Minding the Gap: Bridging Causal Disconnects in System Provenance*

Hanke Kimm, Sagar Mishra and R. Sekar
Stony Brook University, NY, USA
{hkimm,sagmishra,sekar}@cs.stonybrook.edu

***Abstract*—Advanced Persistent Threats (APTs) are long-running stealthy attack campaigns that routinely evade modern defense mechanisms. As a result, defenders rely heavily on post-compromise forensic analysis to understand attack progression and scope. System call provenance derived from audit logs is widely regarded as the most comprehensive source for reconstructing host activity and "connecting the dots" of multi-stage attacks. However, in practice, audit-derived provenance can be incomplete, or may contain errors. This can happen due to missing context, gaps in event capture or reporting, or incorrect dependency tracking. The resulting errors undermine the central goal of audit data collection, namely, serving as the complete and authoritative source for attack investigation. In this paper, we demonstrate such problems across multiple DARPA Transparent Computing datasets. We discuss the common underlying reasons that contribute to these errors, and then present a new system IMPROV to address them. IMPROV post-processes audit data at the user-level to add additional context and provenance information by querying the OS. It builds on eAudit [1], adding a modest 2.4% additional overhead.**

I. Introduction

Advanced Persistent Threats (APTs) are organized cyber attack campaigns that are characterized by their length and stealth. In recent years APTs have grown in both scope and frequency, as threat actors target institutions from both the private and public sector [2], [3], [4], [5], [6], [7], [8], [9], [10]. These campaigns routinely evade prevention mechanisms, forcing defenders to rely on post-compromise investigation. Such investigations depend heavily on system activity logs. Among available sources, system call logs, also referred to as *audit logs* or *provenance logs*, provide the most comprehensive view of host behavior, recording interactions between processes, files, IPC objects, and network endpoints. Because of this visibility, a large body of APT detection and forensic research builds upon system call provenance [11], [12], [13], [14], [15], [16], [17], [18], [19], [20], [21], [22], [23], [24], [25], [26], [27], [28], [29].

*This work was supported by NSF grants 1918667 and 2153056, with Sekar's work additionally supported by Army Research Office grant W911NF-24-1-0051

Workshop on Attack Provenance, Reasoning, and Investigation for Security in the Monitored Environment (PRISM) 2026
23 February 2026, San Diego, CA, USA
ISBN 978-1-970672-00-8
https://dx.doi.org/10.14722/prism.2026.23023
www.ndss-symposium.org

APT campaigns unfold over extended periods and frequently span multiple processes, user sessions, and hosts, often blending malicious actions with normal system activity. Analysts must therefore reconstruct long causal chains that connect events separated by time, execution context, and machine boundaries. This reconstruction depends critically on the completeness and correctness of the underlying provenance. If connections are missing, key stages of the attack such as initial access, lateral movement, or data exfiltration may appear unrelated or entirely invisible, leading to inaccurate damage assessment and incomplete remediation that allows adversaries to retain persistence. Incorrect connections are equally harmful; spurious dependencies can produce large volumes of false causal paths and force analysts to waste an enormous amount of time chasing meaningless leads. As a consequence, analysts may give up on their investigation, or worse yet, mistakenly conclude that an attack doesn't exist.

In this paper, we identify three classes of gaps/errors in audit logs:

- *Missing edges:* A significant number of processes, files and network endpoints are unidentified, leading to edges that go "nowhere." For instance, a notable **150K** files are unidentified in one of the largest DARPA datasets. Other reasons for missing edges include inconsistent naming of the two ends of pipes, and missing reads and writes.
- *Spurious paths:* Many devices such as `/dev/tty` and `/dev/pty/X` are inherently incapable of creating edges because their read and write ends are unrelated. However, this is not reflected in the way entities are named by provenance collection systems, so provenance graphs constructed from them can contain a huge number of spurious paths. Although the number of such entities may be small, their effect is disproportionately large because of their frequent use in almost every process.
- *Out of order events:* Many events are presented out of order, leading to "use-before-define" problems in provenance. For instance, operations on entities may be presented before full information on the entity is available. In other cases, operations relating to a process

may be presented after its exit; or operations on a file after it has been closed.

Collecting system call provenance is expensive, and its cost is justified by the promise of high fidelity forensic reconstruction. However, the volume of missing, spurious, and misordered relationships observed in our motivational analysis in §II undercuts this promise. This paper addresses this problem by developing mechanisms to recover missing provenance context and restore correct event ordering, thus enabling accurate provenance reconstruction.

A. Approach Overview and Key Contributions

We present Improv, a provenance collection system that addresses the above mentioned problems. It accurately reconstructs subject (process and thread) as well as object (file, pipe, and communication endpoint) identities from audit sources without modifying the kernel or relying on heuristics.

Our approach is based on the following observation: although audit records may omit critical fields, the operating system has full knowledge about all the entities, recorded in the kernel data structures. Moreover, a lot of this information is made available at the user level through standardized interfaces such as the `/proc` on Linux. We leverage these sources of additional information to plug the holes that are typically present in provenance information sources such as `auditd` [30], `sysdig` [31] and eAudit [1]. We implement our techniques on top of eAudit [32], leveraging its eBPF based design to operate efficiently and portably on modern Linux kernels. Our implementation consists of modifications to eAudit's eBPF code, plus a user-level daemon that identifies and fixes gaps in provenance. Specifically, we make the following key contributions.

- *Online recovery of missing provenance:* We introduce an online recovery mechanism that reconstructs missing information before the log is written to the disk.
- *Stable identity under reuse and loss:* We present an identity abstraction for objects that remains unique despite name reuse and descriptor reassignment allowing events to be properly linked in the provenance graph.
- *Ordering-aware provenance reconstruction:* We present an ordering strategy that mitigates the effects of multicore event reordering, preventing provenance gaps caused by out of order events.
- *Experimental validation:* We evaluate our approach on diverse real-world benchmarks and show that Improv recovers substantial missing provenance with negligible benchmark slowdown and a modest additional agent overhead of 2.4% over eAudit.

B. Paper Organization

We present our motivating study in §II, describe our recovery approach in §III, evaluate it in §IV, discuss related work in §V, and conclude in §VI.

II. Motivating Study

Our motivating study stems from repeated observations made while using system call provenance for forensic analysis: provenance often lacks information needed to reliably reconstruct dependencies. Since such gaps could originate either from downstream analysis or from the audit capture itself, we first examine what context is already missing or inconsistent in the raw provenance produced by audit pipelines.

We conduct this study using the DARPA Transparent Computing (TC) datasets [33]. TC provides system call level provenance from realistic host activity, is publicly available, and has been widely used in security research. We focus on Engagement 5, the most recent TC engagement, as it reflects the latest capture and processing pipeline released by the program. We analyze the dataset provided by Trace [34], which derives provenance from the Linux Audit Daemon (`auditd`) [30]. Among TC releases, this dataset exhibits the highest provenance completeness we have observed.

Although Trace data is derived from Linux `auditd`, the problems we describe below are not limited to that source. The reasons underlying the problems affect other common audit sources, including `sysdig` [31] and eAudit [1]. Moreover, `auditd` provides the foundation for numerous other provenance collection systems including Spade [35], Winnower [24], MCI [36], MPI [37], BEEP [38] and ALchemist [39].

The first recurring issue is *missing links*, where processes or objects appear in the provenance graph without sufficient identity information to correlate them with concrete system entities. This manifests in three forms as described below.

Problem 1-A: Unknown Files. DARPA data contained instances where events are associated with non-existent files. We are forced to ignore processing the edges for these unknown nodes, since we cannot locate the record (in this case, an `open` event) that provides information about the file (i.e. path, permissions, owner, etc.). As shown in Table I, about 150K files in the dataset are unknown.

OS Artifact	# of Occurrences
Unknown Files	158850
Unknown Processes	11600

Table I: Unidentified files and processes from DARPA Engagement 5 dataset.

Problem 1-B: Unknown Processes. We observed events that involve an operation on a process but the information about that process cannot be found. For example, a record may require the lookup of a subject's command line argument but the `pid` referencing that subject points to a process that doesn't exist. As a consequence, we consider this subject as *unknown* and create a stub record for it. Under these conditions, we found that ∼11K processes in DARPA E5 are unknown. (Table I).

Problem 1-C: Disconnected Pipes. In the DARPA

E5 dataset, the presence of a file record implies that it was opened in the host. Naturally, one would expect that there would be a non-zero number of operations upon that file after it is opened [1]. However, our evaluation found that over 75% of files in DARPA E5 did not have any operations on them (i.e. read, write, load, etc.), as shown in Table II; this represents a high degree of incompleteness in the dataset. Of these disconnected files, 90% of them are pipes ("Disconnected Pipes 1" in Table II). It is unusual to open a pipe and not have a process read from or write to it, so the absence of such operations suggests an issue with locating the associations of pipe file descriptors. Of the pipes that *do* have at least one read or write edge, 6% of them have the characteristic of having *solely* reads *or* write edges connecting to them. We also consider these pipes disconnected because they do not fulfill the purpose of pipes, which is that a process writes data to one end of a pipe and another process reads from the other end of it. These kinds of pipes are denoted in the row "Disconnected Pipes 2" in Table II.

Problem 2: Phantom Connections. These are connections suggested by the data but don't truly exist. The predominant source of such connections are reads and writes on terminal and pseudo-terminal devices, otherwise known as `/dev/tty` and `/dev/pty/X` respectively, as well as operations to `/dev/null`. Consider the following example: A user A logs onto a host machine through `ssh`, creating a connection to `/dev/pts/4`. Faulty provenance capture might produce information flows to that device from processes run during the `ssh` session; specifically one of user A's processes could be recorded writing to `/dev/pts/4`. After some time a user B connects to the machine and runs a process that reads from `/dev/pts/4`. User A and user B's processes never interact, but the provenance data suggests that there is a flow from A's processes to B's processes.

We found that connections between nodes that are linked by the aforementioned devices total to $\sim$4 billion possible paths. The total number of paths in Engagement 5 are in the trillions (there are about 30 million nodes in a single host graph), so one could be misled that phantom connections do not have a significant effect on paths compared to the number of total paths in the provenance graph. However, recall that spurious paths can hinder forensic analysis efforts; we assert that as long as the problem of phantom connections is feasibly solvable, provenance fidelity should be preserved to the fullest extent.

Problem 3: Out of Order Events. We observed that syscall events were arriving to the user level whose timestamps did not reflect the canonical provenance of the host system. To substantiate our finding, we tracked processes and their audit events within the DARPA data. We saw

[1]In the case of syscall-based audit log collection, if a file is never touched, it typically is not even reported.

OS Artifact	# of Occurrences
Disconnected Files	2851321
Disconnected Pipes 1	2557416
Disconnected Pipes 2	87313

Table II: Disconnected files from DARPA Engagement 5 dataset. The row for "Disconnected Files" shows the number of files in the provenance data that have no associated operations. This has the same meaning for pipes; a pipe is also a file in an OS ("Disconnected Pipes"). The "Disconnected Pipes 2" row denotes pipes that have *only* read or write edges flowing in or out from them, respectively.

many instances of the "use-before-define" problem, defined by the following example: given that the data records are processed in the same order they are presented in the dataset, the dataset will provide a record for a process exit, implying that subsequent operations for that process will no longer be seen. But afterwards, the data provider relays events related to that process. To compensate for this, one could reorder the events so that the process entities are defined before they are used. However, this is a non-trivial task that needs to be carried out at a phase *after* the audit logs are produced. It is difficult to truly measure the scope of such a problem (in the future we plan to add an instrumentation to our analysis framework to count out of order events), but we know that out of order events are a challenge for audit loggers since both `auditd` and `sysdig` affix sequence numbers to events so that correct ordering could be constructed at a later time.

III. Approach Description

The study in the last section shows that provenance data sources in use today have gaps and errors that can degrade the accuracy of forensic analysis and reconstruction. In this section, we show how to correct these problems by harnessing information that is available in the host OS.

Specifically, we describe how Improv addresses this semantic gap using two complementary design principles. First, it performs *post-processing context recovery*: when an audit record lacks subject or object attributes, the system consults kernel state to reconstruct the missing information before the event is incorporated into provenance. Second, it enforces *identity stability*: rather than joining events using transient names or file descriptors, Improv represents kernel objects using stable identifiers that remain valid across descriptor duplication and name reuse. These principles directly target the identity gaps, fragmented object histories, and dependency breaks observed in §II. We next examine why existing audit pipelines systematically violate these principles, and then describe how Improv resolves them through online context recovery (§III-B).

A. Sources of Provenance Gaps in Audit Pipelines

In this discussion, we provide explanations of the complexities of audit log collection and how it leads to provenance graph incompleteness. Note that there may be other unexplained causes of provenance collection bugs beyond what we describe, and that our discussion mainly covers observations captured in our research. We outline the primary sources of these complexities below and link them to the issues described in the motivational study in §II.

Problem 1-A & B: Audit Collection After System Boot. After a host boots up, it is best to initiate audit collection immediately so that the most amount of OS artifacts can be resolved, and in turn, the resulting provenance can be more complete. However, this is an unrealistic scenario because (1) some objects/subjects may remain unresolved well after boot time and (2) it is unreasonable to prohibit a user from toggling audit collection on/off at will. Therefore, the challenge of resolving unknown subjects and objects is non-trivial.

Unresolved subjects or objects in provenance can exist for the following reasons: (1) for subjects, their log records contain a reference to their parent process's pid (`ppid`); however cases occur in which that process id points to a record that does not exist because audit collection started after that parent process's information could be fully resolved. Regarding objects, (2) a logger produces a `read` or `write` event pertaining to a file that was previously opened with an `open` system call before collection. The `open` call contains the path of the file, as well as behavior flags and permissions, and while `read`/`write` event only provides a file descriptor to that file. In this scenario, the logger must perform a lookup in the file descriptor table if an `open` was not previously captured, but this effort is too costly to carry out for every OS file. Therefore, for loggers the file information contained in the `open` system call is simply missed.

Problem 1-C: Difficulty in Tracking Pipes. Although pipes are unnamed filesystem entities, they can be tracked by assigning them unique ids. However, existing audit sources may not always provide such ids, and provenance collection systems such as TRACE try to compensate by using file descriptors as ids. Unfortunately, file descriptors get remapped by the `dup` family of system calls, and tracking this renaming can be challenging. Challenges include: failure to account for all flavors of `dup`; coding bugs that can be hard to avoid in complex tracking logic; use of alternate system calls such as `fcntl` to effect remapping; close-on-exec file descriptors; and the possibility of dropping one of these system calls. As a result, the two ends of a pipe may end up with different names, causing the phenomenon of disconnected pipes we reported earlier.

Problem 2: Interactive Network Access. Programs such as `sshd` and `socat` are network communication programs that generate pseudo-terminals, also known as *pty*. For instance, `sshd` allocates a *pty* session as soon as a user authenticates to a remote host so that the user can interact with the host using a terminal. From the host's perspective, all byte communication resembles that of a local terminal and as such, `read` and `write` audit events regarding the shell session will channel through that host's *stdio*. However, this is not the reality. The `sshd` (and programs like it) application is normally conducted over network, so audit events should resemble remote processes reading and writing data to *network addresses*, not devices.

The ramifications this behavior has on provenance is twofold: we already know that operations on devices like `/dev/tty` and `/dev/pty/X` can create phantom connections. Provenance completeness is further compromised because audit loggers may not correctly report events in which network communication programs like `sshd`, `socat`, `telnet`, etc. are writing and reading to/from network endpoints. For instance, programs like `sshd` are what attackers primarily use to obtain shells on victim hosts. Therefore, a missing connection between `sshd` and the victim host's network endpoint prevents the capture of a potential critical step in the attacker's kill-chain for use in forensic analysis.

Problem 3: Lack of Clock Synchronization across CPU Cores. System calls executed on multicore systems naturally produce audit events on different CPUs, each with its own local clock. While the kernel records events as they occur, these records are asynchronously buffered and later collected in user space, where they are typically ordered using timestamps. Because CPU clocks are not perfectly synchronized, timestamps assigned on different cores do not provide a reliable total order. In practice, clocks can be off by hundred nanoseconds or more — something we have observed in practice in our eAudit implementation. Moreover, some audit sources may only provide coarse-grained timestamps, say, at the millisecond granularity. As a result, timestamps cannot be used for determining the correct order of events. Recognizing this, many audit sources do include a serial number, but this may not always be used by the system that generates provenance from it.

The misordering of events directly impacts provenance reconstruction by breaking causal assumptions. In particular, system calls such as *clone* can exhibit a use-before-define pattern, where the child process returns and begins executing on one core before the parent-side creation context is observed from another. When processed naively, this inversion causes child activity to appear before its defining subject context, leading to orphaned processes and broken dependency chains in the provenance graph.

B. Approach Overview

We now describe how IMPROV post-processes records from the audit source to recover missing execution context. This process restores subject and object information absent in the original events before they are transformed into provenance data. As discussed in §III-A, the audit source frequently references processes and resources without the context needed to bind them to stable identities. IMPROV resolves this gap by consulting live kernel state during post-processing and attaching recovered attributes. This ensures that the final provenance data reflects the actual operating system state rather than the partial view provided by the audit source.

Kernel-derived object identifier. Object identity in audit logs is inherently fragmented. File descriptors and path names are transient: descriptors are reused, duplicated, and reassigned, while files may be renamed or removed and recreated. Consequently, these identifiers are *not* reliable indicators of the object involved in an operation. In particular, connections may be missed because the two ends are using different names but the underlying object is the same. Similarly, a spurious connection may result because the same name is being used by two entities that, in fact, have no relationship.

To address this, Improv introduces a stable object identity abstraction that decouples object identity from descriptors and names. Object identity is anchored in kernel resident state and complemented with on-demand name and metadata recovery. This allows events referring to the same file, pipe, or socket to be joined consistently, even when descriptor context is missing, reused, or reordered. At each system call, the eBPF program derives a compact 64-bit identifier for the underlying kernel object (file, pipe, or socket). The identifier represents the object rather than its file descriptor or name, and is derived from information such as the inode, generation count, device id, etc. This identifier is unaffected by system calls such as `dup`. (With such an identifier, `dup` and `fcntl` family of system calls no longer affect provenance and can be ignored).

We link the following contributions as solutions to the issues described in §II.

Solution 1-A: Unknown Files. When a file is opened, the kernel associates the returned file descriptor with a file object corresponding to a canonical filesystem location (inode and mount context). Improv assigns an object identifier to this file object and, if the descriptor resolves to a valid pathname, records the pathname and caches it under the object identifier for future reference. Subsequent accesses to the same file possibly through different descriptors due to duplication or reuse are joined using the cached object identifier rather than the descriptor value.

If a file access event references a descriptor whose filename is missing from the audit record, Improv resolves the descriptor endpoint via */proc/<pid>/fd/<fd>* at post-processing time, when the kernel state is still valid. When resolution succeeds, the recovered pathname is associated with the existing object identifier and cached for future accesses. In both cases, Improv additionally queries the filesystem using `stat(2)` to recover file metadata such as ownership, permissions, and timestamps.

Solution 1-B: Unknown Processes. In Improv, a subject corresponds to an executing process or thread and is represented internally by a unified `subjectinfo` record. As discussed in §III-A, audit collection may begin after system initialization or be restarted during runtime, causing process creation events (e.g., `fork`, `clone`, or `execve`) to be missing or observed out of order. As a result, subsequent system call events may reference a subject using only PID or TID values, without sufficient context to construct a complete `subjectinfo`. If left unresolved, these references appear as unknown or orphaned subjects in the provenance data.

To resolve such missing subjects, Improv reconstructs subject identity during post-processing by using the PID and TID carried in the audit record as the stable subject key. Upon encountering an event whose subject lacks an existing `subjectinfo` entry, Improv queries the live process state via */proc/<pid>* to recover the information necessary to populate the subject record. When the process is still present, Improv recovers the executable path and command line (e.g., from */proc/<pid>/exe* and */proc/<pid>/cmdline*), along with process lineage and thread context, including the parent PID and thread-group ID (TGID). This recovered information is used to construct a complete `subjectinfo` record that accurately represents the executing subject.

The reconstructed `subjectinfo` record is cached using PID/TID-based keys and reused for all subsequent events referencing the same subject. If live recovery fails (e.g., because the process has already terminated or the PID has been reused), Improv creates a minimal, identifier-backed stub `subjectinfo` record. This stub preserves graph connectivity without asserting incorrect execution semantics, ensuring that missing subject context does not break dependency chains required for forensic reconstruction.

Solution 1-C: Disconnected Pipes. Pipes are unnamed IPC objects and are typically accessed through paired file descriptors. At pipe or socket creation, Improv derives a single object identifier from the kernel IPC object and associates both endpoints with this identifier. All subsequent read and write operations performed through either descriptor are joined through the shared object identifier, independent of descriptor number.

Note that descriptor duplication does not affect the underlying file structures that ultimately contain information such as inodes. Thus, ids computed from these structures have no dependence on the file descriptors even if they are changed by `dup`-like system calls.

Solution 2: Phantom Connections. To handle phantom connections, platform-specific semantics of devices should be reflected in the naming. For instance, /dev/tty will be named /dev/tty_out for output operations, and /dev/tty_in for input operations. This will ensure that data written to /dev/tty will cause an information flow to a read operation on /dev/tty. Similarly, platform specific behavior of `sshd`-like programs should be recognized and the input/output to stdin and stdout should instead be reported as the input/output IP address of ssh client.

Solution 3: Out of Order Events. While the mechanisms above restore subject and object identity, provenance correctness also requires that events be incorporated in an order consistent with kernel execution. As discussed in §III-A, audit records are delivered asynchronously and often appear out of order when sorted using user-space timestamps. Events may be generated on different CPUs, buffered per-core, and delivered late, causing timestamp-based ordering to diverge from the kernel's true execution order.

To address this, Improv introduces a sequence number that provides a consistent global order across all system calls. This sequence number is produced inside the eBPF program using an atomic fetch-and-add operation on a global counter, which eBPF supports as a highly efficient primitive. This ensures that the sequence number reflects the exact order in which the events occurred in a multi-core kernel.

However, sequence numbers alone are insufficient for system calls whose semantics involve multiple execution contexts, most notably `clone`. Unlike typical system calls, `clone` returns twice: once in the newly created child process and once in the parent process. Importantly, the child may begin executing and generate audit events before the parent-side return of `clone` is observed. As a result, events attributed to the child process may appear earlier in the sequence than the event that establishes the child's existence. This leads to a use-before-define problem in provenance: operations performed by the child are observed before the subject identity of the child has been fully defined.

Improv resolves this by combining global ordering with semantic pairing of system call records. The system maintains in-flight state to track partial system calls, associating entry and exit records before provenance updates are applied. For calls such as `clone`, the child identity is not committed to the graph until the parent-side creation context has been observed and paired. Complementary records are matched using proximity in sequence numbers, allowing moderate reordering while preserving correctness. To ensure progress under loss or extreme reordering, unmatched partial events are expired using bounded sequence number and time thresholds.

IV. Experimental Evaluation

We evaluate Improv to determine if its post-processing context recovery mechanisms effectively close the gap in provenance data, while maintaining low runtime overhead. Specifically, our evaluation addresses the following questions:

- *Provenance recovery effectiveness (§ IV-B):* To what extent does post-processing reconstruct missing subject and object information that would otherwise appear as unknown or disconnected?
- *Runtime overhead (§ IV-C):* What is the performance impact of enabling post-processing in Improv?

Name	Description
find	uses `find` command to print all file names in `/usr`.
httperf	Web server benchmark generating HTTP request.
redis	Key–value store benchmark issuing database queries.

Table III: Application benchmarks.

A. Experimental Setup and Datasets

Implementation. Improv is implemented using eBPF and is compatible with recent Linux kernels. We build Improv on top of the eAudit open-source prototype [32], which provides an efficient framework for capturing and communicating system call events to user space [1]. Our implementation augments eAudit with user-level components that recover missing provenance context by consulting live kernel state.

Benchmarks. Table III lists the primary application benchmarks adopted from [1], with the addition of the `redis` workload. These benchmarks represent realistic, long-running applications that generate sustained system activity with relatively stable execution contexts.

In addition to application benchmarks, we include a *basic operations* test case that exercises common file and process operations, such as file creation and deletion, rename-with-overwrite, permission changes, truncation, and file descriptor duplication and reuse. This test case is intentionally executed on an active system, where many processes and objects already exist and may lack complete creation records in the audit stream. As a result, it exposes scenarios in which provenance is commonly lost due to identifier reuse, object mutation, or out of order events discussed in §II.

We run Improv in parallel with eAudit and measure the number of resolved subjects and objects across all workloads, following the resolution scenarios described in §III.

Evaluation Platform. All experiments were carried out on an i7-12700 processor (12 cores) with 64 GB of RAM and a 1TB SSD running Ubuntu 22.04.

B. Evaluation of Provenance Recovery

Basic Operations. We evaluate how effectively Improv eliminates gaps in provenance data under the basic operations workload. This workload intentionally exercises file creation, deletion, rename, truncation, permission changes, and descriptor duplication on an already active system, thereby triggering the provenance loss scenarios described in §II. In total, the workload generates approximately 52K successful system calls.

Table IV summarizes the provenance gaps observed when using eAudit alone and the extent to which Improv repairs them.

eAudit leaves 75 of 190 processes unresolved (39%). These correspond primarily to long-lived processes that were already running when audit logging began and therefore lack creation records. Improv recovers the missing

OS Artifact	# of Occurrences	Unresolved (eAudit)	Unresolved (IMPROV)
Processes	190	75	0
Objects	2,561	91	0
Out-of-order events	52,000	1,560	0

Table IV: Resolving missing provenance using basic operations (artifact recovery and ordering repair).

execution context using live kernel state (via */proc*) and resolves all cases of missing subject provenance.

For objects (files, pipes, and sockets), eAudit leaves 91 of 2,561 unresolved (3.5%). These gaps arise from descriptor reuse, missing `open` time context, and IPC tracking inconsistencies. IMPROV resolves *all unresolved objects* through OS lookup and stable object identity assignment, yielding *0% unresolved objects.*

Finally, eAudit exhibits 1,560 out of order events out of 52,000 (3.0%), producing use-before-define anomalies and broken causal chains. IMPROV's ordering-aware reconstruction reconciles all such cases, eliminating ordering-related provenance ambiguity.

Overall, IMPROV repairs *all of the provenance information lost by eAudit* in this workload, leaving no unresolved subjects, objects, or events.

With Benchmarks. We next evaluate IMPROV under realistic, long-running application benchmarks that generate sustained system activity. These workloads stress the provenance pipeline under high event volume and expose provenance loss arising from preexisting processes, descriptor reuse, and event reordering.

Table V reports unresolved provenance observed with eAudit and the remaining unresolved cases after applying IMPROV.

For processes, eAudit leaves a consistent fraction of references unresolved: 64.1% (70/109) for *find*, 62.1% (56/90) for *httperf*, and 61.3% (56/91) for *redis*. These correspond largely to processes that were already running when audit logging began. IMPROV resolves *all such cases*, leaving no unresolved processes in these benchmarks.

For objects, eAudit leaves a smaller fraction unresolved, ranging from 1.2% to 8.1%, depending on reuse patterns. IMPROV resolves all but a small number of objects. Specifically, *find* retains 5 unresolved objects out of 376K total objects ($\approx 0.001\%$), while all of the objects are resolved for *httperf* and *redis*. The rare cases of unresolved objects are due to short-lived objects that cannot be reliably recovered from kernel state.

Across all benchmarks, 2.21 million of 62.7 million events were identified as out of order, and IMPROV resolves all such inconsistencies.

Overall, across all benchmarks and provenance categories, IMPROV recovers *nearly all* provenance information lost by eAudit, leaving only a negligible number of unresolved objects in one workload.

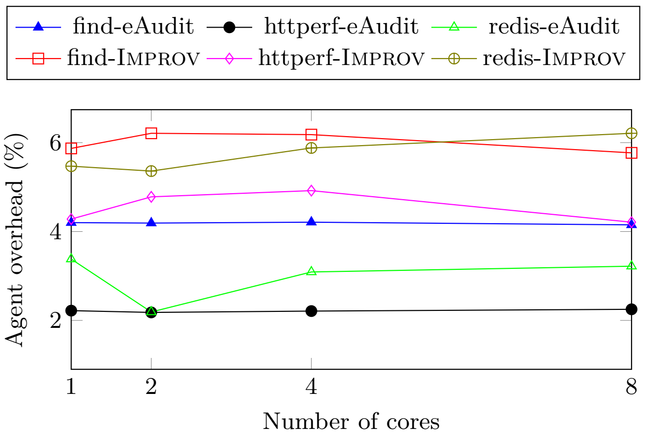

Fig. 1: Agent overhead, defined as the ratio of the CPU time consumed by the agent to the baseline CPU time of the benchmark, for IMPROV and eAudit.

C. Runtime overhead

We now evaluate the runtime cost of enabling post-processing to eliminate gaps in provenance data. Because recovery requires consulting kernel state during event processing, overhead may arise in both the user-level agent and the benchmarked applications.

Agent Overhead. Fig. 1 compares the agent-side overhead of IMPROV and eAudit. While eAudit serves as the baseline, IMPROV performs post-processing to recover missing context from the audit source via */proc* lookups.

Enabling this recovery incurs a modest increase in agent overhead. Averaged across all benchmarks, overhead increases from 3.1% for eAudit to 5.5% for IMPROV. This additional 2.4% reflects the cost of ensuring complete provenance data while remaining low in absolute terms.
Benchmark overhead. Fig. 2 compares the application slowdown of IMPROV against eAudit. This metric reflects the end-to-end cost of post-processing to ensure complete provenance data.

Across all benchmarks, the additional slowdown is minimal. IMPROV incurs an overhead of 17.4%, compared to 16.6% for eAudit. This incremental overhead of 0.8% represents the cost of recovering context missing from the audit source while maintaining performance comparable to the baseline.

D. Portability and Trust Assumptions

Portability and Generalization. Although our prototype is implemented on Linux using eBPF and the `/proc` interface, the recovery methodology itself is not Linux-specific. The core requirement of IMPROV is access to authoritative kernel-maintained execution state at the time audit events are processed. Modern operating systems maintain equivalent state for processes, files, sockets, and IPC objects as part of normal kernel operation, and expose portions of this information through kernel introspection, tracing, or system interfaces.

Benchmark	Total Events	Processes			Objects			Out-of-order Events	
		Total	Unresolved (eAudit)	Unresolved (IMPROV)	Total	Unresolved (eAudit)	Unresolved (IMPROV)	eAudit	IMPROV
find	34.06 M	109	70	0	376 K	4.5 K	5	1.4 M	0
httperf	20.6 M	90	56	0	2209	30	0	412 K	0
redis	8.1 M	91	56	0	2821	229	0	405 K	0

Table V: Unresolved provenance reported by eAudit and resolution with IMPROV.

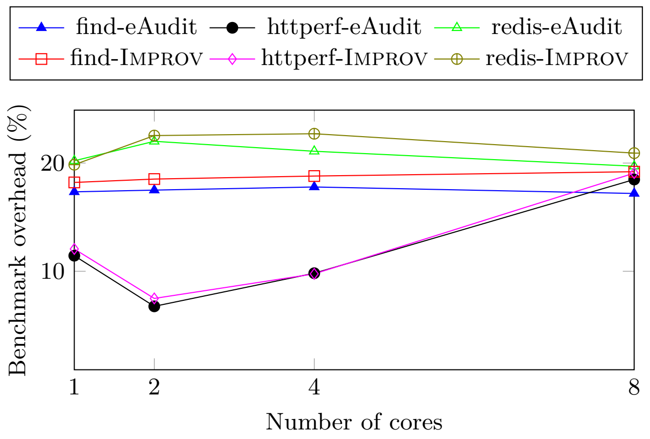

Fig. 2: Benchmark Overhead: Percentage increase in benchmark wall clock time as a result of provenance data collection, for IMPROV and eAudit.

Windows maintains process, thread, file, and handle state within kernel objects. ETW (Event Tracing for Windows) provides low-overhead event streams, while process and object state can be resolved through kernel object and handle introspection APIs. Similarly, BSD-derived systems provide varying degrees of access to process and file information through interfaces such as `sysctl` and, in some variants, `procfs`-style mechanisms. In all these systems, audit or tracing records can be augmented using live kernel state before constructing provenance graphs.

Thus, IMPROV should be viewed as an architectural pattern rather than a Linux-specific technique: provenance derived from event logs is reconciled with the operating system's current execution state to restore missing context and enforce stable identities. eBPF enables an efficient realization of this pattern on Linux, but porting the design to other platforms primarily involves adapting the context-recovery layer to the target OS's kernel state interfaces, not redesigning the provenance model itself.

Trust Assumptions and Kernel Compromise. IMPROV assumes that kernel-maintained state is trustworthy at the time of recovery. This assumption is consistent with the threat model of audit-based provenance systems in general. If an attacker has compromised the kernel, they can tamper with kernel data structures, system call handling, and audit output itself. Under such a full system compromise, both raw audit logs and any recovery mechanism derived from kernel state become untrustworthy. In other words, kernel compromise represents a class of attacks that invalidates the trustworthiness of all host-based logging and provenance mechanisms, not just IMPROV. Our goal is to address structural provenance loss that occurs in benign or partially compromised systems where the kernel remains intact. Defending against kernel-level rootkits would require complementary techniques such as hardware-backed isolation or secure logging hardware which are orthogonal to the recovery problem addressed in this work.

V. Related Work

Gates and Bishop introduce *broken provenance* to describe situations where expected provenance records are missing, incomplete, or contain incorrect information at creation time [40]. For recovery, they propose architectural strategies such as querying original generators, leveraging redundancy across multiple providers, or retrieving archived copies of provenance records. These mechanisms aim to reconstruct or approximate the provenance records themselves using saved logs, replicas, or cooperative components. However, their approach assumes that missing information was recorded elsewhere and remains retrievable. It is less focused on cases where essential context (e.g., file paths, process attributes, or object state) was never logged and existed only transiently in kernel state. As a result, provenance loss that originate directly from runtime system state may fall outside the scope of their recovery model.

Subsequent work moves from conceptualizing broken provenance to systematically evaluating and detecting provenance incompleteness. Chan et al. introduced provenance *expressiveness benchmarking*, which aims to clarify how operating system activities are reflected in the provenance graphs produced by different capture systems and to provide objective criteria for assessing provenance data quality in terms of correctness and completeness [41]. ProvMark was later developed as an automated framework that operationalizes this idea, generating provenance benchmarks for common system call behaviors and enabling qualitative comparison of how different provenance systems represent the same activities [42]. Building on this foundation, Chan et al. extend the approach toward provenance *integrity checking*, using the benchmarking framework to identify missing or abnormal provenance patterns caused by unreliable sources or dropped audit records [43]. Ahmad et al. further extend discrepancy detection to distributed environments, identifying missing or inconsistent relations across whole-network provenance views [44]. Together, these efforts provide increasingly systematic ways to evaluate the expressiveness and integrity of captured provenance and to localize structural gaps in recorded graphs. Their primary focus, however, is on analyzing recorded provenance and comparing it against expected or reference models, rather than on reconstructing missing subject or object bindings or recovering runtime system state that was not originally logged.

In contrast, our work targets recovery rather than detection. We reconstruct missing subject and object context by consulting kernel-resident state, enabling stable entity binding and causal connectivity even when audit derived provenance is incomplete or partially ordered. This bridges the key gap left open by prior work: moving from identifying provenance loss to repairing the provenance data so it remains usable for attack reconstruction and security analysis.

VI. Conclusion

We presented Improv, a provenance recovery system that eliminates gaps in provenance data without modifying the kernel. Improv restores missing subject and object context by leveraging live kernel state and assigns stable identities to files, pipes, and sockets to prevent fragmentation caused by descriptor reuse and incomplete records. It also reconciles out of order audit events to preserve causal structure.

Our evaluation shows that Improv recovers nearly all provenance information lost by conventional audit pipelines across diverse workloads, while imposing negligible additional application slowdown and only modest agent overhead. These results demonstrate that high-fidelity provenance can be achieved without modifying the kernel and with minimal runtime cost.

References

[1] R. Sekar, H. Kimm, and R. Aich, “eAudit: A fast, scalable and deployable audit data collection system,” in *IEEE S&P*, 2024.

[2] “Actions taken by equifax and federal agencies in response to the 2017 breach,” https://www.gao.gov/assets/700/694158.pdf.

[3] “Source: Deloitte breach affected all company email, admin accounts,” https://krebsonsecurity.com/2017/09/source-deloitte-breach-affected-all-company-email-admin-accounts/.

[4] “Lessons learned from the HSE cyber attack,” https://www.hhs.gov/sites/default/files/lessons-learned-hse-attack.pdf.

[5] SentinelOne, “The new frontline of geopolitics: Understanding the rise of state-sponsored cyber attacks,” https://www.sentinelone.com/blog/the-new-frontline-of-geopolitics-understanding-the-rise-of-state-sponsored-cyber-attacks/, Aug 2025.

[6] “Marriott data breach 2020: 5.2 million guest records were stolen,” https://securityboulevard.com/2020/04/marriott-data-breach-2020-5-2-million-guest-records-were-stolen/.

[7] CNN, “Casino giant MGM expects $100 million hit from hack that led to data breach,” https://www.cnn.com/2023/10/05/business/mgm-100-million-hit-data-breach/index.html, Nov 2025.

[8] L. French, “Echoes of solarwinds: Jetbrains teamcity servers under attack by russia-backed hackers,” 2023, accessed: 2024-08-22. [Online]. Available: https://www.scmagazine.com/news/echoes-of-solarwinds-jetbrains-teamcity-servers-under-attack-by-russia-backed-hackers

[9] K. Collier, “National guard hacked by chinese 'salt typhoon' campaign for nearly a year, DHS memo says,” https://www.nbcnews.com/tech/security/national-guard-was-hacked-chinas-salt-typhoon-group-dhs-says-rcna218648, accessed: 2026-01-18.

[10] F. N. P. Office, “Fbi identifies lazarus group cyber actors as responsible for theft of $41 million from stake.com,” https://www.fbi.gov/news/press-releases/fbi-identifies-lazarus-group-cyber-actors-as-responsible-for-theft-of-41-million-from-stakecom, accessed: 2026-01-18.

[11] M. N. Hossain, S. M. Milajerdi, J. Wang, B. Eshete, R. Gjomemo, R. Sekar, S. Stoller, and V. Venkatakrishnan, “SLEUTH: Real-time attack scenario reconstruction from COTS audit data,” in *USENIX Security*, 2017.

[12] S. M. Milajerdi, R. Gjomemo, B. Eshete, R. Sekar, and V. Venkatakrishnan, “HOLMES: Real-time APT detection through correlation of suspicious information flows,” in *IEEE S&P*, 2019.

[13] W. U. Hassan, S. Guo, D. Li, Z. Chen, K. Jee, Z. Li, and A. Bates, “NODOZE: Combating threat alert fatigue with automated provenance triage,” in *NDSS*, 2019.

[14] P. Gao, X. Xiao, D. Li, Z. Li, K. Jee, Z. Wu, C. H. Kim, S. R. Kulkarni, and P. Mittal, “SAQL: A stream-based query system for real-time abnormal system behavior detection,” in *USENIX Security*, 2018.

[15] M. N. Hossain, S. Sheikhi, and R. Sekar, “Combating dependence explosion in forensic analysis using alternative tag propagation semantics,” in *IEEE S&P*, 2020.

[16] E. Manzoor, S. M. Milajerdi, and L. Akoglu, “Fast memory-efficient anomaly detection in streaming heterogeneous graphs,” in *Proceedings of the 22nd ACM SIGKDD International Conference on Knowledge Discovery and Data Mining*, 2016.

[17] X. Han, T. Pasquier, A. Bates, J. Mickens, and M. Seltzer, “UNICORN: Runtime provenance-based detector for advanced persistent threats,” in *NDSS*, 2020.

[18] J. Zeng, Z. L. Chua, Y. Chen, K. Ji, Z. Liang, and J. Mao, “WATSON: Abstracting behaviors from audit logs via aggregation of contextual semantics,” in *NDSS*, 2021.

[19] J. Zeng, X. Wang, J. Liu, Y. Chen, Z. Liang, T.-S. Chua, and Z. L. Chua, “Shadewatcher: Recommendation-guided cyber threat analysis using system audit records,” in *IEEE S&P*, 2022.

[20] B. Bhattarai and H. Huang, “Steinerlog: prize collecting the audit logs for threat hunting on enterprise network,” in *ACM CCS*, 2022.

[21] Z. Cheng, Q. Lv, J. Liang, Y. Wang, D. Sun, T. Pasquier, and X. Han, “Kairos: Practical intrusion detection and investigation using whole-system provenance,” in *IEEE S&P*, 2024.

[22] M. Rehman, H. Ahmadi, and W. Hassan, “Flash: A comprehensive approach to intrusion detection via provenance graph representation learning,” in *IEEE S&P*, 2024.

[23] A. Alsaheel, Y. Nan, S. Ma, L. Yu, G. Walkup, Z. B. Celik, X. Zhang, and D. Xu, “ATLAS: A sequence-based learning approach for attack investigation.” in *USENIX Security*, 2021.

[24] W. Ul Hassan, M. Lemay, N. Aguse, A. Bates, and T. Moyer, “Towards scalable cluster auditing through grammatical inference over provenance graphs,” in *NDSS*, 2018.

[25] Y. Liu, M. Zhang, D. Li, K. Jee, Z. Li, Z. Wu, J. Rhee, and P. Mittal, “Towards a timely causality analysis for enterprise security,” in *NDSS*, 2018.

[26] S. Li, F. Dong, X. Xiao, H. Wang, F. Shao, J. Chen, Y. Guo, X. Chen, and D. Li, “Nodlink: An online system for fine-grained apt attack detection and investigation,” in *Proceedings 2024 Network and Distributed System Security Symposium*, 2024.

[27] K. Pei, Z. Gu, B. Saltaformaggio, S. Ma, F. Wang, Z. Zhang, L. Si, X. Zhang, and D. Xu, “Hercule: Attack story reconstruction via community discovery on correlated log graph,” in *ACSAC*, 2016.

[28] F. Dong, L. Wang, X. Nie, F. Shao, H. Wang, D. Li, X. Luo, and X. Xiao, “DISTDET: A Cost-

Effective distributed cyber threat detection system," in *32nd USENIX Security Symposium (USENIX Security 23)*. Anaheim, CA: USENIX Association, Aug. 2023, pp. 6575–6592. [Online]. Available: https://www.usenix.org/conference/usenixsecurity23/presentation/dong-feng

[29] W. U. Hassan, A. Bates, and D. Marino, "Tactical provenance analysis for endpoint detection and response systems," in *IEEE S&P*, 2020.

[30] S. Grubb, "Linux audit," https://people.redhat.com/sgrubb/audit/, accessed: 2025-11-19.

[31] A. Kili, "Sysdig – a powerful system monitoring and troubleshooting tool for linux," https://www.tecmint.com/sysdig-system-monitoring-and-troubleshooting-tool-for-linux/, accessed: 2025-07-19.

[32] S. S. Lab, "eAudit: a fast, scalable and deployable audit data collection system," http://eprov.org/, 2024, accessed: 2025-04-26.

[33] "DARPA transparent computing Engagement 3 data release (also includes Engagement 5 data)," https://github.com/darpa-i2o/Transparent-Computing/, accessed: 2025-3-8.

[34] H. Irshad, G. Ciocarlie, A. Gehani, V. Yegneswaran, K. H. Lee, J. Patel, S. Jha, Y. Kwon, D. Xu, and X. Zhang, "TRACE: Enterprise-wide provenance tracking for real-time APT detection," *IEEE TIFS*, 2021.

[35] A. Gehani and D. Tariq, "SPADE: support for provenance auditing in distributed environments," in *International Middleware Conference*, 2012.

[36] Y. Kwon, F. Wang, W. Wang, K. H. Lee, W.-C. Lee, S. Ma, X. Zhang, D. Xu, S. Jha, G. Ciocarlie, A. Gehani, and V. Yegneswaran, "MCI: Modeling-based causality inference in audit logging for attack investigation." in *NDSS*, 2018.

[37] S. Ma, J. Zhai, F. Wang, K. H. Lee, X. Zhang, and D. Xu, "MPI: Multiple perspective attack investigation with semantic aware execution partitioning," in *USENIX Security*, 2017.

[38] K. H. Lee, X. Zhang, and D. Xu, "High accuracy attack provenance via binary-based execution partition," in *NDSS*, 2013.

[39] L. Yu, S. Ma, Z. Zhang, G. Tao, X. Zhang, D. Xu, V. E. Urias, H. W. Lin, G. F. Ciocarlie, V. Yegneswaran, and A. Gehani, "ALchemist: Fusing application and audit logs for precise attack provenance without instrumentation." in *NDSS*, 2021.

[40] C. Gates and M. Bishop, "One of these records is not like the others," in *USENIX TaPP*, 2011.

[41] S. C. Chan, A. Gehani, J. Cheney, R. Sohan, and H. Irshad, "Expressiveness benchmarking for system-level provenance," in *USENIX TaPP*, 2017.

[42] S. C. Chan, A. Gehani, and J. Cheney, "Provmark: A benchmark for testing the expressiveness of provenance systems," in *ACM CCS*, 2019.

[43] S. C. Chan, J. Cheney, A. Gehani, and H. Irshad, "Integrity checking and abnormality detection of provenance records," in *USENIX TaPP*, 2020.

[44] R. Ahmad, E. Jung, C. de Senne Garcia, H. Irshad, and A. Gehani, "Discrepancy detection in whole network provenance," in *USENIX TaPP*, 2020.