Collecting Provenance via the Xen Hypervisor

Peter Macko
Harvard University

Marc Chiarini
Harvard University

Margo Seltzer
Harvard University

Abstract

The Provenance Aware Storage Systems project (PASS) currently collects system-level provenance by intercepting system calls in the Linux kernel and storing the provenance in a stackable filesystem. While this approach is reasonably efficient, it suffers from two significant drawbacks: each new revision of the kernel requires reintegration of PASS changes, the stability of which must be continually tested; also, the use of a stackable filesystem makes it difficult to collect provenance on root volumes, especially during early boot. In this paper we describe an approach to collecting system-level provenance from virtual guest machines running under the Xen hypervisor. We make the case that our approach alleviates the aforementioned difficulties and promotes adoption of provenance collection within cloud computing platforms.

1 Introduction

The PASS project [9] has successfully addressed the task of collecting and processing system-level provenance [8]. While the implementation of PASS has served as a useful prototype, its continued evolution is unsustainable, because the implementation requires reintegration and testing with every new Linux kernel version. For example, our stackable filesystem, *Lasagna*, is mounted on top of a traditional filesystem such as Ext3. Lasagna transparently interfaces with the Linux Virtual FileSystem (VFS) to pass analyzed provenance back and forth from/to the lower filesystem. This is convenient because it enables operations on provenance without the need to modify the kernel implementation of the underlying filesystem. However, Lasagna remains sensitive to changes in the VFS interface and other kernel facilities upon which it relies. One of the most important goals for provenance collection is to be certain that provenance is true and reliable. PASS users can ill afford to have a collection mechanism that records partial or erroneous information (or worse, causes a crash) simply because the system was upgraded to a new kernel. Continuing this research makes finding a portable method of provenance collection vital. To that end, we are exploring hypervisor-based provenance collection.

In this paper, we propose an approach for extending Xen [6] to transparently collect operating system-level provenance from its guest VMs, without requiring any modifications to the guest kernel. Although we expect our approach to be implementable across a variety of hypervisors, we chose to prototype in Xen because it is a mature virtualization technology that is under active development; its code is fairly clean and minimal; it is licensed under GNU GPLv2, which allows it to be changed and distributed freely; and it is already used by several increasingly popular providers of Infrastructure as a Service (IaaS). Our implementation is still a work in progress, but it has already exhibited great promise.

2 Background

2.1 The PASS Architecture

Figure 1 shows the PASS architecture[1]. The *interceptor* is a set of system call hooks that extract arguments and other necessary information from kernel data structures, passing them to the *observer*. Currently, PASS intercepts `execve`, `fork`, `exit`, `read`, `readv`, `write`, `writev`, `mmap`, `open`, `pipe`, and the kernel operation `drop_inode`. These calls are sufficient to capture the rich ancestry relationships between Linux files, pipes, and processes. This raw "proto-provenance" goes to the *observer*, which translates proto-provenance into provenance records. For example, when a process *P* reads a file *A*, the observer generates a record that includes the fact that *P* depends on *A*. The *analyzer* then processes

[1] We strongly urge the reader to consult our previous paper [9] that describes the PASS project in more detail.

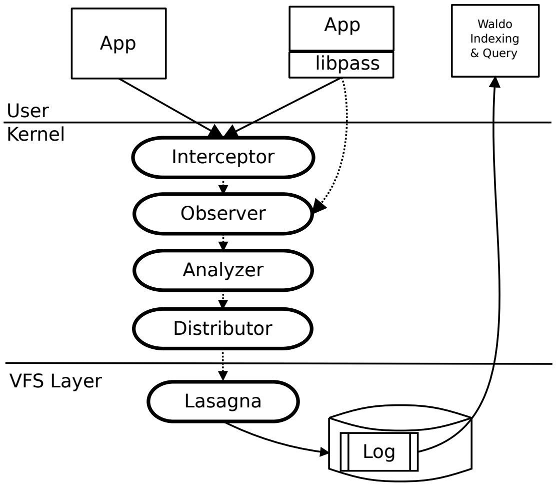

Figure 1: A diagram of the PASS Architecture

the stream of provenance records to eliminate duplicates, ensuring that cyclic dependencies do not arise.

The *distributor* caches provenance for objects that are not persistent from the kernel's perspective, such as pipes, processes and application-specific objects (e.g., a browser session or data set) until they need to be materialized on disk. *Lasagna* is the provenance-aware file system that stores provenance records along with the data. Internally, it writes provenance to a log. The log format ensures consistency between the provenance and data appearing on disk. Finally, *Waldo* is a user-level daemon that reads provenance records from the log and stores them in a database. Waldo is also responsible for accessing the database on behalf of a provenance query engine.

2.2 The Xen Architecture

The Xen hypervisor (Figure 2) is a minimal kernel that normally runs on the "bare metal" of a system. The hypervisor oversees multiple virtual machines known as domains, with one domain (Dom0) having special privileges. Dom0 serves as the administrative center of a Xen system, is the first domain started on boot, and is the only domain with direct access to real hardware. It runs an operating system (usually Linux) that has been modified to communicate with the hypervisor. Guest VMs, also referred to as *DomUs*, rely on the hypervisor for privileged operations such as communicating with devices. A DomU can run an operating system with one of two kinds of kernels: a para-virtualized (PV) kernel or an unmodified kernel. A PV kernel is modified to perform privileged operations (e.g., a page table update) only via *hypercalls*. Similar to an application issuing a system call, a hypercall is a software trap from a domain to the hypervisor. If the CPU supports virtualization extensions, such as Intel VT-x, the DomU can use an unmodified kernel, which we refer to as a Hardware Virtual Machine (HVM).

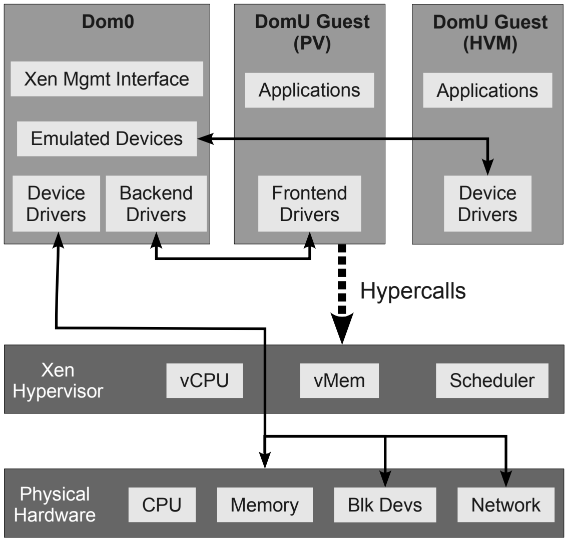

Figure 2: A diagram of the Xen Architecture

3 Approach

In this section we describe our approach to modifying the Xen hypervisor such that we can collect provenance from running guest kernels.

3.1 Assumptions

Our approach assumes that the target VM (the one from which we will collect provenance) is running a PV kernel, but targeting an unmodified kernel (such as Windows) should be possible as well. Linux, Minix, GNU Hurd, OpenSolaris, NetBSD, and FreeBSD all have officially supported PV kernels. The remainder of this paper assumes that we are running a Linux guest kernel; the same approach is easily adapted to the other types of guests. We also assume that the Dom0 and DomU kernels and the Xen hypervisor itself are uncompromised, such that we may trust and verify any generated provenance. Finally, we assume that the debug symbols for the given build of the kernel are available. The reason for this will become apparent in the design section.

3.2 Design

Our approach intercepts DomU system calls by placing an appropriate hook in Xen's `syscall_enter` mechanism. This hook provides the interceptor with the system call number and its numerical arguments. The interceptor in turn determines which of the arguments are numbers and which are pointers to data structures and strings in user space by looking up the system call in a table. It then proceeds to create a system call record with the following fields:

- system call number
- numerical arguments
- referenced strings and structs
- a placeholder for the return value
- the task group ID (TGID) of the calling process as returned by the `getpid()` system call.
- the current working directory (CWD), including its volume ID, if the system call references a string that contains a file name, such as in the case of `creat()` and `open()`
- the environment variables, if necessary

The interceptor determines fields such as TGID or CWD by using the supplied kernel debug symbols in order to traverse the guest kernel's internal data structures and to pull out the appropriate values. Similarly, it uses the debug information to determine the sizes of the referenced structs that it needs to copy out of user space. The interceptor places the record in a ring buffer, blocking if there is not enough free space. Finally, another hook in Xen's `do_iret()` code places the return value of the system call into the appropriate field.

Concurrently, a user space daemon runs in a privileged domain (Dom0 or a special "provenance-processing domain") to periodically:

1. consume records from the ring buffer,
2. process them using a provenance *analyzer*,
3. output provenance logs,
4. and feed the logs into Waldo (the online provenance database).

We can run this process on a dedicated CPU/core in a multiprocessor/multicore system to improve guest performance. The user space daemon, as well as other supporting software, communicates with Xen's provenance subsystem via a dedicated hypercall, which is accessible only in the privileged domain.

4 Advantages

Our proposed design starts to collect provenance immediately after the guest is powered on. This provides us with provenance from the root filesystem and from the early stages of the boot process, which is currently difficult to do with our in-kernel PASS approach. Early provenance can then be used for diverse tasks such as system configuration troubleshooting, startup optimization, and intrusion analysis. This is in addition to the benefits of system-level provenance already demonstrated in previous research [8, 9].

The decoupling of the interceptor from the provenance analyzer allows us to move the CPU-intensive analysis onto another CPU or core, so that the single-thread performance of the guest will be only minimally affected. This is important because average performance of the hypervisor is certain to be affected negatively by provenance collection, especially if the guest VMs are hosting I/O-intensive applications. Once we have implemented a provenance-aware Xen prototype, we will be in a better position to evaluate and address any resulting impact upon performance.

We expect that adapting the interceptor to new guest kernel releases will not be difficult, because we will have to account only for major changes to basic kernel data structures. We will not need to worry about most other changes that are tangential to the kernel paths from which we collect provenance. More important, we will be protected from worrying about changes to the API referenced by stackable filesystems; these changes tend to happen often and to be fairly substantial. Also, while the Xen implementation is sure to increase in complexity, changes occur at a slower rate than that of UNIX-derived kernels. The execution path for trapping system calls in Xen is relatively insulated even from fundamental changes to the guest kernels; this will remain true for as long as operating systems offer services via a system call interface.

5 Pitfalls

One obstacle of our approach is that the interceptor requires a large ring buffer to hold the records of all system calls that have not yet been consumed by the analyzer. A typical Linux guest generates several tens of read/write system calls per second that involve standard input and output streams. We cannot easily filter them out at the interceptor, because these streams may have been redirected previously to a file or a pipe. In addition to consuming a large amount of memory, buffer operations typically increase the pressure on processor caches and add traffic to the CPU interconnect (due to cache coherence if the provenance is processed on a different CPU than the one on which the provenance was collected). The amount of recorded data can be vastly reduced by moving the analysis process from the proposed user space daemon directly into the path of the intercepted system calls. The tradeoff for a reduction in space and cache pressure would be a reduction in single-thread performance. This balance will be the subject of future study.

Another difficulty with our design comes from the fact that the interceptor can see file paths only as strings. A path can cross multiple mount points, and include any number of symbolic links and hard links. As a result, multiple distinct path strings can refer to the same object. We can address this issue by examining the file system state before the guest is powered on and then updating the state by simulating the effect of relevant system calls (e.g., `mkdir()` and `link()`). This is an expensive solution that would not be suitable for most environments.

PASS solves this problem by using inode numbers to uniquely identify files in the filesystem, making sure that if an inode number is reused, the given file has a different associated provenance identifier. We may be able to implement this solution by mapping file paths to inodes using data structures in the guest kernel's VFS layer, which we can access using the kernel's debug symbols. This would be done on a per-mounted-filesystem basis, because inodes are only meaningful in the context of a specific filesystem.

Finally, a hardware failure or a software bug in Xen or the Dom0 kernel could cause an irreversible loss of system call records that have not yet been processed - and thus a loss of several seconds of provenance. In our PASS implementation we used a *write-ahead protocol* to prevent any loss of provenance. There is currently no elegant way of handling this issue. This might be fixed by placing the ring buffer into NVRAM (battery-backed RAM should be sufficient), or by adding a UPS system and flushing the buffer to disk on a power failure or a system panic as done in the Panasas Parallel File System [12].

6 Adoption

There are several arguments to be made in favor of adopting hypervisor-based provenance collection methods. The strongest argument is the growing demand for cloud computing services, most of which use virtualization to achieve economies of scale and to improve resource utilization efficiency. Many cloud providers currently use Xen or allow Xen as their virtualization layer. These include Amazon EC2 [1], Rackspace Cloud [4], Cloud.com, Flexiant (Flexiscale) [3], and Eucalyptus [2].

Another key to adoption is reduced complexity, both on the developer and the client side. For developers, working inside the Xen hypervisor will be easier and less error-prone than modifying multiple native kernels to collect provenance. Developers will still require knowledge of kernel data structures but they will only be debugging code in a single hypervisor and its associated modules, instead of debugging frequently changing kernel code. For system administrators and other users, configuration of provenance collection and the subsequent querying and presentation of provenance from multiple virtual machines can be controlled from a unified interface.

7 Related Work

As of this writing, the authors are unaware of any published work that describes attempts to collect provenance using a hypervisor. However, there have been several projects that enable resource *introspection* of a virtualized guest, often for security purposes.

For example, VMWare's EPSEC [5] provides a library and API for introspection into file activity at the hypervisor layer. Similarly, XenAccess [10] is a monitoring library for operating systems running on Xen. It provides virtual memory introspection and virtual disk monitoring capabilities, allowing monitor applications to safely and efficiently access the memory state and disk activity of a virtual machine.

Introspection approaches are not sufficient to collect provenance from all areas of the operating system. This requires system call interception that tracks the creation, access, and destruction of processes, files, pipes, and (ideally) sockets.

Two projects come closer to providing these capabilities: Lares [11] and Ether [7]. Lares enables a specialized security VM to insert hooks inside a Windows guest VM running under Xen. This allows active monitoring of guest behavior and interception and denial of proscribed events. Lares does not specifically intercept the required system calls outlined above, but could be modified to do so. Also, insertion of hooks requires installation of a trusted kernel driver in the guest OS. This may be a good path to follow in our own implementation.

The Ether project is most similar to our own proposal, although it is used for malware analysis instead of provenance collection. It resides in two parts of the Xen architecture: a hypervisor component detects important events inside the DomU target, including instruction execution, system call execution, memory writes, and context switches; a user space component in Dom0 determines which processes and events in the guest should be monitored. This component also contains analysis logic that can perform tasks such as translating a system call number into a system call name or extracting the content of system call arguments based on their type. We expect that a more thorough study of Ether will yield several useful techniques.

8 Conclusion

We have proposed a method for collecting system-level provenance from virtual machines running under the Xen Hypervisor. We believe that this is the only viable longterm approach to collecting system provenance, and that it also provides benefits above and beyond those available from other provenance systems. Some technical hurdles remain, but provenance collection via hypervisor is undoubtedly the future.

9 Acknowledgments

The authors would like to thank the other members of the PASS group at Harvard University for their comments and insights on this research.

10 Availability

A working prototype is not yet available. However, readers are encouraged to periodically check the website below for news and updates.

`http://www.eecs.harvard.edu/syrah/pass/`

References

[1] Amazon elastic compute cloud. `http://aws.amazon.com/ec2/`. Retreived 2011-04-04.

[2] Eucalyptus enterprise edition 2.0 datasheet. `http://www.eucalyptus.com/themes/eucalyptus/pdf/Eucalyptus_EEE_DS.pdf`. Retreived 2011-04-04.

[3] Flexiant – utility computing on demand. `http://www.flexiant.com/`.

[4] Rackspace cloud computing. `http://www.rackspace.com/cloud/`. Retreived 2011-04-04.

[5] Vmware vshield endpoint. `http://www.vmware.com/files/pdf/vmware-vshield-endpoint-ds-en.pdf`. Retreived 2011-04-04.

[6] BARHAM, P., DRAGOVIC, B., FRASER, K., HAND, S., HARRIS, T., HO, A., NEUGEBAUER, R., PRATT, I., AND WARFIELD, A. Xen and the art of virtualization. *ACM SIGOPS Operating Systems Review 37*, 5 (2003), 164–177.

[7] DINABURG, A., ROYAL, P., SHARIF, M., AND LEE, W. Ether: malware analysis via hardware virtualization extensions. *Proceedings of the 15th ACM conference on Computer and communications security* (2008), 51–62.

[8] HOLLAND, D. A., SELTZER, M. I., BRAUN, U., AND MUNISWAMY-REDDY, K. PASSing the provenance challenge. *Concurr. Comput. : Pract. Exper. 20* (Apr. 2008), 531–540.

[9] MUNISWAMY-REDDY, K., BRAUN, U., HOLLAND, D. A., MACKO, P., MACLEAN, D., MARGO, D., SELTZER, M., AND SMOGOR, R. Layering in provenance systems. In *2009 USENIX Annual Technical Conference* (San Diego, California, 2009).

[10] PAYNE, B. D., CARBONE, M., AND LEE, W. Secure and flexible monitoring of virtual machines. In *Computer Security Applications Conference, Annual* (Los Alamitos, CA, USA, 2007), vol. 0, IEEE Computer Society, pp. 385–397.

[11] PAYNE, B. D., CARBONE, M., SHARIF, M., AND LEE, W. Lares: An architecture for secure active monitoring using virtualization. In *Security and Privacy, IEEE Symposium on* (Los Alamitos, CA, USA, 2008), vol. 0, IEEE Computer Society, pp. 233–247.

[12] WELCH, B., UNANGST, M., ABBASI, Z., GIBSON, G., MUELLER, B., SMALL, J., ZELENKA, J., AND ZHOU, B. Scalable performance of the panasas parallel file system. *Proceedings of the 6th USENIX Conference on File and Storage Technologies* (2008), 2:1–2:17.