KU LEUVEN

TU Delft

DistriNet

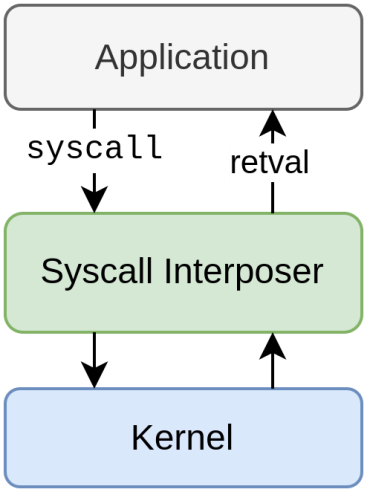
lazypoline: System Call Interposition Without Compromise

Adriaan Jacobs, Merve Gülmez, Alicia Andries, Stijn Volckaert, Alexios Voulimeneas

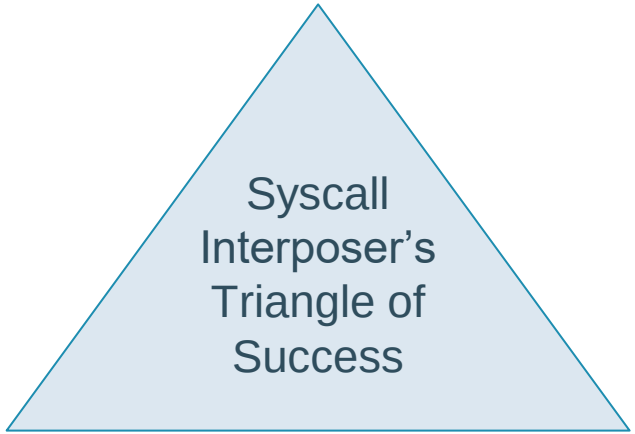
KU LEUVEN GENT

System Call Interposition - What & Why

- Syscalls are the primary way to interface with the OS
- Perform user-supplied function instead of syscall
- Purposes:
  - Monitoring
  - Tracing & Debugging
  - Record & Replay
  - Checkpoint & Restore
  - Virtualization/Emulation
  - Syscall filtering/Sandboxing
  - …



System Call Interposition - Goals

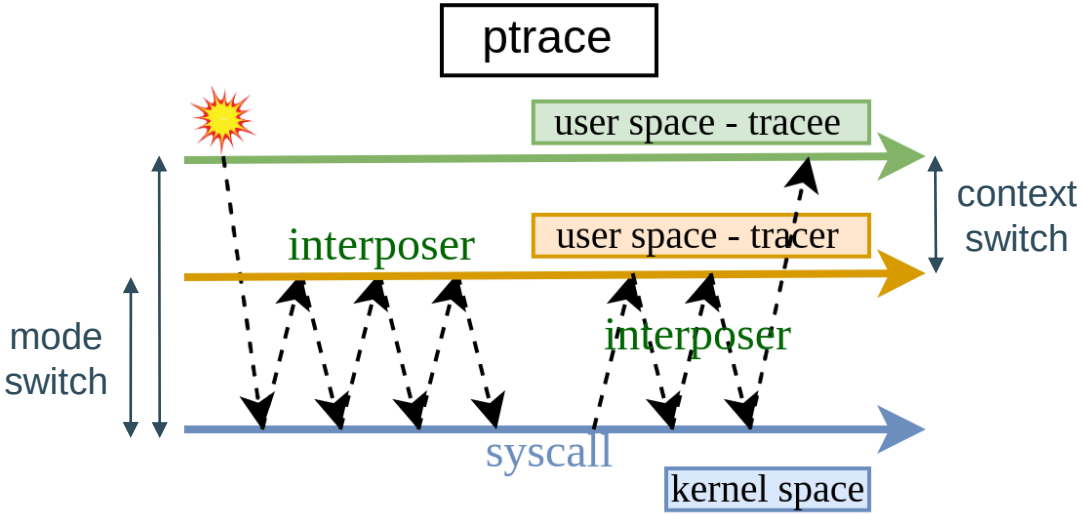


Interposer should have no restrictions on its actions

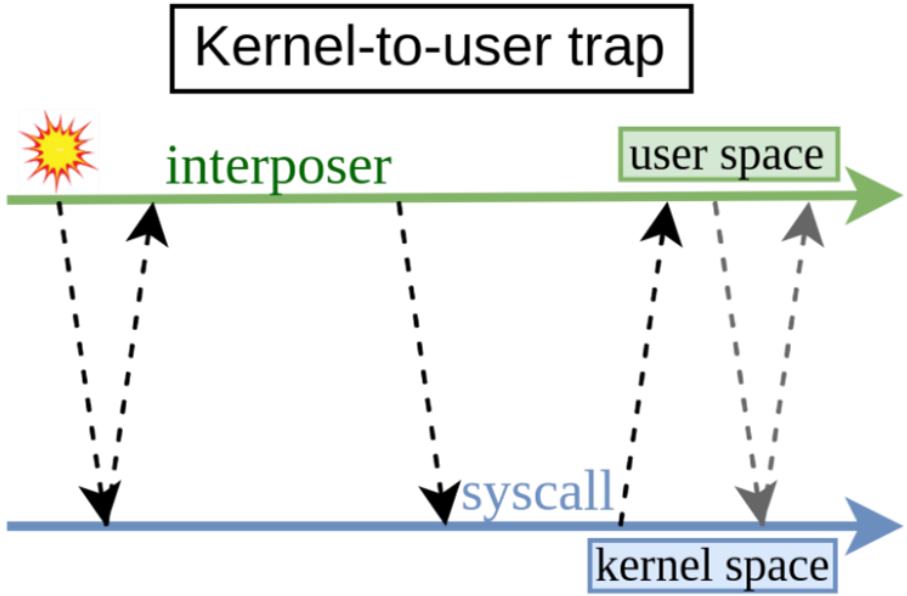
Interposition should minimally impact performance

All syscalls should be interposed

Historically: Linux' `ptrace`

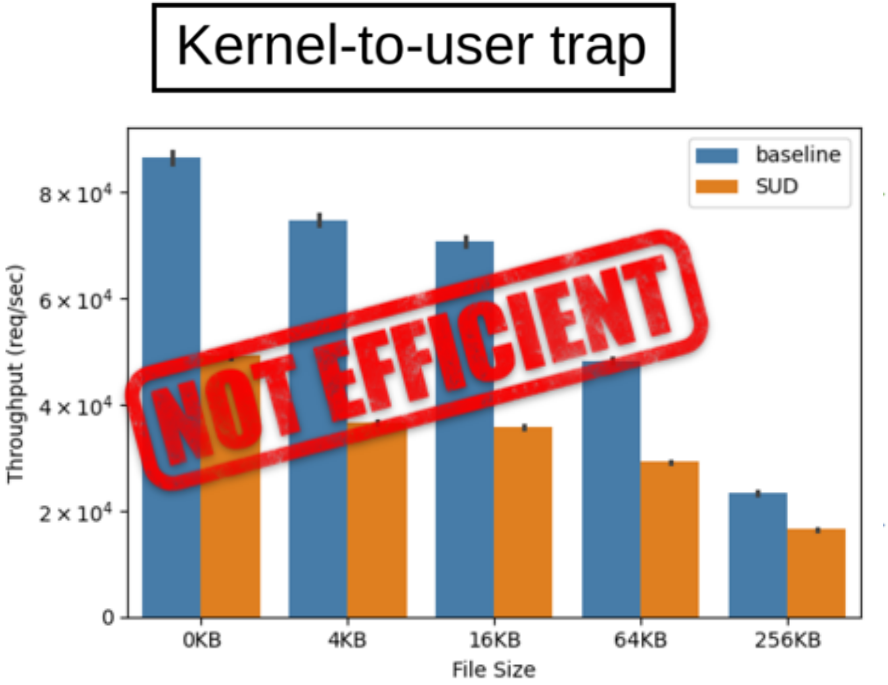


Modern “kernel-to-user trap”



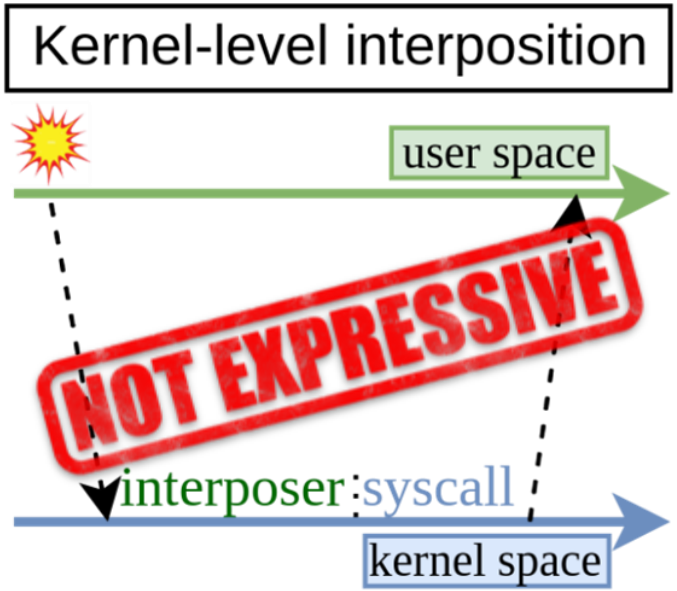
“Syscall User Dispatch (SUD)”

Modern “kernel-to-user trap”

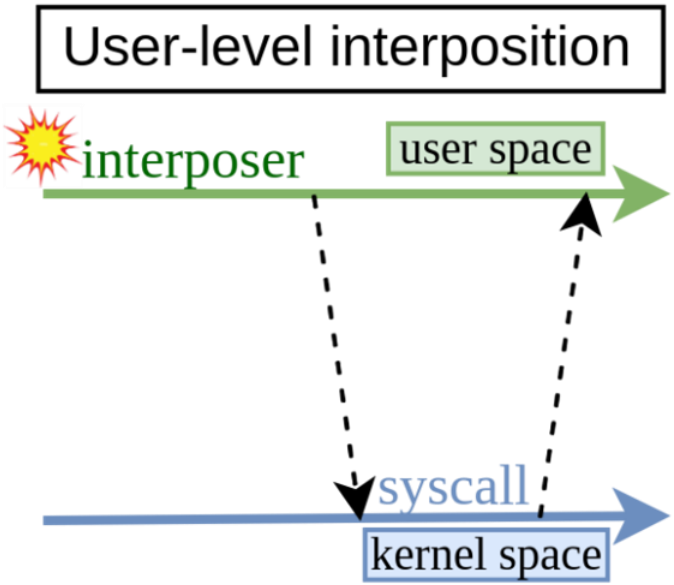


“Syscall User Dispatch (SUD)”

Kernel-level Interposition: LKM/`seccomp-bpf`



User-level Interposition: Binary Rewriting



SaBRe

intel

IIJ
Internet Initiative Japan

KU LEUVEN

User-level Interposition: Binary Rewriting

1. Identify `syscall` instructions
   - Coverage vs Correctness
   - Code vs data
   - Unaligned instructions -> heuristics
   - Obfuscation
   - Dynamically loaded/generated code
2. Rewrite `syscall` instructions
   - Direct `jmp`/`call` > 2 bytes
   - Assumptions about surrounding code
   - *zpoline*!

User-level Interposition: Binary Rewriting

1. Identify `syscall` instructions
   - Coverage vs Correctness
   - Code vs data
   - Unaligned instructions -> heuristics
   - Obfuscation
   - Dynamically loaded/generated code
2. Rewrite `syscall` instructions
   - Direct `jmpcall` > 2 bytes
   - Assumptions about surrounding code
   - *zpoline*!

NOT EXHAUSTIVE

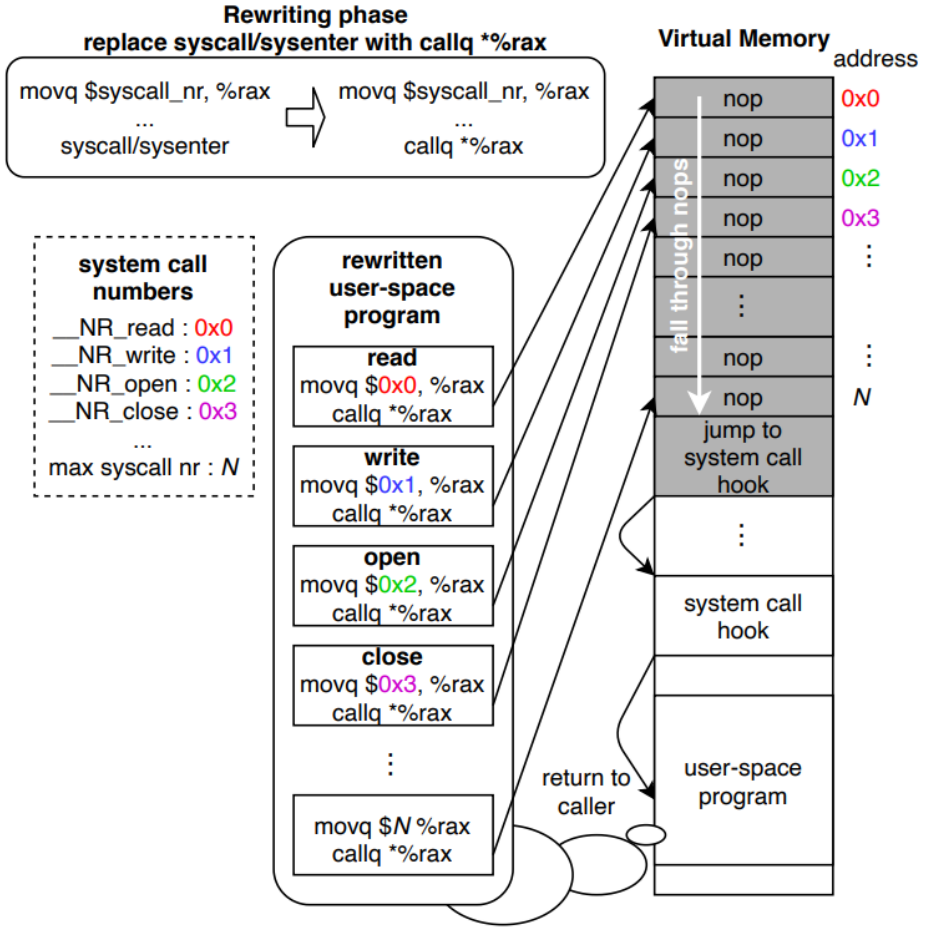
NOT EXHAUSTIVE

zpoline

- USENIX ATC 2023
- "syscall" → "call rax"

usenix
THE ADVANCED COMPUTING SYSTEMS ASSOCIATION

BEST PAPER AWARD



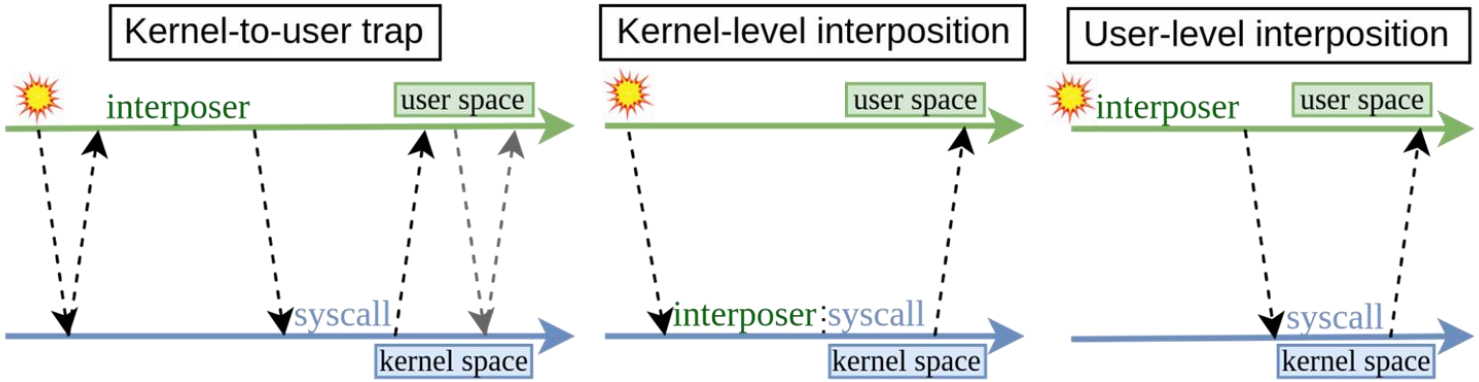
Yasukata, Kenichi, et al. "zpoline: a system call hook mechanism based on binary rewriting." 2023 USENIX Annual Technical Conference (USENIX ATC 23). 2023.

zpoline

- USENIX AT
- "syscall"

Yasukata, Kenichi, et al. "zpoline: a system call hook mechanism based on binary rewriting." 2023 USENIX Annual Technical Conference (USENIX ATC 23). 2023.

State of the Art: Recap



Syscall User Dispatch
seccomp-user

Expressive
Exhaustive
Not Efficient

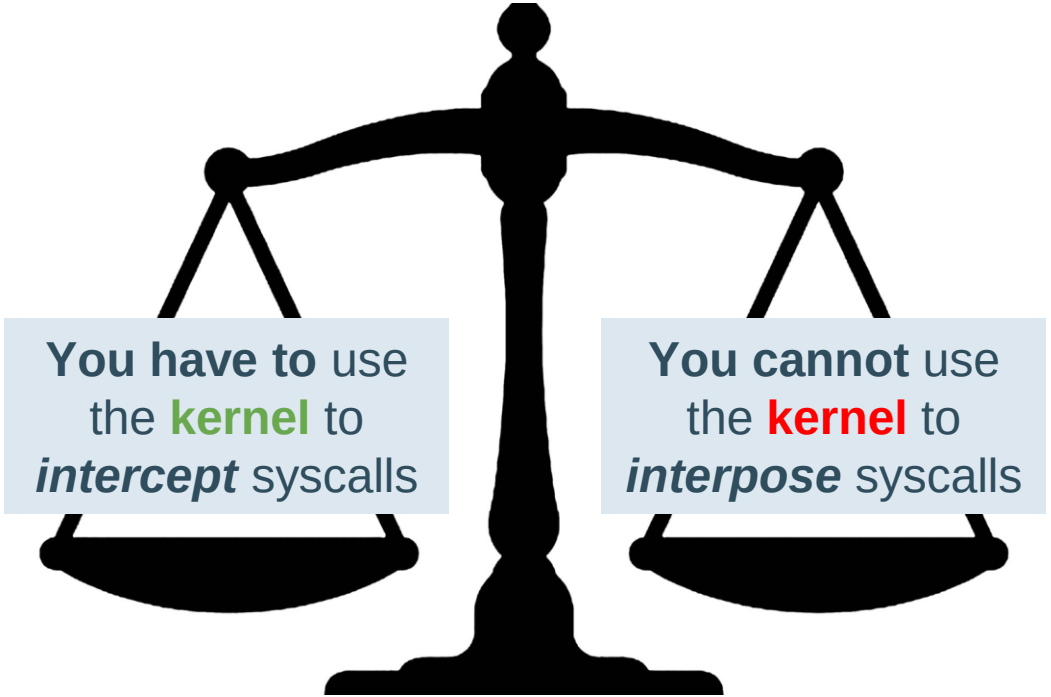
seccomp-BPF

Not Expressive
Exhaustive
Efficient

Binary Rewriting

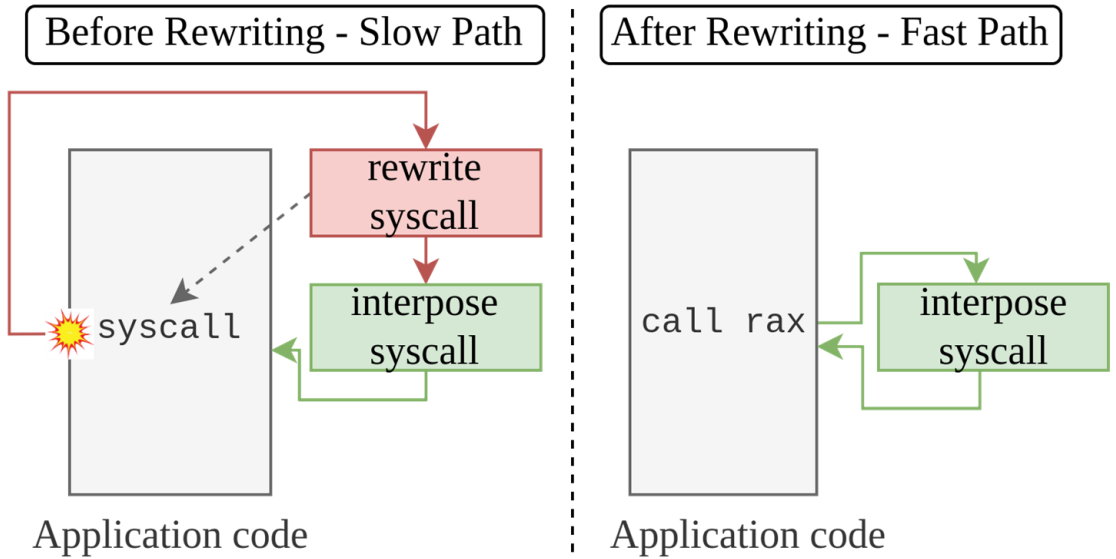
Expressive
Not Exhaustive
Efficient

The Paradox of an Ideal Syscall Interposer



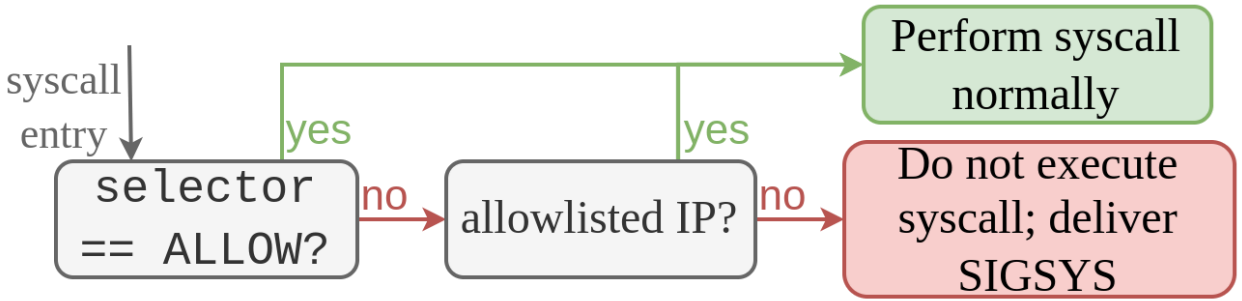
Dynamic Syscall Identification → Lazy Rewriting

1. *Use the kernel at first*: identify syscalls on their first use
2. *Stop using the kernel*: rewrite syscalls *on the fly*



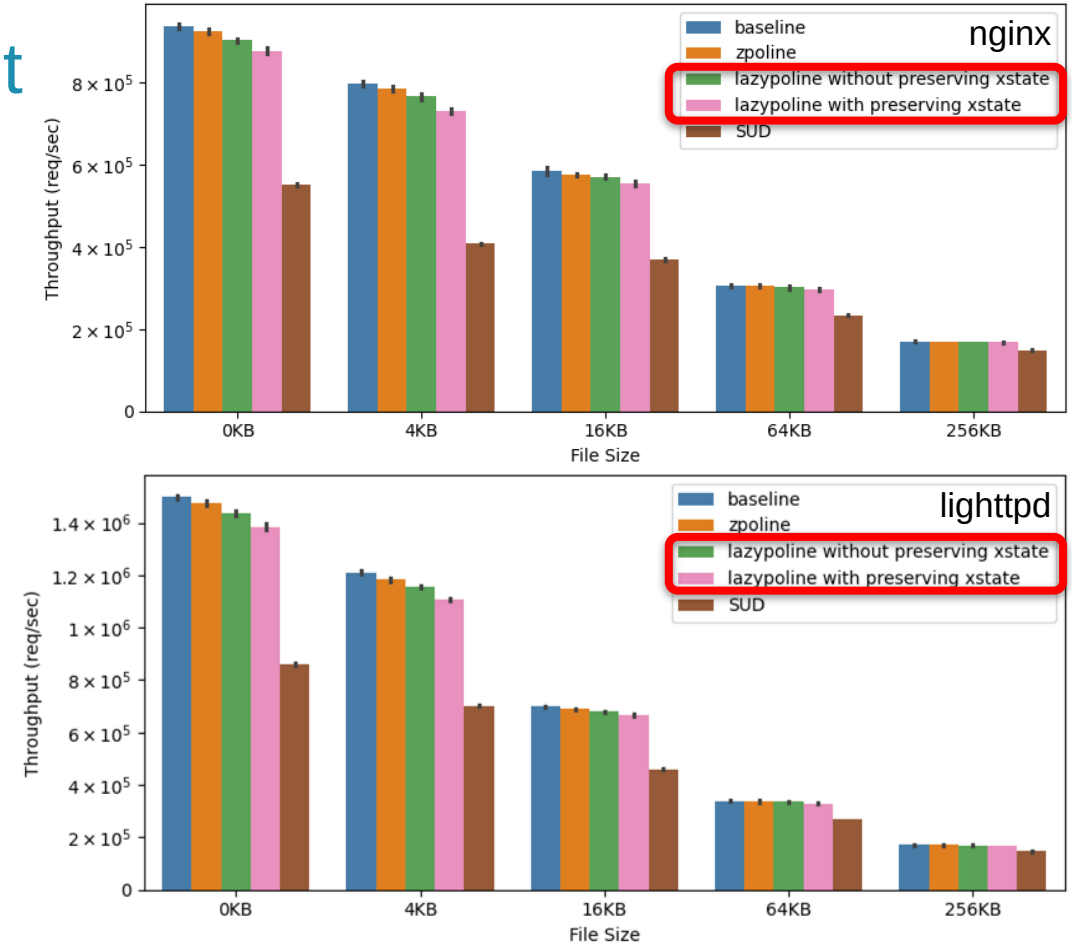
Implementation of Lazy Rewriting With SUD

- Enable SUD via `prctl(SUD_ON, &selector, [[allow_ip_range]])`
  - `selector` and allowlisted `IP` range control interposition
- Rewrite `syscall` from `SIGSYS` handler
- Invoke fast-path entry point



Web Server Throughput Benchmarks

- Maximal throughput setup
- `wrk` client and nginx/lighttpd server communicate over **localhost**
- Baseline throughput maxes out at **1.5M req/s** with 12 workers (lighttpd)



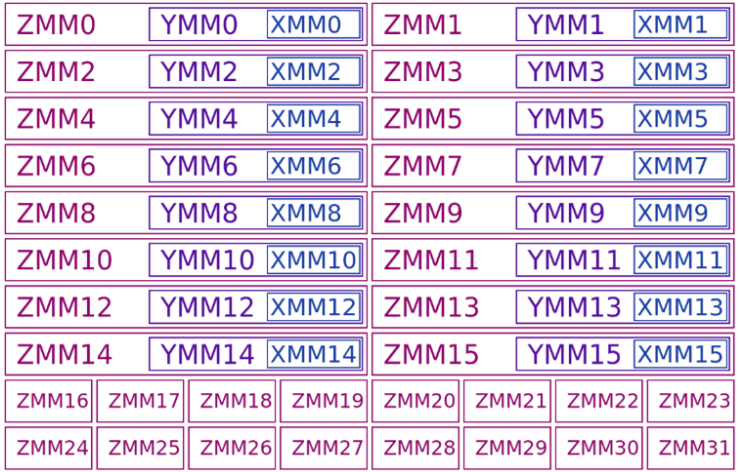
KU LEUVEN

Preserving ABI Compatibility

- `syscall` clobbers `rax`, `rcx`, `r11`
- Preserves everything else

- Binary rewriters preserve all GPRs, but nothing else
- *May clobber* everything else

Current binary rewriters break the syscall ABI.
Hinders expressiveness!



Tracking Register Preservation Expectations

- Intel Pin tool dynamically tracks register read/writes & syscalls
- Problematic pattern:

```
write reg
syscall
read reg
```

Must preserve `reg`

- Very compiler- and arch-dependent

Listing 1: Simplified disassembly of pthread initialization routine. `r12` contains `&__stack_user`, a list of threads with user-provided stacks in use.

```
mov xmm0, r12          ; load into both
punpcklqdq xmm0, xmm0  ; halves of xmm0
; ... irrelevant
syscall                ; set_tid_address
; ... irrelevant
syscall                ; set_robust_list
; ... irrelevant
movups [r12], xmm0     ; write ‘&__stack_user‘
                       ; to ‘prev‘ + ‘next‘
```

Tracking Register Preservation Expectations

- Intel Pin tool dynamically tracks register read/writes & syscalls
- Problematic pattern:

```
write reg
syscall
read reg
```

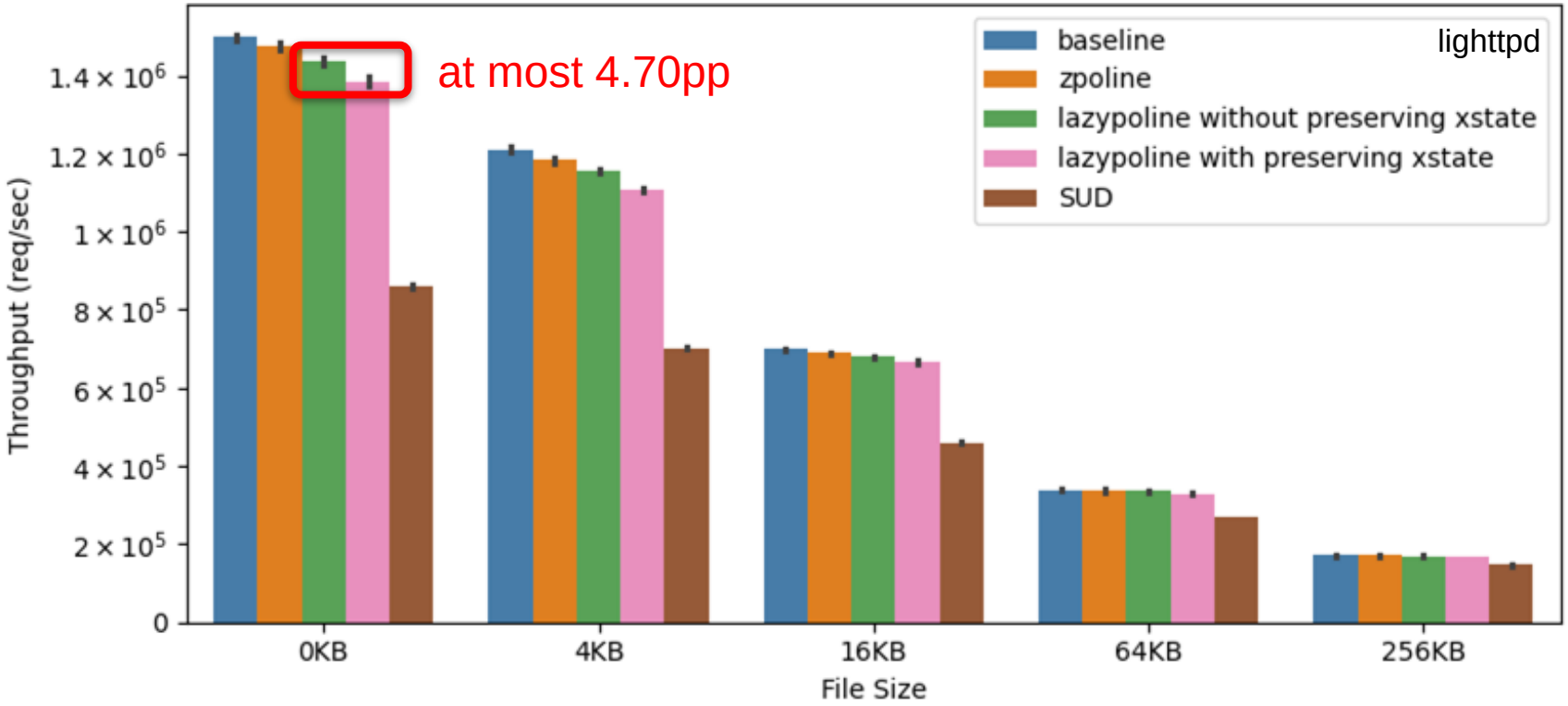
Must preserve `reg`

- Very compiler- and arch-dependent

Coreutils	x86-64v1	x86-64v3
ls	✓	✓
pwd	✗	✓
chmod	✗	✓
mkdir	✓	✓
mv	✓	✓
cp	✓	✓
rm	✗	✓
touch	✗	✓
cat	✗	✓
clear	✗	✓

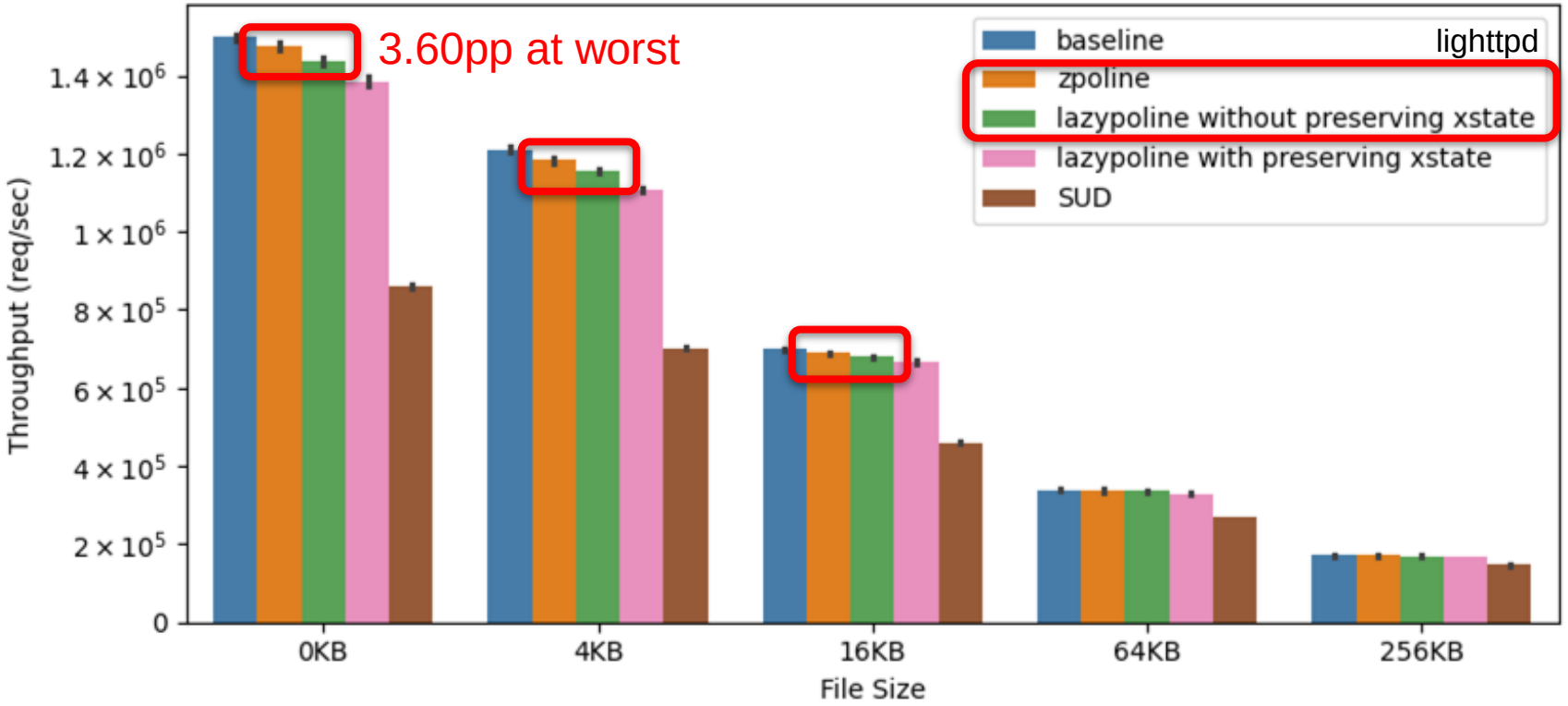
TABLE III: Ten popular coreutils evaluated with our Pin tool on two different Linux distributions. ✓ indicates that the program expected an extended state component to be preserved across at least one syscall. ✗ means we found no such issues.

Web Server Throughput Benchmarks



KU LEUVEN

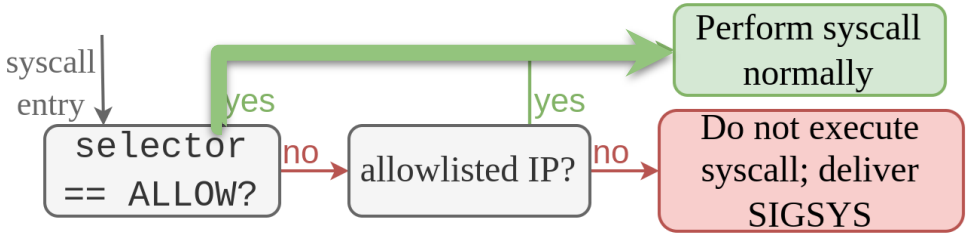
Web Server Throughput Benchmarks



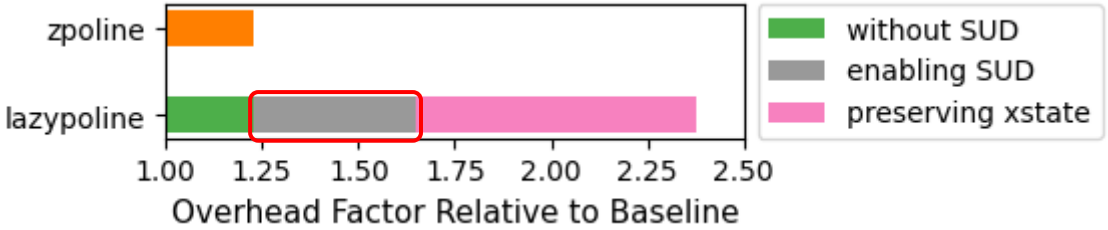
Microbenchmark

- Execute 100M non-existent syscalls (sysno 500) in a tight loop

zpoline	1.23x
lazypoline without `xstate` preservation	1.66x
lazypoline	2.38x
SUD	20.8x
baseline with SUD enabled (`selector=ALLOW`)	1.42x

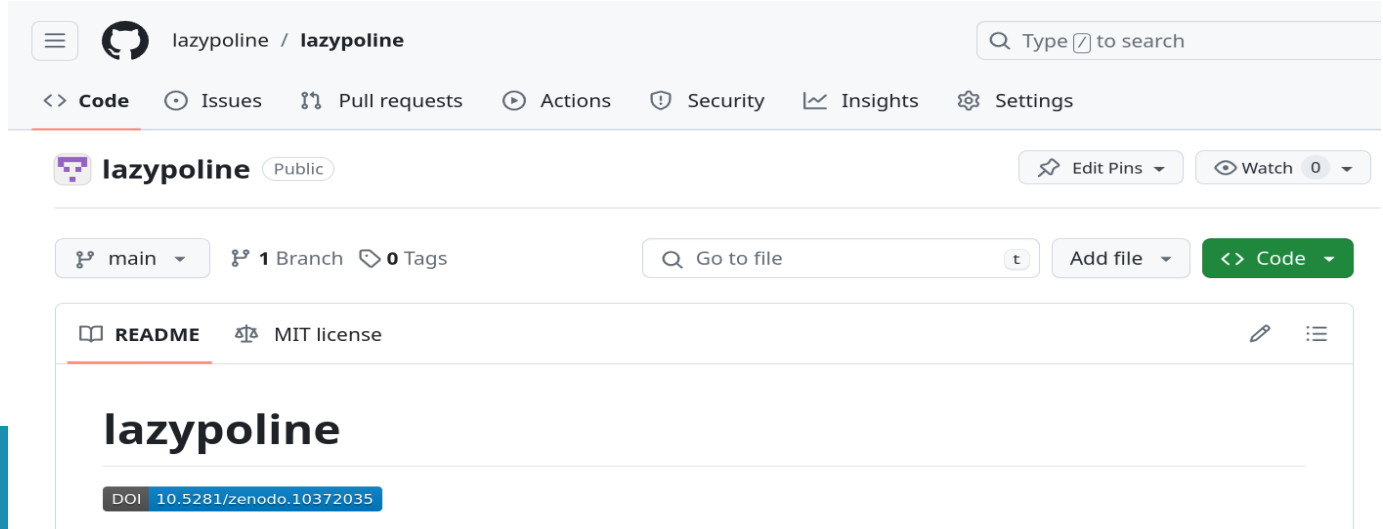


- Finding: SUD *still* adds overhead to permitted syscalls



So What Now?

- We designed the *first* syscall interposer that is simultaneously **efficient**, **expressive**, and **exhaustive** → facilitates new interposition use cases!
- Our interest: *secure* syscall interposition to build **sandboxes**
- Contributions welcome!



KU LEUVEN

lazypoline: System Call Interposition Without Compromise

Adriaan Jacobs *Merve Gülmez* *Alicia Andries* *Stijn Volckaert* *Alexios Voulimeneas*

Check out the code!

Questions?

Read the paper!

KU LEUVEN