Take Only What You Need: Leveraging Mandatory Access Control Policy to Reduce Provenance Storage Costs

Adam Bates, Kevin R.B. Butler
University of Florida
{adammbates,butler}@ufl.edu

Thomas Moyer
MIT Lincoln Laboratory
thomas.moyer@ll.mit.edu

Abstract

When performing automatic provenance collection within the operating system, inevitable storage overheads are made worse by the fact that much of the generated lineage is *uninteresting*, describing noise and background activities that lie outside the scope the system's intended use. In this work, we propose a novel approach to policy-based provenance pruning – leverage the confinement properties provided by Mandatory Access Control (MAC) systems in order to identify subdomains of system activity for which to collect provenance. We consider the assurances of *completeness* that such a system could provide by sketching algorithms that reconcile provenance graphs with the information flows permitted by the MAC policy. We go on to identify the design challenges in implementing such a mechanism. In a simplified experiment, we demonstrate that adding a policy component to the Hi-Fi provenance monitor could reduce storage overhead by as much as 82%. To our knowledge, this is the first practical policy-based provenance monitor to be proposed in the literature.

1. Introduction

Provenance-aware systems offer unprecedented insight into the workings of computing systems, but retaining provenance demands considerable storage space. Braun et al. identify excessive storage overhead as a fundamental challenge to automatic provenance collection [3]. While subsequent work has leveraged compression techniques to reduce storage burdens [18–20], compression fails to address one of the root causes of storage overhead; namely, that much of the provenance we collect is simply not useful. Even when compressed, this *uninteresting* provenance will continue to occupy space indefinitely. To date, this has been an unfortunate concession in provenance-aware systems; in order to assure the *completeness* of the provenance description of interesting system activities, we must conservatively collect *everything*.

The Lincoln Laboratory portion of this work was sponsored by the Assistant Secretary of Defense for Research & Engineering under Air Force Contract #FA8721-05-C-0002. Opinions, interpretations, conclusions and recommendations are those of the author and are not necessarily endorsed by the United States Government.

TaPP 2015, July 8–9, 2015, Edinburgh, Scotland.

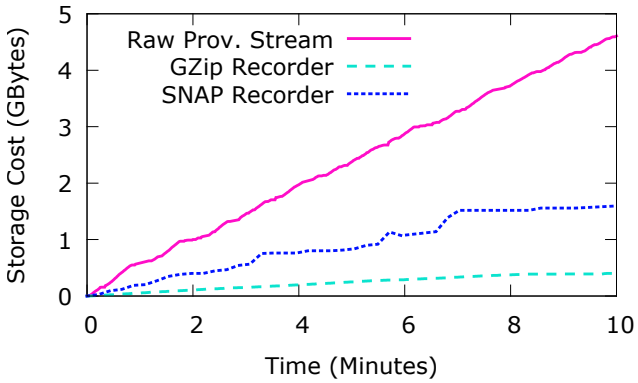


Figure 1: Growth of Hi-Fi's provenance log during kernel compilation using different LPM storage backends. Log growth is only manageable when compressed with `gzip`, but in this form it cannot be efficiently queried.

We consider the matter of provenance collection for Mandatory Access Control (MAC) enabled systems, where a statically defined policy dictates where system objects are permitted to flow. MAC policies encode potentially invaluable information for the purposes of provenance collection, stating explicitly that certain system objects will never interact. It follows that a MAC-aware provenance monitor could conceivably avoid the recording of uninteresting provenance, instead taking only what it needs to describe a particular application or set of applications.

We describe the requirements of such a provenance monitor in this work, analyzing a system's existing security contexts in order to record *complete* provenance over a *subset* of system operations. We call our system *Provenance Walls*, named after our method of MAC policy analysis that extends Vijayakumar et. al.'s *Integrity Walls* [16] project. Provenance Walls leverages the confinement properties of the SELinux Multi-Level Security (MLS) policy to record a complete provenance history within an individual application's Trusted Computing Base (TCB). We propose an architecture for Provenance Walls based on the Linux Provenance Modules (LPM) framework [2]. By performing a proof-of-concept experiment by making minimal modifications to the Hi-Fi system [15], we show that our approach can reduce storage overheads by as much as 81%. Provenance Walls thus stands to dramatically improve the cost-benefit ratio of provenance collection.

2. Background

A fundamental issue in automated provenance collection is the incurred storage overhead. Provenance can quickly grow to dwarf

the data it describes. This is especially the case with fine-grained, system-level provenance collection mechanisms such as Hi-Fi [15] and PASS [14]. Even worse, such systems are prone to collecting information that is completely uninteresting or irrelevant to the system's purpose [3], making it harder to justify the incurred costs.

To demonstrate, we plotted the growth of the provenance store for LPM's Hi-Fi module during a kernel compilation benchmark. In LPM, a recorder daemon in user space processes and stores a kernel relay provenance stream [2, 15]. The results are shown in Figure 1. We first used Zlib to write provenance to a compressed file that occupied only 450 MB of storage; however, the provenance could not be easily queried in this form. Without compression, the same routine generated almost 5 GB of provenance. We attempted to provide a better balance between storage and performance by replacing the GZip daemon with one that wrote provenance to an in-memory graph database using the SNAP library [10]. This provided highly efficient querying, as well as some compression by avoiding the creation of redundant graph components. We approximated the storage overhead through polling the virtual memory consumed by the daemon process in the `/proc` file system. However, the SNAP graph still grew to 1.64 GB in size. This problem is not isolated to Hi-Fi; PASSv2 reports similar storage overheads, about 1.28 GB, in its own kernel compilation benchmark [13].

This motivates the need for *provenance pruning* [3] — schemes that selectively remove provenance to save space, while also maintaining sufficient information for the provenance to be reconstructed efficiently and completely. Web compression and deduplication have been employed to reach storage reduction ratios of 3.31:1 [20], and the "Web+Dictionary" technique further improves the compression ratio up to approximately 5:1 [18, 19].

While promising, these approaches fail to address a root cause of storage overhead – when performing system-wide provenance capture, space is inevitably wasted by the provenance of background activities and other system noise (e.g., `crond`). The behavior of such activities is well understood, and can be secured through traditional type enforcement. This provenance is therefore unlikely to be interesting to an administrator, but if collected it will continue to occupy space even after compression. The notion of *interesting* provenance is often domain-specific, leading Braun et. al. to call for a policy-driven method of provenance pruning [3], but to the best of our knowledge a policy-based collection mechanism has yet to appear in the literature. We propose an approach to policy-based pruning in this work.

3. Provenance Walls

In this section, we propose an approach to policy-based provenance pruning that is implemented in the provenance collection mechanism itself. We call this approach *Provenance Walls*, as it leverages a method of policy generation that extends Vijayakumar et al.'s Integrity Walls project [16]. Below, we describe the challenges associated with designing a policy for the selective collection of provenance.

Suppose an administrator was interested in the provenance of just a single application on a system, such as the Apache web server (`httpd`). Let us assume that this administrator has access to a policy-based provenance monitor, allowing them to describe the system events for which they would like to collect provenance. What would the administrator include in the policy? A naïve approach would be to specify a policy to only generate provenance for the operations performed by `httpd`. The immediate problem that such an approach poses is that the provenance history will not be complete. First, this provenance will not include the actions of helper processes (e.g., `htaccess`) that inform the execution of `httpd`. Even if the administrator expanded the policy to include these helper processes, they cannot account for the possibility that other objects on the system will flow into `httpd`, causing its provenance graph to be incomplete. An incomplete provenance graph cannot provide a full explanation as to the ancestry of objects or the events that have taken place on a system. This diminishes the value of the graph in benign environments, and renders it useless in malicious environments where an adversary could operate covertly in the unmonitored space.

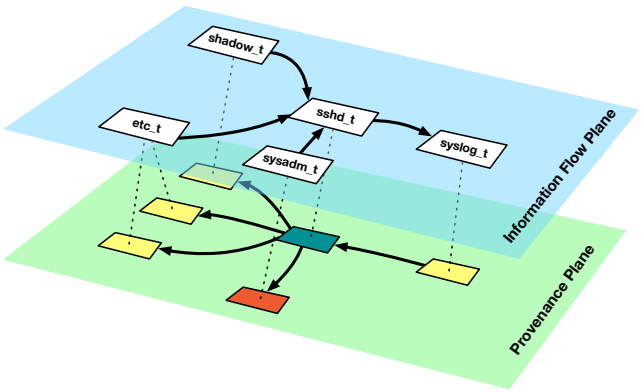
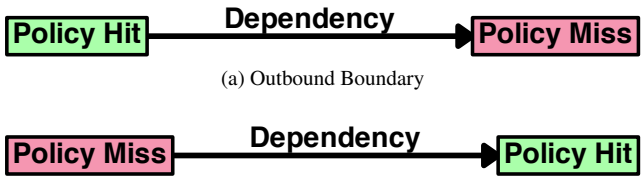


Figure 2: The intuition to our approach is that provenance objects and relations can be mapped to mandatory access control labels and transitions, forming an *Information Flow* overlay to the provenance graph. In the information flow plane, edges encode permissible future actions (e.g., *MayWrite*), whereas in the provenance plane edges encode historic events (e.g., *WasGeneratedBy*).

We propose that Mandatory Access Control provides an environment for the selective recording of provenance while preserving completeness properties. In a MAC-enabled system, every system object is assigned a security label, and a policy dictates the permissible interactions between different labels. Many applications, including Apache, publish policy modules that define application-specific labels and access controls. We observe that such a policy, like provenance, can be expressed in graph form. Provenance graphs encode the history of *actual flows* between objects during a system's execution, while MAC policy graphs encode the *permissible flows* between objects during system execution. Additionally, security labels provide a complete mapping between the provenance namespace and the MAC namespace. We can therefore imagine a MAC policy as an overlay to the provenance graph, shown in Figure 2. Pictured in the simplified example is a traditional provenance graph that has used 3 files in order to generate 1 file. In the Information Flow Plane, we can see that these objects map to security labels, and that all of the operations in the provenance plane are also described in the MAC policy. While the mapping is not 1:1 (e.g., `etc_t`), all events described in the provenance history will have been previously authorized in the MAC policy.

Given the nature of this overlay, we can define a provenance policy in terms of security labels in the MAC namespace. In Provenance Walls, the provenance policy is defined as a list of security contexts, such that the new provenance record will be created if and only if one or more of the objects associated with the event has a security context that appears in the provenance policy. However, In order for this approach to be effective, we require a means of identifying subsets of security labels that account for all flows in a given subdomain of system operation. For convenience, we will describe this property as ***selective completeness*** – the ability for the provenance monitor to produce a complete description of a given system subdomain in perpetuity. Next, we describe how to identify sets of security labels that provide such an assurance.



(a) Outbound Boundary

(b) Inbound Boundary

Figure 3: To preserve selective completeness, Provenance Walls must record system events where a tracked system object interacts with an untracked object. This introduces *boundaries* in the provenance graph, where collection halts at a relation between one node within the policy (Policy Hit) and one node outside of the policy (Policy Miss). In these cases, we must provide an explanation to the user as to why provenance collection halted after the boundary.

3.1 Policy Composition

How can we define a policy that provides the *selective completeness* property? Because provenance must offer a complete history of the data it describes, selective provenance collection is a difficult problem. Failure to collect all of the necessary information could result in gaps in a provenance history, rendering it unfit for later use. However, if the provenance policy is carefully constructed to leverage MAC separation of domains, then it will have a formal assurance that no object *outside* of the policy can flow to an object that matches the policy.

For further guidance, we can turn to past work on the static analysis of MAC policy. We extend Vijayakumar et. al.'s *Integrity Walls* method of mining SELinux policies to identify an application's Minimal Trusted Computing Base (MTCB) [16]. Integrity Walls builds an MTCB for a subject application s by partitioning a system policy P into a set of trusted labels I_s and an untrusted set O_s. Given a base SELinux policy, an application policy module, and its dependency modules, Integrity Walls identifies the following groups of labels: *executable writers* that have permission to write s's executable file, *kernel subjects* with permission to write s's underlying kernel objects, and *helper subjects*, which are distinct applications whose subject labels appear in s's policy module and are trusted by s. Together, these labels make up I_s and form a trusted computing base of the subject application of s.

Because the label set I_s provides a complete description of system objects that can flow into the subject application s, I_s can be used as a provenance policy that provides selective completeness. Below, we imagine the existence of a fully implemented version of Provenance Walls, and identify the properties that this system would need to be able to provably demonstrate to the user.

3.2 System Properties

We now consider several additional capabilities that the Provenance Walls system would need to provide in order to be useful. The methodology described above is sufficient to provide the selective completeness property; however, Provenance Walls will still produce an incomplete graph of system execution. In practice, it will be necessary for the monitor to be able to produce "explanations" of the content of the provenance graph. The below explanations can be produced by Provenance Walls using the MAC policy, the provenance policy, and the actual provenance log.

Why is this subgraph missing?

Provenance Walls introduces *boundaries* in the provenance graph, represented by a relation between two nodes in which one node falls within the provenance policy while the other node does not. In analyzing the graph, users may require an explanation as to why a boundary exists. There are two kinds of boundaries that need be considered. Shown in Figure 3, Provenance Walls records the "next hop" for all objects within the policy, creating the potential for outbound and inbound boundaries. The outbound boundary (Figure 3a) implies that an object within the policy is dependent on an object that falls outside of the policy. In fact, such a boundary cannot exist in our graph due to the way in which I_s is constructed; the labels for all objects that can flow into application s are necessarily a part of s's MTCB, and therefore will be included in the graph. The inbound boundary (Figure 3b) represents a flow that has left the MTCB, and is therefore possible. In this case, it is sufficient to demonstrate to the user that this missing subgraph represents a relation from a member of I_s to a member of O_s. If the validity of the provenance graph is in question, one could further perform reachability analysis to prove that the omitted subgraph cannot flow back into the MTCB.

Where can this data go?

Our approach to policy generation assures that no system object outside of I_s can flow into I_s; however, data produced by members I_s may flow to other parts of the system, e.g., into the network. Unfortunately, Provenance Walls does not generate provenance for these activities. However, through analysis of the MAC policy, Provenance Walls may still provide an explanation as to what will happen to such data. When the user comes across an inbound boundary as shown in Figure 3b, they could request an explanation of all the security labels that a given provenance object might flow to. Provenance Walls would then return this subgraph of the MAC policy for the user's review. The user may be surprised to find that a particular object in the provenance graph can flow to certain security labels, e.g., can be sent out over the network, facilitating iterative a re-configuration of the MAC policy.

4. Challenges

In this section, we sketch an architecture that could provide the capabilities described in Section 3. Our system will interoperate with the Linux reference monitor provided by Linux Security Modules (LSM) [17]. While our approach could be generalized to any LSM, we choose SELinux because it is ubiquitous on Linux systems and its MLS policy has been extensively analyzed in the literature [8]. Below, we consider several of the most critical design decisions for the Provenance Walls architecture, and argue the validity of our choices.

Accessing Security Contexts. The most fundamental challenge in Provenance Walls is that of creating a provenance collection mechanism that is fully aware of MAC enforcement. While some of a system's security labels can be accessed by privileged processes in user space (e.g., viewing the security labels on files with `getfattr`), other context is opaque to user space (e.g., a process transitioning to a different privilege state). Because of this, our collection mechanism must reside within the kernel. It is not necessary to integrate our collection mechanism into the reference monitor itself due to the fact that all kernel modules have unrestricted access to kernel memory. Within the kernel, security contexts are embedded within the struct of all major kernel objects as a `void` pointer. Our collection mechanism can interpret these fields by including the security struct declarations of the active LSM. For example, for SELinux, our collection mechanism would need to include `objsec.h`.

Provenance Collection Mechanism. Provenance Walls would be most effective as a lightweight retrofit of an existing provenance-

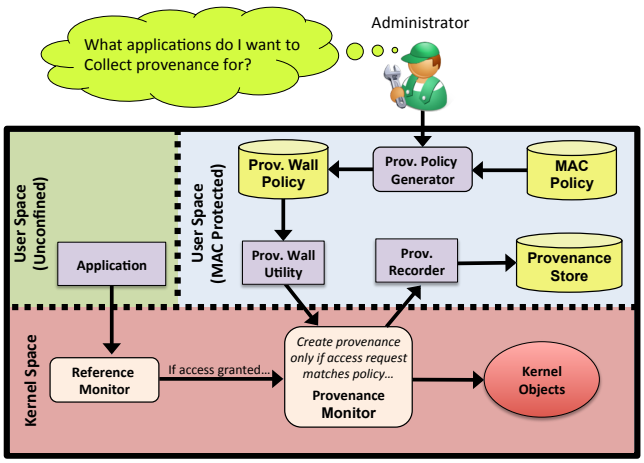


Figure 4: Design of the Provenance Walls Architecture.

aware system. This approach is the simplest, and also the most reliable, as creating an OS layer provenance monitor is a time consuming and error prone process. The two most viable candidates to serve as the foundation of our system are PASS [13] and Hi-Fi [15]; SPADE is not an option because it does not run in kernel space [7]. For our system, Hi-Fi is the superior choice because it is already implemented as an LSM. This will provide tighter integration between the reference monitor and our collection mechanism. More specifically, we will use the LPM port of the Hi-Fi system. Use of LPM's Hi-Fi port permits permits us to enable SELinux without relying on module stacking [5] while still providing complete mediation guarantees [2].

Early Boot Provenance. To provide early boot security, the Linux kernel loads and registers LSM prior to the virtual file system layer or user space. As a consequence, during the early boot, our collection mechanism will not have access to the provenance policy. Pohly et al. face a similar problem in [15], in which Hi-Fi's user space daemon cannot be loaded until late in the boot process. They solve this problem by storing early boot provenance in a separate boot buffer, then registering a callback function to flush the boot buffer once the VFS had been initialized. We can extend this solution to account for the fact that our system will not have immediate access to the provenance policy.

Early in the boot, our system will conservatively create provenance records for each system event, storing them in the buffer along with the associated security identifiers (SIDs) of the event. Each SID is only 32 bits long, and SELinux provides tools for translating SIDs into character string security labels. Once Provenance Walls is notified that the policy is loaded, it can replay the buffer and perform policy checks of each of the associated SIDs, sending events to the recorder only if they match the policy. Provenance Walls may need to increase the early boot buffer size compared to Hi-Fi's because of the space required to record security contexts. However, Hi-Fi's buffer was only 1 KB in size [15], and recording SIDs will require at most 100 bits per event, so we are confident that the increase in memory overhead will be reasonable.

Overview. A full overview of the Provenance Walls design is pictured in Figure 4. Like Hi-Fi, Provenance Walls places a *monitor* in kernel space. The collected provenance is communicated over a kernel relay to a *recorder* that interprets the byte stream and records the provenance. Provenance Walls also introduces several new components. The first is a *policy generator* that combines administrator input with static analysis of an SELinux *MAC policy* in order to synthesize a *provenance policy*. This policy is transmitted to kernel space using the `securityfs` virtual file system. Once the policy is loaded, the monitor begins to selectively create provenance according to the policy.

5. Storage Reduction Test

We conducted a basic experiment to gain a preliminary sense of the potential storage savings with Provenance Walls. Our testbed was a modestly provisioned Linux VM that was running both the SELinux LSM as well as a modified copy of the Hi-Fi LPM. In this test, we imagined a scenario in which we did not wish to collect provenance for kernel compilations due to the large amount of provenance that is produced by this task.

We made the following minimal changes to Hi-Fi: first, we included the header file that defines the SELinux object security structs, as well as the header file that defines a set of functions for performing SID to context translation; second, we hardcoded a simple policy check to see if the event's associated SIDs mapped to the `user_t` type. This label is associated with the user's home directory. We then booted into the system and ran a kernel compilation benchmark from within our home directory under two conditions. In the first condition (*tracked*), events with the `user_t` type were recorded, and the second condition (*untracked*) they were ignored. This allowed us to filter the provenance of the kernel compilation task from the second condition's provenance log.

The tracked condition's compressed provenance log was 54 MB in size, while the untracked condition's compressed provenance log was just 10 MB, representing an 82% reduction in storage overhead. This test serves as anecdotal evidence of the potential savings using our approach. By dramatically reducing the size of the provenance graph, Provenance Walls would also improve the performance of subsequent queries on the graph. Of course, we note that this is just one example, and not a complete evaluation. The level of storage reduction will vary depending on the policy and the amount of system activity that can be filtered.

6. Related Work

Provenance Walls is a MAC-aware implementation of a *provenance monitor* [12]. McDaniel et al. define a provenance monitor as a provenance collection mechanism that provides the reference monitor guarantees laid out by Anderson [1]: complete mediation, tamperproofness, and verifiability. In other words, a provenance monitor is a recording instrument that provably captures complete provenance in a manner and is secure against attack or circumvention. Pohly et al.'s Hi-Fi system is a security module that collects *whole-system provenance* [15], detailing the actions of processes, IPC mechanisms, and even the kernel itself (which does not exclusively use system calls). Hi-Fi provides a partial implementation of a provenance monitor [15], but its reference monitor guarantees do not apply to layered or distributed provenance, nor is a secure deployment demonstrated in which Hi-Fi can self-protect from attack. Bates et al. introduce Linux Provenance Modules (LPM), a general-purpose framework for the design of provenance monitors [2]; LPM includes a port and extension of Hi-Fi that addresses these shortcomings. We intend to build Provenance Walls as a new LPM provenance module.

The majority of provenance-aware systems proposed in the literature were designed for benign environments [4], and are therefore not provenance monitors. Lyle and Martin sketch the design for a secure provenance monitor based on trusted computing [11]. However, they conceptualize provenance as a TPM-aided proof of code execution, overlooking interprocess communication and other system activity that could inform execution results, offering coarse-

grained metadata compared to other systems such as PASS [13, 14] and SPADE [6, 7].

7. Future Work

This work marks just the beginning of our exploration of the interactions between provenance and security mechanisms. The complex architecture sketched in Section 4 must first be developed and implemented, including an exhaustive analysis of the system's properties and a formal demonstration of *selective completeness* that integrates policy and implementation. We are just now beginning to implement our LPM-based provenance module. Because the storage reductions provided by Provenance Walls are domain specific, we also intend to characterize the classes of applications for which our approach is most beneficial.

Finally, we have uncovered in this work that selectively recording provenance creates a tradeoff between cost and flexibility. Whereas the metadata collected by systems such as PASS and Hi-Fi are general enough to support a variety of uses, our approach omits provenance that is critical to certain environments. For example, because Provenance Walls does not track flows that have left an application's MTCB, it is ill suited for performing forensics related to data exfiltration [9]. As part of our upcoming study, we hope to better characterize this tradeoff. Additionally, we will investigate other methods of policy generation that can provide selective completeness properties.

8. Conclusion

The inevitable storage overheads associated with automatic provenance collection are made worse by the fact that much of the generated lineage is *uninteresting*, describing system noise and background activities that lie outside the scope of user interest. In this work, we have proposed a new method of policy-based provenance pruning. We sketched the design a policy-based provenance collection mechanism, and proposed a means of demonstrating the *completeness* property through reconciling the provenance log with MAC information flows. Using a minimally modified copy of the Hi-Fi monitor, we conducted a simplified experiment to discover that our approach can reduce storage overheads by as much as 82%. Provenance Walls thus promises to significantly decrease the overheads associated with provenance collection.

Acknowledgements

We would like to thank Rob Cunningham, Alin Dobra, Patrick McDaniel, Daniela Oliveira, Nabil Schear, and Patrick Traynor for their valuable comments and insight, as well as Devin Pohly for his sustained assistance in working with Hi-Fi. This work was supported in part by the US National Science Foundation under grant numbers CNS-1118046, CNS-1254198, and CNS-1445983.

References

[1] J. P. Anderson. Computer security technology planning study. Technical Report ESD-TR-73-51, Air Force Electronic Systems Division, Hanscom AFB, Bedford, MA, Oct. 1972.

[2] A. Bates, D. Tian, K. R. Butler, and T. Moyer. Trustworthy Whole-System Provenance for the Linux Kernel. In *Proceedings of 24th USENIX Security Symposium on USENIX Security Symposium*, Aug. 2015.

[3] U. Braun, S. L. Garfinkel, D. A. Holland, K.-K. Muniswamy-Reddy, and M. I. Seltzer. Issues in Automatic Provenance Collection. In L. Moreau and I. T. Foster, editors, *International Provenance and Annotation Workshop*, volume 4145 of *Lecture Notes in Computer Science*, pages 171–183. Springer, 2006.

[4] L. Carata, S. Akoush, N. Balakrishnan, T. Bytheway, R. Sohan, M. Seltzer, and A. Hopper. A primer on provenance. *Commun. ACM*, 57(5):52–60, May 2014.

[5] J. Edge. Another LSM stacking approach. `https://lwn.net/Articles/518345/`, Oct. 2012.

[6] A. Gehani and U. Lindqvist. Bonsai: Balanced Lineage Authentication. In *Proceedings of the 23rd Annual Computer Security Applications Conference*, ACSAC'07, Dec 2007.

[7] A. Gehani and D. Tariq. SPADE: Support for Provenance Auditing in Distributed Environments. In *Proceedings of the 13th International Middleware Conference*, Middleware '12, Dec 2012.

[8] B. Hicks, S. Rueda, L. St.Clair, T. Jaeger, and P. McDaniel. A Logical Specification and Analysis for SELinux MLS Policy. *ACM Trans. Inf. Syst. Secur.*, 13(3):26:1–26:31, July 2010.

[9] S. N. Jones, C. R. Strong, D. D. E. Long, and E. L. Miller. Tracking Emigrant Data via Transient Provenance. In *3rd Workshop on the Theory and Practice of Provenance*, TAPP'11, June 2011.

[10] J. Leskovec and R. Sosič. SNAP: A general purpose network analysis and graph mining library in C++. `http://snap.stanford.edu/snap`, June 2014.

[11] J. Lyle and A. Martin. Trusted Computing and Provenance: Better Together. In *2nd Workshop on the Theory and Practice of Provenance*, TaPP'10, Feb. 2010.

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

[13] K.-K. Muniswamy-Reddy, U. Braun, D. A. Holland, P. Macko, D. Maclean, D. Margo, M. Seltzer, and R. Smogor. Layering in Provenance Systems. In *Proceedings of the 2009 Conference on USENIX Annual Technical Conference*, ATC'09, June 2009.

[14] K.-K. Muniswamy-Reddy, D. A. Holland, U. Braun, and M. Seltzer. Provenance-aware Storage Systems. In *Proceedings of the Annual Conference on USENIX '06 Annual Technical Conference*, Proceedings of the 2006 Conference on USENIX Annual Technical Conference, June 2006.

[15] D. Pohly, S. McLaughlin, P. McDaniel, and K. Butler. Hi-Fi: Collecting High-Fidelity Whole-System Provenance. In *Proceedings of the 28th Annual Comptuer Security Applications Conference*, ACSAC'12, Orlando, FL, USA, 2012.

[16] H. Vijayakumar, G. Jakka, S. Rueda, J. Schiffman, and T. Jaeger. Integrity Walls: Finding Attack Surfaces from Mandatory Access Control Policies. In *Proceedings of the 7th ACM Symposium on Information, Computer and Communications Security*, ASIACCS '12, pages 75–76, New York, NY, USA, 2012. ACM.

[17] C. Wright, C. Cowan, S. Smalley, J. Morris, and G. Kroah-Hartman. Linux security modules: General security support for the linux kernel. In *USENIX Security Symposium*, pages 17–31, 2002.

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

[19] Y. Xie, K.-K. Muniswamy-Reddy, D. Feng, Y. Li, and D. D. E. Long. Evaluation of a Hybrid Approach for Efficient Provenance Storage. *Trans. Storage*, 9(4):14:1–14:29, Nov. 2013.

[20] Y. Xie, K.-K. Muniswamy-Reddy, D. D. E. Long, A. Amer, D. Feng, and Z. Tan. Compressing Provenance Graphs. In *3rd Workshop on the Theory and Practice of Provenance*, TAPP'11, June 2011.