SoK: History is a Vast Early Warning System: Auditing the Provenance of System Intrusions

Muhammad Adil Inam*, Yinfang Chen*, Akul Goyal*, Jason Liu*, Jaron Mink*, Noor Michael*, Sneha Gaur*, Adam Bates*, Wajih Ul Hassan†
*University of Illinois at Urbana-Champaign
†University of Virginia

2023 IEEE Symposium on Security and Privacy (SP) | 978-1-6654-9336-9/23/$31.00 ©2023 IEEE | DOI: 10.1109/SP46215.2023.10179405

***Abstract*—Auditing, a central pillar of operating system security, has only recently come into its own as an active area of public research. This resurgent interest is due in large part to the notion of *data provenance*, a technique that iteratively parses audit log entries into a dependency graph that explains the history of system execution. Provenance facilitates precise threat detection and investigation through causal analysis of sophisticated intrusion behaviors. However, the absence of a foundational audit literature, combined with the rapid publication of recent findings, makes it difficult to gain a holistic picture of advancements and open challenges in the area.**

In this work, we survey and categorize the provenance-based system auditing literature, distilling contributions into a layered taxonomy based on the audit log capture and analysis pipeline. Recognizing that the *Reduction Layer* remains a key obstacle to the further proliferation of causal analysis technologies, we delve further on this issue by conducting an ambitious independent evaluation of 8 exemplar reduction techniques against the recently-released DARPA Transparent Computing datasets. Our experiments uncover that past approaches frequently prune an overlapping set of activities from audit logs, reducing the synergistic benefits from applying them in tandem; further, we observe an inverse relation between storage efficiency and anomaly detection performance. However, we also observe that log reduction techniques are able to synergize effectively with data compression, potentially reducing log retention costs by multiple orders of magnitude. We conclude by discussing promising future directions for the field.

I. Introduction

Auditing is one of the fundamental tenants of operating system security [1]. Dating as far back as Anderson's seminal 1972 Computer Technology and Planning Study [2], audit capabilities were identified as essential in any resource sharing system to detect breaches and penetration attempts. Lampson identifies the three pillars of his access control "Gold Standard" as Authorization, Authentication, and Audit [3].[1] When *proactive* security measures like authorization and authentication fail, audit forms the basis for all forms of *reactive* security, allowing system defenders to identify and mitigate intrusions before they escalate.

In spite of its foundational importance, auditing has been largely ignored in the system security literature – little attention has been given historically to the design of efficient, secure, and effective auditing mechanisms. As a demonstrative example, the early 2000s saw the emergency of key OS security mechanisms such as the NSA's SELinux [4, 5] and Linux Security Modules [6], which sparked a flurry of research on the correctness of SELinux policies (e.g., [7–9]) and the placement of LSM's authorization hooks (e.g., [10–13]). Just a few years later, Red Hat released the Linux Audit Subsystem (LAuS) [14]; we are aware of no such efforts to specify secure LAuS auditing configurations or validate its correctness, even though LAuS was critical to Linux's certifications under the Common Criteria's Control Access Protection Profile (CAPP) [15] and Labeled Security Protection Profile (LSPP) [16].

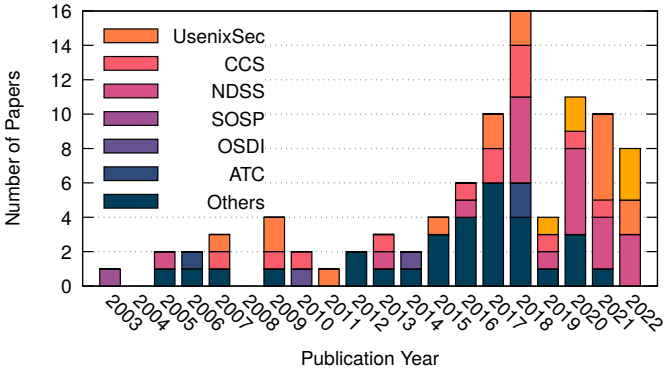

Fig. 1: Publication frequency for data provenance and system auditing papers over the past two decades. Auditing, an operating system security concept as foundational as access control and authorization, is a subject of resurgent interest in the literature.

In the present moment, however, our collective appreciation of auditing is beginning to change due to the failings of proactive security. Increasingly sophisticated and well-financed threat groups have demonstrated the ability to penetrate networks, seemingly at-will (e.g., [17–23]). Following their initial incursion, attackers can dwell for weeks or even months without notice, steadily inflicting more damage to the target [24]. That these attacks may take place over a period of many months creates an opportunity for reactive security approaches to detect and respond to intruders before they reach their ultimate objectives. Thus, auditing and log analysis techniques are more important than ever when defending systems [25–28].

As a result of this reckoning, system auditing is experiencing a well-deserved renaissance in the security literature. Figure 1 demonstrates this trend, plotting the number of system auditing papers to have appeared at notable conferences over the past two decades. As can be seen, the number of system

[1]Lampson's "Gold Standard" plays on the fact that all three pillars of access control begin with the letters "Au," the chemical symbol for gold.

DOI 10.1109/SP46215.2023.00018

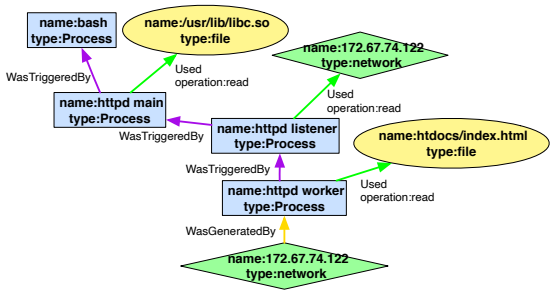

Fig. 2: Provenance graph represented in Open Provenance Model (OPM) [29] of Apache webserver serving single user request. Vertices represent the system subjects and system objects, while edges represent the causal relationship between those entities.

auditing papers to appear between 2004 and 2014 is equal to the number appearing in 2018 alone. This trend has largely been driven by the notion of *data provenance*. Provenance describes the totality of system execution and facilitates causal analysis of system activities by reconstructing the chain of events that lead to an attack (backward tracing) as well as the ramifications of the attack (forward tracing). However, the absence of a foundational system auditing literature, combined with the deluge of recent results, makes it difficult to gain a holistic picture of the state of the art of auditing. What are the central challenges posed by system auditing, and what progress has been made to overcome these obstacles?

In this work, we set out to answer these questions by conducting a comprehensive systematization of the system auditing literature. We focus our survey around recently-reported results on *data provenance* techniques, allowing us to understand how the notion of data provenance has shifted the landscape of system auditing. Our survey identifies and taxonomizes data provenance related techniques within five layers of the auditing stack: *Capture, Reduction, Infrastructure, Detection, and Investigation*. In so doing, we identify several layer-specific and cross-cutting challenges, then discuss the extent to which these obstacles have been addressed in prior work, providing the first holistic picture of provenance-based system auditing research.

Recognizing that the *Reduction Layer* represents a bottleneck to the further proliferation of provenance-based system auditing, we continue our study by conducting an independent comparative analysis of 8 exemplar log reduction techniques using the DARPA Transparent Computing datasets. We find that reduction performance can vary considerably based on the workloads of the target machine. We also observe a ceiling effect when attempting to apply these techniques in tandem; the most aggressive technique (S-DPR [30]) reduces log size by 87.3%, as compared to 90.7% when all techniques are applied. Moreover, we uncover a disturbing trend in which aggressive log reduction may lead to reduced anomaly detection performance. Encouragingly, though, we observe significant synergy between reduction techniques and traditional data compression – while filtering provides up to 10.8X log reduction and compression provides 22.8X, combined these techniques provide 185.4X reduction, allowing analysts to store years worth of logs in the space originally consumed by just a few days. We conclude by discussing our findings in the context of future directions for the field.

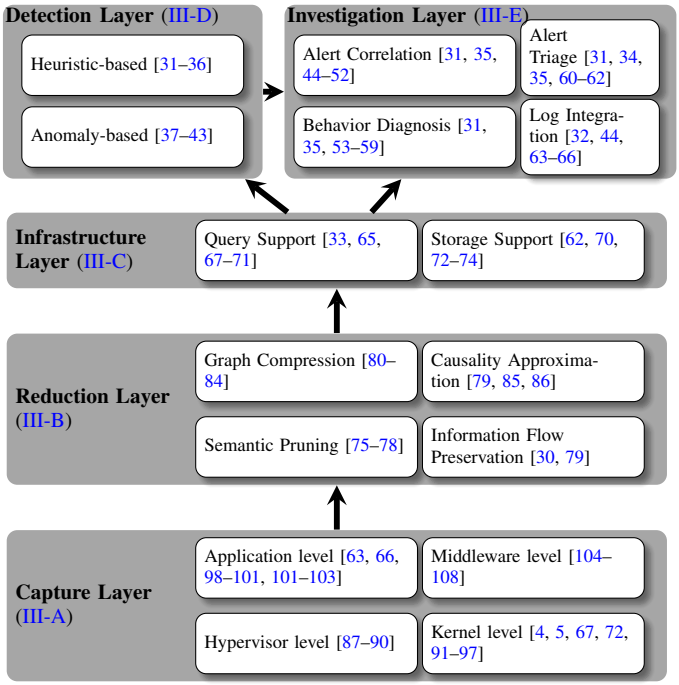

Fig. 3: We systematize provenance-based system auditing literature based on a taxonomy of the log capture and analysis pipeline.

II. Data Provenance Primer

Auditing captures documentary evidence of the events that took place on a target system.[2] Today, most operating systems come with auditing frameworks, such as the Linux Audit Subsytem [91], Event Tracing for Windows (ETW) [109], and FreeBSD's DTrace [110]. These frameworks use audit hooks and system call interception to monitor accesses between system subjects (e.g., processes) and system objects (e.g., files, sockets, pipes, etc.). Configured properly, audit logs can contain enough information to establish what events occurred on the system and how and who caused those events [111]. Auditing thus complements purely post-mortem forensic methods such as disk [112], memory [113], and malware analysis [114].

Historically, inspection of these logs has been tedious and error prone, involving long series of join statements to retrieve the relevant log entries from a relational database. To address this, recent literature has leveraged the notion of *data provenance*, a simple example of which is given in Figure 2. Rather than manually piece together individual pieces of evidence from raw logs, provenance-based systems can construct dependency graphs that explain the relationships between each event, simplifying the detection and investigation of attacks. Using these graphs, analysts can issue graph

[2]Our focus in this work is on auditing hosts/endpoints for evidence system intrusions. We note that, more broadly, auditing relates to a variety of other security topics such as cryptocurrencies electronic voting, which are out of scope in this work.

traversal queries to quickly identify the root causes of an attack (backtrace queries) or the impact of an attack (forwardtrace queries). A growing body of evidence seems to suggest that understanding the historical context of attacks is essential to addressing the shortcoming of current security products [31, 31, 34, 35, 51, 52, 62, 115, 116].

III. Provenance-based system auditing

In this section, we explore the literature on provenance-based system auditing. We organize contributions into five distinct layers according to the pipeline through which audit logs are collected and analyzed: 1) the **Capture layer** generates individual audit log records in various software architectures; 2) the **Reduction layer** eliminates extraneous information in the audit logs to improve storage, analysis, and query efficiency; 3) the **Infrastructure layer** processes and manages voluminous audit data; 4) the **Detection layer** performs automated analysis of audit data to search for indicators of attack; and 5) the **Investigation layer** exposes an interface for analysts to effectively inspect and interpret audit data. A visualization of this pipeline can be found in Figure 3. As some papers describe multi-layered or even end-to-end auditing systems, we decompose these works into individual contributions, discussing each in the appropriate subsections.

A. Capture Layer

The quality of security investigations depends on the collection of ample evidence, including the capture of raw streams of audit events. Typically, digital forensics entails extracting evidence from non-cooperative artifacts such as memory snapshots (e.g., [117]), or storage disks (e.g., [112]). The capture of audit logs is thus a peculiar tool in the forensic analyst's toolkit, as it is a runtime activity that must be configured in advance on a cooperating machine. However, not all audit streams are created equal – the vantage point used for auditing impacts the granularity of evidence and even the conclusions reached by analysts. Further, different approaches to log capture affect the security and deployability of the entire auditing pipeline.

Hypervisor-level Monitoring. In hypervisor-level monitoring, the target host is contained in a virtual machine while being audited by a virtual machine monitor (VMM). Such approaches are based on virtual machine introspection – whenever a guest application invokes an auditable event, the VMM examines the virtual machine's state, extracts the relevant event context, and generates a log event. The Backtracker system [87] is not only among the first hypervisor-level auditing frameworks, but also the first demonstration of data provenance as a means of investigating system intrusions. In Backtracker, both the event logging and the provenance graph generation components are present in the hypervisor.

More recent hypervisor-based approaches have taken advantage of the ease with which record-and-replay (RR) techniques can be embedded in VMMs [89, 90]. Most notably, Ji et al. [88] leverage this approach to enable replay of instruction-by-instruction execution of a virtual machine and collect high-fidelity data provenance. A key advantage of this approach is that, beyond the trace necessary to produce the replay, runtime auditing is not needed in an RR system; instead, analysts can extract evidence of the intrusion in a post hoc fashion at any level of granularity they wish.

Kernel-level Monitoring. In kernel-level monitoring, audit logs describe accesses to data entities using operating system abstractions such as processes, files, and network sockets. Because these objects are universally employed by higher software layers, kernel-level auditing provides a broad, system-wide perspective to the events on the system.

Kernel: System Call API. Commodity auditing frameworks, particularly the Linux Audit Subsystem (LAuS) [91] and Event Tracing for Windows (ETW) [92] provide capture services for many of the papers we discuss in the higher layers of the pipeline (e.g., SPADE [72]). LAuS primarily monitors the system call interface, creating a new log entry when a system call attempt matches one of its configurable auditing rules. LAuS and other commodity auditing frameworks were designed with consideration for the Common Criteria [15, 16] and other certifications, leading to certain features such as the ability to audit failed access attempts. To improve on the overheads of commodity audit frameworks, Ma et al.'s ProTracer [118] leverages custom instrumentation to perform targeted provenance capture of security-sensitive system calls. We note that the syscall API is not the only valid interface over which to audit the kernel. For example, Muniswamy-Reddy et al.'s influential PASS system [93, 94] instruments the virtual file system interface. Because the file abstraction is used as the basis for nearly all kernel services in UNIX systems, this approach grants similar insight into system events.

Kernel: Security API. Another promising approach within the kernel layer is to employ the security interface (e.g., LSM [6]) for auditing purposes. A key advantage of this approach is that the security interface has already been the subject of extensive scrutiny to verify that it can be used to mediate all security-sensitive operations to controlled data types within a system [10–13]. Thus, auditing this API inherits these assurances of completeness. Further, such systems facilitate "whole-system" provenance collection; this is because the security subsystem is invoked earlier in the boot sequence than userspace syscall auditing, facilitating the capture of a fully-connected provenance graph where all system activities can be traced back to `init`. A notable downside, however, is that the security interface is unable to reliably audit failed access attempts because it is is invoked after argument validating and discretionary access checks have already occurred. Audit streams at this interface may also be more difficult for analysts to interpret in comparison to the better-known POSIX API.

Many security modules, like SELinux's Access Vector Cache [4, 5], already provide basic auditing services to help administrators debug and interactively improve security policy. The notion of *provenance monitors*, a provenance-aware system that satisfies the reference monitor concept [2], was first proposed by McDaniel et al. [95] and explored in Pohly et al.'s Hi-Fi [96] system. Bates et al. [97] generalized this

Capture Level	Exemplar Systems	Platform Modification?	Program Modification?	Config / Policy Change?	Complete Mediation?	Tamperproof?	Verifiable?	Whole-System Provenance?	Inter-Process Tracing?	Exec. Unit Partitioning?	Semantic Insight?
		Deployability			Security			Forensic Capabilities			
Hypervisor	BackTracker [87]	●	–	–	✔	✔	?	✔	✔	✖	Low
Kernel: Syscall	LAuS [91]	–	–	●	?	✖	?	✖	✔	✖	Low
Kernel: Security	SELinux [4], LPM [97]	◐	–	◐	✔	✔	✔	✔	✔	✖	Low
Application	BEEP [98], Omegalog [63]	–	◐	–	✖	✖	?	✖	✖	✔	High
Middleware	Scippa [104]	●	–	–	?	✖	?	✖	✔	✔	Med.

TABLE I: Comparative evaluation of the potential deployability properties, security properties, and forensic capabilities of different provenance-based system auditing capture mechanisms. ● or ✔ denotes fully satisfying the property, ◐ denotes conditionally satisfying the property, and – or ✖ denotes that the property is not satisfied. ? indicates that evidence for or against the property is not presently available in the literature.

approach in Linux Provenance Modules (LPM) and provide additional tamperproofing and mediation features including a secure boot sequence, authenticated networking, and attested disclosure of audit streams from higher levels. Pasquier et al.'s Camflow system [67] replicates these approaches and demonstrates a higher-performance and easier-to-maintain model for provenance capture in the kernel.

Application Level Monitoring. Auditing at lower levels can paint an imprecise or incomplete picture of system execution. For example, system logs suffer from *dependency explosion* [98] – due to the opacity of internal process activities, each process output event must be assumed to be causally dependent on all preceding inputs, leading to false dependencies in the provenance graph. Using the application layer such issues can be addressed by integrating (or layering [63, 94, 97, 119]) system and application telemetry data.

A common approach to application monitoring is to instrument the programs of interest with custom audit capabilities. While early work suggested that developers could annotate their own programs with custom provenance libraries [120, 121], efforts shifted towards semi-automating the instrumentation process through program analysis. Lee et al.'s BEEP system [98] mitigates dependency explosion through decomposing the log activity of long-running event-based processes into autonomous units of work. Through observation of training runs of a program, BEEP is able to automatically identify the main event handling loop of the process, then instrument it with event logging such that the program explicitly declares when a new execution unit is beginning. Ma et al. propose MPI [99], a complementary technique that can perform execution partitioning over high-level data structures (e.g., browser tabs) when event handling loops are not present, but requires the user to annotate such data structures in source code. Beyond execution units, it is possible to extract additional application state of interest; for example, Inam et al. [103] leveraged program transformation to audit an applications' security-sensitive configuration fields in memory.

Ideally, it would be possible to extract the necessary information from the application level without depending on instrumentation. Ma et al. [102] observe that execution units can often be inferred by observing the log outputs of an application. Kwon et al. [100] develop an alternate modeling-based inference technique through the use of lightweight dual execution [122] to differentiate sequences of system calls with different inter-dependencies. Hassan et al. [63] demonstrate that execution units can be identified through static analysis of the existing event logging callsites in application binaries.

The opacity of system logs is especially apparent in desktop applications such as web browsers and windowing systems. Yang et al. [66] propose a user interface based execution partitioning scheme correlates system calls and individual UI elements and events. To improve forensic analysis in the context of browser-based attacks, Li et al. [101] design a mechanism that captures fine-grained details pertaining to the execution of JavaScript within the browser, such as navigation events and changes to the DOM.

Middleware Level Monitoring. Application level monitoring helps to address many of the limitations of pervasive system logging, but typically depends on assumptions about application instrumentation (e.g., developer annotations [99]), type (e.g., browser-based [101]), or behavior (e.g., contains logging statements [63]). In contrast, middleware level monitoring can often strike a balance between invasiveness and precision when auditing. Android middleware has been instrumented to trace [104] and even authenticate [105] inter-app communication to prevent confused deputies and diagnose device disorders. Similar to the trend noted above, Android middleware telemetry is often integrated [106, 107] with low-level system call metadata [123] to improve forensic capabilities. Middleware level monitoring has also been proposed as a method of holistic audit for the Internet of Things [108]. While we are not aware of any studies that investigate dependency explosion in middleware, it stands to reason that middleware events could be used for execution partitioning in a manner similar to Yang et al.'s UI-based scheme [66].

Challenges at the Capture Layer

Table I summarizes capture layer challenges.

Deployability. While higher layers of the auditing pipeline are largely system agnostic, the capture layer must consider the invasiveness and costs of system modification. We cluster host, operating system, and middleware modifications together as "Platform Modifications" to reflect their similar deployment costs; while lower-level approaches are typically associated with platform modification, the availability of commodity

kernel auditing frameworks such as LAuS and SELinux offer visibility into system events while only requiring changes to system configuration. Similarly, higher-level approaches often depend on program modification (e.g., BEEP [98]), although we note a promising trend in which this information can sometimes be transparently inferred (e.g., Omegalog [63]).

Security. We evaluate the security of audit capture mechanisms based on the extent to which they satisfy the reference monitor properties [2] of complete mediation, tamperproofness, and verifiability. Due to community efforts in the verification of LSM authorization hooks, we are confident that auditing at the Security API provides complete mediation (or, in this case, *observation*) of security-sensitive events in the system; therefore, a VM introspection system can also satisfy complete mediation (i.e., place watches on the authorization hooks). While it stands to reason that syscall (e.g., POSIX) or middleware (Android Permissions API) could also satisfy the complete mediation property, we are not aware of any formal demonstration that this is the case. In contrast, application level auditing cannot provide complete mediation.

When evaluating tamperproofness, we consider whether an audit framework or its logs can be manipulated/disabled by an adversary with root privilege. While hypervisor exploits are periodically discovered, hypervisor level auditing as a design concept is tamperproof, as is Security API auditing when deployed on a hardened least-privilege kernel (e.g., [97]). The same cannot be said of syscall auditing, where root privilege is usually sufficient to control the auditing subsystem. In fact, anti-forensic tampering of system logs is a widespread phenomenon [124], and it has recently been demonstrated that these records can be manipulated in memory [124] before tamper-evident protections (e.g., [74, 125, 126]) can be applied on disk. Middleware and application level auditing are also vulnerable to general tampering, and have recently been demonstrated to be subject to *execution repartitioning* [127] attacks in which a compromised program may equivocate about unit boundaries to confuse investigators.

We also consider whether different capture methods can be *verified* to audit the necessary security-sensitive events in a tamperproof/tamperevident manner. Verification entails that target implementation satisfies a high-level design specification; as such, we are aware of very little evidence in the literature supporting verifiable audit properties. At the Security API, Bates et al. demonstrate a policy-based auditing technique that can produce a minimally complete provenance graph for a target application [77, 128]. Their approach analyzes the system's SELinux type enforcement policy to determine which system entities may (in)directly influence the target application. At the Syscall API, LAuS has been certified as part of Linux's Common Criteria profiles (CAPP [15], LSPP [16]), but the requirements of these certifications fall short of reference monitor guarantees.

Forensic Capabilities. Finally, we highlight the forensic capabilities of different capture methods. Whole-system provenance depends on the audit mechanism being initiated prior to the first user space process, so it can be provided at the hypervisor or Security API, but not at commodity syscall auditing frameworks. However, capture approaches up to the middleware level support inter-process tracing of user space programs. Conversely, higher-level approaches provide deeper insight into program behaviors. Execution unit partitioning can consistently be achieved at the application level, and at times attainable within middleware. Beyond execution partitioning, richer forms of application-specific semantics can only be audited within the applications themselves. The forensic capabilities of different capture methods have a direct impact on the semantic insight of collected logs. While lower level approaches (hypervisor and kernel) offer a broad view of the system activity, they suffer from a notable semantic gap – the captured logs lack descriptions of higher-level application behaviors that are often pivotal to attack reconstruction. On the other hand, while higher level approaches do reduce the semantic gap, they often suffer from high deployability costs.

B. Reduction Layer

Audit frameworks are known to generate an enormous volume of logs, upwards of of gigabytes per day on a single machine [78, 97], posing serious storage and management challenges. At times, this creates a "needle-in-a-haystack" problem for analysts as they sift through extraneous log data searching for real evidence. Often, though, log volume ironically leads to the destruction of key evidence; companies are forced to purge old logs just days after their capture before an intrusion has even been detected [129, 130]. Prior work has attempted to develop techniques that reduce log size while retaining relevant forensic evidence.

Graph Compression. Generic compression tools, while useful for cold storage, do not retain logs in a queryable format and thus are not ideal for real-time investigations. The earliest reduction work in the provenance community sought to address this problem through the development and application of graph-specific compression schemes. Xie et al. [81–83] and propose adaptations of web graph compression and dictionary encoding schemes to provenance graphs. In their work, they leverage the web graph compression schemes and adapt them in the context of provenance graphs. Similarly, Ding et al. [131] leverage DNN based methods to learn optimal character encoding for audit events. As Xie's techniques were offline/batch-based, Ahmad et al. extend these approaches to streaming settings [80]. Most recently, Fei et al. adopt the data mining notion of Query-Friendly Compression to provenance graphs [84]. These compression techniques are able to reconstitute a lossless representation of the log, but add decompression latency and do not solve the "needle-in-a-haystack" problem.

Semantic Pruning. As the notion of provenance-based auditing took hold in the security community, the research observed that much of logs' contents described application activities that would never be of use in an investigation. Semantic pruning techniques aim to leverage the knowledge of application behaviors to filter out these events from the log. Lee et al.'s LogGC system performs graph analysis to

identify and prune temporary file I/O and other "dead end" application activities [75, 118], reasoning that dead events that do not inform the current state of the system are not useful in investigations. Ma et al.'s KCAL [78] framework featured a kernel-level cache to remove redundant causal events and reduce the overhead of log transfer from kernel to user-space. NodeMerge [76] builds on the observation that applications often load dozens of files at launch that are globally read-only (e.g., shared object libraries); allowing them to be reduced. Rather than pruning based on a general application behavior, Bates et al.'s ProvWalls [77] examines an application's security policy to determine the subjects and objects on the system that form its Trusted Computing Base (TCB), then prune all events that fall outside the TCB.

Information Flow Preservation. Beyond log events describing forensically-irrelevant application activities, an even greater source of overhead in logs is their sheer redundancy – applications consistently issue hundreds or thousands of repetitive system calls when performing I/O, far more than is necessary to accurately determine a causal link between entities. Information flow preservation seeks to address this redundancy by retaining only those log events that are necessary to produce an accurate information flow graph of system execution. Xu et al. propose this concept in their Causality Preserving Reduction (CPR) system [79]. Rather than naïvely eliminate every repeated system call between a source and destination entity, CPR tests for *interleaved flows*, i.e., whether any new inputs have been received at the source between the two system calls. An interleaved flow indicates that the system call may not actually be redundant and thus should be preserved. Ma et al.'s ProTracer [118] accomplishes a similar effect to CPR at the kernel level by alternating between taint analysis and logging. Hossain et al. further relax the assumptions of information flow preservation in their Dependency Preserved Reduction (DPR) systems [30]. Rather than preserve a full flow graph, DPR reasons that a reduced flow graph is sufficient as long as it can identify the same entities as the full graph when queried. To achieve this, the DPR systems selectively drop flow events that are not necessary to correctly traverse every entity's ancestors (S-DPR), or ancestors and successors (F-DPR). Thus, while the events that causally link the entities involved in an intrusion may be lost, an analyst is still able to correctly identify all of the implicated entities.

Causality Approximation. Beyond information flow preservation, a variety of systems have attempted to apply bounded approximation of audit logs, accepting some loss of accuracy in exchange for space efficiency. Xu et al.'s Process-centric Causality Approximation Reduction (PCAR) [79] extends the CPR system, aggressively eliminating redundant events (even for interleaved flows) when a "bursty" process exceeds some number of system calls per second. Hassan et al. propose Winnower, a framework for summarizing the provenance graphs of hundreds or thousands of replicated cloud applications using Deterministic Finite Automata (DFA) induction [85]. To account for non-determinism and other low-level variations between application instances, they define a set of heuristics for abstracting process, file, and socket labels. Michael et al. propose a similar approach to efficiently summarize repeated executions of the same program on a single host. Their system, LogApprox [86], performs bounded regular expression learning over file I/O events such that typical behaviors are approximated while atypical behaviors are losslessly retained.

		Forensic Validity			Analysis Costs		
Technique	**System**	*Node Loss?*	*Edge Loss?*	*Attr. Loss?*	*Query at rest?*	*Training Required?*	*Compute Complex.?*
Compress.	Generic	○	○	○	✗	✗	Local
	Web+Dict [82]	○	○	○	✓	✗	Local
	SEAL [84]	◐	●	◐	✓	✗	Local
Pruning.	LogGC [75]	●	●	○	✓	✗	Local
	NodeMerge [76]	●	●	○	✓	✓	Local
	ProvWalls [77]	●	●	○	✓	✗	Global
Preserve.	CPR [79]	○	●	○	✓	✗	Local
	DPR [30]	○	●	○	✓	✗	Global
Approx.	PCAR [79]	○	●	●	✓	✗	Local
	Winnower [85]	◐	●	●	✓	✓	Global
	LogApprox [86]	◐	●	●	✓	✓	Local

TABLE II: Comparative evaluation of the potential forensic validity concerns and analysis costs of reduction layer techniques. The remaining reduction layer challenge, storage efficiency, is explored at length in Section IV.

Challenges at the Reduction Layer

Table II summarizes reduction layer challenges.

Storage Efficiency. The goal of the reduction layer is to reduce the costs of storing and managing audit logs. Unfortunately, our understanding of storage efficiency is primarily based on self-reported findings by each system's authors. These results are often based on custom or closed-source datasets, making it difficult to compare the storage efficiencies of different systems or assess the general applicability of each approach. To address this shortcoming, we have independently re-implemented and evaluated a representative set of 8 exemplar reduction techniques against public datasets. We report our findings in Section IV.

Forensic Validity. Forensic validity was proposed by Michael et al. as a means of measuring the security utility of a reduced audit log [86]. They define metrics that can be used to evaluate log utility against specific attacks, but it is unclear whether their findings generalize to all forensic scenarios. Instead, we use the term here to describe general sources of context that may be stripped from the provenance graph by the reduction technique: node loss, edge loss, and the loss of any other source of non-structural graph information (i.e., attributes). As can be seen, semantic pruning techniques are associated with node and edge loss, which may undermine forensic validity if an attack relates to the application behaviors targeted by the technique. For example, the original LogGC implementation pruned network activity from closed sockets, suggesting that key evidence be lost in the case of data exfiltration attacks. In contrast, information flow preservation techniques only incur edge loss. However, the implications for

forensic validity are system-dependent – while CPR only drops "redundant" flow events, the DPR systems may drop unique flow events which fully severs the causal relation between two entities. Causal approximation techniques often "merge" distinct causal events together, resulting in the loss of certain causal attributes in the graph. For example, PCAR merges multiple information flows between the same two entities, obscuring the happens-before relations between events. LogApprox even merges flows to or from related data entities, such as files in the same directory, resulting in coarser-grained causal attribution. Regardless of the reduction technique, it can be difficult to reason about the tradeoff between space efficiency and forensic validity because it depends on assumptions about the suspected attacker's low-level behaviors.

Analysis Costs. A variety of performance considerations dictate the practical deployment of reduction techniques. While generic compression is advantageous for its ability to losslessly reproduce the provenance graph, its main drawback is the difficulty of querying the reduced log at rest. The latency associated with decompression is a serious throughput concern when analysts are attempting to react to an in-progress attack, especially because a portion of analyst queries to the log fit a random access profile [62]. A number of systems require training runs in order to function, which suggests ongoing administrative costs to ensure that their models of system activity stay up-to-date. As a coarse-grained measure of computational complexity, we consider whether each reduction technique can be applied to local/streaming graph segments or requires a global computation. As can be seen, there is no trend between technique categories and computational complexity. It is necessary for techniques to have reasonable throughputs and be applicable locally in order to scale, which is the central focus of the next layer.

C. Infrastructure Layer

The infrastructure layer is responsible for the storage and management of provenance and audit log data, making it accessible to the upper layers of the analysis pipeline. Compared to other layers, the infrastructure layer is particularly benefited by generic research on big data and distributed systems. Here, however, we focus on work that leverages the unique characteristics of audit and forensics to improve infrastructure support. For brevity, we have inlined our discussion of challenges.

Query Support. Provenance graphs pose an interesting problem for data querying – because analysts are often interested in the root causes of events, access patterns do not necessarily exhibit structural or temporal locality. As a result, relatively simple queries have been known to take days to return when processed by generic database management systems [68]. Prior work has attempted to reduce *time-to-insight* through the design of improved provenance query engines and interfaces.

Query Engines. If knowledge about attack patterns can improve space efficiency at the reduction layer, perhaps it can also improve query efficiency. Liu et al. propose such an approach in their PrioTracker [68] system, an intelligent graph traversal algorithm that prioritizes paths likely to lead to attack events. PrioTracker selects rare paths for further traversal based on a reference model of typical system behavior, and also de-prioritizes breadth searches of high-fanout nodes, to dramatically reduce query latency to disk-based graphs. However, intelligent traversal also suggests a new class of anti-forensic attack — an intruder patterns their behavior such that it falls on de-prioritized search paths — the feasibility of which has not yet been investigated. Hossain et al. demonstrate an alternate approach, Morse [33], which uses information flow tag propagation to produce consise responses to graph queries about attack behavior. Morse associates entities in the causal graph with information flow tags that assert their confidentiality and/or integrity states, then defines rules about the propagation and decay of tags within the graph to deprioritize unrelated benign activities during graph traversal.

Inspired by vertex-centric graph processing systems, Pasquir et al. present CamQuery [69], a mechanism that permits precompiled queries to be applied to provenance graphs in real time as they are constructed. By creating a streaming graph query engine, the CamQuery system is able to collapse the log capture and analysis pipeline such that query can be embedded within endpoint capture agents (e.g., a Linux Security Module). Inlining provenance capture and analysis also suggests a practical model for the deployment of provenance-based access control (e.g., [132–136]) at the operating system layer, although to our knowledge this has not been investigated. In concurrent work, Gao et al. propose SAQL [71], another non-vertex-centric streaming audit querying that offers more flexible multi-event support. Unfortunately, by their very nature forensic queries cannot be pre-compiled; as a result, this approach is amenable to detection and authorization but cannot improve query latency for post-mortem investigations.

Query Interfaces. While most analyst depends on relational (e.g., SQL) or graph (e.g., Cypher [137]) query APIs to access log data, research has also explored the design of specialized query interfaces for audit analysis. The CamQuery [69] system, discussed above, defines a custom language for expressing pre-compiled queries over system provenance. For forensic tasks, Gao et al. proposes domain-specific languages for querying attack behaviors [65, 70, 71], enabling analysts to retrieve multi-event log data by expressing sophisticated sets of contraints over event patterns and inter-event depencies. For example, in an APT intrusion exercise Gao demonstrates that AIQL can be used to detect exfiltration attempts by calculating a per-process moving averages of data transfers or forward track system activity for malware ramification [70].

Storage Support. Access performance for audit data is also dependent on the underlying storage structure. Prior work has leveraged the unique properties of audit data to improve on the performance provided by off-the-shelf solutions.

Data Model. Relational data models are poorly suited for storing audit data; in order to trace through a sequence of events in a relational model, analysts are forced to iteratively join an ever-expanding list of records together that represent connected graph components [138]. As a result, graph-based data models such as Neo4j [139] have been a popular op-

tion for provenance data stores (SPADE [72], LPM [97]). Unfortunately, general graph data models quickly struggle to support causal analysis; the access patterns of forensic queries lack the temporal and structural locality properties common to graph processing workloads (e.g., social networks). To address these shortcomings, specialized audit storage models have been proposed in the literature. Gao et al.'s AIQL system uses spatial and temporal properties of the audit log to partition data, reducing query latencies because related records can be retrieved in fewer data seeks [70].

Storage Backends. Regardless of data model, audit logs can either be stored on disk or in memory. Disk-based approaches (e.g., [34, 68]) suffer from significant I/O overheads, resulting in hours to respond to a single query at times [62], potentially delaying attack investigations. In contrast, while in-memory storage (e.g., [36, 97, 140–142] can largely mask query latencies to forensic analysts, the unwieldy size of provenance graphs on a long-running system can make this approach impractical. Based on analyzing the access patterns of provenance graph processing, Hassan et al. propose a hierarchical storage system for audit data [62]. Their approach leverages two in-memory event pools, one of which caches recent events to exploit the temporal locality of graph construction while the second retains the most suspicious events according to an anomaly scoring algorithm.

Distributed Storage. Attackers destroy logs after compromising a machine [28], underscoring the importance of replicating and distributing audit data. In most enterprise networks today, log data is transmitted by endpoints to a centralized storage server, leaving a single point of failure in the event of an attack. In contrast, Gehani and Lindqvist's Bonsai system provides distributed storage of provenance graphs for faster analysis, including a replication scheme to prevent partial loss of the graph as a result of offline nodes [143]. One challenge when storing audit data in a distributed fashion is the need to authenticate log/graph data when it is accessed [136]. Gehani and Kim's Mendel builds on the Mendel approach to support distributed storage across multiple trust domains by efficiently verifying subgraphs of the reconstituted graph as they are traversed in queries [73]. More recently, Paccagnella et al.'s Custos system simultaneously replicates logs as it verifies them for tamper-evidence in near-real-time through use of an SGX-based decentralized challenge protocol [74].

D. Detection Layer

The Detection Layer in the provenance-based system auditing pipeline is responsible for analyzing audit streams in an automated fashion in order to alert analysts to potential threats. Based on detection techniques, host intrusion detection can be broadly classified into two categories as heuristic- (or rule-) based and anomaly-based detection. Unlike whole-system intrusion detection, malware detection works in the context of a single program where program samples are typically analyzed in sandbox environments [144–147] for extracting signature-based [148, 149], or behavior-based [150–154] features. Signature-based schemes inspect program code and data for evidence of malware signatures, but struggle to detect previously-unseen samples (0-days) or obfuscated malware. Alternatively, behavior-based schemes observe program behaviors (execution paths, system calls, etc.) with monitoring tools and are able to detect obfuscated malware as well. Below, we consider how two classic approaches to host intrusion detection, heuristic- (or rule-) based and anomaly-based, have grown to incorporate data provenance and causal analysis in recent years.

Heuristic Detection. Heuristic detection approaches leverage existing or anticipated knowledge of attack behaviors to define event matching rules, which are then matched against the audit stream to detect attacks. Today, most Endpoint Detection & Response (EDR) products employ a heuristic approach, using rules based in part on the MITRE ATT&CK knowledge base of attacker Tactics, Techniques, and Procedures (TTPs) [155]. Recent work has considered how to adapt EDR-like event matching rules to causal graphs. Milajerdi et al.'s HOLMES system defines a set of graph matching rules based on TTPs related to advanced persistent threat (APT) behaviors [31]. When multiple alerts are fired from the same event stream, HOLMES is then able to construct a high-level scenario graph describing the attacker's behavior. Hossain et al. define TTP-like rules in their SLEUTH system, with the added constraint that these rules only fire when certain confidentiality or integrity conditions are satisfied according to a tag-based information flow propagation system [36]. Rather than constructing graph patterns to match TTP's, Milajerdi et al. also demonstrate that these event patterns can also be procedurally inferred from the text in Cyber Threat Intelligence (CTI) reports [32]. Bridging the gap between legacy EDRs and causal graph analysis, Hassan et al.'s NoDoze [34] and RapSheet [35] systems accept alerts fired by existing TTP rulesets, then perform causal analysis to assess the severity of the alerts. In contrast to the process-centric rules of legacy EDRs, a potential advantage of graph-based heuristic detection is the potential to express more complex matching rules that span multiple processes in the graph structure.

Anomaly Detection. Rather than speculating about the nature of future attacks, anomaly-based detection defines a model of typical system behavior based on historical log data, raising an alert if execution deviates significantly from this model. Anomaly detection is also a classic approach to a variety of security tasks that has more recently been applied to causal graphs. One challenge with anomaly-based detection is converting complex and arbitrarily large graphs into fixed-length vectors that can be used for modeling. Manzoor et al.'s Streamspot demonstrates a viable cluster-based modeling approach [39]. In Streamspot, a label is created for each node based on a local graph traversal, then converted into multiple fixed-length chunks. All graph chunks are then hashed to produce a fixed-length binary vector in a manner that preserves cosine similarity, then summed together to describe the entire graph. Han et al.'s Unicorn [38, 40] also visits each node to create a label but uses these labels to produce a histogram description of the graph that is then hashed into a fixed-

Technique	Systems	Known Attacks?	0-Day Attacks?	Alert Fatigue?	Event Attrib.?	Subgraph Attrib.?	Host Attrib.?
		Type 1		Type 2			
		Detection Error			Explainability		
Heuristic	Legacy EDR [34, 35]	●	◐	✔	●	–	–
	Prov EDR [31, 36]	●	◐	✔	–	●	–
Anomaly	Whole Graph [38–40, 158]	◐	●	?	–	–	●
	Path-based [37, 157, 159]	◐	●	?	–	●	–

TABLE III: Comparative evaluation of detection layer techniques.

length vector that preserved Jaccard similarity [156]. Unicorn also uses an evolving graph model to account for changes to benign system activity over time. Wang et al. suggest that a potential issue with the above techniques is that they attempt to describe *all* system behavior, resulting in a noisier model that may advantage attackers. Their ProvDetector [37] system addresses this by first downsampling the causal graph to a limited number of paths based on an out-of-band anomaly scoring algorithm [34]. ProvDetector also adapts doc2vec and Local Outlier Factor (LOF) as the embedding model and learning models, respectively. Similar to [37], Xie et al.'s PIDAS [157] assigns an anomaly degree to each path within a provenance graph to identify intrusions. PIDAS-Graph [158] extends PIDAS and calculates the anomaly degree of the whole provenance graph to detect intrusions that are described in multiple paths. More recently, Xie et al. proposed P-Gaussian [159] that uses a gaussian distribution scheme to additionally detect variants with transformed intrusion behavior sequences. Recent work has also shown that causal graph learning is effective at detecting specific attack behaviors such as malware installation [41]. While our focus here is on causal graph anomaly detection, we note the rich literature of sequence-based log analysis, including recent works such as DeepLog [42] and DeepCase [43].

Challenges at the Detection Layer

Table III evaluates detection layer challenges.

Detection Error and Overhead. Detection error continues to undermine intrusion detection systems as they move to causality-based approaches. We evaluate these systems based issues related to Type 1 (i.e., False Negative) and Type 2 (i.e., False Positive) errors. Anomaly-based approaches are better positioned to detect wholly unanticipated 0-day attacks that are not contained in heuristic knowledge bases. Conversely, Heuristic-based approaches are capable of detecting known attack patterns with precision. In practice, we would expect either technique to provide some detection capability against either attack type; this is because 0-day attacks will likely be partially comprised of known attack patterns, while known attacks will likely be partially comprised of behaviors that deviate from normality. Historically, both heuristic- and anomaly-based detection systems are prone to high rates of false positives, creating threat alert fatigue problems for analysts [160]. Whether causality-based detection suffers from these problems has not been investigated, although it seems a likely possibility. Regarding detection overhead, while these systems are typically designed for streaming/evolving (e.g., [38, 39]) settings, offline detectors (e.g., [34, 37]) may induce high detection latency.

Explainability The explainability of machine learning outputs is of increasing concern in the AI community [161]. In this context, explainability refers to a detection model's ability to explain to an analyst the circumstances that caused it to raise an alert. When considering the frequency of false alarms in detection models, explainability is key to helping an analyst quickly investigate (or dismiss) an alert. Although provenance graphs offer an intrinsic explanation of system activity, a detection model that returns an entire system graph, perhaps comprised of millions of nodes, is too coarse-grained an explanation. As heuristic-based approaches are defined based on reasonable small event patterns, it is simple for these detection models to return a small provenance subgraph that explains the context of the alert. Current work on anomaly-based detection has primarily focused on graph classification, offering poor explanations for alarms. Systems which downsample the whole graph [37, 159], offer a reasonable trade-off between these extremes because they can isolate the anomalous activity to just a handful of paths in the graph. We note that explainability is not an innate limitation of anomaly-based approaches; if future work investigated node classification within causal graphs, it would be able to offer finer-grained explanations similar to heuristic approaches.

E. Investigation Layer

The investigation layer enables the analysts to perform threat alert validation and post-mortem analysis of incidents. These tasks are related to, and at times interdependent on, complementary approaches such as memory forensics (e.g., [162, 163]) or disk forensics (e.g., [164, 165]). Memory forensics acquires and explores the semantic contents of interest from volatile memory, whereas disk forensics entails extracting forensic information from digital storage media. The analysis of volatile memory provides valuable live evidence during forensic investigations, and is typically easier to perform compared to disk analysis that can be hindered through data encryption [166, 167]. At times, the evidence being extracted may in fact be fragments of log events (e.g., [168].) Thus, while auditing is by no means the only form of forensic investigation, it is telling that 75% of incident response specialists consider logs to be the most valuable form of investigation artifact [28].

We divide the investigation layer into four subtasks performed by the analysts during the investigation process. Given an alert generated by the detection layer, the analyst first correlates that alert with all the alerts that happened in the past. After that, the analyst triages those correlated alerts based on their severity. Once alerts are triaged, the analyst generates contextual history to validate given alerts. Finally, the analyst analyzes the generated contextual information to understand

attack behaviors and initiates appropriate incident response and recovery process.

1) Alert Correlation: Existing threat detection systems are prone to high rates of false alarms, leading to issues of threat alert fatigue [160]. Therefore, a common procedure is to correlate and cluster alerts to reduce the total number of incidents that need to be investigated. These techniques can be divided into similarity-based and causality-based approaches.

Similarity-Based Correlation. In similarity-based techniques, alerts are clustered based on alert attributes' statistical and temporal similarity. Pei et al.'s HERCULE [44] system used the Louvain method of community detection to discover attack communities in heterogeneous audit logs and then leveraged these attack communities as a basis to derive correlations among threat alerts. Valdes and Skinner [45] designed an alert correlation system that finds similarities among attributes of different alerts using probabilistic reasoning. Valeur et al. used a sliding window that stored recently triggered alerts in a time-ordered queue [46]. This queue allowed authors to derive correlations among alerts based on the timing similarity metric. Similarly, Debar and Wespi [47] proposed an alert correlation methodology that not only correlates alerts based on time sequence but also finds duplicate alerts (i.e., alerts that are related to the same attack) using clustering algorithms and fuse such alerts together. Similarity-based correlation is widespread in industry, where security information and event management (SIEM) tools [48–50] employ statistical- and rule-based similarity metrics to correlate and aggregate alerts.

Causality-Based Correlation. Causality-based alert correlation techniques leverage information flow between different system entities, such as network sockets and processes, to derive alert correlation. Zhai et al. [51] were the first to use kernel-level audit logs to derive correlation among alerts. They leverage the Backtracker system [87] to generate dependency graphs from kernel-level audit logs and then use these graphs to correlate threat alerts. Similarly, HOLMES [31] and Rapsheet [35] leverage data provenance to derive correlations among threat alerts. On the other hand, BotHunter [52] leverages network-level communication to identify the stages of a botnet infection and correlate those stages. What is more, CLARION [115] and ALASTOR [116] consider constructing and investigating provenance in the cloud computing context.

2) Alert Triage: To further alleviate threat alert fatigue, the analysts perform alert triage to investigate high severity alerts before low severity alerts. DEPIMPACT [61], NoDoze [34] and Swift [62] prioritize generated alerts based on their anomalous contextual history. NoDoze and Swift first assign an anomaly score to each event in the provenance graph based on the frequency with which related events have happened before in the enterprise. While DEPIMPACT calculates the weight scores for edges according to multiple features including timing, data flow amount, and node degree. After that, they aggregate those anomaly scores along the neighboring edges of the provenance graph and use those aggregated scores to triage alerts. Unfortunately, NoDoze and Swift are only applicable if anomalous events exist in the network. Stealthy attacks often use "living-off-the-land" attack strategies where attackers abuse benign applications to perform their actions and avoid anomalous activities. To triage alerts related to such actions, RapSheet [35] leverages the notion of the tactical provenance graphs (TPG) that, rather than encoding low-level system event dependencies, reasoned about causal dependencies between threat alerts. RapSheet proposed a threat scoring scheme that assesses each alert's severity based on its TPG, enabling effective triage of alerts. Elsewhere in literature, Zhong et al. [60] mined past analysts' security operation traces to learn alert triage rules and then used these rules to automate the alert triage process.

3) Log Integration: To aid threat alert investigation, it is important to collect and integrate audit logs from different vantage points to gain better contextual information. Besides system logs collected from the kernel layer, the application layer also provides important contextual information related to the system execution. Hassan et al.'s OmegaLog [63] merges application event logs with the system log to generate a universal provenance graph (UPG). This graph combines the causal reasoning strengths of whole-system logging with the rich semantic context of application event logs. HERCULE [44] correlates entries from dispersed and heterogeneous application logs based on common features present in the logs. To extract such features from the logs, the authors wrote several correlation rules. In a similar vein of research, ALchemist [64] applies log normalization techniques and manually-written Datalog rules to integrate application and system logs without any program instrumentation. Besides application logs, UIScope [66] claims that the native UI elements and events of the GUI applications can be used to offer meaningful contextual information for causality analysis. It applies these high-level GUI components to help partition the low-level system logs generated in the long-running process without instrumentation, helping to solve the dependency explosion problem. Finally, ThreatRaptor [65] extracts threat behaviors from open-source Cyber Threat Intelligence text using NLP techniques and combines such behaviors with the audit logs. ThreatRaptor then enables the analyst to hunt these threat behaviors using a domain-specific query language.

4) Behavior Diagnosis: Several systems have recently been proposed to help analysts quickly understand the attack behaviors from low-level audit events. Zeng et al.'s [53] WATSON automatically abstracts and clusters high-level system behaviors from low-level audit events. WATSON performs Depth-First Search on each data object to summarize the system behavior and then uses the machine learning techniques to infer each audit event's semantics based on its contextual usage. The semantics of each behavior is obtained by aggregating the semantics of each composed event. Finally, behaviors with similar semantics are clustered together in the embedding space. Other learning-based techniques have also been proposed to facilitate analysts to interpret the provenance graph and help attack investigation [54–58]. Another work, ProPatrol [59], also tries to distinguish high-level system behaviors. Unlike BEEP [98] which needs instrumentation, ProPatrol performs

fine-grained execution partition using derived inference rules based on the domain knowledge of the Internet-facing application. For instance, a tab is an execution unit for browser applications, and an email is a unit for an email service application. ProPatrol regards each execution unit as a bin and classifies the incoming syscall to its corresponding bin. Other machine learning techniques, specifically natural language processing, can also be leveraged to achieve efficient attack scenario construction [54]. HOLMES and RapSheet provide high-level attack visualizations from low-level audit logs using MITRE TTPs to accelerate the investigation process.

Challenges at the Investigation Layer

A summary of the investigation layer challenges is given in Table IV and below we discuss those challenges in detail.

Automation. The goal of the investigation layer is to allow the analyst to quickly perform forensic analysis. Unfortunately, several techniques require analyst involvement that can slow down the whole investigation process. During the alert correlation subtask, statistic- and rule-based techniques require analysts to manually write parsers and rules to aggregate threat alerts. In contrast, information flow based techniques automatically extract causality between threat alerts without any analyst involvement. To prioritize threat alerts, behavioral-based techniques, such as [35] and [31], require the analyst to manually assign scores to each TTP for prioritization. On the other hand, anomaly- and learning-based techniques automatically perform alert triage. During the log integration subtask, methods that employ program analysis and parsing rules often require the analyst to provide annotations and write parsers to fuse different log entries, while NLP and temporal sequencing techniques require no analyst engagement. Finally, during attack behavior diagnosis, techniques based on learning and MITRE TTP are fully automated, while rule-based techniques asks analyst to write inference rules to generate high-level application tasks.

Accuracy. When considering different investigation techniques, it is essential to consider the accuracy of different techniques. In the case of alert correlation and log integration, statistic- and rule-based approaches are less accurate than information flow-based and program analysis techniques because such techniques capture true causality, not merely correlations. Moreover, alert correlation and log integration techniques that leverage temporal ordering suffer from low accuracy because several programs generate unordered and intertwined log entries due to multithreading and asynchronous programming models. Finally, to bridge the semantic gap between low-level audit events and high-level attack behaviors, all the existing techniques provide high accuracy.

	Technique	Systems	OS/Network	Application	GUI	Threat Intelligence	Automation	Accuracy
			Logs Used					
Alert Correlation	Statistic/Prob.	[44, 45]	✔	✔	✖	✖	◐	◐
	Temporal Ordering	[46, 47]	✔	✖	✖	✖	●	○
	Statistic/Rule	[48–50]	✔	✔	✔	✔	◐	◐
	Information Flow	[31, 35, 51, 52, 115, 116]	✔	✔	✖	✖	●	●
Alert Triage	Anomaly	[34, 61, 62]	✔	✖	✖	✖	●	–
	Behavioral	[31, 35]	✔	✖	✖	✖	◐	–
	Learning	[60]	–	–	–	–	●	–
Log Integration	Program Analysis	[63]	✔	✔	✖	✖	◐	●
	Rule/Pattern Match	[32, 44, 64]	✔	✔	✖	✔	◐	◐
	NLP	[65]	✔	✖	✖	✔	●	◐
	Temporal Ordering	[66]	✔	✖	✔	✖	●	○
Behavior Diagnosis	Learning	[53–58]	✔	✖	✖	✖	●	●
	Inference/Rule	[59]	✔	✔	✖	✖	◐	●
	MITRE TTPs	[31, 35]	✔	✖	✖	✖	●	●

TABLE IV: Description of various subtasks performed during the alert investigation process. The solidness of the marked circle reflects the automation level and accuracy of technique: High (●), Medium (◐), and Low (○);

IV. Efficiency Analysis of Reduction Techniques

Audit logs are invaluable to forensic audits, but can quickly grow to unwieldy sizes [78, 86, 169]. In practice, fine-grained logs are quickly discarded – if captured at all [129, 130, 170] – preventing the real-world use of the provenance-based investigation techniques that have gained popularity in the literature. Recognizing that the *Reduction Layer* represents a bottleneck to the further proliferation of provenance-based system auditing, we now continue our evaluation of the reduction layer by conducting the first empirical comparative analysis of the space efficiency of different audit log reduction techniques. To do so, we have re-implemented 8 exemplar log reduction techniques from the literature: LogGC [75], NodeMerge [76], CPR [79], F-DPR [30], S-DPR [30], PCAR [79], LogApprox [86], and a graph induction technique based on Winnower [85] that can be applied to individual processes instead of Linux containers. To the best of our knowledge, this is the first systematic comparative study of log reduction techniques.

A. Dataset

We make use of datasets released by DARPA Transparent Computing Program, Engagements 3 and 5 [171]. These engagements were conducted in 2018 and 2019, respectively. Each engagement contains event streams from multiple hosts and documented ground truth of the attacks conducted on the machines. We make use of the Linux-based datasets from hosts running the Theia and Trace systems. We refer to these 4 datasets as: E3-Theia, E3-Trace, E5-Theia, and E5-Trace, which respectively comprise 6.9 GB, 10.5 GB, 56 GB, and 252 GB of log data when translated to Linux audit log format.

B. Measurement Setup

We implement the reduction techniques as a filter mechanism that operates over a stream of log events. The mechanism is implemented in C++ and builds an in-memory graph representation of the streaming events using the SNAP graph library [172]. On an event-by-event basis, we invoke each reduction technique to determine if the event should be reduced, then

Dataset	Application	Events	Reduced Events of Application							
			LogGC	NodeMerge	CPR	F-DPR	S-DPR	PCAR	LogApprox	Induction
E3-Theia	fluxbox	24%	1%	<1%	1%	99%	99%	2%	92%	1%
	stat	11%	10%	8%	13%	90%	94%	30%	57%	7%
	firefox	13%	13%	13%	15%	94%	96%	37%	56%	5%
E3-Trace	firefox	76%	2%	4%	10%	19%	21%	12%	12%	<1%
	/home/admin/du	6%	<1%	<1%	11%	56%	65%	11%	2%	0
	thunderbird	2%	13%	11%	21%	73%	82%	27%	18%	2%
E5-Theia	bash	23%	<1%	5%	75%	95%	97%	93%	3%	1%
	stat	16%	<1%	2%	62%	96%	97%	70%	8%	17%
	landscape-sysinfo	8%	<1%	65%	11%	70%	87%	18%	52%	14%
E5-Trace	Xvnc4	53%	0	<1%	72%	>99%	>99%	>99%	<1%	0
	pulseaudio	9%	17%	33%	19%	85%	92%	35%	28%	2%
	mandb	3%	16%	33%	16%	85%	92%	33%	30%	2%

TABLE V: Reduction rates of different techniques for the Top 3 applications of each dataset by number of system calls. For each technique, the green and red shaded cells denote their best and worst performance across the listed applications.

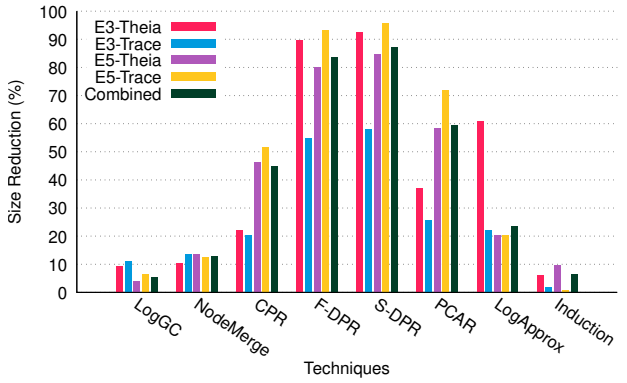

Fig. 4: Observed reduction percentages for each technique.

tally its decision. While many techniques can be applied to a local subgraph (e.g., LogGC), other techniques, such as DPR, perform a global graph operation. We release the implementation of these techniques to foster future research[3]. To adapt these techniques to a streaming setting and reduce memory consumption, we process log events in batches of 100,000, then invoke the technique. This approach may underestimate the space efficiency of the global techniques; that said, we experimented with larger batch sizes (up to 200,000) and observed at most a 0.7% change in events reduced across all techniques combined.

C. Space Efficiency of Individual Techniques

We first report on the space efficiency of individual reduction techniques in terms of log size reduction in Figure 4.[4] Most techniques' performance was roughly consistent with the reported performance from their original papers, with DPR boasting the strongest reduction rates. DPR has been observed to delete attack-relevant data [86] (a trend we further validate in IV-G), so we handle it as a separate case throughout this evaluation. In contrast, a system that appears to underperform as compared to original reports is LogGC [75]. The reason for this is that the DARPA TC datasets chose not to log many of the termination events (e.g., EXIT and CLOSE), preventing the LogGC algorithm from activating. When we tested LogGC against in-lab datasets, we found its performance to be more consistent with its original evaluation, although it is perhaps telling that the DARPA performers deemed these events unnecessary. Another underperforming system is our Winnower-like induction algorithm [85]; this approach was designed for reducing provenance graphs from identical containers, and seems to struggle as a general tool.

Surprisingly, we also observe that the performance of the different log reduction techniques varied significantly by dataset. In the most extreme case, the DPR techniques achieve over 90% reduction in E5-Trace but under 60% reduction in E3-Trace. E3-Trace contains a significantly large proportion of process events as compared to other datasets. Most reduction techniques target I/O behavior, which explains the lower overall reduction rates for E3-Trace. This result is concerning, as it suggests that log reduction may not be a general solution but is instead workload-specific.

D. Variance in Space Efficiency By Workload

To gain a better understanding of the observed variances in reduction between datasets, we mapped each log record in the datasets back to its issuing process to track the most active applications. Table V reports on the reduction rates for the Top 3 applications by the number of system calls. On this level, significant variance in reduction rates can be observed – S-DPR ranges from 21% to over 99%, and LogApprox ranges from under 1% to 92%. The performance also varies by the behavior of the application, as can be seen by comparing Firefox in E3-Theia to Firefox in E3-Trace. These results underscore the difficulty of reporting on the general efficacy of log reduction techniques; results must be qualified by the kinds of applications and workloads used in the evaluation.

E. Synergy Between Reduction Techniques

A previously unexplored topic is whether applying multiple reduction techniques leads to greater space efficiency. Because our filter mechanism attempts to apply every technique to every log event, we are able to track the extent to which techniques synergize or are redundant. Consider two techniques A and B with filtered event counts a and b, respectively. One option would be to use Jaccard similarity, but this coefficient

[3] https://bitbucket.org/sts-lab/faust

[4] We also analyzed space efficiency in terms of events dropped, but observed no significant difference in trends as compared to size reduction.

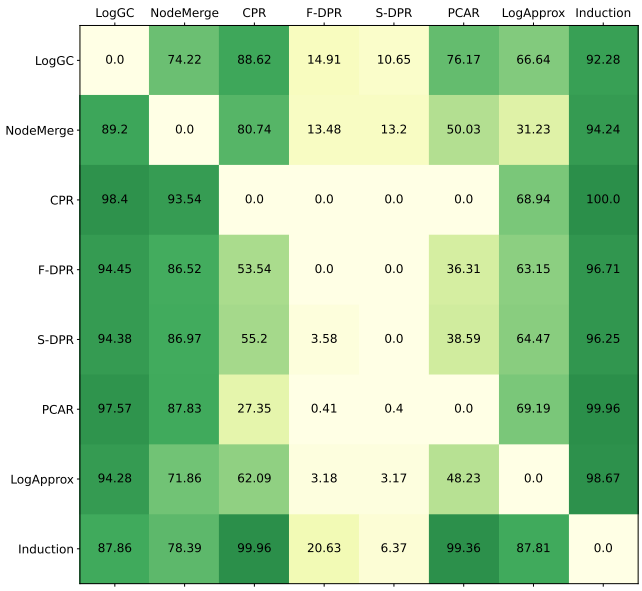

Fig. 5: Percentage unique events filtered by technique A (row), in comparison to technique B (column) i.e. |a−b|/|a| - Higher value represents greater uniqueness and vice versa.

	Technique 1	Technique 2	Technique 3	Size Red.
Without DPR	PCAR	Induction	-	65.77% (2.9X)
	PCAR	NodeMerge	-	66.94% (3.0X)
	PCAR	LogApprox	-	67.96% (3.1X)
	PCAR	LogApprox	LogGC	71.93% (3.6X)
	PCAR	NodeMerge	Induction	72.55% (3.6X)
	PCAR	LogApprox	Induction	73.84% (3.8X)
With DPR	S-DPR	LogApprox	-	88.32% (8.6X)
	S-DPR	LogGC	-	88.53% (8.7X)
	S-DPR	NodeMerge	-	89.17% (9.2X)
	S-DPR	LogGC	LogApprox	89.48% (9.5X)
	S-DPR	NodeMerge	Induction	89.63% (9.6X)
	S-DPR	NodeMerge	LogGC	90.24% (10.3X)
All Techniques Combined				90.70% (10.8X)

TABLE VI: Top 3 2-sized and 3-sized subsets of techniques (with and without DPR) for all 4 datasets combined and their corresponding reduction rates i.e both log reduction percentage (1 - Reduced Log / Raw Log) and in parenthesis the log reduction factor (Raw Log / Reduced Log).

would be misleading when a and b differ significantly in size. We instead calculate the percentage of unique events filtered by A as compared to B as $|a-b|/|a|$.

The results across all datasets are visualized in Figure 5. In the figure, each matrix row visualizes the percentage unique reduction by the given technique compared to the technique in the column. Intuitively, techniques that are the basis of other techniques do not filter any unique events in comparison, such as CPR for F-DPR, S-DPR, and PCAR. Reading down the column for S-DPR, the most aggressive filter, indicates which filters can provide any supplementary benefit – LogGC and NodeMerge, two of the worst performers in isolation, are most effective at filtering events that S-DPR does not. In fact, reading across the LogGC and NodeMerge rows, it appears that semantic pruning techniques consistently filter a unique subset of events in comparison to other techniques. Unfortunately, most other approaches offer only limited synergy with DPR.

To confirm this, we calculate the total log reduction when multiple techniques are applied. Table VI reports on the size reduction when two, three, or all techniques are applied for all 4 datasets combined. We report size reduction in terms of both percentage of original log size as well as the "log reduction factor," which can be interpreted as the number of times the reduced log can be fit into the storage footprint of the original log. Withour DPR in consideration, total reduction increases from 66-68% with two techniques to 72-74% with three techniques. With DPR, total reduction increases from 88-89% with two and to 89-90% with three. Unfortunately, there are diminishing returns as more techniques are applied – with all techniques combined, we achieve just 90.7% reduction, just a single order of magnitude. We conclude that current reduction techniques are highly redundant, filtering nearly the same amount combined as the top individual performers.

Technique	% Reduction	Reduction Factor
Reduce Only	90.70	10.8X
Hybrid Only	64.13	2.8X
Gzip Only	95.61	22.8X
Reduce & Hybrid	95.47	22.1X
Reduce & Gzip	99.46	185.4X

TABLE VII: Comparison of log reduction rates between different filtering techniques and 2 compression schemes for all 4 datasets combined. For ease of reference, we report both of the 2 different statistics used in prior work; log reduction percentage (1 - Reduced Log / Raw Log) and the log reduction factor (Raw Log / Reduced Log).

Data Format	Mean (s)	Min (s)	Max (s)	Std (s)
Uncompressed	0.000416	0.000035	0.016698	0.001193
Hybrid comp.	0.005701	0.000342	0.51132	0.026425
Gzip comp.	0.581273	0.580776	0.59905	0.001321

TABLE VIII: Query times for 500 random backtrace queries on the provenance graph associated with E3-Theia.

F. *Synergy with Compression*

Although we observed diminishing returns when applying multiple reduction techniques, there may exist synergy between log reduction and data compression. However, because data compression agressively deduplicate sources of redundancy in the log without regard to its structure or semantics, it is not a foregone conclusion that reduction techniques can provide any added value. To investigate, we make use of two data compression techniques – simple gzip compression, and a hybrid graph compression scheme based on Xie et al.'s [81, 82]. The hybrid scheme performs web compression on the graph structure as well as dictionary encoding on edge and vertex labels, to deduplicate data. Notably, while querying a gzip-compressed log requires the entire file to be decompressed, the hybrid scheme supports decompression of individual paths in the graph.

Storage Overhead. We report on the total storage overhead across the four datasets combined in Table VII. The first three rows report individual reduction results for the combined reduction techniques, hybrid compression, and gzip compression, while the last two rows report on reduced-

Data	% Reduction	DeepLog						StreamSpot		ProvDetector
		TP	FP	TN	FN	Precision (%)	Recall (%)	AS Attack	AS Benign	% top-k AP retained
Unfiltered	0	27	24	48253	4	52.94	87.10	0.52	0.08	100
Induction	6.18	27	24	48253	4	52.94	87.10	0.52	0.08	100
LogGC	9.35	27	24	48253	4	52.94	87.10	0.52	0.08	100
NodeMerge	10.4	27	24	48253	4	52.94	87.10	0.52	0.08	100
CPR	22.04	25	29	48248	6	46.30	80.65	0.49	0.08	95
PCAR	37.13	25	29	48248	6	46.30	80.65	0.49	0.08	95
LogApprox	60.77	23	47	11415	7	32.86	76.67	0.56	0.15	85
F-DPR	89.52	4	63	5226	24	5.97	14.29	0.44	0.27	55
S-DPR	92.45	4	78	5104	24	4.88	14.29	0.44	0.25	55

TABLE IX: Detection results of 3 representative anomaly detection systems for all 8 reduced datasets. The first row represents the baseline numbers for the unfiltered dataset. Accuracy numbers are omitted as they are not representative of forensic utility due to large number of TNs (AS = anomaly score, AP = anomalous paths).

then-compressed logs.[5] To our surprise, however, pre-filtering the log using reduction techniques results in a significantly smaller storage footpring than pure compression - *a company that previously could only store 3 days worth of logs [129] before purging them would be able to store 32 days with log filtering only, 68 days with gzip compression only, and 1.5 years when applying both filtering and gzip compression!* This suggests that reduced logs can be very efficiently retained in cold storage. We also observe encouraging synergy for the more query-friendly hybrid compression, enabling 67 days of retained data given an original budget of 3 days.

Query Overhead. To compare query overhead, we randomly select 500 nodes from the E3-Theia dataset and measure the time taken to complete a backtrace query using each compressed representation. The baseline query runs on uncompressed logs, while in the gzip test the log must first be fully decompressed, whereas in the hybrid test we are able to lazily decompress the graph based on the paths of the backtrace traversal. The query times for all 3 cases are reported in Table VIII. In this experiment, we use only a subset (15%) of the E3-Theia dataset, which is why even the slowest compression system (GZip) enjoyed sub-second response times. This performance would degrade on larger graphs; the real takeaway is Gzip compression on average incurs 1450X more query overhead compared to the baseline, while the hybrid scheme provides middle ground by only increasing the query overhead by 13X.[6]

G. Forensic Utility of Reduced Logs

We next set out to determine if reduction techniques impact the forensic utility of the audit log. To do so, we analyzed the reduced logs using three exemplar anomaly detection systems: DeepLog[7] [42], StreamSpot [39], and ProvDetector [37]. Due to the high costs of model training over large datasets, we performed this experiment only on E3-Theia. To create comprehensive ground truth labels for the attack events in the dataset, we performed back traces on each of the attack steps described in the E3 documentation, then manually pruned the results to remove false dependencies. We then applied each reduction technique on the labeled dataset.

Using the optimal configurations reported in the original papers, we then trained each anomaly detection system using a 70/30 train/test split. DeepLog performed classification at the granularity of individual log sequence windows of 1000 ms of log data, of which there were almost 50 thousand. We report TP, FP, TN, FN, recall and precision for this model.[8] Streamspot and ProvDetector perform whole-graph classification, but the test dataset is arguably just one graph. To circumvent this obstacle, we constructed two synthetic test samples, one fully-benign graph and one fully-malicious graph. While not realistic, these can be thought of as an upper bound for the performance of the classifier because there is no "noise" in the attack graph. For Streamspot, we report the mean anomaly score for the attack and benign samples. For ProvDetector, which downsamples the graph to k paths before classification ($k = 20$), we report the proportion of paths from the unfiltered attack log that were still selected for embedding in the reduced version of the log.

The results are shown in Table IX. Across all systems, we observe an inverse relation between storage efficiency and anomaly detection performance. For DeepLog, low-reduction techniques are not associated with any change in detection performance, while the most aggressive techniques see a dramatic drop in the model's ability to detect attack sequences and avoid false positives. Results are less extreme for Streamspot and ProvDetector, both of which correctly detected an anomaly in the synthetic attack graph. However, it is clear for both systems that the distinction between attack and benign activity is beginning to blur. For Streamspot, the difference in attack/benign anomaly scores closes as reduction techniques grow more aggressive, while in ProvDetector up to 45% of the paths identified as suspicious have changed when DPR is active. PCAR and CPR, moderately space efficient reduction

[5]We note that the "Hybrid Only" and "Reduce & Hybrid" results are actually not for a log representation of the data, but for the size of two serialized data objects describing the log. This representation without compression is nearly identifcal in size to the raw log file.

[6]Unfortunately, we are unable to provide a comparison with SEAL [84], concurrent work that uses a hybrid compression scheme but is unreleased and was not evaluated on public datasets.

[7]We utilized a 3rd party implementation of DeepLog for our experiments: https://github.com/nailo2c/deeplog

[8]Accuracy can be calculated from the provided data, but is highly misleading due to the large number of true negatives.

techniques, appear to offer the best tradeoff between reduction performance and forensic utility.

V. Future Directions

In this work, we have conducted a thorough analysis of the provenance-based system auditing literature using both qualitative and empirical methods. Throughout Section III, we identify a number of open challenges in isolated layers of log capture and analysis pipeline – the need for verified implementations at the capture layer, efficient query capabilities at the infrastructure layer, assessment of alert fatigue issues at the detection layer, etc. *We have also developed a unified framework for the evaluation of future log reduction techniques, which will be open sourced upon publication.* We conclude by considering cross-cutting challenges that implicate multiple layers of our taxonomy, discussing possible directions of future research for each.

Cross-Cutting Challenge #1: Reducing Time-to-Insight. The need to extract actionable insight from massively dense provenance graphs has been a driving force for research in provenance-based system auditing. Combatting dependency explosion (e.g., [98]), removing forensically-irrelevant events (e.g., [75]), and unifying multiple event streams (e.g., [63]) all provide partial solutions, but there is still more work to be done. In particular, we anticipate that the community will soon reach the limits of what can be achieved through technical solutions alone – time-to-insight is fundamentally a socio-technical problem.

Future research should engage expert analysts and industry stakeholders to better understand today's challenges and the potential implications of novel causal analysis techniques. We are already seeing commercial products that incorporate provenance-like graph visualizations to explain alerts (e.g., Crowdstrike [173], Sophos [174], and Comodo [175]) – does this information improve threat investigation efficiency, and if so how? Can it be made more effective through incorporating techniques from the academic literature? While we are encouraged by academic-industry partnerships that have anecdotally shed light on real-world issues (e.g., NEC Laboratories [68], Symantec [35]), what is most sorely needed is principled measurement and intervention through human subjects research. We envision recruiting analysts for surveys, laboratory experiments that track their threat hunting practices in a controlled environment, experimental manipulations that test the efficacy of causal analysis techniques, and new software systems that are co-designed and evaluated against such experiments. Through such study, the academic community will gain deeper insights into the challenges of adapting and using provenance-based system in practice, and even gain a sense of the efficacy of these systems against attacks in-the-wild, spurring further exploration in the technical space.

Cross-Cutting Challenge #2: End-to-End Audit Security. The literature has largely waived away the threat of active attacks against auditing systems. Due to the real-world threat of anti-forensic attacks (e.g., [176, 177]), this position is untenable. The limited work on audit security that has appeared has focused on the capture layer, e.g., kernel hardening [97] and tamper-evident auditing [74, 124–126, 178, 179], but threats to the integrity of the audit pipeline are pervasive at all layers. For example, recent work by Carter et al. [180] demonstrates control flow manipulation of compromised applications at the capture layer, which affects the integrity of investigation layer routines such as behavior diagnosis and log unification. This example underscores the importance of broadening our understanding of audit security beyond simply assuring the integrity of at-rest log data. For example, we envision further research into the forensic validity of log reduction techniques, the security implications of intelligent causal graph query algorithms, and the feasibility of evasion attacks against causal graph detection systems. The correctness of our response to system intrusions depends on assuring the end-to-end custody of audit streams.

Cross-Cutting Challenge #3: Scaling Up Holistic Auditing. To date, the infrastructure layer has received less attention than its peers; in fact, many of the key papers we discussed in this layer were the result of a concerted effort by a single company, NEC Laboratories [62, 65, 68, 70, 71]. While understanding challenges at the infrastructure layer are difficult without industry insight, it is clear that innovative solutions to the distributed management of audit logs are required for causal analysis to become widely deployed. We are encouraged by existing work that has attempted to solve log management through cross-layer solutions, such as the KCAL [78] and CamQuery [69] systems that respectively embed reduction and query solutions into endpoint capture mechanisms. We envision future research that designs systems in anticipation of massive deployments of tens of thousands of machines.

Audit logs can quickly grow to unwieldy sizes with logs generation rates reported to be anywhere between 1 GB [169] to 33 GB [78] per day per machine. While most commercial systems utilize commodity auditing frameworks to collect audit logs for EDR systems, many large organizations are simply unwilling to pay for long-term retention of audit logs. Commercial products often store the logs in a ring buffer that is typically allocated to provide just a few months [130], or in some cases only days [129, 170], of storage. These retention periods are simply insufficient when considering the duration of APTs in high-profile data breaches (e.g., [17–23]). As a result, log storage and retention costs contribute to the difficulty of effectively detecting and responding to threats. Therefore, the reduction layer is essential to the proliferation of the various provenance techniques explored in our survey. However, our efficiency analysis paints a mixed picture of the current state-of-the art. We found that reduction techniques were not synergistic with one another, but enjoyed multiplicative improvements in log size reduction when deployed alongside traditional data compression. Most excitingly, the log reduction factor achieved with gzip provides an extraordinarily efficient means of audit log cold storage, enabling organizations to retain orders of magnitude more log data when it is not in active use. An important caveat to this result is that the resulting log will experience impaired

forensic utility – our anomaly detection experiments uncover a disturbing trend in which aggressive log reduction led to reduced intrusion detection performance. To further explore this issue, we recommend that future research on audit log reduction conduct security analyses not just through demonstrative anecdotal examples, but also through data-driven analysis of the impacts on security monitoring software. Further, we envision future research on forensically-optimized hybrid compression techniques, distributed and lossless deduplication of log data, anomaly-detection-friendly log reduction and compression approaches, and novel metrics for evaluating the security of log reduction techniques.

VI. Conclusion

Auditing is an indispensable element of the "Gold Standard" of operating system security [3], a fact that has only recently been reflected in the security literature. It is our hope that this work serves as a summary of the communities efforts to date and a launching point for further inquiry.

Acknowledgment

We thank our shepherd, Grant Ho, and the anonymous reviewers for their comments and suggestions. We also thank the members of the research community, in particular Shiqing Ma, Kangkook Jee, and Thomas Moyer for their feedback on the preprint version of our paper. Muhammad Adil Inam was partially supported by the Sohaib and Sara Abbasi Computer Science Fellowship. This work was supported in part by the NSF under contracts CNS-16-57534, CNS-17-50024 and CNS-20-55127. Any opinions, findings, conclusions, or recommendations expressed in this material are those of the authors and do not necessarily reflect the views of their employers or the sponsors.

References

[1] T. Jaeger, "Operating system security," *Synthesis Lectures on Information Security, Privacy, and Trust*, vol. 1, no. 1, 2008.

[2] J. P. Anderson, "Computer Security Technology Planning Study," Air Force Electronic Systems Division, Tech. Rep. ESD-TR-73-51, 1972.

[3] B. Lampson, "Perspectives on protection and security," in *SOSP History Day 2015*, ser. SOSP '15. Association for Computing Machinery, 2015.

[4] P. Loscocco and S. Smalley, "Integrating Flexible Support for Security Policies into the Linux Operating System," in *Proceedings of the 2001 USENIX Annual Technical Conference*. USENIX Association, 2001.

[5] S. Smalley, C. Vance, and W. Salamon, "Implementing SELinux as a Linux security module," *NAI Labs Report*, vol. 1, 2001.

[6] C. Wright, C. Cowan, S. Smalley, J. Morris, and G. Kroah-Hartman, "Linux security modules: General security support for the linux kernel," in *USENIX Security Symposium*, 2002.

[7] B. Hicks, S. Rueda, L. St.Clair, T. Jaeger, and P. McDaniel, "A Logical Specification and Analysis for SELinux MLS Policy," *ACM Trans. Inf. Syst. Secur.*, vol. 13, no. 3, 2010.

[8] T. Jaeger, R. Sailer, and X. Zhang, "Analyzing integrity protection in the selinux example policy," in *USENIX Security Symposium*, 2003.

[9] K. Sueyasu, T. Tabata, and K. Sakurai, "On the security of selinux with a simplified policy," in *Proceedings of the IASTED International Conference on Communication, Network, and Information Security*, M. Hamza, Ed., 2003.

[10] A. Edwards, T. Jaeger, and X. Zhang, "Runtime verification of authorization hook placement for the linux security modules framework," in *ACM CCS*, 2002.

[11] V. Ganapathy, T. Jaeger, and S. Jha, "Automatic placement of authorization hooks in the linux security modules framework," in *ACM CCS*, 2005.

[12] T. Jaeger, A. Edwards, and X. Zhang, "Consistency Analysis of Authorization Hook Placement in the Linux Security Modules Framework," *ACM Trans. Inf. Syst. Secur.*, vol. 7, no. 2, 2004.

[13] X. Zhang, A. Edwards, and T. Jaeger, "Using CQUAL for Static Analysis of Authorization Hook Placement," in *USENIX Security Symposium*, 2002.

[14] SUSE LINUXAG, "Linux Audit-Subsystem Design Documentation for Linux Kernel 2.6, v0.1," Available at http://uniforum.chi.il.us/slides/HardeningLinux/LAuS-Design.pdf, 2004.

[15] N. S. Agency, "Controlled Access Protection Profile, Version 1.d," https://www.niap-ccevs.org/Profile/Info.cfm?PPID=14&id=14, 1999.

[16] ——, "Labeled Security Protection Profile, Version 1.b," https://www.niap-ccevs.org/Profile/Info.cfm?PPID=17&id=17, 1999.

[17] "Equifax Says Cyberattack May Have Affected 143 Million in the U.S." https://www.nytimes.com/2017/09/07/business/equifax-cyberattack.html.

[18] G. Kurtz, "Operation Aurora Hit Google, Others," 2010, available at http://securityinnovator.com/index.php?articleID=42948&sectionID=25.

[19] "Inside the Cyberattack That Shocked the US Government," https://www.wired.com/2016/10/inside-cyberattack-shocked-us-government/.

[20] N. Perlroth and D. E. Sanger, "Cyberattacks Put Russian Fingers on the Switch at Power Plants, U.S. Says," https://www.nytimes.com/2018/03/15/us/politics/russia-cyberattacks.html, 2018.

[21] "APT3," https://attack.mitre.org/groups/G0022/, 2019.

[22] "APT29," https://attack.mitre.org/groups/G0016/, 2019.

[23] "Target Missed Warnings in Epic Hack of Credit Card Data," https://bloom.bg/2KjElxM.

[24] Crowdstrike, "Why Dwell Time Continues to Plague Organizations," https://www.crowdstrike.com/blog/why-dwell-time-continues-to-plague-organizations/, 2019.

[25] S. Morgan, "Global Cybersecurity Spending Predicted To Exceed $1 Trillion From 2017-2021," https://cybersecurityventures.com/cybersecurity-market-report/, 2019.

[26] "Endpoint Detection and Response Solutions Market," https://www.gartner.com/reviews/market/endpoint-detection-and-response-solutions, 2019.

[27] J. Goepfert, K. Massey, and M. Shirer, "Worldwide Spending on Security Solutions Forecast to Reach $103.1 Billion in 2019, According to a New IDC Spending Guide," https://www.businesswire.com/news/home/20190320005114/en/, 2019.

[28] Carbon Black, "Global incident response threat report," https://www.carbonblack.com/global-incident-response-threat-report/november-2018/, 2018, last accessed 04-20-2019.

[29] L. Moreau, B. Clifford, J. Freire, J. Futrelle, Y. Gil, P. Groth, N. Kwasnikowska, S. Miles, P. Missier, J. Myers *et al.*, "The Open Provenance Model Core Specification (v1. 1)," *Future Generation Computer Systems*, vol. 27, no. 6, 2011.

[30] M. N. Hossain, J. Wang, O. Weisse, R. Sekar, D. Genkin, B. He, S. D. Stoller, G. Fang, F. Piessens, E. Downing *et al.*, "Dependence-preserving data compaction for scalable forensic analysis," in *USENIX Security Symposium*, 2018.

[31] 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 Symposium on Security and Privacy (SP)*, 2019.

[32] S. M. Milajerdi, B. Eshete, R. Gjomemo, and V. Venkatakrishnan, "Poirot: Aligning attack behavior with kernel audit records for cyber threat hunting," in *CCS*, 2019.

[33] M. N. Hossain, S. Sheikhi, and R. Sekar, "Combating dependence explosion in forensic analysis using alternative tag propagation semantics," in *IEEE Symposium on Security and Privacy (SP)*, 2020.

[34] W. U. Hassan, S. Guo, D. Li, Z. Chen, K. Jee, Z. Li, and A. Bates, "Nodoze: Combatting threat alert fatigue with automated provenance triage," in *NDSS*, 2019.

[35] W. U. Hassan, A. Bates, and D. Marino, "Tactical provenance analysis for endpoint detection and response systems," in *IEEE Symposium on Security and Privacy (SP)*, 2020.

[36] 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 Symposium*, 2017.

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

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

[39] E. Manzoor, S. M. Milajerdi, and L. Akoglu, "Fast memory-efficient anomaly detection in streaming heterogeneous graphs," in *ACM SIGKDD International Conference on Knowledge Discovery and Data Mining*, 2016.

[40] X. Han, T. Pasquier, T. Ranjan, M. Goldstein, and M. Seltzer, "Frappuccino: Fault-detection through runtime analysis of provenance," in *HotCloud*, 2017.

[41] 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 *USENIX Security Symposium*, 2021.

[42] M. Du, F. Li, G. Zheng, and V. Srikumar, "Deeplog: Anomaly detection and diagnosis from system logs through deep learning," in *CCS*, 2017.

[43] T. van Ede, H. Aghakhani, N. Spahn, R. Bortolameotti, M. Cova, A. Continella, M. van Steen, A. Peter, C. Kruegel, and G. Vigna, "Deepcase: Semi-supervised contextual analysis of security events," in *IEEE Symposium on Security and Privacy (SP)*, 2022.

[44] 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.

[45] A. Valdes and K. Skinner, "Probabilistic alert correlation," in *International Workshop on Recent Advances in Intrusion Detection*, 2001.

[46] F. Valeur, G. Vigna, C. Kruegel, and R. A. Kemmerer, "Comprehensive approach to intrusion detection alert correlation," *IEEE Transactions on Dependable and Secure Computing*, vol. 1, no. 3, 2004.

[47] H. Debar and A. Wespi, "Aggregation and correlation of intrusion-detection alerts," in *International Workshop on Recent Advances in Intrusion Detection*. Springer, 2001.

[48] "Endpoint Monitoring & Security," https://logrhythm.com/solutions/security/endpoint-threat-detection/, 2019.

[49] "Logz.io," https://logz.io/.

[50] Splunk Inc., "splunk," https://www.splunk.com, Last accessed August 2018.

[51] Y. Zhai, P. Ning, and J. Xu, "Integrating ids alert correlation and os-level dependency tracking," in *International Conference on Intelligence and Security Informatics*. Springer, 2006.

[52] G. Gu, P. Porras, V. Yegneswaran, and M. Fong, "BotHunter: Detecting malware infection through ids-driven dialog correlation," in *USENIX Security Symposium*, 2007.

[53] 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.

[54] 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 Symposium*, 2021.

[55] G. D. L. T. Parra, L. Selvera, J. Khoury, H. Irizarry, E. Bou-Harb, and P. Rad, "Interpretable federated transformer log learning for cloud threat forensics," in *NDSS*, 2022.

[56] A. Tabiban, H. Zhao, Y. Jarraya, M. Pourzandi, M. Zhang, and L. Wang, "Provtalk: Towards interpretable multi-level provenance analysis in networking functions virtualization (nfv)," in *NDSS*, 2022.

[57] Z. Xu, P. Fang, C. Liu, X. Xiao, Y. Wen, and D. Meng, "Depcomm: Graph summarization on system audit logs for attack investigation," in *IEEE Symposium on Security and Privacy (SP)*, 2022.

[58] 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 Symposium on Security and Privacy (SP)*, 2022.

[59] S. M. Milajerdi, B. Eshete, R. Gjomemo, and V. N. Venkatakrishnan, "Propatrol: Attack investigation via extracted high-level tasks," in *International Conference on Information Systems Security*. Springer, 2018.

[60] C. Zhong, J. Yen, P. Liu, and R. F. Erbacher, "Automate cybersecurity data triage by leveraging human analysts' cognitive process," in *IEEE BigDataSecurity*, 2016.

[61] P. Fang, P. Gao, C. Liu, E. Ayday, K. Jee, T. Wang, Y. F. Ye, Z. Liu, and X. Xiao, "Back-propagating system dependency impact for attack investigation," in *USENIX Security Symposium*, 2022.

[62] W. U. Hassan, D. Li, K. Jee, X. Yu, K. Zou, D. Wang, Z. Chen, Z. Li, J. Rhee, J. Gui *et al.*, "This is why we can't cache nice things: Lightning-fast threat hunting using suspicion-based hierarchical storage," in *ACSAC*, 2020.

[63] W. U. Hassan, M. A. Noureddine, P. Datta, and A. Bates, "Omegalog: High-fidelity attack investigation via transparent multi-layer log analysis," in *NDSS*, 2020.

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

[65] P. Gao, F. Shao, X. Liu, X. Xiao, Z. Qin, F. Xu, P. Mittal, S. R. Kulkarni, and D. Song, "Enabling efficient cyber threat hunting with cyber threat intelligence," in *2021 IEEE 37th International Conference on Data Engineering (ICDE)*, 2021.

[66] R. Yang, S. Ma, H. Xu, X. Zhang, and Y. Chen, "Uiscope: Accurate, instrumentation-free, and visible attack investigation for gui applications," in *NDSS*, 2020.

[67] T. Pasquier, X. Han, M. Goldstein, T. Moyer, D. Eyers, M. Seltzer, and J. Bacon, "Practical whole-system provenance capture," in *Symposium on Cloud Computing*, 2017.

[68] 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.

[69] T. Pasquier, X. Han, T. Moyer, A. Bates, O. Hermant, D. Eyers, J. Bacon, and M. Seltzer, "Runtime analysis of whole-system provenance," in *CCS*, 2018.

[70] P. Gao, X. Xiao, Z. Li, F. Xu, S. R. Kulkarni, and P. Mittal, "Aiql: Enabling efficient attack investigation from system monitoring data," in *USENIX ATC*, 2018.

[71] 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 Symposium*, 2018.

[72] A. Gehani and D. Tariq, "Spade: Support for provenance auditing in distributed environments," in *ACM/IFIP/USENIX International Conference on Distributed Systems Platforms and Open Distributed Processing*. Springer, 2012.

[73] A. Gehani and M. Kim, "Mendel: Efficiently Verifying the Lineage of Data Modified in Multiple Trust Domains," in *hpdc10*, ser. HPDC'10, 2010.

[74] R. Paccagnella, P. Datta, W. U. Hassan, A. Bates, C. W. Fletcher, A. Miller, and D. Tian, "Custos: Practical Tamper-Evident Auditing of Operating Systems Using Trusted Execution," in *NDSS*, 2020.

[75] K. H. Lee, X. Zhang, and D. Xu, "Loggc: garbage collecting audit log," in *CCS*, 2013.

[76] Y. Tang, D. Li, Z. Li, M. Zhang, K. Jee, X. Xiao, Z. Wu, J. Rhee, F. Xu, and Q. Li, "Nodemerge: template based efficient data reduction for big-data causality analysis," in *CCS*, 2018.

[77] A. Bates, D. Tian, G. Hernandez, T. Moyer, K. R. Butler, and T. Jaeger, "Taming the Costs of Trustworthy Provenance through Policy Reduction," *ACM Trans. on Internet Technology*, vol. 17, no. 4, 2017.

[78] S. Ma, J. Zhai, Y. Kwon, K. H. Lee, X. Zhang, G. Ciocarlie, A. Gehani, V. Yegneswaran, D. Xu, and S. Jha, "Kernel-supported cost-effective audit logging for causality tracking," in *USENIX ATC*, 2018.

[79] Z. Xu, Z. Wu, Z. Li, K. Jee, J. Rhee, X. Xiao, F. Xu, H. Wang, and G. Jiang, "High fidelity data reduction for big data security dependency analyses," in *CCS*, 2016.

[80] R. Ahmad, M. Bru, and A. Gehani, "Streaming provenance compression," in *Provenance and Annotation of Data and Processes*, K. Belhajjame, A. Gehani, and P. Alper, Eds. Springer International Publishing, 2018.

[81] Y. Xie, D. Feng, Z. Tan, L. Chen, K.-K. Muniswamy-Reddy, Y. Li, and D. D. Long, "A Hybrid Approach for Efficient Provenance Storage," in *Proceedings of the 21st ACM International Conference on Information and Knowledge Management*, ser. CIKM '12, 2012.

[82] Y. Xie, K.-K. Muniswamy-Reddy, D. Feng, Y. Li, and D. D. E. Long, "Evaluation of a Hybrid Approach for Efficient Provenance Storage," *Trans. Storage*, vol. 9, no. 4, 2013.

[83] Y. Xie, K.-K. Muniswamy-Reddy, D. D. E. Long, A. Amer, D. Feng, and Z. Tan, "Compressing Provenance Graphs," in *tapp11*, ser. TAPP'11, 2011.

[84] P. Fei, Z. Li, Z. Wang, X. Yu, D. Li, and K. Jee, "Seal: Storage-efficient causality analysis on enterprise logs with query-friendly compression," in *USENIX Security Symposium*, 2021.

[85] W. U. Hassan, L. Aguse, N. Aguse, A. Bates, and T. Moyer, "Towards

scalable cluster auditing through grammatical inference over provenance graphs," in *NDSS*, 2018.
[86] N. Michael, J. Mink, J. Liu, S. Gaur, W. U. Hassan, and A. Bates, "On the forensic validity of approximated audit logs," in *ACSAC*, 2020.
[87] S. T. King and P. M. Chen, "Backtracking intrusions," in *Proceedings of the Nineteenth ACM Symposium on Operating Systems Principles*, ser. SOSP '03. ACM, 2003.
[88] Y. Ji, S. Lee, E. Downing, W. Wang, M. Fazzini, T. Kim, A. Orso, and W. Lee, "Rain: Refinable attack investigation with on-demand inter-process information flow tracking," in *CCS*, 2017.
[89] Y. Ji, S. Lee, and W. Lee, "Recprov: Towards provenance-aware user space record and replay," in *Provenance and Annotation of Data and Processes*, M. Mattoso and B. Glavic, Eds. Springer International Publishing, 2016.
[90] M. Stamatogiannakis, P. Groth, and H. Bos, "Decoupling provenance capture and analysis from execution," in *USENIX TaPP*, 2015.
[91] "System administration utilities," man7.org/linux/man-pages/man8/auditd.8.html/.
[92] K. Bridge, K. Sharkey, D. Coulter, M. Jacobs, and M. Satran, "About Event Tracing," https://docs.microsoft.com/en-us/windows/win32/etw/about-event-tracing, 2018.
[93] K.-K. Muniswamy-Reddy, D. A. Holland, U. Braun, and M. Seltzer, "Provenance-aware Storage Systems," in *Proceedings of the Annual Conference on USENIX '06 Annual Technical Conference*, ser. atc06, 2006.
[94] K.-K. Muniswamy-Reddy, U. Braun, D. A. Holland, P. Macko, D. Maclean, D. Margo, M. Seltzer, and R. Smogor, "Layering in Provenance Systems," in *atc09*, ser. ATC'09, 2009.
[95] P. McDaniel, K. Butler, S. McLaughlin, R. Sion, E. Zadok, and M. Winslett, "Towards a Secure and Efficient System for End-to-End Provenance," in *USENIX TaPP*, 2010.
[96] D. Pohly, S. McLaughlin, P. McDaniel, and K. Butler, "Hi-Fi: Collecting High-Fidelity Whole-System Provenance," in *Proceedings of the 2012 Annual Computer Security Applications Conference*, ser. ACSAC '12, 2012.
[97] A. Bates, D. Tian, K. R. Butler, and T. Moyer, "Trustworthy Whole-System Provenance for the Linux Kernel," in *USENIX Security Symposium*, 2015.
[98] K. H. Lee, X. Zhang, and D. Xu, "High accuracy attack provenance via binary-based execution partition." in *NDSS*, 2013.
[99] 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 Symposium*, 2017.
[100] Y. Kwon, F. Wang, W. Wang, K. H. Lee, W.-C. Lee, S. Ma, X. Zhang, D. Xu, S. Jha, G. F. Ciocarlie *et al.*, "Mci: Modeling-based causality inference in audit logging for attack investigation." in *NDSS*, 2018.
[101] B. Li, P. Vadrevu, K. H. Lee, and R. Perdisci, "JSgraph: Enabling Reconstruction of Web Attacks via Efficient Tracking of Live In-Browser JavaScript Executions," in *NDSS*, 2018.
[102] S. Ma, K. H. Lee, C. H. Kim, J. Rhee, X. Zhang, and D. Xu, "Accurate, low cost and instrumentation-free security audit logging for windows," in *Proceedings of the 31st Annual Computer Security Applications Conference*, ser. ACSAC 2015. ACM, 2015.
[103] M. A. Inam, W. U. Hassan, A. Ahad, A. Bates, R. Tahir, T. Xu, and F. Zaffar, "Forensic analysis of configuration-based attacks," in *NDSS*, 2022.
[104] M. Backes, S. Bugiel, and S. Gerling, "Scippa: System-centric ipc provenance on android," in *Proceedings of the 30th Annual Computer Security Applications Conference*, ser. ACSAC '14. ACM, 2014.
[105] M. Dietz, S. Shekhar, Y. Pisetsky, A. Shu, and D. S. Wallach, "Quire: Lightweight provenance for smart phone operating systems," in *Proceedings of USENIX Security '11*. USENIX Association, 2011.
[106] C. Yang, G. Yang, A. Gehani, V. Yegneswaran, D. Tariq, and G. Gu, *Using Provenance Patterns to Vet Sensitive Behaviors in Android Apps*. Springer International Publishing, 2015.
[107] X. Yuan, O. Setayeshfar, H. Yan, P. Panage, X. Wei, and K. H. Lee, "Droidforensics: Accurate reconstruction of android attacks via multi-layer forensic logging," in *ASIA CCS*. ACM, 2017.
[108] Q. Wang, W. U. Hassan, A. Bates, and C. Gunter, "Fear and logging in the internet of things," in *NDSS*, 2018.
[109] "Event Tracing," https://docs.microsoft.com/en-us/windows/desktop/ETW/event-tracing-portal.
[110] "DTrace on FreeBSD," https://wiki.freebsd.org/DTrace.
[111] National Institute of Standards and Technology, "NIST special publication 800-53 (rev. 4), security controls and assessment procedures for federal information systems and organizations," 2013.
[112] S. L. Garfinkel, "Automating disk forensic processing with sleuthkit, xml and python," in *2009 Fourth International IEEE Workshop on Systematic Approaches to Digital Forensic Engineering*. IEEE, 2009, pp. 73–84.
[113] M. H. Ligh, A. Case, J. Levy, and A. Walters, *The art of memory forensics: detecting malware and threats in windows, linux, and Mac memory*. John Wiley & Sons, 2014.
[114] C. Willems, T. Holz, and F. Freiling, "Toward automated dynamic malware analysis using cwsandbox," in *IEEE Symposium on Security and Privacy (SP)*, 2007.
[115] X. Chen, H. Irshad, Y. Chen, A. Gehani, and V. Yegneswaran, "Clarion: Sound and clear provenance tracking for microservice deployments," in *USENIX Security Symposium*, 2021.
[116] P. Datta, I. Polinsky, M. A. Inam, A. Bates, and W. Enck, "Alastor: Reconstructing the provenance of serverless intrusions," in *USENIX Security Symposium*, 2021.
[117] A. Case and G. G. Richard III, "Memory forensics: The path forward," *Digital Investigation*, vol. 20, pp. 23–33, 2017.
[118] S. Ma, X. Zhang, and D. Xu, "ProTracer: Towards Practical Provenance Tracing by Alternating Between Logging and Tainting," in *NDSS*, 2016.
[119] D. Tariq, M. Ali, and A. Gehani, "Towards Automated Collection of Application-Level Data Provenance," in *USENIX TaPP*, 2012.
[120] R. Hasan, R. Sion, and M. Winslett, "SPROV 2.0: A Highly-Confgurable Platform-Independent Library for Secure Provenance," in *ACM CCS*, 2009.
[121] P. Macko and M. Seltzer, "A General-purpose Provenance Library," in *tapp12*, ser. TaPP'12, 2012.
[122] Y. Kwon, D. Kim, W. N. Sumner, K. Kim, B. Saltaformaggio, X. Zhang, and D. Xu, "LDX: Causality inference by lightweight dual execution," in *ASPLOS*. ACM, 2016.
[123] N. Husted, S. Qureshi, D. Tariq, and A. Gehani, "Android provenance: Diagnosing device disorders," in *USENIX TaPP*, 2013.
[124] R. Paccagnella, K. Liao, D. Tian, and A. Bates, "Logging to the danger zone: Race condition attacks and defenses on system audit frameworks," in *CCS*, 2020.
[125] V. Karande, E. Bauman, Z. Lin, and L. Khan, "SGX-Log: Securing System Logs With SGX," in *Proceedings of the 2017 ACM on Asia Conference on Computer and Communications Security*, ser. ASIA CCS '17, 2017.
[126] A. Ahmad, S. Lee, and M. Peinado, "Hardlog: Practical tamper-proof system auditing using a novel audit device," in *2022 IEEE Symposium on Security and Privacy (SP)*. IEEE Computer Society, 2022, pp. 1554–1554.
[127] C. Yagemann, M. Noureddine, W. U. Hassan, S. Chung, A. Bates, and W. Lee, "Validating the integrity of audit logs against execution repartitioning attacks," in *Proceedings of the 2021 ACM SIGSAC Conference on Computer and Communications Security*, ser. CCS '21, 2021.
[128] A. Bates, K. R. B. Butler, and T. Moyer, "Take Only What You Need: Leveraging Mandatory Access Control Policy to Reduce Provenance Storage Costs," in *tapp15*, ser. TaPP'15, 2015.
[129] "About purging reports," https://support.symantec.com/us/en/article.howto129116.html, 2019.
[130] "Evaluating Endpoint Products," https://redcanary.com/blog/evaluating-endpoint-products-in-a-crowded-confusing-market/, 2018.
[131] H. Ding, S. Yan, J. Zhai, and S. Ma, "Elise: A storage efficient logging system powered by redundancy reduction and representation learning," in *30th USENIX Security Symposium (USENIX Security 21)*, 2021, pp. 3023–3040.
[132] J. Park, D. Nguyen, and R. Sandhu, "A Provenance-Based Access Control Model," in *Proceedings of the 10th Annual International Conference on Privacy, Security and Trust (PST)*, 2012.
[133] L. Sun, J. Park, and R. Sandhu, "Towards provenance-based access control with feasible overhead," in *2014 International Conference on Information Science, Electronics and Electrical Engineering*, vol. 2, 2014.
[134] D. Nguyen, J. Park, and R. Sandhu, "A provenance-based access control model for dynamic separation of duties," in *2013 Eleventh Annual Conference on Privacy, Security and Trust*, 2013.
[135] ——, *Adopting Provenance-Based Access Control in OpenStack Cloud IaaS*. Springer International Publishing, 2014.
[136] A. Bates, B. Mood, M. Valafar, and K. Butler, "Towards secure provenance-based access control in cloud environments," in *ACM*

CODASPY, 2013.
[137] "Cypher Query Language," https://neo4j.com/developer/cypher/, 2021.
[138] D. A. Holland, U. Bruan, D. Maclean, K.-K. Muniswamy-Reddy, and M. I. Seltzer, "Choosing a Data Model and Query Language for Provenance," in *ipaw08*, ser. IPAW'08, 2008.
[139] "Neo4j," https://neo4j.com, 2021.
[140] H. Lim, D. Han, D. G. Andersen, and M. Kaminsky, "Mica: A holistic approach to fast in-memory key-value storage," in *NSDI*, 2014.
[141] R. Nishtala, H. Fugal, S. Grimm, M. Kwiatkowski, H. Lee, H. C. Li, R. McElroy, M. Paleczny, D. Peek, P. Saab *et al.*, "Scaling memcache at facebook," in *10th USENIX Symposium on Networked Systems Design and Implementation (NSDI 13)*, 2013.
[142] "Redis," https://redis.io/, 2021.
[143] A. Gehani and U. Lindqvist, "Bonsai: Balanced Lineage Authentication," in *ACSAC*, 2007.
[144] T. Mandl, U. Bayer, and F. Nentwich, "Anubis analyzing unknown binaries the automatic way," in *Virus bulletin conference*, vol. 1, 2009, p. 02.
[145] T. K. Lengyel, S. Maresca, B. D. Payne, G. D. Webster, S. Vogl, and A. Kiayias, "Scalability, fidelity and stealth in the drakvuf dynamic malware analysis system," in *Proceedings of the 30th annual computer security applications conference*, 2014, pp. 386–395.
[146] "Cuckoo Sandbox tool," https://cuckoosandbox.org/.
[147] C. Willems, R. Hund, and T. Holz, "Cxpinspector: Hypervisor-based, hardware-assisted system monitoring," *Ruhr-Universitat Bochum, Tech. Rep*, p. 12, 2013.
[148] M. G. Schultz, E. Eskin, F. Zadok, and S. J. Stolfo, "Data mining methods for detection of new malicious executables," in *IEEE Symposium on Security and Privacy (SP)*, 2001.
[149] M. Zheng, M. Sun, and J. C. Lui, "Droid analytics: a signature based analytic system to collect, extract, analyze and associate android malware," in *2013 12th IEEE International Conference on Trust, Security and Privacy in Computing and Communications*. IEEE, 2013, pp. 163–171.
[150] C. Kolbitsch, P. M. Comparetti, C. Kruegel, E. Kirda, X.-y. Zhou, and X. Wang, "Effective and efficient malware detection at the end host." in *USENIX Security Symposium*, vol. 4, no. 1, 2009, pp. 351–366.
[151] A. Lanzi, D. Balzarotti, C. Kruegel, M. Christodorescu, and E. Kirda, "Accessminer: using system-centric models for malware protection," in *Proceedings of the 17th ACM conference on Computer and communications security*, 2010, pp. 399–412.
[152] L. Liu, S. Chen, G. Yan, and Z. Zhang, "Bottracer: Execution-based bot-like malware detection," in *International Conference on Information Security*. Springer, 2008, pp. 97–113.
[153] A. Moser, C. Kruegel, and E. Kirda, "Exploring multiple execution paths for malware analysis," in *IEEE Symposium on Security and Privacy (SP)*, 2007.
[154] S. Das, Y. Liu, W. Zhang, and M. Chandramohan, "Semantics-based online malware detection: Towards efficient real-time protection against malware," *IEEE transactions on information forensics and security*, vol. 11, no. 2, pp. 289–302, 2015.
[155] "MITRE ATT&CK," https://attack.mitre.org, 2019.
[156] D. Yang, B. Li, L. Rettig, and P. Cudré-Mauroux, "Histosketch: Fast similarity-preserving sketching of streaming histograms with concept drift," in *IEEE ICDM*, 2017.
[157] Y. Xie, D. Feng, Z. Tan, and J. Zhou, "Unifying intrusion detection and forensic analysis via provenance awareness," *Future Generation Computer Systems*, vol. 61, pp. 26–36, 2016.
[158] Y. Xie, D. Feng, Y. Hu, Y. Li, S. Sample, and D. Long, "Pagoda: A hybrid approach to enable efficient real-time provenance based intrusion detection in big data environments," *IEEE Transactions on Dependable and Secure Computing*, vol. 17, no. 6, pp. 1283–1296, 2018.
[159] 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.
[160] FireEye, Inc., "How Many Alerts is Too Many to Handle?" https://www2.fireeye.com/StopTheNoise-IDC-Numbers-Game-Special-Report.html, 2019.
[161] D. Gunning and D. Aha, "Darpa's explainable artificial intelligence (xai) program," *AI Magazine*, vol. 40, no. 2, 2019.
[162] B. Saltaformaggio, R. Bhatia, Z. Gu, X. Zhang, and D. Xu, "Guitar: Piecing together android app guis from memory images," in *CCS*, 2015.
[163] F. Pagani and D. Balzarotti, "Back to the whiteboard: a principled approach for the assessment and design of memory forensic techniques," in *USENIX Security Symposium*, 2019.
[164] M. Alazab, S. Venkatraman, and P. Watters, "Effective digital forensic analysis of the ntfs disk image," *Ubiquitous Computing and Communication Journal*, vol. 4, no. 1, pp. 551–558, 2009.
[165] S. L. Garfinkel, "Automating disk forensic processing with sleuthkit, xml and python," in *2009 Fourth International IEEE Workshop on Systematic Approaches to Digital Forensic Engineering*. IEEE, 2009, pp. 73–84.
[166] F. Adelstein, "Live forensics: diagnosing your system without killing it first," *Communications of the ACM*, vol. 49, no. 2, pp. 63–66, 2006.
[167] E. Casey and G. J. Stellatos, "The impact of full disk encryption on digital forensics," *ACM SIGOPS Operating Systems Review*, vol. 42, no. 3, pp. 93–98, 2008.
[168] C. C.-C. Cheng, C. Shi, N. Z. Gong, and Y. Guan, "Logextractor: Extracting digital evidence from android log messages via string and taint analysis," *Forensic Science International: Digital Investigation*, vol. 37, p. 301193, 2021.
[169] K. H. Lee, X. Zhang, and D. Xu, "LogGC: Garbage Collecting Audit Log," in *Proceedings of the 2013 ACM SIGSAC conference on Computer and Communications Security*, ser. CCS '13. ACM, 2013.
[170] Symantec EDR 4.6 Docs, "How Symantec EDR purges data from the Symantec EDR database," https://techdocs.broadcom.com/us/en/symantec-security-software/endpoint-security-and-management/endpoint-detection-and-response/4-6/Settings/how-purges-data-from-the-database-v106460598-d38e46998.html, 2022.
[171] "Darpa transparent computing. 2020. transparent computing engagement 3 data release," 2020.
[172] J. Leskovec, "Stanford network analysis package," *Online, http://snap. stanford. edu*, 2009.
[173] "Crowd Strike," https://www.crowdstrike.com/blog/tech-center/hunt-crowdstrike-falcon/.
[174] "Sophos," https://support.sophos.com/support/s/article/KB-000036359?language=en_US.
[175] "Comodo," https://help.comodo.com/topic-444-1-905-11917-.html.
[176] C. Cimpanu, "Hackers are increasingly destroying logs to hide attacks," https://www.zdnet.com/article/hackers-are-increasingly-destroying-logs-to-hide-attacks/, last accessed 04-20-2019.
[177] S. Hales, "Last door log wiper," https://packetstormsecurity.com/files/118922/LastDoor.tar, last accessed 04-20-2019.
[178] A. A. Yavuz and P. Ning, "Baf: An efficient publicly verifiable secure audit logging scheme for distributed systems," in *2009 Annual Computer Security Applications Conference*. IEEE, 2009.
[179] S. A. Crosby and D. S. Wallach, "Efficient data structures for tamper-evident logging." in *USENIX Security Symposium*, 2009.
[180] C. Yagemann, M. Noureddine, W. U. Hassan, S. Chung, A. Bates, and W. Lee, "Validating the integrity of audit logs against execution repartitioning attacks," in *CCS*, 2021.