On the Forensic Validity of Approximated Audit Logs

Noor Michael
University of Illinois at Urbana-Champaign
nsm2@illinois.edu

Jaron Mink
University of Illinois at Urbana-Champaign
jaronmm2@illinois.edu

Jason Liu
University of Illinois at Urbana-Champaign
jdliu2@illinois.edu

Sneha Gaur
University of Illinois at Urbana-Champaign
sg2@illinois.edu

Wajih Ul Hassan
University of Illinois at Urbana-Champaign
whassan3@illinois.edu

Adam Bates
University of Illinois at Urbana-Champaign
batesa@illinois.edu

ABSTRACT

Auditing is an increasingly essential tool for the defense of computing systems, but the unwieldy nature of log data imposes significant burdens on administrators and analysts. To address this issue, a variety of techniques have been proposed for *approximating* the contents of raw audit logs, facilitating efficient storage and analysis. However, the security value of these approximated logs is difficult to measure—relative to the original log, it is unclear if these techniques retain the forensic evidence needed to effectively investigate threats. Unfortunately, prior work has only investigated this issue anecdotally, demonstrating sufficient evidence is retained for specific attack scenarios.

In this work, we address this gap in the literature through formalizing metrics for quantifying the *forensic validity* of an approximated audit log under differing threat models. In addition to providing quantifiable security arguments for prior work, we also identify a novel point in the approximation design space—that log events describing typical (benign) system activity can be aggressively approximated, while events that encode anomalous behavior should be preserved with lossless fidelity. We instantiate this notion of *Attack-Preserving* forensic validity in LogApprox, a new approximation technique that eliminates the redundancy of voluminous file I/O associated with benign process activities. We evaluate LogApprox alongside a corpus of exemplar approximation techniques from prior work and demonstrate that LogApprox achieves comparable log reduction rates while retaining 100% of attack-identifying log events. Additionally, we utilize this evaluation to illuminate the inherent trade-off between performance and utility within existing approximation techniques. This work thus establishes trustworthy foundations for the design of the next generation of efficient auditing frameworks.

CCS CONCEPTS

• **Security and privacy → Intrusion detection systems**.

ACSAC 2020, December 7–11, 2020, Austin, USA

ACM ISBN 978-1-4503-8858-0/20/12…$15.00
https://doi.org/10.1145/3427228.3427272

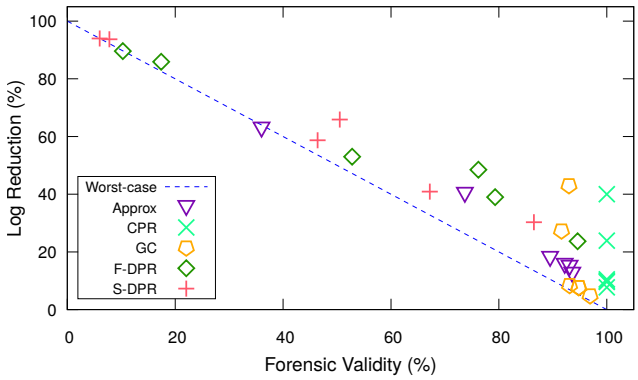


Figure 1: We present a novel analysis of the performance-utility tradeoff for different log approximation techniques according to the notion of *Causality-Preserving Forensics*, which was introduced in [77] and formally defined as a measurable property in this work. Each point on the graph corresponds to a different attack scenario described in Section 6.

KEYWORDS

Auditing, Data Provenance, Digital Forensics

ACM Reference Format:
Noor Michael, Jaron Mink, Jason Liu, Sneha Gaur, Wajih Ul Hassan, and Adam Bates. 2020. On the Forensic Validity of Approximated Audit Logs. In *Annual Computer Security Applications Conference (ACSAC 2020), December 7–11, 2020, Austin, USA.* ACM, New York, NY, USA, 14 pages. https://doi.org/10.1145/3427228.3427272

1 INTRODUCTION

Auditing is vital to the defense of computing systems. With alarming regularity [9, 39, 57, 66, 72], sophisticated threat actors are able to breach perimeter defenses and subsequently wreak havoc on organizations. Given our struggles keeping intruders *out* of systems, the onus shifts to quickly detecting and responding to threats in order to minimize their impact. Audit logs have proven invaluable to these tasks; today, 75% of cyber analysts report that logs are the most important resource when investigating threats [10]. The importance of audit logs will only increase as state-of-the-art *causal analysis* techniques for detection [22, 26, 52, 54, 74], alert triage [29, 30], and investigation [32, 34, 42, 61] become widely available.

Unfortunately, the capabilities provided by system auditing come at a cost. Commodity audit frameworks are known to generate tremendous volumes of log data, upwards of a terabyte per machine in a single month [22, 45, 49]. Not only is storing and managing this data a burden, but the unwieldy nature of these audit logs also slows down time-sensitive investigation tasks during in-progress attacks. For example, Liu et al. observed that even a simple backtrace query to determine the root cause of an event may take days to return [45]. At present, the inefficiencies of system auditing seriously undermine its use for effectively combatting real-world threats.

In response to this issue, researchers have called for optimizing the contents of audit logs, based largely on the observation that most events described by the log are not strictly necessary when investigating threats. A variety of methods have been proposed for achieving this goal, ranging from the filtering of events that describe deleted system entities [43], do not connote a new information flow [77] or do not effect the conclusions reached by standard forensic queries [36], among many others [1, 4, 6, 8, 14, 24, 28, 48, 49, 51, 71, 75, 76]. We refer to these methods as *approximation* techniques in this work. With many of these techniques reporting orders of magnitude reduction in log size, the goal of holistic auditing of large organizational networks seems within reach.

While these results are encouraging for the perspective of performance, it is much more difficult to quantify the *loss in utility* that arises from approximating the original log. Is an approximated log equally useful when investigating threats? If not, under what circumstances can we expect the approximation routine to introduce error? For example, Lee et al.'s pioneering LogGC system deletes "dead end" events describing entities that no longer exist on the system [43], but given the ephemeral nature of network sockets it is likely that LogGC may destroy log events describing data exfiltration tactics. Unfortunately, prior work has offered only anecdotal arguments for the utility of approximated logs.

In this work, we conduct the first independent analysis of the utility of audit logs that have been subjected to approximation techniques. We argue that utility is a measure of the "forensic validity" of a log for investigating different kinds of threats. To enable measurement of forensic validity, we formalize three metrics for characterizing approximated logs: *Lossless*, *Causality-Preserving*, and *Attack-Preserving*. While these metrics are distilled from prior work, we make the insight that each metric encodes a set of relationships that should be preserved under a specific threat model; we enumerate what these threat models are and which metric is most appropriate.

Using these metrics, we conduct an independent utility analysis of a set of exemplar approximation techniques. We discover that prior work often filters attack-related events when approximating logs, compressing logs by as much as 93.7% but *retaining as little as 7.3% of attacker-related forensic evidence* in the process. Whether this sacrifice in utility is acceptable depends in practice on an organization's resources and threat model. To aid system defenders in making these decisions, we also use the notion of forensic validity to reason about the trade-offs between the storage performance of audit logs and their security utility. Figure 1 shows a portion of our findings under the *Causality-Preserving Forensics* metric, which measures the percent of events signifying information flow that are retained in the approximated log. Each approximation technique analyzed has 6 points on the graph, each corresponding to a different attack scenario. The blue dashed line denotes the worst-case baseline in which all dropped events are forensically relevant, with points above the line indicating the technique outperforms the baseline. We can see that Xu et al.'s CPR system [77] (which this metric is based on) retains 100% of relevant log entries, but at the cost of storage efficiency. Conversely, we see that other techniques achieve storage efficiency almost proportionally to the percent of causality-preserving events filtered. We report on performance-utility tradeoffs for our other metrics in Section 6.4.

While optimizing logs for *Attack-Preserving Forensics* is preferable for the efficiency of audit logs, it requires a method of delineating typical process activity from unexpected system events. To address this gap, we present LogApprox[1], a regular expression (regex) learning approach to approximation. LogApprox targets the most space-intensive events found in logs, namely the file I/O activity which can account for up to 90% of log contents.[2] Once a regex for a given process has been learned, LogApprox matches and eliminates new events that match the regex. Through the design of a carefully-constructed learning algorithm, we demonstrate that it is possible to generate a set of regexes that faithfully describe typical process activity while simultaneously avoiding the filtering of any attack-specific behaviors. We show that LogApprox retains 100% utility under the attack-preserving model while exhibiting comparable performance to state-of-the-art techniques.

The contributions of this work are as follows:

- *Forensic Validity Measurement.* We formalize a set of metrics that can be used to quantify the utility of logs under different threat models. We conduct an independent analysis of an exemplar set of approximation techniques, discovering in the process that removal of attack-relevant events is common.
- *Attack-Preserving Approximation Techniques.* We present LogApprox, an approximation method that is optimized for attack-preserving forensics. LogApprox performs a bounded regex-learning routine over process executions to learn their file I/O behaviors. While aggressively filtering events related to known file behaviors, LogApprox retains a complete record of process-to-process and process-to-network dependencies, thus facilitating the causal analysis techniques.
- *Evaluation and Attack Engagements.* We evaluate the performance of LogApprox and subject it to a series of attack scenarios through which we measure forensic validity. These engagements confirm that LogApprox satisfies attack-preserving forensics while offering comparable reduction rates to prior work. Our code and datasets are available on request.

2 BACKGROUND

Logging is critical to defending systems, facilitating intrusion detection and post-mortem forensics of attacks. Audit logs can be generated at different software layers; for example, Windows Event Logs is an application-level framework for Active Directory environments [12], while Event Tracing for Windows [11], Linux Audit [65], and DTrace [21] are kernel-level frameworks that primarily

[1] Approx is an acronym for Attack-Preserving Provenance Reduction Over regeXes
[2] We observe in our evaluation datasets that 88.97% of events describe system calls associated with file I/O.

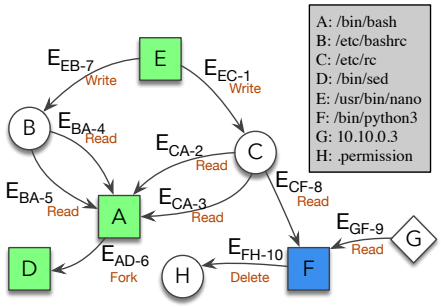


Figure 2: Example provenance graph. In this provenance graph boxes, diamonds and ovals represent processes, sockets and files, respectively. Green boxes represent live processes while blue boxes represent terminated processes.

log system calls. While all logs can be of potential use during threat investigations, low-level (kernel-level) audit frameworks are especially useful in threat hunting because they can be used to reliably trace dependencies between applications running on the hosts.

An audit log is a sequence of temporally-ordered event tuples (i.e., $< subject, object, access, timestamp >$), which can be further parsed into a causal dependency (i.e., *provenance*) graph by incrementally linking entities associated with each event tuple according to the access type (see, e.g., [5, 25, 38, 56]). Specifically, in a provenance graph $G = (V, E)$ each vertex $v \in V$ corresponds to a system object such as processes, files, and sockets, v while each $e \in E$ encodes a dependence relationship between those objects and roughly corresponds to a single log event. These directed edges point backward in time, denoting a provenance (historical) relation between events. An example provenance graph is shown in Figure 2. Here, a process named `/bin/bash` represented by vertex A reads a file named `/etc/rc` represented by vertex C. The edge between vertices A and C is represented by $E_{CA-2}read$ where 2 is a timestamp while *read* denotes a read event type from C to A vertices. Multiple edges between two vertices denote the same events occurring at different times, e.g., edges E_{CA-2} and E_{CA-3} denote two *read* events. Finally, at the time of analysis, vertices vertices can be either in a live or terminated state. In Figure 2, vertices A, D, and E are live processes while vertex F is a terminated process.

Provenance graphs allow investigators to perform forensic analysis. Investigators can perform tracing queries on the provenance graph to figure out the root-cause and ramifications of a cyber attack. We formally define these tracing queries as follows:

Definition 2.1. Backward Trace: A backward trace of edge e is the subgraph of G reachable from e (or equivalently, the destination vertex of e).

Definition 2.2. Forward Trace: A forward trace of edge e is the subgraph of G reachable from e in the reverse graph of G (or equivalently, the source vertex of e).

A backward trace enables an analyst to identify the *root cause(s)* of a particular event, e.g., the point of entry of an intruder into the system. In Figure 2, the backward trace of the event (edge) E_{AD-6} with event type *Fork* will traverse edges E_{BA-4}, E_{BA-5}, E_{CA-2}, E_{CA-3}, and E_{EC-1}, identifying process vertex E (`/usr/bin/nano`) as the root-cause. Note that even though E_{EB-7} is reachable from E_{AD-6} in the graph, it is not included in analysis because event E_{EB-7} happened after event E_{AD-6}. Conversely, a forward trace identifies the impact of a particular event. A forward trace from the root cause summarizes all actions taken by the attacker. For example, in Figure 2, to find the impact of event $EC-1$, a forward trace will return a subgraph consisting of E_{CA-2}, E_{CA-3}, E_{AD-6}, E_{CF-8}, and E_{FH-10}.

2.1 Audit Log Approximation

Although audit frameworks are extremely helpful to threat investigation, their use is impeded by the sheer volume of logs generated. Log overheads vary depending on the load of the machine, but have been reported to be anywhere between 3 GB [43] and 33 GB [49] per day for web servers and around 1 GB per day for workstations [43]. As a result, storing and analyzing these logs is often infeasible in practice — log data is often purged just a few days after its creation [70], and when it is retained for longer periods, simple trace queries may take days to return [45].

To combat the limitations of pervasive system auditing, many techniques for *log approximation* have been proposed. These approximation methods analyze the structure and semantics of the provenance graph to identify components (i.e., log events) that are unlikely to be of use to an analyst and can therefore be removed. While a variety of approaches to log approximation have been proposed based on filtering policies [4, 6], taint analysis [8, 51], templatization [24, 28, 71], or simple compression [75, 76], we describe at length three influential exemplar approaches below:

Garbage Collection (GC). First proposed by Lee et al. [43], garbage collection is based on the observation that subgraphs that exclusively describe *dead* system entities do not affect the present state of the system and can therefore be removed. Consider a process that generates a temporary file, then deletes it. If no other process accesses that temporary file, all log events associated with that file can be removed from the graph. A visualization of garbage collection is given in Figure 3 (a). Garbage collection has since been incorporated into other log analysis systems, e.g., [48, 49, 51].

Causality-Preserving Reduction (CPR). Introduced by Xu et al. [77], CPR observes that many log events are redundant because they do not denote a new information flow. Consider a process that writes to a file twice. If the process did not read from any other object between the two writes, we can remove the log event describing the second write because the process state did not change between writes. Note that, in actuality, the data buffer may have been completely different in each write event; however, because log events do not include data buffers, the graph must conservatively assume that *all* process state is transferred during each information flow event, thus the second write is completely redundant. A visualization of CPR is given in Figure 3 (b). CPR has been adopted or extended by subsequent log analysis systems, including [30, 36, 45, 71].

Dependence-Preserving Reduction (DPR) Building on the CPR concept, Hossain et al. propose that preserving causality is unnecessary so long as the provenance graph returns the correct system entities to forward and backward trace queries [36]. Consider a vertex for which there are two causal paths back to its root cause; this represents a potential redundancy and one of the two edges

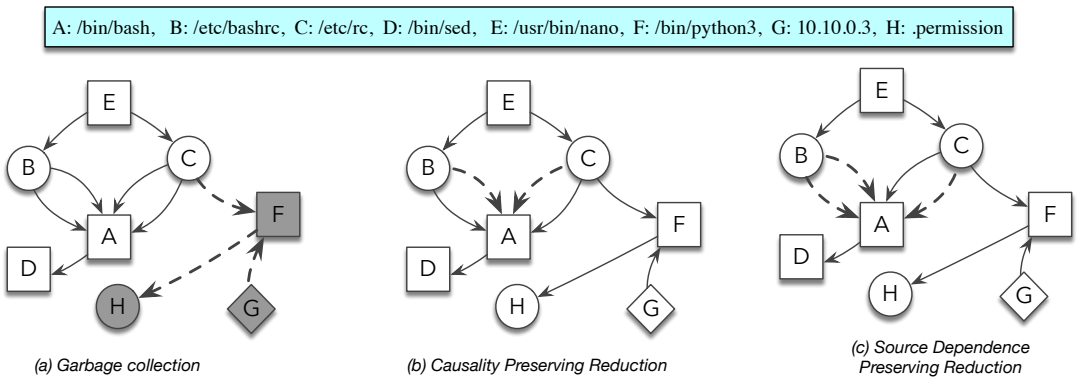


Figure 3: Visualizations of exemplar log approximation techniques. We used the provenance graph example from Figure 2 to apply each exemplar log approximation technique. Graph edges marked in dotted lines and vertices which are grayed out represent events that are filtered by each technique. (a) Provenance graph after applying LogGC technique [43]. (b) Provenance graph after applying CPR technique [77]. (c) Provenance graph after applying S-DPR technique [36].

may be deleted, provided that the edge is not necessary to preserve dependency for some other node. Hossain et al. introduce two variants of DPR, Source Dependency-Preserving Reduction (S-DPR) in which only backward trace reachablity is preserved, and Full Dependency-Preserving Reduction (F-DPR) in which both backward and forward trace reachability are preserved. A visualization of S-DPR is given in Figure 3 (c). DPR was incorporated and discussed in recent work [35] and is also significant in boasting among the highest reduction rates in the literature.

Limitations of Prior Work. While the performance characteristics of these approaches were effectively evaluated in prior work (i.e., storage overheads), the security characteristics of the approximated logs have proven more difficult to quantify. When approximating the log, problems of graph reachability and interpretability may arise. Further, if events are merged or deleted, then interpreting the graph becomes more difficult. Worse yet, key details of the attack behaviors may be lost in ways that were unanticipated by the designers of the approximation technique.

Without exception [31, 36, 43, 71, 77], prior work was evaluated exclusively through attack scenario *case studies* in which a forensic analyst needs to answer a specific query during investigation. While illustrative of the potential benefits of a technique, this approach is ultimately anecdotal; it may be that analysts need to answer a broader range of queries that were not considered in the case study, or that the semantics of the specific attack scenario did not adequately capture the forensic utility of the approximated log. Worse yet, under this evaluation the efficacy of approximation methods is reduced to a binary 'yes' or 'no' depending on whether or not the analyst is able to achieve a hand-selected forensic goal. In the following section, we present a set of nuanced and descriptive methods for characterizing the security of an approximation technique. These metrics provide a continuous (i.e., non-binary) value for characterizing forensic validity.

3 FORENSIC VALIDITY METRICS

To better characterize the security utility of log approximation techniques, we formalize three complementary *forensic validity metrics* that can be used to evaluate an approximated log. Each metric encodes the amount of preserved evidence (i.e., log events) under a threat model that an analyst may encounter during the course of an investigation. In order to provide generalizable statements on the utility of approximation algorithms, we make no assumptions on the goals of the analysis or the events needed to return a specific query, but rather the number of events that remain intact. The key insight behind these metrics is that, rather than anecdotally demonstrating value in a specific attack behavior, utility should be measured as a property of the approximated graph or log. We propose three such metrics: *Lossless*, *Causality-Preserving*, and *Attack-Preserving*.

3.1 Lossless Forensics

This metric assumes **the adversary may not abide by system level abstractions for communication and coordination of malicious processes**. The adversary may use avenues such as obscure system-layer timing channels to exfiltrate data from a colluding process (e.g., [13, 69]). In such scenarios, the audit log may not explicitly contain all causal relationships but may encode inferrable implicit relations. For instance, the frequency and timing of system calls used to communicate within a covert channel may also be used by the system to infer whether a covert channel, and thus a causal relationship, exists. Unfortunately, such channels have been found in nearly every sector of computer architecture and are continually introduced as the hardware beneath operating systems evolves [23]; therefore, is it unreasonable to predict which events may contain implicit relations and thus we conservatively assume all events may encode such dependencies. Lossless Forensics assumes every dropped event potentially destroys implicit causal relations and thus is defined as follows.

Definition 3.1. Lossless Forensics: Given $G = (V, E)$ and approximation $G' = (V', E')$ in which $V' \subseteq V$ and $E' \subseteq E$, a lossless log implies that $E' = E$. Distance from losslessness can be measured as a continuous variable using the formula $1 - \frac{|E \setminus E'|}{|E|}$.

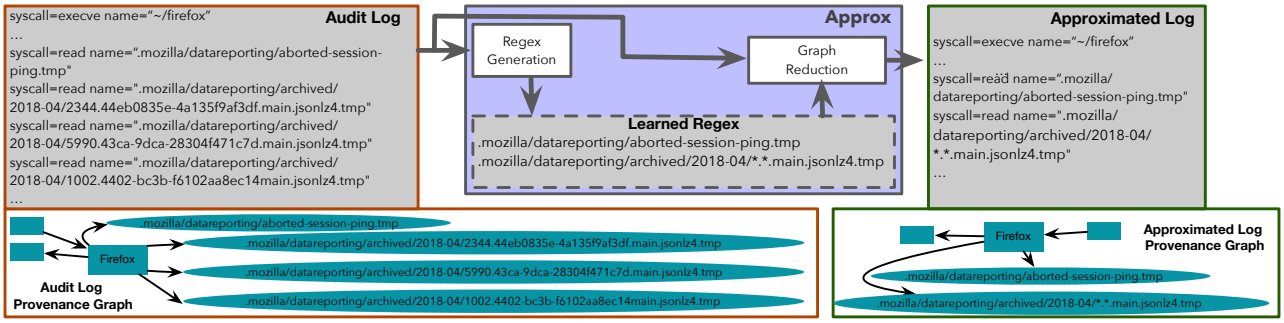


Figure 4: Overview of the LogApprox architecture and workflow. File activity and audit logs are analyzed by the *Regex Generation* routine, which generates bounded regular expressions that each describes a specific semantic behavior of the benign process (e.g., "loads shared objects"). These regex are then applied to the original log and matching entries are pruned. Causality is preserved in the associated provenance graph by connecting associated edges to the newly-collapsed node.

In other words, losslessness can be measured by the fraction of edges in E that are missing from E'. Note that it is not necessary to directly test the completeness of V, as each edge is a log event and the associated system entities in V are extracted from these tuples.

3.2 Causality-Preserving Forensics

This metric assumes that **the adversary abides by system level abstractions for communications and coordination of malicious processes**. As only explicit information flows are utilized by the adversary, one can be sure that the complete log faithfully contains an explicit description of all causal relationships. Within such a model, we used the well-studied subject of information flow (IF) to describe causal relationships and thus encode Xu et al.'s Causality-Preserving Reduction technique [77] as our metric. In a log that satisfies causality-preserving forensics, all events that encode new causal relationships are retained, whereas edges that are causally redundant are discarded. We define the metric as follows.

Definition 3.2. Causality-Preserving Forensics: Information flows from G are preserved in the approximated graph G'. An information flow is defined by the existence of a path between two edges in G. The following describes two situations where two edges describe the same information flow:

- Two read edges e_1, e_2 describe the same information flow if they have the same endpoints process p and file f, and no write to f or read from p to a file $f' \neq f$ occurred between e_1 and e_2.
- Two write edges e_1, e_2 describe the same information flow if they have the same endpoints process p and file f, read from p to a file occurred between e_1 and e_2.

Let E_{IF} be the set of edges matching this definition of distinct information flow in G. G' preserves causality iff $E_{IF} \in E'$. Distance from causality preservation can be measured using the formula $1 - \frac{E_{IF} \setminus (E' \cap E_{IF})}{E_{IF}}$.

3.3 Attack-Preserving Forensics

Under Attack-Preserving Forensics, **the adversary's system-level actions deviate from benign behavior**. With few exceptions, past approximation techniques apply reduction uniformly to all log events, regardless of whether or not the events describe malicious or benign activity. This is because, understandably, it is difficult to predict ahead of time which events are attack-relevant; however, in many cases it is safe to make assertions about predictable benign process behaviors. In a similar fashion to host anomaly detection, a suite of approximation reduction techniques attempt to distinguish benign from anomalous behavior and selectively reduce based on that classification. Building on causality-preservation, our final metric captures an approximation technique's ability to preserve attack-relevant causal relations without penalizing the reduction of attack-irrelevant log events.

Definition 3.3. Attack-Preserving Forensics: Given a causality-preserving approximated provenance graph G_{IF}, let $G_B \subseteq G_{IF}$ be the subgraph of naturally occurring benign system behavior and $G_A = G_{IF} \setminus G_B$ be the subgraph describing an attack campaign A. The approximated graph $G' \subseteq G_{IF}$ that contains $G'_A \subseteq G'$ is said to satisfy attack-preservation if G'_A is a causality-preserving approximation of G_A. Distance from attack-preservation can be measured using the formula $1 - \frac{|E_A \setminus E'_A|}{|E_A|}$, where E_A and E'_A are the edge sets of G_A and G'_A, respectively.

We make the following observations. First, this metric uses the causality-preserving approximated graph, not the lossless provenance graph, as its baseline. Additionally, this metric not only disregards unrelated benign activity, but exclusively considers the causal edge set that *uniquely describes* the attack campaign A. In other words, events that appear in the attack path of A that also appear in G_B are not considered in measurement. The intuition behind this approach is that edges that are shared between benign and behaviors hold little forensic value in investigations; they do not uniquely signify malicious activity, and hence are candidates for approximation. That said, because G'_A must still be causality-preserving, the attack-preservation metric assures that connectivity between processes and other essential information flow is preserved.

4 DESIGN

While the attack-preserving forensic validity represents a desirable trade-off between the utility and efficiency of audit logs, designing of an approximation technique that satisfies attack-preservation

in is quite challenging. Trivially, a causality-preserving graph is implicitly also an attack-preserving graph, but does not capitalize on the increased reduction opportunities afforded by the attack-preserving metric. To do so, it is necessary to gain an understanding of typically-occurring benign-process events and make generalized assertions about benign events that may occur in the future. Simultaneously, it is also necessary to assure that generalizations about benign process activity are not so general that attackers may abuse them to conceal malicious behaviors.

4.1 Overview

In this section, we present LogApprox, a novel solution to attack-preserving log approximation. An architectural overview for LogApprox is given in Figure 4. LogApprox searches for log reduction opportunities by generating regular expressions that describe benign process file I/O. These regexes are crafted in such a way to avoid overgeneralization, which might lead our system to filter events that describe unique attack patterns. After the regexes are generated, they are applied to past and future file I/O events (e.g., `creat`, `open`, `read`, `write`) in the audit log. For all events that match the same regex, the original filename is replaced with the regex pattern; then causality-preserving reduction is applied to remove events that are redundant from an information flow perspective. When the log is parsed into a provenance graph, the original file vertices are replaced with a single approximated file vertex. Because the information flows into and out of the regex remain unchanged, causality is otherwise preserved.

Our approach focuses exclusively on file activity for two important reasons. First, file I/O dominates the overall storage space of audit logs — 88.97% of all events in our evaluation datasets — and causality-preserving reduction [77] is not a total solution to this overhead because applications often write to a tremendous number of different files. Second, other system events (e.g., process, network) are more important to retain because they are essential to causal analysis. Process events are already low overhead and are needed to preserve the process tree, which is the "backbone" of causal analysis, while network events are needed to trace attacks across multiple hosts in the network. We thus choose to focus on file I/O in our design.

4.2 Reduction Algorithm

Our reduction algorithm begins by identifying, for each process, which files it has interacted with. Our provenance graph encodes this information by edges to (`read`) and from (`write`) a process node. From this list of files, we will generate groups of files with similar filenames. Replacing each group of files with a single placeholder in the provenance graph allows us to reduce the graph complexity and hence filter redundant log entries.

4.2.1 Regular Expression Learning. Our goal is to distribute a list of filenames associated with a process into groups of similar files. For a particular filename (eg. /usr/bin/ls) we distinguish the path (eg. /usr/bin/) and the name itself (eg. ls).

We define the *distance* between two names as the Levenshtein edit distance. Corresponding to this edit distance is an optimal alignment, which will be useful later in constructing a regular expression. We define similarity between two names x_1, x_2 as $(max\_len - \textsc{Dist}(x_1, x_2))/max\_len$, where $max\_len = \textsc{Max}(\textsc{Len}(x_1), \textsc{Len}(x_2))$. For the distance between two paths, we treat each directory name as a token. We only consider the distance when both paths are of equal depth, because differences in depth contain semantically relevant information. We define the path distance between two paths as the number of differing directory names.

Algorithm 1: Generate Groups

```
Data: List of filenames f_1, ..., f_n
Result: Groups of filenames G
1 Empty list G;
2 while not added file f_i do
3     Set G_i ← {f_i};
4     for filenames f_j not added to a group do
5         if Distance(Path(f_i), Path(f_j)) ≤ path_threshold and
            Similarity(Name(f_i), Name(f_j)) ≥ name_threshold then
6             Add f_j to G_i;
7     Append G_i to G;
8 Return G;
```

Algorithm 2: Generate Regular Expressions

```
Data: Groups of filenames G = [G_1, ..., G_n]
Result: Regular Expressions R = [r_1, ..., r_n]
1 Empty list R;
2 for every G_i ∈ S do
3     Set r_i ← G_i[1];
4     for filenames f ∈ G_i[2, ..., m] do
5         Find alignment a between r_i and f;
6         Replace modifications in a with wildcards;
7         Set r_i to a;
8     Append r_i to R;
9 Return R;
```

We compute the path distance and name similarity between all pairs of filenames. We choose groups such that all filenames within a group have a path distance below a specified threshold and a name similarity above a specified threshold. We empirically determined our path threshold to be 1 and our name threshold to be 0.7. We found this to be a good tradeoff that allows for aggressive log reduction capabilities while avoiding over-generalization.

Algorithm 1 groups the files into sets, where each file is at most a certain distance from another file in its set. These thresholds are computed separately for both file paths and file names, as shown on line 5. We return a list of such groups, each of which corresponds to a regex.

Algorithm 2 shows how to compute the regex corresponding to a group of files. We compute the path regex and name regex individually using this approach. To compute a regex from two strings, we find the edit distance alignment, and for every location where the tokens do not match, we replace it with a placeholder. We reduce this binary operation across the list of filenames. We coalesce placeholders and replace each with a token matching zero or more occurrences of a wildcard. We then concatenate the path and name regexes to generate a regex matching all files in the group. If there is only one element in the group, we return that filename as the corresponding regex.

This algorithm uses a binary regex generation function, taking as input the current progress and the next filename. It reduces this function across all filenames in a group. If the current regex matches the next filename, it will remain unchanged.

Algorithm 3: Log Reduction

```
   Data: Log and Provenance Graph
   Result: Reduced Log
1  for every process p do
2      Compute list of groups G;
3      Compute list of regexes R;
4      for every group G_i ∈ G do
5          for all reads e to file f ∈ G_i (in order) do
6              if since the last read, there was a write to f or a read from p to a
                file ≠ f then
7                  Keep e and overwrite f with r_i;
8              else
9                  Delete e;
10         for all writes e to file f ∈ G_i (in order) do
11             if since the last read, there was a read from p to a file ≠ f then
12                 Keep e and overwrite f with r_i;
13             else
14                 Delete e;
15 Return log;
```

4.2.2 Log Reduction. The overall log reduction procedure is presented in Algorithm 3. For every process, we generate a list of filenames corresponding to file accesses initiated by the process. These filenames are grouped as per Algorithm 1, and their corresponding regular expressions are generated as per Algorithm 2. For every group of filenames, we reduce the log entries between the process and these files preserving information flow. We do not reduce filenames corresponding to regular expressions with a length below a certain threshold, in our case, 10 characters, since they can overgeneralize. In effect, we are treating this group of filenames as one large file. For the log entries that have not been removed, we overwrite the filename with the regular expression corresponding to the group, as shown in lines 7 and 12.

This reduction algorithm acts on every log entry corresponding to a file access. It either keeps or deletes a log entry based on the information-flow preservation criteria outlined in the background.

5 IMPLEMENTATION

We implemented a log analysis tool that parses Linux Audit (`audit`) logs and CDM provenance graphs, the DARPA Engagement data format. Our tool generates a provenance graph in memory using the `SNAP` graph library [44]. Our provenance graph representation has nodes corresponding to processes, files, and other file-like objects (e.g. VFS, network sockets). Edges correspond to individual log entries. Our entire tool is implemented in 4000 lines of C++ code (calculated with `cloc` [2]). Our reduction filter, LogApprox, is implemented in 1000 lines of code.

To evaluate LogApprox against a representative set of exemplar systems, we also re-implemented Log Garbage Collection, introduced in Lee et. al [43], Causality-Preserving Reduction, introduced in Xu et. al [77], and Full and Source Dependence Preserving Reduction, introduced by Hossein et. al [36], based on their descriptions in the original paper. Our implementation of garbage collection is based on "Basic GC" in the original LogGC system [43], while we implement the techniques as described in [77], [36] as faithfully as possible. We do note that, because we did not have access to the authors' source code, it is possible that our implementation of these techniques deviate from the original systems, which could in part explain some of the differences in observed reduction rates in our experiments (e.g., Table 1). This said, we are confident that each of our implementations is consistent with the methodology presented in the original works, and are thus satisfactory for exploring different points in the design space of approximation strategies.

Approximation Technique	Originally Reported Reduction	Observed Reduction
CPR [77]	1.3-3.4X (23% - 71%)	1.3X (23%)
GC [43]	0-77.0X (0% - 99%)	1.6X (38%)
F-DPR [36]	4.5-91.5X (78% - 99%)	6.6X (85%)
S-DPR [36]	4.5-122.5X (78% - 99%)	11.2X (91%)

Table 1: Comparison of log reduction rates between reports from prior work and our own tests using the Theia dataset. For ease of reference, we report both of the 2 different statistics used in prior work; log reduction factor (Raw Log / Reduced Log) and in parenthesis the log reduction percentage (1 - Reduced Log / Raw Log).

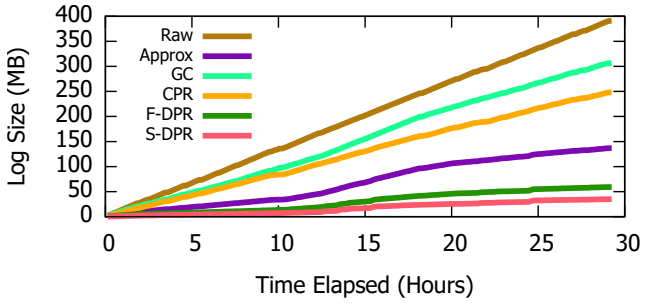
Because our tool is based on an in-memory database, it was necessary to manage memory overheads by partitioning the the provenance graph into epochs of time, each of which spans 5 minutes in our implementation. At the end of each epoch, the approximation technique(s) is (are) applied and the remaining log events are written to disk, at which point the process resets. We argue that this approach is not only more practical, but also provides approximately real-time telemetry for security monitoring services. However, this also means that redundant events between epochs are not identified in our implementation, making the observed log reduction rates in our evaluation an underestimation of the optimal reduced logs. Fortunately, in experimenting with different epoch sizes, we observed that the differences in log reduction rates were negligible, and that 5 minute epochs were sufficient to achieve similar log reduction rates to those reported in prior work.

6 EVALUATION

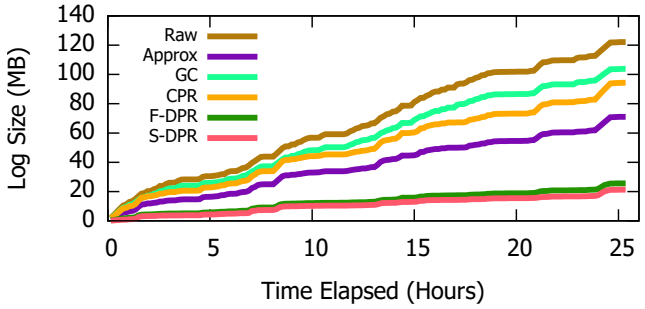
6.1 Datasets

We leverage the DARPA Transparent Computing Program dataset for our evaluation [15]. This dataset was collected during an APT simulation exercise (Engagement #3) in April 2018, containing events from a series of target hosts, along with ground truth information about the attacks. It has been used in prior work as a source for authentic examples of adversarial behavior [17, 34]. We select the Linux-based logs corresponding to two hosts from the DARPA Engagement, Trace and Theia.

In addition to the Transparent Computing engagements, we also select 6 attack scenarios leveraged in prior work (e.g., [30, 32, 41, 54]) to evaluate the efficacy of log reduction systems. The `unrealircd` [18], `vsftpd` [19], and `webmin` [20] exploits all leverage input vulnerabilities to achieve remote code execution (RCE). A payload is then executed, which launches a reverse shell back to the attacker machine that executes commands such as `ifconfig`. The Wordpress vulnerability [63] and Webshell are exploits that execute a payload through a web server that calls back to the attacker machine as before. The Firefox vulnerability is an exploit on Firefox 54.0.1 that gains execution through a malicious ad server. This vulnerability was exploited as part of DARPA TC



(a) Theia Dataset



(b) Trace Dataset

Figure 5: Cumulative Log Size Stored after different reduction techniques on the Theia and Trace datasets. In both datasets, LogApprox outperforms all approaches besides Full and Source Dependence.

Engagement #3 [15]. We use the subset of logs associated with this particular exploit during this evaluation.

6.2 Performance Evaluation

To evaluate the performance of our reduction technique at scale, we concatenate all logs from a particular dataset and run our reduction algorithm. The end of an epoch corresponds to either the end of a log file or when 5 minutes elapse.

6.2.1 Re-Implementation Validity. We begin by comparing the reduction rates of our re-implementations to the findings of the original works. The results for the Theia dataset are shown in Table 1. As log reduction has been reported differently in different papers, we provide both the reduction factor and reduction percentage statistics. While some of our implementations did not reach the peak reduction rates observed in prior work, they are all consistent with prior observations of these systems. The differences in performance can likely be attributed to the process behaviors used in the test datasets. Our implementation of CPR performed closest to the reports of the original paper. This confirms our intuition that CPR is the most generally applicable of past approximation techniques and is thus a solid basis for our attack-preserving extension.

6.2.2 Reduction Performance. We now compare the reduction performance of LogApprox to other approximation techniques using the Theia and Trace datasets. Figures 5a and 5b show the growth

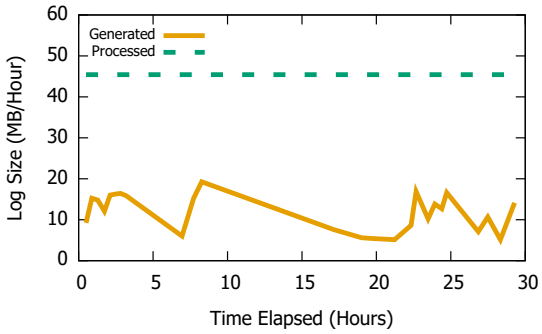


Figure 6: Log Processing speed relative to the generation of logs in the Theia dataset, which we replayed in real-time.

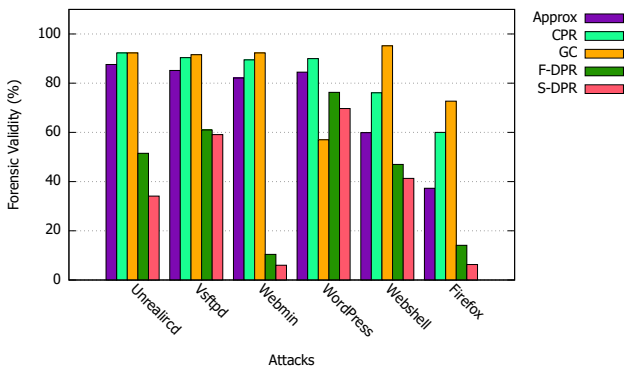
in log size for all approximation techniques over the course of the attack engagements in Theia and Trace respectively. Note that the final position of the lines in Figure 5a corresponds to the reduction rates observed by past techniques in Table 6. We observed that all approximation techniques performed better on the Theia dataset; this is because the Theia engagement had many significantly more events, offering more opportunities for reduction. In contrast, the Trace dataset contains many small and disjoint log traces, creating fewer reduction opportunities.

We observed that LogApprox consistently performed between GC/CPR and F-DPR/S-DPR. LogApprox's reduction factor was 2.87X and 1.72X (removing 42% and 65% of the raw log) on the Theia and Trace datasets, respectively. Hossain et al.'s F-DPR and S-DPR [36] had the largest reduction factors, respectively boasting up to 85% and 91% reductions in log size during the Theia engagement. In Section 6.4, we determine whether this extreme efficiency comes at the cost of valuable forensic information.

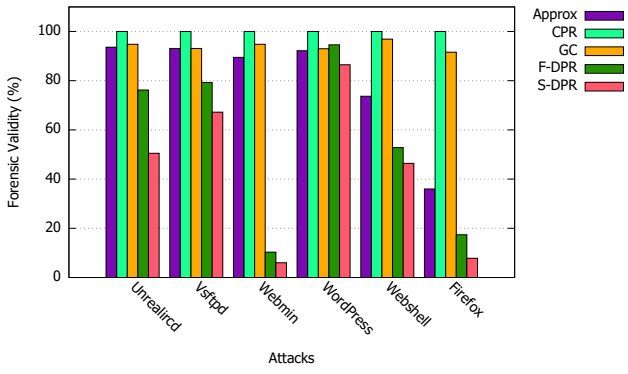
6.2.3 Log Ingest Performance. A final consideration for the performance of our LogApprox system is whether or not its reduction analysis can keep pace with the speed of log event creation. Figure 6 compares the event generation rate of the Theia dataset. Because LogApprox was evaluated by replaying this dataset, as opposed to observing events as they occur in realtime, we calculate the maximum ingest rate of LogApprox in MB per hour by observing how long it took LogApprox to process the 391 MB of Theia log files. We see that LogApprox's ingest rate of 45.44 MB per hour far outpaces the amount of logs generated during the Transparent Computing engagement and is thus suitable for general use.

6.3 Utility Evaluation

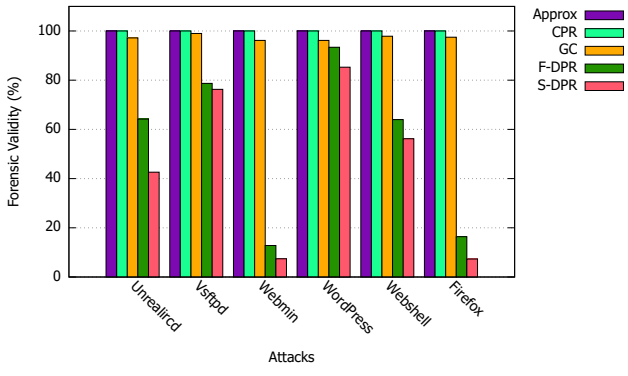
We now evaluate the utility of approximated logs through our forensic validity metrics. The Trace and Theia datasets contain multiple intrusion attempts, only some of which are well-documented, so they are not an effective ground truth for use with our metrics. Instead, we evaluate the utility of these systems based on a curated set of real-world program exploits. Each exploit was implemented by the authors based on public documentation or downloaded from public repositories (e.g., exploitdb), then launched in a controlled VM environment to capture the associated audit logs. Following the test, two authors reviewed the log entries and marked all events



(a) Lossless



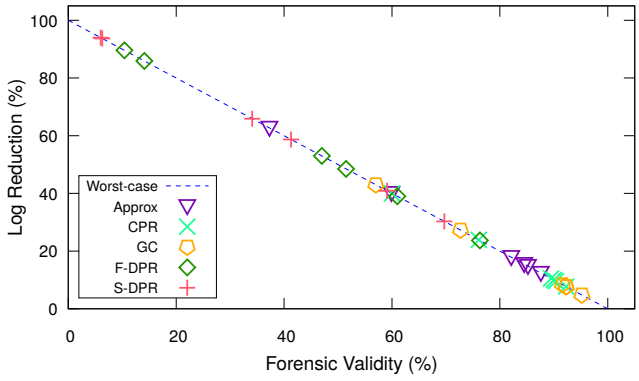
(b) Causality-Preserving



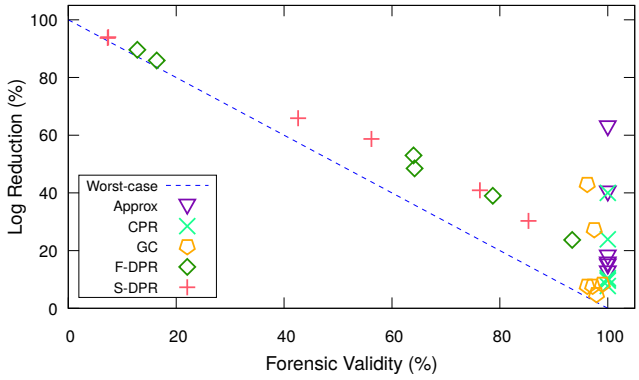
(c) Attack-Preserving

Figure 7: Forensic validity measures for different reduction techniques (taller is better). Only LogApproxand CPR fully satisfy Attack-preserving Forensics, while other techniques sacrifice significant attack-relevant information.

found on the forward trace paths from the point of entry. There was no disagreement between authors on which entries fell on the path. All log events (including those not directly on the attack paths) are included in the below tests.



(a) Lossless



(b) Attack-Preserving

Figure 8: Scatterplots of the performance-utility tradeoff according to the notion of *Lossless* and *Attack-Preserving* Forensics. Each point on the graph represents one of the 6 attack scenarios considered by each approximation technique.

Figure 7 shows our results for each of the forensic validity metrics. Taller bars are better as they signify that more forensic evidence was retained. For the Lossless Forensics metric in 7a, we see that all sacrifice significant forensic context, especially F-DPR and S-DPR because they boast the largest log reduction rates. This is not surprising, or even concerning, as prior approaches to log approximation did not consider the extreme threat model for which timing side channels need be accounted. An interesting direction for future work would be to consider approximation techniques that preserve information about the timing distribution of repeated events.

For the Causality-Preserving Forensics metric, we confirm that CPR preserves 100% of forensic information as expected. GC retains a consistently high amount of forensic evidence, retaining 91% or better in all cases. LogApprox appears moderately conservative in forensic reduction, preserving 90% causality or greater in 4 of the 6 scenarios while retaining 36% in the worst-case. F-DPR and S-DPR's aggressive nature retains between 45%-80% in 3 of the 6 scenarios, while retaining as much as 95% in the best case scenarios, and as little as 6.0%-17.0% in the 2 worst-cases. This suggests that, even by

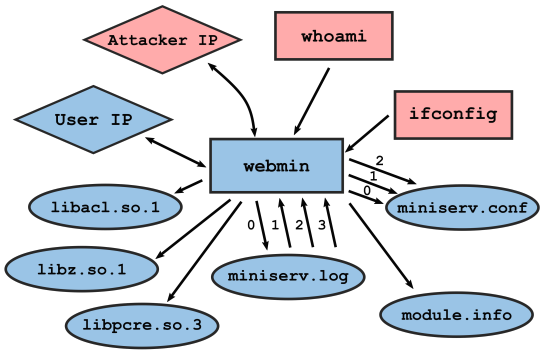
established measures of forensic validity such as information flow preservation, these techniques are sacrificing potentially important forensic context. Whether or not the lost data is actually relevant to analysis of a particular attack is likely case-specific. For example, Hossain et al. applied F-DPR on Trace while retaining the ability to identify the attack [35]. The lost data may negatively impact other analysis that was unneeded for their particular purposes.

Results for Attack-Preserving Forensics are shown in Figure 7. LogApprox retains 100% of forensic evidence under the attack-preserving metric, as does CPR. Because attack-preserving evidence is a subset of causality-preserving evidence, the 3 remaining approximation techniques all perform slightly better under this model. GC comes close to satisfying attack-preserving validity, achieving between 96% and 99% in all scenarios. This result makes sense as GC was specifically designed to clean-up process behaviors that are associated with benign process activity, such as temporary file I/O. Unlike causality-preserving forensics, attack-preserving forensics does not penalize deletion of typical process activity. S-DPR and F-DPR retain similar forensic validity under attack-preserving as they did under causality-preserving, maintaining between 42%-80% in 3 cases, 85%-94% in the best case, and 7%-17% in the 2 worst cases. This indicates that S-DPR and F-DPR filter log events without regard for typical or atypical behavior.

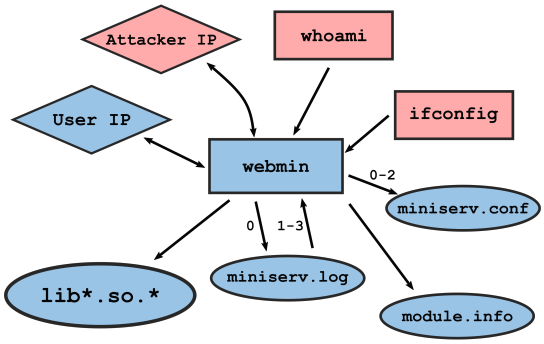
6.4 Performance vs. Utility

We plot the performance-utility tradeoff per attack scenario in Figures 1 (Causality-Preserving), 8a (Lossless), and 8b (Attack-Preserving), where utility is one of the forensic validity metrics defined in Section 3. The blue dashed line denotes the worst-case utility an algorithm could achieve in a scenario where all dropped events are forensically relevant (and thus no redundancies exist). An "ideal" technique would place its points in the top right corner, where both the validity and reduction are maximized. The optimal and worst-case utility curves are both metric and scenario specific but plotting trends of reduction algorithms against such strawman metrics still provides useful insight. These plots enable informed decision-making about the value of additional log retention under different threat models. For example, S-DPR trades off very high reduction (and thus space efficiency) for lower forensic validity. If space is a limiting factor, it may be preferable to save sparser data spanning a longer time period with a highly reducing technique like S-DPR over denser data spanning a shorter time period from less approximating techniques. This would allow analysts to reconstruct long-term basic facts about an attack (e.g., root cause) at the expense of short-term details.

Intuitively, no technique can outperform the worst-case utility baseline under the *Lossless Forensics* metric (Figure 8a), as every dropped event is considered a utility loss. Several interesting patterns emerge when analyzing the tradeoffs in causality preservation (Figure 1). Approx, F-DPR, and S-DPR all exhibit significant variance between attack scenarios, but their performance-utility ratio remains roughly proportional. This variability in tradeoffs reflects the design goals of these systems; e.g., S-DPR focuses exclusively on identifying the correct system entities in backtraces. CPR, by design, maxes out utility here, but GC performs nearly as well. This



(a) Lossless Provenance Graph



(b) LogApprox's Attack-Preserving Provenance Graph

Figure 9: Simplified provenance graphs of the webmin exploit. (a) depicts the raw audit log, while (b) shows the provenance graph produced after LogApprox is applied.

may not be the case in attacks that involve significant destruction of system entities, e.g., an attacker exfiltrates, then destroys data.

The value of Attack-Preserving Forensics becomes most evident in Figure 8b. As only a subset of causality-preserving events uniquely describe attack semantics, LogApprox is able to match CPR in utility while significantly outperforming it in log reduction in each scenario. While GC is not as aggressive, it continues to perform well under this metric, validating Lee et al.'s design decisions in the earliest log approximation paper to appear in the literature.

6.5 LogApprox Case Study

To better understand the benefits of LogApprox as compared to prior work, we now examine the Webmin exploit above in more detail. Webmin is a web-based configuration tool for Unix systems, thus it is a prime target for attackers as it can be leveraged for lateral movement on the network. The exploit allows for unauthenticated remote code execution when the web server is configured for users with an expired password to enter a new one. With the appropriate payload, a reverse shell can be spawned and post-exploitation tools (eg. `LinEnum.sh` [64]) can be downloaded and run on the server machine. We chose to run commands manually, namely `whoami` and `ifconfig`, to demonstrate that we have remote access.

Figure 9 displays provenance graphs corresponding to the unapproximated log (9a) and LogApprox's attack-preserving log (9b). These graphs have been simplified for visualization purposes, as the unapproximated and reduced log contain 13026 and 11013 edges respectively. The numbered edges in the graph correspond to timestamps, and attack-specific vertices are shaded in red. Due to space, we omit the causality-preserving attack graph produced by CPR; it is the same as LogApprox's graph except that the `lib*.so.*` vertex is replaced by the original 3 vertices.

As can be seen, is is possible to reconstruct the attack completely through interpreting the graph. The CPR filter applied to the LogApprox graph does not meaningfully effect investigation because information flow is preserved. However, the complexity of LogApprox is reduced due to the fact that benignly-occurring graph components are approximated by a regular expression. Note that some forensic information that is potentially attack-relevant is sacrificed by LogApprox. Specifically, because the libraries are loaded into memory prior to the attack occurring, it is possible that their contents informed outbound data transfers to the remote Attacker IP. While this is possible, our argument is that lossless descriptions of these library files remain unuseful because their significance to the attack will not be apparent from the system-layer audit log; these libraries are loaded all of the time by `webmin` during typical activity, so they do not provide any threat intelligence even though they may technically be relevant to the attack. Hence, it is reasonable to prune and approximate these graph components.

7 DISCUSSION

7.1 Optimality of LogApprox

We do not argue that LogApprox is the ideal instantiation of attack-preserving forensics. In this work, we only target one major subsystem of system activity, namely file I/O, because it commonly accounts for a large percentage of log volume and can be safely filtered without disrupting the causality of more complex attack behaviors such as lateral movement or data exfiltration. Approximating other system entities, e.g., remote IP addresses, is much more perilous, as even a common remote connection may describe an important attack behavior during lateral movement. It is interesting to consider how the notion of attack-preserving forensics could be extended in future work.

7.2 Limitations of Attack Corpus

One notable limitation that we share with prior work is that we are still limited in the corpus of attack behaviors that we evaluate against. A standardized battery of attack behaviors that coincide with our metrics may enable us to capture more nuances in the performance-utility tradeoffs of approximation techniques, but maintaining such an attack corpus is extremely difficult due to ever-changing adversary techniques and tactics [55]. We attempt to mitigate this limitation by using state-of-the-art attack engagements produced for the DARPA Transparent Computing program. Regardless, our datasets are sufficient to validate our hypothesis that approximation techniques must be evaluated both for storage performance as well as forensic utility.

7.3 Security Analysis of LogApprox

We now consider the security implications of LogApprox' regex-based File IO compression procedure. An intrusion is comprised of an inter-dependent set events that includes process, network, and file activities, all of which are captured by the unmodified audit log. LogApprox does not affect process and network log events in any way, meaning that network-based activities (e.g., lateral movement) and the interdependencies of attacker-controlled processes are fully documented. In the case of file activities, we can further divide File IO events of the attacker-controlled process between those that are similar to the benign process (i.e., regex matches) and those that are wholly distinct (i.e., regex non-matches). We argue that it is safe to approximate matched File IO events; this is because such events cannot be used to detect an intrusion, and if a process is linked to in an intrusion by other means than the files it accesses are also implicated. As a result, the intrusion's incident response plan would necessarily entail re-imaging the compromised process, causing any malicious data hidden by the attacker in a 'benign' file to be erased. In contrast, File IO events that don't match a regex are retained faithfully by LogApprox. These non-match events are far more important to threat detection and investigation – in order for the attacker to accomplish their objectives on the system, they will need to establish persistence, escalate privilege, etc. This means that they will need to deviate from the behaviors of the benign application, which is unlikely to have accessed sensitive system files needed for these tactics. Avoiding compression of uncommon File IO event also ensures that this information remains available to intrusion detection tools.

If an attacker is aware of LogApprox's presence, they may pattern their intrusion so as to to minimize their use of uncommon File IO behaviors. Recall that LogApprox retains a lossless version of the process tree, network links, and uncommon file behaviors; the worst case scenario is therefore that the attacker is able to spread through the system via common benign file behaviors. *Even in this scenario,* LogApprox *is able to produce an end-to-end attack graph of the intrusion*, but the graph will admit a limited number of false dependencies as compared to an attack graph produced by the uncompressed log. Specifically, if an attack path flows through a LogApprox regex, then the processes that interacted with other files matched by that regex will be incorrectly included in the graph. That said, in practice such a feat is unlikely. As shown in Figures 4 and 9, in practice we observe that LogApprox primarily generates regexes that describe "dead-end" or "one-way" IO behaviors such as temporary files, configuration files, or read-only shared objects. These file IO behaviors cannot create process-to-process links, so they would not lead to false dependencies in an attack graph.

A final consideration is whether or not the adversary can manipulate the way in which LogApprox generates regexes. It would be highly convenient for the attacker to create an overly broad regex, e.g., "`match *`", which would cause all file IO to be erased from the log. However, due to LogApprox' parameterizable name and path similarity thresholds, it is not possible to distort benign access patterns into an overly broad regex – path matching can deviate from concrete access patterns by at most one directory level, and files can only be matched if they are at least 70% similar to one another. Instead, an attacker could repeatedly issue variations on

the malicious access pattern until a regex is created; however, this is far from covert. In order for this to happen, the attacker essentially needs to repeatedly issue the very command they were hoping to conceal. To further reduce the likelihood of this occurring, the administrators could fine-tune LogApprox's parameters (e.g., set path distance to zero) to improve security at the cost of diminished reduction capacity.

8 RELATED WORK

This work is the first to formalize and quantify the forensic validity of audit logs that have been subjected to approximation techniques. We discuss past approximation techniques in Section 2. Like LogApprox, Tang et al.'s NodeMerge [71] templatizes process behaviors for compression purposes, but only operates on *read-only* file activity at the *beginning* of process execution. As a result, NodeMerge could likely create a template for the Figure 9 example, but not for other process behaviors accommodated by LogApprox such as temporary file I/O. We chose not to evaluate NodeMerge because it requires a training phase that is highly deployment-specific, making it difficult to understand how the approach generalizes.

Beyond the challenge of log reduction, another important consideration is log security; if audit logs can be manipulated by the attacker, they cannot be trusted in an investigation. A variety of cryptographic approaches enable tamper-evident logging (e.g., [7, 16, 27, 33, 47, 67, 68, 78, 79]). Research has also explored software-based solutions to securing audit logs, demonstrating that reference monitor guarantees [3] are sufficient to assure log integrity [5, 53, 60, 62]. Additionally, recent works utilize cryptographic primitives for log integrity within trusted execution environments [37, 58] and kernel audit frameworks [59]. Like all other work in the space, LogApprox depends on the presence of mechanisms that can assure and attest to the integrity of audit logs.

Considerable attention has given to extracting high-level semantic insights from low-level system logs and graphs. A central issue with system logs is *dependency explosion*, a semantic gap problem in which long-lived processes (or data objects) appear to have a large number dependencies when viewed from the system layer. A variety of execution partitioning techniques have been proposed to partition opaque dependencies into small autonomous units of work [30, 32, 40–42, 48–51]. Execution partitioning creates additional opportunities for log reduction [43], but these opportunities are not considered in our work because execution partitioning solutions are application-specific, not system-wide. Regardless, we believe these techniques are interoperable with LogApprox, although they may not be as essential because LogApprox already removes a large percentage of false dependencies related to benign execution units. Prior work also considers related semantic gap problems, including the reconciliation of system-level logs with application logs [32, 61] and the identification of high-level semantic behaviors [34, 46, 73]. These techniques should also be compatible with LogApprox, provided that the analyst is only interested in fully reconstructing attack-related sequences of events.

Increasingly, causal analysis techniques are being incorporated into intrusion detection tasks. Manzoor et al. [52], Han et al. [26], and Wang et al. [74] present anomaly detection algorithms based on the analysis of provenance graphs. Hassan et al. address the false alert problem common in commercial threat detection software using provenance-based alert triage [30]. Milajerdi et al. present a rule-based approach for detecting attacker tactics [54], similar to commercially-available Endpoint Detection & Response (EDR) software, but based on provenance graph structures instead of flat audit event sequences. Subsequently, Hassan et al. extended a commercial EDR tool with lightweight provenance-based alert correlation [29]. Their approach to making their technique practical for large enterprise environments is to aggressively filter provenance graphs such that only queries about inter-alert dependency can be answered by the approximated log. In contrast to other approximation techniques discussed in this work, Hassan et al.'s approach is not intended for use in generic threat investigation scenarios where many forms of causal query must be supported. While we primarily consider threat investigation, our forensic validity metrics can also be interpreted as an indicator of how log approximation techniques may assist or impair intrusion detection tasks.

9 CONCLUSION

The security utility of log approximation techniques has proven difficult to measure; in this work, we codify a set of forensic validity properties that can measure the utility of approximated logs. We present LogApprox, a new approach to log approximation that leverages repetition in file I/O behavior to learn regular expressions that characterize common access patterns, allowing aggressive reduction of benign log events. We conduct a principled evaluation of both the performance and utility of LogApprox, comparing it to a representative set of techniques from prior work. It is our hope that this work provides expressive baselines for the evaluation of future log approximation systems.

ACKNOWLEDGMENTS

The authors would like to thank the anonymous referees for their valuable comments and helpful suggestions. This work is supported in part by NSF 17-50024. The views expressed are those of the authors only.

REFERENCES

[1] Raza Ahmad, Melanie Bru, and Ashish Gehani. 2018. Streaming Provenance Compression. In *Provenance and Annotation of Data and Processes*, Khalid Belhajjame, Ashish Gehani, and Pinar Alper (Eds.). Springer International Publishing, Cham, 236–240.
[2] AlDanial. 2019. `cloc`: *Count Lines of Code.*
[3] James P. Anderson. 1972. *Computer Security Technology Planning Study.* Technical Report ESD-TR-73-51. Air Force Electronic Systems Division.
[4] Adam Bates, Kevin R. B. Butler, and Thomas Moyer. 2015. Take Only What You Need: Leveraging Mandatory Access Control Policy to Reduce Provenance Storage Costs. In *7th Workshop on the Theory and Practice of Provenance* (Edinburgh, Scotland) *(TaPP'15).*
[5] Adam Bates, Dave Tian, Kevin R.B. Butler, and Thomas Moyer. 2015. Trustworthy Whole-System Provenance for the Linux Kernel. In *Proceedings of 24th USENIX Security Symposium* (Washington, D.C.).
[6] Adam Bates, Dave Tian, Grant Hernandez, Thomas Moyer, Kevin R.B. Butler, and Trent Jaeger. 2017. Taming the Costs of Trustworthy Provenance through Policy Reduction. *ACM Trans. on Internet Technology* 17, 4 (sep 2017), 34:1–34:21.
[7] Mihir Bellare and Bennet Yee. 1997. *Forward integrity for secure audit logs.* Technical Report. Computer Science and Engineering Department, University of California at San Diego.
[8] Y. Ben, Y. Han, N. Cai, W. An, and Z. Xu. 2018. T-Tracker: Compressing System Audit Log by Taint Tracking. In *2018 IEEE 24th International Conference on Parallel and Distributed Systems (ICPADS).* 1–9. https://doi.org/10.1109/PADSW.2018.8645035

[9] Tara Siegel Bernard, Tiffany Hsu, Nicole Perlroth, and Ron Lieber. 2019. Equifax Says Cyberattack May Have Affected 143 Million in the U.S. https://www.nytimes.com/2017/09/07/business/equifax-cyberattack.html. Last accessed October 16, 2020.

[10] Carbon Black. 2018. Global Incident Response Threat Report. https://www.carbonblack.com/global-incident-response-threat-report/november-2018/. Last accessed 04-20-2019.

[11] Microsoft: Windows Dev Center. 2018. *About Event Tracing.*

[12] Microsoft: Windows Dev Center. 2018. *Event Logging.*

[13] Ang Chen, W. Brad Moore, Hanjun Xiao, Andreas Haeberlen, Linh Thi Xuan Phan, Micah Sherr, and Wenchao Zhou. 2014. Detecting Covert Timing Channels with Time-Deterministic Replay. In *11th USENIX Symposium on Operating Systems Design and Implementation (OSDI 14)*. USENIX Association, Broomfield, CO, 541–554. https://www.usenix.org/conference/osdi14/technical-sessions/presentation/chen_ang

[14] Chen Chen, Harshal Tushar Lehri, Lay Kuan Loh, Anupam Alur, Limin Jia, Boon Thau Loo, and Wenchao Zhou. 2017. Distributed Provenance Compression. In *Proceedings of the 2017 ACM International Conference on Management of Data* (Chicago, Illinois, USA) *(SIGMOD '17)*. ACM, New York, NY, USA, 203–218. https://doi.org/10.1145/3035918.3035926

[15] DARPA Transparent Computing. 2020. *Transparent Computing Engagement 3 Data Release.*

[16] Scott A. Crosby and Dan S. Wallach. 2009. Efficient data structures for tamper-evident logging. In *In Proceedings of the 18th USENIX Security Symposium.*

[17] Birhanu Eshete, Rigel Gjomemo, Md Nahid Hossain, Sadegh Momeni, R. Sekar, Scott D. Stoller, V. N. Venkatakrishnan, and Junao Wang. 2016. Attack Analysis Results for Adversarial Engagement 1 of the DARPA Transparent Computing Program. *ArXiv* abs/1610.06936 (2016).

[18] Exploit-DB. 2010. *UnrealIRCd 3.2.8.1 - Backdoor Command Execution.*

[19] Exploit-DB. 2011. *vsftpd 2.3.4 - Backdoor Command Execution.*

[20] Exploit-DB. 2019. *Webmin 1.920 - Unauthenticated Remote Code Execution.*

[21] FreeBSD. 2019. DTrace on FreeBSD. https://wiki.freebsd.org/DTrace. Last accessed October 16, 2020.

[22] Peng Gao, Xusheng Xiao, Ding Li, Zhichun Li, Kangkook Jee, Zhenyu Wu, Chung Hwan Kim, Sanjeev R. Kulkarni, and Prateek Mittal. 2018. SAQL: A Stream-based Query System for Real-Time Abnormal System Behavior Detection. In *27th USENIX Security Symposium (USENIX Security 18)*. USENIX Association, Baltimore, MD, 639–656. https://www.usenix.org/conference/usenixsecurity18/presentation/gao-peng

[23] Qian Ge, Yuval Yarom, David Cock, and Gernot Heiser. 2018. A survey of microarchitectural timing attacks and countermeasures on contemporary hardware. *Journal of Cryptographic Engineering* 8, 1 (2018), 1–27.

[24] Ashish Gehani, Minyoung Kim, and Jian Zhang. 2009. Steps Toward Managing Lineage Metadata in Grid Clusters. In *1st Workshop on the Theory and Practice of Provenance* (San Francisco, CA) *(TaPP'09)*.

[25] Ashish Gehani and Dawood Tariq. 2012. SPADE: Support for Provenance Auditing in Distributed Environments. In *Proceedings of the 13th International Middleware Conference* (ontreal, Quebec, Canada) *(Middleware '12)*. Springer-Verlag New York, Inc., New York, NY, USA, 101–120. http://dl.acm.org/citation.cfm?id=2442626.2442634

[26] Xueyan Han, Thomas Pasqueir, Adam Bates, James Mickens, and Margo Seltzer. 2020. Unicorn: Runtime Provenance-Based Detector for Advanced Persistent Threats. In *27th ISOC Network and Distributed System Security Symposium (NDSS'20)*.

[27] Ragib Hasan, Radu Sion, and Marianne Winslett. 2009. The Case of the Fake Picasso: Preventing History Forgery with Secure Provenance. In *Proceedings of the 7th USENIX Conference on File and Storage Technologies (FAST'09)*. San Francisco, CA, USA.

[28] Wajih Ul Hassan, Nuraini Aguse, Mark Lemay, Thomas Moyer, and Adam Bates. 2018. Towards Scalable Cluster Auditing through Grammatical Inference over Provenance Graphs. In *Proceedings of the 25th ISOC Network and Distributed System Security Symposium (NDSS'18)*. San Diego, CA, USA.

[29] Wajih Ul Hassan, Adam Bates, and Daniel Marino. 2020. Tactical Provenance Analysis for Endpoint Detection and Response Systems. In *41st IEEE Symposium on Security and Privacy (SP) (Oakland'20)*.

[30] Wajih Ul Hassan, Shengjian Guo, Ding Li, Zhengzhang Chen, Kangkook Jee, Zhichun Li, and Adam Bates. 2019. NoDoze: Combatting Threat Alert Fatigue with Automated Provenance Triage. In *26th ISOC Network and Distributed System Security Symposium (NDSS'19)*.

[31] Wajih Ul Hassan, Mark Lemay, Nuraini Aguse, Adam Bates, and Thomas Moyer. 2018. Towards Scalable Cluster Auditing through Grammatical Inference over Provenance Graphs. In *25th Annual Network and Distributed System Security Symposium, NDSS 2018, San Diego, California, USA, February 18-21, 2018*. The Internet Society. http://wp.internetsociety.org/ndss/wp-content/uploads/sites/25/2018/02/ndss2018_07B-1_Hassan_paper.pdf

[32] Wajih Ul Hassan, Mohammad Noureddine, Pubali Datta, and Adam Bates. 2020. OmegaLog: High-Fidelity Attack Investigation via Transparent Multi-layer Log Analysis. In *27th ISOC Network and Distributed System Security Symposium (NDSS'20)*.

[33] Jason E. Holt. 2006. Logcrypt: Forward Security and Public Verification for Secure Audit Logs. In *Proc. of the Australasian Information Security Workshop (AISW-NetSec)*.

[34] Md Nahid Hossain, Sadegh M. Milajerdi, Junao Wang, Birhanu Eshete, Rigel Gjomemo, R. Sekar, Scott Stoller, and V.N. Venkatakrishnan. 2017. SLEUTH: Real-time Attack Scenario Reconstruction from COTS Audit Data. In *26th USENIX Security Symposium (USENIX Security 17)*. USENIX Association, Vancouver, BC, 487–504. https://www.usenix.org/conference/usenixsecurity17/technical-sessions/presentation/hossain

[35] Md Nahid Hossain, Sanaz Sheikhi, and R. Sekar. 2020. Combating Dependence Explosion in Forensic Analysis Using Alternative Tag Propagation Semantics. In *Proceedings of the 2020 IEEE Symposium on Security and Privacy (S&P)*.

[36] Md Nahid Hossain, Junao Wang, R. Sekar, and Scott D. Stoller. 2018. Dependence-preserving Data Compaction for Scalable Forensic Analysis. In *Proceedings of the 27th USENIX Conference on Security Symposium* (Baltimore, MD, USA) *(SEC'18)*. USENIX Association, Berkeley, CA, USA, 1723–1740. http://dl.acm.org/citation.cfm?id=3277203.3277331

[37] Vishal Karande, Erick Bauman, Zhiqiang Lin, and Latifur Khan. 2017. SGX-Log: Securing System Logs With SGX. In *Proceedings of the 2017 ACM on Asia Conference on Computer and Communications Security (ASIA CCS '17)*.

[38] Samuel T. King and Peter M. Chen. 2003. Backtracking Intrusions. *SIGOPS Oper. Syst. Rev.* 37, 5 (Oct. 2003), 223–236. https://doi.org/10.1145/1165389.945467

[39] Brendan I. Koerner. 2019. Inside the Cyberattack That Shocked the US Government. https://www.wired.com/2016/10/inside-cyberattack-shocked-us-government/. Last accessed October 16, 2020.

[40] Yonghwi Kwon, Dohyeong Kim, William Nick Sumner, Kyungtae Kim, Brendan Saltaformaggio, Xiangyu Zhang, and Dongyan Xu. 2016. LDX: Causality Inference by Lightweight Dual Execution. In *Proceedings of the Twenty-First International Conference on Architectural Support for Programming Languages and Operating Systems* (Atlanta, Georgia, USA) *(ASPLOS '16)*. ACM, New York, NY, USA, 503–515. https://doi.org/10.1145/2872362.2872395

[41] Yonghwi Kwon, Fei Wang, Weihang Wang, Kyu Hyung Lee, Wen-Chuan Lee, Shiqing Ma, Xiangyu Zhang, Dongyan Xu, Somesh Jha, Gabriela Ciocarlie, Ashish Gehani, and Vinod Yegneswaran. 2018. MCI: Modeling-based Causality Inference in Audit Logging for Attack Investigation. In *Proc. of the 25th Network and Distributed System Security Symposium (NDSS'18)*.

[42] Kyu Hyung Lee, Xiangyu Zhang, and Dongyan Xu. 2013. High Accuracy Attack Provenance via Binary-based Execution Partition. In *Proceedings of NDSS '13* (San Diego, CA).

[43] Kyu Hyung Lee, Xiangyu Zhang, and Dongyan Xu. 2013. LogGC: Garbage Collecting Audit Log. In *Proceedings of the 2013 ACM SIGSAC conference on Computer and Communications Security* (Berlin, Germany) *(CCS '13)*. ACM, New York, NY, USA, 1005–1016. https://doi.org/10.1145/2508859.2516731

[44] Jure Leskovec. 2009. *SNAP: Stanford Network Analysis Project.*

[45] Yushan Liu, Mu Zhang, Ding Li, Kangkook Jee, Zhichun Li, Zhenyu Wu, Junghwan Rhee, and Prateek Mittal. 2018. Towards a Timely Causality Analysis for Enterprise Security. In *Proceedings of the 25th ISOC Network and Distributed System Security Symposium (NDSS'18)*. San Diego, CA, USA.

[46] Sadegh M. Milajerdi, Birhanu Eshete, Rigel Gjomemo, and Venkat N. Venkatakrishnan. 2018. ProPatrol: Attack Investigation via Extracted High-Level Tasks. In *Information Systems Security*, Vinod Ganapathy, Trent Jaeger, and R.K. Shyamasundar (Eds.). Springer International Publishing, Cham, 107–126.

[47] Di Ma and Gene Tsudik. 2009. A new approach to secure logging. *ACM Transactions on Storage (TOS)* 5, 1 (2009).

[48] Shiqing Ma, Kyu Hyung Lee, Chung Hwan Kim, Junghwan Rhee, Xiangyu Zhang, and Dongyan Xu. 2015. Accurate, Low Cost and Instrumentation-Free Security Audit Logging for Windows. In *Proceedings of the 31st Annual Computer Security Applications Conference* (Los Angeles, CA, USA) *(ACSAC 2015)*. ACM, New York, NY, USA, 401–410. https://doi.org/10.1145/2818000.2818039

[49] Shiqing Ma, Juan Zhai, Yonghwi Kwon, Kyu Hyung Lee, Xiangyu Zhang, Gabriela Ciocarlie, Ashish Gehani, Vinod Yegneswaran, Dongyan Xu, and Somesh Jha. 2018. Kernel-Supported Cost-Effective Audit Logging for Causality Tracking. In *2018 USENIX Annual Technical Conference (USENIX ATC 18)*. USENIX Association, Boston, MA, 241–254. https://www.usenix.org/conference/atc18/presentation/ma-shiqing

[50] Shiqing Ma, Juan Zhai, Fei Wang, Kyu Hyung Lee, Xiangyu Zhang, and Dongyan Xu. 2017. MPI: Multiple Perspective Attack Investigation with Semantic Aware Execution Partitioning. In *26th USENIX Security Symposium.*

[51] Shiqing Ma, Xiangyu Zhang, and Dongyan Xu. 2016. ProTracer: Towards Practical Provenance Tracing by Alternating Between Logging and Tainting. In *Proceedings of NDSS '16* (San Diego, CA).

[52] Emaad Manzoor, Sadegh M. Milajerdi, and Leman Akoglu. 2016. Fast Memory-Efficient Anomaly Detection in Streaming Heterogeneous Graphs. In *Proceedings of the 22nd ACM SIGKDD International Conference on Knowledge Discovery and Data Mining* (San Francisco, California, USA) *(KDD '16)*. Association for Computing Machinery, New York, NY, USA, 1035–1044. https://doi.org/10.1145/2939672.2939783

[53] P. McDaniel, K. Butler, S. McLaughlin, R. Sion, E. Zadok, and M. Winslett. 2010. Towards a Secure and Efficient System for End-to-End Provenance. In *Proceedings of the 2nd conference on Theory and practice of provenance*. USENIX Association, San Jose, CA, USA.

[54] S. Momeni Milajerdi, R. Gjomemo, B. Eshete, R. Sekar, and V. Venkatakrishnan. 2019. HOLMES: Real-Time APT Detection through Correlation of Suspicious Information Flows. In *2019 2019 IEEE Symposium on Security and Privacy (SP)*. IEEE Computer Society, Los Alamitos, CA, USA. https://doi.org/10.1109/SP.2019.00026

[55] MITRE. 2019. MITRE ATT&CK. https://attack.mitre.org. Last accessed October 16, 2020.

[56] Kiran-Kumar Muniswamy-Reddy, Uri Braun, David A. Holland, Peter Macko, Diana Maclean, Daniel Margo, Margo Seltzer, and Robin Smogor. 2009. Layering in Provenance Systems. In *Proceedings of the 2009 Conference on USENIX Annual Technical Conference* (San Diego, California) *(USENIX'09)*. USENIX Association, Berkeley, CA, USA, 10–10. http://dl.acm.org/citation.cfm?id=1855807.1855817

[57] Capital One. 2019. Information on the Capital One Cyber Incident. https://www.capitalone.com/facts2019/. Last accessed October 16, 2020.

[58] Riccardo Paccagnella, Pubali Datta, Wajih Ul Hassan, Adam Bates, Christopher W. Fletcher, Andrew Miller, and Dave Tian. 2020. Custos: Practical Tamper-Evident Auditing of Operating Systems Using Trusted Execution. In *27th ISOC Network and Distributed System Security Symposium (NDSS'20)*.

[59] Riccardo Paccagnella, Kevin Liao, Dave Tian, and Adam Bates. 2020. Logging to the Danger Zone: Race Condition Attacks and Defenses on System Audit Frameworks. In *Proc. of the ACM Conference on Computer and Communications Security (CCS)*.

[60] Thomas Pasquier, Xueyuan Han, Mark Goldstein, Thomas Moyer, David Eyers, Margo Seltzer, and Jean Bacon. 2017. Practical Whole-system Provenance Capture. In *Proceedings of the 2017 Symposium on Cloud Computing* (Santa Clara, California) *(SoCC '17)*. ACM, New York, NY, USA, 405–418. https://doi.org/10.1145/3127479.3129249

[61] Kexin Pei, Zhongshu Gu, Brendan Saltaformaggio, Shiqing Ma, Fei Wang, Zhiwei Zhang, Luo Si, Xiangyu Zhang, and Dongyan Xu. 2016. HERCULE: Attack Story Reconstruction via Community Discovery on Correlated Log Graph. In *Proceedings of the 32Nd Annual Conference on Computer Security Applications* (Los Angeles, California, USA) *(ACSAC '16)*. ACM, New York, NY, USA, 583–595. https://doi.org/10.1145/2991079.2991122

[62] D.J. Pohly, S. McLaughlin, P. McDaniel, and K. Butler. 2012. Hi-Fi: Collecting High-Fidelity Whole-System Provenance. In *Proceedings of the 2012 Annual Computer Security Applications Conference (ACSAC '12)*. Orlando, FL, USA.

[63] Rapid7. 2018. *WordPress Admin Shell Upload*.

[64] rebootuser. 2019. *LinEnum*.

[65] RedHat. 2019. *Linux Audit*.

[66] Michael Riley, Ben Elgin, Dune Lawrence, and Carol Matlack. 2019. Target Missed Warnings in Epic Hack of Credit Card Data. https://bloom.bg/2KjElxM. Last accessed October 16, 2020.

[67] Bruce Schneier and John Kelsey. 1998. Cryptographic Support for Secure Logs on Untrusted Machines.. In *Proc. of the USENIX Security Symposium (USENIX)*.

[68] Bruce Schneier and John Kelsey. 1999. Secure audit logs to support computer forensics. *ACM Transactions on Information and System Security (TISSEC)* (1999).

[69] Gaurav Shah, Andres Molina, and Matt Blaze. 2006. Keyboards and Covert Channels. In *Proceedings of the 15th Conference on USENIX Security Symposium - Volume 15* (Vancouver, B.C., Canada) *(USENIX-SS'06)*. USENIX Association, USA, Article 5, 17 pages.

[70] Symantec. 2019. About purging reports. https://help.symantec.com/cs/SYMANTECEDR_4.0/EDR/v118097546_v128933990/About-purging-reports?locale=EN_US.

[71] Yutao Tang, Ding Li, Zhichun Li, Mu Zhang, Kangkook Jee, Xusheng Xiao, Zhenyu Wu, Junghwan Rhee, Fengyuan Xu, and Qun Li. 2018. NodeMerge: Template Based Efficient Data Reduction For Big-Data Causality Analysis. In *Proceedings of the 2018 ACM SIGSAC Conference on Computer and Communications Security* (Toronto, Canada) *(CCS '18)*. ACM, New York, NY, USA, 1324–1337. https://doi.org/10.1145/3243734.3243763

[72] Jordan Valinsky. 2020. Clearview AI has billions of our photos. Its entire client list was just stolen. https://www.cnn.com/2020/02/26/tech/clearview-ai-hack/index.html. Last accessed October 16, 2020.

[73] Fei Wang, Yonghwi Kwon, Shiqing Ma, Xiangyu Zhang, and Dongyan Xu. 2018. Lprov: Practical Library-aware Provenance Tracing. In *Proceedings of the 34th Annual Computer Security Applications Conference* (San Juan, PR, USA) *(ACSAC '18)*. ACM, New York, NY, USA, 605–617. https://doi.org/10.1145/3274694.3274751

[74] Qi Wang, Wajih Ul Hassan, Ding Li, Kangkook Jee, Xiao Yu, Kexuan Zou, Junghwan Rhee, Zhengzhang Zhen, Wei Cheng, Carl A. Gunter, and Haifeng chen. 2020. You Are What You Do: Hunting Stealthy Malware via Data Provenance Analysis. In *27th ISOC Network and Distributed System Security Symposium (NDSS'20)*.

[75] Yulai Xie, Dan Feng, Zhipeng Tan, Lei Chen, Kiran-Kumar Muniswamy-Reddy, Yan Li, and Darrell D.E. Long. 2012. A Hybrid Approach for Efficient Provenance Storage. In *Proceedings of the 21st ACM International Conference on Information and Knowledge Management* (Maui, Hawaii, USA) *(CIKM '12)*.

[76] Yulai Xie, Kiran-Kumar Muniswamy-Reddy, Dan Feng, Yan Li, and Darrell D. E. Long. 2013. Evaluation of a Hybrid Approach for Efficient Provenance Storage. *Trans. Storage* 9, 4, Article 14 (Nov. 2013), 29 pages. https://doi.org/10.1145/2501986

[77] Zhang Xu, Zhenyu Wu, Zhichun Li, Kangkook Jee, Junghwan Rhee, Xusheng Xiao, Fengyuan Xu, Haining Wang, and Guofei Jiang. 2016. High Fidelity Data Reduction for Big Data Security Dependency Analyses. In *Proceedings of the 2016 ACM SIGSAC Conference on Computer and Communications Security* (Vienna, Austria) *(CCS '16)*. ACM, New York, NY, USA, 504–516. https://doi.org/10.1145/2976749.2978378

[78] Attila Altay Yavuz and Peng Ning. 2009. BAF: An efficient publicly verifiable secure audit logging scheme for distributed systems. In *Proc. of the Annual Computer Security Applications Conference (ACSAC)*.

[79] Attila A Yavuz, Peng Ning, and Michael K Reiter. 2012. Efficient, compromise resilient and append-only cryptographic schemes for secure audit logging. In *Proc. of the International Conference on Financial Cryptography and Data Security (FC)*.