Building a Rigorous Foundation for Assurance into Information Assurance Education

Susan Older and Shiu-Kai Chin
Center for Systems Assurance
Department of Electrical Engineering and Computer Science
Syracuse University
Syracuse, NY 13244
Email: sueo@ecs.syr.edu, skchin@syr.edu

Abstract— Syracuse University is one of thirty-six National Security Agency designated Centers of Academic Excellence in Information Assurance Education. Our IA program was developed within the Center for Systems Assurance (CSA), whose mission is to promote improvement in systems and information assurance through research, education, and technology transfer.

The goal of the CSA educational program is to develop students with a broad background in security and information assurance who distinguish themselves by their ability to (1) analyze, synthesize, and make judgments based on engineering and computer-science principles, and (2) use analytical techniques to evaluate the implications of policies, standards, and procedures; the ramifications of changes; and the potential dangers of refinements.

A key—and, we believe, unique—component of our program is our emphasis on using formal mathematics and logic to provide a rigorous basis for the assurance of information and information systems. All students in our program must take a combination of courses that provide hands-on experience both in building systems and in using formal models to analyze and evaluate system behavior.

In this paper, we discuss our experiences in developing and delivering a Systems Assurance program in which mathematical logic is an integral component.

I. Introduction

According to the National Security Agency's definition [1], Information Assurance is the use of

> "[i]nformation operations ... [that] protect and defend information and information systems by ensuring their availability, integrity, authentication, confidentiality, and nonrepudiation. This includes providing for restoration of information systems by incorporating protection, detection, and reaction capabilities."

At the Center for Systems Assurance (CSA) at Syracuse University, we have adopted a concept of Systems Assurance that extends this definition by emphasizing the need for high confidence that a system behaves correctly, including its availability, integrity, confidentiality, and scalability.

This emphasis on high-confidence design permeates both our research and our educational efforts. CSA's educational purpose is to prepare students to design, develop, and deploy complex systems with confidence in the systems' ability to meet assurance requirements. With this purpose in mind, we have developed a graduate-level Certificate of Advanced Study in Systems Assurance. Our goal is that the successful students of our Systems Assurance program will develop a broad background in security and information assurance, distinguishing themselves by their ability to

1) analyze, synthesize, and make judgments based on engineering and computer science principles; and
2) use analytical techniques to evaluate the implications of policies, standards, and procedures; the ramifications of changes; and the potential dangers of refinements.

A key—and, we believe, unique—component of our program is our emphasis on using formal mathematics and logic to provide a rigorous basis for the assurance of information and information systems. Security is an intricate property that is achieved only through a combination of sufficiently strong cryptographic algorithms and protocols, correct implementation of hardware and software, and appropriate assumptions about trusted authorities. Assuring that all of these factors are present and correctly integrated to form a secure system is difficult—if not impossible—without the use of rigorous and formal analytical techniques.

In this paper, we discuss our experiences in developing a Systems Assurance program in which mathematical logic is an integral component. Section II describes the high-level educational outcomes of our program, while Section III summarizes the primary requirements of our program. In Section IV, we describe three separate courses in our program that demonstrate our approach to Information Assurance education. In Section V, we discuss some of the challenges we have encountered in teaching these courses, as well as some student feedback. We conclude in Section VI with a summary of our experiences.

II. Educational Outcomes

Syracuse University's Center for Systems Assurance (CSA) was established in July 2000 as a center within the Department of Electrical Engineering and Computer Science (EECS), with the mission of promoting improvement in systems and information assurance through research, education, and technology transfer.

Within that context, CSA developed a Master's-level Certificate of Advanced Study in Systems Assurance (CASSA). This program is available to both Computer Engineering (CSE) and Computer Science (CIS) graduate students. We are now in the process of establishing the CSA as a multidisciplinary center within Syracuse University. The model CSA has adopted for its educational programs allows each academic unit to define its own discipline-specific foundation, while taking advantage of other units' offerings for electives to produce well-rounded students.

The CSE and CIS Certificate of Advanced Study embodies this approach, incorporating the EECS department's long-standing emphasis on the use of mathematical and logical methods in engineering, computer science, and security. For example, the department's top-level educational outcomes for students in Computer Engineering include the following three outcomes:

- An understanding of principles of construction of computer systems
  Principles of construction that have a formal mathematical basis (such as linear systems theory) offer the benefits of component-based design, predictability, and management of complexity. The key characteristic is compositional reasoning where the behavior of arbitrarily complex structures can be equated to purely behavioral descriptions.
- An understanding of abstraction and refinement of computing systems
  Being able to relate abstract descriptions of behavior with concrete implementations provides a way to manage complexity, support reuse, and validate notions of correctness. How implementations or concrete designs are derived or refined from abstract descriptions or specifications is one important aspect of design.
- An ability to Predict and reason about correctness and performance of computing systems
  An important part of design is the ability to evaluate the quality of design. Specifically, engineers must be able to evaluate the correctness and performance of a design prior to actual construction. Central to the evaluation process is the ability to use mathematical and logical techniques to calculate correctness and performance. Whereas the previous educational outcome emphasized the process of refining a specification into an implementation, this outcome emphasizes the techniques used to evaluate design quality.

Similarly, the department's goal for its Computer Science program is to produce graduates with the following attributes, among others:

- general problem-solving skills and, in particular, some facility for applying mathematics to the solution of problems;
- practical experience in carrying out a proposed solution of a computational problem in a principled and organized fashion;
- experience in taking a proposed solution to a computational problem and analyzing its correctness and efficiency.

In developing the CASSA program, we focused on our expectations for those students who successfully complete our program. Our goal was to develop a comprehensive, coherent collection of courses to ensure the following educational outcomes:

1) Students comprehend the concepts underlying security and system assurance.
2) Students can apply those concepts to construct assured systems.
3) Students can critically analyze and evaluate systems' conformance to their requirements.

These outcomes also reflect the department's top-level outcomes for our Computer Engineering and Computer Science graduates

III. Program Requirements

The CASSA program builds on foundations established in standard undergraduate Computer Science and Computer Engineering programs. Incoming students are expected to have undergraduate-level mastery of discrete mathematics, predicate calculus, algorithms, networking, and operating systems. This mastery provides the necessary foundation for advance courses that, taken together, provide both hands-on systems-building experience and practice with applying rigorous techniques for evaluating system assurance.

Providing secure systems involves three activities: protection, detection (when protection fails), and reaction (recovery upon intrusion detection). Furthermore, making rigorous arguments about system assurance requires tools for specifying desired system behavior and relating implementations to those behavioral specifications. Therefore, all students must take a combination of courses that provide hands-on experience both in building systems and in using formal models to analyze and evaluate system behavior.

To enhance their breadth of understanding, students must also take at least one course covering nontechnical aspects of security, such as ethics, legal issues, privacy, public policy, management, and e-commerce. To promote

CIS 628	Introduction to Cryptography
CIS 632	Modeling Concurrent Systems
CSE 774	Principles of Network Security
CIS 690	IA Foundations Independent Project

TABLE I
Foundations for Assurance track

CSE 758	Internet Security
CSE 785	Computer Security
CIS 690	IA Applications Independent Project

TABLE II
Assurance Applications track

discussions and to increase exposure to these myriad topics, students must also participate in the Systems Assurance Seminar.

Because the CASSA program exists within the framework of the standard CIS and CSE Master's programs, students must satisfy all requirements of their home program. In addition, the students must satisfy several CASSA-specific requirements. The primary requirements are as follows:

- Students must complete the CIS or CSE core, as appropriate.
  The CIS core consists of four courses: CIS 623 (Formal Methods & Structured Programming), CIS 655 (Computer Architecture), CIS 657 (Operating Systems), and CIS 675 (Analysis of Algorithms).
  The CSE core consists of three courses: CIS 607 (Logical Basis for Computing), CSE 661 (Advanced Computer Architecture), and CSE 681 (Software Modeling & Analysis).
- Students must successfully complete CIS/CSE 583, the Systems Assurance Seminar.
  This seminar course serves as a gateway to more advanced Assurance courses, introducing basic terminology and many of the nontechnical issues related to information assurance.
- Students must successfully complete a total of five courses from the Foundations for Assurance and Assurance Applications tracks, including at least two courses from each track. These tracks are listed in Table I and Table II. The catalog descriptions for these courses appear in the Appendix.
  Via this requirement, students are assured to receive hands-on experience both in building systems and in using formal methods.
- Students must successfully complete one course from the list of Nontechnical Assurance electives in Table III.

IST 618	Survey of Telecom. & Information Policy
IST 642	Electronic Commerce
IST 643	U.S. Federal Information Policies
IST 682	Telecommunications Policy and Regulation
LAW 779	Internet Law

TABLE III
Nontechnical Assurance electives

  The purpose of this requirement is to enhance students' breadth of IA-related knowledge.

IV. Selected Examples From Courses

As the CASSA requirements indicate, we view mathematical rigor as an integral component of Information Assurance education: we want our students to be able not only to build secure systems but to analyze them critically.

In this section, we describe three separate courses in our program that demonstrate our approach to incorporating mathematical logic into Information Assurance education. These three courses are CSE 607 (Logical Basis for Computing), CIS 632 (Modeling Concurrent Systems), and CSE 774 (Principles of Network Security). All three courses are open to all Computer Science and Computer Engineering graduate students.

Two of the three courses are part of the Foundations for Assurance track; the third (CSE 607) is a required core course for the CSE Master's program and a prerequisite for CSE 774. Of the three, only CSE 774 is strictly speaking a security course. However, the other two courses include security-related topics, and both embody CSA's broader definition of Systems Assurance.

A. CSE 607 – Logical Basis for Computing

One objective of this course is to enable computer engineering students to use predicate calculus to reason about specifications and implementations and how they are related, much as they use Laplace transforms to reason about communications systems.

In this course, students do formal proofs using the tableau system put forth by Manna and Waldinger [2]. The tableaux are tables with lists of assumptions $A_1 \cdots A_n$ and goals $G_1 \cdots G_m$. The interpretation of a table with assumptions $A_1 \cdots A_n$ and goals $G_1 \cdots G_m$ is

$$[((\forall *).A_1) \wedge \cdots \wedge ((\forall *).A_n)] \supset [((\exists *).G_1) \vee \cdots \vee ((\exists *).G_m)],$$

where $(\forall *).(\mathcal{A})$ denotes the universal quantification of all free variables in $\mathcal{A}$ and $(\exists *).(\mathcal{G})$ denotes the existential quantification of all free variables in $\mathcal{G}$.

The inference rules in the tableau system manipulate goals and assumptions in such a way that the validity of the interpretation is unchanged. The inference rules include unification, resolution, and splitting of goals and assumptions. These rules are much like the inference rules used by theorem provers and proof checkers such as the HOL (Higher Order Logic) system [3].

The course starts with a simplified tableau reasoning system for propositional logic followed by the introduction of predicate calculus. By the end of the course students are using structural induction to reason about natural numbers and lists.

Students are also expected to apply formal logic to computing or engineering applications. In Fall 2001, for example, students were required to read Ferraiolo, Barkley, and Kuhn's paper on role-based access control [4] and to formally prove all the theorems and corollaries in the paper. For example, one such theorem concerns the consistency of static separation of duty and containment:

> If non-empty roles i and j are mutually exclusive, then neither can contain the other.

Formally, this theorem can be stated as follows, where $RM[i]$ denotes the set of users authorized to role i, E_a is a set of pairs of mutually exclusive roles, and $\succeq$ is the containment relation:

$$\forall i, j : role.((RM[i] \neq \emptyset \wedge RM[j] \neq \emptyset) \wedge (i, j) \in E_a) \supset \neg(i \succeq j \vee j \succeq i).$$

One benefit for students of formally proving the theorems is that they are able to get a precise understanding of the definition of roles, containment, and their properties.

B. CIS 632 – Modeling Concurrent Systems

The primary goal of this course is to provide students with an in-depth understanding of one particular approach—the process-algebraic approach—for specifying, modeling, and analyzing system behavior. Process algebras such as CSP [5], [6] and CCS [7] provide a way to describe system behavior in terms of the events (i.e., abstract actions deemed "observable") that can occur. The underlying theory also includes several useful notions of program equivalence and refinement, which are useful for compositional reasoning and analysis.

In the most recent offering of the course, we focused on CSP and its use in specifying and verifying concurrent systems and in understanding the emergent behavior of such systems. The first portion of the semester is spent on introducing the basic operators of CSP, their operational semantics, and the trace and failures models, both of which induce notions of refinement. The trace model of CSP, for example, formally describes a process's behavior in terms of the set of traces—that is, sequences of events—that it can perform. These trace sets provide a basis for comparing, equating, and refining processes. Two processes are trace-equivalent when they have precisely the same sets of traces. A process Q refines P in the trace model (written $P \sqsubseteq_T Q$) if every trace of Q is also a trace of P. Intuitively, if P corresponds to a specification of permissible behavior and Q refines P, then Q is guaranteed to exhibit only permissible behaviors.

We also introduce the model checker FDR2 [8], which provides automated support for applying these notions in practice. The advantage of using a model checker is that one can analyze much larger systems than is possible by hand. In addition, model checkers can be useful in debugging designs: for example, when a desired refinement $P \sqsubseteq_T Q$ fails, FDR2 produces a witness to the failure (i.e., a trace of Q that is not allowed by the specification P).

The final third of the course is spent on using CSP and FDR2 to analyze several different problems and protocols. Among these applications, we discuss Gavin Lowe's use of CSP and FDR2 to uncover a previously unknown flaw in the Needham-Schroeder key-exchange protocol [9]. This analysis uses CSP to model the system as the parallel composition of an initiator, responder, and intruder; CSP is also used to describe the desired initiator and responder authentication properties. FDR2 can then be used to try to validate refinement relationships between the system and the two desired properties. The result of FDR2's analysis is a witness trace that highlights a possible man-in-the-middle attack.

As an integral part of the course, students are required to do a final project and write a report describing their work: each project must use CSP to implement and analyze a protocol or other design. In Fall 2001, 8 out of 23 students chose a security-related analysis for their final project.

C. CSE 774 – Principles of Network Security

This course focuses on cryptographic protocols—e.g., key distribution protocols and X.509 credentials—and their properties. Higher-order logic and several specialized formal systems, such as belief logics and logics for reasoning about principals and their statements [10], are used to describe and verify protocols and their properties.

A significant part of the course is dedicated to reasoning about credentials, networks of certification authorities, and how cryptographic keys come to be trusted (i.e., the basis for concluding that a particular key speaks for or represents a principal acting in a particular role). The course uses the calculus of Abadi to reason about principals and their statements [10], and closely examines the TAOS operating system as a detailed example [11]. The fundamental theorem within this example relates credentials, principals, and statements about authorization:

$$\vdash (\forall x : credential).M(x) \supset (Q(x) \text{ speaks for } P(x)).$$

In this theorem, x is a credential (boot certificate, role membership certificate, delegation certificate, channel certificate, domain certificate, etc.) whose interpretation is defined by the meaning function $M(x)$, and $P(x)$ and $Q(x)$ are both principals.

In effect, this theorem states that, if the credentials used in the Taos operating system are valid as defined by the interpretation $M(x)$, then the relationship between principals $P(x)$ and $Q(x)$ is also valid.

V. Challenges and Reactions

There are multiple challenges inherent in incorporating mathematical and logical rigor in an Information Assurance curriculum.

The first challenge is that very few textbooks use mathematical logic to describe and verify IA-related designs, systems, and protocols. As a result, we typically must develop and write many of the examples ourselves. We have had success in incorporating research papers in our instruction, but doing so is not without its own difficulties: every author chooses his own notation, and it can be confusing for students to continually have to adapt to new notation while also trying to integrate new concepts.

The second challenge is that students come into our program with different levels of preparation and experience using mathematical logic. Furthermore, many students are initially resistant to our demands for mathematical rigor: many view discrete mathematics and logic as necessary evils of a computer-science or computer-engineering program, but they are doubtful of its use in "real" work. Others are intimidated by the subject matter and hope to be able to skate by the math en route to the more hands-on aspects of security. The lack of IA-related textbooks that use mathematical logic further encourages such views.

Despite the challenges, students have responded positively to these courses. For example, two students provided the following anonymous comments in their CSE 607 course evaluations in Fall 2001:

> "I particularly enjoyed working on the security paper. Thank you. I enjoyed the course and am looking forward to taking CSE 774 [Principles of Network Security]."

> "It would be great if we [had] another example like 'Role-based access control' or hardware design to put the logic in use, engineering use."

Similarly, one student in the Fall 2001 offering of CIS 632 chose to use CSP to analyze the Diffie-Helman key-agreement protocol. This student concluded his final-project write-up with the following paragraph:

> "This class and project has given me a very strong thirst for the field of computer security in general. Throughout college, I was uninterested in the mathematical aspects of cryptography and was slightly turned off to it for that reason, but I liked being able to analyze something such as a protocol, and then proving its validity under certain conditions. CSP is a method for formalizing specifications which seems to make a lot of sense to me and seems very practical, so it's easy to apply to a system."

VI. Conclusions

Security properties are affected by timing and timeliness (e.g., present versus past runs of a protocol), trust (or lack thereof) in various principals and authorities, and cryptographic properties. Furthermore, there are nuances that arise between different levels of abstraction. The value of formal methods is that they help in the detection of weaknesses and possible attacks, as well as to make explicit any necessary assumptions being made. More to the point, using formal and rigorous methods enables one to perform a principle-based examination of the details upon which information assurance depends.

Such formal methods are useful, however, only when people are sufficiently knowledgeable about how to use them. Our experience is that many of the available methods are accessible to engineers with an understanding of predicate logic. We have taught several formalisms—including process algebras and specialized logics—to both computer-science and computer-engineering students, at both the graduate and undergraduate level. Once students are able to do formal analysis, they are better able to consider the implication and consequences of their designs.

Finally, because students learn and use different formal techniques across several courses, they can begin to appreciate how different methods are appropriate for different purposes. Different logical systems and formal models can provide insight into security issues from a variety of viewpoints. In our experience, the choice of a particular formalism is driven in part by the properties one wishes to prove and also by the tools available. Experience with multiple formalisms not only helps students identify the appropriate formalisms to use in new situations but also gives them the confidence to learn new formalisms on their own as assurance needs change.

References

[1] National Security Telecommunications and Information Systems Security Instruction (NSTISSI) No. 4009, January 1999.

[2] Z. Manna and R. Waldinger. The Deductive Foundations of Computer Programming. Addison-Wesley, New York, 1993.

[3] M.J.C. Gordon and T.F. Melham. Introduction to HOL: A theorem proving environment for higher order logic. Cambridge University Press, New York, 1993.

[4] David F. Ferraiolo, John F. Barkley, and D. Richard Kuhn. A role-based access control model and reference implementation within a corporate intranet. ACM Transactions on Information and Systems Security, 2(1):34–64, February 1999.

[5] C. A. R. Hoare. Communicating Sequential Processes. Series in Computer Science. Prentice Hall, London, 1985.
[6] A. W. Roscoe. The Theory and Practice of Concurrency. Series in Computer Science. Prentice Hall, London, 1998.
[7] Robin Milner. A Calculus of Communicating Systems, volume 92 of Lecture Notes in Computer Science. Springer-Verlag, 1980.
[8] Formal Systems (Europe) Ltd, Oxford. Failures-Divergence Refinement: FDR2 User Manual, October 1997.
[9] Gavin Lowe. Breaking and fixing the needham-schroeder public-key protocol using fdr. Software Concepts and Tools, 17:93–102, 1996.
[10] Martin Abadi, Michael Burrows, Butler Lampson, and Gordon Plotkin. A calculus for access control in distributed systems. ACM Transactions on Programming Languages and Systems, 15(4):706–734, September 1993.
[11] Edward Wobber, Martin Abadi, Michael Burrows, and Butler Lampson. Authentication in the Taos Operating System. ACM Transactions on Computer Systems, 12(1):3–32, February 1994.

Appendix

CIS 583 (Systems Assurance Seminar)
: Basic terminology of assurance and cryptography. Social and privacy issues of assurance. Ethics in computing. Legal aspects and implications of systems assurance. Weekly presentations by external and internal speakers.

CIS 628 (Introduction to Cryptography)
: An introduction to classical and public-key cryptography. Topics include: classical cryptosystems and their cryptanalysis, RSA and other public-key cryptosystems, pseudo-random sequences, zero-knowledge protocols, related ethical and social concerns.

CIS 632 (Modeling Concurrent Systems)
: Formal methods for specifying, modeling, and analyzing concurrent systems, and the mathematical basis for such methods. Automated and semi-automated tools to apply these methods to analyze emergent behavior of computing-related applications.

CSE 758 (Internet Security)
: TCP/IP architectural model. IP addressing. TCP/IP network protocol security. Network layer and application layer security. Firewalls. Router security. Denial of service attacks. Domain name server attacks.

CSE 774 (Principles of Network Security)
: Specification, verification, and design of secure networks using formal logic. Includes cryptographic algorithms and reasoning about key distribution protocols, delegation, access control, electronic mail, and certification authorities.

CSE 785 (Computer Security)
: Traditional cryptography. Private-key versus public-key cryptography. Steganography. Digital signatures. Key certificates. Unix security. Theft of information. Theft of service.