Poster: A Shadow Verification Method for Hot-patch

Yaoju He, Shengyou Chen, Yi Guo, Ao Ju, and Yonggang Li (Corresponding Author)
School of Computer Science and Technology/School of Artificial Intelligence, China University of Mining and Technology
Email: 08231261@cumt.edu.cn, 08230883@cumt.edu.cn, gdd@cumt.edu.cn, 1290186645@qq.com, liyg@cumt.edu.cn

***Abstract*—Serving closed-source hot-patches to commercial off-the-shelf (COTS) software introduces a critical "dual black-box" dilemma: it erodes the verifiability of patch security and renders traditional static analysis ineffective. To address this issue, we design Spat, a hypervisor-based framework for evaluating opaque hot-patches. By synergizing hardware-software co-tracing techniques (INT3 injection, GOT hijacking, trampoline injection, and Intel Processor Trace (Intel PT)), Spat constructs a 3D monitoring profile that captures system calls, library function invocations, and instruction streams. Leveraging a synchronized differential execution architecture, Spat captures microbehavioral deviations between patched and unpatched binary instances. Through practical cross-validation, Spat automates the verification of hot-patch security and reliability, while identifying the root causes of vulnerabilities in the original program—improving the assessment of opaque patches.**

***Index Terms*—hot-patches; black-box; patch assessment**

1. Introduction

Modern COTS software heavily depends on hot-patching for timely vulnerability and critical fault remediation. Yet the coexistence of closed-source host binaries and opaque binary patches gives rise to a severe dual black-box trust problem. This highly opaque execution model leaves administrators no practical way to perform security validation. Blind patch therefore exposes systems to serious risks, including incomplete vulnerability mitigation, unexpected crashes, semantic regressions, and even backdoor injection.

Existing binary analysis techniques fail to address these issues. Static binary differencing tools (e.g., BinDiff) rely heavily on CFG matching [1], but hot-patching mechanisms such as trampolines and inline hooking severely distort CFG structures, leading to high false positives and an inability to capture real runtime semantic differences. Traditional dynamic binary instrumentation (DBI) frameworks (e.g., Intel Pin) introduce substantial runtime overhead and are easily detected by anti-analysis mechanisms widely used in COTS software. Most program analysis approaches require source code or intermediate representations (IRs) [2], making them less applicable to pure binary scenarios. Prior work on runtime verification, shadow execution, and differential testing has explored behavioral comparison across program variants, but typically assumes source-level visibility, controlled instrumentation, or single-level tracing.

To address this challenge, we design and implement Spat, a hypervisor-based synchronized dual-system security evaluation framework. Spat supports high-fidelity, multi-dimensional execution monitoring—capturing system calls, library function invocations, and instruction streams—and aligns execution traces at the microsecond level via high-precision timestamps. Leveraging its synchronized dual-system architecture, Spat enables fine-grained behavioral matching and differential analysis. Through practical cross-validation, it automates security auditing of closed- source hot-patches and identifies the root causes of vulnerabilities in original programs.

2. System Design

Spat is a closed-source binary differential analysis framework built on a Type-1.5 lightweight hypervisor (Fig. 1), with a 5-layer architecture forming a closed loop for end-to-end patch evaluation (top to bottom).

Offline Preprocessing. This layer minimizes runtime overhead by prepping metadata and static probes: (1) Binary differencing identifies patch entry/exit points; (2) Metadata generator parses dynamic libraries to extract function signatures; (3) Static probe injector embeds INT3 breakpoints at target offsets (backing up original instructions).

Hypervisor Layer. A customized hypervisor manages runtime execution. It routes VM-Exit exceptions via CPU virtualization and uses Extended Page Tables (EPT) to isolate hypervisor structures, establishing trusted isolation between VMX root/non-root modes.

Execution Control Layer This layer hijacks control flows and synchronizes dual instances. Dual-instance sync gateway coordinates state/instruction streams of unpatched (Host) and patched (Shadow) instances. Probe micro-cycle controller (integrated with Trap Flag) enables on-demand probe triggering/stealthy reinsertion.

3D Tracing Layer. Core monitoring for patch boundaries via 3-tier tracing: Macro: DRx registers/#DB exceptions capture high-precision system calls (e.g., do_syscall_64); Meso: GOT hijacking/trampoline code extract library function parameters; Micro: Encapsulated Intel PT (IP-filtered) traces only patch code segments (filtering noise).

Analysis Engine Layer User-space engine aggregates heterogeneous tracing data: (1) Multi-dimensional temporal aligner synchronizes system calls, library parameters, and PT instruction streams via high-precision timestamps; (2) Differential analysis engine identifies micro-behavioral

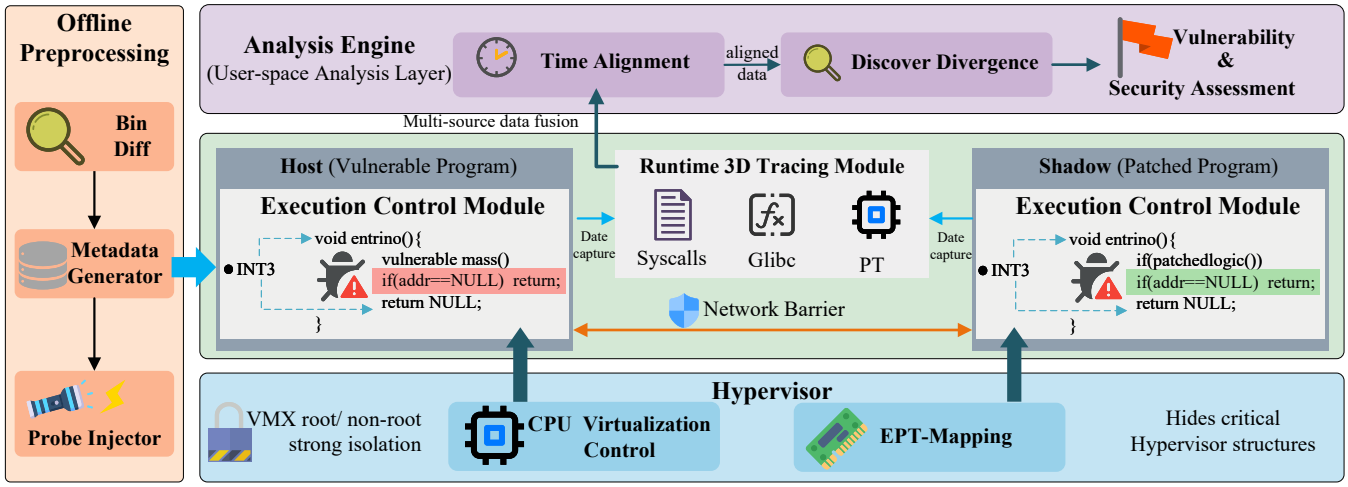


Figure 1. Overall system architecture of Spat

divergences between Host/Shadow instances; (3) Pattern matching against a vulnerability behavior library identifies root causes; (4) Automated generation of patch security/ vulnerability root-cause reports.

3. Evaluation

Figure 2 presents Spat's performance trade-offs: (1) **High-Performance User-Space Tracing:** Spat's hardware-assisted Intel PT and GOT hijacking incur negligible overhead ($\sim 1.0\times$). In stark contrast to dynamic instrumentation frameworks like Frida ($> 20\times$), Spat largely avoids the prohibitive costs of cross-language execution environments and frequent process context switches, delivering low-overhead execution for resolvable user-space events. (2) **Depth vs. Speed in Syscall Interception:** While our DRx-triggered Syscall tracing (8–$10\times$) exhibits higher overhead than sandboxed eBPF ($3.5\times$), this is a deliberate trade-off for analysis fidelity. Spat utilizes `copy_from_user` to proactively traverse and decode pointer-based arguments, capturing richer memory states rather than truncated data. Crucially, Spat employs a mutually exclusive fallback logic: Syscall tracing is only invoked for unresolvable symbols. This synergy allows the heavy Syscall path is largely bypassed, significantly mitigating overall overhead in *mixed* and *glibc* workloads.

TABLE 1. SECURITY EVALUATION

Program	Version	CVE ID	Divergence Pattern
FFmpeg	6.1.1	CVE-2024-31578	Early-Return
SQLite	3.43.1	CVE-2024-0232	Branch Deviation
GNU Binutils	2.43	CVE-2025-0840	Different-Judge-Arg
Ghostscript	10.05.0	CVE-2025-59799	Branch Deviation
SQLite	3.49.0	CVE-2025-3277	Early-Return

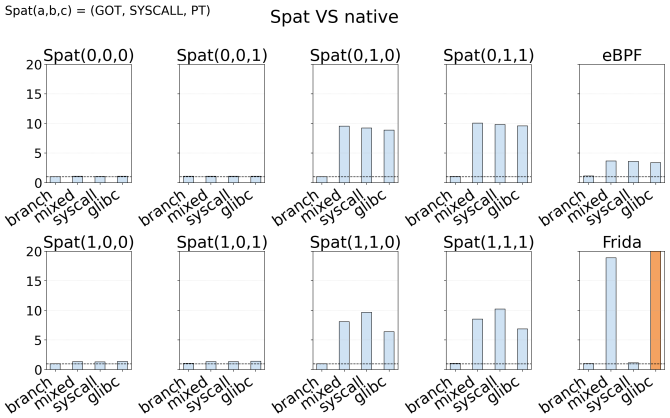


Figure 2. Microbenchmark Performance Overhead

In Table 1, we evaluated CVEs, consistently observing distinct behavioral divergences between unpatched and patched versions. Our pipeline aligns traces, isolates divergences, matches them against a *behavioral paradigm library* to identify vulnerabilities. For instance, in **SQLite CVE-2024-0232**, we identified a critical control-flow divergence at boundary validation. Lacking upfront state checks, the unpatched version allows malformed payloads to trigger deep processing, generating extensive library calls. Conversely, the patched version's strict sanitization forces an immediate early-return, severing the malicious path and yielding a zero-call trace.

References

[1] A. Zhou, Y. Hu, X. Xu, and C. Zhang, "ARCTURUS: Full coverage binary similarity analysis with reachability-guided emulation," *ACM Trans. Softw. Eng. Methodol.*, vol. 33, no. 4, pp. 1–31, 2024.

[2] Y. S. et al., "(State of) The Art of War: Offensive Techniques in Binary Analysis," in *Proc. IEEE Symp. Security Privacy (SP)*, 2016, pp. 138–157.

Poster: A Shadow Verification Method for Hot-patch

Yaoju He, Shengyou Chen, Yi Guo, Ao Ju, and Yonggang Li

School of Computer Science and Technology / School of Artificial Intelligence, CUMT

Email: liyg@cumt.edu.cn

Abstract

Serving closed-source hot-patches to COTS software creates a **"dual black-box"** dilemma, eroding patch verifiability.

Spat is a lightweight hypervisor-based synchronized dual-system framework designed to evaluate opaque hot-patches:

- **Synergized Co-Tracing:** Integrates INT3 injection, GOT hijacking, trampolines, and Intel PT to capture a 3D profile of system calls, library invocations, and instruction streams.
- **Synchronized Dual-System Architecture:** Concurrently executes Host and Shadow programs within a strongly isolated hypervisor environment, enabling safe and precise state alignment.

KEY INNOVATION:
Automated capture of microbehavioral deviations to **verify patch security** and **pinpoint root vulnerabilities** instantly, bypassing traditional static/dynamic limits.

1. Why It Matters: The Dual Black-Box Dilemma

Modern systems rely on hot-patching, but blind deployment introduces severe risks (unexpected crashes, malicious backdoors).

Limitations of Existing Tools:

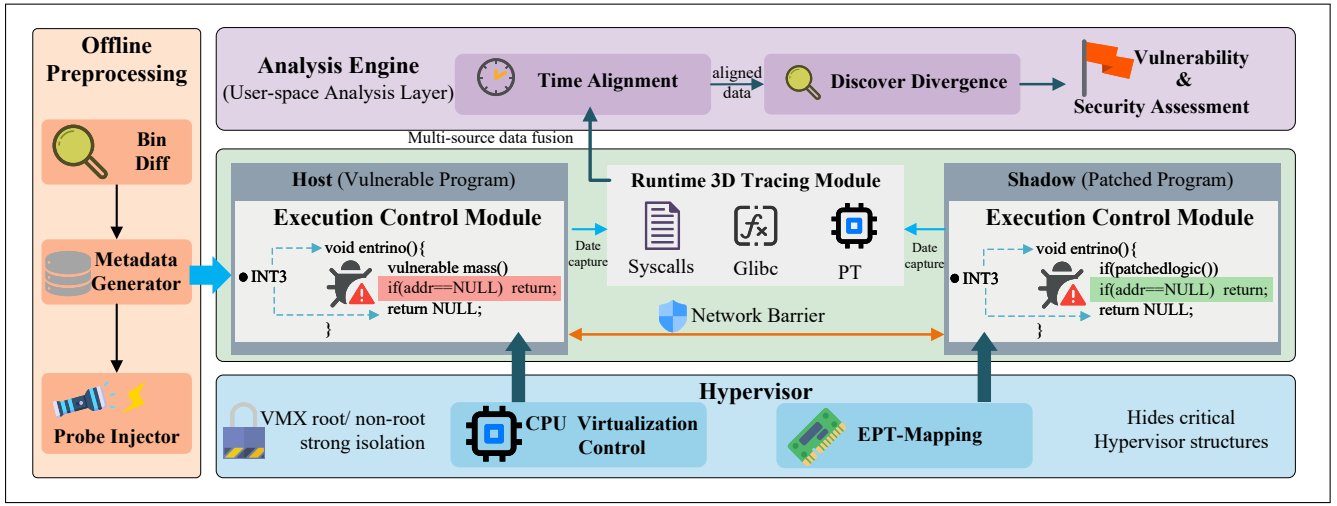
- **Static (BinDiff):** High false positives due to CFG distortions.
- **Dynamic (Intel Pin):** Excessive overhead & anti-analysis vulnerabilities.
- **Formal Verification:** Requires unavailable source code or IR.

2. How It Works: A 5-Layer Analytical Loop

Spat breaks the barrier using multi-dimensional monitoring and microsecond-level synchronization on a Type-1.5 hypervisor.

- **Offline Preprocessing:** Constructs metadata baseline; injects static probes (INT3) to minimize overhead.
- **Hypervisor Layer:** Conceals structures via EPT, isolating VMX modes.
- **Execution Control:** Synchronizes *Host* (unpatched) and *Shadow* (patched) instances stealthily.
- **3D Tracing Layer:**
  - *Macro:* DRx / #DB (Syscalls)
  - *Meso:* GOT hijacking (Libraries)
  - *Micro:* Intel PT (Instructions)
- **Analysis Engine:** Aggregates data, performs automated differential analysis, and traces root causes.

System Workflow: Synchronized Dual-System Execution



3. Performance Evaluation & Overhead

Microbenchmarks validate Spat's performance trade-offs against eBPF and Frida:

- **Lightweight Config (GOT/PT):** Maintains near-native performance (∼1.0x) with virtually no overhead.
- **SYSCALL Interception:** Overhead rises to **8–10x**. This is inherent to the stealthy hypervisor architecture: intercepting syscalls via DRx triggers costly `vmexit` traps and cross-layer `copy_from_user` operations.
- **Cross-Tool Comparison:** While eBPF is faster (∼3.5x), it strictly prohibits complex memory access. Conversely, Frida suffers severe degradation (>20x) on high-frequency calls due to cross-language switching.

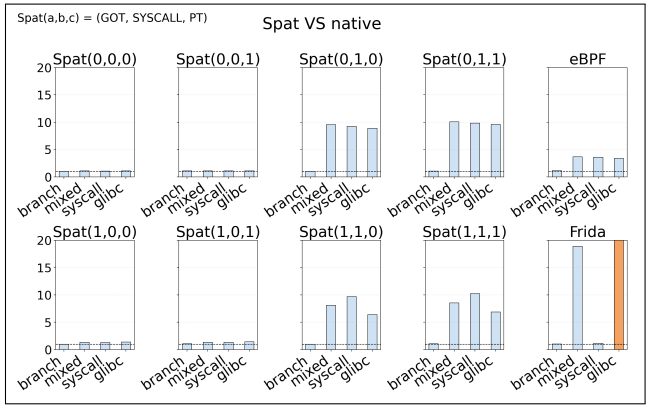


Figure: *Takeaway: Spat trades necessary hardware-level overhead for unrestricted memory access and robust dynamic analysis.*

4. Validation: Real-world Vulnerability Capture

Spat was evaluated against real COTS vulnerabilities. The visual matrix below demonstrates our zero-day/N-day capture capability:

Table: Spat Divergence Analysis on Real-World CVEs

Program	CVE	Type	Divergence
FFmpeg 6.1.1	2024-31578	Use-After-Free	Early-Return
SQLite 3.43.1	2024-0232	Use-After-Free	Branch Deviation
Binutils 2.43	2025-0840	Stack Overflow	Different-Judge-Arg
Ghostscript 10.05	2025-59799	Stack Overflow	Branch Deviation
SQLite 3.49.0	2025-3277	Integer Overflow	Early-Return

Conclusion & Impact

Spat acts as a **high-fidelity truth serum** for opaque software updates, successfully dismantling the "dual black-box" dilemma of COTS hot-patching—where both the patch logic and the host binary are proprietary and untrusted.

- **Unprecedented Visibility:** Bypasses the blind spots of static CFG distortion and the evasion tactics of traditional dynamic analysis via hypervisor-backed 3D tracing.
- **Semantic Precision:** Multidimensional differential execution accurately captures micro-behavioral divergences between patched and unpatched instances.
- **End-to-End Auditing:** Automates the rigorous security validation of binary patches while instantly pinpointing the root causes of original vulnerabilities.