My research develops language-based abstractions for secure distributed computing. These abstractions support reasoning about security and performance, simplify application design and implementation, and admit efficient, scalable implementations in real systems. Poor abstractions are often disastrous for security—something I learned first-hand as a vulnerability analyst at the National Security Agency. Burdening programmers with low-level decisions about remote communication, data storage, and security can make even simple high-level designs hard to implement correctly. Unfortunately, today’s programmers must largely rely on their own vigilance to secure their programs and make them scale.

Current programming models are particularly inadequate for federated systems, where systems span multiple independent administrative domains. Securing such systems has always been challenging. In fact, a classic approach to securing corporate networks was to avoid the complexity of federation using a closed system model, where the devices and software permitted on the network are strictly controlled. However, our increasing reliance on personal mobile devices has forced many corporate networks to federate, permitting limited access to devices not within their control.

The situation will soon become even more acute. Far more devices will be connected to existing networks in the near future, a phenomenon some call the Internet of Things. These devices present real security concerns. Not only will sensors like power meters, microphones, and cameras collect sensitive information, but many devices will also have significant impact on the physical world; consider, for example, self-driving cars and remotely-operated surgical robots. Centralized, closed-system approaches such as “app stores,” in which all software is manually audited to some extent, are too expensive and will never be able to cope with the full range of applications, devices, or desired security guarantees. In fact, app stores are already overwhelmed by malicious and vulnerable applications: the security firm FireEye recently reported [1] that over 2,800 iOS applications in the Apple app store could be used to capture audio and steal data.

My research addresses these challenges with automated, decentralized techniques for building distributed applications that are secure by construction. I focus on two fundamental perspectives on security: authorization, which concerns who trusts whom, and information flow control, which concerns who may learn a secret or influence a value. In particular, my research explores how these two perspectives interact. Real systems use secret and partially-trusted data for authorization all the time—for instance, Facebook uses a user’s secret list of friends to perform many authorization checks. In my research, I have shown that language-based techniques are useful for reasoning about authorization and information flow. For instance, Mobile Fabric [2] enforces information security in mobile code and Flow-Limited Authorization [3] provides secure authorization and information flow without side channels.

Developing strong security techniques does not guarantee that programmers will adopt them. Systems programmers tend to favor low-level abstractions, and with good cause: many existing abstractions restrict functionality and reduce performance. However, low-level abstractions provide programmers with far fewer security guarantees. For instance, almost all of the vulnerabilities I discovered while at the NSA were in low-level systems code that provided no memory safety.

Because of this experience, I believe making a significant impact on system security requires not only expressive high-level abstractions, but also high-performance implementations. My research has also shown that static and dynamic program analysis can significantly
improve the performance of high-level abstractions while preserving their semantics. For example, Pyxis \[4\] automatically improves throughput and latency in database applications, and Warranties \[5\] enable strong, scalable consistency with less overhead.

**Secure mobile code (S&P’12 \[2\])** Mobile code is pervasive in modern applications, in both client-side Javascript or “apps” from an app-store. However, current security abstractions for mobile code are unsatisfying on two counts: they significantly restrict functionality but provide limited security guarantees. Consider the *same-origin policy* for Javascript, which restricts communication to the website providing the code. This policy not only prohibits useful cross-site applications, it fails to provide any real security guarantees. For instance, advertisers routinely track users across websites without violating the same-origin policy.

I developed Mobile Fabric with my colleagues to enforce the confidentiality and integrity of mobile code in distributed applications using information flow control. The architecture, based on Fabric \[6, 7\], gives mobile code expressive power lacking in other platforms: it can securely access distributed, persistent, and shared information from multiple trust domains, unlike web applications bound by the same-origin policy. The key innovation of Mobile Fabric is labeling the code itself with information flow policies just like other data in the system. These policies form the basis of *provider-bounded label checking*, a mechanism that protects against malicious code providers. Provider-bounded label checking improves security, but also increases functionality. For instance, even untrusted code can safely process confidential data—provider-bounded label checking ensures the code cannot leak information.

**Flow-Limited Authorization (CSF’15 \[3\])** Interactions between information flow and authorization create security vulnerabilities not fully identified or addressed in prior work, especially in distributed settings. I introduced Flow-Limited Authorization, an integrated approach to distributed authorization and information flow control. The Flow-Limited Authorization Model (FLAM) captures the information security of three aspects of authorization mechanisms: first, delegation of authority between principals; second, revocation of previously delegated authority; third, information flows created by the mechanisms themselves.

All authorization and information flow decisions derivable in FLAM are secure: attackers cannot influence authorization decisions or learn confidential trust relationships by processing or issuing FLAM queries. The security guarantees provided by FLAM are particularly attractive since they require very few assumptions about the system: malicious nodes may issue arbitrary queries, forge delegations between principals, and participate in distributed query processing.

Prior to FLAM, characterizing the information security of such mechanisms was difficult. Likewise, several vulnerabilities in existing mechanisms stem from imperfect reasoning about the consequences of dynamic trust relationships. FLAM provides a new foundation for both authorization and information flow that eliminates these vulnerabilities.

**Making high-level abstractions perform** The above work develops high-level abstractions, but a common concern is that abstraction kills performance. Two of my projects demonstrate that abstractions can perform well by automatically and dynamically specializing their implementations to a target workload. These techniques improve performance without forcing the programmer to reason directly about network communication, the placement of code and data, or data consistency.

**Pyxis (VLDB’12 \[4\], CIDR’13 \[8\])** Novices write database applications that communicate frequently with the database, resulting in high-latency and wasted server resources.
Expert database programmers thus factor out communication-heavy procedures into stored procedures executed on the server. Unfortunately, this process is difficult. Worse, executing too much code on the database server can overload it.

Pyxis automatically partitions standard database applications between an application server and a database server. I designed Pyxis’s partition graph, a key contribution that represents the dependencies, communication costs, and computational load of a distributed application. In addition, I designed and developed the program analyses to automatically construct partition graphs and partition assignments for Java database applications. Pyxis-generated partitions are much faster than naive implementations and have similar performance to that of custom stored-procedure implementations. Pyxis demonstrates that high-level programming abstractions can be executed with high performance.

Warranties (NSDI’13 [5]) Strong consistency offers abstractions that are easy to reason about since the distributed reads and writes of one program execution are logically isolated from those of other executions. Unfortunately, the overhead of consistency protocols often hurts performance. Rather than adopt a weaker consistency model, warranties generalize optimistic concurrency control, which provides strict serializability. A warranty is a time-limited assertion about one or more distributed objects. These assertions improve throughput since clients holding warranties need not communicate to verify the warranty’s assertion. Warranties can be expressed using language-level computations, and can be integrated into a programming model as a form of memoization. Like Pyxis, warranties allow programmers to use abstraction without giving up scalability or performance, but are applicable to an even wider class of distributed computation.

Future goals: Securing blockchain applications and the Internet of Things The abstractions programmers use today are not only insufficient, they are the root of the problem. The high-level abstractions of Mobile Fabric and Flow-Limited Authorization provide an excellent foundation for securing distributed applications as systems become increasingly federated; Pyxis and warranties demonstrate that the right abstractions yield adaptive, high-performance implementations. My future research goals seek to move beyond enforcing only confidentiality and integrity to ensure applications exhibit arbitrary end-to-end properties critical to security, reliability, and scalability, even in the presence of malicious nodes.

• Secure invariants in open federated systems. Flow-Limited Authorization provides a foundation for end-to-end enforcement of confidentiality and integrity despite changes in trust relationships. Other properties may be equally important: for instance, performance invariants that ensure workloads don’t overload devices with limited capabilities. Enforcing invariants across trust boundaries is problematic since information necessary for enforcement may sensitive or untrustworthy. I plan to use Flow-Limited Authorization to build new languages and systems that enforce end-to-end invariants in federated systems securely and automatically.

• Securing blockchain applications. There will soon be an explosion of applications that use blockchain technology (e.g., BitCoin [9] or Ethereum [10]) as a trusted third-party for a huge range of transactions from simple purchases to complex smart contracts. These applications are difficult to get right, and bugs can be costly. Not only are confidentiality and integrity important, but also other properties like incentive compatibility. I plan to extend my research in Flow-Limited Authorization and secure mobile code to build blockchain applications, and have joined the Initiative for CryptoCurrencies & Contracts [11] to collaborate with other researchers.
References


