The Engineering Challenges of Trustworthy Computing

Greg Morrisett
Harvard University, USA

Most of the software that we depend upon, including operating systems, communication stacks, file systems, databases, and embedded control software, is coded in low-level, error prone languages such as C and C++. These languages are reasonably portable and provide direct access to hardware and software resources with minimal overhead. But because they fail to enforce even the simplest of abstractions, we continue to be subjected to well-known attacks such as buffer over-flows.

In principle, higher-level languages, such as Java and C#, can help to stop these attacks through a combination of static and dynamic type-checking. In practice, Java-like languages are ill-suited for many domains because of the high space and time overheads. Even where performance isn't a problem, re-coding existing C software in a higher-level language can cost too much—who wants to pay to re-code the 50 million lines of code in Windows? Furthermore, a modern Java run-time system includes well over a half million lines of (error-prone) C code. Finally, a type system such as Java's only offers a relatively weak safety guarantee. So what assurance do we really gain by (re)writing software in Java?

We will talk about a range of language, compiler, and verification techniques that can be used to address safety and security issues in systems software today. Some of the techniques, such as software fault isolation, are aimed at legacy software and provide relatively weak but important guarantees, and come with significant overhead. Other techniques, such as proof-carrying code, offer the potential of fine-grained protection with low overhead, but introduce significant verification challenges. None of these approaches is a panacea, but each offers some insight into practical approaches to software security.

References for the topics that we will address include the following:

References

1. Martín Abadi, Mihai Budiu, Úlfar Erlingsson, and Jay Ligatti. *Control-Flow Integrity: Principles, Implementations, and Applications*. Proceedings of the 12th ACM Conference on Computer and Communications Security (CCS'05), Alexandria, 2005.
2. Úlfar Erlingsson, Martín Abadi, Michael Vrable, Mihai Budiu, and George C. Necula. *XFI: Software Guards for System Address Spaces*. Proceedings of the 7th Usenix Symposium on Operating Systems Design and Implementation (OSDI'06), Seattle, 2006.
3. Trevor Jim, Greg Morrisett, Dan Grossman, Michael Hicks, James Cheney, and Yanling Wang. *Cyclone: A Safe Dialect of C*. USENIX Annual Technical Conference, Monterey, 2002.
4. Stephen McCamant and Greg Morrisett. *Evaluating SFI for a CISC Architecture*. 15th USENIX Security Symposium, Vancouver, 2006.
5. G. Morrisett, D. Walker, K. Crary, and N. Glew. *From System F to Typed Assembly Language*. Conference Record of POPL '98: The 25th ACM SIGPLAN-SIGACT Symposium on Principles of Programming Languages. San Diego, pp. 85–97, 1998.
6. G. Necula and P. Lee. *Safe Untrusted Agents Using Proof-Carrying Code*. LNCS 1419, p. 61, 1988
7. George C. Necula, Scott McPeak, and Westley Weimer. *CCured: Type-safe Retrotting of Legacy Code*. Twenty-Ninth ACM Symposium on Principles of Programming Languages, 2002.
8. R. Wahbe, S. Lucco, T. E. Anderson, and S. L. Graham. *Efficient Software-based Fault Isolation* . Proceedings of the 14th ACM Symposium on Operating Systems Principles, pp. 203–216, 1993.