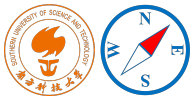
MirrorShield: Cross-Architecture Linux Malware Analysis via Lightweight Emulation and LLM

Zihan Zhang, Wenqi Lin, Lingna Sun, Hongyi Wang, Jingyi Wang, Yanshu Mei, and Fengwei Zhang

Department of Computer Science and Engineering
Southern University of Science and Technology

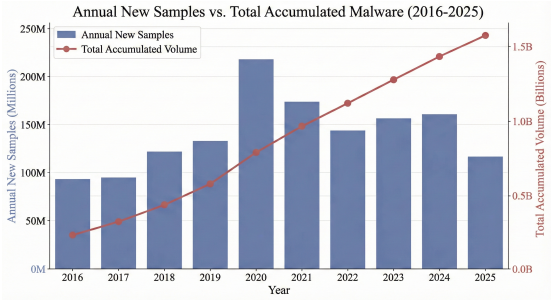
DIMVA 2026 Oral Presentation, Greece



Threat Landscape

Key message

Malware volume keeps growing, and the target has shifted to Linux and the IoT long tail.



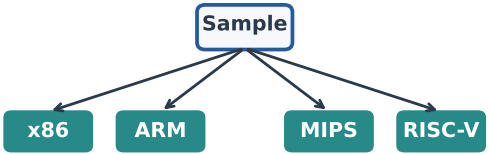
- Accumulated volume has passed **1.5 billion** samples and keeps climbing.
- Center of gravity shifted from Windows to **Linux and IoT**.
- Targets fragment across **diverse CPU architectures**: ARM, MIPS, RISC-V, ...

So what? every sample still needs analysis, across all those architectures.

Requirements and Bottlenecks

Key message

Two jobs a dynamic analyst cannot skip: *run* the sample across architectures, and *trust* the explanation.



Job 1 – Actually *run* it

Static analysis only sees code, while dynamic must execute the sample across ISAs.

Job 2 – *Trust* the explanation

Verdicts must be grounded in trace-backed behavior and IOCs.

The real bottleneck

Running a sample across many architectures & **Producing** reliable runtime evidence.

Limitations of Existing Approaches

Key message

Existing approaches trade coverage, cost, noise, or explainability.

A lightweight, reproducible, cross-ISA forensic path is still missing.

Approach	Cross-ISA	Low overhead	Low noise	Auditable output
Full-system VM sandbox	✓	✗	✗	✗
`strace`/ptrace tracing	✓	✗	✗	✗
Rules or black-box ML	Partial	✓	Partial	✗
Raw trace into LLM	✓	Partial	✗	Partial
MirrorShield	✓	✓	✓	✓

Our Solution

Key message

Grounded LLM reasoning comes from a controlled evidence budget, not from a larger model.

Evidence pipeline

- **Execution**: run heterogeneous Linux binaries on one x86 host.
- **Collection**: collect low-noise syscalls with no in-guest agents.
- **Curation**: structure reports so every claim is checkable.

Our key insight

Execution + **Collection** + **Curation**

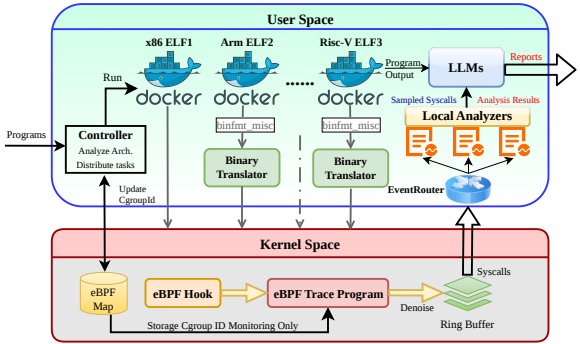
$\Longrightarrow$ **Grounded LLM reasoning**

Control the evidence budget

Overview of MirrorShield

Key message

One x86 host chains execution, telemetry, curation, and LLM reporting.

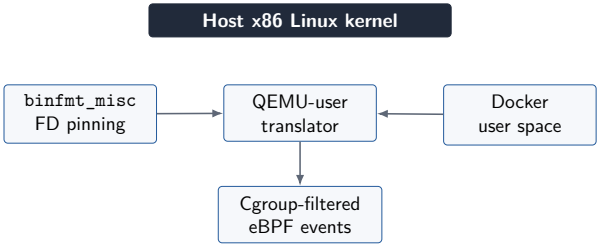


- **Execution**: QEMU-user runs foreign binaries in containers.
- **Telemetry**: eBPF collects container-scoped syscalls.
- **Multi-layer Curation**: high-signal syscalls packed into the LLM budget.
- **Reporting**: analyzers + LLM $\rightarrow$ grounded reports.

Lightweight Cross-Architecture Execution

Key message

We get cross-architecture execution without VM boot costs, while keeping kernel-side visibility.



- `binfmt_misc` redirects foreign `execve` into pinned QEMU-user interpreters.
- **Docker** supplies the target user space and resource isolation.
- **eBPF** watches the *host* kernel; cgroup filters keep only analysis-container events.

Monitor sits outside the sample's privilege domain → harder to evade.

The Challenge of Long-Context Evaluation in Tracing

Key message

Raw traces exceed the LLM's context budget, and the malicious signal is buried in benign noise.

Two hard limits

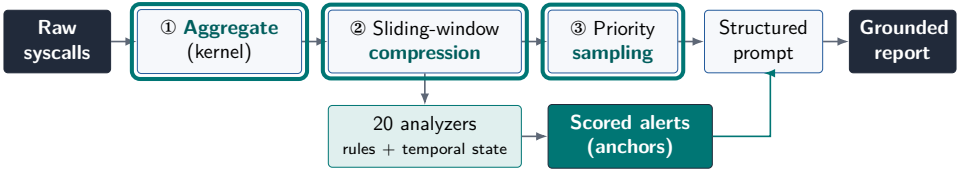
- **Bounded context:** syscall volume exceeds the prompt budget by orders of magnitude.
- **Fragile attention:** malicious signals are buried mid-context by benign noise.

So the goal shifts

Not "fit *more* tokens" but fit the **right** tokens:

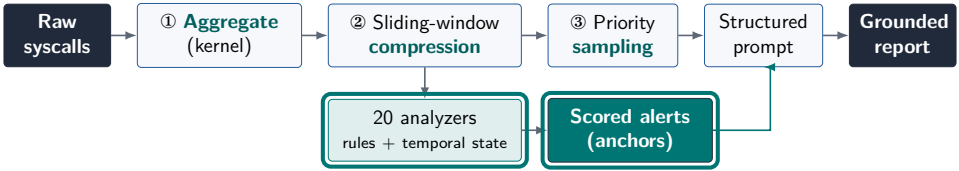
max **signal per token within the budget**

Pipeline Design: Multi-Layer Trace Curation



① **Aggregate:** Count `.so` loads, summarize `mmap`.
② **Compress:** Collapse repeats, preserve execution logic.
③ **Sample:** Keep rare events, head, and tail.

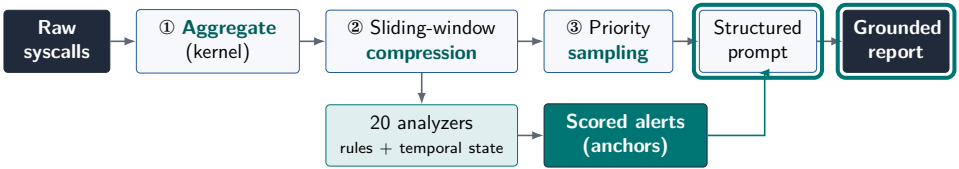
Pipeline Design: Multi-Layer Trace Curation



① **Aggregate:** Count `.so` loads, summarize `mmap`.
② **Compress:** Collapse repeats, preserve execution logic.
③ **Sample:** Keep rare events, head, and tail.

Key: **Alerts** survive sampling → LLM attention stays on the attack.

Pipeline Design: Multi-Layer Trace Curation



① **Aggregate:** Count `.so` loads, summarize `mmap`.
② **Compress:** Collapse repeats, preserve execution logic.
③ **Sample:** Keep rare events, head, and tail.

Key: **Alerts** survive sampling → LLM attention stays on the attack.

Case Study: Analysis of an Early-Crashing Sample

Key message

Same Mirai variant: sandboxes give up or stay generic. MirrorShield reconstructs the attack.

Commercial sandboxes

Joe Sandbox: runtime panic → aborts, falls back to a generic static (YARA) verdict.

Dr. Web vxCube: runs, but reports only coarse tags ("deletes file", "daemon"), with no targets.

MirrorShield: forensic reconstruction

- **Killer:** `UNLINKAT(/etc/.ares)` → competitor removal.
- **Persistence:** embedded shell loop + cron.
- **Anti-forensics:** removes `libdlrpcld.so` hook.

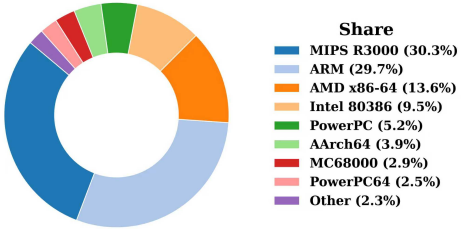
Why it works: pre-crash syscalls $\xrightarrow{\text{curation + grounding}}$ auditable narrative.

Dataset Composition

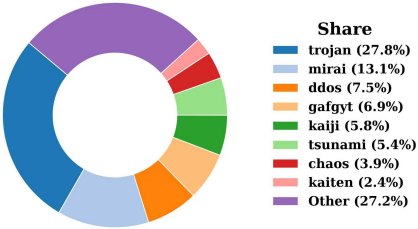
Key message

844 malware (10 families, ∼11 ISAs) + 844 benign ELF binaries (7 ISAs).

Architecture Distribution



Top 8 Threat Tags



Left: targeted CPU architectures. Right: top threat tags (botnet-dominated). Samples from MalwareBazaar, families cross-referenced via VirusTotal.

Impact of Model Selection on Accuracy

Key message

Four different LLMs land in the same accuracy band on the same structured evidence.

Best global accuracy

94.7%

DeepSeek V3.2; $F1_{mal} = 0.946$.

Backend range

91.6–94.7%

Identical evidence and prompts.

Analyzer	Accuracy	Behavior
DeepSeek V3.2	**94.7%**	balanced
Gemini 3.1 Pro	93.5%	conservative
OpenAI GPT-5.2	92.0%	fewer false rejections
Qwen Plus	91.6%	higher recall trade-off

Interpretation

Evidence quality drives detection. Model choice only shifts the precision–recall balance.

Robustness Across Architectures

Key message

Detection holds across 11 ISAs, even when execution itself crashes.

Mainstream ISAs

92.4–94.5%

x86, ARM, MIPS stay stable.

Long-tail ISAs

89.3–100%

RISC-V, PowerPC, SPARC, SuperH, m68k still yield traces.

Execution errors

12.9%

Crashes, timeouts, dependency mismatches.

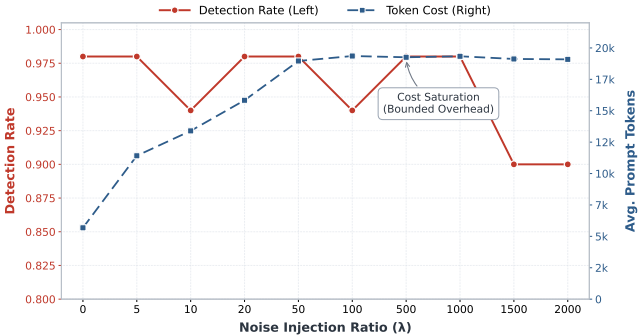
Why accuracy holds

eBPF captures syscalls *before* failure. Imperfect execution still yields usable evidence.

Context Flooding Stays Bounded

Key message

Noise can grow 2000×, but the evidence budget into the LLM does not.



Three observations

- Noise: up to **2000×** malicious volume.
- Detection stays **>90%**.
- Prompt plateaus at **19k tokens**.

Why: sampling caps the LLM input.

Report Auditability

Key message

***Evidence Match* links each report claim back to an observed runtime logs.**

Supported claims

81.4–95.0%

Across four LLM backends.

Reports checked

844

Line-level matching vs. raw JSONL traces.

Trace logs	Semantic claim
`UNLINKAT(".../linux_kill")`	competitor removal
`WRITE("#!/bin/sh while...")`	persistence script
`READ("/etc/init.d/sshd")`	service recon

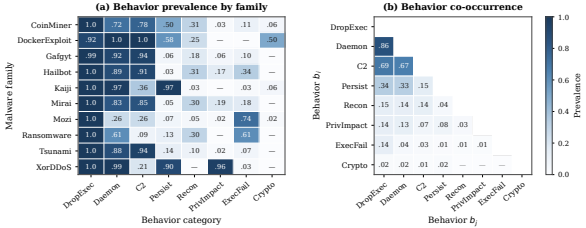
Takeaway

The output is not just a verdict. It is an auditable forensic object.

Behavior Structure Emerges

Key message

Reports yield a structured 8-dim behavior profile: a universal core + sharp family fingerprints.



Per-family prevalence $p_{f,k}$ (left) and pairwise co-occurrence $p_{j,k}$ (right).

Concrete fingerprints ($p_{f,k}$)

- **Universal:** *Drop+Exec* $\approx$ **1.0**.
- **Kaiji:** *Persistence* **0.97**.
- **XorDDoS:** *Privilege Impact* **0.96**.
- **DockerExploit:** *Crypto/SSH/TLS* (unique).

Core triad: $\langle$*Drop+Exec, Daemon, C2*$\rangle$.
Other pairs $<$**20%**.

System Overhead

Key message

Kernel-side eBPF adds only *single-digit* overhead. ptrace and full-system VMs cost 100–6000× more.

Environment	Lat. (ms)	Overhead
Native host (x86)	0.049	1.00×
Native `strace`	16.01	327×
MirrorShield — x86_64	0.215	**4.39×**
MirrorShield — RISC-V 64	0.326	6.65×
MirrorShield — MIPS	0.446	9.10×
MirrorShield — ARM64	0.472	9.63×
LiSa (x86 / ARM)	1.44 / 109.5	29× / 2235×
ELFEN	7.52	154×
QEMU VM + `strace`	297.1	6063×

LMbench `lat_syscall`, steady-state per-call latency.

MirrorShield overhead

4.4–9.6×

Across x86, RISC-V, MIPS, ARM.

Why it matters

Low per-syscall cost makes tracing *every* sample at scale practical.

Limitations and Future Work

Key message

The next step is not to make MirrorShield heavier, but to make its analysis more adaptive and complete.

Behavior coverage	Evidence completeness	Isolation trade-off
A single execution may miss dormant or environment-triggered behavior. **Next:** guided re-execution with adaptive inputs and environments.	Not every semantic claim is fully supported by runtime evidence. **Next:** event-level citations, verification, and abstention.	Lightweight containers are suitable for triage, but not every high-risk sample. **Next:** selectively escalate suspicious samples to stronger sandboxes.

Towards adaptive malware analysis

lightweight triage $\rightarrow$ evidence verification $\rightarrow$ selective re-execution

Conclusions

- **The structure of the input, rather than the LLM backend, largely determines analysis quality.**
- **Incomplete executions can still preserve enough behavior for reliable analysis.**
- **Grounded reports reveal reusable behavioral fingerprints across malware families.**
- **Cross-architecture malware analysis can remain lightweight without sacrificing reliability.**

Acknowledgements

With thanks

- Thanks to the anonymous reviewers and COMPASS members for their insightful comments.
- Supported by the Special Funds for the Cultivation of Guangdong College Students' Scientific and Technological Innovation, and NSFC grants U2541211, 62372218, and U24A6009.
- Thanks to the CS315 Computer Security course at SUSTech, Professor Hugh Anderson, and the "Defence Against the Dark Arts" summer workshop at NUS.

Southern University of Science and Technology

Thank You!

Q&A

计算机系统安全实验室
COMPuter And System Security Lab

MirrorShield

Cross-Architecture Linux Malware Analysis via Lightweight Emulation and LLM

Zihan Zhang, Wenqi Lin, Lingna Sun, Hongyi Wang, Jingyi Wang, Yanshu Mei, and Fengwei Zhang

Email: jacky-bai@foxmail.com

Personal homepage

xxbai.space

GitHub repository

MirrorShield

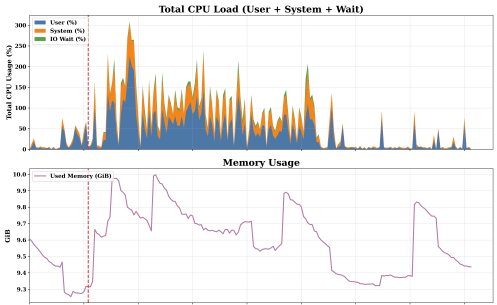
Backup — End-to-End Throughput & Cost

Key message

Cost is dominated by remote inference—local tracing is cheap and memory-stable.

176-sample run

- Local mode (no API): **108.8 s**.
- With DeepSeek API (conc. 7): **438.9 s**.
- CPU 69.9% avg; memory **9.4–10.0 GiB**, no growth.
- ≈ **$0.29 / 100 samples** (DeepSeek pricing).



CPU load and memory during a representative batch.

Backup — Ablation: Sampling + Alerts Are Both Needed

Key message

Removing either curation component drops detection to ∼85%; together they recover 94.7%.

Configuration (fixed trace budget)	Malicious-only detection
No sampling (first n syscalls only)	∼85%
No alerts (analyzer hints withheld)	∼85%
Full curation (sampling + alerts)	**94.7%**

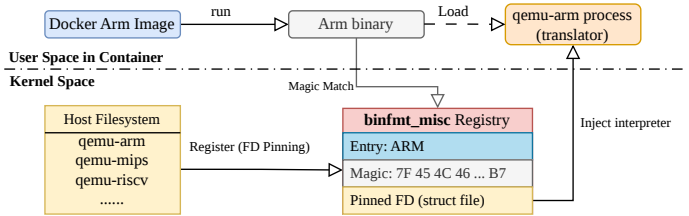
Why

Sampling controls volume; alert grounding restores high-signal events that naive truncation drops. Alerts are conservative—they fire on only **6.6%** of benign reports.

Backup — Execution Internals: `binfmt_misc` + FD Pinning

Key message

Foreign `execve` redirects to a pinned QEMU-user interpreter on the host.



- ELF header parsing selects the matching ISA Docker image.
- Container Cgroup IDs are registered in a BPF map for scoped tracing.
- The interpreter is opened once on the host—no per-container copy needed.
- `binfmt_misc` matches the magic and dispatches to the pinned (`F`) QEMU-user

Key message

Analyzers encode kill-chain rules with per-sample state; they emit conservative, severity-scored anchors.

Category	Analyzer	Detection logic / constraint
Memory & injection	FilelessExec	`exec /proc/*/fd/*` or `memfd_create`
	ProcessInjection	`process_vm_writev` after `ptrace`/`memfd`
Container & kernel	ContainerEscape	`setns`/`pivot_root`, `mount(proc,sysfs,host)`
	KernelManip	`init_module`/`finit_module`, `bpf(cmd=5)`
Persistence & FS	Persistence	18 patterns: `ld.so.preload`, `systemd`, `cron`
	SSHBackdoor	write `.ssh/authorized_keys`, `sshd_config`
Network & exfil	ReverseShell	`connect` → `dup2(fd,{0,1,2})`; per-PID state
Resource & exploit	ForkBomb	`fork`/`clone` $>$ 50 within a 2 s window
Access control	AccessControl	`setuid(0)` / traversal / `/etc/shadow` access

Argument-level constraints suppress false positives; an `AnalyzerManager` tracks multi-step temporal patterns. Alerts fire on only **6.6%** of benign reports.

Backup — In-Kernel Noise Reduction (Layer ①)

Key message

Benign volume is collapsed *at the source*, before it ever leaves the kernel.

Two aggregation strategies

- **Path categorization** (`categorize_path`): benign `.so` loads are *counted* in a BPF array (`agg_counters`), not pushed to the ring buffer.
- **Statistical summarization**: instead of every `mmap`, track count + max allocation (`mmap_stats`); flags large payloads without log saturation.

Five monitored categories

- **Process:** `execve`, `clone`, `sched_*`
- **Filesystem:** `openat`, `getdents64`
- **Network:** `connect`, `bind`, `sendto`
- **Memory:** `memfd_*`, `mprotect`
- **Evasion:** `ptrace`, `seccomp`, `bpf`

Backup — Behavior Profiling: Method & Numbers

Key message

Each report becomes a binary vector $\mathbf{b} \in \{0,1\}^8$; we measure prevalence and co-occurrence.

How each flag is set (hybrid rule)

Keyword match on `summary` / `suspicious_behaviors` / `key_evidence`, *plus* direct syscall evidence:

- `clone` + `dup2` → *Daemon*
- `socket` + `connect` → *C2*
- `kill` on `PID 1` → *Privilege Impact*

Metrics

$$p_{f,k} = \frac{1}{|F_f|} \sum_{i \in F_f} b_{i,k} \quad \text{(prevalence)}$$

$$p_{j,k} = \frac{1}{N} \sum_{i} b_{i,j}\, b_{i,k} \quad \text{(co-occurrence)}$$

Over N=844: *Drop+Exec*≈1.0 (universal); core triad ⟨*Drop+Exec, Daemon, C2*⟩ dominant; other pairs $<$**20%**.