PeX: A Permission Check Analysis Framework for Linux Kernel

Tong Zhang[1], Wenbo Shen[2], Dongyoon Lee[3], Changhee Jung[4], Ahmed M. Azab[5], Ruowen Wang[5]

[1] Virginia Tech, [2] Zhejiang University, [3] Stony Brook University, [4] Purdue University, [5] Samsung Research America, now at Google

Permission Control in Linux Is Complex

1. DAC (Discretionary Access Controls)

   e.g., drwxr-xr-x for /root

2. Capabilities — **38** in Linux Kernel v4.18.5

   /bin/ping only has cap_net_raw (no more suid, full root)

3. LSM (Linux Security Module) — **190** hooks in Linux Kernel v4.18.5

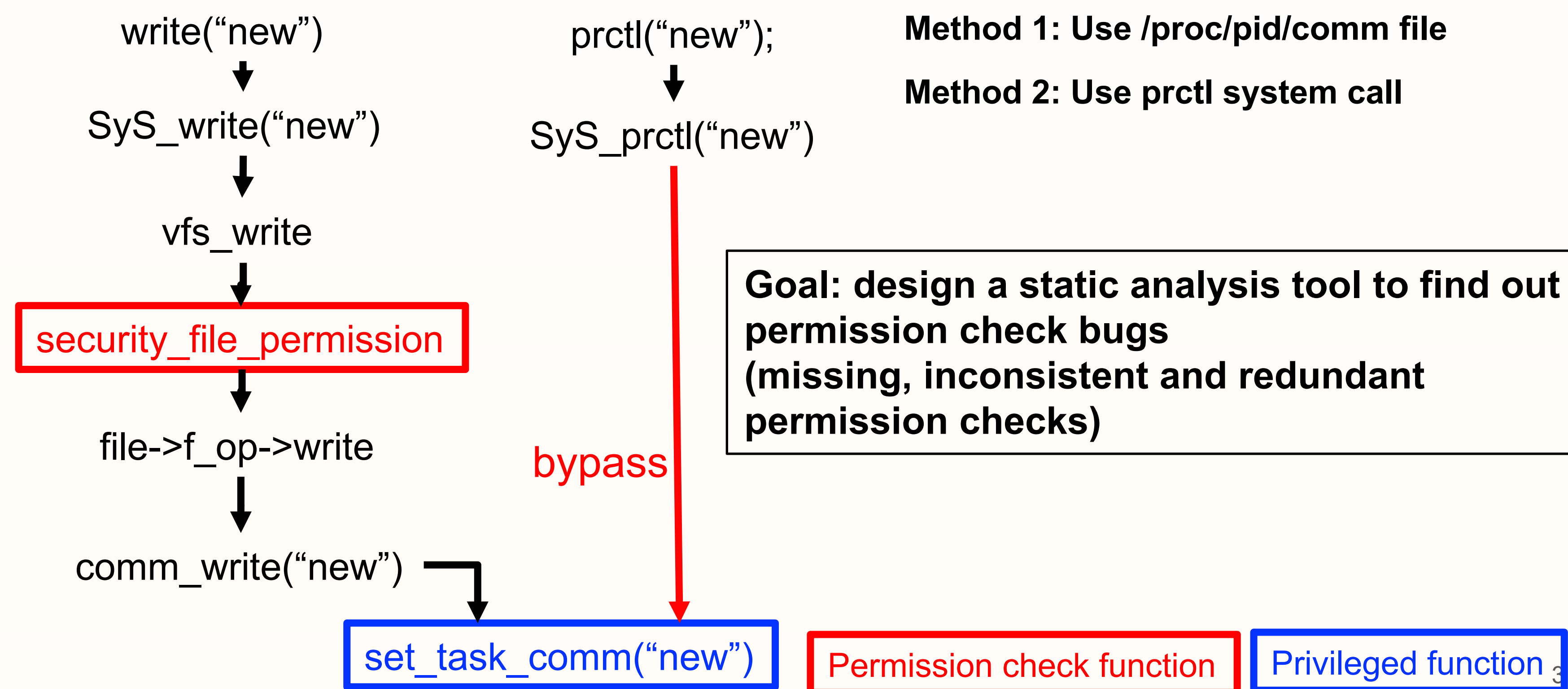
   e.g., SELinux, AppArmor

Many permission checks are placed in an ad-hoc manner,
hard to guarantee all of them are placed correctly

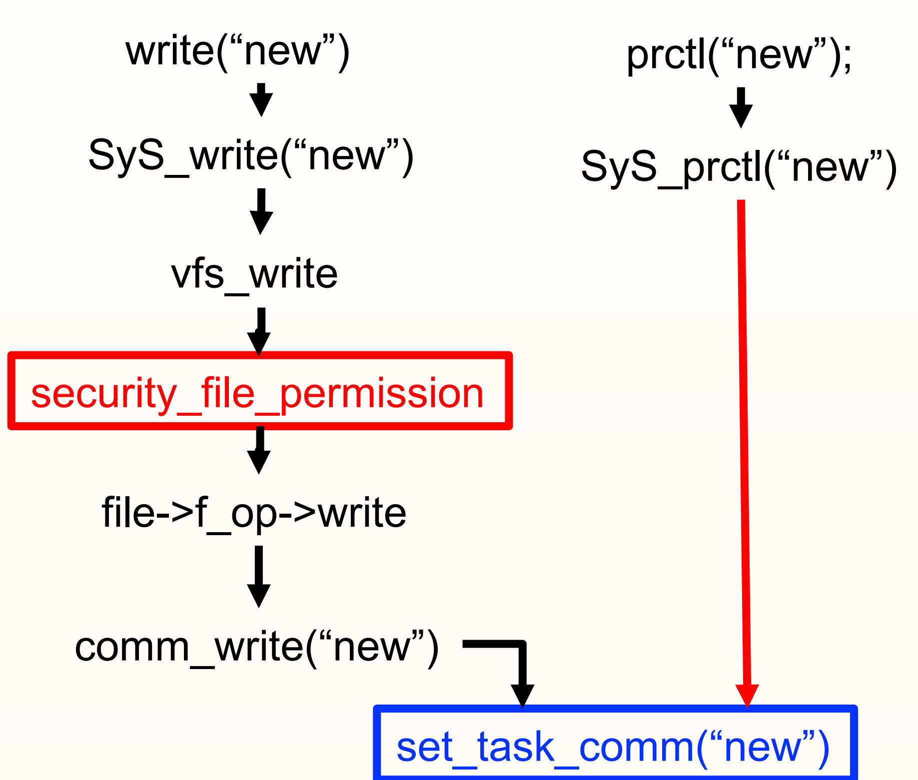
x

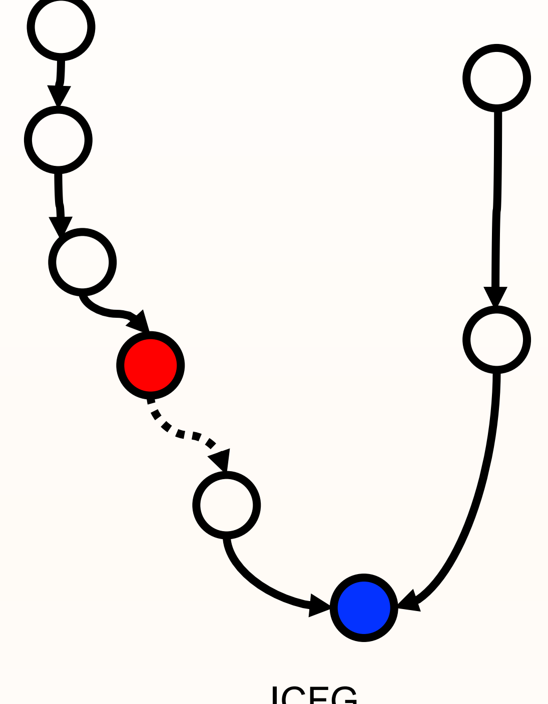
Example: Missing Permission Check is a Problem

Two methods to change a process name



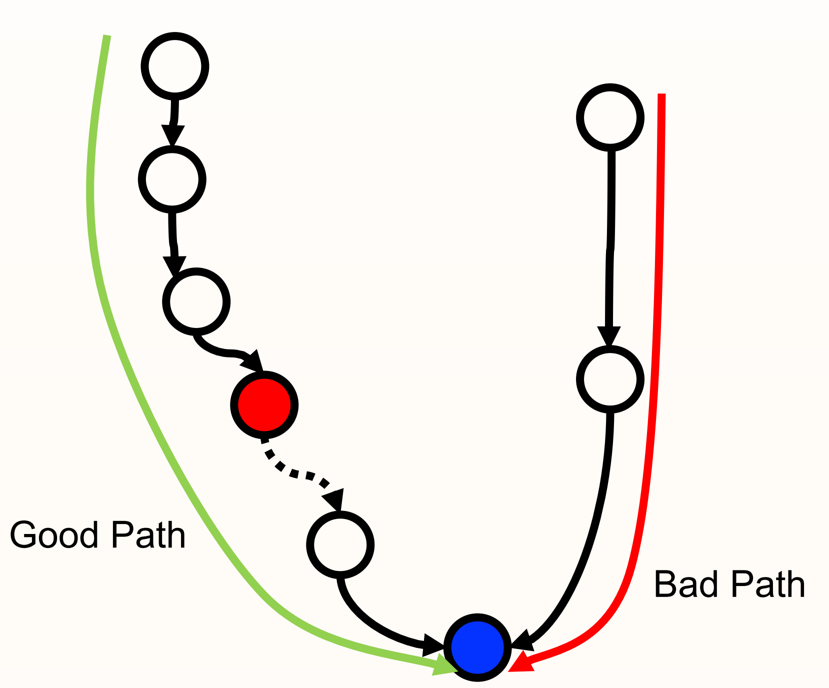
Path Can Be Represented in **I**nterprocedural **C**ontrol **F**low **G**raph





ICFG

Traverse Interprocedural **C**ontrol **F**low **G**raph to Find Bugs



Explore ICFG for all user reachable path to find out bugs

First thing: we need to build an ICFG

Challenge 1: Indirect Calls Makes Precise ICFG Hard to Build

115K indirect callsites in Linux Kernel v4.18.5

Kernel frequently uses function pointer to call real driver implementation

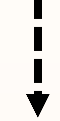
VFS layer

file->f_op->write_iter

- ext4_file_write_iter
- btrfs_file_write_iter
- cifs_file_write_iter
- nfs_file_write

Network Layer

sk->sk_prot->sendmsg



- ipv4 inet_send_msg
- ipv6 inet_send_msg

---► Indirect Call

Challenge 1: No Precise and Scalable Solution

- **Typed based approach (function signature) – imprecise**

```
ssize_t __vfs_write(struct file* file,…)
{
    file->f_op->write(file, p, count, pos);
}
```

✓ **ssize_t Write (struct file *, char __user *, size_t , loff_t *)**

✗ **ssize_t Read (struct file *, char __user *, size_t , loff_t *)**

- **Advanced pointer analysis: not scalable**

  SVF[2](used by K-Miner[1],a static tool kernel analysis)

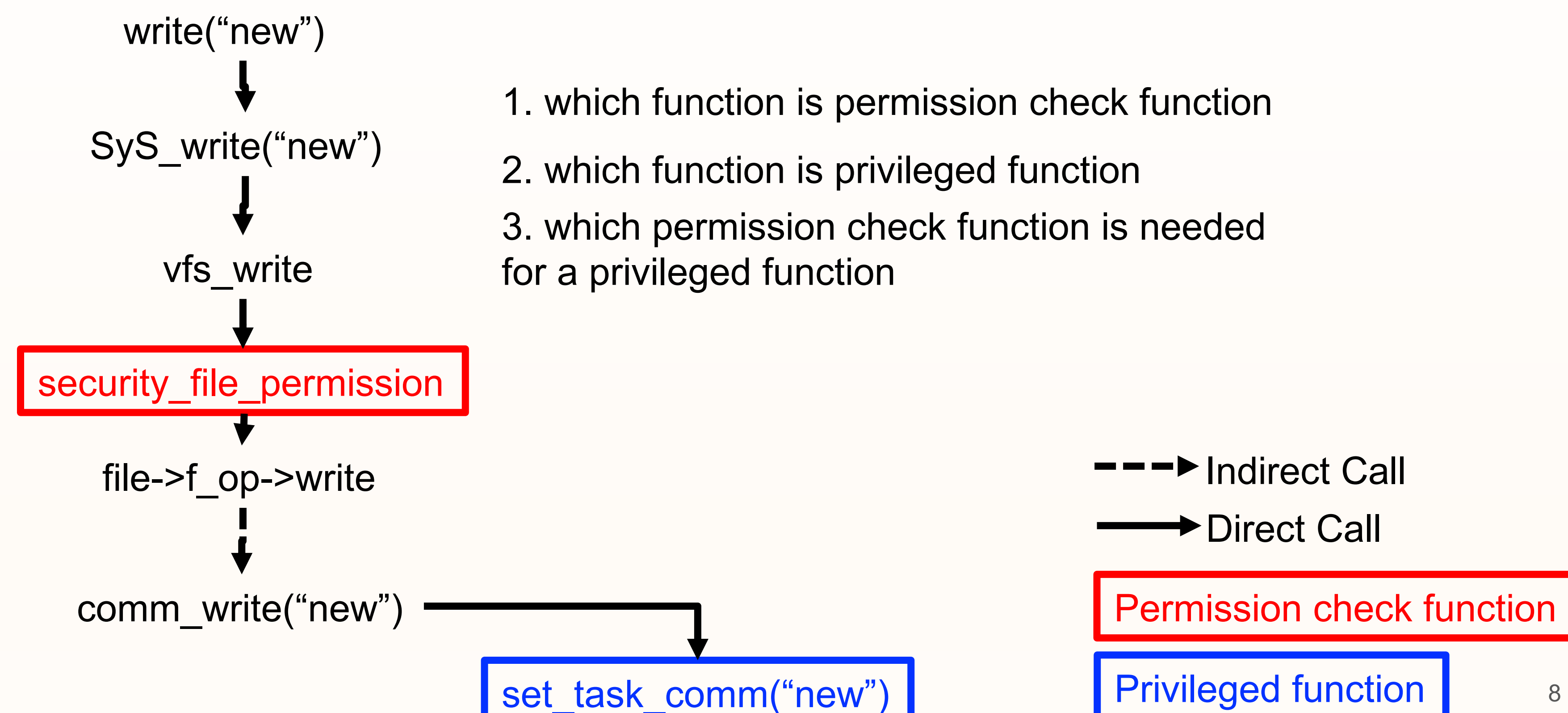
  They do not scale for Linux kernel (~16 MLoC)

  Can be applied to a smaller codebase, which harms soundness

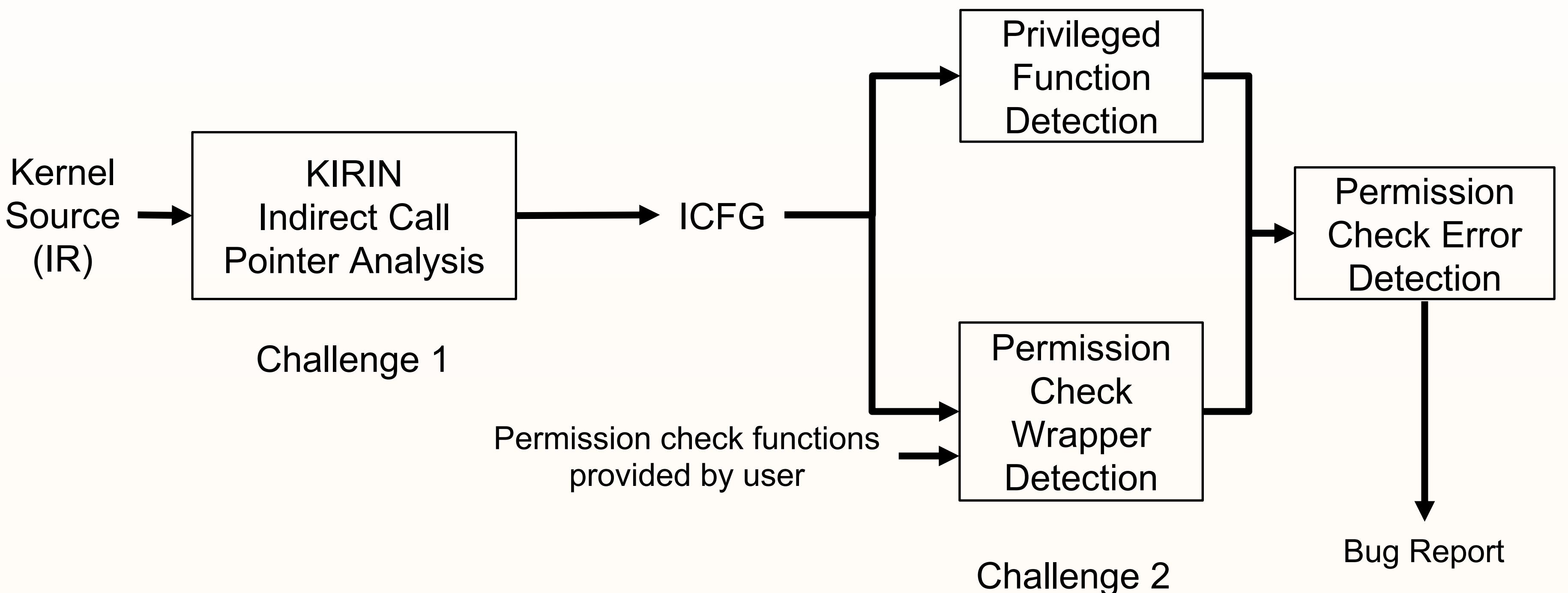
x

K-Miner: Gens, David, et al. "K-Miner: Uncovering Memory Corruption in Linux." NDSS. 2018.
SVF: Sui, Yulei, and Jingling Xue. "On-demand strong update analysis via value-flow refinement." Proceedings of the 2016 24th ACM SIGSOFT international symposium on foundations of software engineering. ACM, 2016.

Challenge 2: Three Other Things We Don't Know



PeX Workflow



KIRIN Observation: Most Indirect Calls(~90%) in Linux Kernel Use Well Defined Interface

Filesystem

```
struct file_operations {
 loff_t (*llseek) (struct file *, loff_t, int);
 ssize_t (*read) (struct file *, char __user *, size_t,
loff_t *);
 ssize_t (*write) (struct file *, char __user *, size_t,
loff_t *);
 int (*open) (struct inode *, struct file *);
 int (*release) (struct inode *, struct file *);
 …
}
```

Network protocol

```
struct proto_ops {
  int (*connect) (struct socket *sock, struct sockaddr
*vaddr, int sockaddr_len, int flags);
  int (*listen) (struct socket *sock, int len);
  int (*sendmsg) (struct socket *sock, struct msghdr
*m, size_t total_len);
  int (*recvmsg) (struct socket *sock, struct msghdr
*m, size_t total_len, int flags);
…
}
```

KIRIN Step 1: Trace and Collect All Struct Initializations

```
file_operations proc_pid_set_comm_operations
{
    .open      = comm_open,
    .read      = seq_read,
    .write     = comm_write,
    .llseek    = seq_lseek,
    .release   = single_release,
}
```

```
…
.write = ext4_file_write
.write = vfat_file_write
…
```

KIRIN trace all statically and dynamically initialized struct

KIRIN Step 2: Match Indirect Call Target Using Interface

Step 2 analyze the callsite

✔ better precision than type-based method

✔ better scalability than SVF because the analys is simpler

```
ssize_t __vfs_write(struct file* file,…)
{
    file->f_op->write
```

Calling write function in struct file_operations

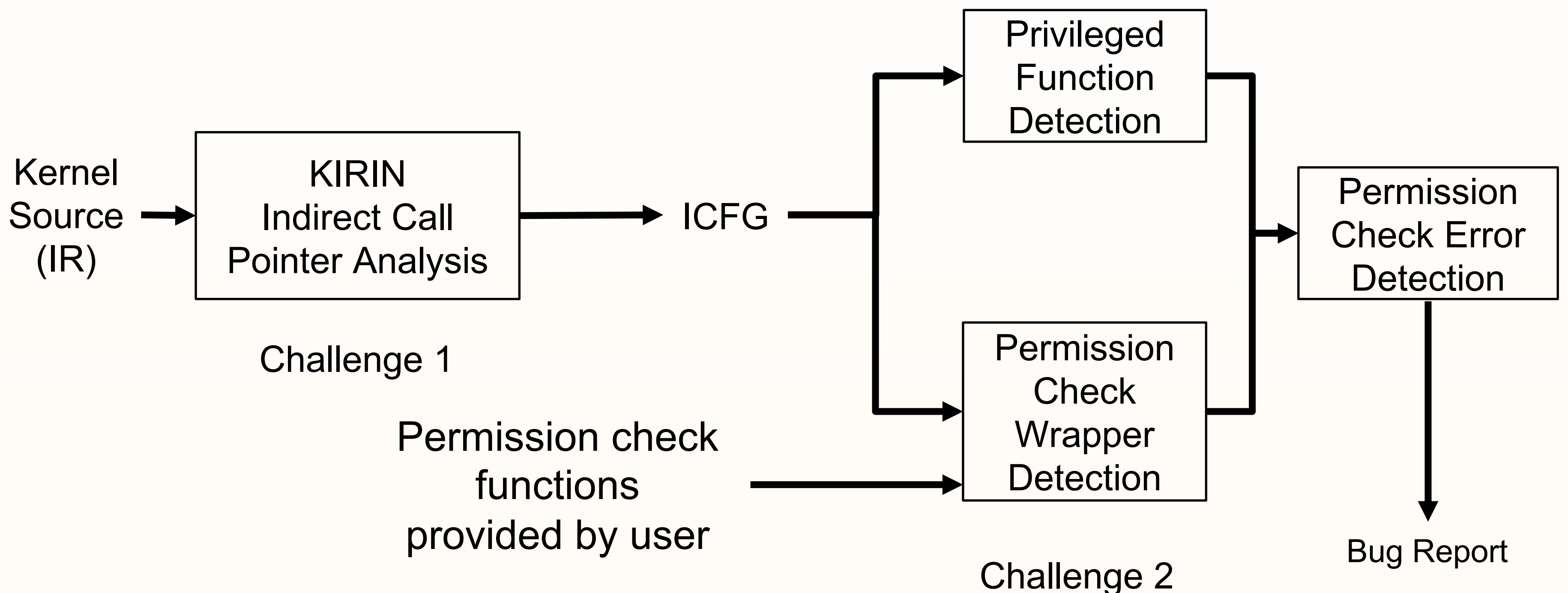
Possible callee: comm_write

1.Match Interface

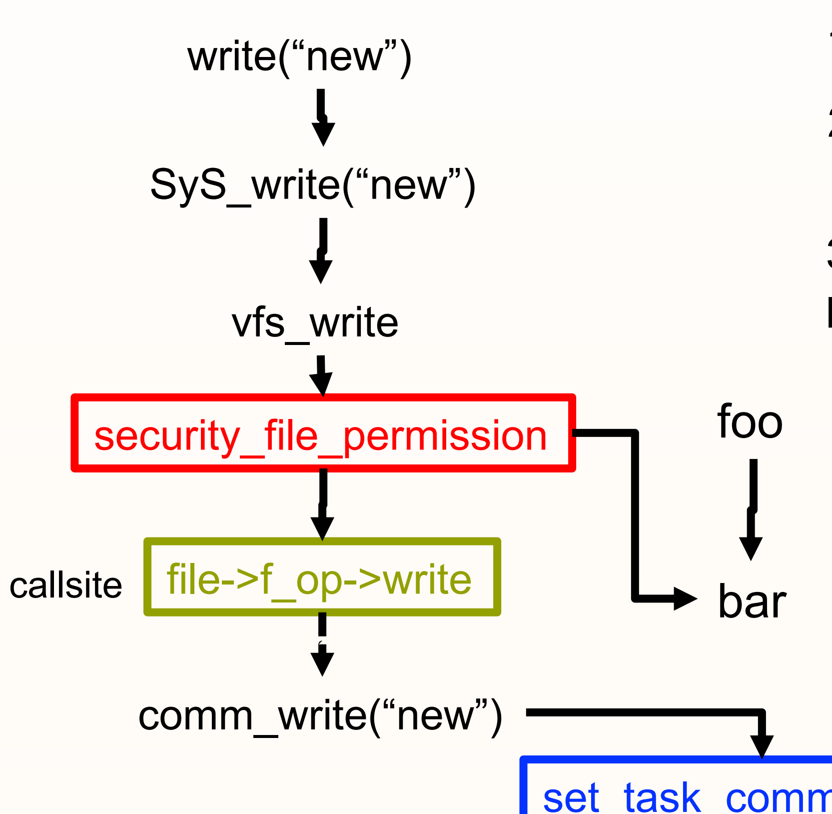
2.Match member

```
struct file_operations {
   …
 ssize_t (*read) (struct file *, char __user *, size_t, loff_t *);
 ssize_t (*write) (struct file *, char __user *, size_t, loff_t *);
   …
}
```

PeX Workflow



Dominator Based Privileged Function Detection



1. User reachable path, starting from system call

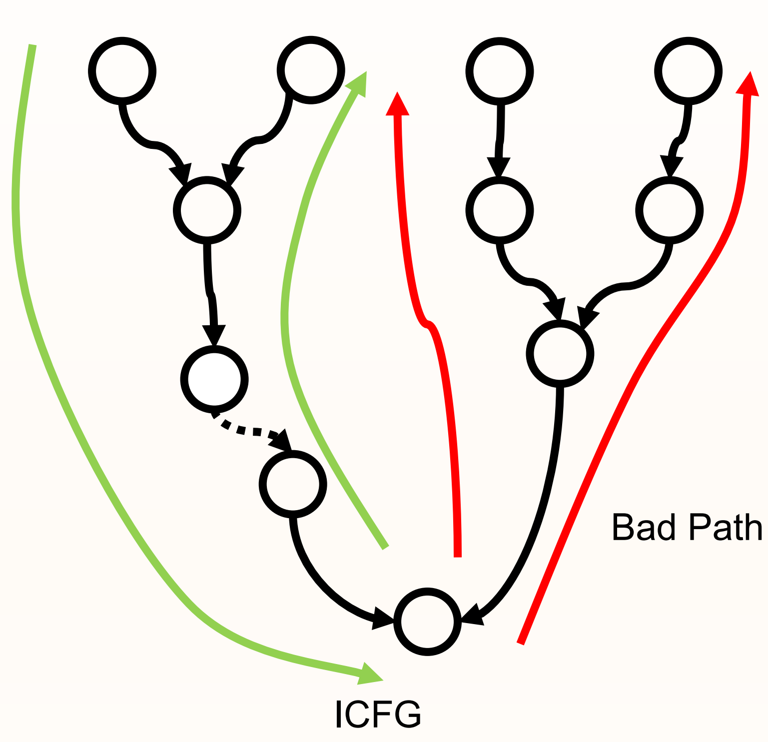
2. A callsite protected by the permission check
 - the dominator analysis

3. Mark callee of the privileged function call as privileged function

Permission check function

Privileged function call

Traverse ICFG for Permission Check Error Detection



1. Traverse ICFG for user reachable path, starting from system call

2. Find a control flow path with no permission check in a backward search manner

x

Implementation and Evaluation

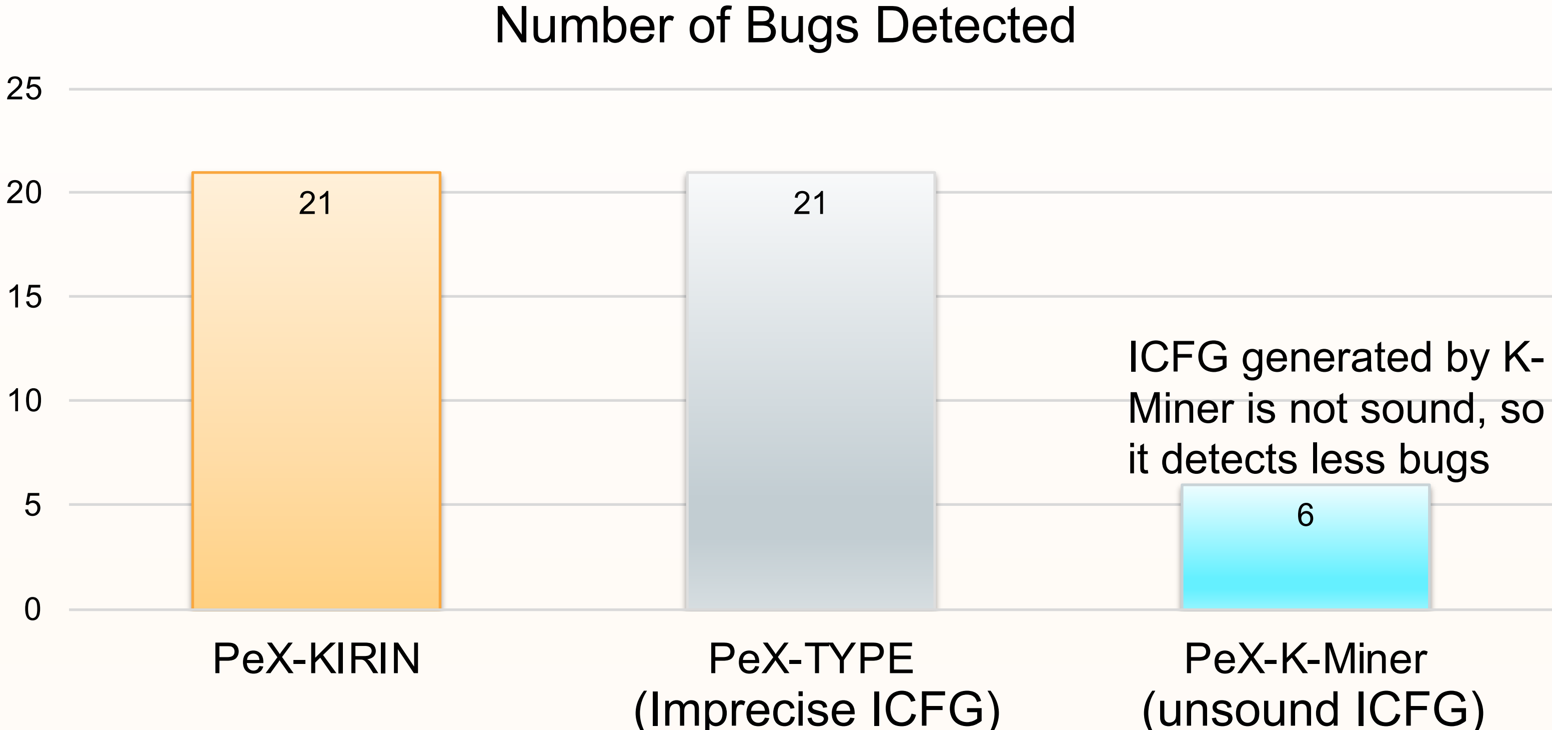
Implementation

- LLVM/Clang-6
- Generate a single-file vmlinux.bc using wllvm

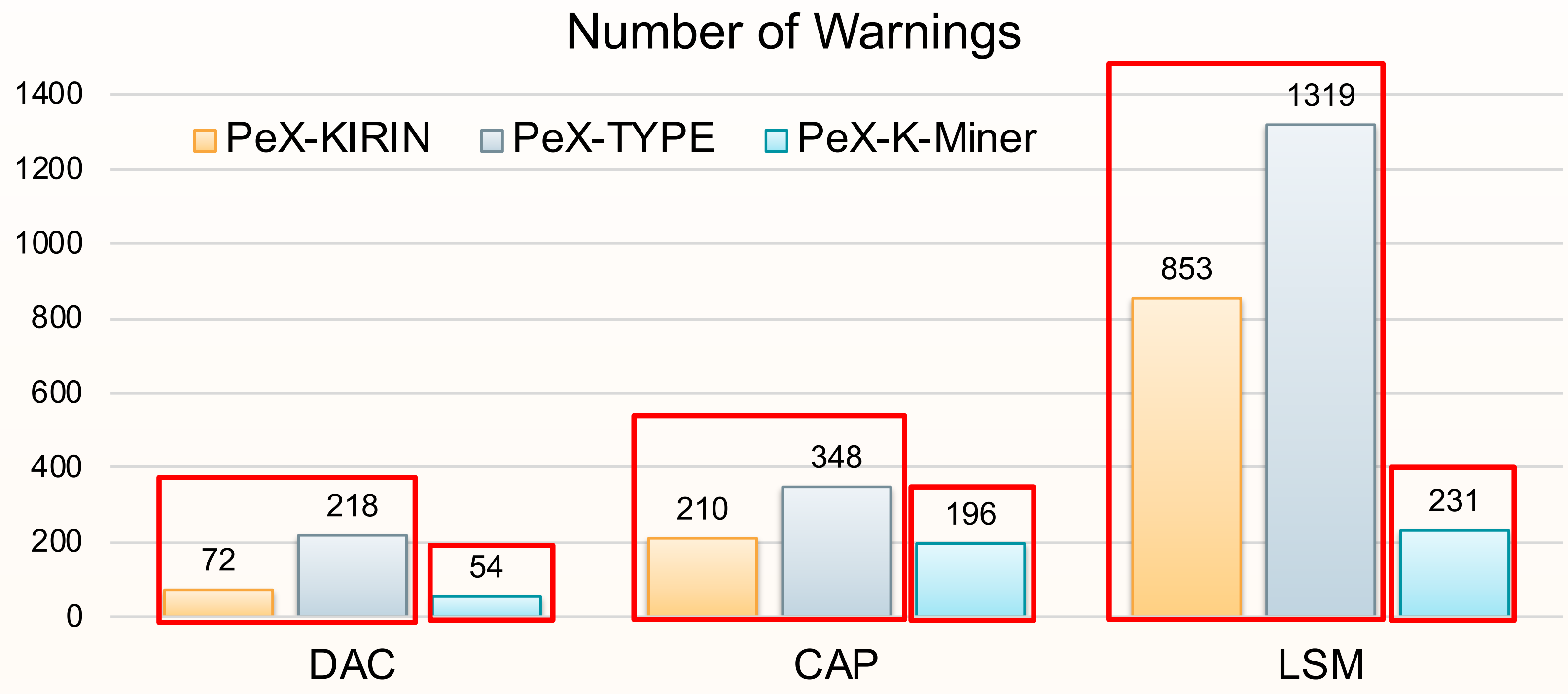
Evaluation

- Linux-v4.18.5
- defconfig(2.4M LoC)
- allyesconfig(15.9MLoC)

Detection Capability – defconfig (2.4M LoC), PeX-KIRIN is Better



Detection Capability – defconfig (2.4M LoC), PeX-KIRIN is Better



1. ICFG generated by KIRIN is more precise than type approach, so it generates less warnings
2. ICFG generated by K-Miner-SVF is unsound, so it generates less warnings

Conclusions

- PeX: a static permission check analysis framework for Linux kernel
- KIRIN: kernel call graph analysis
- Permission check functions/Privileged functions and their mappings
- Evaluated Linux kernel v4.18.5 and found 36 permission check bugs

Thank you !