Editors: Matt Bishop, bishop@cs.ucdavis.edu
Debra Frincke, frincke@cs.uidaho.edu

Teaching Constructive Security

Cynthia E. Irvine
Department of Computer Science, Naval Postgraduate School

In an earlier installment of this department (Sept./Oct. 2003, pp. 64–67), the author recommended attack understanding as an approach to teaching information assurance at the university level. Attack understanding plays a key role in a cybersecurity education program, but students also must learn cybersecurity's core principles and how to construct secure systems. Students taught to understand the attacks on and vulnerabilities of current systems—and, presumably, how to create and install repairs to mitigate these attacks—only address the systems' superficial weaknesses. They will be doomed to the Sisyphean purgatory of penetrate and patch. This approach serves neither our students nor the cyber infrastructure's stability. We must prepare our students to build a better future, and we can achieve this only by teaching them constructive security principles.

Security philosophy

The constructive security philosophy is based on the assumption that that for certain critical operations, a system always must do the "right thing." What the "right thing" is depends on the intended security policy, but we need assurance that the system won't do something else. Thus, we must demonstrate the absence of unspecified functionality—a manifestation of security's negative requirement. Because we must demonstrate the absence of something in a way that will promote user confidence, it is necessary to build systems to demonstrably meet the negative requirement.

Attack Understanding

Clearly, students must have attack understanding. They should understand that for poorly built systems, an attacker always has the advantage and that they cannot address the asymmetric threat posed by the attacker through vigilance and repairs to inherently flawed systems. Students learn that inspecting an already-built complex system—attempting to demonstrate that it contains no security flaws—is tantamount to asking for magic. How can they inspect or test such a system? Not only are there innumerable parameters and system states, there is no way to know if all of the parameters have been articulated. The system might contain security flaws as artifacts of sloppy development practices, or someone might have implanted something in the system to bypass its security mechanisms.

Attack understanding at the university level should make students cognizant of how attackers could exploit system weaknesses. More importantly, students should be able to characterize a general taxonomy of flaws that might render a system vulnerable to attack and understand that many flaws result from discipline failure in software design and programming practices. In addition, through attack understanding, students gain an appreciation of the threat posed by a malicious element intent on subverting the system. A number of classic and recent papers and reports address accidental and intentional security flaws. Many of these should be part of a classroom unit to motivate the study of building systems securely.[1] An appreciation of the ways that system flaws can be exploited by an adversary permits system designers to describe the threats that must be addressed when specifying system requirements.

Penetrate and patch is an endless cycle in which the defender never knows whether a devastating flaw still lurks within the system. It is not a systematic way to achieve secure systems. At this point you might think that I have presented students with an impossible task, that they must build secure systems without relying on post-development techniques to ensure the absence of unspecified functionality. What can we teach them to make this task feasible? The solution lies in the use of constructive security.

Constructive Security

We all learned how to build constructive proofs in high school geometry class. This methodical approach to proof is a computer security educational program's backbone. Of course, our construction involves more tools and techniques than the straight edge, protractor, and compass in our geometry class. As constructive security tools, students use security policy models, formal methods, specification and refinement techniques, system layering, modularity, data hiding, and other development methods along with effective procedures for system lifecycle management. Another difference between geometry and constructive security is in their outcomes. Geometry is pure mathematics, with absolute proofs. Constructive security verifications, a combination of mathematical methods and engineering, are always approximations.

Thus, cybersecurity students must understand that an absolute proof of security is an abstract ideal; the concepts of computer security must be applied to the real world where we actually build components and networks. The best you might achieve is a high degree of confidence that the ideal has been approached.

Models and methods

As an enabling technology, security is a property of systems that provide other services. In this context, students learn how to articulate security along with other requirements. We view automated systems as an extension of ourselves: productivity tools amplify our ability to accomplish the activity at hand. These systems let us manage, manipulate, and communicate information better, more quickly, and with greater ease. We have a set of "dos and don'ts" associated with our information handling such as, "Do put Suzie on the mailing list for our reading group this month," and, "Under no circumstances give away John's personal information to any but a carefully vetted set of authorized individuals." Such policies determine how we want a computer system to behave when handling our information. Of course, because computers lack judgment and do only as they are programmed, we need a verifiable technical interpretation of the policy that lets us express our requirements in a way that permits us to automate them in a system. Ultimately the policy must be mapped to silicon where the decisions are binary: yes/no; on/off.

Policy models and formal methods can tell us whether we are attempting to implement a policy that is logically inconsistent or contains circularities that will prohibit us from implementing a useful system, or if the system does not accurately implement the abstract policy. By studying policies, students can understand the difference between those that are discretionary (add Suzie to the reading group) and those that are global and persistent (always protect John's personal information). To understand whether constructive methods are successful, the policy to be enforced— the objectives—must be known.

We use mechanisms to implement policy, but mechanism and assurance are tightly bound in a highly secure system's construction. Many of the techniques used to construct a useful secure system also contribute to our assurance that it meets its specifications and contains no extra functionality. Constructive security students learn the general principles of protection[2]—the notion of least privilege—the importance of layering, modularity, and information hiding; and gain an appreciation of the challenges in designing a system to avoid overt and covert channels. Students also learn the formal and informal aspects of covert-channel analysis.

It is worth noting that, because so few systems today take advantage of it, far too many members of the security community view elaborate hardware support for security as useless. This is not the result of useless hardware, but a failure to teach students how to use it well. Students in a constructive security program recognize hardware's value as a way to achieve the protection, isolation, and support for the least privilege principle needed to implement security policies. For example, by carefully examining the Intel x86 processor family's or the HP Precision Architecture–Reduced Instruction Set Computer's (PA-RISC) features, students understand how hardware can facilitate protection mechanisms' implementation (which would be either extremely complicated or impossible using simpler processor architectures lacking security features).

To construct a useful implementation, students must know not only the engineering techniques, but also the procedural and social processes of an engineering group. Too often, computer-science programs use individual projects that tend to isolate students. In the world beyond the classroom, building a high-assurance system actually requires a team. Security students must learn to put their egos aside and become players in a cooperative effort. Through collaborative projects, they learn the importance of specifications, and how development team members can provide a system of checks and balances that ensures the translation of high-level specifications into a verifiable concrete implementation. They also learn how to minimize the implementation to demonstrate that every line of code in the system is there because its absence would render the system non-functional and unable to enforce the security policy.

Students in a constructive security program recognize hardware's value as a way to achieve the protection, isolation, and support for the least privilege principle needed to implement security policies.

Educational approach

High on constructive security's learning objectives list is an appreciation of configuration and life cycle management. Such techniques are applicable to a highly secure system's construction, as well as to many of the commodity architectures students will encounter after graduation. Our educational approach is not restricted to high-assurance development. Students learn practical management techniques they can apply to current operational networks and modern approaches on how a combination of well-designed hosts and communications protocols can support the use of distributed systems.

Clearly, it is impossible for educators to teach constructive security if they don't have the materials or framework from which to begin. In an effort to help, the Naval Postgraduate School is engaged in two projects. The first is the development of a commercial-quality interactive computer game intended to convey security concepts as part of an entertaining activity. A player assumes the system administrator's role, maintaining network security while keeping the user population happy and not exceeding the IT budget. Various virtual adversaries make the player's task difficult. An underlying game engine consumes scenarios written in a game-specific language and produces user-level simulations. The game can support a wide variety of scenarios to teach information assurance concepts ranging from the need for non-technical security measures such as user education to the intricacies of discretionary and non-discretionary policy enforcement in distributed networks. Embedded in the game is an encyclopedia, which we are supplementing with ancillary tutorial materials. We are also constructing tools to assist instructors in scenario generation and player assessment, as well as a baseline set of scenarios to convey information assurance concepts and terminology. The initial release of the game will be for our DoD sponsors, and it is anticipated that this version will become the basis of a widely available product.

The second project is the Trusted Computing Exemplar (TCX). For over a decade, the construction of highly trusted computing systems has been a neglected and almost forgotten discipline. At the Naval Postgraduate School, we have assembled a group with the experience and critical mass required to build a high assurance system. In the TCX project, we intend to bring that experience to the larger community by constructing a high-assurance separation kernel. When completed, we'll use the kernel in several appliance-like components in our high-assurance multilevel security architecture. What makes the TCX project unusual is that it is the first high-assurance development effort that will reveal the inner process of constructive security from start to finish. All aspects of the project will be open so that both academia and industry can benefit from its exposition.

We hope that these two projects not only provide a context in which to teach constructive security principles to our own students, they also will help others in the educational community convey these essential ideas to the next generation of security experts and system developers.

An educational program that recognizes constructive security as a fundamental principle underlying IT system design and teaches students how to build security into systems will let us break the costly cycle of attack and repair that consumes many of our resources today. The security built into systems by our graduates will then enable our grander visions for the information age to become reality. □

References

1. P.A. Karger, and R.R. Schell, "Thirty Years Later: The Lessons from the Multics Security Evaluation," *Proc. Annual Computer Security Application Conf.*, Dec. 2002, pp. 119-126; http://csdl.computer.org/comp/proceedings/acsac/2002/1828/00/1828toc.htm.
2. J.H. Saltzer and M.D. Schroeder, "Protection of Information in Computer Systems," *Proc. IEEE*, vol. 63, no. 9, Sept. 1975, pp. 1278–1308.

Cynthia E. Irvine *is an associate professor of Computer Science, and Director of the Naval Postgraduate School Center for Information Systems Security Studies and Research. Her research interests include information assurance in the context of computer security, multilevel security and high-assurance systems, secure interoperability, network security, and applications for high-assurance systems. She has a BA in physics from Rice University in Houston and a PhD in astronomy from Case Western Reserve University in Cleveland. She is a senior member of the IEEE. Contact her at irvine@nps.navy.mil.*