ANCORA: Accurate Intrusion Recovery for Web Applications

Yihao Peng†, *Graduate Student Member, IEEE*, Biao Ma†,
Hai Wan, Xibin Zhao, *Senior Member, IEEE*

***Abstract*—Modern web application recovery presents a critical dilemma. Coarse-grained snapshot rollbacks cause unacceptable data loss for legitimate users. However, surgically removing an attack's impact is hindered by a fundamental challenge in high-concurrency environments: it is difficult to attribute the resulting file, database modifications, etc., to a specific attack related request.**

We present ANCORA, a system for precise intrusion recovery in web applications without invasive instrumentation. Our approach first isolates the full sequence of syscalls triggered by a single malicious request. Based on this sequence, ANCORA addresses file and database modifications separately. To trace file modifications, it builds a provenance graph that reveals all changes, even those made by exploit-spawned processes. To attribute database operations, a more difficult challenge due to connection pooling, ANCORA introduces a novel technique based on a spatiotemporal anchor. This anchor uses the request's network connection tuple and its active time window to pinpoint the exact database operations. With all malicious file and database operations precisely identified, ANCORA performs a unified rewind and selective replay recovery. It first reverts the system to a clean snapshot taken before the attack. Then, it selectively re-applies only the legitimate operations to both the file system and the database. This process completely removes the attack's effects while preserving all concurrent legitimate data. We evaluated ANCORA on 10 web applications and 20 CVE-based attack scenarios with concurrency up to 150 connections. Our experiments demonstrate that ANCORA achieves 99.9% recovery accuracy with manageable overhead: up to 19.8% response latency increase and 17.8% QPS decrease in worst-case scenarios, and recovery throughput of 110.7 database operations per second and 27.2 affected files per second, effectively preserving legitimate data.

***Index Terms*—Intrusion recovery, provenance tracking, selective replay, database forensics, web application recovery, syscall analysis.**

I. INTRODUCTION

WEB applications have become primary targets for cyberattacks [1], [2]. While intrusion detection systems (IDS) offer a first line of defense [3]–[10], they are not infallible. Consequently, the ability to precisely recover from successful intrusions to ensure data integrity and service continuity is paramount. Modern applications manage distributed and heterogeneous state, encompassing files scattered across disks, relational and non-relational databases, and external web services [11]–[17]. Handling any external request (including attack requests) may involve file operations, database operations, and interactions with external web services.

Industrial solutions typically rely on snapshot-based rollbacks, which indiscriminately discard all post-attack operations, including legitimate user transactions, leading to unacceptable data loss [18]–[21]. A more effective paradigm is "surgical" recovery, which precisely analyzes and reverses only malicious changes. *The central task of such recovery is to identify all modifications—spanning files, databases, and external web services—caused by a single malicious request.* However, achieving this in high-concurrency environments is exceedingly challenging. Interleaved streams of low-level syscalls and database logs from numerous concurrent users create an ambiguous many-to-many mapping, severely complicating the attribution of specific database operations and file modifications to a single originating http request.

Previous Solutions and Limitations. Prior research has explored two main avenues to address this challenge.

- **Taint analysis based on metadata propagation** instruments an application's runtime, such as the language interpreter or middleware, to propagate a unique identifier from each http request to its subsequent operations [12]–[16]. This allows for direct correlation. However, this approach is fundamentally limited by its high cost and invasiveness, requiring deep, complex modifications to application frameworks that are impractical in production environments. Furthermore, it is inherently blind to unexpected execution paths created by exploits, such as remote command execution, because these actions bypass the instrumented logic and thus remain untainted and untraceable.
- **Black-box behavior pattern matching** avoids instrumentation by using external monitoring and machine learning to infer correlations between high-level http requests and low-level system events [11], [17]. The main limitation of this approach is its diminished accuracy in high-concurrency settings. The dense, interleaved event streams make precise attribution challenging for learning-based models, often resulting in misclassifications. These systems are also typically restricted to monitoring predefined directories for performance reasons, leaving them unable to detect malicious file writes in arbitrary, unpredictable locations.

Our Solution. This paper presents ANCORA, a system for precise intrusion recovery in web applications. ANCORA comprehensively traces and reverses all system state changes, across both files and databases, caused by a malicious request. It achieves this with only application-agnostic, framework-level instrumentation.

Inspired by attack investigation techniques from forensics

Yihao Peng, Biao Ma, Hai Wan, and Xibin Zhao are with the Beijing National Research Center for Information Science and Technology (BNRist), Key Laboratory for Information System Security, Ministry of Education (KLISS), School of Software, Tsinghua University, Beijing 100084, China (e-mail: xy20130630@gmail.com; mb_thu@163.com; wanhai@tsinghua.edu.cn; zxb@tsinghua.edu.cn).

†These authors contributed equally to this work.

analysis, which trace the causal chain of events forward from an initial point of compromise [22]–[27], ANCORA begins by using forward analysis to isolate the complete syscall sequence corresponding to a given malicious http request. This sequence alone is sufficient to identify and report all interactions with external web services. However, recovering the internal state requires solving two further challenges at a higher semantic level:

1) Database Correlation Challenge: Accurately linking an http request to its specific database operations through complex layers of application connection pooling and asynchronous database processing.
2) File Correlation Challenge: Tracing covert file writes that bypass normal application logic, often triggered by vulnerabilities like command injection.

To solve the database challenge, we deconstruct the complex application-database interaction model by using network communication metadata as a *spatiotemporal anchor*. From the request's syscall sequence, ANCORA extracts the unique network connection tuple (4-tuple) and its precise active time window. This anchor allows us to pinpoint the exact database server thread that handled the request and then filter its I/O on database log files, thereby isolating the exact database operations belonging to the malicious request.

To solve the file challenge, we employ *causality tracking based on syscalls* [28], [29]. Instead of passively monitoring a few directories, ANCORA actively analyzes the request's syscalls to construct a complete provenance graph. This graph reveals the entire causal chain of process execution triggered by the initial malicious act, making any covert file writes by descendant processes visible, regardless of their location on the file system.

With all malicious operations identified, ANCORA executes a unified *Rewind & Selective Replay* recovery. It first rolls the system back to a clean snapshot. Then, it performs a fine-grained replay: for the database, it replays only legitimate transactions from the database's own logs; for the file system, it uses a hybrid strategy, restoring critical system areas from baseline backups while incrementally replaying legitimate writes in user data directories. This process eradicates the attack's impact while preserving all concurrent legitimate user data.

Experiments and Evaluations. We evaluated ANCORA on 10 web applications spanning 4 programming languages. The experiments included 20 real CVE vulnerabilities and covered three common SQL and NoSQL databases: `MySQL`, `PostgreSQL`, and `MongoDB`. The results show that ANCORA achieves 99.9% accuracy under concurrency up to 150 connections with manageable overhead: up to 19.8% response latency increase and 17.8% QPS decrease in worst-case scenarios, and recovery throughput of 110.7 database operations per second and 27.2 affected files per second.

Our Contributions:

- We design and implement ANCORA, a novel recovery system that uses causal tracing of low-level behaviors to precisely restore file and database state and identify external interactions, requiring only non-intrusive, framework-level instrumentation.
- We propose a database operation tracing method that deconstructs the asynchronous execution model using network metadata as an anchor, solving the correlation problem in the presence of connection pools and multi-threading.
- We design a file system repair mechanism that combines causality tracking with a hybrid rewind-and-replay strategy, enabling reliable recovery of file tampering at any path by discovering covert writes through a provenance graph.
- We demonstrate through extensive experiments on diverse applications and attack scenarios that ANCORA achieves 99.9% accuracy and incurs manageable performance overhead with response latency increase up to 19.8% and QPS decrease up to 17.8% in worst-case scenarios under typical workloads.

In the spirit of open science, we make our tool available at https://anonymous.4open.science/r/Ancora/.

II. BACKGROUND AND MOTIVATION

A. Provenance Graph

A provenance graph is a directed acyclic graph that captures the causal and informational dependencies among system entities [28]. Nodes in the graph represent subjects, such as processes and threads, and objects, such as files and network sockets. Directed edges represent the information flow and causal relationships between them. Every low-level system event, such as a process u writing to a file v at time t, is abstracted as an edge in the graph [30]. When the object of one event becomes the subject of a subsequent event, a causal chain is formed, providing a theoretical foundation for tracing complex behavioral sequences. *Forward analysis* on a provenance graph interprets these causal relationships to understand an attack's impact [29], [31]. It begins from a known malicious source and traverses all subsequent events triggered by it, thereby assessing the full scope of the compromise.

ANCORA leverages this capability to address the challenge of covert file tampering caused by exploits. When a malicious http request is identified, ANCORA uses its corresponding initial process or thread as the starting point for analysis. By performing forward analysis, it constructs a complete causal execution chain. This chain exposes all file write operations performed by the malicious request and any of its descendant processes, regardless of their location. This provides the critical foundation for ANCORA's comprehensive attack footprint identification and precise file system recovery.

B. Threat Model

We consider an external attacker who sends one or more malicious http requests to exploit vulnerabilities in a web application, such as SQL injection, remote command execution, or arbitrary file uploads. The immediate goal of these attacks is to tamper with the system's internal state, specifically by making unauthorized modifications to the file system (e.g., writing a webshell, altering configuration files) or performing malicious database operations (e.g., stealing, modifying, or

deleting data). The core objective of ANCORA is to precisely recover all file and database state changes caused by such malicious requests after they are detected.

Assumptions. Our threat model relies on the following assumptions:

- The operating systems of the web and database servers provide reliable audit logging that records system-level events (i.e., syscalls), including process/thread identifiers, timestamps, and specific operations.
- The audit logging mechanism and the ANCORA system are part of the Trusted Computing Base (TCB). We assume their integrity and availability are maintained during an attack and that they cannot be easily tampered with or disabled. This is a common and necessary assumption for many log-based security analysis systems. [32]–[34].
- The target database can be configured to enable comprehensive logging of modification statements, such as setting `log_statement = 'mod'` in `PostgreSQL`. This assumption, shared with other database recovery research [11], [16], [17], ensures that the raw text of all database operations is captured, providing the necessary data for subsequent analysis and recovery.
- The database environment possesses common and well-established mechanisms for creating snapshots and performing restorations (e.g., `mysqldump` for `MySQL`, `pg_dump/pg_restore` for `PostgreSQL` [35]–[37]), providing the essential basis for system state regression and the assurance of data consistency.

Scope. ANCORA focuses on attacks initiated by external requests that produce observable state changes at the system level. Our model does not directly address attacks that operate purely within the application's logic without triggering discernible differences in database or file I/O at the syscall level. Furthermore, the following attack vectors are considered out of scope: side-channel attacks, hardware-level exploits (e.g., Spectre), resource-exhaustion denial-of-service attacks (e.g., network floods), passive remote system fingerprinting, various network spoofing attacks, and any threats requiring prior physical access or an established foothold within the internal network. This threat model aligns with the consensus in existing web application recovery research [11], [16], [17], allowing us to concentrate on the core challenge of reversing multi-faceted state corruption caused by external attacks.

III. SYSTEM DESIGN

A. System Design Overview

The core objective of ANCORA is to achieve comprehensive and accurate recovery by precisely identifying and reversing all state changes caused by a malicious request, across both files and databases, while preserving all concurrent legitimate operations. As illustrated in Figure 1, the ANCORA workflow proceeds in three stages after a malicious http request is identified by an external IDS or a human analyst:

1) **Request Partition.** This foundational stage isolates the actions of a single malicious request. The *Log Collection* module gathers syscall audit logs from the instrumented web application. We use lightweight, framework-level instrumentation to inject special *Delimiter Logs*, each with a unique request ID, into the log stream. The *Unit Partition* module then uses these delimiters to deterministically extract the exact sequence of *Malicious Syscalls* corresponding to the malicious request from the interleaved system-wide log. This sequence serves as the input for the next stages and directly reveals any interactions with external services, enabling the generation of *Notifications to Remote Web Services*. In parallel, ANCORA maintains data baselines through *Incremental Backup of Files* and *Periodic Database Backup*.
2) **Post-Attack File System Recovery.** This stage restores the file system. It takes the *Malicious Syscalls* as input. Instead of passively scanning directories, the *Execution Dependency Analysis* module actively builds a provenance graph from the syscalls. This process traces the full causal chain, revealing all *Implicit File Syscalls* triggered by the initial request or any of its descendant *Malicious Processes/Threads*, thus exposing covert writes at any location. The *Rewind & Selective Replay* module then uses this precise footprint, along with the *Incremental Backup of Files*, to perform a surgical recovery, yielding the *Recovered Files*.
3) **Post-Attack Database State Recovery.** This stage restores the database. Starting with the same *Malicious Syscalls*, the *DB Connection Analysis* module extracts the "spatiotemporal anchor" of the request's database interaction: its network connection tuple and precise active time window. This produces *Malicious Connection Fragments*. The *DB Operation Extraction* module then uses this anchor to pinpoint and extract the exact *Malicious DB Operations* from the database's own logs (*Database Audit Logs* and *Database App Logs*). Finally, the *Rewind & Selective Replay* module uses these operations and a *Periodic Database Backup* to roll back and selectively replay only legitimate transactions, producing the *Recovered DB State*.

Ultimately, this two-pronged recovery restores the system to a clean and consistent *Recovered System State*, eradicating the attack's impact while maximizing the preservation of legitimate data. We detail the methods for each stage in Sections III-B, III-C, and III-D.

B. Request Partition

The prerequisite for precise recovery in ANCORA is the ability to isolate the complete, independent causal chain of system activities triggered by a single malicious http request from a noisy, interleaved stream of low-level system behaviors. In a production environment, syscall logs from the web server, database, and other system processes are voluminous and highly interleaved, which makes attributing low-level behaviors to a specific high-level request a significant challenge.

To address this, the *Request Partition* module deterministically partitions the commingled stream of system-wide syscall audit logs into discrete behavioral units, each corresponding to a single http request. This module adopts a delimiter-logging approach, inspired by prior work on syscall partitioning [22], which uses lightweight, application-agnostic instrumentation

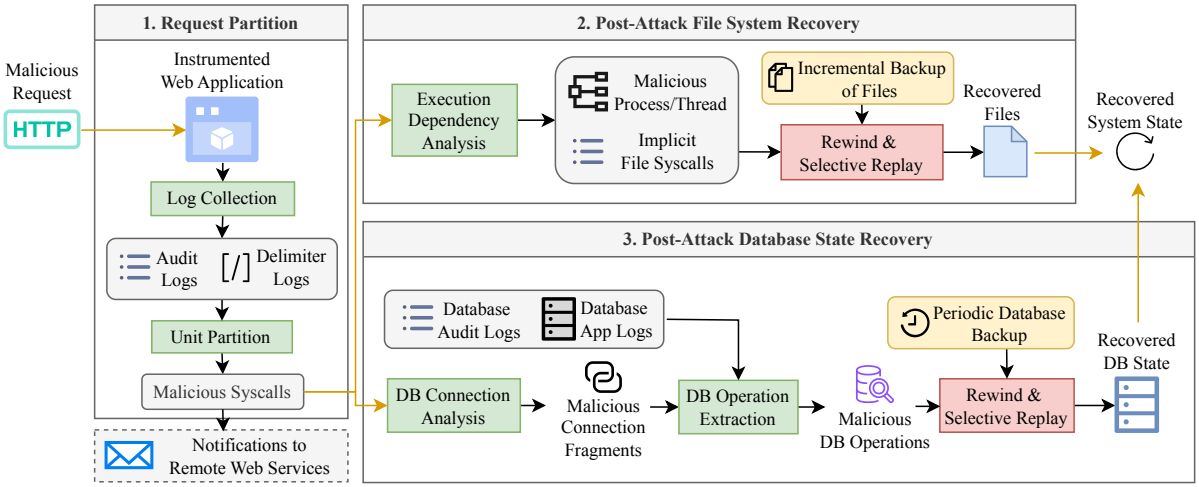

Fig. 1: The workflow of ANCORA.

at the web framework level. At critical points in a request's lifecycle, such as its initiation and context switches, our instrumentation injects special *delimiter logs* containing a unique request ID into the syscall stream. These delimiters act as markers, delineating the boundaries of different requests' activities within the continuous log data.

ANCORA adapts its parsing strategy based on the web server's concurrency model:

- **Process/Thread-based Concurrency.** For servers that assign a dedicated process or thread to each request, such as certain modes of `Gunicorn`, the process ID (PID) and thread ID (TID) in syscall logs provide a natural grouping. Within a single PID/TID context, requests are handled sequentially. Therefore, our injected delimiters clearly mark the boundary points where the processing of one request ends and the next begins.
- **Asynchronous/Coroutine-based Concurrency.** For asynchronous models like those used by `FastAPI` and `Node.js`, multiple coroutines handling different requests execute concurrently within the same system thread. Their syscalls become deeply interleaved, making PID/TID insufficient for separation. In this scenario, our instrumentation is more fine-grained, injecting delimiters not only at the beginning of a request but also at every coroutine context switch, when one request's execution is suspended and another's resumes. These frequent, ID-tagged delimiters allow us to accurately unravel the intertwined syscalls from the single thread's log stream and re-attribute them to their respective source requests.

Using this approach, the *Request Partition* module efficiently and accurately extracts the dedicated syscall sequence for every incoming malicious request. This sequence is the cornerstone of all subsequent analysis in ANCORA: it serves as the input for the file system and database recovery modules, and it also directly exposes the request's interactions with external web services (manifested as network-related syscalls like `sendto`), providing precise data for generating alerts and enabling cross-system remediation measures.

C. Post-Attack File System Recovery

Once the malicious request's syscall sequence is isolated, the next step is to recover the file system from any resulting modifications. The primary challenge is that attackers often exploit vulnerabilities, such as remote command execution (RCE), to create unexpected execution paths that bypass the application's standard file I/O logic. For instance, a compromised application might spawn a `curl` or `wget` process to download and write a webshell. Such actions are invisible to systems that only monitor application-level APIs.

To address this, ANCORA's file recovery module employs a two-stage strategy: *active causal tracking* followed by *rewind and seletive replay*. First, the *Execution Dependency Analysis* stage actively constructs a provenance graph from syscalls to reveal all file operations triggered by the malicious request, directly or indirectly. Second, the *Rewind & Selective Replay* stage uses this precise attack footprint, in conjunction with backups, to surgically excise the malicious changes.

Execution Dependency Analysis. This stage constructs a complete causal chain based on syscalls to identify the full impact of a malicious request on the file system. Instead of passively monitoring predefined directories, this analysis actively constructs a provenance graph. As shown in Figure 2, the analysis begins with the malicious syscall sequence from the *Request Partition* module. We treat this sequence as the root of the attack and use process creation syscalls, such as `fork` and `clone`, as key events for extending the causal chain. ANCORA recursively tracks the activities of these newly spawned processes and threads (and their descendants) throughout the global system log.

This process builds a complete provenance graph rooted in the initial malicious request, forming a comprehensive execution chain. The graph captures all file modifications, including those performed by multi-layered descendant processes in arbitrary system locations. The output of this stage is a detailed list of affected files, each with its path, the exact operation performed, and a timestamp.

Rewind & Selective Replay. After precisely identifying all

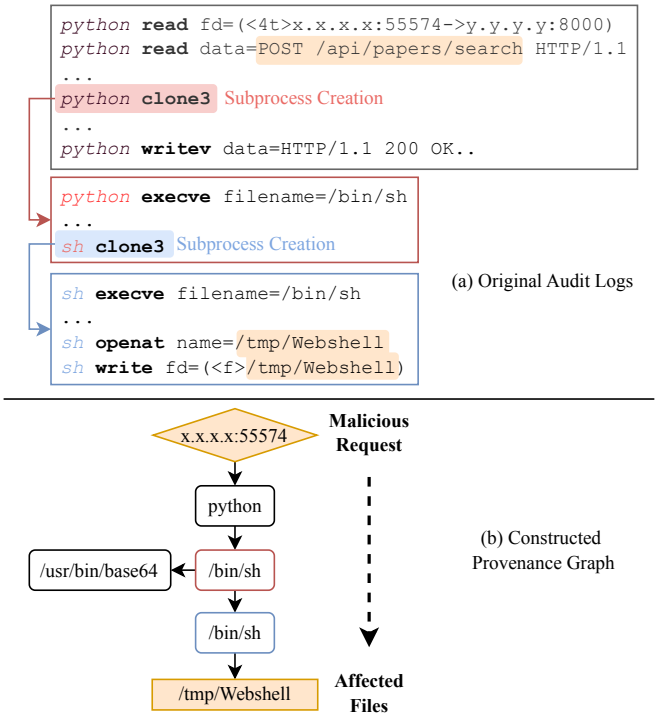

Fig. 2: An example of file operation matching.

malicious file operations, the core principle of recovery is to eradicate the attack's impact while preserving all legitimate data generated concurrently. A simple system-wide rollback to a pre-attack snapshot is unacceptable as it would cause significant data loss by discarding all concurrent legitimate user operations.

ANCORA therefore employs a *Rewind & Selective Replay* mechanism. The process first reverts the file system to a clean state from before the attack using a snapshot, and then replays only the write operations that have been confirmed as legitimate. Since logging all file writes or backing up every file update is prohibitively expensive, we implement a differentiated recovery strategy that categorizes file system regions to reduce overhead:

- **System and Application Directories.** These areas, such as `/bin`, `/etc`, or the web application's own code directories, are expected to be static during normal operation. If the causal analysis detects that an attack has written files in these locations, ANCORA's recovery strategy is to overwrite them from a clean baseline backup.
- **Data Directories.** In directories for user uploads (e.g., `/uploads`) or dynamically generated content, both legitimate and malicious requests may write data. For these, ANCORA uses a more granular, incremental approach. It first restores an affected file to its latest pre-attack backup version. Then, it consults the log of all write operations that occurred during the attack period, filters out those identified as malicious in the previous stage, and replays the remaining legitimate writes in chronological order. This process precisely reconstructs all legitimate user data while eliminating the attack's effects. Data directories are identified either through pre-deployment analysis of the application's normal behavior or via manual configuration by an administrator.

Implementation Details. *1) Provenance Graph Construction.* We use `Sysdig` as our underlying syscall collection tool, which captures detailed syscall events with low overhead. To build the provenance graph, we map these structured log events to graph components. For example, a `fork` or `clone` syscall creates a new process node and a `CREATE_PROCESS` edge from the parent to the child. File operations like `openat` or `write` generate a file node (if not already present) and a corresponding `WRITE` or `OPEN` edge from the process node, annotated with attributes like timestamp and path. Calls like `unlink` or `rename` generate `DELETE` or `RENAME` edges, precisely capturing file state transitions. This mapping allows ANCORA to automatically transform the low-level syscall stream into a high-level, analyzable provenance graph.

2) Write Log for Replay. To enable granular replay for data directories, ANCORA captures legitimate write operations' content. Similar to other systems [11], we use a generic, application-agnostic kernel module that, upon intercepting a `write` syscall to a data directory, logs the event metadata alongside the full data payload. This logged data is buffered in-memory and written to disk asynchronously by a background thread, minimizing performance impact. These captured payloads form the "write log" for replay, enabling ANCORA to re-apply legitimate data changes onto files restored from a baseline.

3) Backup and Snapshot Implementation. ANCORA's backup strategy is twofold, designed to efficiently protect both dynamic data and static system files.

- *Incremental Backup (For Data Directories):* For efficient, low-storage backups of data directories, we combine `Watchdog` (a Python library based on Linux's `inotify`) and `rsync`. `Watchdog` monitors for file changes in real-time and triggers an incremental `rsync` task. By using `rsync`'s `--link-dest` option, unchanged files in a new backup are created as hard links to their counterparts in the previous backup, rather than being copied. This significantly reduces storage consumption and enables lightweight maintenance of a file system snapshot chain.
- *Baseline Backup (For System/Application Directories):* Static system and application directories require a "known-good" baseline, whose creation depends on the deployment environment. In a containerized environment, the baseline is the original, immutable container image, requiring no additional backup; ANCORA restores any tampered file by fetching a clean version from the read-only image layer. In a traditional VM or bare-metal environment, we perform a one-time, full backup post-initial deployment and configuration, before serving traffic. This archive is stored securely and serves as the authoritative baseline for all subsequent recoveries.

4) Interactive Recovery for Complex Files. For structured files, such as binaries, where simply omitting malicious writes during replay could lead to corruption, ANCORA provides an

interactive recovery mode. It presents the administrator with the file's last-known-good state and a detailed log of all subsequent write operations, explicitly flagging those identified as malicious. This allows the administrator to make an informed decision: perform a complete rollback, selectively replay only legitimate writes, or manually repair the file. This human-in-the-loop approach ensures the safe recovery of complex data structures, preventing file corruption that could arise from fully automated procedures.

D. Post-Attack Database State Recovery

Restoring database state presents a unique challenge rooted in the communication patterns of modern web architectures. Web applications typically use connection pooling to communicate with databases, allowing multiple http requests to reuse the same underlying TCP connection. Concurrently, database servers employ highly asynchronous models to process requests from multiple connections. This combination of concurrency models results in a database execution stream where operations from many high-level requests are deeply interleaved. This makes it difficult to answer the critical question of *which specific database operations were executed by a given malicious http request*.

To overcome this, ANCORA introduces a novel method for precise database operation attribution by deconstructing the asynchronous execution model. Before detailing the implementation, we first establish its underlying principle.

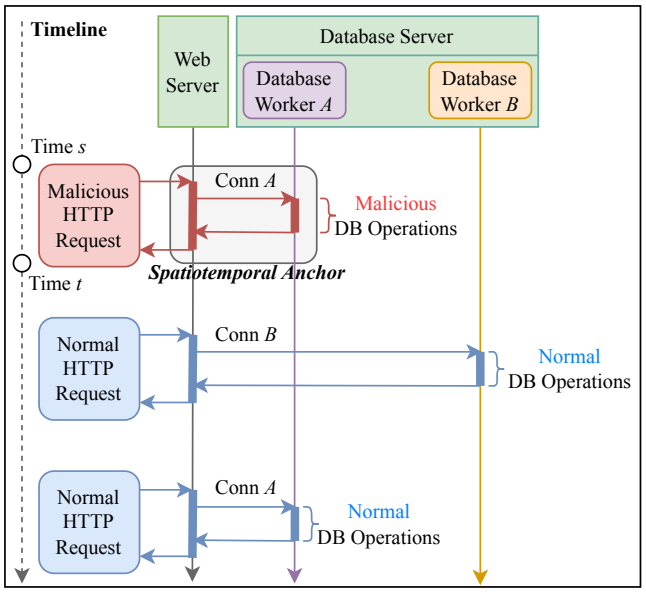

Fig. 3: The principle of database operation matching.

Principle. Our core insight is that by deconstructing the two layers of concurrency between a web application and its database, we can identify a stable *spatiotemporal anchor* that bridges the asynchronous gap, establishing a precise causal link between a high-level request and its low-level database operations. We examine each concurrency model separately:

- *Web Application Connection Pooling.* To avoid the high overhead of creating and destroying TCP connections, web applications use connection pools. A request "borrows" a connection from the pool, uses it for all its database operations, and then "returns" it. The key property here is *exclusive use during time-division multiplexing*: although a single connection (`Conn A`) is used by different requests over its lifetime, it is used exclusively by one request for the duration of that request's processing.
- *Database Server Concurrency.* Mainstream relational databases like `MySQL` and `PostgreSQL` typically employ a "thread/process-per-connection" model. When a client establishes a TCP connection, the database server dedicates a worker thread to handle all communication on that connection until it closes. This creates a stable one-to-one mapping between a network connection and a database worker thread[1].

Combining these models reveals the path to precise attribution, as illustrated in Figure 3. When the *Malicious Web Request* begins at `Time_s`, it borrows `Conn_A`. For its entire processing duration (from `Time_s` to `Time_t`), it has exclusive use of `Conn_A`. On the database server, `Conn_A` is handled exclusively by *Database Worker A*. Therefore, we can conclude that within the precise time window [`Time_s`, `Time_t`], all database operations executed by *Database Worker A* must have originated from the *Malicious Web Request* and no other. This *spatiotemporal anchor*, composed of the network connection tuple (identifying `Conn_A`) and the processing time window (identifying [`Time_s`, `Time_t`]), is the cornerstone of our method. It allows us to deterministically attribute behavior across the two independent, asynchronous execution domains.

DB Connection Analysis. Based on the principle above, this stage aims to extract the precise spatiotemporal anchor from the malicious request's syscall sequence. This anchor consists of the network 4-tuple (source IP, source port, destination IP, destination port) and the exact time window of its activity.

The analysis begins with the malicious syscall sequence from the *Request Partition* module. ANCORA parses the network-related syscalls in this sequence, filtering for those targeting the database server's IP and port. Upon such identification, it extracts the full network tuple. Concurrently, it records the timestamps of the first and last syscalls using this tuple during the request's lifecycle, defining the time window. This stage outputs one or more of these anchors (*Malicious Connection Fragments* in Figure 1), serving as input for the next stage.

DB Operation Extraction. With the spatiotemporal anchor, this stage "fishes out" the exact database operations triggered by the malicious request from the database server's logs. We shift our analysis to the database server's own syscall logs.

As established, a TCP connection maps to a specific database worker thread. We first use the network tuple from

[1]This model is a widely adopted architecture. We acknowledge that more complex patterns, such as the use of database proxies (e.g., ProxySQL) or advanced internal threading models in some modern databases, could disrupt this one-to-one mapping. We discuss this as a limitation in Section V.

the anchor to identify the corresponding worker thread (via its TID) in the database server's audit logs. After locking onto the target thread, we apply the time window constraint from the anchor, examining only the syscalls executed by this thread within that window. Specifically, we focus on `write` syscalls made to the database's own application log files (e.g., `PostgreSQL`'s statement log). Since the database logs operations after executing them, the content of these `write` syscalls contains the raw text of the executed database operations. This enables ANCORA to precisely extract the *Malicious DB Operations* from the voluminous, interleaved log I/O.

This method is broadly applicable. For some databases, such as `MongoDB`, the application-level audit log itself records the client IP and port. In such cases, the process is simplified: ANCORA can directly query the application log using the network and time information from the anchor, bypassing the need for syscall-level thread identification.

Rewind & Selective Replay. After identifying all malicious database operations, the final recovery stage aims to completely remove their effects while preserving all concurrent legitimate transactions. The process is as follows:

1) *Determine Recovery Baseline.* ANCORA identifies the earliest timestamp among the malicious operations and selects the latest clean database snapshot (*Periodic Database Backup*) created before this time.
2) *Rollback to Baseline.* The system uses the database's native tools (e.g., `pg_restore`) to revert the entire database to the chosen clean snapshot. This erases all changes, both malicious and legitimate, that occurred after the snapshot was taken.
3) *Prepare Legitimate Operations.* To restore lost legitimate data, ANCORA gathers the complete set of all database operations from the database's application logs for the period between the snapshot time and the present. It then filters out the operations previously identified as malicious.
4) *Selective Replay.* Finally, ANCORA re-executes the remaining list of legitimate database operations, in their original chronological order, on the restored baseline database.

Through this granular "rollback-filter-replay" process, ANCORA achieves a precise restoration of the database, producing a *Recovered DB State* that is both clean and maximally preserves legitimate user data, thus avoiding the unacceptable data loss inherent in traditional snapshot-only recovery.

E. System State Restoration

The final, recovered system state is achieved by composing the outcomes of the file system and database recovery processes detailed in Section III-C and Section III-D, respectively. Once the *Rewind & Selective Replay* mechanism has surgically repaired the file system and the database state, the system as a whole reaches the *Recovered System State* shown in Figure 1. By preserving all concurrent legitimate operations across both stateful components, ANCORA ensures business continuity and overcomes the critical data loss problem inherent in coarse-grained, snapshot-only rollback approaches.

IV. EVALUATION

We designed experiments to answer the following questions:

1) How accurately does ANCORA match database operations and file operations to web requests under high concurrency compared to the state-of-the-art?
2) How effective is ANCORA in recovering from real-world attacks while preserving legitimate data?
3) How does ANCORA's methodology enable superior attack tracing and recovery in a practical scenario where other methods fail?
4) What is the runtime and recovery performance overhead of ANCORA?

A. Evaluation Setup

To comprehensively evaluate ANCORA, we designed experiments covering diverse modern web application ecosystems and real-world attack scenarios.

TABLE I: Vulnerability scenarios used in our experiments.

App	CVE ID	Type	Language	Database
`FastAPI`	N/A	RCE	`Python`	`PostgreSQL`
`pgAdmin`	CVE-2022-4223 CVE-2023-5002	RCE RCE	`Python`	`PostgreSQL`
`Django`	CVE-2019-14234	SQLi	`Python`	`PostgreSQL`
`Mongo Express`	CVE-2019-10758	RCE	`Node.js`	`MongoDB`
`Joomla`	CVE-2017-8917 CVE-2015-8562	SQLi RCE	`PHP`	`MySQL`
`CMSMS`	CVE-2021-26120 CVE-2019-9053	RCE SQLi	`PHP`	`MySQL`
`Gitlist`	CVE-2018-1000533	RCE	`PHP`	No DB
`Solr`	CVE-2019-17558 CVE-2019-0193 CVE-2017-12629-RCE	RCE RCE RCE	`Java`	No DB
`Ofbiz`	CVE-2024-38856 CVE-2023-51467 CVE-2023-49070 CVE-2020-9496 CVE-2024-45507 CVE-2024-45195	RCE RCE RCE RCE RCE RCE	`Java`	`PostgreSQL`
`GeoServer`	CVE-2023-25157 CVE-2024-36401	SQLi RCE	`Java`	`PostGIS` (`PostgreSQL`)

Benchmark Applications and Vulnerabilities. To construct a realistic evaluation environment, we selected 9 open-source web applications and one custom application (see Table I). These applications cover four mainstream languages: `PHP`, `Python`, `Node.js`, `Java` and three database types - `MySQL`, `PostgreSQL`(both the standard version and one with the `PostGIS` extension), `MongoDB` - encompassing both SQL and NoSQL models. We reproduced 20 real-world, CVE-based attack scenarios, including high-severity vulnerabilities like SQL injection and RCE. All vulnerable environments, except our custom application, were built using images from the [38] project to ensure realism and reproducibility.

Instrumentation. We implemented framework-level instrumentation to enable request partitioning, requiring no modification to the application code:

- *Python*: Implemented via monkey patching, supporting `FastAPI` and other ASGI-compliant frameworks.

- *Node.js*: Implemented using the `async_hooks` mechanism, compatible with `Express.js`.
- *Java*: Implemented using a Java Agent, compatible with thread-pool-based applications like `Solr`.

Experimental Platform. All experiments were conducted on KVM-based virtual machines (Ubuntu 22.04.5, 8 vCPUs, 16GB RAM). All services were deployed in containers using Docker version 28.3.2 to ensure environmental consistency.

Workload Generation. We used `Locust v2.32.6` to generate concurrent http traffic with up to 150 users, sufficient to trigger complex request interlacing. The traffic targets real APIs, ensuring that **legitimate requests also induce complex state changes in the database and file system**, creating a highly interleaved access sequence that increases the evaluation's challenge and realism.

Ground Truth and Baselines. We established ground truth by running tests under single-concurrency conditions and manually labeling the causal relationships between requests and operations. We use SANARE [11] as the baseline for all comparative analyses.

B. Matching Accuracy

This section quantifies ANCORA's ability to accurately correlate http requests with their underlying file and database operations under high concurrency. This precise matching is critical for surgically removing attack impacts while preserving legitimate user data. We use Precision, Recall, and F1-Score for a direct comparison against SANARE.

1) **Matching Accuracy for Benign Operations.**

As shown in Table II and Table III, ANCORA demonstrates superior accuracy and stability when processing benign requests.

- **ANCORA**: Achieved nearly perfect matching accuracy (F1-Score=1.0) across almost all tests. A minor dip in `Mongo-Express` (F1-Score of 0.987) was caused by occasional event loss in the underlying `sysdig` tool under extreme load, not a flaw in ANCORA's mechanism.
- **SANARE**: Accuracy deteriorated sharply with increasing concurrency. As concurrency rose from 1 to 150, its F1-score for file and database operation matching dropped by an average of 68.63% and 60.25%, respectively. For instance, in `FastAPI`'s database matching, SANARE's precision plummeted from 100% to 13.91%.

These results show that SANARE's black-box approach fails to learn stable patterns in "noisy" high-concurrency environments, leading to massive misattributions. In contrast, ANCORA's deterministic correlation method is immune to such interference.

2) **Matching Accuracy for Vulnerability-Exploitation Operations.**

ANCORA's advantage is even more pronounced in real-world attack scenarios, especially those involving RCE.

- **ANCORA**: As shown in Table IV, ANCORA maintained a near-perfect F1-Score of 1.0 across all 17 real-world vulnerability scenarios [2] and all concurrency levels. This confirms its deterministic tracking can precisely follow unexpected execution paths created by exploits.
- **SANARE**: Completely failed to handle these attacks, with its F1-Score remaining at 0 in all tests. Its machine learning model could not recognize malicious inputs absent from training data, nor could it track execution paths that deviated from normal logic.

These experiments clearly demonstrate that ANCORA's matching method maintains near-perfect accuracy against realistic exploits, whereas the black-box approach fails entirely due to its inherent design limitations. This highlights ANCORA's decisive advantage in handling complex attack scenarios.

TABLE II: Precision(P), Recall(R), and F1-Score(F1) of Matching Benign Database Operations under Different Concurrency Levels: Comparison between ANCORA and SANARE

Concurrency		1		50		150	
App	Metric	ANCORA	SANARE	ANCORA	SANARE	ANCORA	SANARE
FastAPI	P	1.000	1.000	**1.000**	0.375	**1.000**	0.139
	R	1.000	1.000	1.000	1.000	1.000	1.000
	F1	1.000	1.000	**1.000**	0.545	**1.000**	0.244
Django	P	1.000	1.000	**1.000**	0.315	**1.000**	0.132
	R	1.000	1.000	1.000	1.000	1.000	1.000
	F1	1.000	1.000	**1.000**	0.479	**1.000**	0.234
pgAdmin	P	1.000	1.000	**1.000**	0.138	**1.000**	0.089
	R	**1.000**	0.722	**1.000**	0.630	**1.000**	0.621
	F1	**1.000**	0.839	**1.000**	0.227	**1.000**	0.157
Joomla	P	1.000	1.000	**1.000**	0.847	**1.000**	0.835
	R	**1.000**	0.714	**1.000**	0.678	**1.000**	0.598
	F1	**1.000**	0.833	**1.000**	0.753	**1.000**	0.697
CMSMS	P	1.000	1.000	**1.000**	0.352	**1.000**	0.382
	R	**1.000**	0.897	**1.000**	0.797	**1.000**	0.726
	F1	**1.000**	0.946	**1.000**	0.489	**1.000**	0.501
Ofbiz	P	1.000	1.000	**1.000**	0.225	**1.000**	0.224
	R	**1.000**	0.750	**1.000**	0.740	**1.000**	0.582
	F1	**1.000**	0.857	**1.000**	0.346	**1.000**	0.324
Mongo-Express	P	1.000	1.000	**1.000**	0.158	**0.987**	0.063
	R	1.000	1.000	**1.000**	0.666	**0.987**	0.829
	F1	1.000	1.000	**1.000**	0.256	**0.987**	0.117
GeoServer	P	1.000	1.000	**1.000**	0.931	**1.000**	0.841
	R	1.000	1.000	1.000	1.000	1.000	1.000
	F1	1.000	1.000	**1.000**	0.964	**1.000**	0.914
Average	P	1.000	1.000	**1.000**	0.418	**0.998**	0.338
	R	**1.000**	0.885	**1.000**	0.814	**0.998**	0.795
	F1	**1.000**	0.936	**1.000**	0.507	**0.998**	0.399

C. Recovery Effectiveness

This experiment evaluates the final recovery outcome. We use a strict accuracy criterion: a recovery is accurate only if the set of restored operations $\mathcal{P}$ is exactly equal to the ground-truth set $\mathcal{Q}$ (i.e., $\mathcal{P} = \mathcal{Q}$), ensuring no omissions or collateral damage. The accuracy is defined as:

$$Accuracy = \frac{\#\ of\ accurately\ recovered\ requests}{\#\ of\ total\ requests}. \quad (1)$$

[2] SANARE could not be trained on `Mongo Express` due to the absence of normal file operation data in the experimental scenario, and the two SQL injection vulnerabilities CVE-2019-9053 and CVE-2017-8917 could not be exploited to inject statements that modify the database.

TABLE III: Precision(P), Recall(R), and F1-Score(F1) of Matching Benign File Operations under Different Concurrency Levels: Comparison between ANCORA and SANARE

Concurrency		1		50		150	
App	Metric	ANCORA	SANARE	ANCORA	SANARE	ANCORA	SANARE
FastAPI	P	1.000	1.000	**1.000**	0.151	**1.000**	0.056
	R	1.000	1.000	1.000	1.000	1.000	1.000
	F1	1.000	1.000	**1.000**	0.262	**1.000**	0.106
Django	P	1.000	1.000	**1.000**	0.537	**1.000**	0.433
	R	1.000	1.000	1.000	1.000	**1.000**	0.997
	F1	1.000	1.000	**1.000**	0.699	**1.000**	0.604
pgAdmin	P	**1.000**	0.963	**1.000**	0.520	**1.000**	0.804
	R	1.000	1.000	**1.000**	0.707	**1.000**	0.520
	F1	**1.000**	0.981	**1.000**	0.600	**1.000**	0.632
Joomla	P	1.000	1.000	1.000	1.000	1.000	1.000
	R	1.000	1.000	**1.000**	0.962	**1.000**	0.963
	F1	1.000	1.000	**1.000**	0.980	**1.000**	0.981
CMSMS	P	1.000	1.000	**1.000**	0.180	**1.000**	0.152
	R	1.000	1.000	1.000	1.000	1.000	1.000
	F1	1.000	1.000	**1.000**	0.304	**1.000**	0.264
Gitlist	P	1.000	1.000	**1.000**	0.064	**1.000**	0.029
	R	1.000	1.000	1.000	1.000	1.000	1.000
	F1	1.000	1.000	**1.000**	0.121	**1.000**	0.056
Solr	P	**1.000**	0.000	**1.000**	0.000	**1.000**	0.000
	R	**1.000**	0.000	**1.000**	0.000	**1.000**	0.000
	F1	**1.000**	0.000	**1.000**	0.000	**1.000**	0.000
Ofbiz	P	1.000	1.000	**1.000**	0.027	**1.000**	0.014
	R	1.000	1.000	**1.000**	0.824	**1.000**	0.726
	F1	1.000	1.000	**1.000**	0.052	**1.000**	0.027
GeoServer	P	1.000	1.000	**1.000**	0.214	**1.000**	0.091
	R	**1.000**	0.586	**1.000**	0.455	**1.000**	0.512
	F1	**1.000**	0.739	**1.000**	0.291	**1.000**	0.154
Average	P	**1.000**	0.885	**1.000**	0.299	**1.000**	0.287
	R	**1.000**	0.843	**1.000**	0.772	**1.000**	0.858
	F1	**1.000**	0.858	**1.000**	0.368	**1.000**	0.314

TABLE IV: F1-Score of Matching Malicious Operations under Different Concurrency Levels: Comparison between ANCORA and SANARE

Concurrency	1		50		150	
App (Vulnerability)	ANCORA	SANARE	ANCORA	SANARE	ANCORA	SANARE
Self-Design, RCE	**1.000**	0.000	**1.000**	0.000	**0.999**	0.000
CVE-2022-4223, RCE	**1.000**	0.000	**1.000**	0.000	**1.000**	0.000
CVE-2023-5002, RCE	**1.000**	0.000	**1.000**	0.000	**1.000**	0.000
CVE-2015-8562, RCE	**1.000**	0.000	**1.000**	0.000	**0.993**	0.000
CVE-2021-26120, RCE	**1.000**	0.000	**1.000**	0.000	**1.000**	0.000
CVE-2018-1000533, RCE	**1.000**	0.000	**1.000**	0.000	**1.000**	0.000
CVE-2019-17558, RCE	**1.000**	0.000	**1.000**	0.000	**1.000**	0.000
CVE-2019-0193, RCE	**1.000**	0.000	**1.000**	0.000	**1.000**	0.000
CVE-2017-12629, RCE	**1.000**	0.000	**1.000**	0.000	**0.994**	0.000
CVE-2024-38856, RCE	**1.000**	0.000	**1.000**	0.000	**1.000**	0.000
CVE-2023-51467, RCE	**1.000**	0.000	**1.000**	0.000	**1.000**	0.000
CVE-2023-49070, RCE	**1.000**	0.000	**1.000**	0.000	**1.000**	0.000
CVE-2020-9496, RCE	**1.000**	0.000	**1.000**	0.000	**1.000**	0.000
CVE-2024-45507, RCE	**1.000**	0.000	**1.000**	0.000	**1.000**	0.000
CVE-2024-45195, RCE	**1.000**	0.000	**1.000**	0.000	**1.000**	0.000
CVE-2024-36401, RCE	**1.000**	0.000	**1.000**	0.000	**1.000**	0.000
CVE-2023-25157, SQLi	**1.000**	0.000	**1.000**	0.000	**1.000**	0.000
CVE-2019-14234, SQLi	**1.000**	0.000	**1.000**	0.000	**0.978**	0.000
Average	**1.000**	0.000	**1.000**	0.000	**0.998**	0.000

As summarized in Table V and Table VI, ANCORA's recovery effectiveness far surpasses the baseline.

- **ANCORA**: Attained 100% recovery accuracy across the vast majority of applications and concurrency levels, validating its robustness. Minor dips (e.g., 99.79% for `Django` at 150 concurrency) are consistent with occasional event loss from the underlying logging tool.
- **SANARE**: Performance degraded drastically as concurrency increased. Its average recovery accuracy for file operations dropped from 77.16% at single concurrency to 26.08% at 150 concurrency; for database operations, the average fell from 74.37% to 24.18%. In worst-case scenarios like `Gitlist`, accuracy fell to 0% at just 50 concurrency.

This performance gap stems from the systems' fundamental principles. In high-concurrency environments, operations from different requests are tightly interleaved. SANARE's statistical model is susceptible to this "crosstalk", erroneously associating temporally adjacent operations and thus failing our strict criterion. In contrast, ANCORA's deterministic causal tracing, built on provenance graphs, creates isolated causal chains for each request. This guarantees the atomicity and accuracy of recovery, maintaining exceptionally high fidelity even in highly concurrent scenarios.

D. Case Study

We present a case study on a multi-stage attack on `CMS Made Simple v2.2.9.1` to evaluate ANCORA's accuracy in a realistic scenario. The attack chain leverages CVE-2021-26120 to achieve remote code execution (RCE), causing modifications to both the database and the file system.

1) **Attack Chain.** The attack, requiring administrator privileges, proceeds in two stages:

- **Stage 1 (Template Injection).** The attacker injects a Server-Side Template Injection (SSTI) payload into a template via the backend editor. The payload exploits a vulnerability in the Smarty engine:

```
{function name='rce(){};
    system("curl -o /tmp/Webshell http://...")
    ;
function '}
{/function}endrce
```

- **Stage 2 (RCE and Backdoor).** When a user visits the page rendering this template, the server executes the payload, which downloads and saves a webshell.

This attack overwrites template content in the database and creates a malicious file on the file system.

2) **Recovery Evaluation.** We labeled the two attack requests as malicious and triggered recovery for both SANARE and ANCORA. To create a high-noise environment, we simulated 30 concurrent users for one minute, generating heavy, interleaved background traffic (e.g., page visits, configuration updates). This workload produced 385 total http requests, 374.43 MiB of syscall logs (598,432 events), 5,398 database write operations, and 2,128 file system modification operations. The recovery goal was to precisely revert the two malicious database writes and the single webshell file creation amidst this activity.

TABLE V: Database Operations Recovery Accuracy (%) of ANCORA and SANARE under Different Concurrency Levels

Concurrency	1		5		10		50		100		150	
APP	ANCORA	SANARE	ANCORA	SANARE	ANCORA	SANARE	ANCORA	SANARE	ANCORA	SANARE	ANCORA	SANARE
FastAPI	100.00	100.00	**100.00**	34.48	**100.00**	28.67	**100.00**	36.27	**100.00**	26.00	**100.00**	33.65
Django	100.00	100.00	**100.00**	86.73	**100.00**	75.53	**100.00**	57.40	**100.00**	42.80	**99.79**	39.72
pgAdmin	**100.00**	72.22	**100.00**	48.31	**100.00**	36.78	**100.00**	8.10	**100.00**	5.86	**100.00**	5.50
Joomla	**100.00**	50.00	**100.00**	30.77	**100.00**	25.88	**100.00**	26.43	**100.00**	22.73	**100.00**	19.62
CMSMS	**100.00**	72.73	**100.00**	7.14	**100.00**	0.98	**100.00**	0.00	**100.00**	1.45	**100.00**	0.83
Ofbiz	**100.00**	0.00	**100.00**	0.00	**100.00**	0.00	**100.00**	0.00	**100.00**	0.00	**100.00**	0.00
Mongo Express	100.00	100.00	**100.00**	47.22	**100.00**	29.89	**100.00**	10.92	**98.73**	10.38	**98.70**	9.73
GeoServer	100.00	100.00	100.00	100.00	**100.00**	98.97	**100.00**	93.14	**100.00**	84.87	**100.00**	84.35
Average	**100.00**	74.37	**100.00**	44.33	**100.00**	37.09	**100.00**	29.03	**99.84**	24.26	**99.81**	24.18

TABLE VI: File Operations Recovery Accuracy (%) of ANCORA and SANARE under Different Concurrency Levels

Concurrency	1		5		10		50		100		150	
APP	ANCORA	SANARE	ANCORA	SANARE	ANCORA	SANARE	ANCORA	SANARE	ANCORA	SANARE	ANCORA	SANARE
FastAPI	100.00	100.00	**100.00**	66.00	**100.00**	44.83	**100.00**	43.48	**100.00**	39.46	**100.00**	37.44
Django	100.00	100.00	**100.00**	90.83	**100.00**	86.67	**100.00**	67.97	**100.00**	67.72	**100.00**	62.57
pgAdmin	**100.00**	94.44	**100.00**	73.26	**100.00**	41.28	**100.00**	36.45	**100.00**	44.09	**100.00**	37.90
Joomla	100.00	100.00	100.00	100.00	**100.00**	98.33	**100.00**	95.23	**100.00**	93.59	**100.00**	95.61
CMSMS	100.00	100.00	**100.00**	46.27	**100.00**	25.69	**100.00**	1.38	**100.00**	1.96	**100.00**	1.18
Gitlist	100.00	100.00	**100.00**	22.40	**100.00**	3.47	**100.00**	0.00	**100.00**	0.00	**100.00**	0.00
Solr	**100.00**	0.00	**100.00**	0.00	**100.00**	0.00	**100.00**	0.00	**100.00**	0.00	**100.00**	0.00
Ofbiz	100.00	100.00	**100.00**	4.00	**100.00**	0.67	**100.00**	0.00	**100.00**	0.00	**100.00**	0.00
GeoServer	**100.00**	0.00	**100.00**	0.00	**100.00**	0.00	**100.00**	0.00	**100.00**	0.00	**100.00**	0.00
Average	**100.00**	77.16	**100.00**	44.31	**100.00**	33.55	**100.00**	27.39	**100.00**	27.31	**100.00**	26.08

We first evaluated SANARE, which failed in both stages. In the first stage (database recovery), the attack caused only two database writes. However, SANARE's time-window-based approach incorrectly identified 13 additional, legitimate writes from background traffic as malicious. This resulted in an **87.5% false positive rate** (13 out of 15), causing irreversible state corruption by rolling back valid user data. In the second stage (file system recovery), SANARE completely missed the webshell creation. The execution path (RCE leading to a file write) was not present in its training data, so it could not associate the request with the file operation. Consequently, the webshell persisted after the recovery attempt.

In contrast, ANCORA **achieved precise recovery in both stages**. For the database recovery, ANCORA correctly identified only the two malicious writes by tracking the request to its specific database connection (`172.18.0.3:50564 -> 172.18.0.2:3306`) and worker thread (`ID=337278`), precisely isolating the operations within their execution window (`T_start=<15:00:19.700041883>, T_end=<15:00:19.810219831>`). No legitimate data was affected. Similarly, for the file system recovery, ANCORA correctly identified the webshell creation. It constructed a full provenance graph from syscalls (Figure 4) that linked the http request through the `PHP` and forked `curl` processes to the final file write, allowing the recovery process to correctly delete the webshell without affecting other files.

In summary, this case study demonstrates that ANCORA can accurately recover from a multi-stage attack with interleaved background traffic. While SANARE suffered from a high false positive rate in database recovery and completely missed the file system damage, ANCORA's fine-grained causality tracking enabled a complete and precise restoration of the system state.

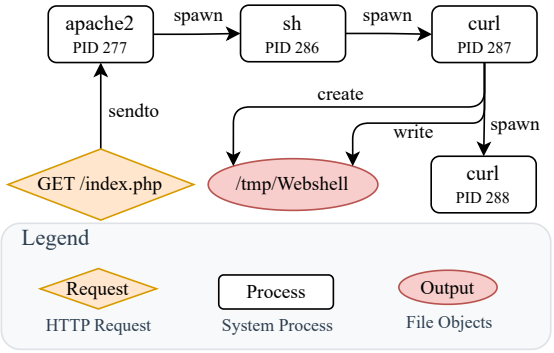

Fig. 4: File Provenance Graph of Case Study.

E. Performance Overhead

We evaluate the performance overhead of ANCORA in two phases: runtime and recovery. All experiments are conducted in the environment described in Section IV-A.

1) **Runtime Overhead.** Runtime overhead stems from two sources: computation for request attribution and recovery, and I/O for state backup.

a) Computation Overhead.

The computational overhead is caused by three main activities: (1) framework instrumentation for request partitioning (e.g., monkey patching `FastAPI`); (2) real-time syscall tracing with `sysdig` for causality tracking; and (3) periodic backup of the file system and database.

TABLE VII: Per-request Storage Overhead (KiB) for Different Applications under Various Conditions

App	GeoServer	Ofbiz	Solr	GitList	CMSMS	Joomla	Django	pgAdmin	FastAPI	Mongo Express
Storage Overhead (KiB)	118.77	1676.50	460.40	613.12	1615.27	612.43	94.81	190.74	344.71	78.07

We stress-tested ANCORA using `Locust` with concurrency up to 150, targeting API endpoints that trigger database and file system writes. In the worst-case scenario, ANCORA introduced a 19.8% increase in average response latency and a 17.8% decrease in Queries Per Second (QPS). As shown in Figure 5, the overhead varies with application type and concurrency:

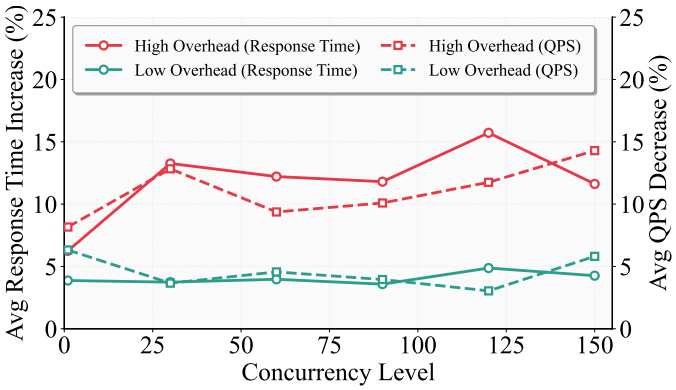

Fig. 5: Avg Response Time Increase and QPS Decrease for High-Overhead (`CMSMS`, `pgAdmin`, `Django`) and Low-Overhead Applications under Increasing Concurrency Levels.

- **High-overhead Applications:** For I/O-intensive or complex applications like `CMSMS`, `Django`, and `pgAdmin`, the response time increased by 6.25%–15.72%, and QPS decreased by 8.16%–14.29% as concurrency increased.
- **Low-overhead Applications:** For the remaining 7 applications, the overhead was minimal and stable. The average response time increase was 3.5%–4.9%, and the QPS decrease was 3.0%–6.3% at all concurrency levels.

In summary, the performance overhead of ANCORA remains within a reasonable range, demonstrating that its design achieves precise recovery capabilities while maintaining operational efficiency.

b) Storage and I/O Overhead.

This overhead is induced by `sysdig` logging, state backups, and our write-log kernel module. We measured storage consumption across tens of thousands of mixed requests on the ten applications from Table I. On average, ANCORA consumed 580.48 KiB of storage per request, with a maximum of 1.64 MiB. The per-request overhead, detailed in Table VII, varies significantly with application I/O patterns:

- **Lightweight:** `Django` and `Mongo Express` required only 94.81 KiB and 78.07 KiB per request, respectively.
- **Moderate:** Most applications, such as `GeoServer` (118.77 KiB), `pgAdmin` (190.74 KiB), and `Solr` (460.40 KiB), fell into a medium range.
- **I/O-intensive:** Complex applications like `Ofbiz` and `CMSMS` incurred the highest overhead, at 1.64 MiB and 1.58 MiB per request, respectively.

2) **Recovery Overhead.** We evaluate the efficiency and scalability of ANCORA's "rollback–filter–replay" recovery mechanism. To simulate attacks, we label a subset of requests as "malicious" to trigger recovery and measure the **recovery time** and **scalability**.

a) Recovery Time.

Recovery time is the duration to roll back the system state and replay legitimate operations. This time depends on the application and the extent of the damage.

TABLE VIII: Database Operations Recovery Time (ms) under Different Operation Counts

App	200	400	600	800	1000
`Mongo Express`	2233.142	3541.279	5016.447	6663.838	7981.847
`FastAPI`	881.530	1112.621	1334.913	1570.776	1825.448
`Django`	597.667	746.039	876.747	997.427	1167.025
`pgAdmin`	913.301	1064.009	1211.451	1478.809	1615.741
`Joomla`	1108.840	1254.408	1336.578	1458.148	1560.833
`CMSMS`	2828.231	3401.275	4064.171	5076.487	5581.244
`Ofbiz`	44248.147	44832.673	45158.676	45745.375	46209.707
`GeoServer`	5218.507	5415.005	5536.577	5811.295	6276.799
Average	7252.421	7670.925	8066.945	8600.269	9027.331

TABLE IX: File System Recovery Time (ms) under Different Numbers of Affected Files

App	20	40	60	80	100
`FastAPI`	37.654	62.901	92.632	116.481	145.346
`Django`	122.873	237.509	345.362	479.495	580.510
`pgAdmin`	106.504	223.449	328.780	412.827	539.478
`Joomla`	828.145	1754.703	2495.804	3400.287	4047.482
`CMSMS`	1547.066	2676.764	4241.084	5308.870	6765.862
`Gitlist`	237.192	458.865	664.263	862.598	1094.465
`Solr`	3248.172	6816.581	8415.476	11592.727	13923.036
`GeoServer`	747.417	1084.144	1555.632	1889.531	2261.270
Average	859.378	1664.365	2267.379	3007.852	3669.681

As shown in Tables VIII and IX, recovering from 1,000 malicious database operations took 9.03 s on average (max 46.2 s). Restoring 100 affected files took 3.67 s on average (max 13.9 s).

- **Database recovery:** Replaying 1,000 legitimate operations was fast for lightweight applications like `Django` (1.17 s) and `Joomla` (1.56 s), but took 46.2 s for the transaction-heavy `Ofbiz`.
- **File system recovery:** Restoring 100 files took 1.1 s in `Gitlist` (simple creation/deletion) but 13.9 s in `Solr` (complex modifications).

b) Scalability of Recovery.

We evaluated scalability by measuring recovery time against an increasing number of entities to restore. As shown in Figure 6, ANCORA exhibits excellent scalability. The recovery

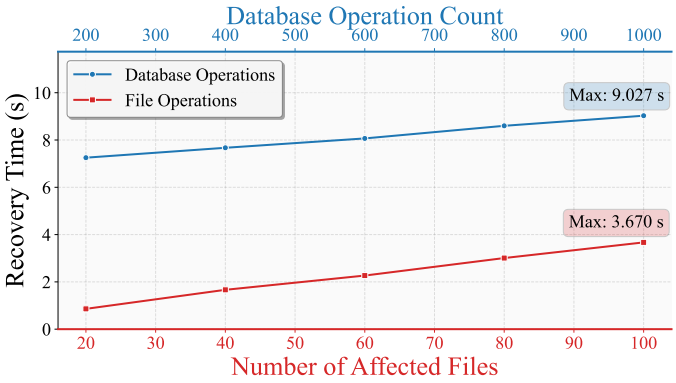

Fig. 6: Recovery Time Scaling with Increasing Workload

time for both database and file system restoration scales linearly with the number of replayed operations and repaired files. This demonstrates that the recovery overhead is predictable and manageable even for large-scale incidents.

V. Limitation and Discussion

1) **Advanced Database Architectures.** ANCORA's database operation tracing relies on a stable, one-to-one mapping between a client network connection and a database worker thread. While effective for standard deployments of `MySQL` and `PostgreSQL`, this assumption is violated by architectures involving database proxies (e.g., `ProxySQL`) or advanced concurrency models. These intermediaries obscure the client connection, breaking our attribution method. Extending ANCORA to these settings is a direction for future work.

2) **Recovery of Remote Services.** ANCORA is designed to restore the server's local state (file system and database). It can identify malicious outbound requests to external web services (e.g., payment gateways) but cannot automatically revert their effects. While ANCORA provides the forensic data for manual compensation, developing frameworks for automated cross-system remediation remains an open challenge.

3) **Coverage of Other Internal State.** Our current model focuses on file systems and databases. It does not cover other forms of state, such as distributed caches (e.g., `Redis`) or message queues (e.g., `Kafka`). An attack could poison these components, leading to delayed or cross-service damage. Extending ANCORA's causal tracking and recovery mechanisms to stateful middlewares is an avenue for future research.

VI. Related Work

Research on web application intrusion recovery can be broadly categorized.

- **Taint-based Analysis.** Systems like SHUTTLE [16] use deep instrumentation of the application runtime to propagate taint markers from http requests to low-level operations. This approach, while precise in theory, is highly invasive, making it difficult to deploy and maintain in complex production applications. It also fails to track operations from unexpected execution paths created by vulnerabilities like RCE. In contrast, ANCORA uses non-invasive, framework-level instrumentation and syscall-based causal analysis to cover all execution paths.
- **Black-box Correlation.** Systems like SANARE [11] and RECTIFY [17] treat the application as a black box, using machine learning to correlate high-level requests with low-level system events. Their accuracy degrades significantly under high concurrency due to the noisy and interleaved nature of system events. Furthermore, their file monitoring is often restricted to predefined directories. ANCORA achieves superior accuracy through deterministic partitioning and attribution, and its provenance analysis provides whole-system file tracking.
- **System-wide Causal Tracking.** Approaches like RETRO [39] build a global graph of system actions to support recovery. In a high-concurrency web environment, this leads to dependency explosion [22]–[27], creating a massive, tangled graph that is intractable to analyze. ANCORA's core design overcomes this by first partitioning the global log into independent, request-specific causal chains, then using a targeted method to solve the most complex dependency (the database), thus avoiding the need to analyze a monolithic graph.
- **Database-specific Recovery.** Work such as WARP [14] focuses on fine-grained database recovery by rolling back and selectively replaying non-malicious transactions. However, these systems assume that the malicious transactions are already identified and do not address the challenge of attributing transactions to a specific web request in modern architectures with connection pooling. ANCORA's key contribution is solving this attribution problem with our spatiotemporal anchor method, providing the missing link that makes rollback-and-replay strategies like WARP's practical in real-world settings.

VII. Conclusion

This paper introduced ANCORA, a novel system for precise and comprehensive intrusion recovery in modern web applications. To overcome the dependency explosion problem in high-concurrency environments, ANCORA first employs deterministic request partitioning using framework-level delimiter logging to isolate the syscalls of individual requests. Following this partitioning, ANCORA addresses state modifications across both the file system and the database. For the file system, it constructs a syscall-based provenance graph to uncover all attack-induced modifications, including those from unexpected execution paths. For the database, it deconstructs the asynchronous interaction model using a spatiotemporal anchor, the combination of a network connection and its active time window, to deterministically attribute database operations to the originating request. Our evaluation on 10 web applications employing databases including `MySQL`, `PostgreSQL`, and `MongoDB` across 20 CVE-based attack scenarios demonstrates that ANCORA achieves 99.9% accuracy under concurrency levels up to 150 connections. ANCORA successfully reverses attack-induced damage while preserving legitimate user data, with manageable performance overhead: 19.8% average response latency increase, 17.8% QPS reduction, and recovery

throughput of 110.7 database operations per second and 27.2 affected files per second.

REFERENCES

[1] O. Foundation, "OWASP Top Ten | OWASP Foundation." [Online]. Available: https://owasp.org/www-project-top-ten/

[2] J. Shahid, M. K. Hameed, I. T. Javed, K. N. Qureshi, M. Ali, and N. Crespi, "A comparative study of web application security parameters: Current trends and future directions," *Applied Sciences*, vol. 12, no. 8, p. 4077, 2022.

[3] 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 *2024 IEEE Symposium on Security and Privacy (SP)*. IEEE, 2024, pp. 3533–3551.

[4] J. Zengy, 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 *2022 IEEE symposium on security and privacy (SP)*. IEEE, 2022, pp. 489–506.

[5] S. Wang, Z. Wang, T. Zhou, H. Sun, X. Yin, D. Han, H. Zhang, X. Shi, and J. Yang, "Threatrace: Detecting and tracing host-based threats in node level through provenance graph learning," *IEEE Transactions on Information Forensics and Security*, vol. 17, pp. 3972–3987, 2022.

[6] Q. Wang, W. U. Hassan, D. Li, K. Jee, X. Yu, K. Zou, J. Rhee, Z. Chen, W. Cheng, C. A. Gunter *et al.*, "You are what you do: Hunting stealthy malware via data provenance analysis." in *NDSS*, 2020.

[7] A. Goyal, G. Wang, and A. Bates, "R-caid: Embedding root cause analysis within provenance-based intrusion detection," in *2024 IEEE Symposium on Security and Privacy (SP)*. IEEE, 2024, pp. 3515–3532.

[8] X. Han, T. Pasquier, A. Bates, J. Mickens, and M. Seltzer, "Unicorn: Runtime provenance-based detector for advanced persistent threats," *arXiv preprint arXiv:2001.01525*, 2020.

[9] C. Xiong, T. Zhu, W. Dong, L. Ruan, R. Yang, Y. Cheng, Y. Chen, S. Cheng, and X. Chen, "Conan: A practical real-time apt detection system with high accuracy and efficiency," *IEEE Transactions on Dependable and Secure Computing*, vol. 19, no. 1, pp. 551–565, 2020.

[10] X. Han, X. Yu, T. Pasquier, D. Li, J. Rhee, J. Mickens, M. Seltzer, and H. Chen, "{SIGL}: Securing software installations through deep graph learning," in *30th USENIX Security Symposium (USENIX Security 21)*, 2021, pp. 2345–2362.

[11] D. R. Matos, M. L. Pardal, and M. Correia, "Sanare: Pluggable intrusion recovery for web applications," *IEEE Transactions on Dependable and Secure Computing*, vol. 20, no. 1, pp. 590–605, 2023.

[12] P. Ammann, S. Jajodia, and P. Liu, "Recovery from malicious transactions," *IEEE transactions on knowledge and data engineering*, vol. 14, no. 5, pp. 1167–1185, 2002.

[13] İ. E. Akkuş and A. Goel, "Data recovery for web applications," in *2010 IEEE/IFIP International Conference on Dependable Systems & Networks (DSN)*. IEEE, 2010, pp. 81–90.

[14] R. Chandra, T. Kim, M. Shah, N. Narula, and N. Zeldovich, "Intrusion recovery for database-backed web applications," in *Proceedings of the Twenty-Third ACM Symposium on Operating Systems Principles*, 2011, pp. 101–114.

[15] R. Chandra, T. Kim, and N. Zeldovich, "Asynchronous intrusion recovery for interconnected web services," in *Proceedings of the Twenty-Fourth ACM Symposium on Operating Systems Principles*, 2013, pp. 213–227.

[16] D. Nascimento and M. Correia, "Shuttle: Intrusion recovery for paas," in *2015 IEEE 35th International Conference on Distributed Computing Systems*. IEEE, 2015, pp. 653–663.

[17] D. R. Matos, M. L. Pardal, and M. Correia, "Rectify: Black-box intrusion recovery in paas clouds," in *Proceedings of the 18th ACM/IFIP/USENIX Middleware Conference*, 2017, pp. 209–221.

[18] "Data backup & replication | veeam." [Online]. Available: https://www.veeam.com/products/veeam-data-platform/backup-recovery.html

[19] "Hpe zerto software | hpe." [Online]. Available: https://www.zerto.com/

[20] "Rubrik security cloud | rubrik." [Online]. Available: https://www.rubrik.com/products

[21] "Cloud data security and management platform | cohesity data cloud." [Online]. Available: https://www.cohesity.com/products/data-cloud/

[22] R. Wang, Y. Peng, Y. Sun, X. Zhang, H. Wan, and X. Zhao, "Tesec: Accurate server-side attack investigation for web applications," in *2023 IEEE Symposium on Security and Privacy (SP)*. IEEE, 2023, pp. 2799–2816.

[23] K. H. Lee, X. Zhang, and D. Xu, "High accuracy attack provenance via binary-based execution partition." in *NDSS*, vol. 16, 2013.

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

[25] 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 *26th USENIX Security Symposium (USENIX Security 17)*, 2017, pp. 1111–1128.

[26] M. Alhanahnah, S. Ma, A. Gehani, G. F. Ciocarlie, V. Yegneswaran, S. Jha, and X. Zhang, "autompi: automated multiple perspective attack investigation with semantics aware execution partitioning," *IEEE Transactions on Software Engineering*, vol. 49, no. 4, pp. 2761–2775, 2022.

[27] W. U. Hassan, M. A. Noureddine, P. Datta, and A. Bates, "Omegalog: High-fidelity attack investigation via transparent multi-layer log analysis," in *Network and distributed system security symposium*, 2020.

[28] Z. Li, Q. A. Chen, R. Yang, Y. Chen, and W. Ruan, "Threat detection and investigation with system-level provenance graphs: A survey," *Computers & Security*, vol. 106, p. 102282, 2021.

[29] M. N. Hossain, J. Wang, R. Sekar, and S. D. Stoller, "{Dependence-Preserving} data compaction for scalable forensic analysis," in *27th USENIX security symposium (USENIX Security 18)*, 2018, pp. 1723–1740.

[30] Z. Xu, P. Fang, C. Liu, X. Xiao, Y. Wen, and D. Meng, "Depcomm: Graph summarization on system audit logs for attack investigation," in *2022 IEEE symposium on security and privacy (SP)*. IEEE, 2022, pp. 540–557.

[31] H. Zhang, D. D. Yao, N. Ramakrishnan, and Z. Zhang, "Causality reasoning about network events for detecting stealthy malware activities," *computers & security*, vol. 58, pp. 180–198, 2016.

[32] D. J. Pohly, S. McLaughlin, P. McDaniel, and K. Butler, "Hi-fi: collecting high-fidelity whole-system provenance," in *Proceedings of the 28th Annual Computer Security Applications Conference*, 2012, pp. 259–268.

[33] A. Bates, D. J. Tian, K. R. Butler, and T. Moyer, "Trustworthy {Whole-System} provenance for the linux kernel," in *24th USENIX Security Symposium (USENIX Security 15)*, 2015, pp. 319–334.

[34] T. Pasquier, X. Han, M. Goldstein, T. Moyer, D. Eyers, M. Seltzer, and J. Bacon, "Practical whole-system provenance capture," in *Proceedings of the 2017 Symposium on Cloud Computing*, 2017, pp. 405–418.

[35] "pg_dump | extract a postgresql database into a script file or other archive file." [Online]. Available: https://www.postgresql.org/docs/current/app-pgdump.html

[36] "pg_restore | restore a postgresql database from an archive file created by pg_dump." [Online]. Available: https://www.postgresql.org/docs/current/app-pgrestore.html

[37] "mysqldump | a database backup program." [Online]. Available: https://dev.mysql.com/doc/refman/8.4/en/mysqldump.html

[38] "Open-source collection of pre-built vulnerable docker environments for security researchers and educators." [Online]. Available: https://vulhub.org/

[39] T. Kim, X. Wang, N. Zeldovich, and M. F. Kaashoek, "Intrusion recovery using selective re-execution," in *9th USENIX Symposium on Operating Systems Design and Implementation (OSDI 10)*, 2010.

[40] M. Du, F. Li, G. Zheng, and V. Srikumar, "Deeplog: Anomaly detection and diagnosis from system logs through deep learning," in *Proceedings of the 2017 ACM SIGSAC conference on computer and communications security*, 2017, pp. 1285–1298.

[41] F. Liu, Y. Wen, D. Zhang, X. Jiang, X. Xing, and D. Meng, "Log2vec: A heterogeneous graph embedding based approach for detecting cyber threats within enterprise," in *Proceedings of the 2019 ACM SIGSAC conference on computer and communications security*, 2019, pp. 1777–1794.

[42] Y. Xie, Y. Wu, D. Feng, and D. Long, "P-gaussian: provenance-based gaussian distribution for detecting intrusion behavior variants using high efficient and real time memory databases," *IEEE Transactions on Dependable and Secure Computing*, vol. 18, no. 6, pp. 2658–2674, 2019.

[43] S. M. Milajerdi, R. Gjomemo, B. Eshete, R. Sekar, and V. Venkatakrishnan, "Holmes: real-time apt detection through correlation of suspicious information flows," in *2019 IEEE symposium on security and privacy (SP)*. IEEE, 2019, pp. 1137–1152.

[44] W. U. Hassan, L. Aguse, N. Aguse, A. Bates, and T. Moyer, "Towards scalable cluster auditing through grammatical inference over provenance graphs," in *Network and Distributed Systems Security Symposium*, 2018.

[45] 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 *26th USENIX Security Symposium (USENIX Security 17)*, 2017, pp. 487–504.