Towards Verified Security for a Functional Microkernel

Rebekah Leslie[1] and Mark P. Jones[2]

[1] Portland State University
[2] OGI School of Science & Engineering at OHSU

Functional languages are prevalent in academia and are maturing into powerful tools suitable for industrial use. The characteristics of functional languages provide great advantages for developing large systems, particularly when security is a priority. To demonstrate this, we are implementing a microkernel in Haskell with verified separation between domains.

Haskell is a functional language with higher-order functions, polymorphism, and abstract data types. These abstraction mechanisms facilitate the development of modular programs with a high degree of code reuse. Abstraction allows us to apply modular reasoning techniques and reduces the size of the verification effort.

The mathematical semantics of Haskell is another advantage for security. Haskell is a pure language, which means that the result of a function depends only on its arguments. This property leads to programs that we can reason about locally. To maintain purity, the type system tracks the use of side effects. It is generally difficult to reason about side-effecting programs, but tracking the use of effects simplifies this task.

There are some challenges in using Haskell to write an operating system. As a high-level language, Haskell abstracts from details that systems programs must address, such as low-level data structure manipulation. We are working on a language extension to ameliorate this problem. Haskell also depends on a complicated run-time system. Such run-time systems are often unverified. Implementing a minimal, high-assurance run-time system is a topic for future work.

The L4 architecture is the basis of our microkernel implementation. L4 has a minimal design; it only includes features that cannot reside outside of the kernel. As a result, user-space threads implement many standard operating system components, such as device drivers and page-fault handling. This approach leads to a small kernel with complex domain interactions, so proving separation is a tractable, yet interesting challenge. A further benefit of L4 is the number of existing applications we can use to test our implementation, including multiple ports of Linux.

Our recent work focuses on inter-process communication (IPC) in L4. IPC is interesting from a security perspective because it provides rich possibilities for interference. The IPC mechanism in L4 contains several nonstandard features, so reasoning about it is nontrivial. We have an implementation in Haskell and are working on a formalization of separation that accounts for the complexities of IPC.

To formalize separation, the kernel must incorporate a notion of security policy. To achieve this, we supplement the L4 architecture with a dynamically configured data structure for security policies. A trusted domain writes to this *policy structure* to specify permissible domain interactions, and the kernel restricts domain behavior according to this information. We intend this policy-management mechanism to allow grades of interference, as opposed to the absolute notion of interference described by traditional policies. The dynamic and fine-grained nature of policy in our system requires a new strategy for reasoning about separation, the development of which is underway.