Proper: Privileged Operations in a Virtualised System Environment

Steve Muir, Larry Peterson, Marc Fiuczynski, Justin Cappos, John Hartman
Princeton University and the University of Arizona
{smuir,llp,mef}@cs.princeton.edu, {justin,jhh}@cs.arizona.edu

Abstract

Virtualised systems have experienced a resurgence in popularity in recent years, particularly in supporting a large number of independent services on a single host. This paper describes our work designing and implementing Proper, a service running on the PlanetLab system, that allows other services to perform privileged operations in a safe, controlled manner. We describe how implementing such a system in a traditional UNIX environment is non-trivial, and discuss the practical use of Proper.

1 Introduction

Operating systems face a fundamental tension between providing isolation and sharing among applications—they support the illusion that each application has the physical machine to itself, yet also allow those applications to share objects (files, pipes, etc.) with each other. General-purpose OSes typically provide a weak form of isolation (the process abstraction) with relatively unrestricted sharing between applications. In contrast, virtual machine monitors (VMMs) strive to provide strong performance isolation and privacy between virtual machines (VMs), and provide no more support for sharing between VMs than the network provides between physical machines.

The point on the design spectrum supported by any given system depends on the workload it is designed to support. General-purpose OSes run multiple applications on behalf of a single user, making it natural to favor sharing over isolation. Similarly, VMMs are often designed to allow a single machine to host multiple independent applications, possibly on behalf of independent organizations. In such a scenario, the applications have no need to share information and but require a predictable fraction of the physical machine's resources—hence, VMMs heavily favor isolation over sharing.

This paper investigates an alternative design point, motivated by the need for VMs to interact with each other in well-defined and controlled ways. All systems provide a means by which isolated components interact, but we are particularly interested in the problem of 'unbundling' the management of a set of VMs from the underlying VMM. To enable multiple *management services*, it is necessary to 'poke holes' in the isolation barriers between VMs. These holes allow one VM to see and manipulate objects such as files and processes in another VM, providing a natural means for one VM to help manage another.

Toward this end, this paper describes Proper, a new '**Pr**ivileged **Oper**ations' mechanism that VMs can use to poke the holes that they need. Proper is straightforward to implement on a UNIX-based VMM such as that used in PlanetLab, and enables useful management services that run today on PlanetLab.

2 The PlanetLab Virtualised Environment

Our work is conducted in the context of PlanetLab, a global network of 500+ PCs used by researchers at over 250 institutions to develop new network services and conduct network-based experiments. Each PlanetLab node runs a modified Linux kernel, which provides a virtualised Linux environment for each user, with 30–40 users active on each node at any given time. Although Proper was originally developed as a component of the PlanetLab node infrastructure we believe that it is also applicable to other virtualised systems e.g., VMWare, Xen, Denali.

PlanetLab provides each user with one or more *slices*—a set of resources bound to a virtual Linux environment and associated with some number of PlanetLab nodes. Each node runs a Linux kernel modified with two packages: Vservers [3] supports multiple Linux VMs running on a single kernel instance, while CKRM provides a resource management framework. Each PlanetLab slice is instantiated as a combination of a Vserver, providing *namespace isolation*, and a CKRM class, providing *performance isolation*.

Namespace isolation is the property of a virtualised system where each VM executes in its own namespace, and is required so that each VM can configure its own namespace without interference with or by other VMs e.g., a VM should be able to install packages in its own filesystem without concern as to whether another VM requires a different version. Performance isolation between two VMs allows each VM to consume and manage resources e.g., CPU, physical memory, network bandwidth,

Operation	Description
`open_file(name, options)`	Open a restricted file
`set_file_flags(name, flags)`	Change file flags
`get_file_flags(name, flags)`	Get file flags
`mount_dir(source, target, options)`	Mount one directory onto another
`unmount(source)`	Unmount a previously-mounted directory
`exec(context, cmd, args)`	Execute a command in the context (slice) given
`wait(childid)`	Wait for an executed command to terminate
`create_socket(type, address)`	Create a restricted socket
`bind_socket(socket, address)`	Bind a restricted socket

Table 1: Operations supported by Proper 0.3

without affecting other VMs.

Although namespace isolation is essential in order to support multiple VMs on a single physical system, it presents a barrier to cooperation between those VMs. Furthermore, it also prevents VMs from getting full visibility into the details of the host system, thus limiting monitoring and system management. Hence the need to provide a means for selected VMs to break their namespace isolation in carefully controlled ways.

3 Architecture and Implementation

Proper provides unprivileged slices with access to privileged operations in a controlled manner. Here we present the client-visible architecture of Proper, which we believe to be equally applicable to any virtualised system, such as Xen [1], Denali [8], etc., not just the PlanetLab environment.

3.1 The Proper Client API

Proper enables two different types of *'VM escapes'*: inter-VM communication, and access to resources outside the VM e.g., VMM internals, such as VM configuration. While access to VMM internals requires cooperation by the VMM, certain types of inter-VM communication may be possible without a service like Proper. For example, most VMMs permit filesystems between VMs using NFS or SMB. However, the VMM may be able to support direct filesystem sharing in more flexible and/or higher-performing ways, so Proper provides a high-level filesystem-sharing operation using a VMM-specific implementation.

The operations supported by Proper are chosen according to the following principles:

1. Client interactions with Proper should be minimised—once an initial request has been completed subsequent operations should only require interactions with the host VM.

2. Proper operations should be compatible with equivalent intra-VM operations—opening a file through Proper should yield a standard file descriptor.

3. The API should be as general and flexible as possible—there should be a single `open` operation rather than separate operations to read and write files, pipes and devices.

It is important to minimise the overhead due to communication between the client application and the Proper service—a single request grants access to a restricted object, and subsequent requests use the host VM's standard data operations. For example, when opening a file it is acceptable to require the initial 'open' request be sent to Proper, but forcing every read/write request to also use Proper would impose a performance penalty. We discuss such effects later in Section 4.2.

Table 1 shows the current Proper API, with each group of operations briefly discussed in the following section. The API is primarily driven by the requirements of PlanetLab users, and is thus not exhaustive, but gives a good idea of the type of operations we envision being supported.

3.2 Implementing Proper

As well as describing our implementation of Proper, we present a strawman implementation of each operation group that illustrates how one might implement Proper on another virtualised system. We assume that Proper exists as a component of the VMM 'root' context that receives requests from clients running in a VM; alternatives are possible, such as modifying the OS inside each VM to be 'Proper-aware', but we believe that such modifications are unnecessary.

3.2.1 File Operations

The `open_file` operation allows an application to access a file outside its own VM, while the `get_` and `set_file_flags` operations support manipulation of restricted file flags i.e., flags that may be used by the VMM to support copy-on-write sharing of files between VMs, and hence must not be modifiable within the VM.

Opening a file can be supported in UNIX-like VMs by exploiting the similarities between file descriptors and network sockets: Proper opens the restricted file and

passes the data to the client using a network socket. Unfortunately, while the client can readily use a socket in place of a file descriptor for reading and writing data, other operations may reveal that the 'file' is in fact a socket. In PlanetLab we exploit the fact that each VM runs atop the same kernel as Proper to achieve our transparency goal: Proper passes the opened file descriptor, which is indistinguishable from that obtained if the client opened the file directly, to the client using a UNIX domain socket.

3.2.2 Directory Operations

A common requirement in virtualised systems is that two VMs share parts of their filesystem e.g., when one wishes to manage the other's filesystem. As stated earlier, sharing can often be accomplished directly between two VMs using a protocol such as NFS, but the VMM may also support a direct sharing mechanism. For example, in PlanetLab all VMs exist as disjoint subtrees of a single filesystem, so one can take advantage of Linux's 'bind-mount' facility to graft (bind) one subtree onto another. This feature is used by the Stork configuration manager (see Section 4.1) to manage client VMs.

3.2.3 Process Execution

A client application may wish to create processes outside of its own VM: to perform some privileged task in the 'root' context e.g., create a new VM, or to act as, say, a remote access service for another VM by creating processes in that VM in response to authenticated network requests. Both cases may require that long-running processes are created e.g., a remote login shell, so the `exec` operation is actually performed asynchronously, and client use a `wait` operation to wait until the process terminates. The client passes file descriptors for use as standard I/O so Proper need not be involved in data passing between the child process and the client; alternatively, sockets could be used as in the generic `open_file` implementation described above.

3.2.4 Network Socket Manipulation

The final class of operations supported by Proper allow clients within a VM to request privileged access to the VMM's network traffic. For example, a VMM may multiplex a single network interface onto multiple per-VM interfaces, with each VM only being able to 'see' its own traffic. A traffic monitoring or auditing program, running in an unprivileged VM, wishes to receive a copy of every packet sent or received on the physical interface, so uses Proper to create a privileged 'raw' socket—Proper verifies that the auditing service is authorised to do so before configuring the network subsystem to dispatch copies of every packet to the raw socket.

Similarly, an application may wish to bind to a specific network port in order to receive all packets sent to that port on this machine—since port space is often shared by all VMs using a physical network interface this requires coordination with the VMM. In PlanetLab we allow slices to bind ports above 1024 in a first-come, first-served fashion, but require use of Proper to bind restricted ports below 1024, thus ensuring that only the specific slice authorised to bind a particular port can do so.

4 Experiences Using Proper on PlanetLab

Proper has been deployed and in-use on PlanetLab for about 6 months, supporting a number of services. These include both core infrastructure e.g., network traffic auditing, and user services. Most importantly, Proper has enabled elimination of 'privileged' slices that were used in the previous version of the PlanetLab system to perform many of those functions.

4.1 An Example Service: Stork

Stork is a PlanetLab service that provides package management functionality to other services. It allows users to associate a set of packages with their slice and takes care of downloading and installing the software into the slices and keeping it up-to-date. Stork allows package contents to be shared between slices, reducing the software footprint on a PlanetLab node.

Stork is responsible for downloading packages from a package repository, maintaining a per-node package cache, and installing packages in client slices. When a package is installed in a client slice the contents must be transferred from Stork to the client. One way to do this is over a socket, but this is inefficient and prevents sharing of files between slices (see below). Instead, Stork mounts the appropriate package directory read-only into the client's filesystem using `mount_dir`. This gives the client access to the files in the package directory without being able to modify the directory structure.

Of course, many slices may install the same packages, making it desirable to share the package contents between the slices. However, modifications made by one slice should not be visible to any other. Ideally, each slice should have a copy-on-write version of each file; unfortunately, this functionality is not available in PlanetLab. Instead, Stork relies on sharing files read-only between slices. Files that may be modified are not shared; typically these are a few configuration files for each package. Most files are shared, dramatically reducing the amount of disk space required to install packages in slices.

Stork makes shared files read-only using the Proper `set_file_flags` operation to set the NOCHANGE flag on the files when it unpacks them. When unpacking a package the Stork client creates hard links to the read-only files to put the files in the proper locations inside the client filesystem; writable files are copied instead of linked. The installation scripts are then run in the client

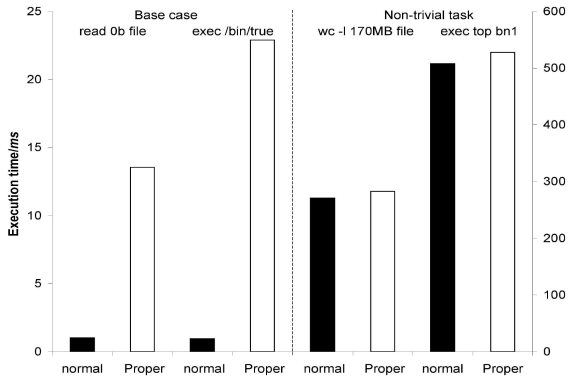

Figure 1: Overhead of Proper for various tasks

slice, and the package directory is unmounted. At this point the client slice has either hard links to or copies of the files in Stork's file system. The client is prevented from modifying shared (linked) files by the NOCHANGE flag, but can remove (unlink) the file to perform a private replacement.

4.2 Evaluation of Proper Overhead

One possible concern with our implementation of Proper is that invoking privileged operations using RPC may impose significant overhead. To address this we measured the overhead for a couple of operations, `open_file` and `exec`, for both trivial base cases and more realistic tasks—reading a large file and running a complex program respectively.

Figure 1 shows these results (measured on an idle 3GHz Pentium 4 with 1.25GB RAM): while the base case overhead, shown in the left half, is about 12–22*ms*, this is negligible for the non-trivial tasks, as shown on the right side of the graph. This overhead consists of two components: a client-side RPC overhead, typically 7.5*ms*, which is operation-independent; and an operation-dependent server-side latency of an additional 5–15*ms*.

5 Related Work

Much existing work on virtualisation techniques has some degree of relevance to Proper. Systems such as *VMWare* [7], *Xen* [1], *Denali* [8], *Zap* [4] and *Solaris Zones* [6] all provide users with virtualised environments, and often utilise a system-specific method for providing direct access from the VM to the VMM. For example, VMWare allows users to install extensions to popular 'guest' OSes, such as Windows, that permit direct access to host files in a similar manner to the bind mounts facilitated by Proper, while Xen allows multiple VMs to access physical devices through a special 'privileged' VM that runs unmodified Linux drives.

Zap and *Zones* are perhaps closest to PlanetLab since the environment they support is essentially the same as that provided by Vservers i.e. a thin layer on top of a UNIX-like kernel. Since each addresses a slightly different problem from PlanetLab it should be illustrative to consider what facilities a service like Proper should provide in those systems.

Another area of related work is in the security community, where researchers have investigated schemes for *system call interposition* e.g., *Ostia [2], Systrace [5]* in order to allow administrators to restrict the ability of unmodified applications to execute certain system calls. These systems adopt many of the same implementation solutions as our PlanetLab implementation of Proper and could thus be leveraged to provide an framework for integrating Proper with unmodified clients.

6 Conclusions

As part of the PlanetLab project we found that it was necessary to give unprivileged clients running inside a virtual machine access to certain privileged operations. This was accomplished with Proper, a user-level service running in the privileged root VM that performs operations on behalf of those unprivileged clients.

The key insight from this work is that supporting communication between VMs requires a degree of support from the underlying virtual machine monitor: virtualisation at the system call level readily supports certain forms of inter-VM communication, whereas more thoroughly virtualised systems are likely to require some modification to support the required forms of sharing and communication.

References

[1] 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. In *Proc. 19th SOSP* (Lake George, NY, Oct 2003).

[2] GARFINKEL, T., PFAFF, B., AND ROSENBLUM, M. Ostia: A Delegating Architecture for Secure System Call Interposition. In *Proc. 2004 Symposium on Network and Distributed System Security* (2004).

[3] LINUX VSERVERS PROJECT. `http://linux-vserver.org/`.

[4] OSMAN, S., SUBHRAVETI, D., SU, G., AND NIEH, J. The Design and Implementation of Zap: A System for Migrating Computing Environments. In *Proc. 5th OSDI* (Boston, MA, Dec 2002), pp. 361–376.

[5] PROVOS, N. Improving Host Security with System Call Policies. In *Proc. 12th USENIX Security Symposium* (Washington, DC, Aug 2003), pp. 257–272.

[6] TUCKER, A., AND COMAY, D. Solaris Zones: Operating System Support for Server Consolidation. In *3rd Virtual Machine Research and Technology Symposium Works-in-Progress* (San Jose, CA, May 2004).

[7] VMWare. `http://www.vmware.com/`.

[8] WHITAKER, A., SHAW, M., AND GRIBBLE, S. D. Scale and Performance in the Denali Isolation Kernel. In *Proc. 5th OSDI* (Boston, MA, December 2002), pp. 195–209.