Poster: The Case for Provenance as a First Class Citizen in the Linux Kernel

Adam Bates, Kevin R. B. Butler
Department of Computer and Information Science
University of Oregon, Eugene, OR
{amb, butler}@cs.uoregon.edu

Thomas Moyer
Lincoln Laboratory
Massachusetts Institute of Technology, Lexington, MA
thomas.moyer@ll.mit.edu

I. Where are all the Provenance Monitors?

Provenance is a well-known concept in the art world, but is relatively new to computer science. The idea is that a system can gather and report metadata that describes the history of each object being processed. This allows system users to track, and understand, how a piece of data came to exist in its current state on the system. While the gathering of provenance metadata is something that is done by each provenance-aware system, these systems operate under very different provenance models and assumptions; provenance-aware systems can monitor and record application behavior, filesystem events, or even network activity. As a result, the community has yet to reach a consensus on how to best prototype new provenance proposals, leading to redundant efforts, slower development, and a lack of independent evaluation.

Exacerbating this problem is that, due to a lack of better alternatives, researchers often choose to implement their provenance-aware systems by overloading other system components, such as the Linux Security Module Framework (LSM) or Virtual File System Layer (VFS). The reason for this is that provenance monitors require similar guarantees to those provided by security reference monitors [2]; they need assurances of *complete observation* of system events, which is a subset of the *complete mediation* requirement that the LSM Framework seeks to provide. Unfortunately, this introduces further security and interoperability problems; in order to enable provenance-aware systems, users currently need to disable their MAC policy (e.g., SELinux [9]) or sacrifice other critical system functionality.

These issues point to a pressing need for a dedicated platform for provenance development. We present the design of the first generalized framework for the development of automated, whole-system provenance collection on the Linux operating system. Our provenance framework was designed with consideration for a variety of automated provenance systems that have been proposed in the literature, including Lineage FS [1], Hi-Fi [8], PASS [7], SNooPy [10], and QUIRE [5], among others [3], [4], [6]. The framework is designed in such a way to allow for experimentation with new provenance collection mechanisms.

II. Linux Provenance Modules

We present the Linux Provenance Module Framework (LPM), enabling the development of provenance monitors in the Linux operating system. Figure 1 shows that LPM forms a provenance layer that observes all activity from within the Linux kernel. LPM does not interfere with security; thus, LPM can be protected by Linux's existing security mechanisms, which is both easier and safer. We wrote an SELinux policy to protect LPM's trusted computing base.

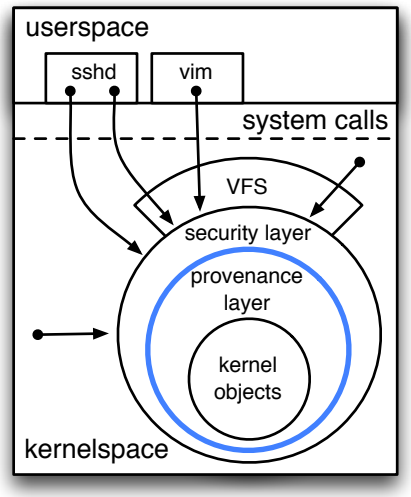

Fig. 1: The LPM Framework runs parallel to the LSM Framework, observing system activity without interfering with the work of the security monitor.

Proposal	Layer	Application Context?	File System?	IPC?	Memory?	Network?	Processes?
HI-FI	Kernel (LSM)		✓	✓	✓	✓	✓
Lineage	Kernel		✓	✓		✓	✓
PASS	Kernel (VFS)	Optional	✓	✓		✓	✓
QUIRE	Platform			✓			
REDUX	Application	✓					
SNooPy	Application					✓	
SProv	Application		✓				
Trio	Application	✓					

Fig. 2: Past proposals for automatic provenance collection vary by scope and operational layer.

Past Proposals. LPM is designed to be a general platform for provenance collection, unifying the functional needs of a variety of provenance projects. Figure 2 shows that these systems vary in the events for which they collect provenance,

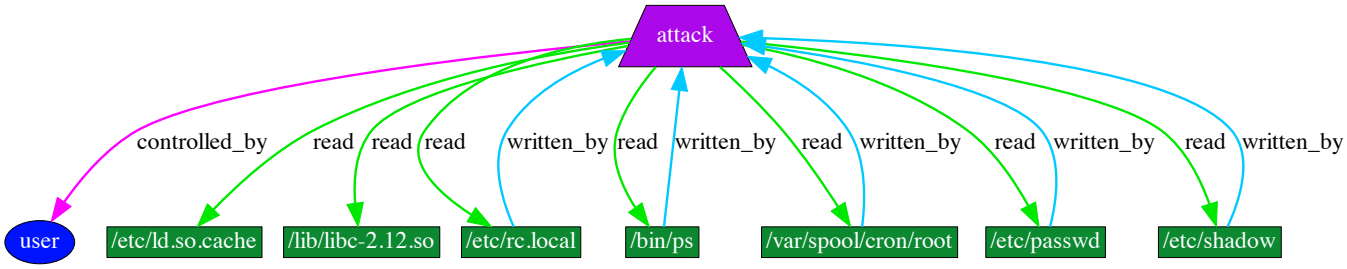

Fig. 2: A provenance graph for a malicious script that obtains persistent access and creates a machine backdoor.

such as application context, files, inter-process communication (IPC), memory, network events, and process executions.

Provenance Hooks. LPM is able to serve all of these needs, observing system activity through a set of 170 provenance hooks that are placed throughout the kernel. We have placed a provenance hook directly after each security hook in the kernel, facilitating provenance collection for all activities permitted by the active security policy. LPM also lets applications annotate provenance for events that cannot be viewed from within the kernel, such as workflows or database queries.

Policy-Reduced Provenance. In practice, provenance monitors record extraneous information, such as the provenance for starting up the system, creating excessive storage overhead. We are developing a new module that can selectively collect provenance based on a user-specified policy. A key insight is that our policy can leverage existing context from a system's security framework. MAC policies that enforce confinement between applications can be utilized to collect complete provenance within a subdomain of system operations.

Securing the Provenance Monitor. Provenance security is its own rich area of study, and yet proposals for provenance-aware systems have at times failed to incorporate security into the foundations of their design. Perhaps the greatest contribution of the LPM Framework will be that developers will no longer need to worry about implementing their own security mechanisms. Instead, they can rely on existing mandatory access control mechanisms such as SELinux. We envision a future in which new experimental provenance modules will be released with an associated SELinux policy, thus allowing for faster, more secure prototyping of provenance-aware systems.

III. Implementation

LPM has been implemented as a patch to the Red Hat Linux 2.6.32 kernel, and currently includes 2 major modules: a re-implementation of Hi-Fi, and a module with a policy mechanism that allows for automated provenance pruning. The latters allows for dramatic reduction in storage overhead, while simultaneously assuring the completeness of provenance collected on a subset of system operations. We have performed the classic kernel compilation macrobenchmark on both of these modules. The Hi-Fi module imposes 2.2% overhead (18 seconds) on kernel compilation compared to the vanilla kernel. For our policy module, we see an 82% decrease in storage costs when kernel compilation falls outside of the specified provenance policy.

IV. Provenance Graphs

LPM can produce provenance graphs (`dot` format) describing the lineage of any Linux kernel object, such as inodes, process executions, and network packets. These graphs can be applied in a variety of ways, such as to explain the impact of an attack. Figure 3 shows the provenance graph of a malicious script that has been executed on our provenance-aware system. The script makes several attempts to obtain persistence on the system, adding a line to `/etc/rc.local`, rewriting `/bin/ps`, and adding a cron job in `/var/spool/cron`. It then creates machine backdoor by modifying `/etc/shadow` and `/etc/passwd` to create a new root user.

V. Conclusion

The LPM Framework will bring usable, secure provenance monitors to the Linux operating system. We will be releasing our source code upon publication, and intend to pursue incorporating LPM into the mainline Linux kernel source tree.

References

[1] Lineage FS. http://crypto.stanford.edu/~cao/lineage.html.

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

[3] R. S. Barga and L. A. Digiampietri. Automatic capture and efficient storage of e-Science experiment provenance. *Concurr. Comput. : Pract. Exper.*, 20:419–429, April 2008.

[4] A. Bates, B. Mood, M. Valafar, and K. Butler. Towards Secure Provenance-based Access Control in Cloud Environments. In *Proceedings of the 3rd ACM Conference on Data and Application Security and Privacy*, CODASPY '13, pages 277–284, New York, NY, USA, 2013. ACM.

[5] M. Dietz, S. Shekhar, Y. Pisetsky, A. Shu, and D. S. Wallach. QUIRE: Lightweight Provenance for Smart Phone Operating Systems. In *Proceedings of the 20th USENIX Security Symposium*, 2011.

[6] 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 *TaPP '10: Proceedings of the 2nd USENIX Workshop on the Theory and Practice of Provenance*, 2010.

[7] K. Muniswamy-Reddy, D. A. Holland, U. Braun, and M. Seltzer. Provenance-Aware Storage Systems. In *Proceedings of the 2006 USENIX Annual Technical Conference*, 2006.

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

[9] S. Smalley, C. Vance, and W. Salamon. Implementing selinux as a linux security module. Technical report, 2002.

[10] W. Zhou, Q. Fei, A. Narayan, A. Haeberlen, B. T. Loo, and M. Sherr. Secure Network Provenance. In *ACM Symposium on Operating Systems Principles (SOSP)*, 2011.