Fuzzing File System via Two-Dimensional Input Space Exploration

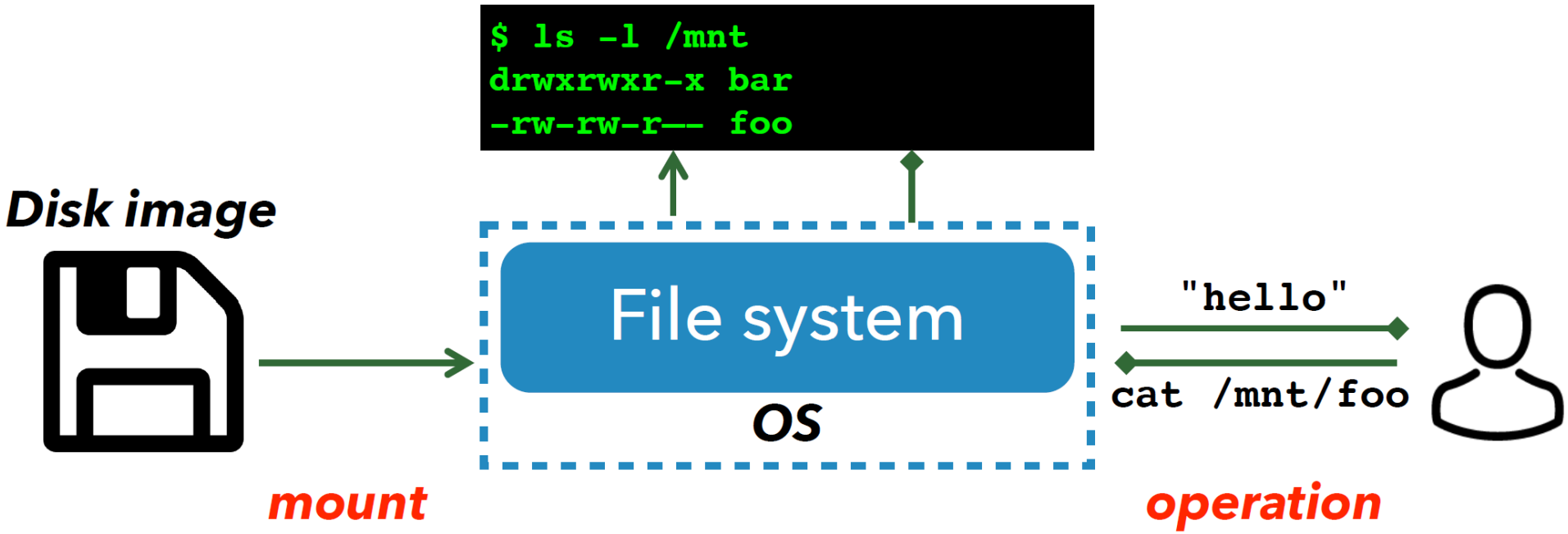
40th IEEE Symposium on Security & Privacy
MAY 20-22, 2019 AT THE HYATT REGENCY, SAN FRANCISCO, CA

Wen Xu Hyungon Moon† Sanidhya Kashyap Po-Ning Tseng Taesoo Kim

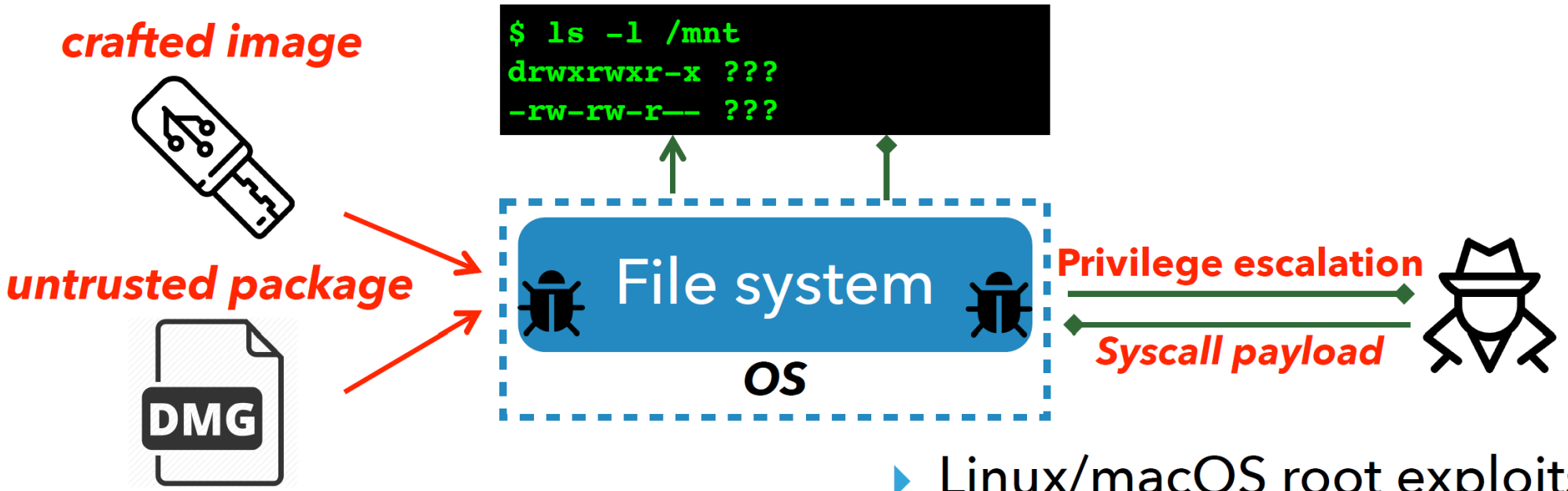
Georgia Institute of Technology
†Ulsan National Institute of Science and Technology

2019.07

File System


$ ls -l /mnt
drwxrwxr-x bar
-rw-rw-r-- foo
Disk image
File system
OS
"hello"
cat /mnt/foo
mount
operation

File System Attack



- Linux/macOS root exploits
- Evil maid attacks
- Air-gapped APT attacks

Complexity

FS	LoC	Active
ext4	50K	✓
XFS	140K	✓
Btrfs	130K	✓

File systems are hard to be *bug-free!*

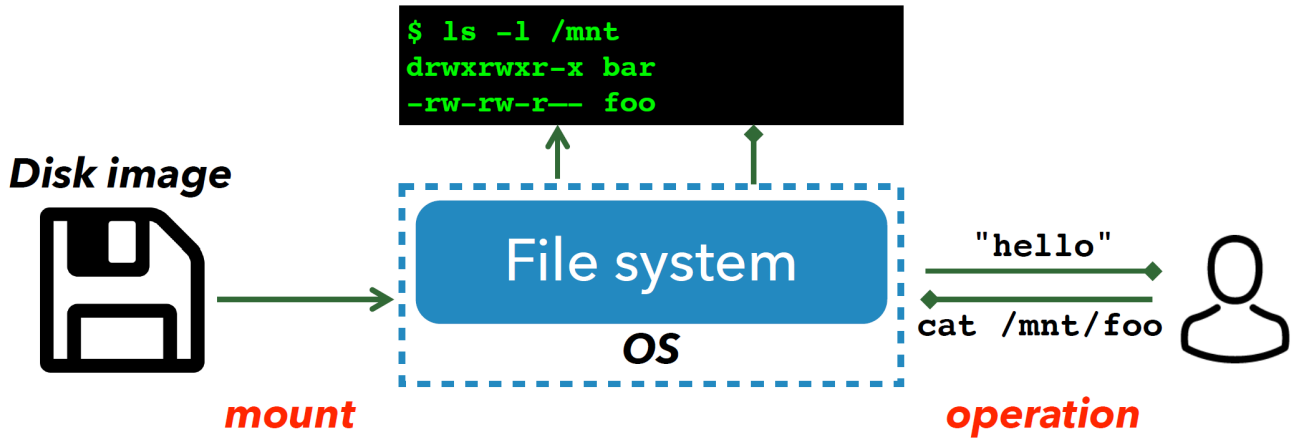
Solution - Fuzzing

AFL

- Fuzzing images as binary blob
- Mount images

Syzkaller

- Fuzzing system calls
- Execute file operation



Challenge – Fuzzing Images

Size

Minimum size of a file system ≫ maximum preferred size of fuzzers

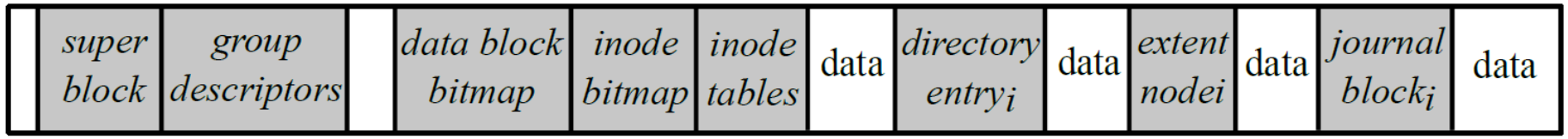
⟹ Huge I/O on loading/saving testcases

- ext4 – 2MB
- XFS – 16 MB
- Btrfs – 100MB
- AFL – 1MB

Challenge – Fuzzing Images

Highly structured metadata

- Metadata is rarely touched (1% of image size)
- OS does not care about data in a file

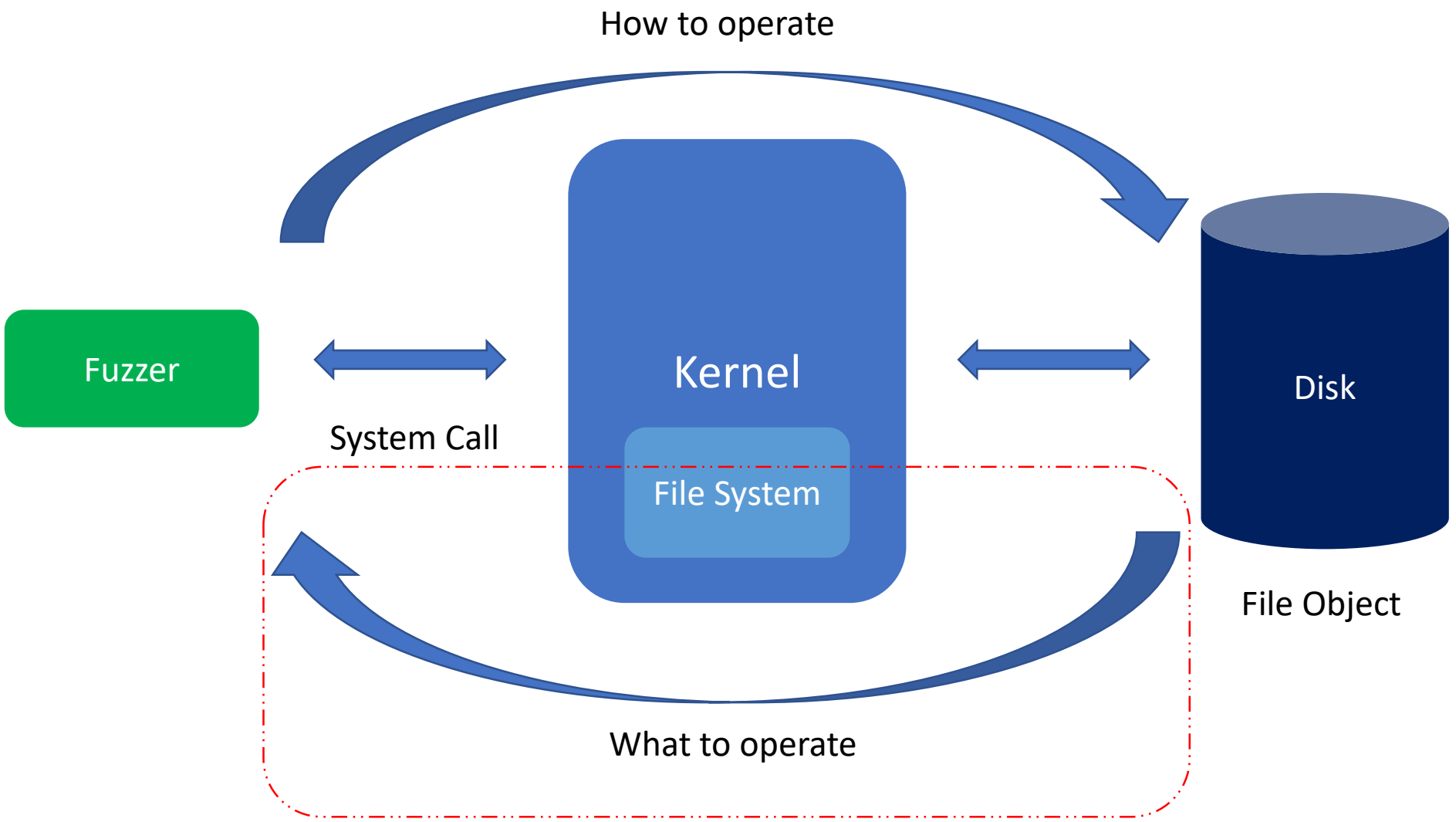


Challenge – Fuzzing Images

Checksum

- Fuzzing the checksum together - inefficient

Challenge – Fuzzing System Call



Challenge – Fuzzing System Call

```
open(filename, flag)
read(fd, buffer, int)
write(fb, buffer, int)
mkdir(filename)
rename(filename, filename)
…
```

Static rules

- Semantically correct
- Context-unaware
- × Inter-dependence between file operation and files

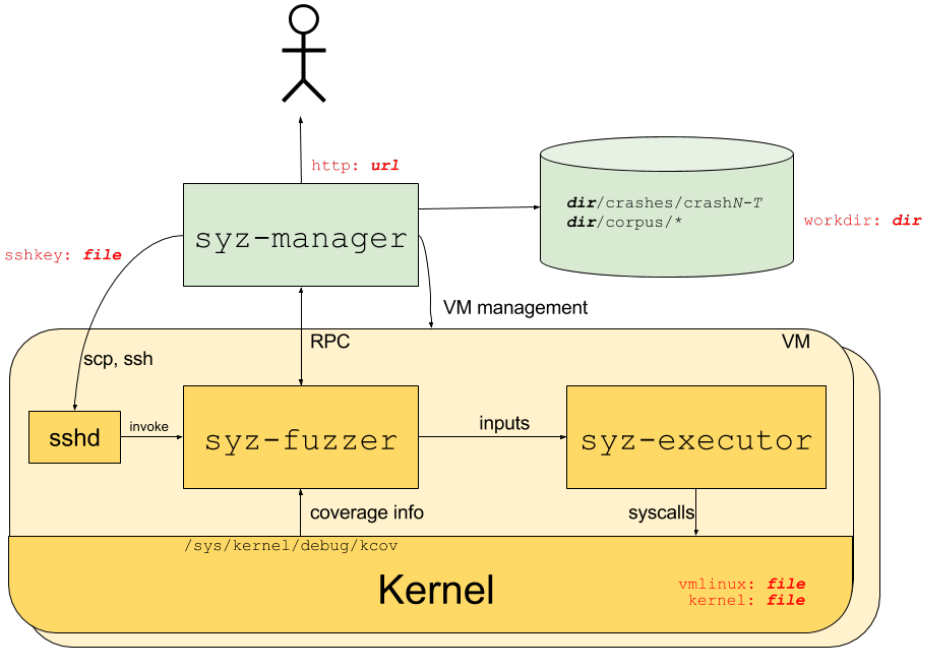

```
mkdir(“A”);
int fd = open(“A”, O_RDWR);
```

```
rename(“A”, “B”);
int fd = open(“A”, O_RDWR);
read(fb, buf, 1024);
```

Challenge – Fuzzing Efficiency with VM

Never reboot until a VM crashes

- Due to the cost of **reboot** or **reverting from snapshot**
- A bug may accumulate the impact of thousands of syscalls
- Aging OS state
  - Unstable executions
  - Hard to reproduce bugs



Challenge – Two-Dimension Input

Jointly fuzzing **images** and **system calls**

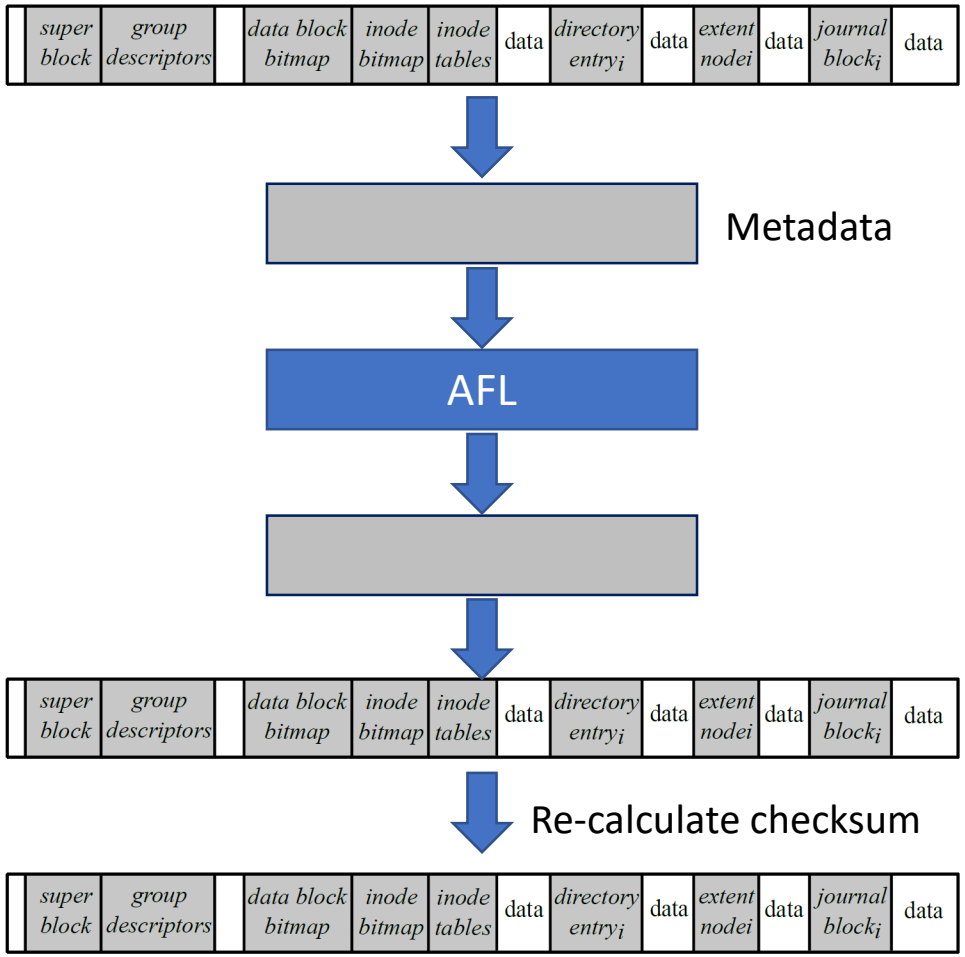
What is JANUS’s solution?

Solution – Fuzzing Images

1. Less frequent I/O when mutating image
2. Extract and mutate metadata only (1%)
3. Fix checksum

- Image parser
- Image mutator

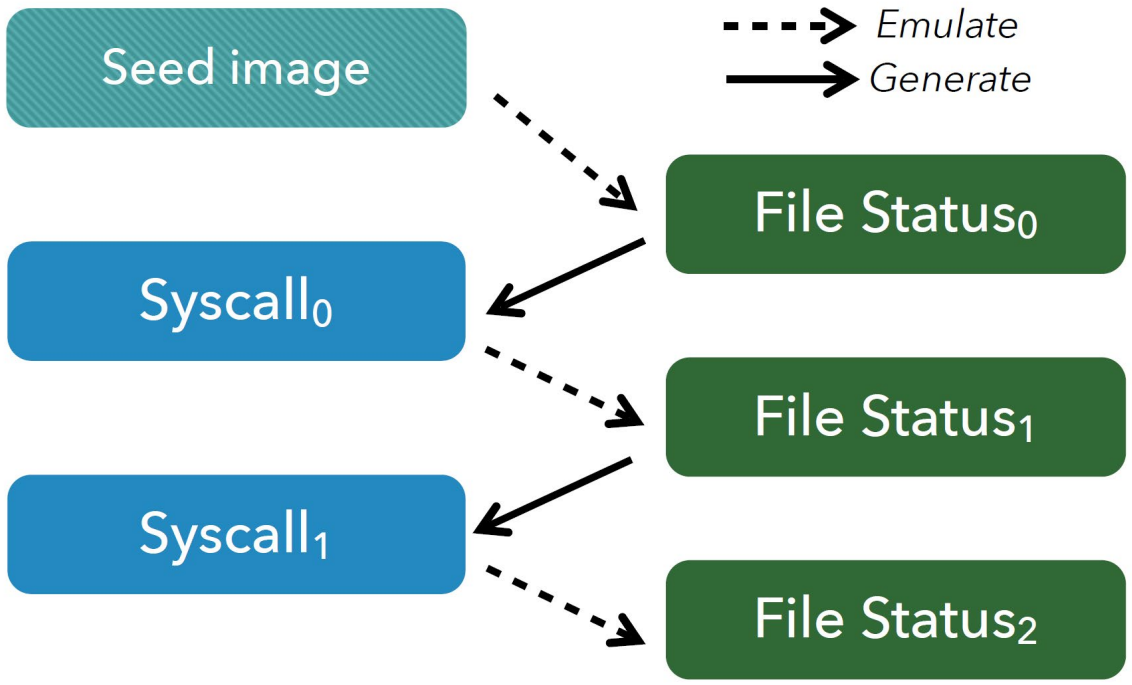
Eight filesystem, user space utilities – mkfs, fsck



Solution – Fuzzing System Calls

Context-aware syscall generation

- Image inspector
  - File object path
  - File object type
  - File object extend attribute
- System call fuzzer



Solution – Fuzzing Efficiency with VM

Library OS – Linux Kernel Library

- Use Linux kernel as a linkable library, so that the non-static functions from the kernel can be called from some external program.

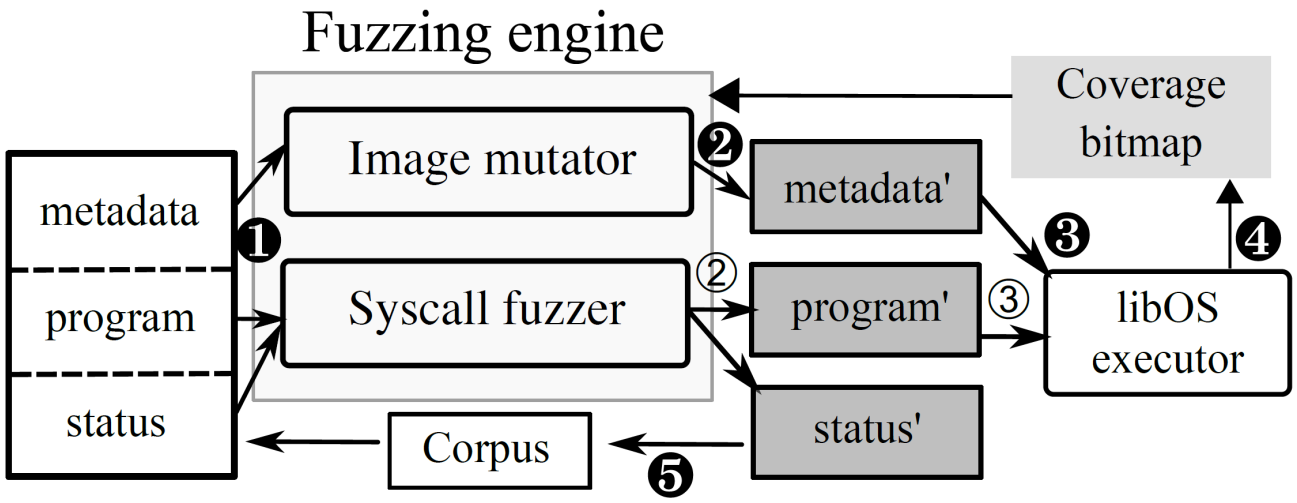
- User-space executor linked with LKL
- Fork a new instance for each test case
  - Invoke LKL system call to mount an image
  - Invoke the generated system calls (file operations)

Re-initialize the OS states within milliseconds

Solution – Two-Dimension Input

Initialize corpus

- Seed image metadata
- Starting program
- File object status

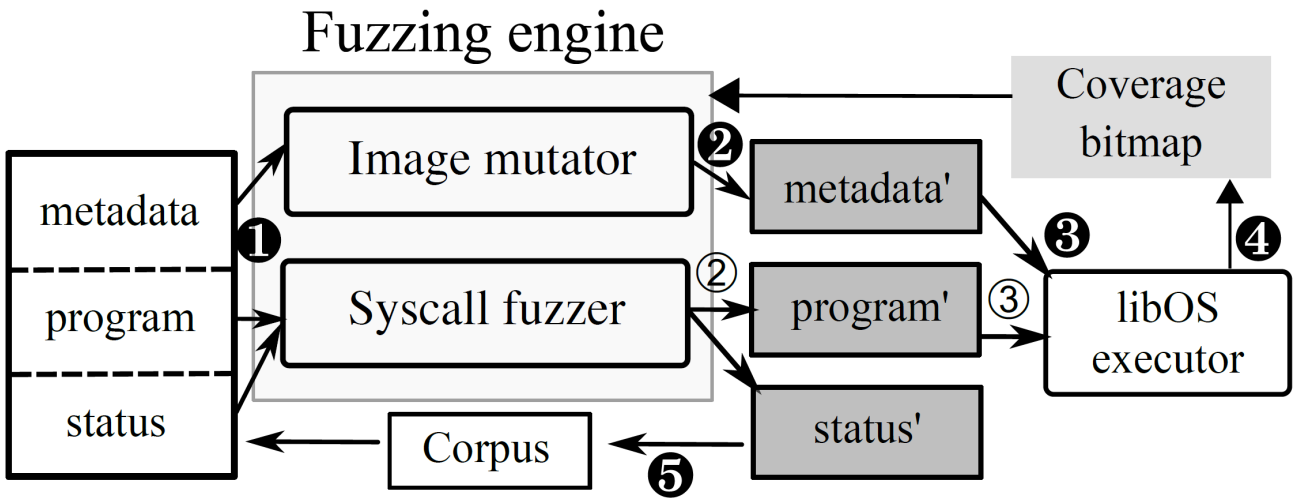



```
./                # root
./foo             # folder
./foo/bar         # folder
./foo/bar/acl     # file protected by ACL
./foo/bar/baz     # normal file
./foo/bar/fifo    # FIFO file
./foo/bar/hln     # hardlink to baz
./foo/bar/sln     # softlink to baz
./foo/bar/xattr # file with an extended attribute
```

Solution – Two-Dimension Input

Mutating image

- Keep the program intact
- Until no more coverage for certain rounds

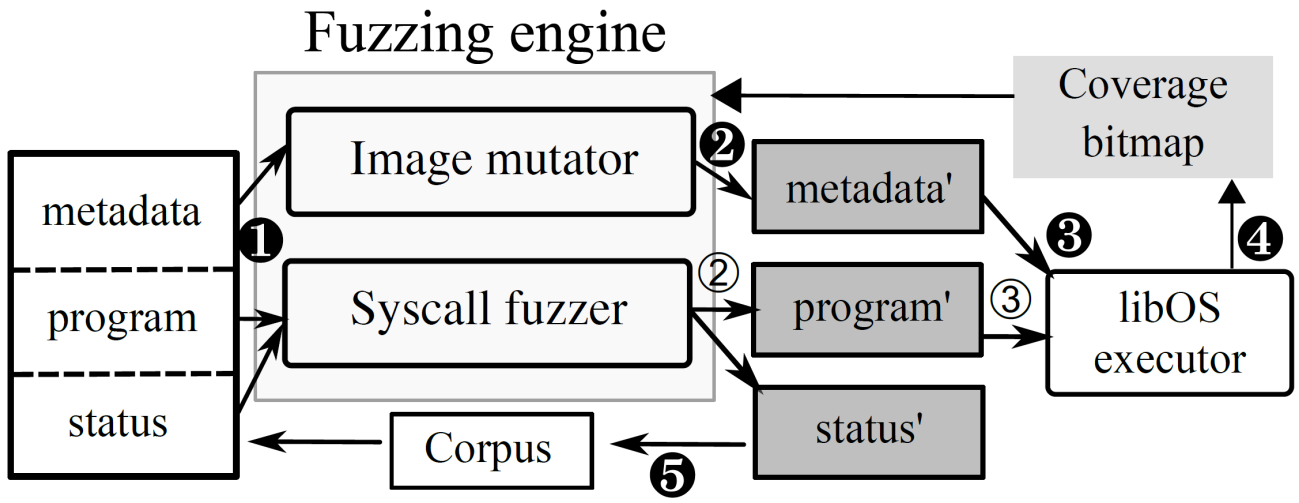



```python
# Class ImageMutator
def mutate_image(meta_buffer):
    choice = Random.randint(0, 8)
    if choice == 0:
        return flip_bit_at_random_offset(meta_buffer)
    elif choice == 1:
        return set_interesting_byte_at_random_offset(meta_buffer)
    elif choice == 2:
        return set_interesting_word_at_random_offset(meta_buffer)
    elif choice == 3:
        return set_interesting_dword_at_random_offset(meta_buffer)
    elif choice == 4:
        return inc_random_byte_at_random_offset(meta_buffer)
    elif choice == 5:
        return inc_random_word_at_random_offset(meta_buffer)
    elif choice == 6:
        return inc_random_dword_at_random_offset(meta_buffer)
    else:
        return set_random_byte_at_random_offset(meta_buffer)
```

Solution – Two-Dimension Input

Mutating system calls

- Keep the metadata intact
- (1) Syscall mutation
- Until no more coverage for rounds
- (2) Syscall generation
- Until no more coverage for rounds



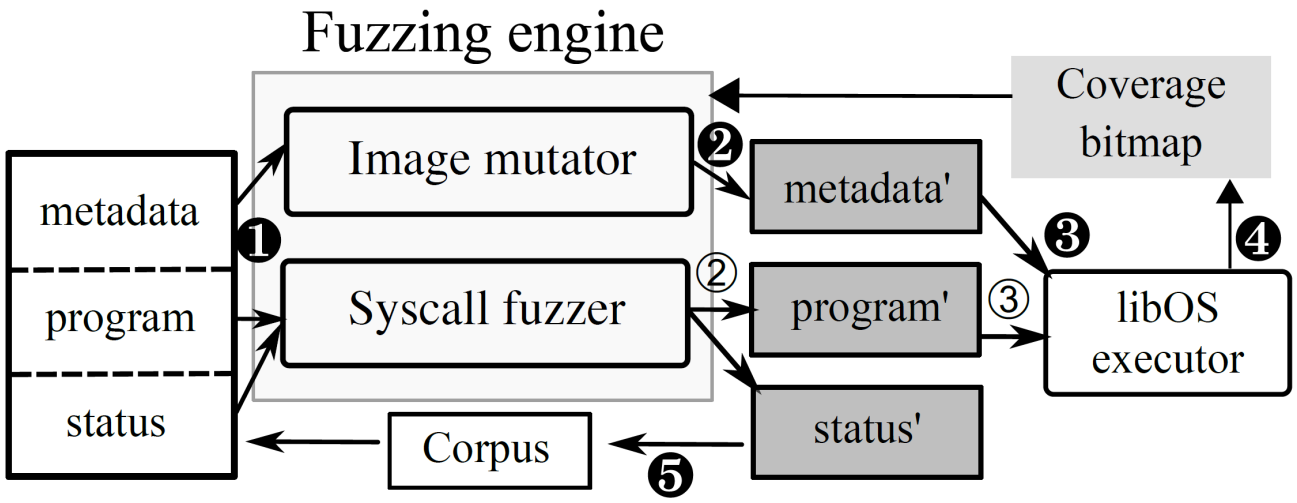
Status

- File descriptor
- Path
- Extended attribute

Solution – Two-Dimension Input

Summary

- Mutate image for rounds
- Mutate argument types for rounds
- Append new syscalls for rounds



- If new code covered, add to corpus

Evaluation

How effective in discovering previously unknown bugs in file systems?

Found more bug than Syzkaller

90 bugs, 62 previously unknown, 36 in wildly used FS – *ext4, XFS, Btrfs*

32 CVEs assigned

25% bugs – mount only a corrupted image

80% bugs – Need to invoke three or more system calls

Most bugs – mounting a corrupted image followed by particular syscalls

Evaluation

How effective in exploring images?

- Syzkaller’s image fuzzing support
- Fuzzing only image

Syzkaller considers only the non-zero chunks as important part

- Miss metadata block
- Include inessential data blocks
- Not fixing checksum

Result – 4.17× coverage

Evaluation

2. How effective in exploring system calls?

- Syzkaller’s system call fuzzer
- Fuzzing only system call

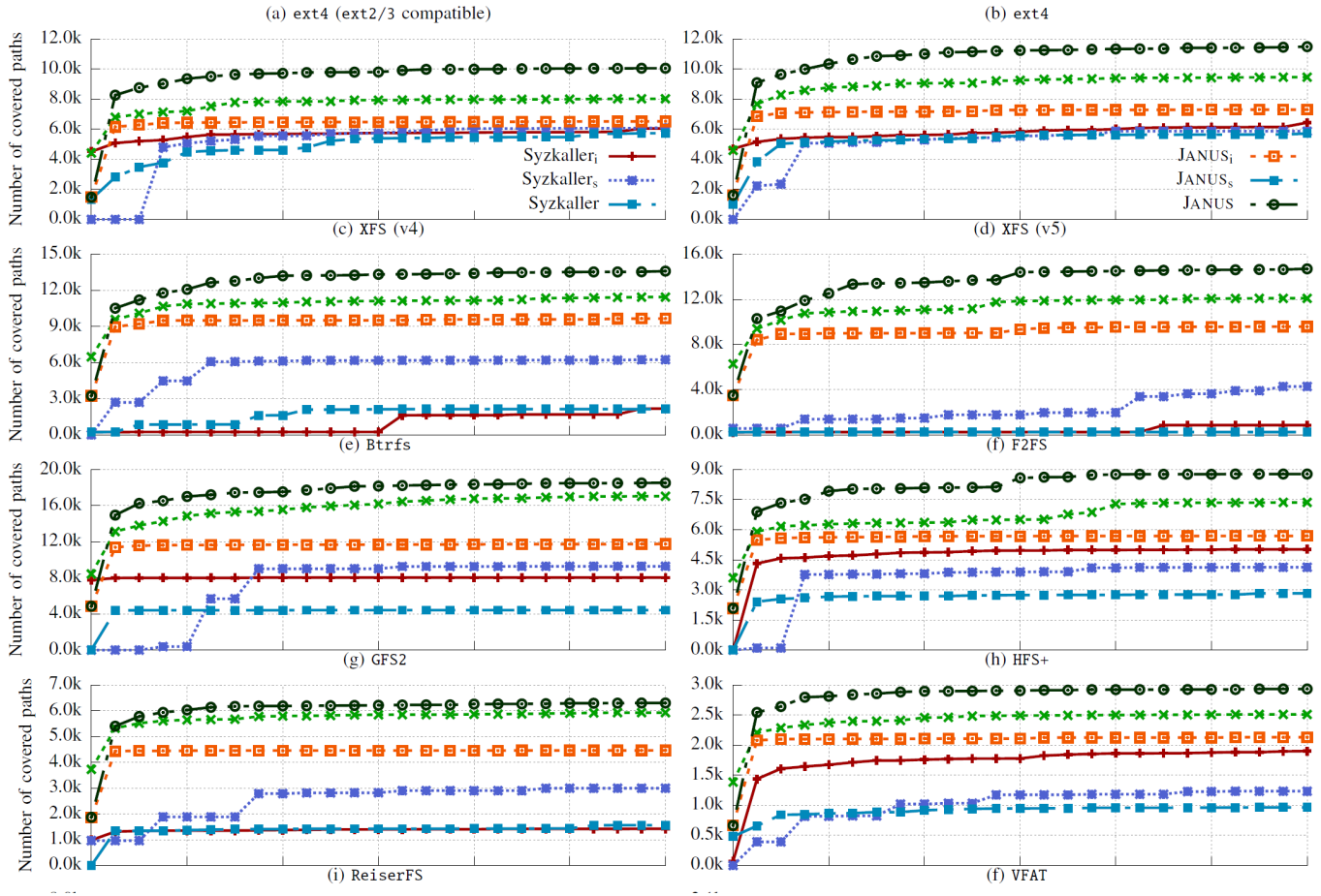
Context-aware system call generation

Result – 2.24× coverage

Evaluation

How effective in exploring two-dimensional input space?

- $Syzkaller_i$
- $Syzkaller_s$
- $Syzkaller$
- $JANUS_i$
- $JANUS_s$
- $JANUS$



Evaluation

How effective in reproducing crashes?

- Syzkaller fails to reproduce any of its found crashes
- JANUS can reproduce 95% of its found crashes
  - *Btrfs* - non-deterministic execution
- Overhead of VM & LKL

Reboot VM	Revert snapshot	LKL
14.5s	1.4s	10.7ms

Evaluation

What else can JANUS contribute to?

- Malicious image sample
- Extend security checks into the user-space tool for hardening - *fsck*

Limitations

Minimal PoC generator – brute force

- Revert every mutated byte
- Remove every invoked file operation
- Check whether the kernel still crashes at the expected location

Multi-threaded bugs?

Thanks.