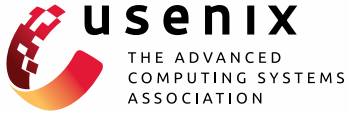

Trustworthy Whole-System Provenance for the Linux Kernel

Adam Bates, Dave (Jing) Tian, and Kevin R.B. Butler, ***University of Florida;***
Thomas Moyer, ***MIT Lincoln Laboratory***

https://www.usenix.org/conference/usenixsecurity15/technical-sessions/presentation/bates

This paper is included in the Proceedings of the 24th USENIX Security Symposium

August 12–14, 2015 • Washington, D.C.

ISBN 978-1-931971-232

Open access to the Proceedings of the 24th USENIX Security Symposium is sponsored by USENIX

Trustworthy Whole-System Provenance for the Linux Kernel

Adam Bates, Dave (Jing) Tian,
Kevin R.B. Butler
University of Florida
{adammbates,daveti,butler}@ufl.edu

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

Abstract

In a provenance-aware system, mechanisms gather and report metadata that describes the history of each object being processed on the system, allowing users to understand how data objects came to exist in their present state. However, while past work has demonstrated the usefulness of provenance, less attention has been given to *securing* provenance-aware systems. Provenance itself is a ripe attack vector, and its authenticity and integrity must be guaranteed before it can be put to use.

We present Linux Provenance Modules (LPM), the first general framework for the development of provenance-aware systems. We demonstrate that LPM creates a trusted provenance-aware execution environment, collecting complete whole-system provenance while imposing as little as 2.7% performance overhead on normal system operation. LPM introduces new mechanisms for secure provenance layering and authenticated communication between provenance-aware hosts, and also interoperates with existing mechanisms to provide strong security assurances. To demonstrate the potential uses of LPM, we design a Provenance-Based Data Loss Prevention (PB-DLP) system. We implement PB-DLP as a file transfer application that blocks the transmission of files derived from sensitive ancestors while imposing just tens of milliseconds overhead. LPM is the first step towards widespread deployment of trustworthy provenance-aware applications.

1 Introduction

A provenance-aware system automatically gathers and reports metadata that describes the history of each object being processed on the system. This allows users to track, and understand, how a piece of data came to exist in its current state. The application of provenance is presently of enormous interest in a variety of disparate communities including scientific data processing, databases, software development, and storage [43, 53]. Provenance has also been demonstrated to be of great value to security by identifying malicious activity in data centers [5, 27, 56, 65, 66], improving Mandatory Access Control (MAC) labels [45, 46, 47], and assuring regulatory compliance [3].

Unfortunately, most provenance collection mechanisms in the literature exist as fully-trusted user space applications [28, 27, 41, 56]. Even kernel-based provenance mechanisms [43, 48] and sketches for trusted provenance architectures [40, 42] fall short of providing a provenance-aware system for malicious environments. The problem of whether or not to trust provenance is further exacerbated in distributed environments, or in layered provenance systems, due to the lack of a mechanism to verify the authenticity and integrity of provenance collected from different sources.

In this work, we present **Linux Provenance Modules (LPM)**, the first generalized framework for secure provenance collection on the Linux operating system. Modules capture *whole-system provenance*, a detailed record of processes, IPC mechanisms, network activity, and even the kernel itself; this capture is invisible to the applications for which provenance is being collected. LPM introduces a gateway that permits the upgrading of low integrity workflow provenance from user space. LPM also facilitates secure distributed provenance through an authenticated, tamper-evident channel for the transmission of provenance metadata between hosts. LPM interoperates with existing security mechanisms to establish a hardware-based root of trust to protect system integrity.

Achieving the goal of trustworthy whole-system provenance, we demonstrate the power of our approach by presenting a scheme for *Provenance-Based Data Loss Prevention (PB-DLP)*. PB-DLP allows administrators to reason about the propagation of sensitive data and control its further dissemination through an expressive policy system, offering dramatically stronger assurances than existing enterprise solutions, while imposing just mil-

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.

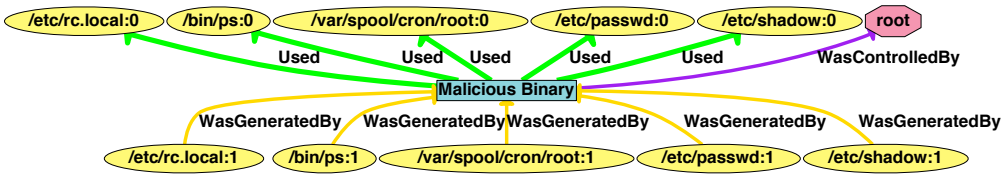

Figure 1: A provenance graph showing the attack footprint of a malicious binary. Edges encode relationships that flow backwards into the history of system execution, and writing to an object creates a second node with an incremented version number. Here, we see that the binary has rewritten `/etc/rc.local`, likely in an attempt to gain persistence after a system reboot.

liseconds of overhead on file transmission. To our knowledge, this work is the first to apply provenance to DLP.

Our contributions can thus be summarized as follows:

- **Introduce Linux Provenance Modules (LPM).** LPM facilitates secure provenance collection at the kernel layer, supports *attested disclosure* at the application layer, provides an *authenticated channel* for network transmission, and is compatible with the W3C Provenance (PROV) Model [59]. In evaluation, we demonstrate that provenance collection imposes as little as 2.7% performance overhead.

- **Demonstrate secure deployment.** Leveraging LPM and existing security mechanisms, we create a trusted provenance-aware execution environment for Linux. Through porting Hi-Fi [48] and providing support for SPADE [29], we demonstrate the relative ease with which LPM can be used to secure existing provenance collection mechanisms. We show that, in realistic malicious environments, ours is the first proposed system to offer secure provenance collection.

- **Introduce Provenance-Based Data Loss Prevention (PB-DLP).** We present a new paradigm for the prevention of data leakage that searches object provenance to identify and prevent the spread of sensitive data. PB-DLP is impervious to attempts to launder data through intermediary files and IPC. We implement PB-DLP as a file transfer application, and demonstrate its ability to query object ancestries in just tens of milliseconds.

2 Background

Data provenance, sometimes called *lineage*, describes the actions taken on a data object from its creation up to the present. Provenance can be used to answer a variety of historical questions about the data it describes. Such questions include, but are not limited to, "*What processes and datasets were used to generate this data?*" and "*In what environment was the data produced?*" Conversely, provenance can also answer questions about the successors of a piece of data, such as "*What objects on the system were derived from this object?*" Although potential applications for such information are nearly limitless, past proposals have conceptualized provenance in different ways, indicating that a one-size-fits-all solution to provenance collection is unlikely to meet the needs of all of these audiences. We review these past proposals for provenance-aware systems in Section 8.

The commonly accepted representation for data provenance is a directed acyclic graph (DAG). In this work, we use the W3C PROV-DM specification [59] because it is pervasive and facilitates the exchange of provenance between deployments. An example PROV-DM graph of a malicious binary is shown in Figure 1. This graph describes an attack in which a binary running with root privilege reads several sensitive system files, then edits those files in an attempt to gain persistent access to the host. Edges encode relationships between nodes, pointing backwards into the history of system execution. Writing to an object triggers the creation of a second object node with an incremented version number. This particular provenance graph could serve as a valuable forensics tool, allowing system administrators to better understand the nature of a network intrusion.

2.1 Data Loss Prevention

Data Loss Prevention (DLP) is enterprise software that seeks to minimize the leakage of sensitive data by monitoring and controlling information flow in large, complex organizations [1].[1] In addition to the desire to control intellectual property, another motivator for DLP systems is demonstrating regulatory compliance for personally-identifiable information (PII),[2] as well as directives such

[1] Our overview of data loss prevention is based on review of publicly available product descriptions for software developed by Bit9, CDW, Cisco, McAfee, Symantec, and Titus.

[2] See NIST SP 800-122

as PCI,[3] HIPAA,[4] SOX.[5] or E.U. Data Protection.[6] As encryption can be used to protect data at rest from unauthorized access, the true DLP challenge involves preventing leakage at the hands of authorized users, both malicious and well-meaning agents. This latter group is a surprisingly big problem in the fight to control an organization's intellectual property; a 2013 study conducted by the Ponemon Institute found that over half of companies' employees admitted to emailing intellectual property to their personal email accounts, with 41 percent admitting to doing so on a weekly basis [2]. It is therefore important for a DLP system to be able to exhaustively explain which pieces of data are sensitive, where that data has propagated to within the organization, and where it is (and is not) permitted to flow.

As DLP systems are proprietary and are marketed so as to abstract away the complex details of their internal workings, we cannot offer a complete explanation of their core features. However, some of the mechanisms in such systems are known. Many DLP products use a *regular expression-based* approach to identify sensitive data, operating similarly to a general-purpose version of Cornell's Spider[7]. For example, in PCI compliance,[3] DLP might attempt to identify credit card numbers in outbound emails by searching for 16 digit numbers that pass a Mod-10 validation check [39]. Other DLP systems use a *label-based* approach to identify sensitive data, tagging document metadata with security labels. The Titus system accomplishes this by having company employees manually annotate the documents that they create;[8] plug-ins for applications (e.g., Microsoft Office) then prevent the document from being transmitted to or opened by other employees that lack the necessary clearance. In either approach, DLP software is difficult to configure and prone to failure, offering marginal utility at great price.

3 Linux Provenance Modules

To serve as the foundation for secure provenance-aware systems, we present *Linux Provenance Modules (LPM)*. We provide a working definition for the provenance our system will collect in §3.1. In §3.2 we consider the capabilities and aims of a provenance-aware adversary, and identify security and design goals in §3.3. The LPM design is presented in §3.4, and in §3.5 we demonstrate its secure deployment. An expanded description of our system is available in our technical report [8].

[3] See https://www.pcisecuritystandards.org
[4] See http://www.hhs.gov/ocr/privacy
[5] Short for the Sarbanes-Oxley Act, U.S. Public Law No. 107-20
[6] See EU Directive 95/46/EC
[7] See http://www2.cit.cornell.edu/security/tools
[8] See http://www.titus.com

3.1 Defining Whole-System Provenance

In the design of LPM, we adopt a model for *whole-system provenance*[9] that is broad enough to accommodate the needs of a variety of existing provenance projects. To arrive at a definition, we inspect four past proposals that collect broadly scoped provenance: SPADE [29], LineageFS [53], PASS [43], and Hi-Fi [48]. **SPADE** provenance is structured around primitive operations of system activities with data inputs and outputs. It instruments file and process system calls, and associates each call to a process ID (PID), user identifier, and network address. **LineageFS** uses a similar definition, associating process IDs with the file descriptors that the process reads and writes. **PASS** associates a process's output with references to all input files and the command line and process environment of the process; it also appends out-of-band knowledge such as OS and hardware descriptions, and random number generator seeds, if provided. In each of these systems, networking and IPC activity is primarily reflected in the provenance record through manipulation of the underlying file descriptors. **Hi-Fi** takes an even broader approach to provenance, treating non-persistent objects such as memory, IPC, and network packets as principal objects.

We observe that, in all instances, provenance-aware systems are exclusively concerned with operations on *controlled data types*, which are identified by Zhang et al. as files, inodes, superblocks, socket buffers, IPC messages, IPC message queue, semaphores, and shared memory [64]. Because controlled data types represent a super set of the objects tracked by system layer provenance mechanisms, we define whole-system provenance as *a complete description of agents (users, groups) controlling activities (processes) interacting with controlled data types during system execution.*

3.2 Threat Model & Assumptions

We consider an adversary that has gained remote access to a provenance-aware host or network. Once inside the system, the attacker may attempt to remove provenance records, insert spurious information into those records, or find gaps in the provenance monitor's ability to record information flows. A network attacker may also attempt to forge or strip provenance from data in transit. Because captured provenance can be put to use in other applications, the adversary's goal may even be to target the provenance monitor itself. The implications and methods of such an attack are domain-specific. For example:

[9] This term is coined in [48], but not explicitly defined. We demonstrate the concrete requirements of a collection mechanism for whole-system provenance in this work.

- **Scientific Computing**: An adversary may wish to manipulate provenance in order to commit fraud, or to inject uncertainty into records to trigger a "Climategate"-like controversy [50].
- **Access Control**: When used to mediate access decisions [7, 45, 46, 47], an attacker could tamper with provenance in order to gain unauthorized access, or to perform a denial-of-service attack on other users by artificially escalating the security level of data objects.
- **Networks**: Provenance metadata can also be associated with packets in order to better understand network events in distributed systems [5, 65, 66]. Coordinating multiple compromised hosts, an attacker may attempt to send *unauthenticated* messages to avoid provenance generation and to perform data exfiltration.

We define a provenance trusted computing base (TCB) to be the kernel mechanisms, provenance recorder, and storage back-ends responsible for the collection and management of provenance. *Provenance-aware applications are not considered part of the TCB.*

We make the following assumption with regards to the TCB. In Linux, kernel modules have unrestricted access to kernel memory, meaning that there is no mechanism for protecting LPM from the rest of the kernel. The kernel code is therefore trusted; we assume that the stock kernel will not seek to tamper with the TCB. However, we do consider the possibility that the kernel could be compromised after installation through its interactions with user space applications. To facilitate host attestation in distributed environments, we also assume access to a Public Key Infrastructure (PKI) for provenance-aware hosts to publish their public signing keys.

3.3 System Goals

We set out to provide the following security assurances in the design of of our system-layer provenance collection mechanism. McDaniel et al. liken the needs of a secure provenance monitor [42] to the reference monitor guarantees laid out by Anderson [4]: complete mediation, tamperproofness, and verifiability. We define these guarantees as follows:

G1 Complete. Complete mediation for provenance has been discussed elsewhere in the literature in terms of assuring *completeness* [32]: that the provenance record be gapless in its description of system activity. To facilitate this, LPM must be able to observe all information flows that pass through controlled data types.

G2 Tamperproof. As many provenance use cases involve enhancing system security, LPM will be an adversarial target. The TCB must therefore be impervious to disabling or manipulation by processes in user space.

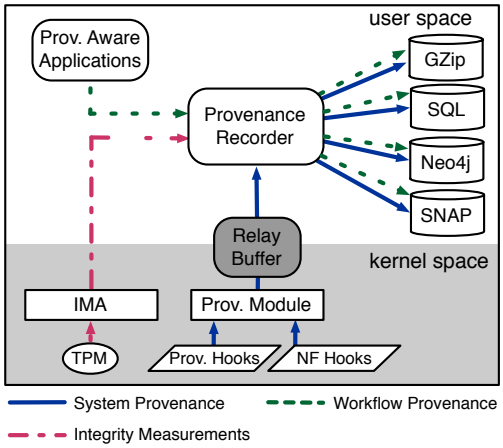

Figure 2: Diagram of the LPM Framework. Kernel hooks report provenance to a recorder in userspace, which uses one of several storage back-ends. The recorder is also responsible for evaluating the integrity of workflow provenance prior to storing it.

G3 Verifiable. The functionality of LPM must be verifiably correct. Additionally, local and remote users should be able to attest whether the host with which they are communicating is running the secured provenance-aware kernel.

Through surveying past work in provenance-aware systems, we identify the following additional goals to support whole-system provenance:

G4 Authenticated Channel. In distributed environments, provenance-aware systems must provide a means of assuring authenticity and integrity of provenance as it is communicated over open networks [7, 42, 48, 65]. *While we do not seek to provide a complete distributed provenance solution in LPM*, we do wish to provide the required building blocks within the host for such a system to exist. LPM must therefore be able to monitor every network message that is sent or received by the host, and reliably explain these messages to other provenance-aware hosts in the network.

G5 Attested Disclosure. Layered provenance, where additional metadata is disclosed from higher operational layers, is a desirable feature in provenance-aware systems, as applications are able to report workflow semantics that are invisible to the operating system [44]. LPM must provide a gateway for

upgrading low integrity user space disclosures before logging them in the high integrity provenance record. This is consistent with the Clark-Wilson Integrity model for upgrading or discarding low integrity inputs [17].

In order to bootstrap trust in our system, we have implemented LPM as a parallel framework to Linux Security Modules (LSM) [60, 61]. Building on these results, we show in Section 4 that this approach allows LPM to inherit the formal assurances that have been verified for the LSM architecture.

3.4 Design & Implementation

An overview of the LPM architecture is shown in Figure 2. The LPM patch places a set of *provenance hooks* around the kernel; a *provenance module* then registers to control these hooks, and also registers several Netfilter hooks; the module then observes system events and transmits information via a relay buffer to a *provenance recorder* in user space that interfaces with a datastore. The recorder also accepts disclosed provenance from applications after verifying their correctness using the Integrity Measurements Architecture (IMA) [52].

In designing LPM, we first considered using an experimental patch to the LSM framework that allows "stacking" of LSM modules [10]. However, at this time, no standard exists for handling when modules make conflicting decisions, creating the potential unpredicted behavior. We also felt that dedicated provenance hooks were necessary; by collecting provenance *after* LSM authorization routines, we ensure that the provenance history is an accurate description of authorized system events. If provenance collection occurred *during* authorization, as would be the case with stacked LSMs, it would not be possible to provide this property.

3.4.1 Provenance Hooks

The LPM patch introduces a set of hook functions in the Linux kernel. These hooks behave similarly to the LSM framework's security hooks in that they facilitate modularity, and default to taking no action unless a module is enabled. Each provenance hook is placed directly beneath a corresponding security hook. The return value of the security hook is checked prior to calling the provenance hook, thus assuring that the requested activity has been authorized prior to provenance capture; we consider the implications of this design in Section 4. A workflow for the hook architecture is depicted in Figure 3. The LPM patch places over 170 provenance hooks, one for each of the LSM authorization hooks. In addition to the hooks that correspond to existing security hooks, we also support Pohly et al.'s Hi-Fi [48] hook that is necessary to preserve Lamport timestamps on network messages [38].

[10] See https://lwn.net/Articles/518345/

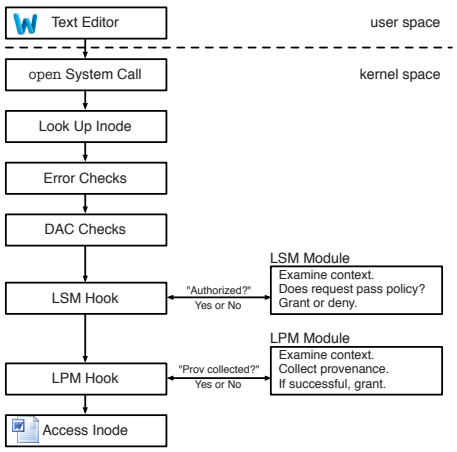

Figure 3: Hook Architecture for the `open` system call. Provenance is collected *after* DAC and LSM checks, ensuring that it accurately reflects system activity. LPM will only deny the operation if it fails to generate provenance for the event.

3.4.2 Netfilter Hooks

LPM uses Netfilter hooks to implement a cryptographic message commitment protocol. In Hi-Fi, provenance-aware hosts communicated by embedding a provenance sequence number in the IP options field [49] of each outbound packet [48]. This approach allowed Hi-Fi to communicate as normal with hosts that were not provenance-aware, but unfortunately was not secure against a network adversary. In LPM, provenance sequence numbers are replaced with Digital Signature Algorithm (DSA) signatures, which are space-efficient enough to embed in the IP Options field. We have implemented full DSA support in the Linux kernel by creating signing routines to use with the existing DSA verification function. DSA signing and verification occurs in the NetFilter `inet_local_out` and `inet_local_in` hooks. In `inet_local_out`, LPM signs over the immutable fields of the IP header, as well as the IP payload. In `inet_local_in`, LPM checks for the presence of a signature, then verifies the signature against a configurable list of public keys. If the signature fails, the packet is dropped before it reaches the recipient application, thus ensuring that there are no breaks in the continuity of the provenance log. The key store for provenance-aware hosts is obtained by a PKI and transmitted to the kernel during the boot process by writing to `securityfs`. LPM registers the Netfilter hooks with the highest prior-

ity levels, such that signing occurs just before transmission (i.e., after all other IPTables operations), and signature verification occurs just after the packet enters the interface (i.e., before all other IPTables operations).

3.4.3 Provenance Modules

Here, we introduce two of our own provenance modules (Provmon, SPADE), as well as briefly mention the work of our peers (UPTEMPO):

- **Provmon**. Provmon is an extended port of the Hi-Fi security module [48]. The original Hi-Fi code base was 1,566 lines of code, requiring 723 lines to be modified in the transition. Our extensions introduced 728 additional lines of code. The process of porting did not affect the module's functionality, although we have subsequently extended the Hi-Fi protocol to capture additional lineage information:

  File Versioning. The original Hi-Fi protocol did not track version information for files, leading to uncertainty as to the exact contents of a file at the time it was read. Accurately recovering this information in user space was not possible due to race conditions between kernel events. Because versioning is necessary to break cycles in provenance graphs [43], we have added a version field to the provenance context for inodes, which is incremented on each write.

  Network Context. Hi-Fi omitted remote host address information for network events, reasoning that source information could be forged by a dishonest agent in the network. These human-interpretable data points were replaced with an assigned random identifier for each packet. We found, however, that these identifiers could not be interpreted without remote address information, and incorporated the recording of remote IP addresses and ports into Provmon.

- **SPADE**. The SPADE system is an increasingly popular option for provenance auditing, but collecting provenance in user space limits SPADE's expressiveness and creates the potential for incomplete provenance. To address this limitation, we have created a mechanism that reports LPM provenance into SPADE's Domain-Specific Language pipe [29]. This permits the collection of whole-system provenance while simultaneously leveraging SPADE's existing storage, remote query, and visualization utilities.

- **Using Provenance to Expedite MAC Policies (UPTEMPO)**. Using LPM as a collection mechanism, Moyer et al. investigate provenance analysis as a means of administrating Mandatory Access Control (MAC) policies [54]. UPTEMPO first observes system execution in a sterile environment, aggregating LPM provenance in a centralized data store. It then recovers the implicit information flow policy through mining the provenance store to generate a MAC policy for the distributed system, decreasing both administrator effort and the potential for misconfiguration.

3.4.4 Provenance Recorders

LPM provides modular support for different storage through *provenance recorders*. To prevent an infinite provenance loop, recorders are flagged as provenance-opaque [48] using the `security.provenance` extended attribute, which is checked by LPM before creating a new event. Each recorder was designed to be as agnostic to the active LPM as possible, making them easy to adapt to new modules.

We currently provide provenance recorders that offer backend storage for *Gzip*, *PostGreSQL*, *Neo4j*, and *SNAP*. Commentary on our PostGreSQL and Neo4j reporters can be found in our technical report [8]. We make use of the Gzip and SNAP recorders during our evaluation in Section 6.

The Gzip recorder incurs low storage overheads and fast insertion speeds. On our test bed, we observed this recorder processing up to 400,000 events per second from the Provmon provenance stream. However, because the provenance is not stored in an easily queried form, this back-end is best suited for environments where queries are an offline process.

To create graph storage that was efficient enough for LPM, we used the SNAP graphing library[11] to design a recorder that maintains an in-memory graph database that is fully compliant with the W3C PROV-DM Model [59]. We have observed insertion speeds of over 150,000 events per second using the SNAP recorder, and highly efficient querying as well. This recorder is further evaluated in Section 6.

3.4.5 Workflow Provenance

To support layered provenance while preserving our security goals, we require a means of evaluating the integrity of user space provenance disclosures. To accomplish this, we extend the LPM Provenance Recorder to use the Linux Integrity Measurement Architecture (IMA) [35, 52]. IMA computes a cryptographic hash of each binary before execution, extends the measurement into a TPM Platform Control Register (PCR), and stores the measurement in kernel memory. This set of measurements can be used by the Recorder to make a decision about the integrity of the a Provenance-Aware Application (PAA) prior to accepting the disclosed provenance. When a PAA wishes to disclose provenance, it opens a

[11] See http://snap.stanford.edu

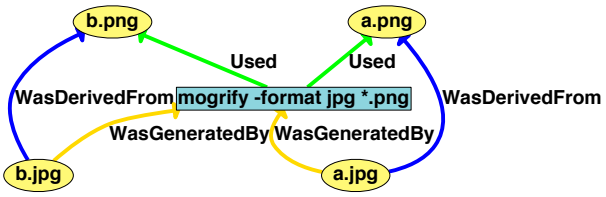

Figure 4: A provenance graph of image conversion. Here, workflow provenance (*WasDerivedFrom*) encodes a relationship that more accurately identifies the output files' dependencies compared to only using kernel layer observations (*Used*, *WasGeneratedBy*).

new UNIX domain socket to send the provenance data to the Provenance Recorder. The Recorder uses its own UNIX domain socket to recover the process's pid, then uses the `/proc` filesystem to find the full path of the binary, then uses this information to look up the PAA in the IMA measurement list. The disclosed provenance is recorded only if the signature of PAA matches a known-good cryptographic hash.

As a demonstration of this functionality, we created a provenance-aware version of the popular ImageMagick utility [12]. ImageMagick contains a batch conversion tool for image reformatting, `mogrify`. Shown in Figure 4, `mogrify` reads and writes multiple files during execution, leading to an *overtainting* problem – at the kernel layer, LPM is forced to conservatively assume that all outputs were derived from all inputs, creating false dependencies in the provenance record. To address this, we extended the Provmon protocol to support a new message, `provmsg_imagemagick_convert`, which links an input file directly to its output file. When the recorder receives this message, it first checks the list of IMA measurements to confirm that ImageMagick is in a good state. If successful, it then annotates the existing provenance graph, connecting the appropriate input and output objects with *WasDerivedFrom* relationships. Our instrumentation of ImageMagick demonstrates that LPM supports layered provenance at no additional cost over other provenance-aware systems [29, 43], and does so in a manner that provides assurance of the integrity of the provenance log.

3.5 Deployment

We now demonstrate how we used LPM in the deployment of a secure provenance-aware system. Additional background on the security technologies use in our deployment can be found in our technical report [8].

[12] See http://www.imagemagick.org

3.5.1 Platform Integrity

We configured LPM to run on a physical machine with a Trusted Platform Module (TPM). The TPM provides a root of trust that allows for a measured boot of the system. The TPM also provides the basis for remote attestations to prove that LPM was in a known hardware and software configuration. The BIOS's core root of trust for measurement (CRTM) bootstraps a series of code measurements prior to the execution of each platform component. Once booted, the kernel then measures the code for user space components (e.g., provenance recorder) before launching them, through the use of the Linux Integrity Measurement Architecture (IMA)[52]. The result is then extended into TPM PCRs, which forms a verifiable chain of trust that shows the integrity of the system via a digital signature over the measurements. A remote verifier can use this chain to determine the current state of the system using TPM attestation.

We configured the system with Intel's Trusted Boot (tboot),[13] which provides a secure boot mechanism, preventing system from booting into the environment where critical components (e.g., the BIOS, boot loader and the kernel) are modified. Intel tboot relies on the Intel TXT[14] to provide a secure execution environment. [15] Additionally, we compiled support for IMA into the provenance-aware kernel, which is necessary in order for the LPM Recorder to be able to measure the integrity of provenance-aware applications.

3.5.2 Runtime Integrity

After booting into the provenance-aware kernel, the runtime integrity of the TCB (defined in §3.2) must also be assured. To protect the runtime integrity of the kernel, we deploy a Mandatory Access Control (MAC) policy, as implemented by Linux Security Modules. On our prototype deployments, we enabled SELinux's MLS policy, the security of which was formally modeled by Hicks et al. [33]. Refining the SELinux policy to prevent Access Vector Cache (AVC) denials on LPM components required minimal effort; the only denial we encountered was when using the PostgreSQL recorder, which was quickly remedied with the `audit2allow` tool. Preserving the integrity of LPM's user space components, such as the provenance recorder, was as simple as creating a new policy module. We created a policy module to protect the LPM recorder and storage back-end using the `sepolicy` utility. Uncompiled, the policy module was only 135 lines.

[13] See http://sf.net/projects/tboot

[14] See https: //www.kernel.org/doc/Documentation/intel_txt.txt

[15] For virtual environments, similar functionality can be provided on Xen via TPM *sealing* and the virtual TPM (vTPM), which is bound to the physical TPM of the host system.

4 Security

In this section, we demonstrate that our system meets all of the required security goals for trustworthy whole-system provenance. In this analysis, we consider an LPM deployment on a physical machine that was enabled with the Provmon module and has been configured to the conditions described in Section 3.5.

Complete (G1). We defined whole-system provenance as a complete description of agents (users, groups) controlling activities (processes) interacting with controlled data types during system execution (§ 3.1). LPM attempts to track these system objects through the placement of provenance hooks (§3.4.1), which directly follow each LSM authorization hook. The LSM's complete mediation property has been formally verified [20, 64]; in other words, there is an authorization hook prior to every security-sensitive operation. Because every interaction with a controlled data type is considered security-sensitive, we know that a provenance hook resides on all control paths to the provenance-sensitive operations. LPM is therefore capable of collecting complete provenance on the host.

It is important to note that, as a consequence of placing provenance hooks beneath authorization hooks, LPM is unable to record failed access attempts. However, inserting the provenance layer beneath the security layer ensures accuracy of the provenance record. Moreover, failed authorizations are a different kind of metadata than provenance because they do not describe processed data; this information is better handled at the security layer, e.g., by the SELinux Access Vector Cache (AVC) Log.

Tamperproof (G2). The runtime integrity of the LPM trusted computing base is assured via the SELinux MLS policy, and we have written a policy module that protects the LPM user space components (§3.5.2). Therefore, the only way to disable LPM would be to reboot the system into a different kernel; this action can be disallowed through secure boot techniques,[13] and is detectable by remote hosts via TPM attestation (§3.5.1).

Verifiable (G3). While we have not conducted an independent formal verification of LPM, our argument for its correctness is as follows. A provenance hook follows each LSM authorization hook in the kernel. The correctness of LSM hook placement has been verified through both static and dynamic analysis techniques [20, 25, 34]. Because an authorization hook exists on the path of every sensitive operation to controlled data types, and LPM introduces a provenance hook behind each authorization hook, LPM inherits LSM's formal assurance of complete mediation over controlled data types. This is sufficient to ensure that LPM can collect the provenance of every sensitive operation on controlled data types in the kernel (i.e., whole-system provenance).

Authenticated Channel (G4). Through use of Netfilter hooks [57], LPM embeds a DSA signature in every outbound network packet. Signing occurs immediately prior to transmission, and verification occurs immediately after reception, making it impossible for an adversary-controlled application running in user space to interfere. For both transmission and reception, the signature is invisible to user space. Signatures are removed from the packets before delivery, and LPM feigns ignorance that the options field has been set if `get_options` is called. Hence, LPM can enforce that all applications participate in the commitment protocol.

Prior to implementing our own message commitment protocol in the kernel, we investigated a variety of existing secure protocols. The integrity and authenticity of provenance identifiers could also be protected via IPsec [36], SSL tunneling,[16] or other forms of encapsulation [5, 65]. We elected to move forward with our approach because 1) it ensures the monitoring of all *all* processes and network events, including non-IP packets, 2) it does not change the number of packets sent or received, ensuring that our provenance mechanism is minimally invasive to the rest of the Linux network stack, and 3) it preserves compatibility with non-LPM hosts. An alternative to DSA signing would be HMAC [9], which offers better performance but requires pairwise keying and sacrifices the non-repudiation policy; BLS, which approaches the theoretical maximum security parameter per byte of signature [12]; or online/offline signature schemes [15, 23, 26, 55].

Authenticated Disclosures (G5). We make use of IMA to protect the channel between LPM and provenance-aware applications wishing to disclose provenance. IMA is able to prove to the provenance recorder that the application was unmodified at the time it was loaded into memory, at which point the recorder can accept the provenance disclosure into the official record. If the application is known to be correct (e.g., through formal verification), this is sufficient to establish the runtime integrity of the application. However, if the application is compromised after execution, this approach is unable to protect against provenance forgery.

A separate consideration for all of the above security properties are Denial of Service (DoS) attacks. *DoS attacks on LPM do not break its security properties*. If an attacker launches a resource exhaustion attack in order to prevent provenance from being collected, all kernel operations will be disallowed and the host will cease to function. If a network attacker tampers with a packet's provenance identifier, the packet will not be delivered to the recipient application. In all cases, the provenance record remains an accurate reflection of system events.

[16] See http://docs.oracle.com/cd/E23823_01/html/816-5175/kssl-5.html

Algorithm 1 Summarizes a's propagation through the system.

```
Require: a is an entity
procedure REPORT(a)
    Locations = [ ]                                  ▷ Assigns an empty list.
    for each s in a, FindSuccessors(a) do
        if s.type is File then
            Locations.Add(< s.disk, s.directory >)
        else if s.type is Network Packet then
            Locations.Add(< s.remote_ip, s.port >)
        end if
    end for
    return Locations
end procedure
```

Algorithm 2 Mediates request to write e to d given *Rules*.

```
Require: e, d are entities
Require: Rules is a PB-DLP policy
procedure PROVWRITE(e, d, Rules)
    for each rule in Rules do
        if d = rule.dst then
            A = FindAncestors(e)
            NumSrcs = length(rule.Srcs)
            for each src in rule.Srcs do
                if src in A then
                    NumSrcs − −
                end if
            end for
            if NumSrcs = 0 then                      ▷ A ⊇ Srcs, deny.
                return PB-DLP_DENY
            end if
        end if
    end for
    return PB-DLP_PERMIT                             ▷ A ⊉ Srcs, permit.
end procedure
```

5 LPM Application: Provenance-Based Data Loss Prevention

To further demonstrate the power of LPM, we now introduce a set of mechanisms for Provenance-Based Data Loss Prevention (PB-DLP) that offer dramatically simplified administration and improved enforcement over existing DLP systems. A provenance-based approach is a novel and effective means of handling data loss prevention; to our knowledge, we are the first in the literature to do so. The advantage of our approach when compared to existing systems is that LPM-based provenance-aware systems already perform system-wide capture of information flows between kernel objects. Data loss prevention in such a system therefore becomes a matter of preventing all derivations of a sensitive source entity, e.g., a Payment Card Industry (PCI) database, from being written to a monitored destination entity (e.g., a network interface).

We begin by defining a policy format for PB-DLP. Individual rules take the form

$$< Srcs = [src_1, src_2, \ldots, src_n], dst >$$

where *Srcs* is a list of entities representing persistent data objects, and dst is a single entity representing either a persistent data object such as a file or interface or an abstract entity such as a remote host. The goal for PB-DLP is as follows – an entity e_1 with ancestors A is written to entity e_2 if and only if $A \not\supseteq Srcs$ for *all* rules in the rule set where $e_2 = dst$. The reason that sources are expressed as sets is that, at times, the union of information is more sensitive than its individual components. For example, sharing a person's last name *or* birthdate may be permissible, while sharing the last name *and* birthdate is restricted as PII.[2]

Below, we define the functions that realize this goal. First, we define two provenance-based functions as the basis for a DLP *monitoring phase*, which allows administrators to learn more about the propagation of sensitive data on their systems. Then, we define mechanisms for a DLP *enforcement phase*.

5.1 Monitoring Phase

The goal of monitoring is to allow administrators to reason about how sensitive data is stored and put to use on their systems. The end product of the monitor phase is a set of rules (a policy) that restrict the permissible flows for sensitive data sources. Monitoring is an ongoing process in DLP, where administrators attempt to iteratively improve protection against data leakage. The first step is to identify the data that needs protection. Identifying the source of such information is often quite simple; for example, a database of PCI or PII data. However, reliably finding data objects that were derived from this source is extraordinarily complicated using existing solutions, but is simple now with LPM. To begin, we define a helper function for system monitoring:

1. *FindSuccessors(Entity)*: This function performs a provenance graph traversal to obtain the list of data objects derived from *Entity*.

FindSuccessors can then be used as the basis for a function that summarizes the spread of sensitive data:

2. *Report(Entity)*: List the locations that a target object and its successors have propagated. This function is defined in Algorithm 1.

The information provided by *Report* is similar to the data found in the Symantec DLP Dashboard [1], and could be used as the backbone of a PB-DLP user interface. Administrators can use this information to write a PB-DLP policy or revise an existing one.

5.2 Enforcement Phase

Possessing a PB-DLP policy, the goal of the enforcement phase is to prevent entities that were derived from sensitive sources from being written to restricted locations. To

do so, we need to inspect the object's provenance to discover the entities from which it was derived. We define the following helper function:

3. *FindAncestors(Entity)*: This function performs a provenance graph traversal to obtain the list of data objects used in the creation of *Entity*.

FindAncestors can be then used as the basis for a function that prevents the spread of sensitive data:

4. *ProvWrite(Entity, Destination, Rules)*: Write the target entity to the destination if and only if it is valid to the provided rule set, as defined in Algorithm 2.

5.3 File Transfer Application

In many enterprise networks that are isolated from the Internet via firewalls and proxies, it is desirable to share files with external users. File transfer services are one way to achieve this, and provide a single entry/exit point to the enterprise network where files being transferred can be examined before being released.[17] In the case of incoming files, scans can check for known malware, and in some cases, check for other types of malicious behavior from unknown malware.

We implemented PB-DLP as a file transfer application for provenance-aware systems using LPM's Provmon module. The application interfaced with LPM's SNAP recorder using a custom API. Before permitting a file to be transmitted to a remote host, the application ran a query that traversed *WasDerivedFrom* edges to return a list of the file's ancestors, permitting the transfer only if the file was not derived from a restricted source. PB-DLP allows internal users to share data, while ensuring that sensitive data is not exfiltrated in the process.

Because provenance graphs continue to grow indefinitely over time, in practice the bottleneck of this application is the speed of provenance querying. We evaluate the performance of PB-DLP queries in Section 6.3.

5.4 PB-DLP Analysis

Below, we select two open source systems that approximate *label based* and *regular expression (regex) based* DLP solutions, and compare their benefits to PB-DLP.

5.4.1 Label-Based DLP

The SELinux MLS policy [31] provides information flow security through a label-based approach, and could be used to approximate a DLP solution without relying on commercial products. Proprietary label-based DLP systems rely on manual annotations provided by users, requiring them to provide correct labeling based on their knowledge of data content. Using SELinux as an exemplar labeling system is therefore an *extremely conservative* approach to analysis.

Within an MLS system, each subject, and object, is assigned a classification level, and categories, or compartments. Consider an example system, with classification levels, $\{A, B\}$ with A dominating B, and compartments $\{\alpha, \beta\}$. We can model our policy as a lattice, where each node in the lattice is a tuple of the form $\{< level >, \{compartments\}\}$. Once the policy is defined, it is possible to enforce the simple and *-properties. If a user has access to data with classification level A, and compartment α, he cannot read anything in compartment $\{\beta\}$ (*no read-up*). Furthermore, when data is accessed in A,$\{\alpha\}$, the user cannot write anything to B,$\{\alpha\}$ (*no write-down*).

In order to use SELinux's MLS enforcement as a DLP solution, the administrator configures the policy to enforce the constraint that no data of specified types can be sent over the network. However, this is difficult in practice. Consider an example system that processes PII. The users of the system may need to access information, such as last names, and send these to the payroll department to ensure that each employee receives a paycheck. Separately, the user may need to send a list of birthdays to another user in the department to coordinate birthday celebrations for each month. Either of these activities are acceptable (Figure 5, Decision Condition 2). However, it is common practice for organizations to have stricter sharing policies for data that contains multiple forms of PII, so while either of these identifiers could be transmitted in isolation, the two pieces of information combined could not be shared (Figure 5, Decision Condition 3).

The MLS policy cannot easily handle this type of data fusion. In order to provide support for correctly labeling fused data, an administrator would need to define the power set of all compartments within the MLS policy. In the example above, the administrator would define the following compartments: $\{\}$, $\{\alpha\}$, $\{\beta\}$, $\{\alpha, \beta\}$. In the default configuration SELinux supports 256 unique categories, meaning an SELinux DLP policy could only support eight types of data. Furthermore, the MLS policy does not support defining multiple categories within a single sensitivity level[18]. This implies that the MLS policy cannot support having a security level for A,$\{\alpha\}$ and for A,$\{\alpha, \beta\}$. Instead, the most restrictive labeling must be defined to protect the data on the system. In contrast, PB-DLP can support an arbitrary number of data fusions.

[17] Two examples of vendors that provide this capability are FireEye (http://www.fireeye.com) and Accellion (http://www.accellion.com/)

[18] See the definition of level statements at `http://selinuxproject.org/page/MLSStatements`

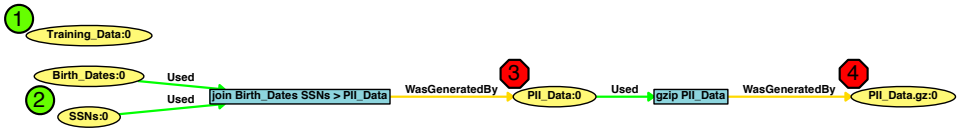

Figure 5: A provenance graph of PII data objects that are first fused and then transformed. The numbers mark DLP decision conditions. Objects marked by green circles *should not* be restricted, while red octagons *should* be restricted. Label-Based DLP correctly handles data resembling PII (1,2) and data transformations (4), but struggles with data fusions (3). Regex-Based DLP correctly identifies data fusions (3), but is prone to incorrect handling of data resembling PII (1) and fails to identify data transformations (4). PB-DLP correctly handles all conditions.

5.4.2 Regex-Based DLP

The majority of DLP software relies on pattern matching techniques to identify sensitive data. While enterprise solutions offer greater sophistication and customizability, their fundamental approach resembles that of Cornell's Spider [7], a forensics tools for identifying sensitive personal data (e.g., credit card or social security numbers). Because it is open source, we make use of Spider as an exemplar application for regex-based DLP.

Regex approaches are prone to *false positives*. Spider is pre-configured with a set of regular expressions for identifying potential PII, e.g., `(\d{3}-\d{2}-\d{4})` identifies a social security number. However, it is common practice for developers to generate and distribute training datasets to aid in software testing (Figure 5, Decision Condition 1). Spider is oblivious to information flows, instead searching for content that bears structural similarity to PII, and therefore would be unable to distinguish between true PII and training data. PB-DLP tracks the propagation of data from its source onwards, and could trivially differentiate between true PII and training sets.

Regex approaches are also prone to *false negatives*. Even after the most trivial data transformations, PII and PCI data is no longer identifiable to the Spider system (Figure 5, Decision Condition 4), permitting its exfiltration. To demonstrate, we generated a file full of random valid social security numbers that Spider was able to identify. We then ran `gzip` on the file and stored it in a second file. Spider was unable to identify the second file, but PB-DLP correctly identified both files as PII since the `gzip` output was derived from a sensitive input.

6 Evaluation

We now evaluate the performance of LPM. Our benchmarks were run on a bare metal server machine with 12 GB memory and 2 Intel Xeon quad core CPUs. The Red Hat 2.6.32 kernel was compiled and installed under 3 different configurations: all provenance disabled (*Vanilla*), LPM scaffolding installed but without an enabled module (*LPM*), and LPM installed with the Provmon module enabled (*Provmon*).

Test Type	Vanilla	LPM	Provmon
Process tests, times in μseconds (smaller is better)			
null call	0.14	0.14 (0%)	0.14 (0%)
null I/O	0.21	0.21 (0%)	0.32 (52%)
stat	1.57	1.6 (2%)	2.8 (78%)
open/close file	2.75	2.42 (-12%)	3.91 (42%)
signal install	0.25	0.25 (0%)	0.25 (0%)
signal handle	1.37	1.29 (-6%)	1.39 (1%)
fork process	380	396 (4%)	401 (6%)
exec process	873	879 (1%)	911 (4%)
shell process	2990	3000 (0%)	3113 (4%)
File and memory latencies in μseconds (smaller is better)			
file create (0k)	11.5	11.2 (-3%)	15.8 (37%)
file delete (0k)	8.51	8.12 (-5%)	11.8 (39%)
file create (10k)	23.4	21.6 (-8%)	28.8 (23%)
file delete (10k)	12.5	12 (-4%)	14.7 (18%)
mmap latency	1062	1053 (-1%)	1120 (5%)
protect fault	0.32	0.3 (-6%)	0.346 (8%)
page fault	0.016	0.016 (0%)	0.016 (0%)
100 fd select	1.53	1.53 (0%)	1.53 (0%)

Table 1: LMBench measurements for LPM kernels. All times are in microseconds. Percent overhead for modified configurations are shown in parenthesis.

6.1 Collection Performance

We used LMBench to microbenchmark LPM's impact on system calls as well as file and memory latencies. Table 1 shows the averaged results over 10 trials for each kernel, with a percent overhead calculation against Vanilla. For most measures, the performance differences between LPM and Vanilla are negligible. Comparing Vanilla to Provmon, there are several measures in which overhead is noteworthy: *stat, open/close, file creation and deletion*. Each of these benchmarks involve LMBench manipulating a temporary file that resides in `/usr/tmp/lat_fs/`. Because an absolute path is provided, before each system call occurs LMBench first traverses the path to the file, resulting in the creation of 3 different provenance events in Provmon's `inode_permission` hook, each of which is transmitted to user space via the kernel relay. While

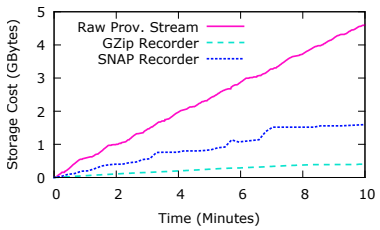

Figure 6: Growth of provenance storage overheads during kernel compilation.

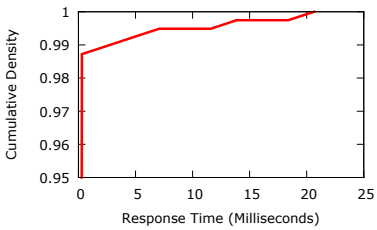

Figure 7: Performance of ancestry queries for objects created during kernel compilation.

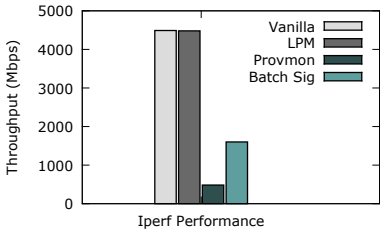

Figure 8: LPM network overhead can be reduced with batch signature schemes.

the overheads seem large for these operations, the logging of these three events only imposes approximately 1.5 microseconds per traversal. Moreover, the overhead for opening and closing is significantly higher than the overhead than reads and writes (*Null I/O*); thus, the higher open/close costs are likely to be amortized over the course of regular system use. A provenance module could further reduce this overhead by maintaining state about past events within the kernel, then blocking the creation of redundant provenance records.

Test	Vanilla	Provmon	Overhead
Kernel Compile	598 sec	614 sec	2.7%
Postmark	25 sec	27 sec	7.5%
Blast	376 sec	390 sec	4.8%

Table 2: Benchmarking Results. Our provenance module imposed just 2.7% overhead on kernel compilation.

To gain a more practical sense of the costs of LPM, we also performed multiple benchmark tests that represented realistic system workloads. Each trial was repeated 5 times to ensure consistency. The results are summarized in Table 2. For the kernel compile test, we recompiled the kernel source (in a fixed configuration) while booted into each of the kernels. Each compilation occurred on 16 threads. The LPM scaffolding (without an enabled module) is not included, because in both tests it imposed less than 1% overhead. In spite of seemingly high overheads for file I/O, Provmon imposes just 2.7% overhead on kernel compilation, or 16 seconds. The Postmark test simulates the operation of an email server. It was configured to run 15,000 transactions with file sizes ranging from 4 KB to 1 MB in 10 subdirectories, with up to 1,500 simultaneous transactions. The Provmon module imposed just 7.5% overhead on this task. To estimate LPM's overhead for scientific applications, we ran the BLAST benchmarks [19], which simulates biological sequence workloads obtained from analysis of hundreds of thousands of jobs from the National Institutes of Health.

[19] See http://fiehnlab.ucdavis.edu/staff/kind/Collector/Benchmark/Blast_Benchmark

For kernel compile and postmark, Provmon outperforms the PASS system, which exacted 15.6% and 11.5% overheads on kernel compilation and postmark, respectively [43]. Provmon introduces comparable kernel compilation overhead to Hi-Fi (2.8%) [48]. It is difficult to compare our Blast results to SPADE and PASS, as both used a custom workload instead of a publicly available benchmark. SPADE reports an 11.5% overhead on a large workload [29], while PASS reports just an 0.7% overhead. Taken as a whole, though, LPM collection either meets or exceeds the performance of past systems, while providing additional security assurances.

6.2 Storage Overhead

A major challenge to automated provenance collection is the storage overhead incurred. We plotted the growth of provenance storage using different recorders during the kernel compilation benchmark, shown in Figure 6. LPM generated 3.7 GB of raw provenance. This required only 450 MB of storage with the Gzip recorder, but provenance cannot be efficiently queried in this format. The SNAP recorder builds an in-memory provenance graph. We approximated the storage overhead through polling the virtual memory consumed by the recorder process in the `/proc` filesystem. The SNAP graph required 1.6 GB storage; the reduction from the raw provenance stream is due to the fact that redundant events did not lead to the creation of new graph components. In contrast, the PASS system generates 1.3 GB of storage overhead during kernel compilation while collecting less information (e.g., shared memory). LPM's storage overheads are thus comparable to other provenance-aware systems.

6.3 Query Performance (PB-DLP)

We evaluated query performance using our exemplar PB-DLP application and LPM's SNAP recorder. The provenance graph that was populated using the routine from the kernel compile benchmark. This yielded a raw provenance stream of 3.7 GB, which was translated by the recorder into a graph of 6,513,398 nodes and 6,754,059

edges. We were then able to use the graph to issue *ancestry queries*, in which the subgraphs were traversed to find the data object ancestors of a given file. Because we did not want ephemeral objects with limited ancestries to skew our results, we only considered the results of objects with more than 50 ancestors.

In the *worst case*, which was a node that had 17,696 ancestors, the query returned in just 21 milliseconds. Effectively, we were able to query object ancestries at line speed for network activity. We are confident that this approach can scale to large databases through precomputation of expensive operations at data ingest, making it a promising strategy for provenance-aware distributed systems; however, we note that these results are highly dependent on the size of the graph. Our test graph, while large, would inevitably be dwarfed by the size of the provenance on long-lived systems. Fortunately, there are also a variety of techniques for reducing these costs. Bates et al. show that the results from provenance graph traversals can be extensively cached when using a fixed security policy, which would allow querying to amortize to a small constant cost [7]. LPM could also be extended to support deduplication [63, 62] and policy-based pruning [6, 13], both of which could further improve performance by reducing the size of provenance graphs.

6.4 Message Commitment Protocol

Under each kernel configuration, we performed *iperf* benchmarking to discover LPM's impact on TCP throughput. iperf was launched in both client and server mode over `localhost`. The client was launched with 16 threads, two per each CPU. Our results can be found in Figure 8. The Vanilla kernel reached 4490 Mbps. While the LPM framework imposed negligible cost compared to the vanilla kernel (4480 Mbps), Provmon's DSA-based message commitment protocol reduced throughput by an order of magnitude (482 Mbps). Through use of `printk` instrumentation, we found that the average overhead per packet signature was 1.2 ms.

This result is not surprising when compared to IPsec performance. IPsec's Authentication Header (AH) mode uses an HMAC-based approach to provide similar guarantees as our protocol. AH has been shown to reduce throughput by as much as half [16]. An HMAC approach is a viable alternative to establish integrity and data origin authenticity and would also fit into the options field, but would require the negotiation of IPsec security associations. Our message commitment protocol has the benefit of being fully interoperable with other hosts, and does not require a negotiation phase before communication occurs. Another option for increasing throughput would be to employ CPU instruction extensions [30] and security co-processor [19] to accelerate the speed of DSA. Yet another approach to reducing our impact on network performance would be to employ a batch signature scheme [11]. We tested this by transmitting a signature over every 10 packets during TCP sessions, and found that throughput increased by 3.3 times to approximately 1600 Mbps. Due to the fact that this overhead may not be suitable for some environments, Provmon can be configured to use Hi-Fi identifiers [48], which are vulnerable to network attack but impose negligible overhead. LPM's impact on network performance is specific to the particular module, and can be tailored to meet the needs of the system.

7 Discussion

Without the aid of provenance-aware applications, LPM will struggle to accurately track dependencies through workflow layer abstractions. The most obvious example of such an abstraction is the copy/paste buffer in windowing applications like Xorg. This is a known side channel for kernel layer security mechanisms, one that has been addressed by the Trusted Solaris project [10], Trusted X [21, 22], the SELinux-aware X window system [37], SecureView [20], and General Dynamics' TVE [21]. Without provenance-aware windowing, LPM will conservatively assume that all files opened for reading are dependencies of the paste buffer, leading to false dependencies. LPM is also unable to observe system side channels, such as timing channels or L2 cache measurements [51], a limitation shared by many other security solutions [18].

Although we have not presented a secure distributed provenance-aware system in this work, LPM provides the foundation for the creation of such a system. In the presented modules, provenance is stored locally by the host and retrieved on an as-needed basis from other hosts. This raises availability concerns as hosts inevitably begin to fail. Availability could be improved with minimal performance and storage overheads through Gehani et al.'s approach of duplicating provenance at k neighbors with a limited graph depth d [27, 28].

Finally, LPM does not address the matter of provenance confidentiality; this is an important challenge that is explored elsewhere in the literature [14, 46]. LPM's Recorders provide interfaces that can be used to introduce an access control layer onto the provenance store.

8 Related Work

While myriad provenance-aware systems have been proposed in the literature, the majority *disclose* provenance within an application [32, 41, 65] or workflow [24, 58]. It

[20] See http://www.ainfosec.com/secureview

[21] See http://gdc4s.com/tve.html

is difficult or impossible to obtain *complete* provenance in this manner. This is because systems events that occur outside of the application, but still effect its execution, will not appear in the provenance record.

The alternative to disclosed systems are *automatic* provenance-aware systems, which collect provenance transparently within the operating system. Gehani et al.'s SPADE is a multi-platform system for eScience and grid computing audiences, with an emphasis on low latency and availability in distributed environments [29]. SPADE's *provenance reporters* make use of familiar application layer utilities to generate provenance, such as polling `ps` for process information and `lsof` for network information. This gives rise to the possibility of *incomplete* provenance due to race conditions. The PASS project collects the provenance of system calls at the virtual filesystem (VFS) layer. PASSv1 provides base functions for provenance collection that observe processes' file I/O activity [43]. Because these basic functions are manually placed around the kernel, there is no clear way to extend PASSv1 to support additional collection hooks; we address this limitation in the modular design of LPM. PASSv2 introduces a Disclosed Provenance API for tighter integration between provenance collected at different layers of abstraction, e.g., at the application layer [44]. PASSv2 assumes that disclosing processes are benign, while LPM provides a secure disclosure mechanism for attesting the correctness of provenance-aware applications. Both SPADE and PASS are designed for benign environments, making no attempt to protect their collection mechanisms from an adversary.

Previous work has considered the security of provenance under relaxed threat models relative to LPM's. In SProv, Hasan et al. introduce *provenance chains*, cryptographic constructs that prevent the insertion or deletion of provenance inside of a series of events [32]. SProv effectively demonstrates the authentication properties of this primitive, but is not intended to serve as a secure provenance-aware system; attackers can still append false records to the end of the chain, delete the whole chain, or disable the library altogether. Zhou et al. consider provenance corruption an inevitability, and show that provenance can detect *some* malicious hosts in distributed environments provided that a critical mass of correct hosts still exist [65]. They later strengthen these assurances through use of provenance-aware software-defined networking [5]. These systems consider only network events, and are unable to speak to the internal state of hosts. Lyle and Martin sketch the design for a secure provenance monitor based on trusted computing [40]. However, they conceptualize provenance as a TPM-aided proof of code execution, overlooking interprocess communication and other system activity that could inform execution results, and therefore offer information that is too coarse-grained to meet the needs of some applications. Moreover, to the best of our knowledge their system is unimplemented.

The most promising model to date for secure provenance collection is Pohly et al.'s Hi-Fi system [48]. Hi-Fi is a Linux Security Module (LSM) that collects *whole-system provenance* that details the actions of processes, IPC mechanisms, and even the kernel itself (which does not exclusively use system calls). Hi-Fi attempts to provide a provenance reference monitor [42], but remains vulnerable to the provenance-aware adversary that we describe in Section 3.2. Enabling Hi-Fi blocks the installation of other LSM's, such as SELinux or Tomoyo, or requires a third party patch to permit module stacking. This blocks the installation of MAC policy on the host, preventing runtime integrity assurances. Hi-Fi is also vulnerable to adversaries in the network, who can strip the provenance identifiers from packets in transit, resulting in irrecoverable provenance. Unlike LPM, Hi-Fi does not attempt to provide layered provenance services, and therefore does not consider the integrity and authenticity of provenance-aware applications.

Provenance collection is a form of information flow monitoring that is related, but fundamentally distinct, from past areas of study. Due to space constraints, our discussion of Information Flow Control (IFC) systems has been relegated to our technical report [8].

9 Conclusion

In this work, we have presented LPM, a platform for the creation of trusted provenance-aware execution environments. Our system imposes as little as 2.7% performance overhead on normal system operation, and can respond to queries about data object ancestry in tens of milliseconds. We have used LPM as the foundation of a provenance-based data loss prevention system that can scan file transmissions to detect the presence of sensitive ancestors in just tenths of a second. The Linux Provenance Module Framework is an exciting step forward for both provenance- and security-conscious communities.

Acknowledgements

We would like to thank Rob Cunningham, Alin Dobra, Will Enck, Jun Li, Al Malony, Patrick McDaniel, Daniela Oliveira, Nabil Schear, Micah Sherr, and Patrick Traynor for their valuable comments and insight, as well as Devin Pohly for his sustained assistance in working with Hi-Fi, and Mugdha Kumar for her help developing LPM SPADE support. This work was supported in part by the US National Science Foundation under grant numbers CNS-1118046, CNS-1254198, and CNS-1445983.

Availability

The LPM code base, including all user space utilities and patches for both Red Hat and the mainline Linux kernels, is available at `http://linuxprovenance.org`.

References

[1] Symantec Data Loss Prevention Customer Brochure. `http://www.symantec.com/data-loss-prevention`.

[2] What's Yours is Mine: How Employees are Putting Your Intellectual Property at Risk. `https://www4.symantec.com/mktginfo/whitepaper/WP_WhatsYoursIsMine-HowEmployeesarePuttingYourIntellectualPropertyatRisk_dai211501_cta69167.pdf`.

[3] R. Aldeco-Pérez and L. Moreau. Provenance-based Auditing of Private Data Use. In *Proceedings of the 2008 International Conference on Visions of Computer Science*, VoCS'08, Sept. 2008.

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

[5] A. Bates, K. Butler, A. Haeberlen, M. Sherr, and W. Zhou. Let SDN Be Your Eyes: Secure Forensics in Data Center Networks. In *NDSS Workshop on Security of Emerging Network Technologies*, SENT, Feb. 2014.

[6] A. Bates, K. R. B. Butler, and T. Moyer. Take Only What You Need: Leveraging Mandatory Access Control Policy to Reduce Provenance Storage Costs. In *7th Workshop on the Theory and Practice of Provenance*, TaPP'15, July 2015.

[7] 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.

[8] A. Bates, D. Tian, K. R. B. Butler, and T. Moyer. Linux Provenance Modules: Trustworthy Whole-System Provenance for the Linux Kernel. Technical Report REP-2015-578, University of Florida CISE Dept, 2015.

[9] M. Bellare, R. Canetti, and H. Krawczyk. Keyed Hash Functions and Message Authentication. In *Proceedings of Crypto'96*, volume 1109 of *LNCS*, pages 1–15, 1996.

[10] M. Bellis, S. Lofthouse, H. Griffin, and D. Kucukreisoglu. Trusted Solaris 8 4/01 Security Target. 2003.

[11] A. Bittau, D. Boneh, M. Hamburg, M. Handley, D. Mazieres, and Q. Slack. Cryptographic protection of TCP Streams (tcpcrypt). `https://tools.ietf.org/html/draft-bittau-tcp-crypt-01`.

[12] D. Boneh, B. Lynn, and H. Shacham. Short Signatures from the Weil Pairing. In C. Boyd, editor, *Advances in Cryptology – ASIACRYPT 2001*, volume 2248 of *Lecture Notes in Computer Science*, pages 514–532. Springer Berlin Heidelberg, 2001.

[13] U. Braun, S. L. Garfinkel, D. A. Holland, K.-K. Muniswamy-Reddy, and M. I. Seltzer. Issues in Automatic Provenance Collection. In *International Provenance and Annotation Workshop*, pages 171–183, 2006.

[14] U. Braun and A. Shinnar. A Security Model for Provenance. Technical Report TR-04-06, Harvard University Computer Science Group, 2006.

[15] D. Catalano, M. Di Raimondo, D. Fiore, and R. Gennaro. Off-line/On-line Signatures: Theoretical Aspects and Experimental Results. In *PKC'08: Proceedings of the Practice and theory in public key cryptography, 11th international conference on Public key cryptography*, pages 101–120, Berlin, Heidelberg, 2008. Springer-Verlag.

[16] S. Chaitanya, K. Butler, A. Sivasubramaniam, P. McDaniel, and M. Vilayannur. Design, Implementation and Evaluation of Security in iSCSI-based Network Storage Systems. In *Proceedings of the Second ACM Workshop on Storage Security and Survivability*, StorageSS '06, pages 17–28, New York, NY, USA, 2006. ACM.

[17] D. D. Clark and D. R. Wilson. A Comparison of Commercial and Military Computer Security Policies. In *IEEE S&P*, Oakland, CA, USA, Apr. 1987.

[18] D. Cock, Q. Ge, T. Murray, and G. Heiser. The Last Mile: An Empirical Study of Some Timing Channels on seL4. In *ACM Conference on Computer and Communications Security*, pages 570–581, Scottsdale, AZ, USA, nov 2014.

[19] J. Dyer, M. Lindemann, R. Perez, R. Sailer, L. van Doorn, and S. Smith. Building the IBM 4758 Secure Coprocessor. *Computer*, 34(10):57–66, Oct 2001.

[20] A. Edwards, T. Jaeger, and X. Zhang. Runtime Verification of Authorization Hook Placement for the Linux Security Modules Framework. In *Proceedings of the 9th ACM Conference on Computer and Communications Security*, CCS'02, 2002.

[21] J. Epstein and J. Picciotto. Trusting X: Issues in Building Trusted X Window Systems -or- WhatâĂŹs Not Trusted About X. In *Proceedings of the 14th Annual National Computer Security Conference*, 1991.

[22] J. Epstein and M. Shugerman. A Trusted X Window System Server for Trusted Mach. In *USENIX MACH Symposium*, pages 141–156, 1990.

[23] S. Even, O. Goldreich, and S. Micali. On-line/off-line Digital Signatures. In *Proceedings on Advances in cryptology*, CRYPTO '89, pages 263–275, New York, NY, USA, 1989. Springer-Verlag New York, Inc.

[24] I. T. Foster, J.-S. Vöckler, M. Wilde, and Y. Zhao. Chimera: AVirtual Data System for Representing, Querying, and Automating Data Derivation. In *Proceedings of the 14th Conference on Scientific and Statistical Database Management*, SSDBM'02, July 2002.

[25] V. Ganapathy, T. Jaeger, and S. Jha. Automatic placement of authorization hooks in the linux security modules framework. In *Proceedings of the 12th ACM Conference on Computer and Communications Security*, CCS '05, pages 330–339, New York, NY, USA, 2005. ACM.

[26] C.-z. Gao and Z.-a. Yao. A Further Improved Online/Offline Signature Scheme. *Fundam. Inf.*, 91:523–532, August 2009.

[27] A. Gehani, B. Baig, S. Mahmood, D. Tariq, and F. Zaffar. Fine-grained Tracking of Grid Infections. In *Proceedings of the 11th IEEE/ACM International Conference on Grid Computing*, GRID'10, Oct 2010.

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

[29] 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.

[30] S. Gueron and V. Krasnov. Speed Up Big-Number Multiplication Using Single Instruction Multiple Data (SIMD) Architectures, June 7 2012. US Patent App. 13/491,141.

[31] C. Hanson. SELinux and MLS: Putting The Pieces Together. In *In Proceedings of the 2nd Annual SELinux Symposium*, 2006.

[32] R. Hasan, R. Sion, and M. Winslett. 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, Feb. 2009.

[33] 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.

[34] T. Jaeger, A. Edwards, and X. Zhang. Consistency Analysis of Authorization Hook Placement in the Linux Security Modules Framework. *ACM Trans. Inf. Syst. Secur.*, 7(2):175–205, May 2004.

[35] T. Jaeger, R. Sailer, and U. Shankar. PRIMA: Policy-reduced Integrity Measurement Architecture. In *Proceedings of the 11th ACM Symposium on Access Control Models and Technologies*, SACMAT '06, pages 19–28, New York, NY, USA, 2006. ACM.

[36] S. Kent and R. Atkinson. RFC 2406: IP Encapsulating Security Payload (ESP). 1998.

[37] D. Kilpatrick, W. Salamon, and C. Vance. Securing the X Window System with SELinux. Technical report, Jan. 2003.

[38] L. Lamport. Time, Clocks, and the Ordering of Events in a Distributed System. *Commun. ACM*, 21(7):558–565, July 1978.

[39] H. Luhn. Computer for Verifying Numbers, Aug. 23 1960. US Patent 2,950,048.

[40] 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.

[41] P. Macko and M. Seltzer. A General-purpose Provenance Library. In *4th Workshop on the Theory and Practice of Provenance*, TaPP'12, June 2012.

[42] 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.

[43] 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.

[44] 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.

[45] D. Nguyen, J. Park, and R. Sandhu. Dependency Path Patterns As the Foundation of Access Control in Provenance-aware Systems. In *Proceedings of the 4th USENIX Conference on Theory and Practice of Provenance*, TaPP'12, pages 4–4, Berkeley, CA, USA, 2012. USENIX Association.

[46] Q. Ni, S. Xu, E. Bertino, R. Sandhu, and W. Han. An Access Control Language for a General Provenance Model. In *Secure Data Management*, Aug. 2009.

[47] J. Park, D. Nguyen, and R. Sandhu. A Provenance-Based Access Control Model. In *Proceedings of the 10th Annual International Conference on Privacy, Security and Trust (PST)*, pages 137–144, 2012.

[48] 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.

[49] J. Postel. RFC 791: Internet protocol. 1981.

[50] A. C. Revkin. Hacked E-mail is New Fodder for Climate Dispute. *New York Times*, 20, 2009.

[51] T. Ristenpart, E. Tromer, H. Shacham, and S. Savage. Hey, You, Get Off of My Cloud: Exploring Information Leakage in Third-Party Compute Clouds. In *Proceedings of the 16th ACM Conference on Computer and Communications Security (CCS'09)*, pages 199–212, Chicago, IL, USA, Oct. 2009. ACM.

[52] R. Sailer, X. Zhang, T. Jaeger, and L. van Doorn. Design and Implementation of a TCG-based Integrity Measurement Architecture. In *SSYM'04: Proceedings of the 13th conference on USENIX Security Symposium*, pages 16–16, Berkeley, CA, USA, 2004. USENIX Association.

[53] C. Sar and P. Cao. Lineage File System. http://crypto.stanford.edu/~cao/lineage.html.

[54] J. Seibert, G. Baah, J. Diewald, and R. Cunningham. Using Provenance To Expedite MAC Policies (UPTEMPO) (Previously Known as IPDAM). Technical Report USTC-PM-015, MIT Lincoln Laboratory, October 2014.

[55] A. Shamir and Y. Tauman. Improved Online/Offline Signature Schemes. In J. Kilian, editor, *Advances in Cryptology — CRYPTO 2001*, volume 2139 of *Lecture Notes in Computer Science*, pages 355–367. Springer Berlin / Heidelberg, 2001.

[56] D. Tariq, B. Baig, A. Gehani, S. Mahmood, R. Tahir, A. Aqil, and F. Zaffar. Identifying the Provenance of Correlated Anomalies. In *Proceedings of the 2011 ACM Symposium on Applied Computing*, SAC '11, Mar. 2011.

[57] The Netfilter Core Team. The Netfilter Project: Packet Mangling for Linux 2.4. http://www.netfilter.org/, 1999.

[58] J. Widom. Trio: A System for Integrated Management of Data, Accuracy, and Lineage. Technical Report 2004-40, Stanford InfoLab, Aug. 2004.

[59] World Wide Web Consortium. PROV-Overview: An Overview of the PROV Family of Documents. `http://www.w3.org/TR/prov-overview/`, 2013.

[60] C. Wright, C. Cowan, J. Morris, S. Smalley, and G. Kroah-Hartman. Linux Security Module Framework. In *Ottawa Linux Symposium*, page 604, 2002.

[61] C. Wright, C. Cowan, S. Smalley, J. Morris, and G. Kroah-Hartman. Linux security modules: General security support for the linux kernel. In *Proceedings of the 11th USENIX Security Symposium*, pages 17–31, Berkeley, CA, USA, 2002. USENIX Association.

[62] 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.

[63] Y. Xie, K.-K. Muniswamy-Reddy, D. D. E. Long, A. Amer, D. Feng, and Z. Tan. Compressing Provenance Graphs, June 2011.

[64] X. Zhang, A. Edwards, and T. Jaeger. Using CQUAL for Static Analysis of Authorization Hook Placement. In *Proceedings of the 11th USENIX Security Symposium*, 2002.

[65] W. Zhou, Q. Fei, A. Narayan, A. Haeberlen, B. T. Loo, and M. Sherr. Secure Network Provenance. In *Proceedings of the 23rd ACM Symposium on Operating Systems Principles*, SOSP'11, Oct. 2011.

[66] W. Zhou, M. Sherr, T. Tao, X. Li, B. T. Loo, and Y. Mao. Efficient Querying and Maintenance of Network Provenance at Internet-Scale. In *ACM SIGMOD International Conference on Management of Data (SIGMOD)*, June 2010.