Using Security Policies to Write Secure Software

Andrew Myers
Cornell University, Ithaca, USA

We explore the idea of integrating security policies into the programming environment and the run-time system. A programming language that incorporates security policies can help programmers reason about the security of the software they are building. We examine how this approach works in the context of the security- typed programming language Jif, which allows programmers to express both information flow policies for confidentiality and integrity, and access control policies. The compiler and run-time system can then use this added information to enforce strong security properties such as noninterference and robust information release.

A variety of language mechanisms make this approach expressive enough to build real software: support for rich language features such as objects and exceptions, automatic inference of security policies, constrained parameterization over security policies, and dependent types that capture security policies determined at run time. We see why these mechanisms help with building real software, how they can be formalized and proved sound in terms of the noninterference security property, and how they are implemented.

Jif supports mechanisms for relaxing confidentiality and integrity policies, called declassification and endorsement respectively. Noninterference no longer holds once these mechanisms are used. What can we say about security? We show that a connection between confidentiality and integrity can be made as robust information release: informally, the adversary cannot cause information to become visible to itself.

Part of the benefit of integrating security policies into the program lies in the opportunity it offers to raise the level of abstraction at which we build software. Modern applications are difficult to build securely. They are distributed applications, so applications need to be secure even when the adversary partially controls the network and perhaps also some of the host machines running the application. Yet programming these applications also requires attention to a host of low-level details that obscure program logic.

By exposing security policies to the compiler and run-time system, it becomes possible for these layers to automatically synthesize security enforcement mechanisms such as partitioning and replication of code and data, digital signatures and encryption, and network protocols. The result is not only a higher level of abstraction for building secure software, but also stronger assurance that the security mechanisms being used do enforce the security requirements of the application.

We look at how different kinds of policies – for confidentiality, integrity, and even availability, can be used automatically to drive the compilation of high-level annotated code into target code that employs a variety of security mechanisms. We also see how this approach can be used to make web applications easier to build securely and efficiently.

We conclude with some thoughts about future directions for research on secure information flow and integrating security policies into programming environments.

References

1. S. Chong, A. C. Myers. *End-to-End Enforcement of Erasure and Declassification.* Proc. "IEEE Computer Security Foundations Symposium", pp. 98–111, 2008.
2. C. Fournet, T. Rezk. *Cryptographically Sound Implementations for Typed Information-Flow Security.* Proc. "ACM POPL 2008", pp. 323–335, 2008.
3. S. Chong, J. Liu, A. C. Myers, X. Qi, K. Vikram, L. Zheng, X. Zheng. *Secure Web Applications via Automatic Partitioning.* Proc. "21st ACM Symposium on Operating Systems Principles" (SOSP'07), pp. 31–44, 2007.
4. S. Chong, K. Vikram, A. C. Myers. *SIF: Enforcing Confidentiality and Integrity in Web Applications.* Proc. "USENIX Security Symposium", pp. 1–16, 2007.
5. L. Zheng, A. C. Myers. *Dynamic Security Labels and Static Information Flow Control.* International Journal of Information Security, 6(2:3), Springer, 2007.
6. A. C. Myers, L. Zheng, S. Zdancewic, S. Chong, N. Nystrom. *Jif 3.0: Java Information Flow. Software release.* 2006.
   Available at `http://www.cs.cornell.edu/jif`
7. S. Chong, A. C. Myers. *Decentralized Robustness.* Proc. "19th IEEE Computer Security Foundations Workshop" (CSFW'06), pp. 242–253, 2006.
8. A. C. Myers, A. Sabelfeld, S. Zdancewic. *Enforcing Robust Declassification and Qualified Robustness.* Journal of Computer Security, 14(2), pp.157–196, 2006.
9. F. Pottier, V. Simonet. *Information Flow Inference for ML.* Proc. "29th ACM Symposium on Principles of Programming Languages" (POPL'02), pp. 319–330. 2002.
10. S. Tse, S. Zdancewic. *Run-Time Principals in Information-Flow Type Systems.* IEEE Symposium on Security and Privacy, 2004.
11. S. Zdancewic, A. C. Myers. *Observational Determinism for Concurrent Program Security.* Proc. "16th IEEE Computer Security Foundations Workshop" (CSFW), pp. 29–43, 2003.
12. L. Zheng, S. Chong, A. C. Myers, S. Zdancewic. *Using Replication and Partitioning to Build Secure Distributed Systems.* Proc. "2003 IEEE Symposium on Security and Privacy", pp. 236–250, 2003.
13. A. Sabelfeld, A. C. Myers. *Language-Based Information-Flow Security.* IEEE Journal on Selected Areas in Communications, 21(1), pp.5–19, 2003.
14. A. C. Myers, B. Liskov. *Protecting Privacy using the Decentralized Label Model.* ACM Transactions on Software Engineering and Methodology, 9(4), pp. 410–442, 2000.
15. A. C. Myers. *JFlow: Practical Mostly-Static Information Flow Control.* Proc. "26th ACM Symposium on Principles of Programming Languages" (POPL'99), pp. 228–241, 1999.
16. J. A. Goguen, J. Meseguer. *Security Policies and Security Models.* Proc. "IEEE Symposium on Security and Privacy", pp. 11-20, 1982.
17. D. E. Denning, P. J. Denning. *Certification of Programs for Secure Information Flow.* Communications of the ACM, 20(7), pp. 504–513, 1977.
18. D. E. Denning. *A Lattice Model of Secure Information Flow.* Communications of the ACM 19(5), pp.236–243, 1976.