# Measuring System Auditing Overheads in Real-Time Systems

Anant Kandikuppa
University of Illinois at Urbana-Champaign
{anantk3}@illinois.edu

## Abstract

Securing Cyber-Physical and Real-Time systems is a growing concern as these systems interact with the physical environment. Off the shelf system auditing solutions like the Linux Audit framework, introduce significant overheads in terms of latency and storage, which are undesirable in this domain, which has strict task deadlines and resource constraints.

This work presents *Ellipsis*, a solution that reduces costs associated with system auditing in these time-sensitive systems, by exploiting the predictability in their task model to drastically cut down audit log generation without losing information valuable for forensic investigations. We evaluate the effectiveness of our solution using ArduPilot, an autopilot application suite, observing a **91%** reduction in storage costs while meeting temporal and auditing requirements of the application. *Ellipsis* provides a way forward for auditing CPS.

## 1 Introduction

## 2 Background

### 2.1 Cyber Physical and Real-Time Systems

### 2.2 System Causality Analysis

### 2.3 Linux Audit Framework

## 3 Measurements

Give context about benchmarking applications and why we chose them and the audit rules and stuff

### 3.1 Buffer Utilization

**Experiment:**

For measuring the utilization of the kaudit buffer, we periodically sampled the buffer state every 2 seconds using the audit command line utility *auditctl*, while executing the ArduPilot application for 100K iterations.

**Observations:** From Figure 2, we see that for Linux Audit, the utilization of the kaudit buffer rises quickly in the beginning and remains close to 100% for the majority of the running time, resulting in loss of audit messages.

**Discussion:** When the kaudit buffer is full, new audit messages are lost; hence, to ensure that suspicious events are recorded, it is essential that *the buffer is never full*. The variations that we see in the plots can be attributed to the scheduling of the non real-time *kauditd* thread that is responsible for sending the outstanding audit messages to user-space for retention on disk. We observe that the backlog builds with time when *kauditd* isn't scheduled and drops sharply when *kauditd* eventually gets CPU time.

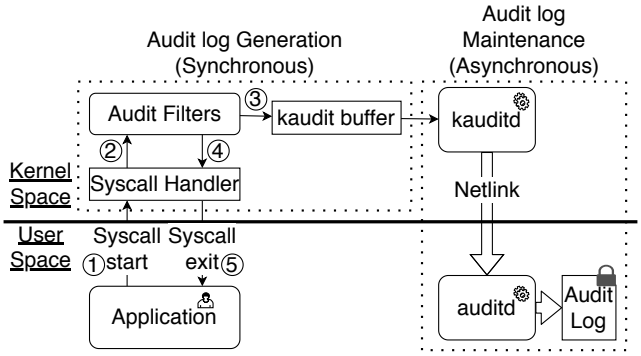


**Figure 1.** Architecture of Linux Audit Framework. Audit logs are generated using auditing hooks in kernel's system call (Syscall) handler and temporarily stored in the kaudit buffer. Log maintenance is handled by two background daemons, kauditd and auditd.

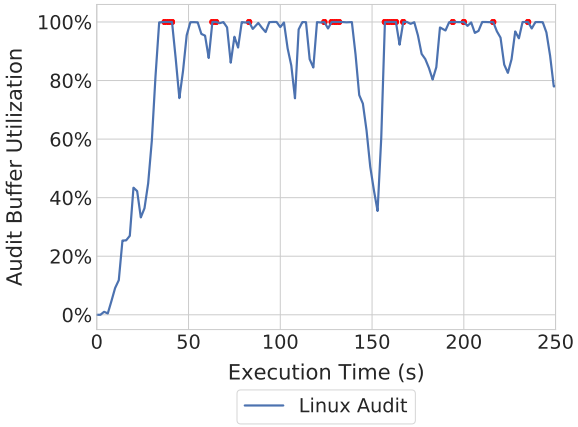


**Figure 2.** Kaudit buffer utilization over time for Linux Audit. Linux Audit's `backlog` value was measured every 2 seconds during the execution of the application and plotted as a percentage of the max backlog limit. Additional red annotations signify all times where buffer utilization was 100% and hence audit events might be lost. At these points Linux Audit performance is actually worse than depicted, but was constrained by the alotted buffer space.

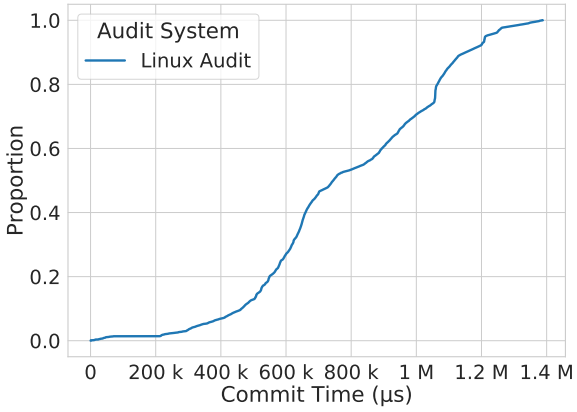


**Figure 3.** Audit commit latencies over time for Linux Audit plotted as a Cumulative Distribution Function (CDF). The experiment measures the time between the generation of an audit message and before the message is written to disk in userspace by the *auditd* thread. For a chosen commit time $t$ on the x-axis, the y-axis shows the fraction of the total messages that encountered delay of less than $t$ microseconds.

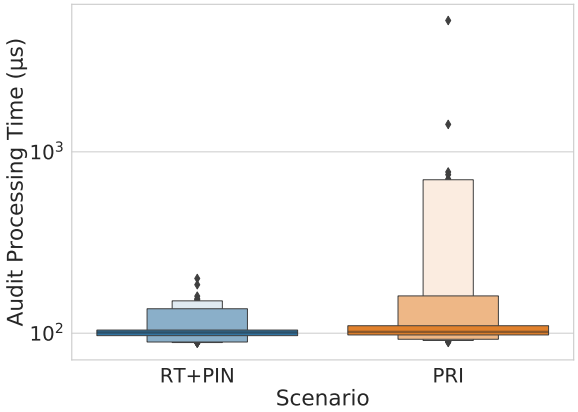


**Figure 4.** Audit Processing Time .

As *kauditd* is a non real-time thread running with no priority, it contends with multiple background processes for CPU time resulting in high buildup of messages in the kaudit buffer. Reducing the pressure of incoming audit messages using a kernel based log reduction scheme like KCAL [18] or increasing the rate of draining the kaudit buffer could be suitable approaches to ensure lossless auditing in resource constrained systems.

### 3.2 Audit Latency

**Experiment:**

For measuring the amount of time each audit message spends in the audit subsystem, the ArduPilot application was executed over 10K iterations and the audit latency was computed from the commit time and the audit time of each audit message obtained from the log file. The non real time *kauditd* and *auditd* threads are assigned a nice value of 0 and -4 respectively.

**Observations:** From Figure 3, we see that the audit latencies across 140k audit messages has a very wide range from 130 microseconds to 1.38 seconds with a median of $\tilde{7}40$ ms.

**Discussion:** The wide range of audit latencies points us to the fact that vanilla Linux Audit maybe a poor choice for real time systems, which typically demand predictable and tightly bound temporal performance guarantees. The low observed minimum audit latency suggests that the implementation of the audit subsystem itself may not entirely to blame for the high *kauditd* buffer utilization and reinforces our earlier observation that scheduling of audit threads is the primary reason for the long tail in audit latencies.

As the audit threads execute at lower non real time priorities, they rely on adequate application slack time to complete execution. With the introduction of synchronous and asynchronous overheads of auditing, the effective slack time reduces further taking away CPU time from the audit subsystem. Reliably accounting for these auditing overheads and increasing priorities of audit threads could help design a schedulable audited RT system and reign in the tail audit latencies.

### 3.3 Audit Throughput

**Experiment:**

For measuring the amount of time required to process a single audit message by the audit subsystem only, we ran a microbenchmark application that consisted of 10 *getpid* system calls and observed the intervals at which audit messages were written to disk over 10 runs. In the RT+PIN scenario, both audit threads were assigned a low real time priority and the *auditd* thread was pinned to core 2 on the machine. As *kauditd* is a kernel thread, we weren't able to pin the thread to a fixed core on the system, instead relying on its real time priority to ensure that it ran ahead of any background process. Whereas in the PRI scenario, the priorities of the audit threads were increased by modifying their nice values, which resembles the default configuration of the audit subsystem.

**Observations:** From Figure 4, we observe that for the PRI case, the interval for processing audit messages varies from 88 $\mu$s to 5ms. Introducing real time priorities and core pinning reduces the variance in measurements and reduces the maximum interval to 200 $\mu$s.

**Discussion:** The order of magnitude reduction in time required to process an audit message corresponding to a system call event highlights the competition for prcocessor time in the system. Assigning a low real time priority to both threads and pinning the userspace *auditd* thread to a core, reduces this competition and allows the audit subsytem to receive sustained processor time, thereby eliminating delays

due to preemption and context switching.
A naive approach to lossless auditing could be to introduce an additional core in the system dedicated to auditing. If the real time application has enough slack to process audit logs generated in a hyperperiod, we can assign a low real-time priority to both audit threads to avoid additional hardware resources. The estimates for audit processing times obtained from the experiment can help arrive at a lower bound on the available slack per hyperperiod. For real time systems which undergo rigorous schedulability analysis before deployment, choosing either apporach can be a conscious decision based on the timing profile of the system.

### 3.4 Summary of Results

Based on the measurements, we find that naively enabling auditing on a real-time system can have serious implications on the completeness of audit logs. Log reduction techniques can help us ensure completeness of audit logs while accounting for synchronous and asynchronous overheads introduced by auditing frameworks can help reliably schedule real-time tasks.

## 4 Discussion

### 4.1 Audit Log Reduction

Our findings show that reducing the rate of incoming audit messages is an important factor in achieving lossless audit logging. Future work can focus on evaluating the applicability of existing kernel based log reduction methods [18, 20] and propose novel solutions that account for the periodic and repetitive nature of real-time tasks to reduce rate of audit log generation, improving kernel buffer utilization and reducing storage costs.

### 4.2 Schedulability Analysis

In order to demonstrate that our estimates for auditing overheads are reasonably accurate, future work can subject our task model to a rigorous schedulability analysis using randomized workloads. Such an analysis would also enable us to compare the relative impact of enhancements to the audit system and understand the tradeoff between temporal integrity and completeness of audit logs.

### 4.3 Log Storage

Our study assumes that audit logs are written to disk, and thus measures the commit time of the audit logs accordingly. Linux Audit supports usage of customizable plugins that enable pushing the logs over the network to a centralized repository for archival and analysis [1]. The network writes may introduce variability in the asynchronous overheads, which would need to be accounted for when deploying a real application.

### 4.4 Applicability to other frameworks

Our study measures the asynchronous performance of the Linux Audit framework. Nevertheless, we find that asynchronous logging mechanisms are ubiquitous and apply to other auditing mechanisms we are aware of as well [5, 17, 20–22]. In fact, block-based I/O operations are always asynchronous within kernels unless explicitly configured for synchronous operation. Still, future work is need to explore the applicabilty of our observations on other platforms.

## 5 Related Work

### 5.1 System Auditing

Due to its value in threat detection and investigation, system auditing is the subject of renewed interest in traditional systems. While a number of experimental audit frameworks have incorporated notions of data provenance [5],[23],[26],[24], the bulk of this work is also based on commodity audit frameworks such as Linux Audit. Techniques have also been proposed to extract threat intelligence from large volumes of log data [9, 11, 14, 25]. In this work, we seek to understand if such techniques are directly applicable to RTS while adhering to the unique temporal and resource constraints of the domain. Our study finds that there is a need for solutions in the RTS space that build upon the notion of execution partitioning of log activity [9, 10, 14, 15, 19], which decomposes long lived applications into autonomous units of work to reduce false dependencies in forensic investigations.

### 5.2 Forensic Reduction

A lot of work has gone into improving the cost-utility ratio of system auditing by pruning, summarizing, or compressing audit data that is unlikely to be used during investigations [3, 4, 6, 8, 12, 16, 17, 27, 31]. Of these, Ma et al.'s KCAL [18] and ProTracer [20] systems are a few that inline their reduction methods into the kernel. Our findings suggest that userspace reduction solutions might not be effective in RTS as resource constraints might cause loss of audit information in the kernel. Although it is unclear if the existing kernel based approaches are suitable for real-time applications, inline kernel log reduction seems to provide a promising path forward to enable efficient system auditing in RTS.

### 5.3 Auditing RTS

Although auditing has been widely acknowledged as an important aspect of securing embedded devices [2, 7, 13], the unique challenges that arise when auditing RTS have received only limited attention. Wang et al. present ProvThings, an auditing framework for monitoring IoT smart home deployments [29], but rather than audit low-level embedded device activity their system monitors API-layer flows on the IoT platform's cloud backend. Tian et al. present a block-layer auditing framework for portable USB storage that can

be used to diagnose integrity violations [28]. Their embedded device emulates a USB flash drive, but does not consider system call auditing of real-time applications. Wu et al. present a network-layer auditing platform that captures the temporal properties of network flows and can thus detect temporal interference [30]. Whereas their system uses auditing to diagnose performance problems in networks, the present study considers the performance problems created by auditing within real-time applications. In contrast to these systems, this study highlights the challenges of RTS auditing by attempting to naively incorporate the vanilla Linux Audit system into the real-time task schedule.

## 6 Conclusion

This project is awesome.

## 7 Metadata

The presentation of the project can be found at:

https://zoom/cloud/link/

The code/data of the project can be found at:

https://github.com/you/repo

## References


[1] audisp-remote(8) - linux man page. https://linux.die.net/man/8/audisp-remote. Last accessed 12-04-2020.

[2] Anderson, M. Securing embedded linux.

[3] Bates, A., Butler, K. R., and Moyer, T. Take only what you need: Leveraging mandatory access control policy to reduce provenance storage costs. In *7th USENIX Workshop on the Theory and Practice of Provenance (TaPP 15)* (Edinburgh, Scotland, July 2015), USENIX Association.

[4] Bates, A., Tian, D., Hernandez, G., Moyer, T., Butler, K. R., and Jaeger, T. Taming the Costs of Trustworthy Provenance through Policy Reduction. *ACM Trans. on Internet Technology 17*, 4 (sep 2017), 34:1–34:21.

[5] Bates, A., Tian, D. J., Butler, K. R., and Moyer, T. Trustworthy whole-system provenance for the linux kernel. In *24th USENIX Security Symposium (USENIX Security 15)* (Washington, D.C., Aug. 2015), USENIX Association, pp. 319–334.

[6] Ben, Y., Han, Y., Cai, N., An, W., and Xu, Z. T-tracker: Compressing system audit log by taint tracking. In *2018 IEEE 24th International Conference on Parallel and Distributed Systems (ICPADS)* (Dec 2018), pp. 1–9.

[7] Day, R., and Slonosky, M. Securing connected embedded devices using built-in rtos security.

[8] Hassan, W. U., Aguse, N., Lemay, M., Moyer, T., and Bates, A. Towards Scalable Cluster Auditing through Grammatical Inference over Provenance Graphs. In *Proceedings of the 25th ISOC Network and Distributed System Security Symposium* (San Diego, CA, USA, February 2018), NDSS'18.

[9] Hassan, W. U., Guo, S., Li, D., Chen, Z., Jee, K., Li, Z., and Bates, A. NoDoze: Combatting Threat Alert Fatigue with Automated Provenance Triage. In *26th ISOC Network and Distributed System Security Symposium* (February 2019), NDSS'19.

[10] Hassan, W. U., Noureddine, M., Datta, P., and Bates, A. OmegaLog: High-Fidelity Attack Investigation via Transparent Multi-layer Log Analysis. In *27th ISOC Network and Distributed System Security Symposium* (February 2020), NDSS'20.

[11] Hossain, M. N., Milajerdi, S. M., Wang, J., Eshete, B., Gjomemo, R., Sekar, R., Stoller, S., and Venkatakrishnan, V. SLEUTH: Real-time attack scenario reconstruction from COTS audit data. In *26th USENIX Security Symposium (USENIX Security 17)* (Vancouver, BC, Aug. 2017), USENIX Association, pp. 487–504.

[12] Hossain, M. N., Wang, J., Sekar, R., and Stoller, S. D. Dependence-preserving data compaction for scalable forensic analysis. In *Proceedings of the 27th USENIX Conference on Security Symposium* (USA, 2018), SEC'18, USENIX Association, pp. 1723–1740.

[13] Kohei, K. Recent security features and issues in embedded systems.

[14] Kwon, Y., Wang, F., Wang, W., Lee, K. H., Lee, W.-C., Ma, S., Zhang, X., Xu, D., Jha, S., Ciocarlie, G., Gehani, A., and Yegneswaran, V. Mci: Modeling-based causality inference in audit logging for attack investigation. In *Proc. of the 25th Network and Distributed System Security Symposium (NDSS'18)* (2018).

[15] Lee, K. H., Zhang, X., and Xu, D. High Accuracy Attack Provenance via Binary-based Execution Partition. In *Proceedings of NDSS '13* (Feb. 2013).

[16] Lee, K. H., Zhang, X., and Xu, D. Loggc: Garbage collecting audit log. In *Proceedings of the 2013 ACM SIGSAC Conference on Computer & Communications Security* (New York, NY, USA, 2013), CCS '13, Association for Computing Machinery, pp. 1005–1016.

[17] Ma, S., Lee, K. H., Kim, C. H., Rhee, J., Zhang, X., and Xu, D. Accurate, low cost and instrumentation-free security audit logging for windows. In *Proceedings of the 31st Annual Computer Security Applications Conference* (New York, NY, USA, 2015), ACSAC 2015, Association for Computing Machinery, pp. 401–410.

[18] Ma, S., Zhai, J., Kwon, Y., Lee, K. H., Zhang, X., Ciocarlie, G., Gehani, A., Yegneswaran, V., Xu, D., and Jha, S. Kernel-supported cost-effective audit logging for causality tracking. In *2018 USENIX Annual Technical Conference (USENIX ATC 18)* (Boston, MA, July 2018), USENIX Association, pp. 241–254.

[19] Ma, S., Zhai, J., Wang, F., Lee, K. H., Zhang, X., and Xu, D. MPI: Multiple Perspective Attack Investigation with Semantic Aware Execution Partitioning. In *26th USENIX Security Symposium* (August 2017).

[20] Ma, S., Zhang, X., and Xu, D. Protracer: Towards practical provenance tracing by alternating between logging and tainting. In *Proceedings of NDSS '16* (2016).

[21] Microsoft. Process monitor v3.60. https://docs.microsoft.com/en-us/sysinternals/downloads/procmon. Last accessed 12-04-2020.

[22] Microsoft. Windows etw framework conceptual tutorial. https://docs.microsoft.com/en-us/windows/win32/wes/windows-event-log. Last accessed 12-04-2020.

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

[24] Pasquier, T. F. J. ., Singh, J., Eyers, D., and Bacon, J. Camflow: Managed data-sharing for cloud services. *IEEE Transactions on Cloud Computing 5*, 3 (2017), 472–484.

[25] Pei, K., Gu, Z., Saltaformaggio, B., Ma, S., Wang, F., Zhang, Z., Si, L., Zhang, X., and Xu, D. Hercule: Attack story reconstruction via community discovery on correlated log graph. In *Proceedings of the 32nd Annual Conference on Computer Security Applications* (New York, NY, USA, 2016), ACSAC '16, Association for Computing Machinery, pp. 583–595.

[26] Pohly, D. J., McLaughlin, S., McDaniel, P., and Butler, K. Hi-fi: Collecting high-fidelity whole-system provenance. In *Proceedings of the 28th Annual Computer Security Applications Conference* (New York, NY, USA, 2012), ACSAC '12, Association for Computing Machinery, pp. 259–268.

[27] Tang, Y., Li, D., Li, Z., Zhang, M., Jee, K., Xiao, X., Wu, Z., Rhee, J., Xu, F., and Li, Q. Nodemerge: Template based efficient data reduction for big-data causality analysis. In *Proceedings of the 2018 ACM SIGSAC*

*Conference on Computer and Communications Security* (New York, NY, USA, 2018), CCS '18, Association for Computing Machinery, pp. 1324–1337.

[28] Tian, D. J., Bates, A., Butler, K. R. B., and Rangaswami, R. Provusb: Block-level provenance-based data protection for usb storage devices. In *Proceedings of the 2016 ACM SIGSAC Conference on Computer and Communications Security* (New York, NY, USA, Oct 2016), CCS '16, ACM.

[29] Wang, Q., Hassan, W. U., Bates, A., and Gunter, C. Fear and Logging in the Internet of Things. In *Proceedings of the 25th ISOC Network and Distributed System Security Symposium* (February 2017), NDSS'18.

[30] Wu, Y., Chen, A., and Phan, L. T. X. Zeno: Diagnosing performance problems with temporal provenance. In *16th USENIX Symposium on Networked Systems Design and Implementation (NSDI 19)* (Boston, MA, 2019), USENIX Association, pp. 395–420.

[31] Xu, Z., Wu, Z., Li, Z., Jee, K., Rhee, J., Xiao, X., Xu, F., Wang, H., and Jiang, G. High fidelity data reduction for big data security dependency analyses. In *Proceedings of the 2016 ACM SIGSAC Conference on Computer and Communications Security* (New York, NY, USA, 2016), CCS '16, Association for Computing Machinery, pp. 504–516.