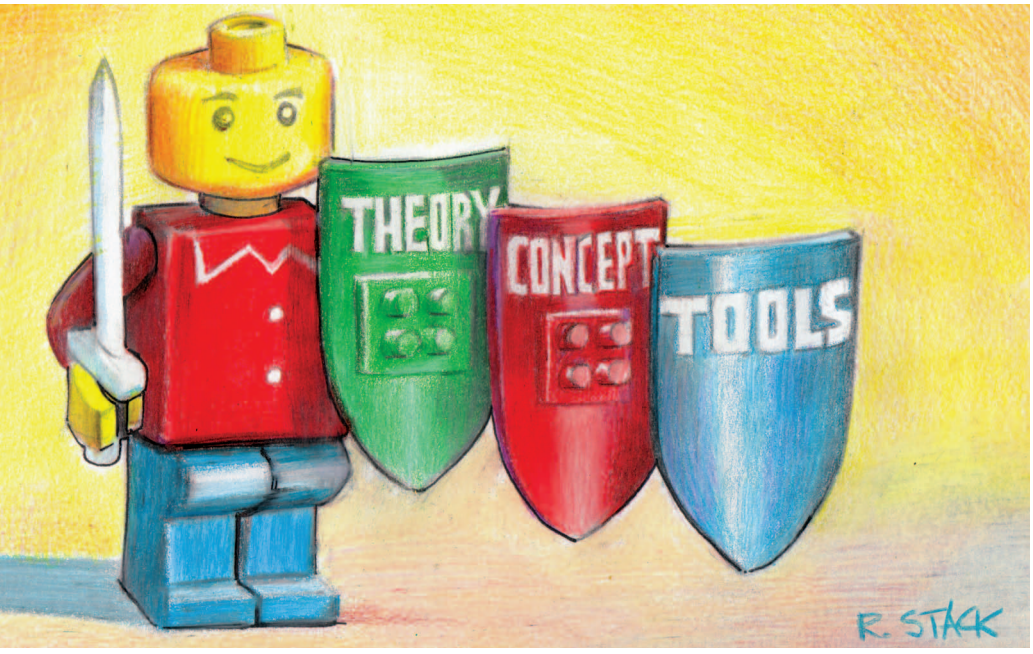
Editors: Cynthia Irvine, irvine@nps.edu | Alec Yasinsac, yasinsac@southalabama.edu

Engineering Assurance at the Undergraduate Level

Susan Older and Shiu-Kai Chin | Syracuse University



There has long been a debate about whether systems designers can develop secure systems on their own or whether development teams must include cybersecurity and assurance experts. Central to (yet often implicit in) this debate are expectations regarding systems designers' capabilities. Because undergraduate computer engineering (CE) and computer science (CS) programs provide the baseline for these capabilities, advancements in undergraduate education could profoundly influence the state of system security.

Our long-term vision is for undergraduate CE and CS programs to routinely produce graduates who can design computer systems that are assured to operate securely. As a step toward that goal, Syracuse University (SU), in partnership with the US Air Force Research Lab (AFRL) Information Directorate and Serco, piloted the 18-credit undergraduate Cyber Engineering Semester (CES) in fall 2011. The objective was to equip undergraduates with three key capabilities. The first was the ability to reason rigorously about security *concepts of operations* (ConOps). The second was the ability to employ security mechanisms as exemplified by Jerome Saltzer and Michael Schroeder's classic paper, "The Protection of Information in Computer Systems."[1] The third was the ability to formally verify ConOps and security mechanisms using computer-assisted reasoning tools.

Our Approach

We designed the CES as a "semester abroad" available to CE and CS majors; the fall 2011 cohort included students from SU, Michigan Technological University, and Texas A&M University. The CES emphasizes mathematical rigor, sound engineering principles, and learning by doing through a combination of five courses and a paid AFRL internship.[2] It also incorporates leadership training and professional development.[3]

Courses and Schedule

Three of the courses integrated security and assurance topics into standard undergraduate CE and CS courses. The Secure Operating Systems course presented security as integral to OS design and emphasized how different OS resources are protected; the Secure Computer Architecture course emphasized virtualization. The Secure Hardware Laboratory course replaced the typical junior CE lab course in which students design and implement processors and controllers using field-programmable gate arrays and modern design tools (for example, hardware description languages). In this version, students implemented a memory management unit for virtual memory en

 Copublished by the IEEE Computer and Reliability Societies

route to building a 12-instruction CPU. The course also enforced secure coding practices.

The Engineering Assurance Laboratory course provided students with the necessary techniques and tools for circumnavigating the "virtuous cycle" of specification, design, and verification. (By virtuous cycle, we mean a collection of tightly coupled, mutually reinforcing activities that increase the degree of assurance and confidence of both the system designers and customers.) The final course, the Cyber Engineering Seminar, provided a real-world, professional problem-solving context for the entire program. Students devised, analyzed, and presented security ConOps for several varied systems. We describe these two courses more fully later.

A key design criterion for the CES was to identify connections across courses so as to mutually reinforce concepts and help students "connect the dots." In particular, we wanted the students to see how they can integrate theory, concepts, and tools to solve real-world problems. Accordingly, CES faculty met biweekly to discuss each course's current topics, the timing of assignments, and how we might incorporate assignments that build on multiple courses. At monthly lunches, faculty and students met to discuss the students' perceptions of the program and to make explicit the connections we wished them to make.

Access Control

Access control—determining who's permitted to perform which operations on which objects under what circumstances—is an important security theme that emerges across domains. Establishing and maintaining confidentiality, integrity, and availability depend on access-control policies at all levels of the system stack.

For example, at the hardware level, the concern is access to physical memory; the OS-level issues are access to resources and capabilities among processes. At the network and even policy levels, the designer must account for recognized authorities, their jurisdiction, their possible delegates, and the flow of command and control. At each level, the same question arises: should the system accept or reject a given request, on the basis of the given policies, trust assumptions, and authorizations? Moreover, the designer should have a way to provide convincing justification for that answer.

With this view in mind, we used the access-control logic described in our textbook *Access Control, Security, and Trust: A Logical Approach*[4] as a common language for thinking and reasoning about security and integrity across domains. Despite this logic's simple syntax, it expresses a variety of security-related concepts, including

- principals,
- the statements and requests they make,
- access-control policies,
- authorities and their jurisdictions,
- credentials and other certificates, and
- delegation.

Throughout the semester, students applied this logic at different abstraction levels, from the control of physical memory and network protocols to organizations' security and integrity policies.

Computer-Assisted Reasoning

In the Engineering Assurance Laboratory, we introduced operational semantics and transition systems as additional techniques for specifying and verifying desired system behavior. Such techniques alone are insufficient, however: computer-assisted reasoning tools are necessary for practical verification and the credible dissemination of results. (The situation is analogous to hardware design, which depends on CAD tools such as design-rule checkers, logic-to-layout verifiers, and model checkers.)

Hence, the Engineering Assurance Laboratory introduced both functional programming and theorem provers. We used the Haskell functional programming language to rapidly prototype ideas, create executable specifications, and perform property-based testing (via QuickCheck[5]). By providing students experience with functional programming and a strong type system, Haskell also helped prepare them to work with the HOL (higher-order logic) theorem prover.[6] (In particular, we used HOL4; http://hol.sourceforge.net.)

We interleaved Haskell and HOL throughout the semester, using them to shed light on the access-control logic, operational semantics, and transition systems. For example, students defined the access-control logic's semantics in Haskell to implement a simple model checker. We also provided students with an HOL implementation of the access-control logic that embedded its syntax and semantics, core inference rules, and the proofs of underlying theorems as conservative extensions to HOL. This implementation provided the basis for machine-checking their proofs throughout the semester.

What We Asked Students to Do

All five courses and the AFRL internship employed the access-control logic or its concepts. By the semester's end, students had applied the logic across three courses and the internship to describe and identify flaws in a variety of architectures, protocols, and operations.

In the Secure Hardware Laboratory, students submitted proofs that their memory management

designs satisfied certain properties. In the Cyber Engineering Seminar, students had to formally model the PlayStation Network's architecture and use those models to identify possible security flaws. They also had to analyze a secure email protocol (RFC 1421) and use the access-control logic to identify the foundations for trust in the RFC (request for comments). As part of the AFRL internship, they analyzed and corrected two flawed protocols for secure communication and control in the cloud.

An extended project in the Engineering Assurance Laboratory helped students execute the virtuous cycle of specification, design, and verification. Building on concepts introduced in the Cyber Engineering Seminar and internship, this project focused on the command-and-control ConOps for a weapon requiring dual authorization for launch. The problem scenario involved coalition operations between a Blue Force and Gold Force, each with a commander, operator, and certificate authority (CA). A Joint Forces CA certified each force's CA. The problem statement specified CAs, keys, and means for authentication; mission roles and their jurisdictions; weapons launch and abort policies; and role assignments and authorizations.

Students defined and verified the weapons launch and abort ConOps using the access-control logic in HOL. They also had to prove their authentication ConOps' validity on the basis of role authorization and key assignments in HOL. Specifically, they had to

- devise the jurisdiction statements, policy statements, and trust assumptions;
- express them using the HOL implementation; and
- prove the validity of inference rules corresponding to weapons launch and abort in HOL.

For this problem, they refined, described, and verified their ConOps at three abstraction levels:

- the highest level, with only roles;
- a middle level, with mission staff assigned to roles; and
- the lowest level, with cryptographic keys assigned to staff.

The final refinement for this project required students to define the semantics of specific message and certificate formats using the access-control logic in HOL. This step enabled them to describe and verify their designs' behavior using concrete message and certificate formats.

The key to our success has been the explicit choice to take a narrow, continuous thread throughout the courses and internship.

Program Assessment and Lessons Learned

As instructors, we had three major questions when the CES began:

1. Were we asking too much?
2. Would our five courses actually reinforce security and integrity concepts so that students would be able to connect the dots?
3. Would students actually be able to do formal proofs of correctness using tools (for example, theorem provers) generally thought to be beyond undergraduates' grasp?

The answers, in short, are (1) almost, (2) mostly, and (3) yes.

Expectations

Students and faculty alike found the CES to be intense, largely owing to the time demands. Students spent roughly 32 hours each week in the classroom and internship. The pilot program's small size let us make midsemester adjustments to alleviate some time pressure and the associated stress. After teaching these courses once, however, we believe we can better use our instruction time to reduce time pressure without reducing student achievement.

Connections

The monthly lunches provided important feedback on the degree to which students could make connections between concepts introduced across courses. When we initially queried students on the connections they were making, we realized that some conceptual leaps were too broad for them to make on their own. We therefore changed tack and articulated the specific connections we intended them to make. Toward the semester's end, one student stated that the extended ConOps project pulled together all the concepts (security policies, organizational roles, delegates acting in roles, jurisdiction, and trust) for him; the other students concurred. Their suggestion (which we plan to honor) was to introduce such a project much earlier in the semester.

Student Achievement

As we discussed earlier, the extended project tested whether students, given an informal description of a high-level problem, could

- devise a ConOps,
- formalize it in the access-control logic, and
- verify its validity in HOL.

Using HOL made assessment both easy and objective: the instructor checked whether HOL accepted their proofs and whether it generated the list of theorems he

expected. Four of the six students completed the full assignment, generating the requisite collection of theorems. The remaining two students proved at least 50 percent of the expected theorems.

The CES students were users of the access-control-logic implementation, not HOL experts. However, they could wield HOL well enough in this domain to get results that they recognized as useful and that passed inspection by an expert (that is, the instructor). Although the students initially found HOL challenging, they reported that they valued the assurance it provided. Whereas they doubted their unchecked hand proofs' correctness, they were confident in their HOL-checked proofs.

The CES provided compelling evidence that mathematically rigorous assurance is feasible in a way that's relevant beyond one course. Because the students came from multiple institutions, we designed the CES on common CE and CS foundations: discrete mathematics, computer organization, and programming experience with a high-level language. Thus, we believe our experiences and results are applicable beyond SU.

We used a small collection of formal methods and tools that fit together nicely and were relevant for applications in which security and access control are important. By the semester's end, students weren't experts in modal logic, Haskell, or HOL. However, they were competent users of these tools who could evaluate their and others' work. The participation of AFRL and Serco—and particularly the internship—was invaluable in helping students see the formal methods' relevance.

There's nothing particularly special about the use of access control, modal logic, functional programming, or theorem provers. What is special, however, is the integration of these concepts and tools in a way that enables undergraduates to devise policies and build systems that incorporate access control and are amenable to rigorous, independent verification. The key to our success has been the explicit choice to take a narrow, continuous thread throughout the courses and internship. By narrowing the breadth, we achieved significant depth.

Our long-term vision is that the ability to build and assure secure systems is a routine outcome of undergraduate CE and CS curricula. The CES provides a testbed for refining our ideas on what's possible and how best to achieve this goal. As this article goes to press, we're in the midst of the second CES offering, with students from five institutions. This time, all five courses will employ Haskell, HOL, or the access-control logic. We're also exploring how we can apply what we've learned to our standard CE and CS programs. We welcome opportunities to share details of our experiences and look forward to learning from other similar efforts. ■

References

1. J. Saltzer and M. Schroeder, "The Protection of Information in Computer Systems," *Proc. IEEE*, vol. 63, no. 9, 1975, pp. 1278–1308.
2. S.-K. Chin et al., "Formal Verification for Mission Assurance in Cyberspace: Education, Tools, and Results," *Proc. 16th Colloquium Information Systems Security Education*, Colloquium for Information Systems Security Education, 2012.
3. E. Devendorf, F. Wieners, and S. Muccio, "Developing the Next Generation of Cyber Leaders," *Proc. 16th Colloquium Information Systems Security Education*, Colloquium for Information Systems Security Education, 2012.
4. S.-K. Chin and S. Older, *Access Control, Security, and Trust: A Logical Approach*, CRC Press, 2011.
5. K. Claessen and J. Hughes, "QuickCheck: A Lightweight Tool for Random Testing of Haskell Programs," *ACM SIGPLAN Notices*, vol. 35, no. 9, 2000, pp. 268–279.
6. M.J.C. Gordon and T.F. Melham, *Introduction to HOL: A Theorem-Proving Environment for Higher-Order Logic*, Cambridge Univ. Press, 1993.

Susan Older is an associate professor in Syracuse University's Department of Electrical Engineering and Computer Science. Contact her at sueo@ecs.syr.edu.

Shiu-Kai Chin is a professor in Syracuse University's Department of Electrical Engineering and Computer Science. Contact him at skchin@syr.edu.