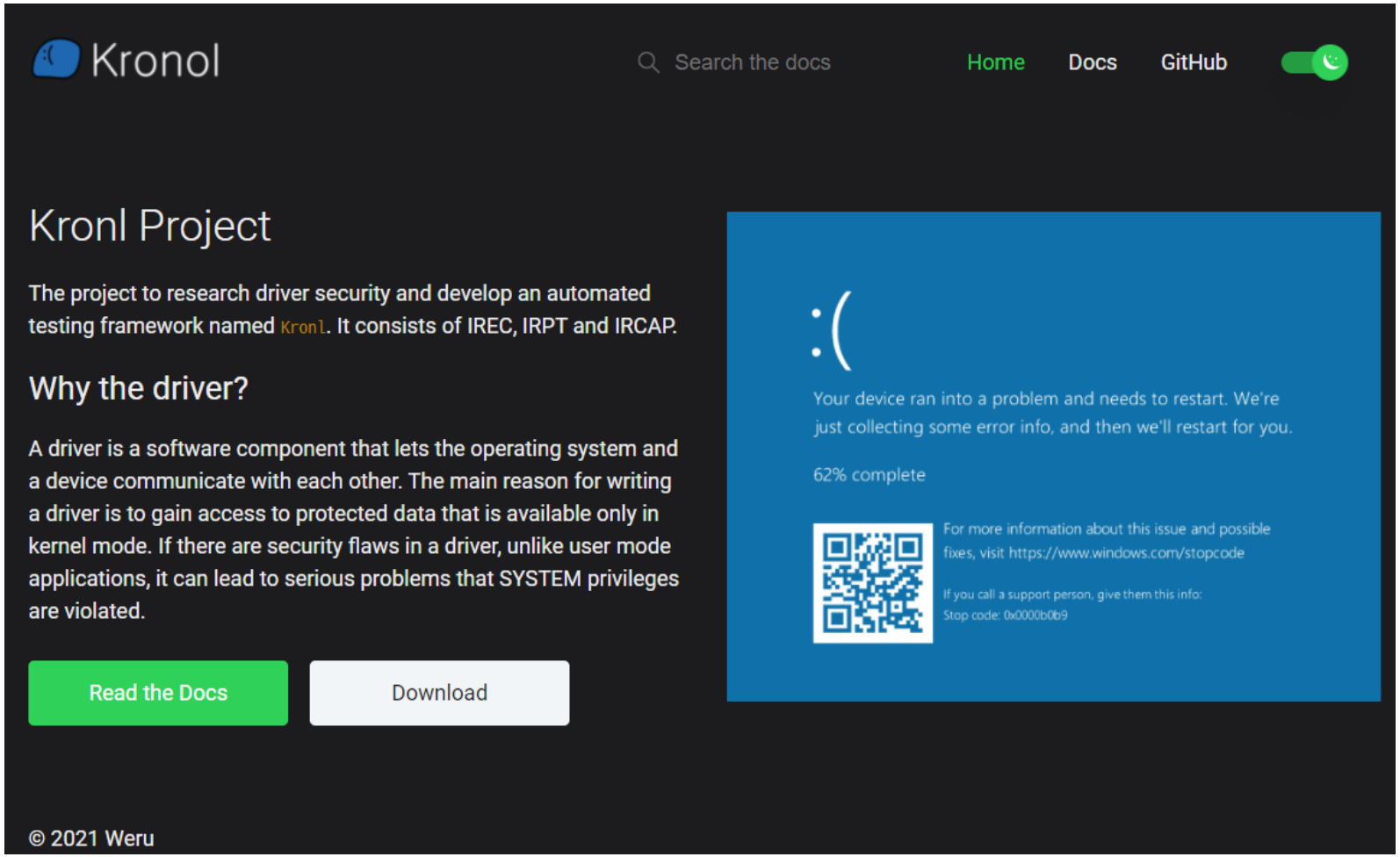
결론

발견한 취약점

Software	Bug Class	Status
AhnLab Safe Transaction	Null Pointer Dereference	KVE-2020-0883
TFG Gaming Center	LPE (Kernel Stack BOF)	KVE-2020-1585
TFG Gaming Center	LPE (Kernel Stack BOF)	KVE-2020-1586
Naver eBook Reader	LPE (Kernel Stack BOF)	NBB-1589
Naver eBook Reader	Null Pointer Dereference	NBB-1593
MSI Dragon Center	LPE (Kernel Stack BOF)	CVE-2021-27965
VMware Workstation Pro	DoS (Kernel Stack Corruption)	Patching
VMware Workstation Pro	Null Pointer Dereference	Patching
F-Secure Anti-Virus	Integer Overflow	Reported
Advanced SystemCare Pro	DoS (Out of Bounds Read)	Patching
AMD Crash Defender	Arbitrary Object Access	Patching

결론

Docs



https://kronl.github.io/

감사합니다.