Disaggregated System Services Through Lightweight Capability Domains

John Regehr, Associate Professor
School of Computing, University of Utah
50 S. Central Campus Dr., Room 3190
Salt Lake City, UT 84112
regehr@cs.utah.edu
+1 801 581 4280

Contacts: Ben Laurie, Mark Miller

Abstract

For several decades the operating system community has accepted that the only way of minimizing the trusted computing base, and constructing more secure, least authority systems, is to reimplement monolithic kernel functionality as a set of isolated microkernel servers. Our work aims to explore a less invasive means of constructing fine-grained, least authority environments. Instead of splitting the core functionality of a traditional operating system into multiple servers, we will develop a mechanism that can securely isolate individual kernel components right inside the address space of the monolithic operating system kernel. The traditional OS boots normally, supporting discovery, configuration, and initialization. However, after the system is initialized, the kernel will transfer execution of some subsystems into isolated protection domains. To implement protection domains, we propose to extend a hypervisor with support for lightweight protection domains and a capability access control model. Although these protection domains will run inside the guest OS's kernel address space, they will be fully protected by the hypervisor, which also provides a mediated capability interface. Effectively, we will execute a traditional operating system on a capability machine, which we implement in the hypervisor rather than on specialized hardware.

In contrast to the traditional microkenelization, our isolated service implementations will not require major modifications to their code. Our goal is to develop a small set of interfaces, communication, and synchronization mechanisms that can run the common kernel subsystems in both monolithic and disaggregated configurations. The main insight which makes our work feasible is that although existing operating system components depend heavily on the rest of the kernel during initialization, during regular operation the code of many subsystems largely independent of the rest of the kernel. This level of independence is due to several decades of engineering effort aimed at modularization of kernel components.

A Motivating Example

Hypervisors have become a de facto solution to the problem of secure isolation. An untrusted application such as a web browser or PDF reader is sealed in its virtual container. Without the shared operating system kernel, and with the minimal TCB, virtual machine monitors significantly reduce the attack surface. Unfortunately, seemingly isolated virtual machines still require support from a number of privileged services, e.g., to access the user's home file system, route network packets through a shared network stack, and reduce the per-VM storage overheads with the copy-on-write block storage stack, etc. An attacker might first exploit a vulnerability in the PDF reader's JavaScript engine, second compromise the guest OS kernel, and finally propagate itself into a privileged virtual machine through an attack on one of the shared services, e.g., file system, network stack, etc. Despite a number of advances in program analysis, software testing, and verification, it is unlikely that the large, semantically rich functionality of the traditional operating system services can be effectively secured.

Research Goals

The main challenge of our work is:

Design a novel mechanism for constructing least authority execution environments that can run nearly unmodified OS components in lightweight protection domains.

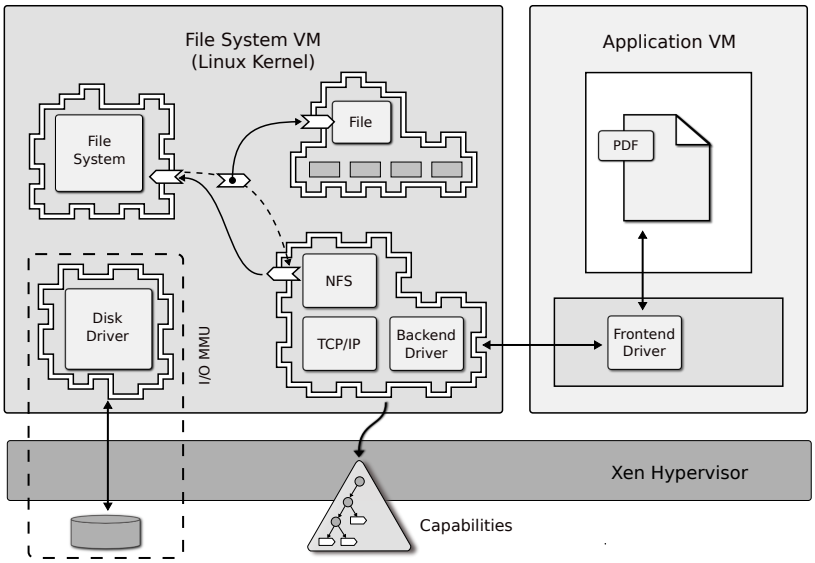

Figure 1. Capability protection domains. The File System VM is a privileged VM which provides a shared file system for untrusted Application VMs. An unmodified Application VM which hosts a PDF reader, uses the traditional backend-frontend split device drivers, TCP/IP stack, and NFS to access the shared file system. However, individual system services are strongly isolated inside capability protection domains that are accessible via capability invocations.

Instead of promoting clean-slate design, our mechanisms—capabilities, lightweight protection domains, and an execution model—are geared towards providing an isolation environment capable of leveraging millions of lines of existing operating system code, which has been in use for decades.

We propose building upon two full-featured, industry standard, open source platforms: Xen and Linux. Over the course of our work we plan to develop the following components of the system. One student (funded by this award) and one staff member, Anton Burtsev, (funded from another source) will work on this project:

1. Lightweight protection domains. A lightweight protection domain isolates a part of the guest's kernel address space. Unlike previous in-kernel protection domains such as Nooks [5], our notion of protection is symmetrical: not only is the kernel isolated from the protection domain, but the reverse is true as well. By avoiding the creation of new address spaces, naming and sharing of resources across protection domains are simplified. We plan to utilize hardware support for nested page tables and tagged TLBs to make protection domains efficient.

2. Capability model. To control information flow across protection domains, we rely on capabilities, borrowing ideas from the seL4 microkernel [3]. Similar to seL4, in our system a capability is an entry in a hypervisor-protected data structure, which can be referenced from guest code via a local name. Inside the hypervisor, each capability describes one of the objects implemented by the hypervisor—a memory region, a capability invocation entry point, or a thread of execution. Capability names are the only means to reference and exchange resources across protection domains, e.g., a memory region named by a capability can be remapped across protection domains. Capability invocation is the only way to transfer control between protection domains. Similar to a function call, the capability invocation provides a way to transfer from the caller to the callee protection domain via a specified entry point. A capability invocation can pass "in" and "out" arguments as simple values and capabilities.

3. Execution environment. To simplify the reuse of existing kernel code, our isolated execution environment will look very much like the standard kernel environment. A typical operating system kernel allows reentrant, concurrent, and parallel execution of the kernel code; we will provide a similar execution model.

Other challenges. A number of design challenges related to the execution model need to be addressed over the course of our work. Can multiple threads enter the same protection domain via capability invocations? Which part of the system provides a stack page when the flow of control enters a new protection domain? How are the state and capabilities of individual threads of execution shared inside the same protection domain? Another set of challenges is relevant to splitting functionality across protection domains, e.g., memory allocation functions, common POSIX functions like memcpy, shared state like asynchronous queues, etc. To address some of the issues we plan to use design experience from the OSKit project [2].

Outcome

The main outcome of our work will be a realization of a least authority, capability-based execution environment in the Linux operating system and the Xen hypervisor. We plan to develop a prototype of a disaggregated least privilege file system. We will implement disaggregation of individual files, deprivilege the file system to run only as a naming service which has no direct access to the block devices, and use isolated NFS stacks to share the file system across multiple untrusted virtual machines. While working on the file system example, we plan to make the following additional contributions. First, our work will produce a practical evaluation of the hardware-supported nested page tables, and tagged TLBs as the mechanisms for creating lightweight protection domains. Second, our work will explore the challenges and opportunities of transferring the code of existing operating system in a set of isolated protection domains. Our experiences in separating the kernel subsystems, designing transparent capability communication mechanisms, and developing use-case scenarios for accessing the traditionally global kernel state via explicit capability mechanisms can be used to influence the design of the future hardware-supported capability architectures like CHERI [6].

Related work

Several open source and commercial projects attempt to secure commodity applications using full-system virtualization (Qubes OS [4], Bromium [1]). In contrast to our work the system services remain largely monolithic and vulnerable to multiple attacks. Microkernels, and most notably seL4, the first formally verified microkernel [3], suggest to isolate functionality of traditional system components in multiple services. At the moment, none of the microkernel projects offers a set of mature disaggregated system services, and "best practice" recipes for constructing least authority environments. Object capability languages are the right abstraction to construct least privilege environments. Similar to microkernels they will require reimplementation of the entire operating system stack. Capability-enabled hardware like CHERI [6] is close to our work. CHERI also builds a capability machine but in hardware. Operating system frameworks like OSKit [2] can potentially achieve similar to architecture. The main problem of such frameworks is that being outside of the mainline kernel trees, the code of such frameworks quickly becomes obsolete. Our goal is to develop a minimal set of mechanisms which can potentially be integrated in the mainline operating system.

References

[1] Bromium, Inc. Bromium Micro-virtualization. Whitepaper, 2010.

[2] Bryan Ford, Godmar Back, et al. The Flux OSKit: a substrate for kernel and language research. In *SOSP*, 1997.

[3] Klein, G. at al. seL4: formal verification of an OS kernel. In *SOSP*, 2009.

[4] Rutkowska, J. and Wojtczuk, R. Qubes OS architecture. Technical report, 2010.

[5] Michael Swift, Brian Bershad, and Henry Levy. Improving the reliability of commodity operating systems. In *SOSP*, 2003.

[6] Watson, R. at al. CHERI: a research platform deconflating hardware virtualization and protection. In *RESoLVE*, 2012.

Our qualifications

The PI (whose recent research has been in other areas) is an experienced embedded system and OS hacker. Moreover, the PI and the PhD student will receive technical assistance from Anton Burtsev, a research staff member with the Flux group who is an extremely seasoned Xen hacker.

Data policy

All software artifacts developed over the course of this work will be released as open source under the GPLv2 license. We plan to provide configuration and setup support for other academic teams interested in using results of our work—as we have already done for XenTT, our deterministic replay engine for Xen.