Poster: Rethinking Operating System Interfaces to Support Robust Applications

W. Michael Petullo (Student) and Jon A. Solworth (Faculty)
Department of Computer Science
University of Illinois at Chicago
Chicago, Illinois
mike@flyn.org, solworth@rites.uic.edu

***Abstract*—In current systems, application developers must provide substantial security-critical code—including code to handle authentication—in their applications. The result is that application flaws often undermine system security. We are building Ethos, an Operating System (OS) that leverages the kernel's complete mediation property to guarantee more security protections—including network encryption and authentication—across all applications. Here we provide an overview of Ethos and a subset of its system call interface.**

I. Introduction

Computer systems are organized into a strata—a layering that begins at hardware and progresses to an application. Each layer exports a set of interfaces to the layer above it. These layers include the aforementioned hardware, an optional Virtual Machine Monitor (VMM), an OS kernel, a programming language, libraries, and an application.

The OS kernel is the first software layer that completely mediates all program access to external resources [6]. Thus the OS kernel is in a unique position with respect to security. Properly designed and implemented, its controls are absolute; they cannot be bypassed by applications. For this reason, we focus on the OS kernel as the decisive component of the strata.

Many design decisions must be balanced to produce a software strata. For example, an OS's system calls embody the interface exported to the stratum above it; an architect may design these system calls to export a very low level of abstraction so that application programmers can achieve high levels of performance for their particular application.

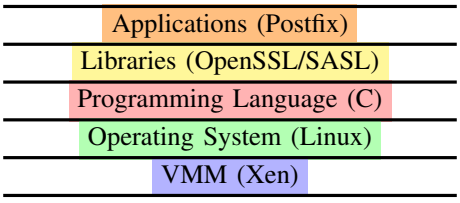

Figure 1. A software strata

When an OS provides a low level of abstraction, it increases the burden on application developers. For example, POSIX networking does not provide for strong authentication. Such security functionality is left to applications or the libraries they build upon. This leads to duplicating critical code; even when authentication is consolidated in a library, the amount and complexity of code required to properly invoke the library is often non-trivial. Furthermore, libraries do not provide complete mediation; an application could be written to avoid authentication entirely.

We believe POSIX provides too many opportunities to make security errors. Consider Postfix, which was written by a security expert. Postfix can be considered a rough limit on our ability to develop a robust application on the POSIX strata. Yet, even Postfix contained a flaw in its use of OpenSSL that resulted in the possibility of a plaintext injection attack [2]. In this case, fixing Postfix does not preclude other applications from having the same flaw. Postfix contains around 2,000 lines of code to support robust networking. The Dovecot IMAP server, Apache, and mod_auth_kerb contain 15,000, 1,800, and 1,500 authentication-related lines of code, respectively. (These counts do not include libraries; OpenSSL is around 250,000 lines.)

II. Ethos: A strata designed for security

Ethos is a clean-slate OS with a primary goal of security [7]. We are developing Ethos because the current software strata does not sufficiently aid writing robust applications.

A. Overview of Ethos system calls

Ethos has relatively few system calls and they provide higher levels of abstraction than POSIX, simplifying application development and authorization policy specification. This differentiates Ethos authorization from SELinux—the latter is complex [3], partly because Linux's system call interface is complex. Ethos' authorization system also has more information available to it. In particular, Ethos can make authorization decisions based on a remote user because network authentication is performed by the OS.

B. A clean slate enables different design choices

In the last decade, we have witnessed the rise of commodity system virtualization platforms. Ethos is built on top of the Xen VMM [1]. A principal advantage of this approach is that Ethos need only support a few virtualized

devices yet can run on any hardware supported by Xen. This dramatically reduces Ethos' code base and decreases the vulnerabilities associated with device drivers.

A Virtual Machine (VM)-based approach also means that Ethos can coexist with other OSs, avoiding the *application trap* [5]. The use of Ethos is justified even if it has only a single desirable application, since Ethos does not preclude using legacy OSs and their applications.

Another opportunity arises because network latency, not processing encryption, is now often the critical performance consideration [4]. Thus Ethos encrypts all network traffic and authenticates using cryptography all network data provided to applications. (There may still be applications that require unencrypted network connections, perhaps for very sensitive performance reasons. Such applications can be run on a legacy OS in a separate VM.)

C. The role of programming languages

Programming languages also have a large impact on system security. For example, language design can remove buffer overflows. In this way, language and system design are complimentary. Over time, languages have become more abstract, allowing programmers to reason about increasingly complicated software. Ethos intends to do the same for system design. Ethos' design allows for the use of any programming language and does not require the use of an application virtual machine (e.g., JVM or CLR).

III. A Case Study: Network Authentication with Zero Lines of Application Code

Figure 2 presents an Ethos network application. A distributor process accepts connections from a remote client. Ethos does not authorize the distributor to read or write the network file descriptor—it must be passed to a server process. The server process must run with the remote user's credentials in order to read or write the descriptor.

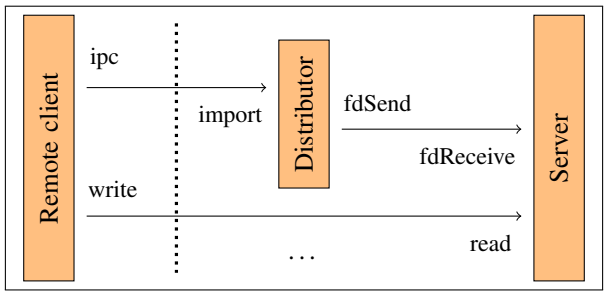

Figure 2. An Ethos network application

Here we provide a more detailed look at the system calls referenced in Figure 2. The client uses the ipc system call to connect to a remote host. Ethos restricts ipc based on a user, program, service, and remote host tuple.

```
ipc (service, remoteHost)
```

The distributor calls import to accept a connection. As the connection is coming from a remote host, Ethos ensures that the traffic is encrypted. Ethos restricts import based on a user, program, and service tuple. Furthermore, the import system call will not return unless Ethos identifies the remote end as an authorized client host and user using cryptographic authentication.

```
netFd, user ←import (serviceFd)
```

Instead of setuid, Ethos provides fdSend/fdReceive, a pair of system calls that pass a file descriptor from one process to another. The distributor sends the descriptor to the server using this mechanism.

```
fdSend (fd, user, program)
```

Finally, the server receives and uses the descriptor.

```
fd ←fdReceive ()
```

Thus Ethos relieves the burden of application developers to provide authentication through system-wide guarantees. In fact, a user that Ethos does not authenticate and authorize never even interacts with user space code.

IV. Status and Future Work

It is already possible to write applications for Ethos in C or Go. Ethos currently provides processes, encrypted networking, and a filesystem. We have completed 39 system calls and related Go packages. We have implemented a shell, basic command-line utilities, and a networked messaging system. The latter provides a case study of our strata.

We are beginning to shift our focus from kernel development to creating a robust user space. Projects underway include writing Go packages to aid application development, designing a graphics subsystem, and developing substantial applications. This, in turn, will provide continued evaluation of the design decisions embodied in our strata.

References

[1] Barham, P. et al. Xen and the art of virtualization. In *SOSP* (2003).

[2] CVE-2011-0411. National Vulnerability Database, 2011.

[3] Hicks, B. et al. A logical specification and analysis for SELinux MLS policy. In *Proc. of the 12th ACM symposium on Access control models and technologies* (2007).

[4] Langley, A. et al. Overclocking SSL. In *Velocity: Web Performance and Operations Conference* (2010). http://www.imperialviolet.org/2010/06/25/overclocking-ssl.html.

[5] Pike, R. System software research is irrelevant, 2000.

[6] Saltzer, J. H. and Schroeder, M. D. The protection of information in computer systems. *Proceedings of the IEEE* (1975).

[7] Solworth, J. A. Robustly secure computer systems: A new security paradigm of system discontinuity. In *NSPW* (2007).