APPARE

2026 IEEE Symposium on Security and Privacy

Fine-Grained Kernel Auditing using Augmented Syscall Reference Behavior Analysis and Virtualized Selective Tracing

Chuqi Zhang[§], Spencer Faith[†], Feras Al-Qassas[†], Theodorus Februanto[†], Zhenkai Liang[§], Adil Ahmad[†]

[§] National University of Singapore · [†] Arizona State University

Presenter: Huan Zhao[§]

github.com/ASTERISC-Release/Appare

BACKGROUND

Audit logs are how we investigate breaches.

Auditor records sensitive interactions — process creation, file access, network I/O — for forensics.

TODAY'S AUDIT LOGS SEE

- openat("/etc/file")
- setsockopt(...)
- write(5, 0xFF00.., 400)
- close(5)
- execve("/bin/sh")

OS services. Coarse-grained. One log entry per syscall.

GRANULARITY

syscall

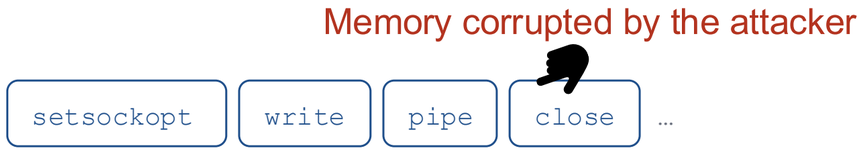
The kernel runs ~41,000 functions. Audit logs see ~400 syscall entry points.

THE PROBLEM

Kernel-level exploits happen inside syscalls.

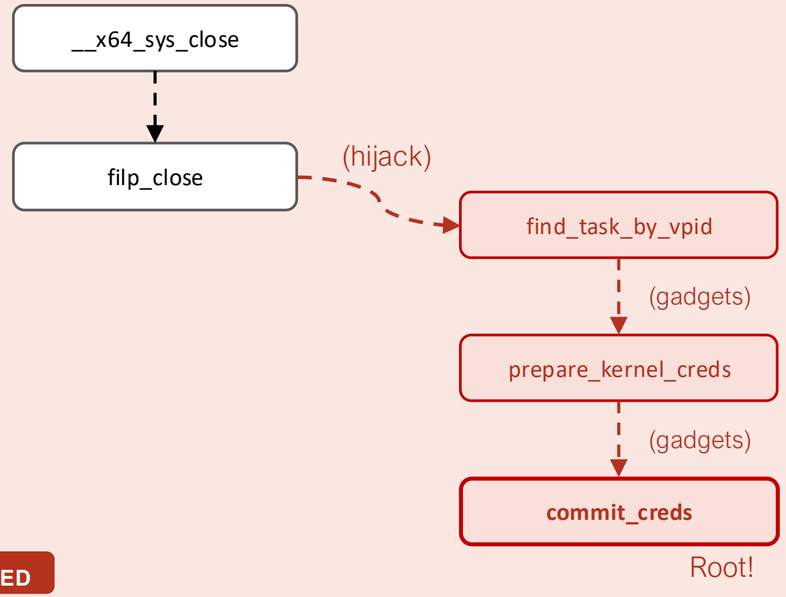
Control-flow hijacks chain in-kernel functions to escalate privilege.

What Auditd (syscall logs) records



An I/O sequence. Cannot infer anything in memory.

What actually executes inside close()



NEVER LOGGED

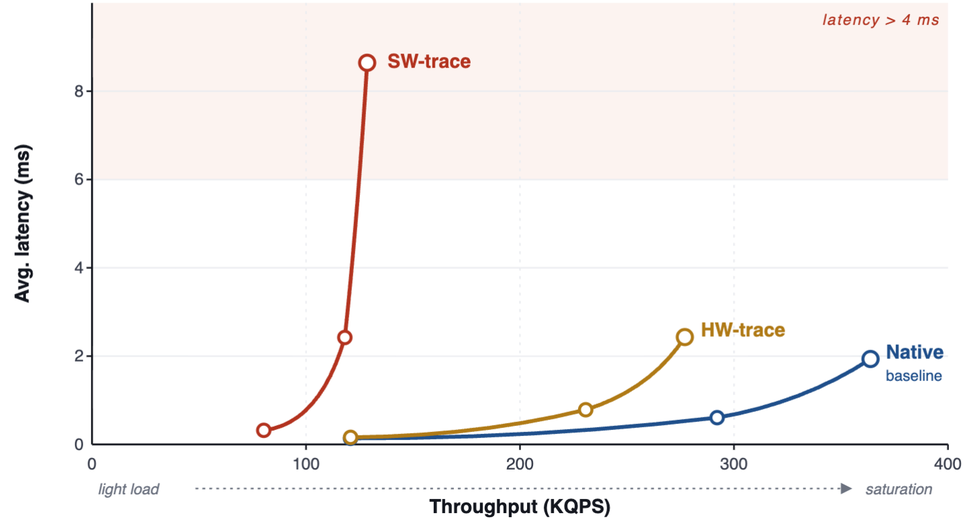
Privilege escalation — control-flow entirely in-memory, entirely invisible.

STRAWMAN SOLUTION

For every syscall, we trace all executed kernel functions.

Blindly tracing all collapses both performance and storage!!

Performance cost (AVG. across Nginx & Redis)



Storage cost (AVG. across Nginx & Redis)

Produces **10× more** data (benign traces drown the only few “anomalous” ones).

SW-TRACE

Linux ftrace — instrument every kernel function entry in software.

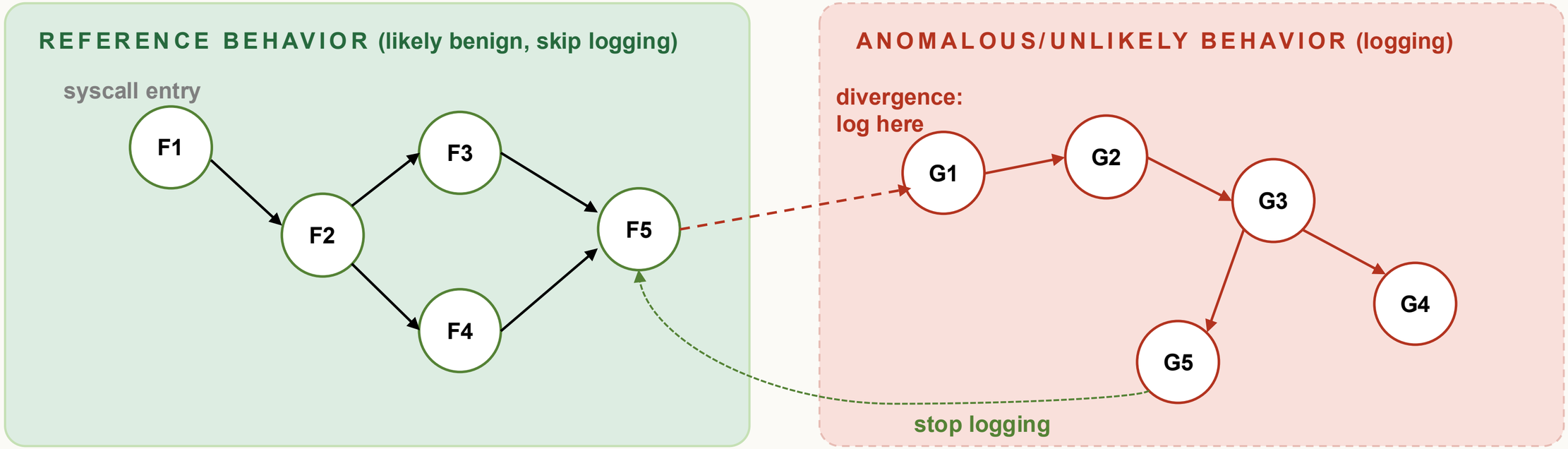
PT-TRACE

Intel Processor Tracing (PT) — hardware-recorded branch packets.

OUR INSIGHT

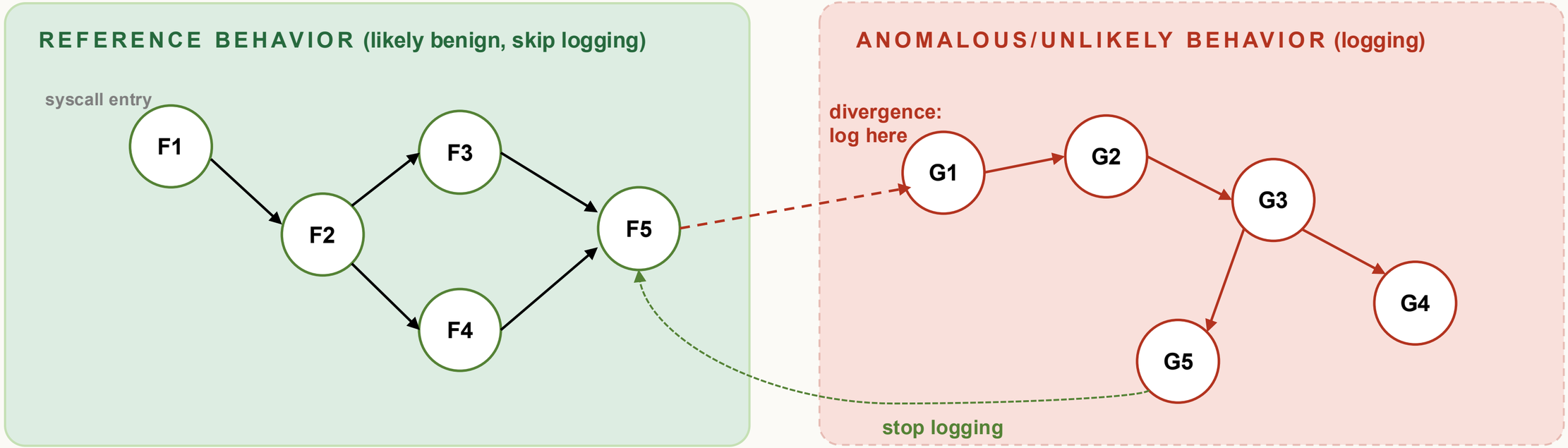
Only trace what falls outside a syscall's reference behavior.

Most paths during a syscall are benign and repetitive. If we know what those *should* be, we only log when execution leaves them — and divergences are rare.



CHALLENGE

Two challenges.



1 How to determine the reference behavior of a syscall?

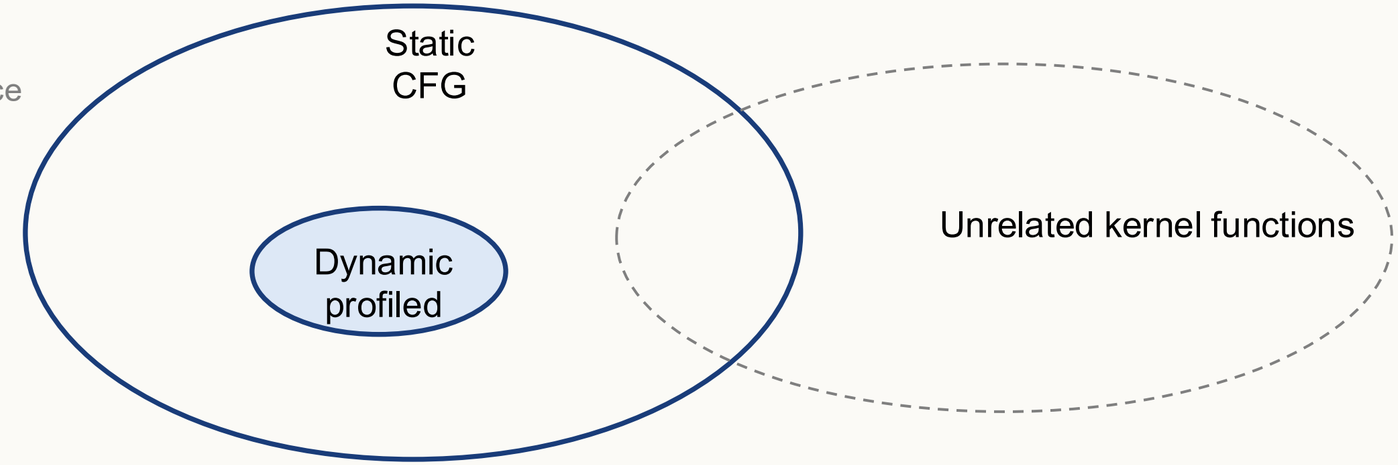
2 How to selectively trace divergences efficiently & securely?

1 REFERENCE BEHAVIOR

Challenge 1: determine reference behavior.

Neither dynamic- nor static-only analysis alone gives the good set.

Derive a syscall's reference behavior:



Dynamic approach ONLY

Under-approximation

Use **some program workloads** to profile actually executed functions in a syscall as its reference behavior.

Static approach ONLY

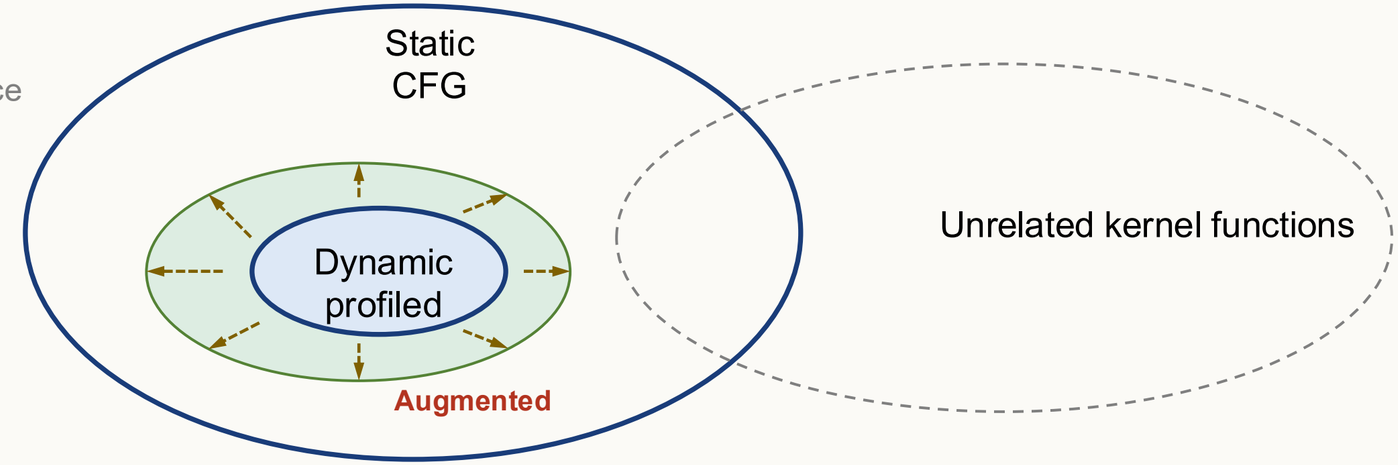
Over-approximation

Use a static CFG to include functions **reachable by a syscall entry** as its reference behavior.

1 INSIGHT

Augment dynamic profiles with LLM semantic reasoning.

Derive a syscall's reference behavior:

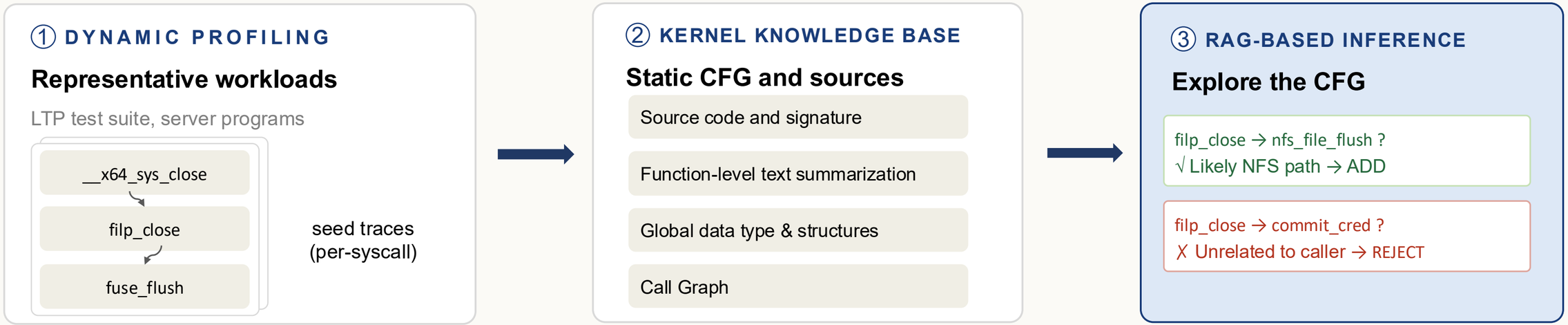


💡APPARE: Program workloads as **seeds,** kernel's static code as a knowledge base -- for proper **context engineering.**

Use LLM's capability to reason about the benign seeds and expand the set.

1 AUGMENTED REFERENCE BEHAVIOR ANALYSIS

Profile seeds. Expand them with RAG-driven LLM reasoning.



2 CHALLENGE

Challenge 2: selective and secure tracing.

EXISTING HW TRACER (ALL TIME TRACING)

Intel PT	hardware, +27% slowdown

No configuration supports function-level selectivity

AND ONCE THE KERNEL FALLS…

The attacker can disable tracing, flush in-memory PT buffers, and clear logs on disk before they're persisted.

In-kernel tracing cannot trust an exploited OS!

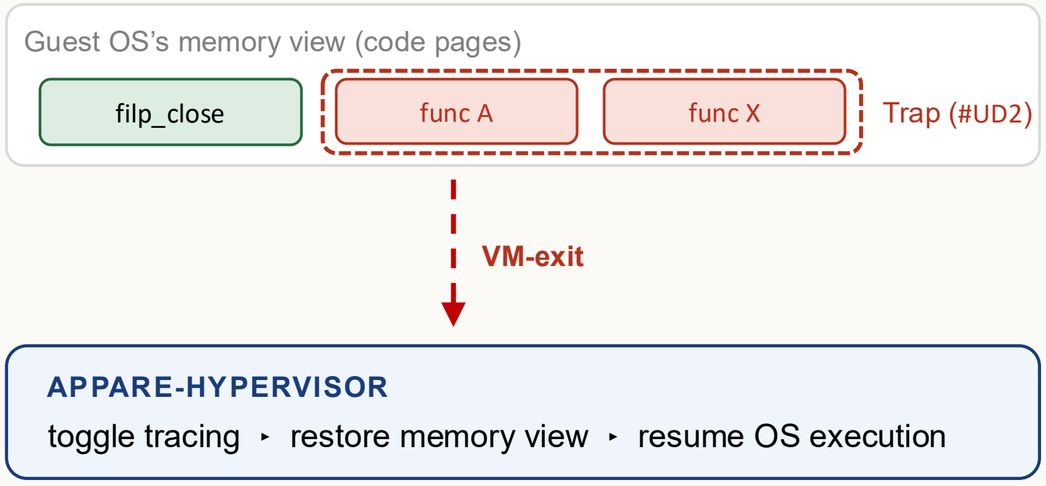
We need **selective tracing toggles** that the kernel can't disable, on **trace buffers** the kernel can't touch.

2 INSIGHT

A hypervisor already has the two primitives we need.

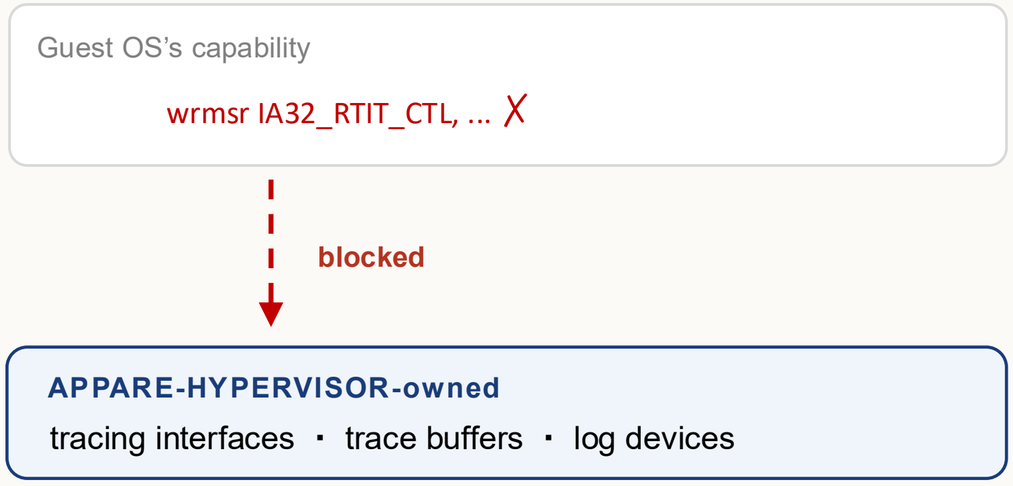
Transparent to the guest — it cannot disable or bypass either.

① TRAP MEMORY (EXECUTABLE PAGES)



→ **Function-level selective toggle**

② PROTECT TRACER CONTROL STATE



→ **Protected (tamper-proof) tracing**

Replace non-reference functions with #UD2. Trap. Trace.

EPT-protected trace buffers

PT interface protection

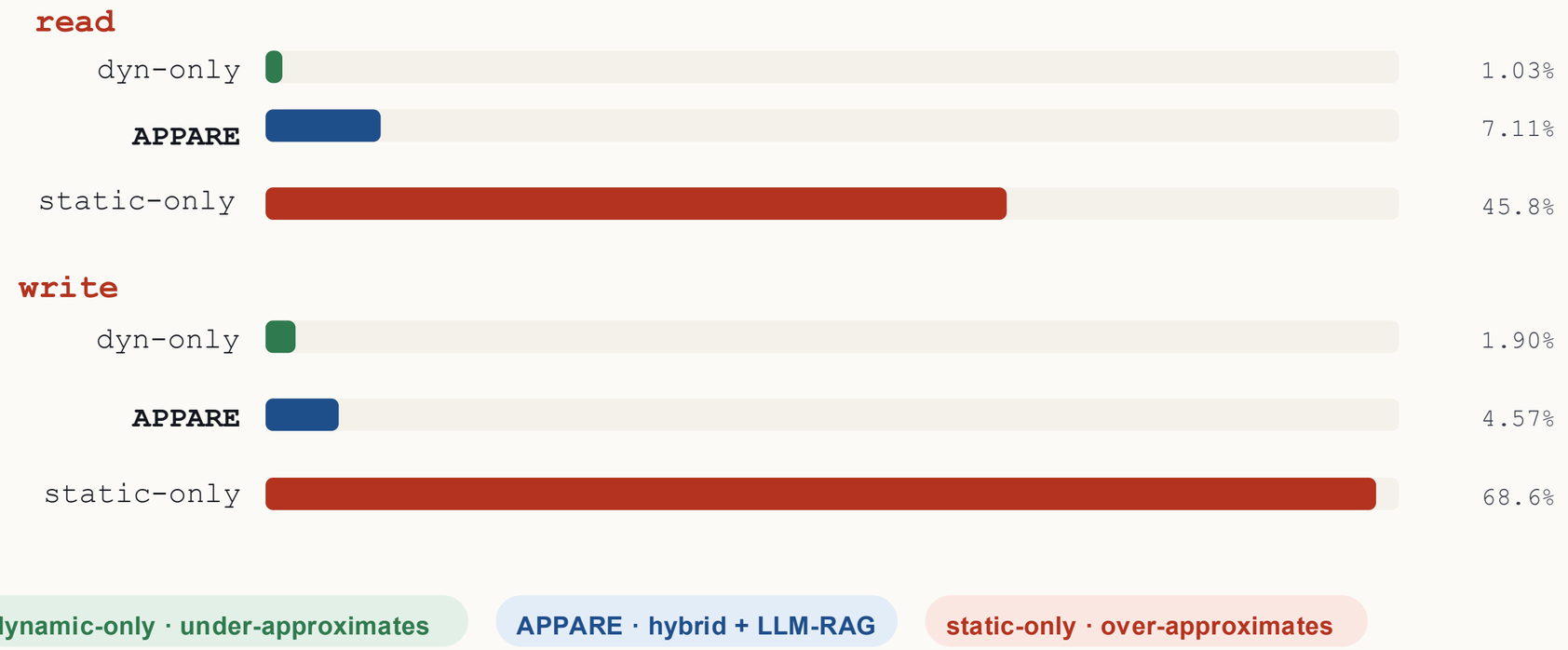
Protected log disk device

VMFUNC for exitless cross-syscall restricted view-switch

EVALUATION · REFERENCE SET SIZE

APPARE keeps reference sets tight.

Selected syscalls; % of all kernel functions included in the reference set.



HEADLINE

75.1%

smaller reference set than naive static expansion (avg).

2.83×

more coverage than dynamic profiling alone.

83.7%

LLM inference accuracy with RAG (vs 69.7% without).

EVALUATION · REAL-WORLD EXPLOIT ANALYSIS

Logged the in-memory chain in real-world kernel CVEs.

11 randomly-selected CVEs covering OOB write, use-after-free, and double-free.

CAPTURED EXPLOIT FUNCTIONS

CVE-2021-22555	commit_creds, switch_task_namespaces ✓
CVE-2022-0185	find_task_by_vpid, commit_creds ✓
CVE-2017-6074	native_write_cr4, commit_creds ✓
CVE-2023-2598	call_usermodehelper_exec, queue_work ✓
CVE-2021-4154	prepare_kernel_cred, commit_creds ✓
CVE-2022-25636	prepare_kernel_cred, commit_creds ✓
… 4 more	all logged ✓
CVE-2022-1015	interrupt context — not yet supported ✗

EXPLOITS WERE CAUGHT

10/11

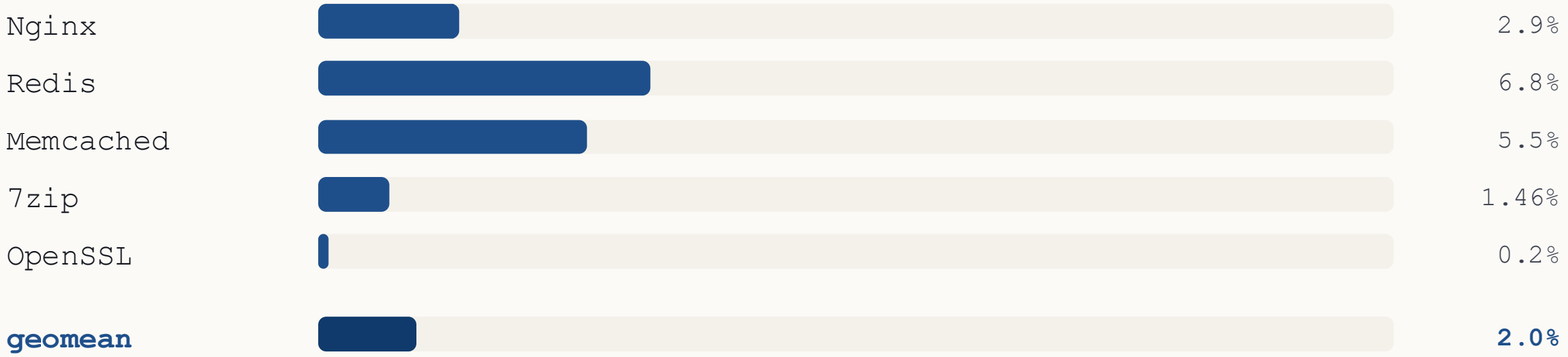
The missed CVE executes in interrupt context, which our prototype currently ignores.

EVALUATION · PERFORMANCE

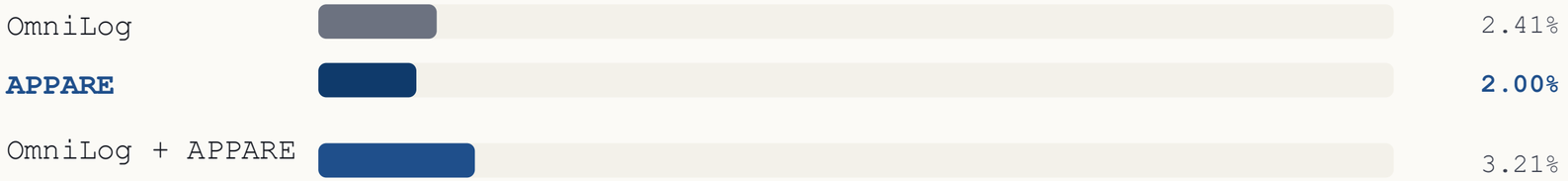
Geomean overhead: 2.0%.

Comparable to syscall-only audit systems — and we log richer data.

RUNTIME OVERHEAD PER APP (APPARE VS NATIVE)



RUNTIME OVERHEAD VS SOTA SYSCALL-AUDIT SYSTEMS (GEOMEAN)



(OmniLog: state-of-the-art syscall-level auditor)

TAKEAWAY

2.0% geomean overhead — richer logs, similar cost as syscall-only auditing.

STORAGE

Nginx full-PT . . 134 MB/s

APPARE . . . . . 4.1 MB/s

~91% less than trace-everything.

TAKEAWAYS

APPARE in three lines.

01 **Audit logs need to see inside syscalls.**

Control flow hijacking-based kernel exploits live in memory; coarse syscall logs miss them by design.

02 **Hybrid profiling + LLM-RAG gives a tight, generic reference behavior.**

A small set of representative workloads, expanded by semantic reasoning over the kernel knowledge base.

03 **Virtualization-aided #UD2 traps make tracing both selective and tamper-proof.**

Per-syscall restricted code views, EPT-protected buffers, asynchronous persistence to a hypervisor-protected log disk.

10/11 real-world CVEs captured | 2.0% geomean overhead | Open source · github.com/ASTERISC-Release/Appare

Thank you.

Questions?