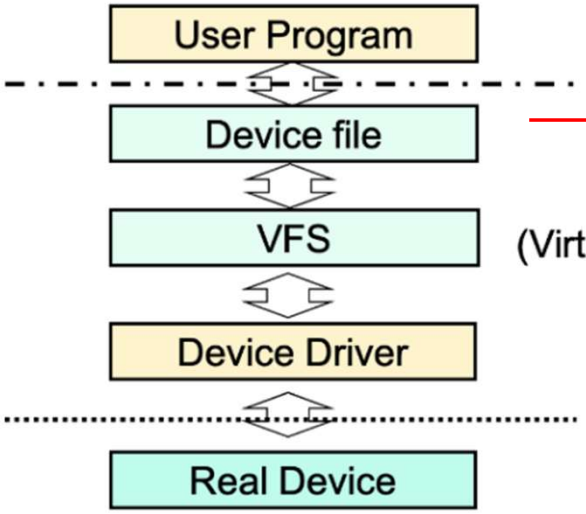
# SYZPARAM: Incorporating Runtime Parameters into Kernel Driver Fuzzing

논문 리뷰 – 발표용

2025/02/06

# background




(Virtual File System)

[system call] open, read, write, ioctl, mmap

ls /dev

```
zp@DESKTOP-VOGAQA9:/sys/module/kernel/parameters$ ls /dev
autofs          fuse            loop0   ptmx        ram5    sg2     tty15  tty29  tty42  tty56  ttyS3    vcsa6
block           hugepages       loop1   ptp0        ram6    sg3     tty16  tty3   tty43  tty57  urandom  vcsu
bsg             hvc0            loop2   ptp_hyperv  ram7    sg4     tty17  tty30  tty44  tty58  vcs      vcsu1
btrfs-control   hvc1            loop3   pts         ram8    shm     tty18  tty31  tty45  tty59  vcs1     vcsu2
bus             hvc2            loop4   ram0        ram9    stderr  tty19  tty32  tty46  tty6   vcs2     vcsu3
char            hvc3            loop5   ram1        random  stdin   tty2   tty33  tty47  tty60  vcs3     vcsu4
console         hvc4            loop6   ram10       rtc     stdout  tty20  tty34  tty48  tty61  vcs4     vcsu5
core            hvc5            loop7   ram11       rtc0    tty     tty21  tty35  tty49  tty62  vcs5     vcsu6
cpu_dma_latency hvc6            mapper  ram12       sda     tty0    tty22  tty36  tty5   tty63  vcs6     vfio
cuse            hvc7            mem     ram13       sdb     tty1    tty23  tty37  tty50  tty7   vcsa     vhost-net
disk            initctl         mqueue  ram14       sdc     tty10   tty24  tty38  tty51  tty8   vcsa1    virtio-ports
dri             kmsg            net     ram15       sdd     tty11   tty25  tty39  tty52  tty9   vcsa2    vport0p0
dxg             kvm             null    ram2        sde     tty12   tty26  tty4   tty53  ttyS0  vcsa3    vport0p1
fd              log             nvram   ram3        sg0     tty13   tty27  tty40  tty54  ttyS1  vcsa4    vsock
full            loop-control    ppp     ram4        sg1     tty14   tty28  tty41  tty55  ttyS2  vcsa5    zero
```

[vfs]

# background

```c
#include <stdio.h>
#include <stdlib.h>
#include <fcntl.h>
#include <unistd.h>
#include <sys/ioctl.h>

#define DEVICE "/dev/mydevice"
#define IOCTL_CMD 0x1234  // 예제용 ioctl 명령

int main() {
    int fd;
    char buf[100] = {0};

    // 파일 열기
    fd = open(DEVICE, O_RDWR);
    if (fd < 0) {
        perror("open");
        return 1;
    }

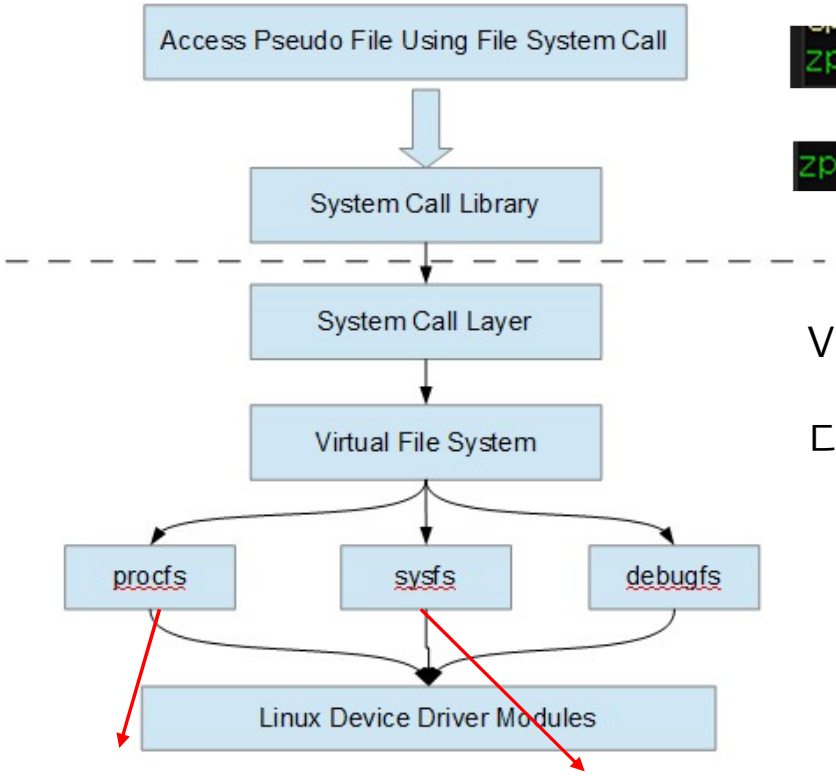
    // ioctl 호출
    if (ioctl(fd, IOCTL_CMD, NULL) < 0) {
        perror("ioctl");
        close(fd);
        return 1;
    }

    close(fd);
    return 0;
}
```

```
zp@DESKTOP-VOGAQA9:/dev$ ls
autofs           cuse        hvc1     kvm
block            disk        hvc2     log
bsg              dri         hvc3     loop-control
btrfs-control    dxg         hvc4     loop0
bus              fd          hvc5     loop1
char             full        hvc6     loop2
console          fuse        hvc7     loop3
core             hugepages   initctl  loop4
cpu_dma_latency  hvc0        kmsg     loop5
zp@DESKTOP-VOGAQA9:/dev$
```

system call

# background




cpu, mem, process 관련
/proc

Kobject, hardware 관련
/sys

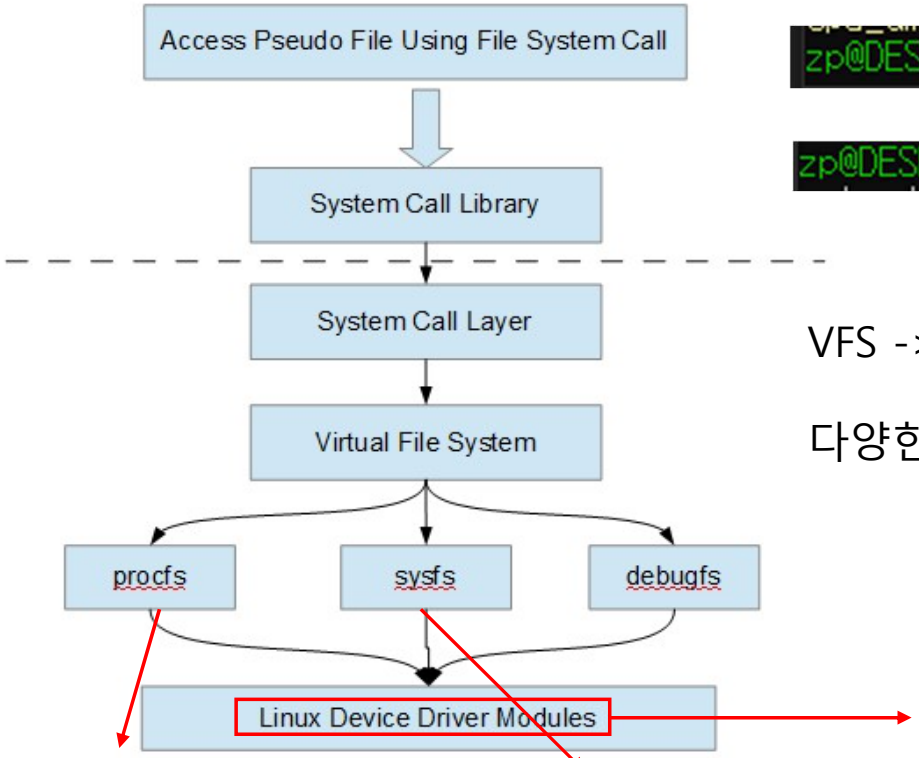
```
zp@DESKTOP-VOGAQA9:/dev$ echo 1 > /proc/sys/net/ipv4/ip_forward
```

```
zp@DESKTOP-VOGAQA9:/dev$ cat /sys/power/mem_sleep
```

VFS -> Hardware 관련 config를 cat, echo로 인터페이스 가능

다양한 hardware를 위해 **추상화**

# background




cpu, mem, process 관련
/proc

Kobject, hardware 관련
/sys

```
zp@DESKTOP-VOGAQA9:/dev$ echo 1 > /proc/sys/net/ipv4/ip_forward
```

```
zp@DESKTOP-VOGAQA9:/dev$ cat /sys/power/mem_sleep
```

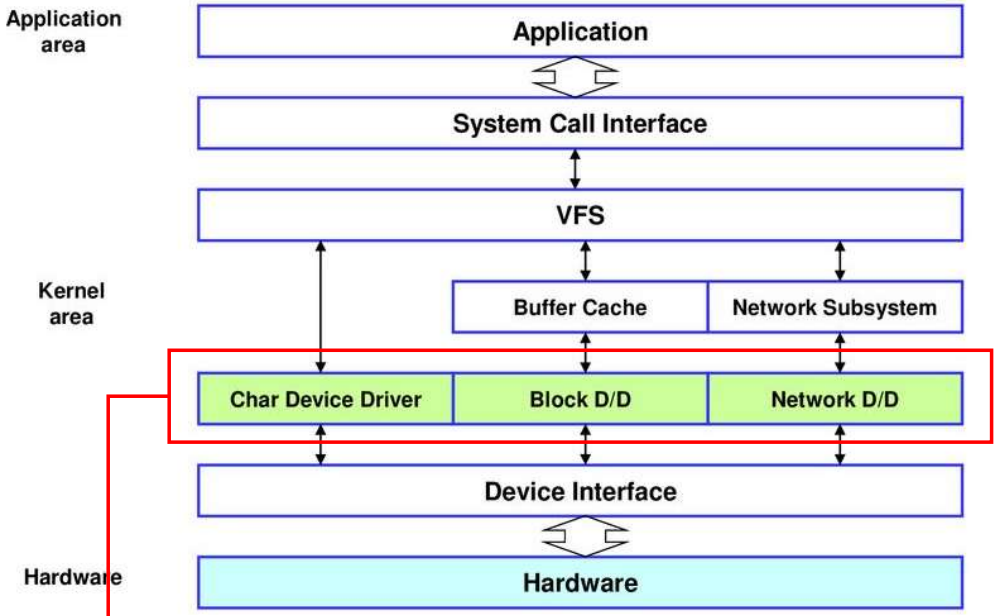
VFS -> Hardware 관련 config를 cat, echo로 인터페이스 가능

다양한 hardware를 위해 **추상화**

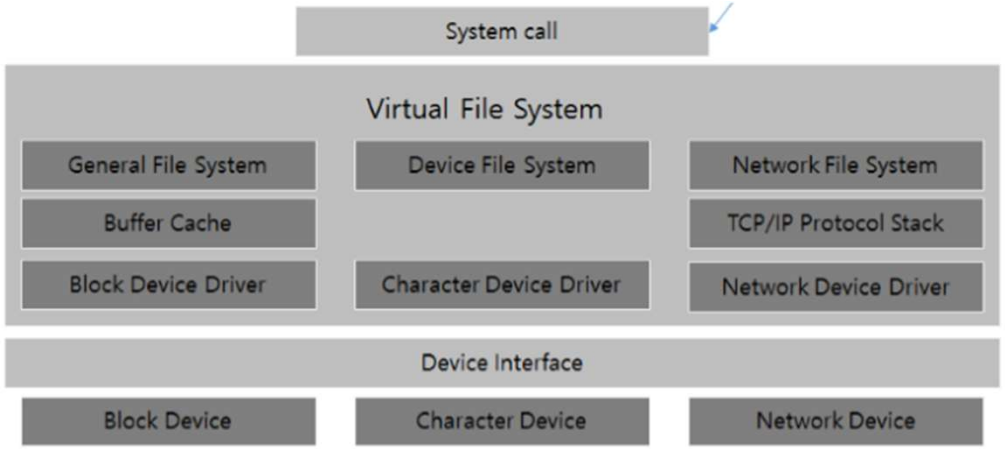
aka. LKDM(linux kernel driver module)

확실한건 아닌데, 논문에서는 이렇게 부름

# background




DD(device driver)는 Char, Block, Net 방식이 존재




| | | |
|---|---|---|
| Char | 1바이트 씩 연속(스트리밍) | /dev/tty, /dev/null, /dev/random |
| Block | 일정 크기(block)단위 입출력 | /dev/sda, /dev/mmcblk, /dev/loop0 |
| Net | 패킷 단위 데이터 전송 | eth0, wlan0, lo |

# background

[init 시]

register_chrdev("major number", "device name", "file_operations 구조체")

char, block은 file_operation structure 사용

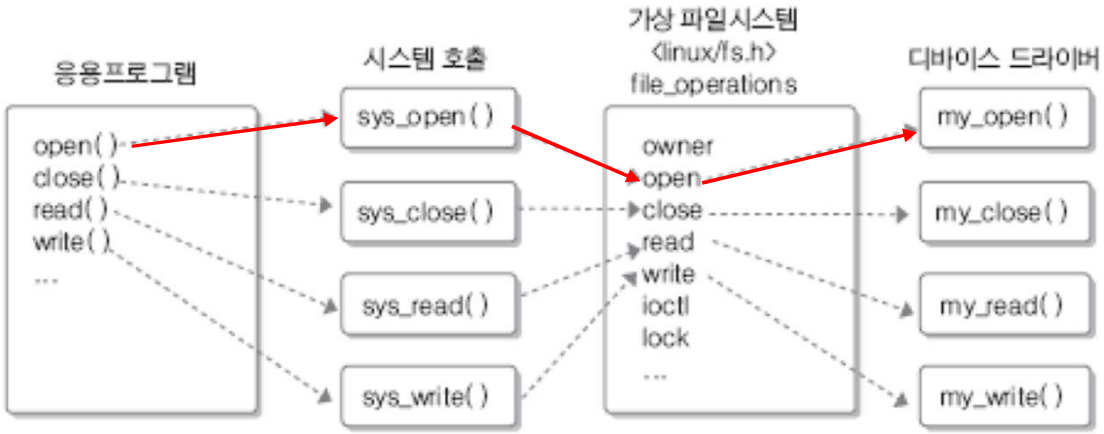
net은 net_device structure 사용

```
struct file_operations {
    struct module *owner;
    loff_t (*llseek) (struct file *, loff_t, int);
    ssize_t (*read) (struct file *, char __user *, size_t, loff_t *);
    ssize_t (*write) (struct file *, const char __user *, size_t, loff_t *);
    ssize_t (*read_iter) (struct kiocb *, struct iov_iter *);
생략
```

[드라이버 로드]

sudo insmod ./device.ko

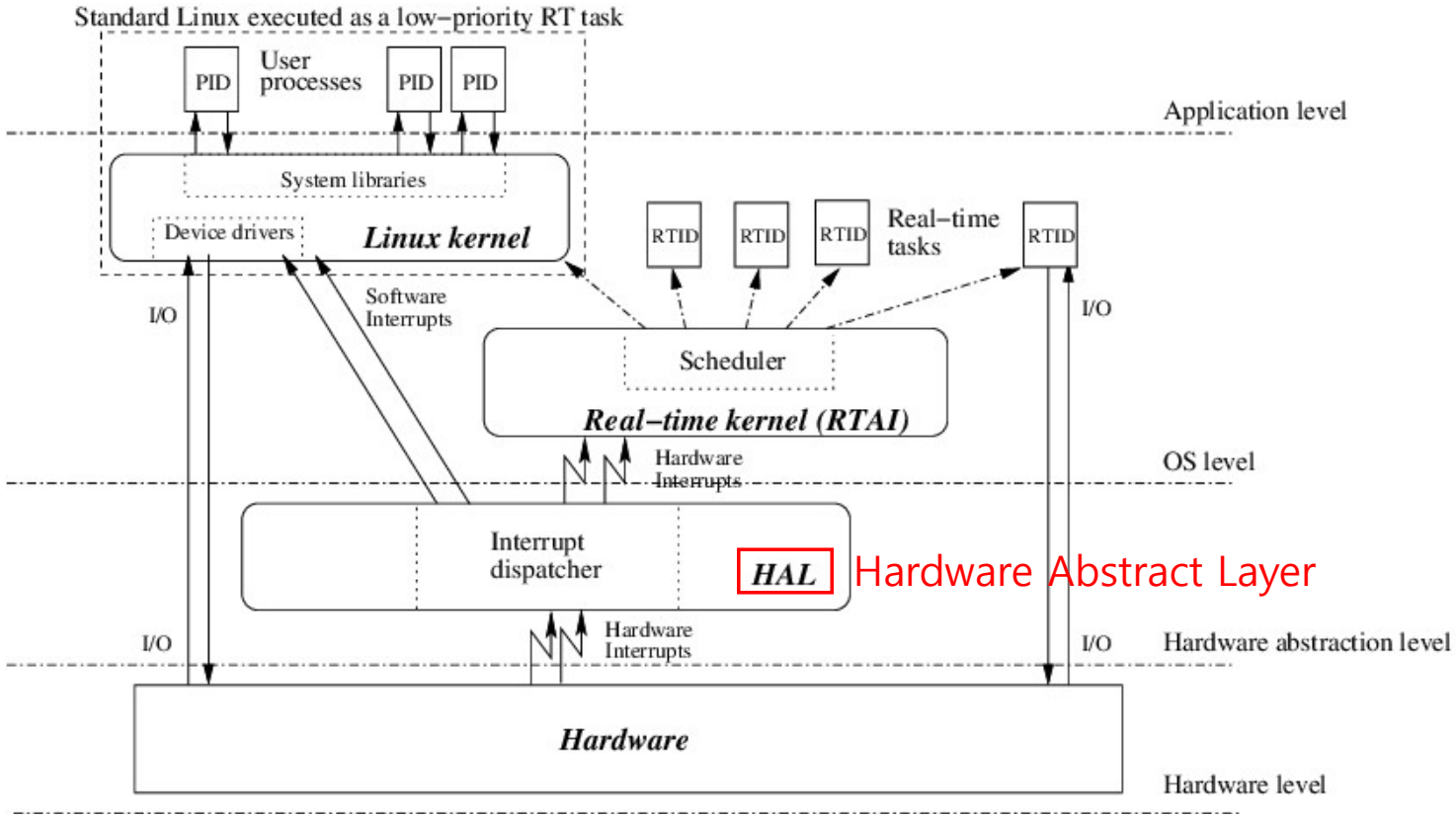
# background




file operations 구조체의 .open 필드에 my_open 등록

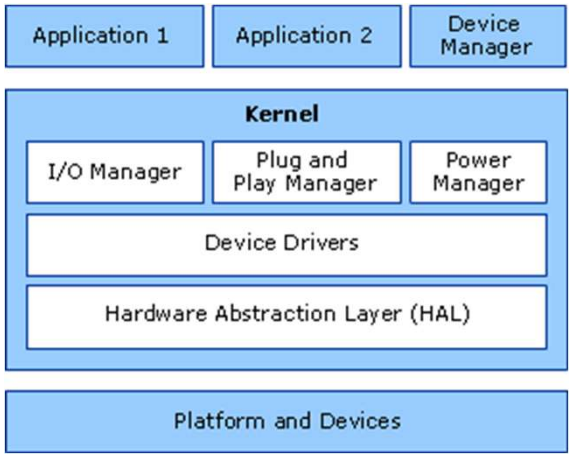
```
int my_open(struct inode *inode, struct file *filp) {
    printk(KERN_ALERT "my_device open function called₩n");
    return 0;
}
```

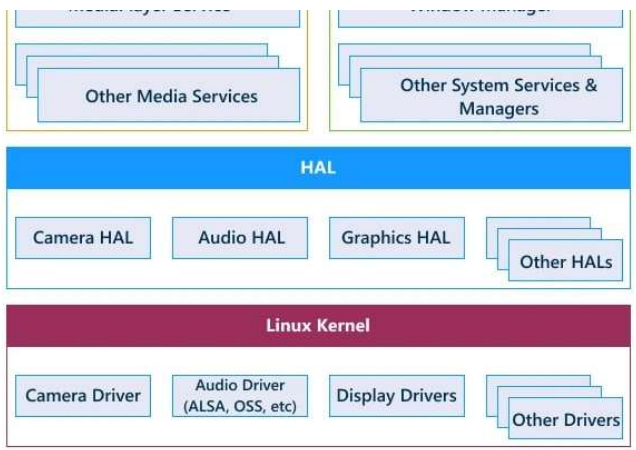
# background

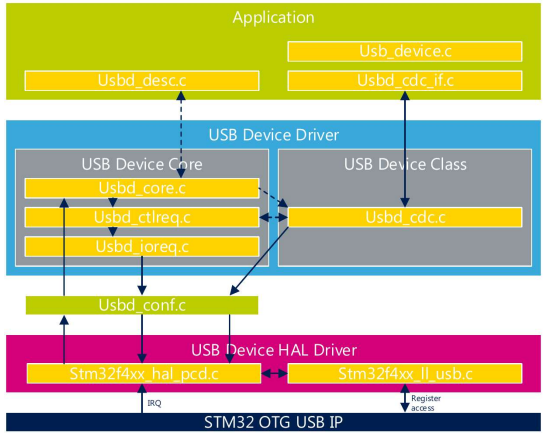

Standard Linux executed as a low−priority RT task
PID
User processes
PID
PID
Application level
System libraries
Device drivers
Linux kernel
RTID
RTID
RTID
Real−time tasks
RTID
I/O
Software Interrupts
I/O
Scheduler
Real−time kernel (RTAI)
Hardware Interrupts
OS level
Interrupt dispatcher
HAL
Hardware Abstract Layer
I/O
Hardware Interrupts
I/O
Hardware abstraction level
Hardware
Hardware level

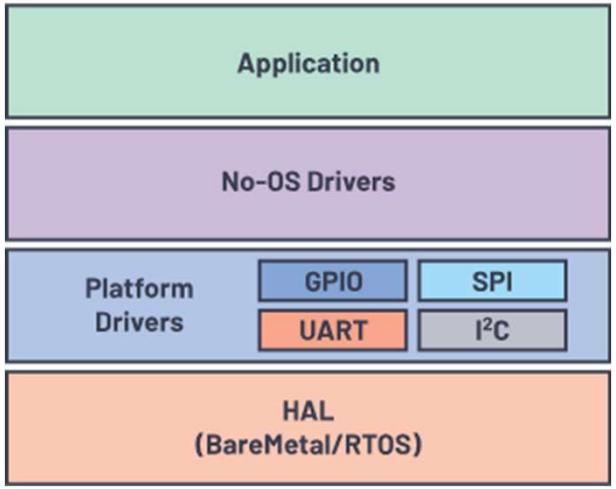
# background

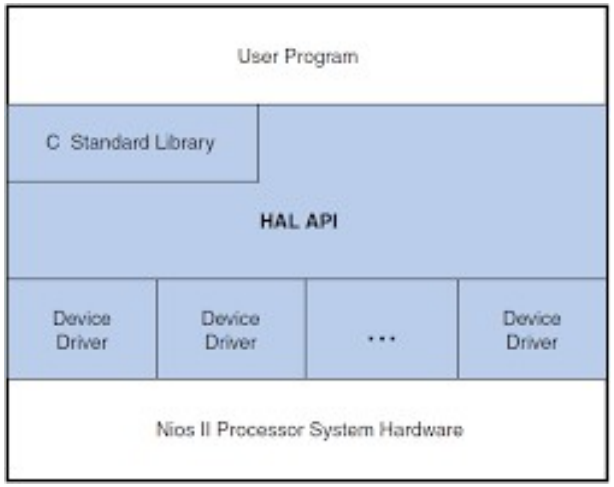
Device Driver와 HAL 의 순서는 아키텍처마다 다를 수 있음
HAL의 추상화 목적에 따라 다름. 이전 슬라이드의 설명은 linux기준

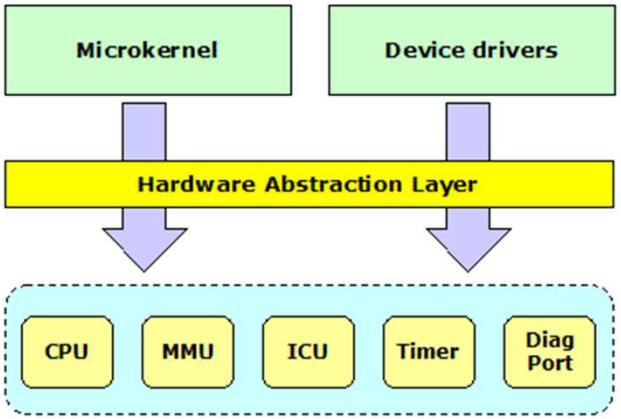


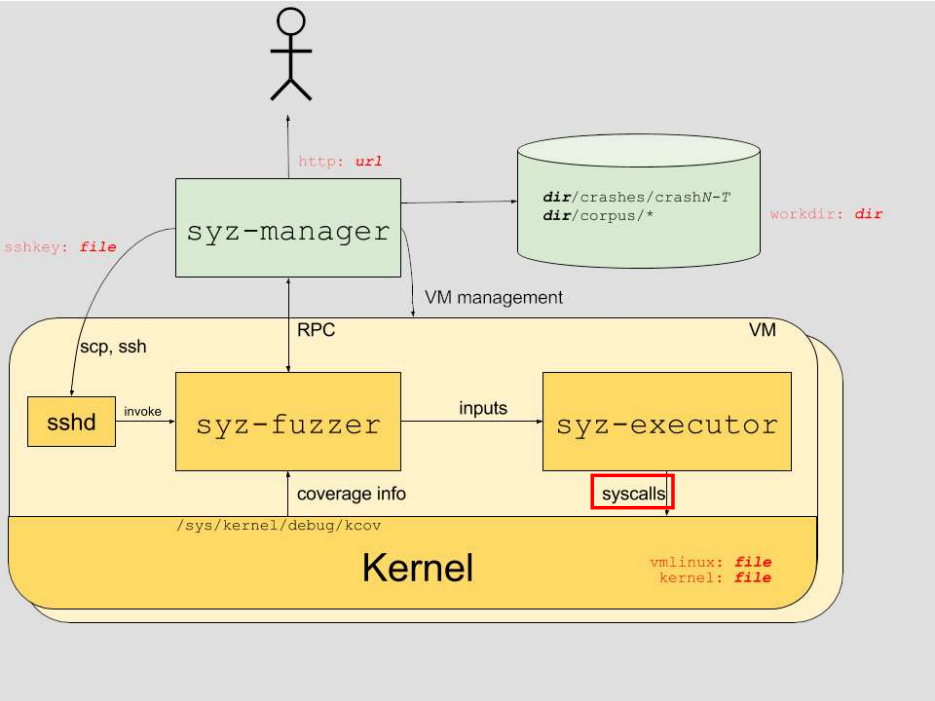



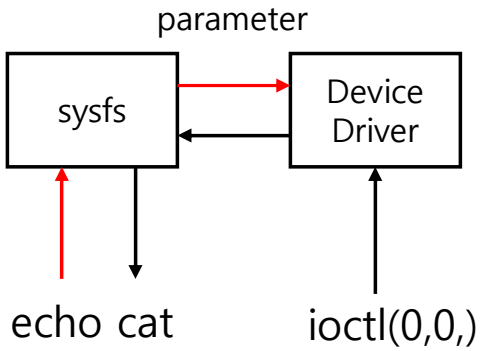



# Problem to Solve

Fuzzing input이 제한적이며, coverage 증가가 한정적




기존 driver fuzzer (syzkaller) – syscall을 input으로 사용

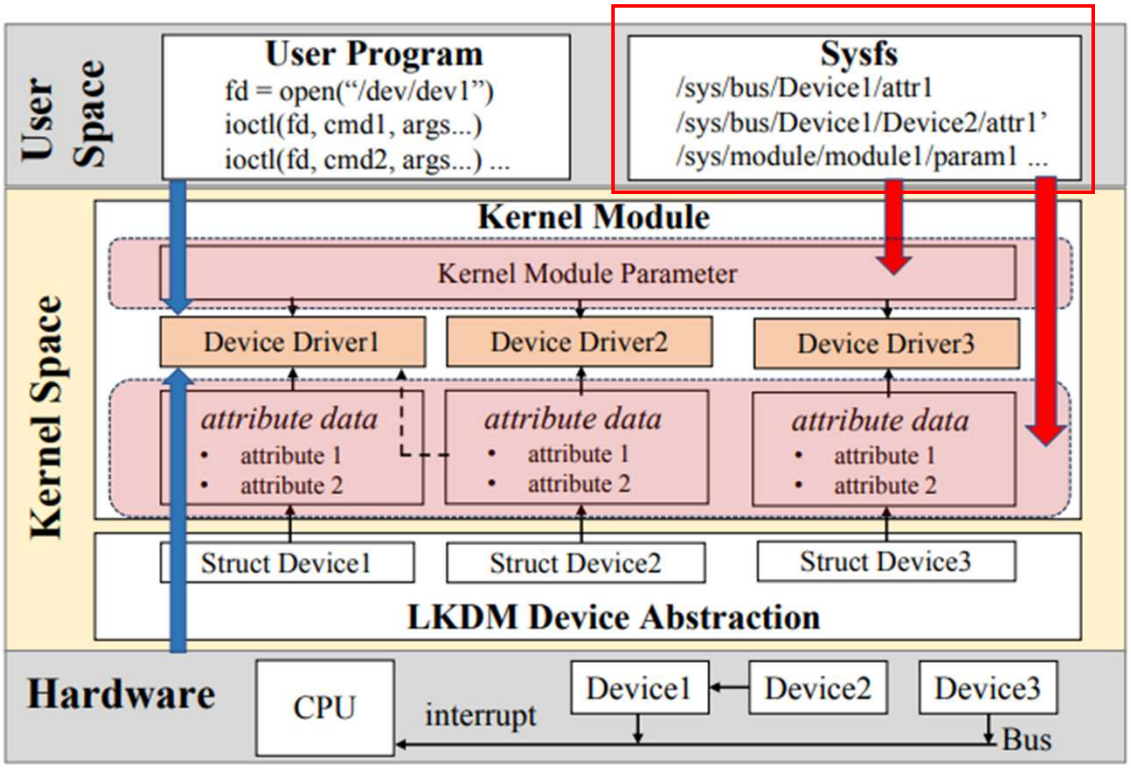



syscall 외에 runtime parameter 도

Device Driver 실행 흐름에 큰 영향을 줌

# PoC

Sysfs를 input으로 사용했을 때, 찾을 수 있는 버그



# PoC - CVE-2021-47375

Sysfs를 input으로 사용했을 때, 찾을 수 있는 버그

```
1  ioctl(/dev/sda, BLKTRACESETUP, blk_user_trace_setup[buf_size])
2  ioctl(/dev/sda, BLKTRACESTART)
3  ioctl(/dev/sdb, BLKTRACESETUP, blk_user_trace_setup[buf_size])
4  ioctl(/dev/sdb, BLKTRACESTART)
5  // Disable blk trace through sysfs while executing other commands
6  echo 0 > /sys/block/sdb/trace/enable &
7  ioctl(/dev/sda, SG_IO, ...)
```

CPU0: ioctl(/dev/sda, SG_IO, ...) CPU1: echo 0 > /sys/block/sdb/trace/enable

```
__blk_add_trace()
  └ trace_note_tsk()
      ...// Itering trace list to sdb
      └ trace_note(sdb)
          └ relay_reserve:
              └ if (buf->offset)  <-- Use-After-Free triggered!
```

```
blk_trace_remove_queue()
 └ blk_trace_free()
      ...
      └ relay_destroy_buf()
          └ kfree(buf)
```

Line 6 에서 sysfs를 통해 sdb의 trace를 disable

Line 7 에서 ioctl을 통해 sda에 전체 디스크 trace를 명령

각 CPU에서 동시에 명령이 실행 -> UAF trigger!

# PoC

Sysfs(runtime parameter)에 영향을 받는 코드가 매우 많다

**Table 1: The number of the branches and basic blocks affected by different user-controlled data.**

| Taint Source | Number | Aff. If | Aff. Switch | Aff. BB |
|---|---|---|---|---|
| Syscall Args | 12839 | 30588 | 1036 | 74172 |
| Module Params | 1494 | 15790 | 46 | 35178 |
| Device Attrs | 1407 | 9226 | 58 | 20875 |

# Challenge

syscall + sysfs(runtime parameter) 를 input으로 하는 퍼저를 만들기 위해 해결할 과제

- sysfs로 runtime parameter를 random하게 mutation하여도, kernel이 엄격하게 validation 한다
  따라서 유의미한 value를 넘겨야함

- 드라이버의 런타임과 관련된 parameter가 무엇인지 식별해야 한다

- Fuzzing engine에 이를 도입해야 한다
  실행마다 parameter를 조작하며, 규칙에 맞는 syscall 코드 또한 생성해야 한다

# Solution

syscall + sysfs(runtime parameter) 를 input으로 하는 퍼저를 만들기 위해 해결할 과제

- sysfs로 runtime parameter를 random하게 mutation하여도, kernel이 엄격하게 validation 한다
  따라서 유의미한 value를 넘겨야함

  driver source code 의 valid check logic 을 파싱하여 조건을 학습 : 유사 정적 분석

# Solution

syscall + sysfs(runtime parameter) 를 input으로 하는 퍼저를 만들기 위해 해결할 과제

- 드라이버의 런타임과 관련된 parameter가 무엇인지 식별해야 한다

  Value Collection / Relation Identification

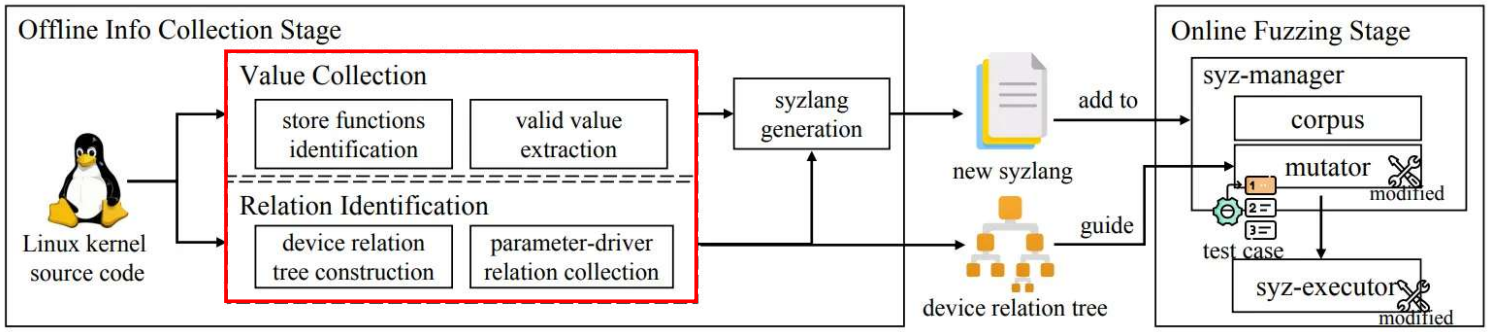



**Fig. 6: The framework overview of SyzParam.**

# Solution

syscall + sysfs(runtime parameter) 를 input으로 하는 퍼저를 만들기 위해 해결할 과제

- Fuzzing engine에 이를 도입해야 한다
  실행마다 parameter를 조작하며, 규칙에 맞는 syscall 코드 또한 생성해야 한다

  sysParam 이라는 새로운 개념과, 기존의 syzkaller를 함께 사용 (syzlang 그대로 사용)

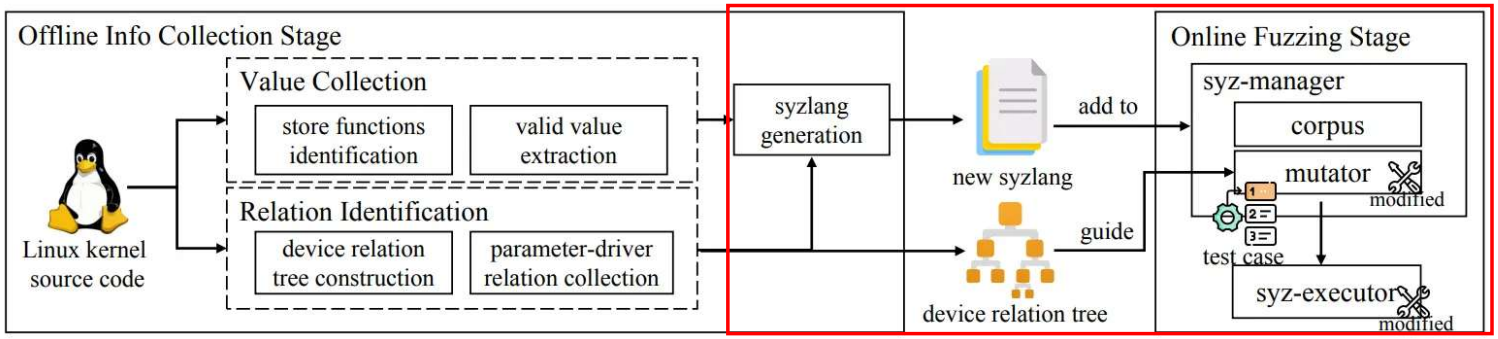



Fig. 6: The framework overview of SyzParam.

# Solution

Challenge 해소

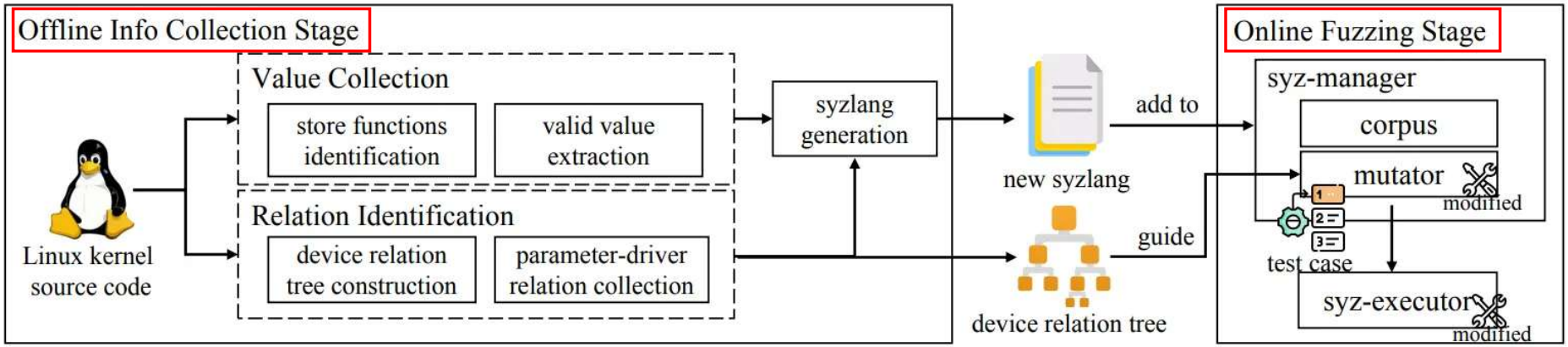


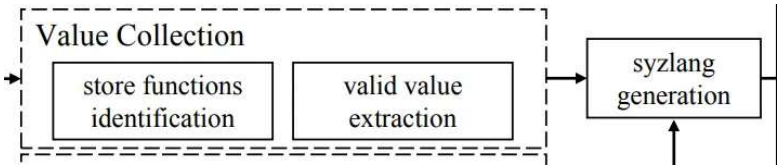

**Fig. 6: The framework overview of SyzParam.**

online / offline stage로 구분한다.

online stage는 기존 syzkaller를 그대로 사용하다.

# Solution - Value Collection

Challenge 해결 방법 (2번째 challenge)




**Value collection**

- store functions identification

  *_DEVICE_ATTR[_*] 패밀리들은 결국 ATTR[_*]을 상속

  매개변수는 struct attribute을 통해서만 유저 스페이스에 노출되므로, 커널 내 모든 관련 구조체를 조사

  읽기 전용(read-only) 속성은 드라이버의 제어 흐름에 영향을 미치지 않으므로 무시

- valid value extraction

  Store함수를 파싱 -> static taint analysis

  Taint sink -> match_string(), strcmp(), strncmp(), memcmp() / kstrtouint(), kstrtol(), simple_strtoul() 추적

  module_param() 매크로 분석 (커널 모듈 매개변수)

# Solution - Relation Identification

Challenge 해결 방법 (2번째 challenge)

| device relation tree construction | parameter-driver relation collection |
|---|---|

**Relation Identification**

- Inter-device relationship

  /sys/devices/* 조회 시 하위 디렉터리가 계층 구조를 띈다

  tree 를 통해 상호 영향이 있을 수 있는 device 분류

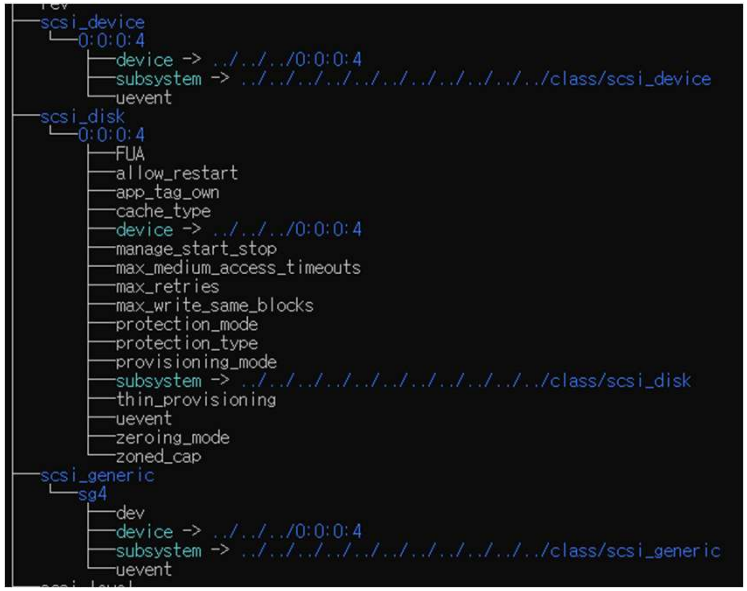
- Parameter-Driver Relation

  /sys/ 의 파라미터가 어느 /dev/ 에 대응하는 지를 mapping

  ex) /sys/block/**sda**/queue/scheduler → /dev/**sda**

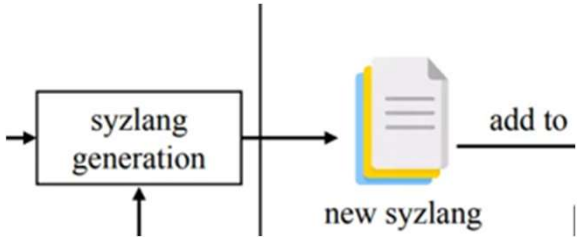
  폴더 이름이 겹치는 것으로 판단(리눅스 설계 기반)

```
/sys/devices/LNXSYSTM:00/LNXSYBUS:00$ tree
```



# Solution - syzlang

Challenge 해결 방법 (3번째 challenge)




```
1  resource fd_fb[fd]
2  // The return value of openat() is a file descriptor, marked as resource in syzlang
3  openat$fb(fd const, file ptr[in, string["/dev/fb0"]], flags flags, mode const[0]) fd_fb
4  // Different syscall types on same resource
5  write$fb(fd fd_fb, data ptr[in, array[int8]], len bytesize[data])
6  read$fb(fd fd_fb, data ptr[out, array[int8]], len bytesize[data])
7  // Same syscall with different arguments
8  ioctl$FBIOCMD1(fd fd_fb, cmd const[FBIOCMD1], arg ptr[out, array[int8, FB_SIZE]])
9  ioctl$FBIOCMD2(fd fd_fb, cmd const[FBIOCMD2], arg ptr[in, fb_cmap_user])
10 // User space structure used in syscalls
11 type fb_cmap_user {
12         start int32
13         len len[red, int32]...
14 }
```

**Fig. 2: An example of *syzlang* provided by Syzkaller. *Syzlangs* are declarative descriptions of syscall interfaces to manipulate programs, which will be used to generate, mutate, execute, minimize, serialize, and deserialize programs.**

syz_mod_dev(param_path, param_val, dev_path, intptr, flags)

syz_mod_dev라는 문법을 추가하였다.

기존에는 ioctl, read, write 등으로 syscall 인자만 설정할 수 있었다.

해당 문법을 통해, parameter를 syzlang으로 결정할 수 있다

# Limitation

논문 제시 + 개인적인 생각

정적 분석 시 indirect call 추적 불가 (다른 논문들과 유사)

정적 분석 시 복잡한 방식은 분석 제한

매개변수 이름 충돌(동일한 폴더 이름 사용) -> 1.71%로 결과에 영향은 크게 없다고 주장

Symbolic Execution이 개선 방안이 될 수 있다고 제시함

# Performance

Syzkaller, SyzDescribe, SyzGen++, SyzParam **Cov** 비교

| Drivers | #Avg. Edge Coverage | | | | | p-value | | |
|---|---|---|---|---|---|---|---|---|
| | Syzkaller | SyzDescribe | SyzGen++ | SyzParam- | ↑Comparing to Syzkaller | Syzkaller | SyzDescribe | SyzGen++ |
| rtc | 3,424 | 3,537 | 3,517 | **5,337** | 55.89% | 0.0060 | 0.0040 | 0.0040 |
| usb | 11,301 | N/A | 11,638 | **16,523** | 46.21% | 0.0040 | N/A | 0.0040 |
| loop | 6,348 | 7,708 | 7,120 | **9,244** | 45.63% | 0.0040 | 0.016 | 0.0040 |
| msr | 1,883 | 2,100 | 2,474 | **2,529** | 34.28% | 0.0040 | 0.0040 | 0.0040 |
| i2c | 3,481 | 3,297 | 4,309 | **4,386** | 26.02% | 0.0040 | 0.0040 | 0.21 |
| sr | 7,301 | 6,954 | N/A | **8,653** | 18.52% | 0.0040 | 0.0040 | N/A |
| mouse | 2,958 | 3,690 | N/A | **3,482** | 17.73% | 0.0040 | 0.0040 | N/A |
| seq | 3,610 | 3,690 | **4,244** | 4,198 | 16.29% | 0.0040 | 0.0278 | 0.85 |

# Performance

Syzgen, SyzParam, combine **Cov** 비교

| Drivers | Syzgen++ | SyzParam- | Combined | ↑ in Edge Cov. |
|---|---|---|---|---|
| rtc | 3,517 | 5,337 | 5,560 | 58.09% |
| usb | 11,638 | 16,523 | 17,671 | 51.84% |
| loop | 7,120 | 9,244 | 9,227 | 29.59% |
| msr | 2,474 | 2,529 | 3,158 | 27.65% |
| i2c | 4,309 | 4,386 | 4,742 | 10.05% |
| seq | 4,244 | 4,198 | 4,759 | 12.13% |

# Performance

Syzkaller, Syzlang **unique crash** 비교

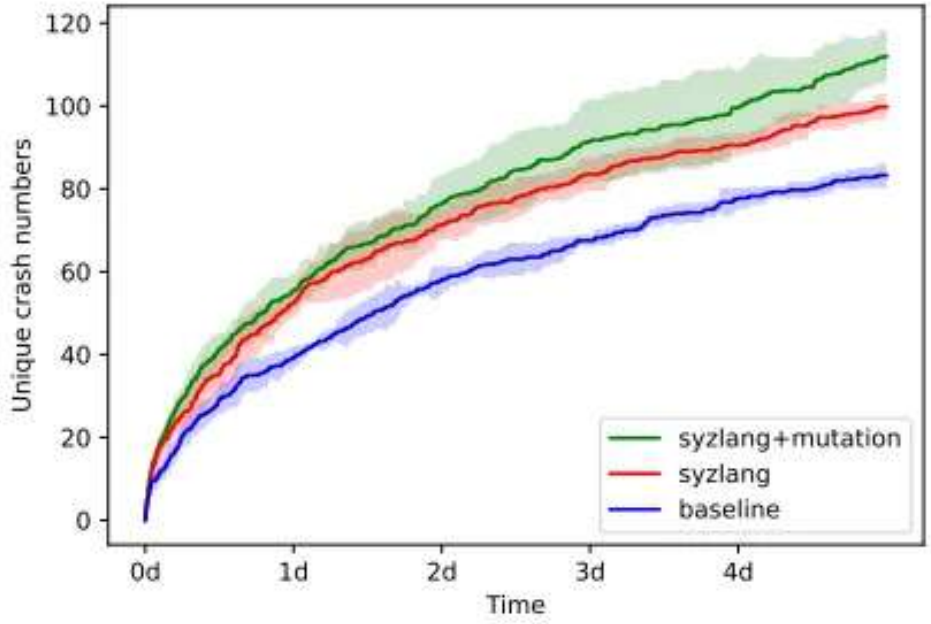



**Fig. 7: Number of unique crashes over time.**

# Performance

새롭게 발견한 취약점

30개 ( 20개 accept / 14 fixed / 9 CVE published )

*CVE가 없는 취약점 지표가 논문에서 의미가 있는지?