THE GEORGE WASHINGTON UNIVERSITY
WASHINGTON DC

THE GEORGE WASHINGTON UNIVERSITY
CYBER SECURITY POLICY AND RESEARCH INSTITUTE

Thoughtful Analysis of Cyber Security Issues

UCDAVIS
UNIVERSITY OF CALIFORNIA

UCDAVIS
COLLEGE OF ENGINEERING

Summit on Education in Secure Software
Final Report

Dr. Diana L. Burley
The George Washington University

Dr. Matt Bishop
University of California, Davis

GW: Report GW-CSPRI-2011-7
UCD: Technical Report CSE-2011-15

Support for this work was provided through the National Science Foundation Directorate for Computer & Information Science & Engineering and the Directorate for Education & Human Resources under Award #1039564. Opinions expressed, conclusions drawn, and recommendations provided do not necessarily reflect the views of the National Science Foundation.

June 30, 2011

Summit in Secure Education: Final Report

Table of Contents

Executive Summary 1
Summit on Education in Secure Software 5
- **Background** 5
- **Summit on Education in Secure Software** 6
- **SESS Objectives** 6
- **SESS Teleconference** 7

Secure Programming Education: Requirements and Challenges 7
- **Summary of Requirements** 7
- **Details of Requirements** 8
- **Summary of Challenges** 9
- **Details of Challenges** 10

The Roadmaps (and Potholes) 11
- **The Roadmaps** 12
- **Computer Science Students** 13
- **Non-Computer Science Students** 17
- **Community College Students** 21
- **K-12 Students** 24
- **Computer Science Professionals** 27
- **Non-Computer Science Professionals** 30

The Role of Higher Education 34
Licensure and Certification 35
Conclusion 37

Appendix A – Summit Overview 41
Appendix B – Organizing Discussion Questions 46
Appendix C – Summit Agenda 47
Appendix D – Participant List 49
Appendix E – Licensure/Certification Presentation Summary 52
About the Authors 58

Summit on Education in Secure Software Final Report

Dr. Diana L. Burley and Dr. Matt Bishop

Executive Summary

Cybersecurity risks pose some of the most serious economic and national security challenges of the 21st century. In order to meet these challenges, the software that controls critical systems and infrastructure must be reliable, robust, and able to satisfy the requirements that are placed upon it. To this end, all people involved in the development and deployment of these systems and infrastructure, from the policymakers who determine what requirements the systems must meet to the businesspeople who provide the support needed to create the systems to the architects, implementers, and operators of these systems, must understand the criticality of reliable, robust, and secure software to control these systems.

The notion of "trustworthy computing" embodies many more facets of computing than software implementation, but poorly implemented software undermines all other aspects of trustworthy computing. Thus, a curriculum designed to meet the cybersecurity challenges of the future must integrate principles and practices of secure programming. Because of the breadth of people who will influence the creation and deployment of this software, students studying a variety of technical and non-technical disciplines must recognize the difference between software that is sufficiently robust and software that is not.

To determine how best to address this need, the National Science Foundation Directorates of Computer and Information Science and Engineering (CISE) and Education and Human Resources (EHR) jointly sponsored the Summit on Education in Secure Software (SESS). The summit focused on how best to educate students and current professionals on secure programming concepts and practices, and to provide roadmaps indicating both the resources required and the problems that had to be overcome. Organized by The George Washington University and the University of California at Davis, the summit began with a teleconference on September 7, 2010 and continued with a two-day workshop in Washington, DC on October 18 and 19, 2010.

SESS participants included members of academia, government, industry, professional organizations, and policy makers from both the public and private sectors. This multi-disciplinary group provided depth to identify technical challenges and breadth to identify operational constraints and opportunities. This enabled the summit participants to examine ways to advance and improve the state of education in secure software.

The goal of SESS was to develop a comprehensive agenda focused on the challenges of secure software education. To meet this goal, SESS had three specific objectives.

1. To have cybersecurity stakeholders from academia, government, industry, and certification and training institutions discuss the goals of teaching secure programming and the current state of that teaching;

2. To use that discussion as the basis of a collaborative effort to suggest new approaches, and improve existing approaches, to improve the quality of that education and to enable it to reach a broader audience; and

3. To outline a comprehensive agenda for secure software education that includes objectives for different audiences, teaching methods, resources needed, and problems that are foreseen to arise.

Summary of the Findings

The findings are presented in the form of "road maps" for constituent groups that describe ways to improve the state of education in secure programming. The road maps explain what the members of the constituent group should know, various methods by which they might be educated, and what resources will be necessary to achieve that level of education. The road maps also identify expected and possible challenges to meeting these goals—the "potholes". Each roadmap concludes with specific recommendations for meeting the articulated educational goals.

Although presented as separate road maps, one for each constituent audience (computer science students; non-computer science students; community college students; K-12 students; computer science professionals; and non-computer science professionals), the roadmaps should be considered as different views of a unified educational program that spans kindergarten through professional development, and audiences ranging from software developers to those who focus on management, policy and use to ordinary computer users. Thus, the educational goals in the road maps should be considered the "core fundamental end points" to be reached through instruction guided by a secure programming curriculum.

Overall, SESS participants asserted that secure programming must be considered within the context of the system design and deployment process. They highlighted 6 critical points that students of secure programming should know:

1. Understanding security, especially during design, requires a holistic approach;

2. Understanding and being able to identify common and emerging attack vectors is a critical component of security;

3. Well-tested principles and frameworks of software development can inhibit attacks;

4. All frameworks have weaknesses and subtleties;

5. Part of secure programming is using strategic approaches to overcome these weaknesses; and

6. Users of tools that aid in secure programming must know how to use those tools, and understand their limitations.

Summary of the Recommendations

Several themes emerged from the recommendations of the individual roadmaps. The ten recommendations below provide a starting point for stakeholders across constituent groups to begin to transform education in secure software.

1. Increase the number of faculty who understand the importance of secure programming principles, and will require students to practice them.

2. Provide faculty support for the inclusion of security content in existing courses through clinics, labs, and other curricular resources.

3. Establish professional development opportunities for college faculty, non-computer science professionals, and K-12 educators to heighten their awareness and understanding of secure programming principles.

4. Integrate computer security content into existing technical (e.g. programming) and non-technical (e.g. English) courses to reach students across a variety of disciplines.

5. Require at least one computer security course for all college students:

    a. For CS students focus on technical topics such as how to apply the principles of secure design to a variety of applications.

    b. For non-CS students focus on raising awareness of basic ideas of computer security.

6. Encourage partnerships and collaborative curriculum development that leverages industry and government needs, resources, and tools.

7. Promote collaborative problem solving and solution sharing across organizational (e.g. corporate) boundaries.

8. Use innovative teaching methods to strengthen the foundation of computer security knowledge across a variety of student constituencies.

9. Develop metrics to assess progress toward meeting the educational goals specified in the roadmaps presented in this document.

10. Highlight the role that computer security professionals should play in key business decision-making processes.

Conclusion

A holistic view of secure software education suggests that programmers and non-programmers alike must be educated in the core principles and practices of secure software design. A paradigmatic shift that adjusts the current emphasis from "students as customers" to "society as customers" will support holistic and comprehensive curricular reform. In order to achieve this level of transformation in the underlying educational system, SESS participants identified several structural enablers that must be considered. Although not explored in detailed during this event, we suggest that topics such as: the cultural shift among faculty and industry stakeholders that supports the development of a holistic view of software security; the identification of measurable objectives and corresponding measurement methods; the development of national licensure programs; and the alignment of expectations for university education and realistic constraints in the system; should be the focus of follow-up efforts.

Summit on Education in Secure Software

"We ... [have a] desperate shortage of people who can design secure systems, write safe computer code, and create the ever more sophisticated tools needed to prevent, detect, mitigate and reconstitute from damage due to system failures and malicious acts."[1]

The term "secure programming" is a misnomer. By rights, it should refer to programming designed to satisfy a specified security policy—the definition of "secure", after all, is "satisfying a security policy."[2] But the term is used more broadly to refer to programming designed to prevent problems that *might* cause security breaches. This style of programming deals with common programming errors (such as failing to check that the size of an input is no greater than the size of where it is to be stored), and should more properly be called "robust programming."[3]

Taking this more comprehensive view of secure programming, the Summit on Education in Secure Software (SESS) examined the questions of what should be taught in our nation's educational programs about secure programming to various constituencies of students, and what would be required to do so. The constituencies of students include computer science students, programmers, and non-technical students at the levels of post-secondary education, and secondary education. This report provides the background for the workshop, the workshop's goals, its organization, and key findings.

Background

President Obama has stated that recruiting and retaining cybersecurity professionals is a national security priority.[4] Almost every facet of society relies upon computer systems and infrastructure; indeed, that infrastructure provides critical support for the global economy, civil infrastructure such as power grids and other mechanisms for distributing utilities, and public safety. This led Mr. Obama to assert that cybersecurity risks pose some of the most serious economic and national security challenges of the 21st century.[5]

In order to meet these challenges, the software that controls these systems and infrastructure must be reliable, robust, and able to satisfy the requirements that are placed upon it. To this end, all people involved in the development and deployment of these systems and infrastructure, from the policymakers who determine what requirements the systems must meet to the businesspeople who provide the support needed to create the systems to the architects of the systems to their implementers, operators, and support personnel, must understand something about what constitutes software that will control these systems. The notion of "trustworthy computing" embodies many more facets of computing than software implementation, but poorly implemented software undermines other aspects of trustworthy computing.

---

[1] K. Evans and F. Reeder. *A Human Capital Crisis in Cybersecurity*. Center for Strategic and International Studies. Washington, DC (2010).

[2] See, for example, M. Bishop, *Computer Security: Art and Science*, Addison-Wesley Professional. Boston, MA (2003) p. 95.

[3] Throughout the rest of this document, we use "secure" to conform to the more common usage. We are, however, speaking of teaching robust coding to anticipate and avoid problems (which we may not yet know about) brought on by fragile (non-robust) programming.

[4] 2009. Cyberspace Policy Review: Assuring a Trusted & Resilient Information & Communication Infrastructure: http://www.whitehouse.gov/assets/documents/Cyberspace_Policy_Review_final.pdf

[5] Ibid

Thus, principles of secure programming must be integrated into a curriculum designed to meet the cyber security challenges of the future.

Because of the breadth of people who will affect, or be affected by, software, understanding the difference between software that is sufficiently robust to avoid common programming errors and software that is not robust is critical to a variety of disciplines, both technical and non-technical. The most appropriate method for teaching this material, and more importantly what resources are necessary to teach it, has not been well explored.

Summit on Education in Secure Software

In response to this challenge, the National Science Foundation Directorates of Computer and Information Science and Engineering (CISE) and Education and Human Resources (EHR) jointly sponsored the Summit on Education in Secure Software. The focus of the summit was on how best to educate both students and current professionals on secure programming concepts and practices, and to provide roadmaps indicating both the resources required and the problems that had to be overcome. Organized by The George Washington University and the University of California at Davis, the summit began with a teleconference on September 7, 2010 and continued with a two-day event held in Washington, DC on October 18 and 19, 2010. The SESS overview document is included as Appendix A, the organizing questions are given in Appendix B, and the agenda is presented in Appendix C.

SESS participants included representatives from academia and professional organizations across the public and private sectors as well as policy makers and government representatives. This multi-disciplinary group brought the needed depth (to identify technical challenges) and breadth (to identify operational constraints and opportunities) to enable the summit to find ways to advance and improve the current state of education in secure software. Appendix D lists the participants in the SESS.

SESS Objectives

The high-level goal of the SESS was to develop a comprehensive agenda focused on the challenges of secure software education. This challenge arises within the context of student education and professional development and training, and is aggravated by curricular and faculty constraints. Additionally, the need to measure attainment in a meaningful, useful way complicates the problem.

The SESS had three objectives:

1. To have cyber security stakeholders from academia, government, industry, and certification and training institutions discuss the goals of teaching secure programming and the current state of that teaching;

2. To use that discussion as the basis of a collaborative effort to develop new approaches, and improve existing approaches, to improve the quality of that education and to enable it to reach a broader audience; and

3. To outline a comprehensive agenda for secure software education that includes methods, resources needed, and problems that are foreseen to arise.

To achieve these objectives, the summit combined activities designed to gather, synthesize, and disseminate insights that enhance the education of students in secure software. A key consideration was capturing what the implementation of those plans would require. We began by posing a set of organizing discussion questions (shown in Appendix A), and asking participants to discuss them using a social networking website before the preliminary teleconference and the actual summit meeting.

SESS Teleconference

Before the summit meeting in Washington DC, invitees participated in a teleconference. This began as a plenum session, and then the group split into three small discussion groups, one each focusing on academia, industry, and government and others. The goal was to develop a preliminary view of the requirements and challenges of teaching secure programming from the perspectives of each of the three groups.

The requirements and challenges are discussed in the next section. A common, key theme emerging from the discussions is that any successful approach to teaching secure programming will have to address concerns within and among the sectors.

Secure Programming Education: Requirements and Challenges

This section of the report provides summaries and detailed listings of the requirements and challenges (see Tables 1 and 2). The members of the groups identified some requirements and challenges critical, and in some cases multiple groups identified the same requirements and challenges. These are highlighted (bold, underlined) in each table. The requirements and challenges were used to seed the development of the roadmaps.

Summary of Requirements

Distinct themes emerged from the group discussions. The academic and industry groups emphasized the importance of educating students about why robust, secure code is critical to security, and (where appropriate) about how to write secure code. They also pointed out the need to be able to critique code, to identify non-robust practices and security vulnerabilities in both other people's and their own code. This includes knowing how to test code. Industry participants focused on peer review as an important element of analysis, and academic participants focused on teaching approaches and frameworks that would provide the tools to do the analysis.

Government participants presented the need to recognize the possible security problems of foreign code and the consequences that might result from the integration of code drawn from multiple sources.

All groups noted that secure programming must be considered within the context of the full system design and deployment process. The requirements discussion highlighted 6 critical points that students of secure programming should know:

1. Understanding security, especially during design, requires a holistic approach;
2. Understanding and being able to identify common and emerging attack vectors is a critical component of security;
3. Well-tested principles and frameworks of software development can inhibit attacks;
4. All frameworks have weaknesses and subtleties;
5. Part of secure programming is using strategic approaches to overcome these weaknesses; and
6. Users of tools that enhance secure programming must know how to use those tools, and understand their limitations.

Details of Requirements

Table 1 shows the requirements for secure programming based on the perspectives of each sector, and highlights the requirements that cross multiple sectors.

Identified Requirements for Secure Programming Education	Academic	Industry	Government
Importance of writing secure, robust programs	X	X	X
Security is not just about coding – it is the **entire design and deployment process.**	X	X	X
Understand that code is international, and that your code may interact with 'foreign' code in unanticipated ways.			X
Common attack vectors, and perpetual emergence of new vectors.	X		X
Know, and be committed to **using best practices**, at all times.		X	
Well-tested principles and frameworks for creating secure programs. Edge cases and how to **test** for them.	X	X	
The importance of peer review.		X	
Weaknesses and subtleties in the frameworks and approaches to overcoming them.	X		
The cost of finding bugs and security flaws at various stages of the design process and after deployment.		X	
How to use tools to **test their designs.**	X		
The **security vulnerabilities** of specific computer languages.		X	
Understand what exploits are being used, and how to defend against them.		X	

Table 1: Requirements for Secure Programming Education Identified by Sector

Summary of Challenges

The challenges to teaching secure programming for each of the groups proved quite different.

The academic participants focused on the challenges related to faculty. Developing faculty skills in security, especially secure programming, and understanding the significance of secure programming is particularly important. Once these skills are developed, the next challenge is to identify effective methods for those faculty members to maintain a current knowledge-base, and to ensure that they learn accurate, realistic and timely scenarios, and are able to communicate them to students.

Industry participants expounded the need to convey the criticality of data integrity and migration strategies. Like the academic participants, they felt that ensuring faculty taught, and students saw, accurate, realistic scenarios, was critical to learning secure programming.

Government (and other) participants highlighted the challenges of understanding the risks associated with highly interconnected systems. Among the complications were connections across sector boundaries, national boundaries, and cultural boundaries. A second, equally important, challenge is mitigating those risks. These challenges were consistent with the requirements that emerged from the government (and other) groups.

Details of Challenges

Table 2 shows the challenges for secure programming based on the perspectives of each sector, and highlights the challenges that cross multiple sectors.

Identified Challenges for Secure Programming Education	Academic	Industry	Government
Developing faculty skills in security, especially secure programming, and understanding the importance of secure programming.	X		X
Teaching materials that are suitable and continuously up-to-date.	X	X	X
Aligning concepts with student abilities.	X		
Finding room (and proper placement) in the curriculum for security topics.	X		
Organizing secure coding rules into a coherent framework.	X	X	
Licensing and NDA restrictions.	X	X	
Helping individuals understand the threats posed by the Internet and interconnected systems.	X		X
Conveying **accurate information about the threat environment; including accurate simulations of attack-defend incidents.**		X	
Designing systems with the concept of least privilege in mind.		X	
Assisting with sensible & informed risk management decisions in light of interconnected systems.			X
Researching and testing best practice tools and techniques.		X	
Ensuring that mitigation strategies are incorporated in the design process.		X	
Ensuring sufficient system tests and evaluations before deployment.		X	

Table 2: Challenges for Secure Programming Education Identified by Sector

The Roadmaps (and Potholes)

A goal of the summit was to develop "road maps" that describe ways to improve the state of education in secure programming. The road maps explain what the students should know, various methods by which they might be educated, and what resources will be necessary to achieve that level of education. The road maps also identify expected and possible challenges to meeting these goals—the "potholes".

The summit used small working group sessions and plenary discussions to achieve this end. The first day began with a plenary session with presentations from an academic (Matt Bishop, University of California at Davis), a non-government practitioner (Alan Paller, SANS), and a government practitioner (Dickie George, National Security Agency). At lunch, the participants heard about models of education from the medical and legal professions. Then the workshop separated into three groups corresponding to the ones in the teleconference, and discussed how to improve the state of education in secure programming. At dinner, deans from several schools addressed the topic of the role of higher education.

On the second day, a professor of education (Melissa Dark, Associate Dean of the Purdue University College of Technology) addressed the group, discussing how to design a curriculum for secure programming. The workshop then split into numerous subgroups, focusing on the intended audiences for the education: computer science students, non-computer science students, community college students, K-12 students, computer science professionals, and non-computer science professionals. The day concluded with an afternoon plenary in which plans for developing a comprehensive agenda were discussed.

Although presented as separate guides, one for each constituent audience, the road maps should be considered as different views of a unified educational program that spans kindergarten through professional development, and audiences ranging from software developers to those who focus on management, policy and use to ordinary computer users. Thus, the educational goals in the road maps should be considered the "core fundamental end points" to be reached through instruction guided by a secure programming curriculum.

Professor Dark eloquently argued that educational goals must consider both curriculum (*what* you teach) and instruction (*how* you teach). Curriculum identifies the desired results of the education, as well as suggesting specific evidence to determine whether those results are met, and planning how to teach the students to ensure they reach those results. According to Prof. Dark, the development of educational goals should take into account Bloom's Taxonomy of Learning, which posits three interrelated dimensions of learning (e.g. cognitive, affective, and psychomotor). This complex learning process requires interdisciplinary assessment tools. For example, the Forced Concept Inventory (FCI), a mechanism commonly used in disciplines such as English, can reveal declarative, conceptual, procedural, principle, and problem solving skills. As such, it allows us to "know what the students know" and can help us understand student needs in a holistic way. Using this approach to design and implement a secure programming education that effectively engages the minds, experiences, and education of novices and experts, can help identify student needs, and match them to appropriate teaching methods and educational venues.

Box 1. "Building a Securing Programming System"

Melissa Dark, Associate Dean, College of Technology, Purdue University

Educational goals must consider both curriculum (*what* you teach) and instruction (*how* you teach). Curriculum provides the opportunity to "identify desired results," which help "determine acceptable evidence," and "plan the learning experience." This approach to building a securing programming system, that more effectively separates the minds, experiences, and education of novices and experts, can aid in the identification and matching of student needs to appropriate teaching methods and educational venues.

The Road Maps

The remainder of this report contains the road maps developed by the workshop participants for each of the constituent groups. The road maps are the collective response to the first two organizing questions:

- What should be taught?
- What resources are required to properly teach these concepts?

They are presented in the following order:

- Computer science students;
- Non-computer science students;
- Community college students;
- K-12 students;
- Computer science professionals; and
- Non-computer science professionals.

All activities, whether related to faculty development or curricular additions, will require money. The report does not specify how much money is required because this will depend upon the precise plans developed, the institutions and people involved in the execution of the plans, and a host of other factors. So, readers should take this as a given. The roadmaps discuss non-financial resources that meeting the educational goals will require. The potholes are the challenges associated with teaching the concepts and acquiring the necessary resources.

Each roadmap begins by defining the constituent group. It then presents the educational goals, required resources and associated potholes. Each roadmap ends with a recommendation for action. Individually, the roadmaps provide specific guides for each constituent group. Collectively, they provide a series of recommended steps to develop a system of secure software education that addresses both overlapping and divergent needs of the constituent groups.

The Roadmap: Computer Science Students

Who are they?

Summit participants defined computer science students as all students majoring in computer science or computer engineering.

The Roadmap

All computer science majors require students to know how to program. The ideas developed by the participants of this group build upon that observation.

All students should understand the very basics of security, for example being able to discuss the meaning of confidentiality, integrity, availability, privacy, separation of duties, layering of security mechanisms (also called "defense in depth", and the idea of least privilege, even if their primary focus is not systems or software. A general-purpose computer security course is an appropriate way to cover this material. Opinions differed on whether the course should be in the first two or last two undergraduate years.

All students should understand the role of security in design as well as implementation. This includes common patterns used to improve security. As was noted in the working group, this is really just good design. Part of this, is understanding what can happen if one does not apply the tools and techniques to identify and resolve potential problems. This also should be discussed, including not only technical issues such as corruption of data and resources but also the non-technical, and often intangible, consequences such as loss of money, customer trust, and damage to reputation.

The material the students learn should be grounded in problems and practices found in industry and government. In particular, the security issues raised should reflect the "real world" as well as teach principles. Usability is a part of this, because security problems often arise when systems and programs are so complex that users, and more importantly system administrators who install and configure these, make mistakes that open the system, site, and organization to attack.

The security lessons need to be integrated into courses that are not primarily security-related. For example, in software engineering courses, security could be framed as a necessary component of software quality. In this way, raising the security implications arising from decisions at each stage of the life cycle would make the students more aware of and sensitive to security considerations. In particular, the various aspects of input validation, the inadequacy of which is currently a critical problem, could be integrated into all courses that require programming.

Most computer science textbooks ignore problems of security, or mention them in passing; they do not explicitly include security considerations in design or implementation. Such considerations should at least be mentioned in design; examples of code should embody these principles in practice. One way to do this would be to work with textbook authors to ensure that the example programs and system designs include security considerations, including the application of secure programming techniques. An alternative is to develop supplements to widely used textbooks that expand upon the application of security principles to the subject matter in the textbook, in a way that integrates with the textbook. More general supplementary lab books could achieve the same effect, but the instructor would have to integrate the material into the course curriculum rather than treating it as simply a part of the text.

Students need to be able to assess existing programs and systems, and be able to work with other people's code. For example, most academic programs are relatively small, and students write them either alone or in small groups. But non-academic projects are typically large and complex, and teams of

programmers work together to develop them. This means that programmers spend much of their time reading and modifying other people's code. Because of the security implications of making such changes and integrating back into the code base, students should know how to analyze existing code as well as develop secure code.

To support their learning how to assess programs, students should have resources available to assess their own programs. Tools and other technologies will help meet this need. Source code review tools and testing tools will cover the back end, but at the front, the use of languages and integrated development environments (IDEs) that restrict the ability of the programmer/student to create non-secure programming constructs is also helpful. Care must be used, though, because requiring students to use a programming language designed for security in all classes will severely impact their ability to use the languages that are common in industry. The problem is not merely that they will be inexperienced in those languages. More importantly, they will not know how to write secure code in those languages because the language and IDEs they use will not require them to apply the principles of secure programming on their own, and thus inhibit their familiarity with the practice of those principles.

Perhaps more valuable than tools is human assessment--someone who can review programs and point out examples of (potential) problems, so the student can then learn how to do so themselves. From such interactions, the student will learn how to do assessments—the practice rather than the theory (which can be taught in regular classes). The idea of a secure programming clinic, which would function much as a writing clinic and writing support systems do in law schools and other departments, was discussed, but concerns were raised about its scalability.

One way to force this upon students is to give students (or teams of students) an assignment, and later in the term require them to extend the programs. However, for the extension, each student (or team) is given another student's (team's) program, or a poorly written, uncommented program from a standard repository. This very quickly teaches students the need to write good, clear code—and as a side benefit, it also encourages good documentation.

Concerns focused on teaching secure programming. Participants noted that graduate students would be needed, both to work with students in doing assessments and to grade assignments (in their roles as teaching assistants). Unfortunately, the participants felt that few graduate students have the skills and experience in secure programming to do this. Worse, many faculty either do not have the skills, or do not believe that students need to continuously use secure programming techniques (because they are ancillary to the material in that particular course), or both.

Participants agreed that a major problem is to raise awareness of the need to know something about security in general and secure programming in particular. One suggestion was that those who are trying to hire graduating students (and interns) should put in their job descriptions a statement of what the job requires with respect to security, such as a basic knowledge of it or the ability to apply the principles of secure programming when writing and/or analyzing code. This would be something industry and government could do to make faculty in particular and academia in general believe that security, and knowing how to write secure programs, is important.

Summary

Educational Goals

- Students should understand the basics of security
- Students should understand the role of security in design as well as implementation.

- Students should know about security issues, problems, and methods currently used in industry and government, and how those expand upon basic security principles.

- Students should understand the basics of usability, especially how a failure to take into account the user when designing interfaces can create or exacerbate security problems.

- Students should be able to assess code written by others.

- Students should have experience working on large projects, and in particular gain experience in using a source code control system.

- Students should be able to identify and question assumptions that relate to security, especially when those assumptions affect coding style and program structure and implementation. This will teach them how to "think like an attacker."

Teaching Methods

- All students should take a general-purpose computer security course that covers the basic ideas and principles of security.

- Classes should teach good program design in the particular area of the class. This also includes the principles and practice of secure programming.

- Use "real world" examples to teach the importance of, and the need for, practicing secure programming techniques.

- Create a repository of fragile, uncommented, non-secure programs and modules for use in exercises teaching students how to analyze, and how to harden, existing code.

- Create a repository of "real world" examples to teach the importance of, and the need for, practicing secure programming techniques.

- Have students use a source code control system for all their programs.

- Have students use commercially available tools to find potential problems in programs.

- Require students to work on programs in teams.

- Have students critique programs and designs in textbooks and other repositories for potential security problems, including a failure to apply principles of secure programming.

- Use widely available lists of vulnerabilities and compromises to demonstrate basic principles of security.

- Provide both tools and assistance for students in assessing code. This will give them hands-on experience in doing assessments.

Resource Requirements

- Graduate students who know security and secure programming to help grade programs and help students learn to assess code. Currently, most graduate students lack this knowledge, so they would need to be taught this.

- Support for clinics and hands-on laboratories, especially people, space, and equipment.

- Tools and other material used in industry.

- A means to learn what currently are the problems that institutions are facing with respect to poor coding; what are the most troublesome vulnerabilities, what are the currently used practices, and so forth.

Potholes/Challenges

- Lack of students to support mentoring in assessments of programs; this includes grading programs in all classes using secure programming criteria.

- Limited space in the curriculum to add new courses or new content to existing courses.

- The lack of faculty who know, understand, and apply secure programming techniques, and will require students to use them in non-security related assignments.

- Motivating faculty to include security content to their curriculum.

- The slow pace of change in academia.

- Finding textbooks that include secure programming considerations, especially at the beginning programming level.

- Getting those who hire students to write the need for a basic knowledge of security into their job descriptions.

- Getting material support from industry and government, in the form of tools and information to guide the faculty in teaching students this material.

Recommendations

- Take steps to increase the number of faculty who know about and are willing to require the application of secure programming in their classes (especially non-security classes).

- Integrate computer security into existing (non-security) courses in minimally invasive ways. Work with publishers and others to develop supplements to textbooks that provide examples that exhibit secure programming practices. In the future, work to get such examples into the textbooks themselves.

- Encourage the use of industry tools and technologies in the classroom and for projects. Among these are code analysis tools and source code control systems, as well as collaborative systems.

- Provide support for faculty and students to assess student programs and projects for robustness, especially to teach students how to assess their own, and others', programs. This can be done through a clinic, classroom instruction, or some other medium.

- Students should take a class in the basic principles and ideas of security, including important security issues, problems, and secure coding practices, and how those relate to principles.

- Take steps to encourage collaboration among people in industry, government, and academia to ensure that faculty have current information about security threats, defenses, and tools. Faculty members should be familiar with the information that prospective employers will want future hires to have.

The Roadmap: Non-Computer Science Students

Who are they?

Summit participants defined non-computer science students as all students other than those who major in computer science. These individuals may be users, managers, or even future computer science professionals who did not major in computer science (e.g. engineering or business majors).

The Roadmap

Students who are not computer science majors often program. They may create web pages, or build programs for others. Even if they do not, increasing their awareness provides them the opportunity and the knowledge to become consumers of programs in their use of appliances, cars, and home computers and web servers. Knowing a little about secure programming will help them become better consumers, and thus increase their ability to advocate for better programming when needed.

A public health analogy conveys the importance of the topic and provides insight into one possible method of delivery. Computer security (including secure programming) should be treated as a public safety issue, analogous to a public health issue. Using this approach, universities could require all incoming students to take a course orienting them to computer security. As an example, universities could include computer security in their orientation programs to heighten awareness and then have a general education course to introduce specific language, vulnerabilities, and the consequences of common attacks.

Like computer science majors, these students should understand the very basics of security, including basic vocabulary terms such as malware, antivirus software, and authentication and encryption; common attacks and their consequences; a basic understanding of privacy issues; and common fallacies in computer security. Unlike the courses designed for computer science majors, a general computer security literacy course would use non-technical means to convey the ideas underlying security and the threats to their personal identities and information. One of the primary ways we create culture is the use of semiotics and literature. The course should build on those materials to tie them into contemporary society and problems, to provide the realism that helps students assimilate material. Among the books suggested were *Neuromancer* (by William Gibson), *Shockwave Rider* (by John Brunner), *Snow Crash* (by Neal Stephenson), and the non-fiction *The Cuckoo's Egg* (by Cliff Stoll). Two issues such a course raises are where in the college curriculum it should go, and how to enable students to fit it into their already full schedules. Having the course satisfy a General Education requirement would solve or ameliorate this problem.

The ideas of security span many areas of our daily lives, as well as many different academic disciplines. Planting the seeds of security knowledge in many different venues will demonstrate the ubiquity of the basic principles and concepts, and will also help those ideas become part of their thought processes and daily lives. One approach is a security literacy or security culture course. Another is to provide self-contained learning modules suitable for use in computer literacy (or other) classes that non-majors take. For example, these modules might be associated with widely used programs like Microsoft Word or Adobe Acrobat Reader, because most students will use these programs. The intent is to make these modules minimally invasive and maximally applicable to the students' lives.

One way to make all students aware of the issues is through the use of simple demonstrations. For example, an instructor might launch a phishing attack on the students in the class, and when the students follow the supplied link, the web site contains information about the dangers of clicking on links without checking the underlying URL. The public health approach suggests that an extension to this idea might be more effective. When a case of meningitis or other such contagious disease occurs on campus, the university issues alerts to the campus community to notify members of the threat, potential consequences, and protective measures. The university could do the same for threats to computer security, the goal being to enable members of that community to protect themselves and slow or stop the spread of the attacks. Another benefit of such a system would be to raise the campus community's, and ultimately the public's, awareness about computer security.

In non-major classes where programming is discussed, show the students bad code (one suggestion was to have a "Bug of the Week"). Web programming, where appropriate, would be a fruitful source of these errors. As for student programs, all felt that students should lose points for non-secure programming. The comments about grading made in the road map for computer science majors apply here.

Participants noted the difficulty of changing behavior based on negatives (such as using "scare tactics" and horror stories). They felt a Socratic approach that uses questions and discussion to engage might be more effective. For example, rather than stating "Don't open attachments from unknown email senders," a more effective approach is to ask, "Would you pick up a bottle of aspirin that was laying in the street, and open it and swallow aspirin from it, because it's free?" and making the analogy to the dangers of doing the same thing with an email attachment, thumb drive, or other similar situation.

Most introductory computer science textbooks ignore problems of security, or mention them in passing; they do not explicitly include security considerations, and in fact many examples fail to show the checking of inputs, or other basic security precautions. Such considerations should at least be mentioned in design; examples of code should embody these principles in practice. One way to do this would be to work with introductory textbook authors to ensure that the example programs reflect the application of secure programming techniques. An alternative is to develop supplements to widely used textbooks that expand upon the application of security principles to the subject matter in the textbook, in a way that integrates with the textbook, or to provide access to a repository of exercises for the instructor to use in class or as homework. More general workbooks could achieve the same effect, but the instructor would have to integrate the material into the course curriculum rather than treating it as simply a part of the text.

An additional advantage to such modules and supplementary material is that faculty, especially non-computer science faculty, need to be educated about security. Computer literacy classes would be an appropriate place to teach this material, and those teachers will be computer literate. They may know little about security, however, and so would need to be educated in that topic. If the ideas underlying security were to be taught elsewhere, for example in a class on science fiction (which might include cyberpunk novels), the instructors would need additional supporting material as well as training. In this case, though, the training would need to be non-technical, much as teaching non- STEM (science, technology, engineering and mathematics) students security requires non-technical approaches.

One other consideration arises in non-computer science classes that teach programming. Students in these classes need to understand the principles of secure programming much as computer science majors do, although non-majors need only know the application to the particular language and environment in which they program. See the computer science major roadmap for a description of the ways suggested to improve students' programming.

Faculty teaching non-computer science classes are likely to resist changes such as the ones above, as will faculty teaching computer science classes. Some participants had experiences running workshops

for faculty. Some faculty members who attended multiple workshops seemed to buy into the ideas of, and the importance of, security and good programming, and became fully engaged in teaching students the material. The problem is getting the faculty to come to these workshops, and there was considerable discussion about what incentives (such as release time to work on class materials or research, or extra funding or salary) would encourage faculty attendance at the workshops and encourage them to teach the material.

Ultimately, the key to educating non-computer science majors is to identify creative methods to create understanding, or deepen their existing understanding, of computer security.

Summary

Educational Goals

- Students should understand the basics of security.
- Students should have a general awareness of the security problems and their consequences.
- Students should understand that most software is not written using secure programming techniques.
- Students should know basic measures they can take to protect themselves.

Teaching Methods

- Use semiotics and literature to teach ideas and underlying concepts.
- Launch demonstrative but harmless attacks against the students to teach them to spot these attacks. For example, send them phishing messages to take them to a site that explains the dangers of clicking on links without knowing the underlying URL and tells them how to find that.
- Use self-contained learning modules on computer and security literacy.
- Teach a "security culture" course for general students.
- Provide examples of poor programming, both from outside the class (such as web programs retrieved from web sites) and from textbooks.
- Use modules focused on programs that students generally use, such as Adobe Acrobat Reader and Microsoft Word.

Resource Requirements

- Time and space in the curriculum for a general-purpose computer security course. Having the course satisfy a General Education requirement would make it attractive to students who must fulfill those requirements anyway.
- Support for hands-on laboratories, especially people, space, and equipment
- Support for faculty to integrate material relevant to computer security into their classes.

- Graduate students who know security and secure programming to help grade programs in introductory and non-computer science classes. Currently, most graduate students lack this knowledge, so they would need to be taught this.

- Students doing *any* type of programming should have access to tools that check for non-secure programming and to assistance in using them and understanding the results.

Potholes/Challenges

- Lack of students to support mentoring in assessments of programs. This includes grading programs in all classes using secure programming criteria.

- Limited space in the curriculum to add a new course, or to integrate new material into existing courses.

- Limited space in a student's schedule.

- The lack of faculty who know, understand, and apply security knowledge in general and secure programming techniques in particular.

- How to incentivize faculty to include security information in their courses.

- The slow pace of change in academia.

- Finding textbooks that include secure programming considerations; especially at the beginning programming level.

Recommendations

- Students should take a class in the basic principles and ideas of security, to make them aware of the importance of secure programming and the effects of them or others not applying it when they program.

- Increase the number of faculty who know about and are willing to teach the ideas and concepts of security, and relate them to literature and other non-technical aspects of culture and life.

- Provide faculty who teach programming in non-computer science disciplines with information and support for introducing their students to secure programming (either through classroom teaching or some other way).

- Integrate computer security into existing courses in minimally invasive way, work with publishers and others to develop supplements to textbooks that provide examples that exhibit secure programming practices. In the future, work to get such examples into the textbooks themselves.

The Roadmap: Community College Students

Who are they?

Summit participants defined community college students as all students attending a two-year institution of higher education. Students in this category may be working to earn an associate's degree, a specialized certificate, or toward the completion of the first two years of college courses in preparation for transfer into a four-year institution. The discussion focused on students working toward a two-year associate's degree.

The Roadmap

Community college students are either planning to matriculate into a four-year college or university, or are working towards a terminal associate's degree. Because the roadmaps for computer science majors and non-computer science majors cover the first case, we focus on the second.

Like non-computer science majors, community college students need to understand basic security ideas such as threats and the consequences of ignoring them, principles, and basic computer security vocabulary such as malware, antivirus software, authentication, and encryption. Participants felt that the community college should provide laboratory exercises to enable students to learn by doing and seeing. Students who use programming in any classes should also know how to write programs that do not have security (or other) faults, and should be able to find and fix faults without adding new ones. All felt these students should be able to address common security vulnerabilities. The college should also aid students in developing their analytical and communication skills.

Participants felt that laboratory work was critical to educating students in computer security at all levels, as was teaching the current threats and problems. As an example, faculty might use the most common coding errors as a basis for teaching good programming practice. This would require expressing those errors in the language being taught, because students seeing the effects of those errors will help them understand the dangers of using those practices, and the benefits of avoiding them. An RSS (Really Simple Syndication) feed of current events related to computer security, such as new attacks and defenses, the effects of new attacks, legislation, and so forth would be a valuable source of information about the current security landscape. It would also help excite (or at least sensitize) faculty about computer security issues, and show them the ubiquity of these problems in many disciplines, especially non-technical ones.

This raises problems of educating students and faculty about secure programming. The methods suggested in the roadmaps for computer science majors and non-computer science majors would be useful here. To help disseminate knowledge about secure programming, a clinic would provide support external to classes. Scaling the clinic would be a problem, but perhaps it could be run as a virtual clinic. This would enable it to serve many community colleges simultaneously, and might be a good way to involve government and industry practitioners as mentors because they would not need to travel to the community colleges to assist students.

Grading programs is time-consuming. Examining programs for non-secure programming practices would increase the burden substantially. Thus, participants suggested developing automated grading tools for programs. Associated with these tools would be modules taught by phenomenal teachers. When the tool detected a non-secure practice, it would report the problem to the student and include a link to the appropriate module, which would discuss the (general class of) errors and teach how to avoid them in the future. This may require access to clinicians to provide additional information to ensure the student understands the problem and the solution. Indeed, access to these tools would be critical for the success of a secure programming clinic.

The problem with this type of intensive mentor-student interaction is the lack of mentors who can teach secure programming. Ways to ameliorate this problem were discussed. One was to provide online delivery of the material both with remote mentoring (as mentioned above) and recorded lectures so that students can study a module, listen to the lecture, and replay the lecture as needed. Another approach focused on recruiting clinicians. Graduate students from nearby universities might be a fruitful source of clinicians. One particularly interesting idea arose from the observation that a national talent search for potential clinicians would get people interested and excited in learning and teaching this material. A benefit for the clinicians is that secure programming is a very marketable skill, and the clinicians will have demonstrated their adeptness in it. Thus, there is a tangible benefit to the clinicians beyond the less tangible benefits of practicing good communication and honing their secure programming skills by analyzing programs that others have written. In any case, the clinicians must know secure programming, and be able to work with students to teach them the material.

Engaging the faculty to enforce good programming standards is another issue. Faculty workshops might get faculty enthusiastic, or at lease convinced of the need, to teach security in general and (where appropriate) secure programming specifically. Participants felt that faculty would not benefit from a course unless they wanted to take it. Encourage attendance at a workshop should involve incentives of some kind, for example giving those faculty access to the RSS feed mentioned above. All felt it imperative that the community college reward faculty who teach these ideas and principles.

Given the complexity of programming, and then adding secure programming principles, participants felt a "spiral" approach would be most effective for those students learning to program. The fear is that if an instructor begins with a long list of principles and practices to be aware of, the students will simply walk away from the class or from programming. More effective is to teach a bit of programming and then the principles and practices of secure programming that are relevant; then more programming, and the relevant practices and principles of secure programming, and so forth. Textbooks, or supplementary material, that taught secure programming would be an invaluable asset. The roadmap for computer science majors discusses this in more detail.

Finally, the participants noted that if statistics are gathered on programming errors, and associated data (such as who makes these errors, when, and how), then the knowledge and insights gleaned from them could be used to improve both teaching and learning.

Summary

Educational Goals

- Students, even those not learning to program, must understand current threats to security, and a basic computer security vocabulary. The threats must be continuously updated.
- Students who are learning programming should be able to write code that avoids faults.
- Students who are learning programming should be able to find and fix faults in existing programs without introducing new ones.

Teaching Methods

- Use well-known and widely accepted lists of common programming errors (tailored to the computer language being taught) to provide examples of poor programming practices, as a basis for discussing the consequences of those practices, and show how to fix those problems.

- Student mentoring in secure programming, such as provided by a secure programming clinic, will prove helpful. Open research questions include how to develop clinics that scale to very large populations, and how to deliver the mentoring as effectively as possible.

- Use automated grading, and provide links to modules and (possibly) human help to the students so they can learn from their errors.

- Provide students with current events, threats, defenses, and practices of computer security in general and secure programming in particular to help them understand the importance of these practices, and the consequences of not following them.

Resource Requirements

- Automated tools to help students, faculty, and mentors teach and practice security and secure programming.

- The resources to develop and deliver instruction in security and secure programming online, including providing the ability for students to play and replay modules.

- The resources to recruit mentors for teaching students secure programming. The mentors may be faculty, graduate students, people from industry, or others.

- A strong set of partnerships with corporations and universities to support the teaching of security would be very beneficial not only to the students but also to the faculty and the community college itself.

- Faculty recruitment and training programs should support security and the development of secure programming knowledge.

- Textbooks, or supplementary material, that teaches security and secure programming. This is particularly critical for community colleges because their instruction is very textbook-driven, and most textbooks do not discuss security or secure programing principles and practices.

Potholes/Challenges

- Students study several different programming languages, and the secure programming content must be tailored for each language. Also, mentors must be trained in each language.

- Finding people who are able and willing to mentor, and who can work with students without intimidating or overwhelming them, may be difficult and require people to be trained.

- Faculty may have little or no incentive to teach this material. Rewarding those who do may reduce this problem, although the specific incentives will vary from institution to institution.

- Corporate and university partnerships are difficult to sustain.

- Community college programs are very textbook-driven. Few textbooks cover security and secure programming at a very basic level. Supplementary material will be required.

- Cultural changes in the community college environment, especially from faculty and administrators, will make the introduction of this material possible.

Recommendations

- All community college students should be exposed to computer security basics in order to become knowledgeable consumers of technology.

- Students who take programming, either through a programming course or who program as part of another course, should learn about common programming errors, their consequences, and how to avoid them.

- Community colleges should develop and strengthen partnerships with 4-year institutions and industry to maintain competency and improve the existing curriculum and faculty skills.

The Roadmap: K-12 Students[6]

Who are they?

Summit participants defined K-12 students as all students in the pre-college years, however specific emphasis was placed on students in secondary (grades 9-12) school.

The Roadmap

K-12 students need to understand the impact and consequences of security failures. In addition to making them better consumers of technology, this will increase their understanding of how launching computer attacks against systems and organizations can harm society and individuals. Developing a security mindset will also help them protect their own (and others') systems. Educational efforts should focus on the conceptual more than the technology, because during the average K-12 student's lifetime, the technology and its attendant procedures will change greatly. They also should continue to develop students' analytical skills, which will prove valuable not only in computer security but also in all fields.

Currently, the computer science Advanced Placement (AP) program is undergoing significant revision. Integrating principles and concepts of computer security and secure programming into this AP effort will cause those principles and concepts to be integrated into computer science AP courses.

Having computer security play a role in national programs such as the National Math and Science Initiative or the Race to the Top would also encourage schools to add material on computer security to their curriculum. One problem is to find space in the K-12 curriculum, which is already quite full. Schools that wish to teach computer security literacy must change courses to include this material (and drop other material that would otherwise be taught). How to do this is probably best left up to the schools, their teachers and administrations, and the national accrediting bodies.

A way to make the learning more palatable to many students is to embody it in a game of some form. The games could be simple ones, or massively multiplayer online games (MMOG) involving playing with others throughout the nation. Although such games could be rolled out nationally, it is reasonable to assume they will spread throughout the world, meaning that students in other nations would also benefit from such a game. The problem would be to make it enticing *as a game* as well as

[6] Although not emphasized during the SESS, some within the IA/CS professional community have discussed preparing all students from preschool on in basic security ethics and concepts using the "911" and "Safety Seats" campaigns as models.

educational. Involving people who have developed MMOGs or MMORPGs (massively multiplayer online role playing games) would be necessary, because they have experience in developing such games both from the technical point of view and from the marketing (psychological) point of view.

K-12 school students often use terms of opprobrium such as "nerds" and "geeks" when describing people studying computer science in general and computer security in particular. This inhibits many students from studying these fields. Students shape both their perceptions of themselves and of particular career fields and fields of study early. So, schools should address this problem directly by cultivating an image of these fields as challenging and significant. Emphasizing that they involve people and understanding the world around us may lessen the widespread belief that people who study and work in these fields are all technocrats focused first and last on the technology, to the exclusion of all else.

Another way to increase acceptance of studying these areas is to involve youth groups in this type of education. For example, the Girl Scouts, Boy Scouts, the Boys' and Girls' Clubs, and other community groups could informally educate their members through informational packets, discussion, and other activities appropriate to the purpose of the groups.

Meeting the above goals will require many teachers. Consider secondary schools. Assuming 30,000 secondary schools in the nation with 3 computer science teachers per school, approximately 90,000 teachers would require professional development training in the field of computer security. Many secondary schools lack any teachers trained in computer science, as do most primary schools; thus, this number is not simply low, but it also understates the amount of training needed, because these teachers will also require education in basic computer science. Perhaps augmenting "technology for teachers" courses offered by or through school districts would meet this need. One drawback to training is that the teachers learn a very marketable skill. They will become much more attractive to industry and government, which may try to recruit them. Thus, schools should anticipate some loss of teachers who participate in the training program.

Partnering with companies working in the computer security field, or that have extensive computer security programs, might enable schools to take advantage of the professional expertise of employees at the company to supplement instruction on a very limited basis. Certainly students would benefit from hearing about the daily activities and adventures of practitioners. Similarly, partnerships with universities and other post-secondary academic institutions may provide teaching experience for computer science majors (and, possibly, entice some of them to consider becoming K-12 teachers).

Expanding existing public-private partnerships with schools, and beginning new ones, will provide some resources to schools.

Finally, students who are studying programming should learn secure programming concepts, principles, and practices in these courses. The roadmap for computer science majors discusses in detail various techniques to achieve such inclusion, and the interested reader should see that section.

Summary

Educational Goals

- Students need to develop security mindset, and understand the impact and consequences of failures of security

- Students who are learning to program should be able to write secure programs, and identify non-secure practices in programs.

- Because students shape both the perception of themselves and that of particular career fields early, K-12 schools should emphasize that computer security is not just a technical subject. It involves people and understanding the world around us.

Teaching Methods

- K-12 education in computer security should emphasize the conceptual rather than the current technology. While both are important, the technology will change during the students' lives, and understanding the conceptual will mean their knowledge of security can evolve along with the technology.

- Use community groups such as the Boys' and Girls' Clubs, the Girl Scouts, the Boy Scouts, and others to educate their members.

- Integrate information about computer security into computer science AP classes.

- Develop computer games, especially MMOGs and MMORPGs, that will help students learn about computer security while they play with others across the nation or the workload.

- Focus on developing students' analytical skills.

Resource Requirements

- Resources to train teachers and the ability to attend such training are necessary.

- Material about computer security for K-12 education must be developed.

Potholes/Challenges

- The number of teachers who will need to be trained is very large; meeting this educational need will require resources on a very large scale.

- The current K-12 curriculum must either provide space for, or be modified to include, computer security material at an age-appropriate level.

Recommendations

- Support national initiatives such as the National Initiative for Cybersecurity Education (NICE) to raise the awareness K-12 teacher awareness of computer security issues, principles and potential career paths for their students.

- Work with community groups to develop ways to teach ideas and concepts of computer security as they apply to the goals of those groups, and in such a way that the teaching does not interfere with the groups' actions and mission.

- Develop and enhance existing public-private partnerships to provide resources for computer security education.

- Encourage the development of age-appropriate materials for computer security education in K-12 schools. This includes the development of computer games that will help students learn about computer security. Some games should be individual-player games; others should be MMOGs or

MMORPGs, because K-12 students often prefer playing with others online, and can learn as a group.

The Roadmap: Computer Science Professionals

Who are they?

Computer science (CS) professionals are practitioners, industry leaders, government officials, managers and executives, and persons who set policy—they are programmers and non-programmers alike. CS professionals typically have attained a degree from a computer science program within an accredited university setting at the undergraduate, graduate, and/or postgraduate levels. If not, they have acquired the equivalent knowledge in other ways.

The Roadmap

Computer science professionals, regardless of their specific job functions, deal with computers and software in one way or another. Thus, security is a critical part of the fabric with which they deal, and they must understand the key concepts and ideas such as confidentiality, integrity, availability, authentication, defense in depth, least privilege, and separation of duties. Those who work with software and hardware must be able to apply those concepts, and also know about the security development life cycle (SDLC) and secure design patterns. This knowledge begins at the university. Students should be able to apply the concepts they learn to specific industry and government problems--in essence, "priming the pump." Hence, the recommendations in the Roadmap for Computer Science Majors all apply here.

One problem in the introduction of security to organizations (commercial, governmental, and academic) is the separation between the computer science and non-computer science professionals. All too often, the latter handle procurement and other functions in which security is an important consideration. Computer security professionals can contribute to the requirements of such procurements in order to ensure that what is procured is suitable with resect to computer security needs. Indeed, approaching the problem of computer security holistically, so everyone—computer science and non-computer science professionals—deals with issues of computer security as a group would educate people about the needs of the organization and the barriers to fulfilling those needs.

Hiring is an important part of this process for two reasons. First, many academic institutions tailor their programs based on what the industries they interact with consider important. Without compromising the education in fundamentals of computer science, teachers can emphasize certain aspects of their application, and draw examples from problems that industry deems important. So, if industry job advertisements state the specific security knowledge that the successful candidate must have, then students would be encouraged to take one or more classes in computer security, and instructors could add appropriate material to other classes to better prepare students for jobs. Second, if organizations hire computer science professions knowledgeable in computer security, then they can contribute to improving the state of the organization's protections. In particular, organizations should be encouraged to hire and compensate candidates with experience and expertise that emphasizes a holistic view of security.

This will require a shift in the hiring culture, in no small part because people who know about security have many job opportunities available, and may be more difficult to hire. In general, organizations and institutions resist change, especially if what they are currently doing seems to work. Thus, changing the organization requires convincing members of problems that the current organizational processes will not handle, and that the consequences of failing to handle them will damage the organization. The best way to do this is to have people within the organization who understand the

problems and can relate basic security concepts and principles to key business metrics and practices. Then they can evaluate the effects of the problems, and the costs of remediating them.

Professional development is necessary on a continuing basis, to keep computer science professionals apprised of new threats and methodologies in secure programming. Implementing these courses requires co-operation among many organizations both within a single sector and spread across many sectors. The depth is important because what an organization in a sector has learned can be passed on to other organizations, and by sharing experiences and ideas, organizations can develop effective countermeasures and means to implement secure programming methodologies more effectively. The breadth is important because problems and solutions may affect one sector before they affect other sectors. In addition, analyzing problems and solutions from other sectors sharpens those critical and analytical skills so essential to secure programming in particular and computer security in general.

Two barriers to change are often cited. Many institutions, especially commercial ones, look at problems in terms of finances; what is the effect of dealing with, or not dealing with, a problem on the finances? Security is a non-revenue producing aspect of computer science for most companies, and so security training and other security-related costs will impact the budget without providing tangible benefit (until, of course, a security problem arises). Thus organizations may be unwilling or unable to pay the price to improve their security-related process. Also, regulations that constrain an institution's ability to change also affect innovation. This is particularly important when regulations are not tailored for the specific sector. Different sectors, and often different organizations with different purposes, require different policies. As these policies and regulations are often determined in the political and legislative arena, those who develop them must also know much about computer security, in order to ensure the policies and regulations solve the right problems and can be administered effectively. This means government can be a driving force to change organizations' security practices—but government must tread carefully, lest ineffective or meaningless regulations add such a burden that organizations, in complying, must use the resources they would otherwise need to respond to security problems and threats effectively.

Licensing and certification is discussed in a separate section of this report, but some observations are pertinent here. Neither certification nor licensing is a panacea. Both have advantages and disadvantages, discussed elsewhere. For computer science professionals, any required licensing or certification must be pertinent to the work they are doing, must be based on a generally accepted course of study, examination(s), a practicum, or some combination of these. Participants felt that a professional group rather than commercial groups should create and administer any forms of licensing and certification that are deemed necessary.

Summary

Educational Goals

- All computer science professionals should understand key security concepts and ideas such as confidentiality, integrity, availability, authentication, defense in depth, least privilege, and separation of duties.

- All computer science professionals should understand how to identify and deal with problems of ethics that arise in computer security.

- Computer science professionals who deal with software or hardware should understand key secure programming concepts such as input validation, the security development life cycle (SDLC), and secure design patterns.

Teaching Methods

- Relate security basics/principles to key business practices and metrics.
- Use examples drawn from the computer science professional's environment or business/work sector.
- Use techniques that expand the computer science professional's critical and analytic thinking.
- Methods suggested in the roadmaps for computer science majors are useful here.

Resource Requirements

- Secure programming professional development efforts must integrate material from all sectors of technology and technology use. They should use current, real-world examples such as case studies drawn from a variety of sectors relevant to the particular students of each class.
- Professional development courses focusing on security principles and concepts, current threats and countermeasures, and computer security ethics must be developed for all sectors in which computer science professionals work.
- Professional development courses focusing on secure programming methodologies, current techniques, and current threats at the system (hardware, software) level must be developed for computer science professionals who deal with software or hardware.
- Industry and government hiring personnel should indicate what specific security needs each job entails in order to disseminate the importance of, and need for, security knowledge among those who are sought.
- If certification and/or licensing is required, then the resources to develop those programs must be provided, as well as the resources to support individuals undergoing those certification and/or licensing programs.
- Organizations must provide sufficient resources for computer science professionals working on projects within (or that affect) the United States, but who live in another country.
- Government assistance in creating and sustaining industry and government co-operation with academic institutions, such as partnerships, is critical to the success of those partnerships.

Potholes/Challenges

- Security costs money, and is a non-revenue generating part of software (and hardware) development. Thus the return on investment (ROI) is not visible until and unless security problems arise.
- Industry and government culture will resist change, hindering the adoption of security and secure programming technology and training.
- Currently there is a separation of duties between computer science and non-computer science (e.g. acquisition) professionals.

Recommendations

- Offer professional development courses on security principles and concepts, current threats and countermeasures, and computer security ethics for all computer science professionals.

- Provide professional development courses on secure programming methodologies, current techniques, and current threats at the system (hardware, software) level for computer science professionals who deal with software or hardware.

- Identify collaborative opportunities for computer science professionals to discuss security and ethics matters among their peers without compromising themselves or their organizations, so that they can benefit from one another's experiences and knowledge.

- Assist industries and universities to partner to teach courses and information relevant to the sector of the industry.

- Develop metrics to assess the strengths and weaknesses of applicants for jobs, so that those who are hired know what professional development would help them be more effective at their jobs.

- Build up the computer science major in universities as suggested in the roadmap for computer science majors.

The Roadmap: Non-Computer Science Professionals

Who are they?

Summit participants defined non-computer science professionals as members of the C-suite, "MBA-types," lawyers and contracting officials. These individuals were defined either by their contextual environment (i.e. working in business and industry) or their role in the organization. While the discussion focused heavily on acquisition professionals, participants did not distinguish between categories of non-computer science professionals for the road map.

<u>The Roadmap</u>

Non-computer science professionals are consumers of software and systems. Thus, their training and education focuses on understanding what computer security is, the role secure programming plays in it, how to assess security in systems and programs, and what the consequences of failure are. This includes being able to interpret information about computer security, make decisions that must take into account security issues, and knowing when to ask for help.

Computer security in general, and software vulnerabilities in particular, are the subject of much misinformation and misperceptions. The expression "Fear, Uncertainty, and Doubt" (FUD) summarizes the usual form—the information is designed to make the listeners fearful of unspecified threats, raising questions in their mind about the effectiveness of existing countermeasures. To counter this, non-computer science professionals need to understand enough to recognize inaccuracies and FUD. Even if they do not know what the correct information is, they must be able to recognize that they do not know it, and should get proper advice. This means that non-computer science professionals need to be well versed in the specific security-related responsibilities for their job.

Among those responsibilities is the ability to understand the return on investment (ROI) of security-related investments. Non-computer science professionals must be able to assess the impact of information assurance decisions on the bottom line. This skill must go beyond the short term, and often superficial, contracting decisions to include more in depth risk assessments. Knowing how to conduct risk assessments relevant to their jobs enhances this skill. Thus, they must be able to work with security professionals to estimate the cost of vulnerabilities and attacks accurately, and to communicate this information clearly and effectively to financial and strategic decision-makers.

Because these professionals do not have the same amount of time to study computer security and secure programming as do students in academic institutions, instruction must take a form that not only minimizes the time needed but also maximizes impact. Research is needed to find the most effective combination of content and delivery methods. Videos like public service announcements have proven successful and memorable in areas such as safety in the workplace, as have peer discussion groups and case studies. Cross-fertilization among a wide variety of jobs and employment sectors will help bring new and different perspectives to non-computer science professionals, and challenge them to exercise their creative, critical, and analytical thinking skills. Further, the knowledge will spread as professionals discuss security and secure programming. Thus, the professional development programs will extend their influence beyond the boundaries of those particular programs.

Non-computer science professionals will benefit from resources to help them evaluate and assess people, systems, and software. For example, if an organization such as Underwriters Laboratories existed for software or systems, non-computer science professionals would have a trustworthy source for information about those products. Similarly, some participants suggested that a noncommercial organization that accredits or licenses programmers would provide a similar service. Licensing and certification is discussed in a subsequent section, and was the subject of much disagreement among participants. Currently, much information on the security of software and systems exists on the web, but that information is often incomplete and of questionable accuracy. A trustworthy repository of information about these security problems would be of great value, especially if that information were vetted before being made available generally.

Because of the focus of non-computer science professionals on non-technical matters, security in programming may prove too granular for them to deal with effectively. Thus, the consequences of not following secure programming practices, especially if expressed in dollars and cents, or in terms of the operational capability of the organization, will bring home the need to allow the computer science professionals to develop practices, and follow best practices when those exist, that support security in software. Non-computer security professionals should be aware of these consequences and their effects.

One suggestion to force improvement to software is to remove the liability protections from software vendors. Participants realized this would drastically change the economic, political, and legislative landscape of software development, sales, and deployment, and so suggested starting with legislation in one particular area—cloud computing—to test this idea. Participants felt that insurers would either want to or be asked to insure providers against security breaches. In order to assess the rates they should charge, the insurance companies would ask questions about the software involved in cloud computing, and about the security of the systems involved. This could lay the basis for a scientific inquiry into improving software and system security.

Summary

Educational Goals

- Non-computer science professionals must learn to recognize what they do not know, and when to seek advice.

- Non-computer science professionals must be able to recognize obvious inaccuracies or exaggerations when discussing computer security.

- Non-computer science professionals must be able to conduct risk assessments in order to learn how computer security, and secure programming, affects their business, especially how the trade-offs impact their bottom line.

- Non-computer science professionals must be able to recognize situation in which they must consult a computer security expert for advice or assistance.

- Non-computer science professionals must be able to resolve conflicts between inconsistent expert opinions.

- Non-computer science professionals must know about threats and specific attacks likely to be used when they are targeted (for example, spearphishing).

Teaching Methods

- Case studies, similar to those used in business schools at universities, will provide both context and familiarity to non-computer science professionals.

- Public service announcements (such as videos about safety in the workplace) targeted at non-computer science professionals should be used.

- Peer groups, in which computer security problems and issues can be discussed freely, will provide non-computer science professionals with insights from other organizations, fields, and sectors that will broaden their knowledge and experience.

- Continuous learning is critical, so non-computer science professionals can enhance their knowledge through continuous adaptation.

Resource Requirements

- Liability legislation drives a need for non-computer science professionals to learn about computer security and secure programming.

- Secure coding standards that non-computer science professionals can understand; perhaps focusing on the consequences of failing to meet these standards will assist non-computer science professionals to understand why they must be applied.

- Some method of assessing the competence of software and system developers will aid non-computer science professionals in determining the qualifications of developers and consultants.

- An organization analogous to Underwriters Laboratory for software would provide a trusted source of information for making decisions about security properties of software and systems.

- A repository to share information about security vulnerabilities in software would support intelligent decision-making.

Potholes/Challenges

- Given emphasis on return on investment or speed of implementation, the low immediate cost of poor software may seem overwhelmingly attractive.

- Many organizations tend to denigrate security problems as human or programmer error, and believe they solve the problem when they reprimand the programmer.

- How does a non-computer science professional, or an organization, measure the impact or value of security?

- As costs of failures of security are difficult to assess, as is the strength of security measures such as secure programming, insurance companies will be reluctant to provide policies against successful attacks.

- The costs associated with professional development, and the higher costs associated with more stringent software procurement requirements, will deter many organizations.

Recommendations

- Provide training and education to strengthen non-computer science professionals' understanding of the costs and return on investment of security related investments.

- Provide ways for non-computer science professionals to enhance their ability to integrate security and other business knowledge; and to resolve inconsistent expert opinions.

- Teach non-computer science professionals to treat security-related issues similar to legal issues and consult expert advice on security-related matters.

- Provide non-computer science professionals with professional training and development for security and secure programming matters.

- Build up the undergraduate curriculum in universities as suggested in the roadmap for non-computer science majors.

The Role of Higher Education

The recommendations across all constituent groups require significant participation from institutions of higher education. Clearly, colleges and universities must drive the educational preparation of future security and non-security professionals alike. But the specific role of higher education in leading the cultural shift required to address the challenges and implement the recommendations outlined above is less clear.

During the summit, a panel of university deans provided insight on the role of higher education. Bill Chu, (*former*) Dean of the College of Computing and Informatics at the University of North Carolina, Charlotte, Dean Michael Feuer of the Graduate School of Education and Human Development at The George Washington University, and Dean Alec Yasinsac of the College of Information Science at the University of South Alabama, led the discussion. All felt that higher education has a significant role in developing, supporting and sustaining efforts to create and maintain secure programming education. This role must focus on student, faculty and curricular efforts.

Students

Students must be better prepared for the challenges that will meet them and must integrate educational policy and curriculum reform to achieve such goals. National scholarship programs should be developed in partnership with entities like the National Science Foundation to increase student involvement in the field.

Faculty

Faculty incentives, including promotion and tenure metrics, must be designed to encourage and support faculty activities necessary to keep security curriculum current. This includes recognizing the value of creating laboratory exercises, developing context-specific laboratory exercises, and keeping them current with observed trends and problems. Increasing the number of hybrid appointments, such as Professor Emeritus to include retired faculty members with the requisite expertise, and using specialists and technical faculty, will help to keep faculty skill-sets current. Deans and department chairs must anticipate limitations in faculty skills and structural barriers.

Curriculum

The curriculum should utilize interdisciplinary lessons, particularly from the fields of medicine and legal education, to advance a clinical model of secure software education. The curriculum must be kept up-to-date and interdisciplinary in approach. It is clear a more secure programming clinic should be integrated into the curriculum, which we should seek to standardize education across schools and departments for students.

Box 2.
The Deans' Perspective: What is the Role of Higher Education?

To attract secure programmers to the field, we must seek different platforms of engagement and conduct research that improves relationships across sectors. Universities, along with their government and industry collaborators, must develop a "new model" for feedback on program strength, the design of curricular elements and hand-on laboratory exercises, and the description of career paths. Such changes will not happen over night, so we must cultivate a culture of change, buy in, and commitment. Universities must lead the cultural shift necessary across all sectors to develop a security mindset among programmers and non-programmers alike.

Licensure and Certification

The role of licensure and certification in the secure software debate is unclear. Specifically, participants reflected two different points of view about these attributes. One group held that licensure and certification was inappropriate for a number of reasons, such as the lack of generally accepted standards of practice.

Another group believed that some form of external professional accreditation would help achieve and maintain a certain level of professional practice in secure software. Granting that, little agreement exists on the most appropriate mechanisms to achieve needed standards—or, indeed, what those standards should be. In an effort to move the debate forward, Professor Candice Hoke of the Cleveland State University School of Law provided a primer on licensure and certification (see Appendix E), and Associate Dean Dr. Yolanda Haywood of The George Washington University School of Medicine led a discussion of how professional standards and licensing have been handled in medicine.

Licensure

Professor Hoke's remarks focused on the critical distinctions between licensure and certification. Though often considered together, licensure and certification are distinct methods of professional legitimation. Licensure is the legal authority, granted by a governmental body, to practice an occupation. A license restricts the occupation to those persons who have demonstrated a minimum level of competency through some combination of education, practice and testing. The governmental licensing authority has sole discretion to determine requirements and grant the license. Licenses may be granted based on federal (e.g. FAA licensing of airline pilots) or state (e.g. state bar exams for lawyers) standards. The breadth of occupations requiring licenses to practice is notable and includes: acupuncturists, architects, bartenders, civil engineers, lawyers, massage therapists, mechanical engineers, medical doctors, nurses, pilots, realtors, and tattoo artists.

Although most licensure programs are state-based, they are often linked to national associations (as in the case of medicine) and grow from the desire of members of the profession to elevate professionalism and draw a clear distinction between trained professionals and knowledgeable members of the general public. This distinction is particularly important when issues of malpractice can lead to significant harm. In the case of secure software, the damage caused by malpractice in any one case (e.g. within a single company) can grow exponentially through virtual connections over a short period of time.

Licensing requirements can make a direct and significant impact on educational programs. Akin to the requirements of accreditation bodies like the Accreditation Board for Engineering and Technology (ABET), licensing authorities can and do influence individual course content, faculty selection, course sequencing, and field-based (e.g. laboratory) requirements. As Professor Hoke noted, bar exam requirements often influence the curricula of law schools. Dr. Haywood also pointed out that the licensing requirements in medicine motivate decisions on the content and structure of medical school curricula.

Certification

Certification grants the right to use a title that reflects specified qualifications (education, skills, and/or experience). Unlike licenses, however, governmental authorities, non-governmental entities, for-profit institutions, other organizations, or some combination of the above can offer certifications. The IT sector is replete with certification offerings that endorse specializations in some aspect of software security. In fact, the breadth of certification providers in the IT field has led some to question the value of certification.

Box 3. The Role of Licensure and Certification
Candice Hoke, Cleveland State University School of Law

"Licensure is often motivated by a desire to distinguish for the consuming public who can be trusted to perform competently when an asymmetry of relevant knowledge can lead to deceit and injury." As such, promoting public safety (as in the case with secure software) is a legitimate rationale for licensure.

Licensure and Certification Considerations

Options for a secure software licensing authority and governance structure include state-based, federal-based, and hybrid options. According to Professor Hoke, state-based structures are the least desirable for two reasons. First, state-based options often suffer from a "race to the bottom" syndrome. Second state board consortia can be cumbersome to establish and operate. Given the rapid pace of development of, and the large number of sub-fields in, the software security arena, this type of structure would likely not be an effective approach. Federally-based licensing could suffer from similar problems.

A federal authority based in the Department of Defense (DoD), for example, might be able to administer to those individuals who will work with DoD civilian and military systems, but the requirements for and processes used by civilian agencies are likely to be quite different. Again, given the software security environment, a large bureaucratic structure could create more problems than it solves.

SESS participants who supported licensing and/or certification preferred the hybrid model that leverages the strength of well-known and respected national professional organizations. Further, a professional society (like the American Medical Association for medical licensing) rather than commercial organizations should act as the licensing or certification authority.

It is important to recognize that compliance with licensing and/or certification requirements does not assure success. Accreditation decisions must be continuously monitored, evaluated, and refined. Dr. Haywood noted that even 100 years after the Flexner report[7], which spurred the professionalization of medical practice, debate on topics like the appropriate balance of classroom education and hands-on practice for medical students, continues.

Additional points to consider in the secure software licensure and/or certification debate include:

- **Program components.** What is the role of licensing and/or certification in educational programs? How much should continuing education requirements be factored into these standards? How can (should) global licensure and/or certification models/exams be used in the US? How will disciplinary actions be enforced? Will an ethics component be included?

- **Candidates.** Who should get licensed or certified, and when (e.g. during college education, after a certain number of years in practice)? Should faculty instructors be licensed or certified as qualified instructors? Will reciprocal licensure an/or certification be permitted? How will current professionals be phased into such a system?

[7] Flexner Report: http://www.carnegiefoundation.org/sites/default/files/elibrary/Carnegie_Flexner_Report.pdf

- **Fields and sub-fields.** What are the specific sub-fields to be included in such a program? Who will define the field and sub-fields? What are the priority sub-fields for program development? Given the rapid pace of change in the field, what process for identifying new sub-fields will be used?

In order to explore the role of licensing and certification, we recommend that a working group be formed to explore the role of licensure and certification in the field of secure programming, including an assessment of how the portfolio of existing commercial certifications might be integrated into a larger licensure and certification structure.

Conclusion

A holistic view of secure software education suggests that programmers and non-programmers alike must be educated in the core principles and practices of secure software design. A paradigm shift that adjusts the current emphasis from "students as customers" to "society as customers" will support holistic and comprehensive curricular reform.

Summary Recommendations

Several themes emerged from the recommendations of the individual roadmaps. The ten recommendations below provide a starting point for individual stakeholders across constituent groups to begin to transform education in secure software.

1. Increase the number of faculty who understand the importance of secure programming principles, and will require students to practice them.

2. Provide faculty support for the inclusion of security content through clinics, labs, and other curricular resources.

3. Establish professional development opportunities for college faculty, non-computer science professionals, and K-12 educators to heighten their awareness and understanding of secure programming principles.

4. Integrate computer security content into existing technical (e.g. programming) and non-technical (e.g. English) courses to reach students across disciplines.

5. Require at least one computer security course for all college students:

    a. For CS students focus on technical topics such as how to apply the principles of secure design to a variety of applications.

    b. For non-CS students focus on raising awareness of basic ideas of computer security.

6. Encourage partnerships and collaborative curriculum development that leverages industry and government needs, resources, and tools.

7. Promote collaborative problem solving and solution sharing across organizational (e.g. corporate) boundaries.

8. Use innovative teaching methods to strengthen the foundation of computer security knowledge across a variety of student constituencies.

9. Develop metrics to assess progress toward meeting the educational goals specified in the roadmaps presented in this document.

10. Highlight the role that computer security professionals should play in key business decision-making processes.

Structural Enablers

In order to achieve transformation in the underlying educational system, SESS participants identified several structural enablers. Although not explored in detailed during this event, these topics should be the focus of follow-up efforts. The questions included after each topic are posed as initial points of inquiry representing the beginning, not the end, of the discussion.

- **Cultural shift**. How can we create a cultural shift among faculty and industry stakeholders that supports the development of a holistic view of software security? How can government and industry funders be encouraged to use financial support to encourage the cultural shift? How can we incentivize faculty members to update and maintain their knowledge?

- **Assessment**. Can we identify measurable objectives and corresponding measurement methods? How can instructors be held accountable for conveying accurate and appropriate security content? Should faculty evaluations incorporate this issue? Can academia and industry agree on the desired educational outcomes?

- **Faculty development**. Is the master/apprentice model appropriate for this environment? If so, for which constituent group (e.g. new professionals)? How should train-the-trainer and peer review processes be incorporated in faculty development activities?

- **Tools**. What should the role of tools be in the curricula? How should the balance between an emphasis on concepts and an emphasis on tools be managed?

- **Sequencing**. When should introductory secure programming skills be taught (course sequencing) and how condensed should the programming curriculum be? At what level of associates/bachelors/master's/doctoral should secure coding be taught? What is the minimal level of security knowledge we want at each level and for what purpose?

- **Methods**. What is the best way to reach non-computer science programmers (e.g. those in other STEM disciplines like physics or engineering)? How can distance learning, in the form of online courses, be effectively used to teach concepts and skills?

- **Expectations**. Are the expectations of educational knowledge attainment aligned appropriately scoped? For example, MDs must extend their formal education through residencies and internships before licensing. Can we expect undergraduate computer science majors to master their craft through 4 years of education? What specifically do we expect to achieve by teaching software security to CS undergraduates? Is there a "type of thinking" that transcends the specifics of secure coding in a specific language? For instance, should students be trained in the mental organization of complexity, adversarial thinking, or reflective thinking?

- **Accreditation**. How should secure coding principles be incorporated into ABET requirements? How can computer security content be incorporated into non-computer science programs given other accreditation requirements and curricular constraints?

- **Licensure**. How might the portfolio of existing commercial certifications be integrated into a larger licensure structure? What are the right components of this licensure structure? Is such a structure appropriate? What standards must be developed to provide a generally accepted basis for certifications and licensure?

Acknowledgements

The Summit on Education in Secure Software, along with the writing of this final report, was made possible through the generous assistance of several individuals. In particular, the authors would like to thank the SESS participants and speakers for their participation; SESS facilitators from KnowInnovation, Andy Burnett and Stavros Michailidis; Dr. Monija Amani, who provided valuable support throughout the planning process; The George Washington University students and staff members who assisted during the summit as scribes and assistants; and Dr. Lesley-Anne Pittard, who worked alongside the organizers to plan, implement and debrief the event, and to draft this final report. In addition, the authors acknowledge resource and staff support from The George Washington University Center for the Study of Learning, The George Washington University Cyber Security Policy and Research Institute, and the University of California at Davis.

The Summit on Education in Secure Software was supported through the National Science Foundation Directorate for Computer & Information Science & Engineering and the Directorate for Education & Human Development under award #1039564.

Appendix A - Summit Overview

This document describes the goals of the summit and was distributed with the invitation to participate. It was designed to be "a work in progress," and subject to change based on feedback, suggestions, and (especially) proposed text.

Focus

Secure software development is a deep and tremendously important subject. Many problems arise from not focusing on the *security* aspects of software development. This workshop focuses on one part of that life cycle, the actual programming—sometimes called *coding*—phase.

We chose this part of the life cycle for two reasons. First, no matter how well the other parts of the life cycle are implemented, if the programming is non-secure, the software will have vulnerabilities. Second, the majority of the vulnerabilities reported to clearinghouses involve programming errors such as inadequate or incomplete input checking and buffer overflows. These errors are examples of poor programming practice, so we focus on that.

The term "secure programming" is something of a misnomer. It speaks to writing programs that satisfy security specifications. For our purposes, the "specifications" include programming practices that allow an attacker to force the program to perform actions that, in general, are undesirable. For example, a buffer overflow may cause a program to crash or execute instructions that it could not otherwise execute. So, we consider that a part of "secure programming" even though the execution of those specific instructions will not cause a violation of the site security policy. Similarly, if the result of an SQL injection attack reveals non-confidential information or grants authorized access, then strictly speaking that SQL injection does not cause a security violation; it only reveals what the attacker could have obtained in other, legitimate ways. Nevertheless, the ability of an attacker to carry out an SQL injection exploit indicates the program is written poorly, and again executes instructions (reveals information) that it would not otherwise execute (reveal) in that fashion. Hence we consider checking for this an example of "secure programming."

Perhaps a better term than "secure programming" is "robust programming," a style of programming in which one anticipates problems and programs to avoid them. A good example is writing a library in such a way that the internals are not available to the caller (modularity) so they can be changed when needed, and that all parameters passed to the library functions are checked. Someone once described this style of programming as paranoia, assuming stupidity, hiding dangerous implements, and never saying "can't happen." This language-independent style of programming addresses many programming errors, including those that "secure programming" addresses.

The focus of this summit is on helping programmers and non-programmers alike learn about "secure programming."[8] The broader scope, that of secure software development in the large, is *not* the focus. While it is a critical problem, we feel that simply teaching, and getting programmers to practice, secure programming is a difficult task. Approaches to meeting this specific focus will take all the time of this summit.

[8] Throughout the rest of this document, we use "secure" to conform to the more common usage. We are, however, speaking of teaching robust coding to anticipate and avoid problems (which we may not yet know about) brought on by fragile (non-robust) programming.

Finally, note that some collateral issues are discussed and are *very* relevant to the summit. These involve making people who *may* program—which includes people who will manage programmers and who will define the policies and procedures constraining the programs people will write—sensitive to the issues so they can provide the support needed for the programmers to program securely. But these will be discussed only as they come up in the course of discussing the main focus.

Specific Objectives

The stated goal of this workshop is "to develop a comprehensive agenda focused on the challenges of secure software education." This means the following.

Currently, people write non-secure code: code that attackers can exploit to compromise systems or activities. The state of software is generally to be of low assurance. The goal of this summit is to figure out how to improve that assurance. Specifically, what needs to be taught, how can it be taught effectively, where should it be taught, and to whom should it be taught? These questions also raise the problem of evaluation: how do we evaluate both the teaching and the learning?

In what follows, we expand on these questions. One central theme of this summit is that we are very unlikely to find "the" right way to do this because there are many "right" ways, the most effective one being dictated by environment and circumstance. So, rather than coming up with the solution, we expect to come up with many possible approaches for teaching the material. As examples, one way to teach secure coding is to require students to take a class in that topic. An alternate approach is to integrate the material into existing classes. A third is to have distinct modules teaching the material, but oriented towards the material being taught in the course; for example, a class on database security would have a module on SQL injection (checking input).

Each approach will require an understanding of the environment and conditions under which it is likely to be successful. Each will require resources to be carried out, and a method for evaluating its effectiveness. The conditions under which each will be most appropriate and effective are also important. Given these parameters, people and institutions can determine which method or methods work best for them.

Who Is To Be Taught?

The first question is whom should we teach? Possible students are:

- All students. As (almost) everyone will have contact with computers, they should know the dangers of using non-secure programs and what the symptoms of such programs are. Of course, the problem is that all non-trivial programs are non-secure given the right conditions. So, if this is our goal, how do we frame the material to make it meaningful, and to show the students what (if anything) they can do about it?

- Students taking any course in programming. As these students are likely to write programs, they should learn about the dangers of writing non-secure code, and how to avoid the most common instances of that. Should they learn more—for example, how to specify the assumptions that their program makes, so those using it will know under what conditions the author believes the program to be "secure"?

- Students in majors involving software development (for example, computer science). These students will develop software systems that are complex and may be widely used. Hence they should be aware of, and practice, good programming styles in order to reduce the number of security vulnerabilities that attackers can exploit.

- Practitioners. These programmers may not have been exposed to secure coding methods, or the risks of not practicing them. Their concerns are different than students who are following a course of study. While they need to learn the foundational principles, it may be more appropriate to focus on specific techniques for their particular environment rather than a more generic one that subsumes several different environments.

One interesting question is whether teaching students alone will be effective. For that reason, we may want to consider non-traditional categories as well:

- Managers and executives of companies that sell software. These folks set delivery schedules and feature sets as well as requirements for software products. If they are aware of the problems in secure coding, and the difficulties of testing and fixing poor code, then they can help by providing the resources and realistic schedules as well as considering the security implications of requirements and environments in which the programs will be used.

- Those who set policies. Often the policies that are set are quite unrealistic, because the limits of software, and the difficulties in writing and maintaining good software, are not understood. A better realization of the problems in software development, specifically good coding practice, may produce more practical policies.

- Students in K-12. The key questions here are how early we start sensitizing students to these considerations, given that many of them may not be technologically inclined or be too immature to assimilate the material (for example, kindergarten students). In the latter case, we may be able to make them aware of the issues in a general form, though. Would this be appropriate?

These categories would not need to learn how to write good code, but they would need to be exposed to why non-secure code is often released.

Who Will Teach Them?

This question raises the issue of teaching the material. Should faculty in existing courses do it, or practitioners, or other instructors specially trained in the topic? What resources will be needed to train or teach them?

What Are They To Be Taught?

Should everyone who programs be taught everything about secure coding? What is "everything"? Clearly, teaching someone who doesn't program how to detect race conditions in a program is not worthwhile, but where do we draw the lines and how? This is basically a curriculum issue, and will depend on the nature of the students. Possible "broad swaths" are:

- How to determine that a program is not written using secure coding techniques. This could teach being aware of the errors and problems that occur, either completely ad hoc or up to less formal (or very formal) testing methods.

- How to analyze programs for non-secure code. This would focus on static and dynamic analysis, with and without tools.

- How to write secure code. Here, students would learn to avoid common errors (such as overflows) and to program defensively (such as checking inputs).

- Non-programmers may also benefit from learning to examine results of programs or systems to understand the effects of poor coding. The two key questions here are:

- What should people unlikely to write programs be taught, if anything?

- How can this material be taught to people who are not programmers, may not be computer-savvy, and may not be particularly interested in computers?

How Do We Teach This?

Delivery and reinforcement of the material is crucial to ensuring the students learn and practice the discipline of secure coding. How can we do this, while students are in school or training programs, and after?

- A special class, or set of classes, on this subject. Students would take this class (these classes) early in their academic career, so they could apply what they learned while in school. Questions here are how to reinforce the material in the class continuously, in order to prevent it from being a class students take to graduate (get a good grade, or whatever) and then not apply the material because "it's just another academic class."

- Modules in existing classes. The idea here would be to have teachers in existing classes develop or use modules presenting ideas and methods of secure programming (or detecting non-secure programming) in existing classes.

- Clinics like a writing clinic. The idea is that the clinic can review programs to help students find potential problems, and can serve as adjunct graders who review turned-in programs for non-secure coding (which the professor can then take into account when assigning a grade for the work). The clinic can function in a variety of ways; for example, it can be tied to particular classes, or serve as a general resource throughout the school.

- Grading. What impact should secure programming have on a student's grades? How should this be done?

- After a person has been hired in a job that involves programming. The advantage to this is that the programmer will be working in a particular environment, with specific tools, and in one or more programming languages. That means the material can be tailored to that environment, those tools, and those languages.

Some of these ideas will be specific to courses. Others, like modules, can serve a number of educational institutions, each instructor modifying the module as he or she sees fit. Still others, like a clinic, could serve several institutions.

Evaluation

Evaluation will help determine how well the method of teaching and the material taught are working. This has several aspects:

- How is the teaching evaluated? Such evaluation must also take into account the resources provided and the desired results. An institution with no resources and an institution with many resources should be evaluated differently, but how to do this is unclear.

- How are the students evaluated? Should there be some "certification test" or some other mechanism for providing a standardized measure of how well a student has mastered the material? If so, what is a meaningful measure (or set of measures)?

- Retention. How well has the programmer retained this material over a period of time? This is important to show that the student can continue to apply the material after graduation.

- Support. How well does the organization/customer for which the programmer is coding support the use of secure programming techniques? Mere verbal direction is not enough; appropriate resources must be provided, including the time needed to write the program and test it to the degree desired.

- Continuous learning. As technology changes, so will the practice of secure programming. What provisions and incentives does a programmer have to stay up to date?

One intriguing approach is to pick some subset of common programming errors (for example, failing to check a bound, failing to check input, and one or two others) and use those as a basis for evaluation. Evaluation could include identification of instances of the problem, how to fix or prevent them, and in some cases how well the students understand why the construct is a problem. Which few programming problems would make a good "test set," and would also provide the most benefit, would be a good topic of discussion.

Appendix B - Organizing Discussion Questions

What is Secure Software?

- How is it defined among different communities (industry vs. academics vs. government etc)
- What are the challenges in crafting an objective definition of secure software

Teaching Secure Software?

- What subjects to teach
- What methods can be utilized to teach
- What resources are needed to teach effectively
- Key challenges to teaching secure software

Why Secure Software?

- What is the importance of understanding secure software?
- Why should this model be adopted by programmers at an early age?

Appendix C - Summit Agenda

DAY 1 (MONDAY, OCTOBER 18)

Morning Session (8:30 AM – 12:30PM)

8:30 AM – 9:00 AM Coffee, Registration
9:00 AM – 9:30 AM Opening Plenary [Diana Burley, Matt Bishop]
9:30 AM – 12:30 PM Keynotes

Perspectives on the state of secure programming education

Academia: Matt Bishop, Professor, UC Davis *[Confirmed]*
Government: Dickie George, NSA *[Confirmed]*
Industry/Certification: Alan Paller, SANS *[Confirmed]*

Luncheon Panel (12:30PM – 2:00PM) *Location: Foreign Service Association

Topic: Models from Other Professions

Yolanda Haywood, Associate Dean, GW School of Medicine *[Confirmed]*
Candice Hoke, Professor of Law, Cleveland State University *[Confirmed]*

Afternoon Session (2:00PM – 5:30PM)
Breakout Discussions by Mixed Group
Topic: How do we improve the state of education secure programming?

Dinner, Panel Discussion (6:00PM – 8:00PM) *Location: Foreign Service Association

Topic: "The Dean's Perspective - What is the Role of Higher Education?"

Michael Feuer, Dean, GW Graduate School of Education and Human Development *[Confirmed]*
Bill Chu, Dean (Former), University of North Carolina at Charlotte, College of Computing and Informatics *[Confirmed]*
Corey Schou, Associate Dean, University of Idaho School of Business *[Confirmed]*
Alec Yasinsac, Dean, College of Information Science, University of South Alabama *[Confirmed]*

DAY 2 (TUESDAY, OCTOBER 19)

Morning Session (8:30 AM – 12:00PM)

8:30 AM – 9:00 AM Coffee, Plenary
9:00 AM – 10:30 AM Morning Keynote

"Building a Securing Programming Curriculum"
Dr. Melissa Dark, Associate Dean, Purdue College of Technology *[Confirmed]*

Mixed Breakout Discussions (10:30 AM – 12:30 PM)

Breakout Discussions by Mixed Group

Networking Lunch (12:30PM – 1:30PM) *Location: State Plaza Hotel

Afternoon Plenary (1:30PM – 3:00PM)

Topic: "Developing a Comprehensive Agenda: Report Content, Structure, and Preparation"

Closing Plenary (3:00PM – 3:30PM)

Summit End

Appendix D - Participant List

Title	First Name	Last Name	Organization
Ms.	Laura	Adolfie	Office of Science & Technology Policy
Dr.	William	Arbaugh	University of Maryland
Ms.	Becky	Bace	In-Q-Tel
Dr.	Andrew	Bernat	Computing Research Association
Dr.	Matt	Bishop	University of California, Davis
Mr.	Tim	Brown	CA Labs
Dr.	Diana	Burley	The George Washington University
Mr.	Brian	Chess	Fortify Software
Mr.	Steve	Christy	MITRE
Dr.	Bill	Chu	University of North Carolina, Charlotte
Dr.	Stephen	Cooper	Stanford University
Dr.	Janice	Cuny	National Science Foundation
Dr.	Melissa	Dark	Purdue University
Ms.	Mary-Ann	Davidson	Oracle
Mr.	Dimitri	DeFigueiredo	Adobe
Dr.	Ron	Dodge	United States Military Academy at West Point
Dr.	Michael	Feuer	The George Washington University
Dr.	Richard	Ford	Florida Institute of Technology

Participant List (continued)

Title	First Name	Last Name	Organization
Dr.	Deborah	Frincke	Pacific Northwest National Laboratory
Mr.	Richard (Dickie)	George	National Security Agency
Mr.	Cassio	Goldschmidt	Symantec
Ms.	Rosey	Greer	Rosey Greer Consulting
Dr.	Yolanda	Haywood	The George Washington University
Dr.	Lance	Hoffman	The George Washington University
Dr.	S. Candice	Hoke	Cleveland State University
Mr.	Mike	Howard	Microsoft
Mr.	Joe	Jarzombek	Department of Homeland Security
Mr.	Jesper	Johannsen	Amazon
Dr.	Yoshi	Kohno	University of Washington
Dr.	Carl	Landwehr	National Science Foundation
Dr.	Karl	Levitt	University of California, Davis
Dr.	Keith	Marzullo	National Science Foundation
Mr.	Jon	McClintock	Amazon
Dr.	Ernest	McDuffie	National Institute of Standards and Technology
Mr.	Robert	McGahem	Office of Science and Technology Policy
Mr.	Gary	McGraw	Cigital
Dr.	John	Mitchell	Stanford University
Dr.	Kara	Nance	University of Alaska, Fairbanks

Participant List (continued)

Title	First Name	Last Name	Organization
Dr.	Christina	Nita-Rotaru	Purdue University
Mr.	Alan	Paller	SANS Institute
Dr.	Sean	Peisert	University of California, Davis
Mr.	Mathew (Pete)	Peterson	Naval Criminal Investigative Service
Dr.	Victor	Piotrowski	National Science Foundation
Dr.	Julie	Ryan	The George Washington University
Dr.	Corey	Schou	Idaho State University
Mr.	Robert	Seacord	Carnegie Mellon University
Mr.	John	Stewart	Cisco Systems
Mr.	George	Swan	Coverity
Dr.	Edward B.	Talbot	Sandia National Laboratory
Dr.	Blair	Taylor	Towson University
Mr.	Richard P.	Tracy	Telos Corporation
Mr.	John	Vicente	Intel
Mr.	John	Viega	Perimeter eSecurity
Dr.	Giovanni	Vigna	University of California, Santa Barbara
Mr.	Sam	Weber	National Science Foundation
Mr.	Jacob	West	Fortify Software
Mr.	John	Wood	Telos Corporation
Dr.	Alec	Yasinsac	University of South Alabama
Dr.	Ty	Znati	University of Pittsburgh
Dr.	Lenore	Zuck	National Science Foundation

Appendix E – Licensure/Certification Presentation Summary

Using Licensure or Certification to Upgrade Secure Programming Skills & Produce More Secure Software

Summary of Presentation – Revised Handout

Candice Hoke

Law School, Cleveland State University

shoke@me.com

0.1 This requested presentation proceeds from an assumption which this audience is presumed to endorse: **The software industry standard of marketing profoundly insecure software is unacceptable and must change**. Market pressures have not succeeded in upgrading the security attributes of software; experts note the trajectory of software quality has been downward over the past 12 years. The software industry's product quality standards and perhaps higher educational expectations (including academic requirements for computer science majors) have continued to be relatively unresponsive to the growing social, economic, personal, and national security injuries posed by insecure software. The questions the SESS organizers presented: Could licensure assist in upgrading software development to produce much more secure programming, and how does licensure differ from certification?

1.0 Definitions:

1.1 Certification: grants the right to use a title that reflects specified qualifications (education, skills, +/or experience); can be issued by government-authorized body, a nongovernmental (NGO) entity, a collaboration of entities, or sometimes by for-profit vendor-specific entities. Often these are voluntary certifications, though particular employers may require them as a prerequisite.

1.2 Licensing (or licensure): refers to legal authority to practice a specific occupation, ***granted by a governmental entity*** that also restricts that occupation or set of tasks to those who have satisfied specified education and proof of minimum competency. Typically licensing requires completing an accredited educational program plus passing a test (or other prerequisites) as determined by the licensing authority. For instance:

1.2.1 Architects: all States require those seeking to practice architecture to pass the national exam (ARE), and most require a practical internship and satisfaction of annual continuing education requirements; the field's credentialing is supervised by a national council (NCARB) but disciplinary issues (e.g., misconduct; practicing without a license) are handled at the State licensing level. The Council also maintains a website on the profession and its credentialing, offers continuing education opportunities, and seeks licensure reciprocity with other nations, http://www.ncarb.org/Becoming-an-Architect/Architecture-Basics.aspx.

1.2.2 Lawyers: almost all States mandate any person seeking to practice law in their State to graduate from an accredited law school and pass their bar exam, a portion of which is controlled by a national testing entity; an internship is not required but continuing education is a standard requirement for maintaining the license to practice. Discipline is handled at the State level (including for substance abuse, misconduct), with penalties including suspension and permanent surrender of the license to practice law in the State.

1.2.3 Airplane/airline pilots: the Federal Aviation Administration requires a pilot "certificate" in at least one of six authorized aviation categories that denote different levels of required skill; the certificate functions as a license to pilot specified aircraft. The FAA specifies educational, medical, and practical experience requirements, plus specific tests for each type of license. The FAA's airline pilot licensure program is ***binding Federal law*** rather than State-based.

2.0 A vast range of occupations & professions require some form of licensure, with many dating back over a century. Examples: Realtors, bartenders, lawyers, medical doctors, nurses, physical and massage therapists, cosmeticians, tattoo artists, acupuncturists, airline pilots, tractor-trailer truck drivers, architects, civil and mechanical engineers. The occupations selected for licensure differ among US States; most licensure is State government-based, but often is linked to and supported via national nonprofit professional organizations. The drive for licensure derives most often from the profession itself, as an attempt to rid itself of "incompetents, charlatans, and quacks." Licensure is often motivated by a desire to distinguish for the consuming public who can be trusted to perform competently when an asymmetry of relevant knowledge can lead to deceit and injury. [*E.g*., architectural/engineering expertise for earthquake locales; HVAC vent placement for uniform air circulation; legal "magic words" needed in contractual or trust documents.]

2.1 "***Junk certifications***": the software industry has a surfeit of these, which has also contributed to prior CS opposition to legally required certification or licensure for software engineers. The record has been dismal. Weeding the certifications would require governmental involvement at some level. Licensure of software development professionals may provide a sound pathway out of the morass.

3.0 Legitimate Rationales for licensure include promoting public health, safety & welfare and improving the ability of those lacking sufficient expertise to judge between competing service/product suppliers

3.1 ***Illegitimate***, *legally invalid rationales and motivations*: occupational job protection; extorting higher compensation by artificially reducing the supply of workers.

3.2 Software's critical importance: Software development/programming field cannot escape the rationales and history of licensing; the field is producing products & systems core to contemporary society; cumulatively, the consequences of insecure coding are socially and economically extremely high.

3.2.1 Software breadth of impact: Virtually all institutions that are charged with public health & safety either use or are required by law to use software-based systems such as automated databanks, websites, and networked communications. These systems are pervasive. Their functional reliability (or lack thereof) exerts extremely high social and economic impact. E.g. police fingerprinting, gun permits, arrest records; hospital & medical records; electoral voter registration and vote tabulations; water purification & distribution; medical devices, diagnostic and surgical equipment; air traffic and train controls; tax & property title records; electrical power grids & distribution controls; timed traffic signals; financial services and banking; surveillance cameras & related equipment; military weapons & defenses; satellite communications & entertainment; GPS devices; telephone services; billing & payment systems, including credit card transactions; automobile brakes & other safety equipment; air traffic control & automatic pilot/aviation software; bridge, dam, & other engineering design: Internet commerce & information access/distribution, especially via websites. All provide high commercial, governmental & personal value that has become central to the US economy and essential governmental services & functions.

3.2.2 Software Integral to All Vital Institutions Thus, the software industry cannot "escape" on the grounds that its products and services are not integrally related to the public

welfare. (Sorry!) The industry has successfully embedded its products in virtually every aspect of life and every major institution-- which means it will be virtually impossible to deny that the practitioners of the industry's core skills of programming must meet minimum standards of performance.

4.0 Structures & Process for Attaining Licensure: There's no "one size fits all" form of licensure; licensing systems can (and should be) tailored to enhance distinctive aspects of the professional field and to minimize harm to the field's public and economic contributions. Thus, structuring an appropriate software development licensure system can be a creative act of "customization." The effort should study errors as well as successes of other licensure programs, and identify the particular public goals the licensing program should achieve and harms the program should avoid. *[More detail at 10.0 - 19.0.]* A dedicated group could work to develop a sound proposal and seek support among professional organizations, before presenting it to the selected governmental level (State or Federal) for approval.

5.0 Hidden Benefits of Licensing include exerting a direct, corrective impact on higher education curricular decisions and instructors' course coverage choices. If software programming licensing required demonstration of secure software coding skills, for instance, CS programming courses (at the university and community college levels) would be revised rather quickly; new educational modules would be sought and taught – for current students and as continuing education for existing programmers; and, a more rapid adoption of these skills could be achieved than by exhortation to the CS educators who themselves were not taught computer security and secure coding practices. For instance, bar exam requirements do make a big difference in many law schools' curricula decisions and in many instructors' course syllabi.

6.0. Fears of licensing ARE justified by some poorly designed licensing or certification systems. *E.g.*, junk certifications pervading the software development field, and also some antiquated licensing programs of other occupations (that have not been updated and generally function as strategic devices to protect work for a chosen few insiders) can be cited. ***But don't assume these errors inhere in all licensing systems or are inevitable***. Many licensure programmatic choices/options exist; a system with fewer risks to valued aspects of the industry and its workers can be crafted. Careful, depth review is needed, more than is possible in my 20 minutes with you today *[but I have a more systematic academic article underway]*. The key is to understand the software development industry, its achievements and its deficits, the skills needed and the deficits that need to be eliminated in software programmers and developers. Then, leaders must structure the licensing program with features that will enhance and extend the good while retarding the deficient aspects (particularly the failure to learn and practice secure coding)**.**

7.0 Cautions and predictions: *If you don't move forward with structuring a licensure system, others will; the downstream impact from insecure software is too grave for a licensing system not to be superimposed on the software development industry.* Litigation & imposition of liability is possible despite software vendors' efforts to disclaim warranties. If a digital 9-11 occurs, the industry and its academic leaders will have far less opportunity to significantly craft the licensure program than if a determined, good faith effort commences now.

7.1 Who should move forward? CS and computer security academics joining with software industry representatives, a few lawyers and government reps, could collaboratively structure a top drawer licensing program that will seek the nation's maximum interest-- which aligns nicely with that of corporate and commercial interests (both the software industry and the purchasers), and that of the software programming workforce. It could be a win-win if pursued now, an effort unencumbered by complications caused by political and public high-pressure demands in the aftermath of a massive

avoidable digital disaster. Part of the decisions must be to determine which governmental level should be the license-granting entity.

Further decisional details follow ->

***Some* Licensure/Certification Questions to Ask & Options to Consider**

10.0 Licensure Program Components to Consider

(a) Initial licensure/entry requirements, possibly including educational prerequisites, demonstration of particular skills either through a test of some other mechanism; ethics, and understanding of social costs/impact of insecure programming;

(b) Continuing Competence in dynamic field (continuing education requirement);

(c) Discipline & penalties e.g., suspension or revocation of licensure.

11.0 Determine the specific field/positions or tasks to be covered by the licensure program; may being with a narrow set and phase-in others.

12.0 Structural Options for the Licensing Authority & Governance

12.1 Hybrid National & State licensure: the licensure authority could be planned to be a nonprofit formed by consortium of esteemed professional and research organizations such as ACM-US, the IEEE Computer Society, USENIX, and others that include representatives of software industry, academia, government, and the informed consuming public. Might include a formal role for specific governmental entities (Council of State Governments; NSA). It would devise the body of knowledge and other prerequisites for licensure. States (by enactment) would effectively delegate specific functions to the nonprofit but retain licensure and discipline at State level.

12.2 State Licensing Boards that form a consortium of State Boards as in medical & architectural fields, but 1 national testing organization? (May require too much time to generate and thus be too late for a realistic software industry option.)

12.3 State-based licensing (least recommended); "race to the bottom" problems.

12.4 Federally-based licensing similar to aviation pilot licenses under the FAA's rulemaking authority could be initiated and exert broad impact within the software industry. This option could be omnibus, as in aviation with the FAA controlling the terms under which any person may pilot an aircraft in US airspace. Or, software licensing could be differentiated between federal departments. For example, the Food & Drug Administration has regulatory power over medical devices, including those containing software. The FDA might require licensure of programmers of software used with human safety-critical medical devices, or all devices it regulates, as a pre-condition of filing for pre-market regulatory approval. The DoD might require licensure of programmers in military systems of specified sorts. These department-specific licensing systems might be inconsistent and more costly to generate and manage than one omnibus federal software licensing system.

13.0 Provide levels of licensure *or* minimum (entry level) qualifications only? Aviation piloting licensing carries at least 6 types of licenses; an individual can qualify to maintain several licenses. Lawyers and physicians are licensed at the State level with entry-level qualification. To distinguish physician qualifications, medical doctors can become "Board-certified" in a field beyond their initial license to practice medicine. An increasing number of States Bars are offering advanced

certifications in specialized areas of law, for some of the same reasons as licensure occurs—to assist the non-lawyer public in sorting through qualifications for particular legal matters.

14.0 Permit Reciprocal Licensure? – if State governments are the licensing unit instead of a federal agency, whether licensure by one State can result in "waiving" into other States by filing proof.

15.0 Include an Ethics component? Provide an enforcement & disciplinary process for misconduct? *E.g.*, for inserting back doors or other options for sabotage or unauthorized access.

16.0 Options for obtaining licensure – *could permit **one or all** options*

16.1 Completion of approved educational program that the Governance Board "certifies" having combination of content, skills (clinical component?), professional ethics, and other specified learning, thus rendering those students (a) qualified to take the exam, or (b) exempt from the exam, or (c) required to take only a mini-test;

16.2 Testing: licensure may require passing a test. Many questions follow, including *testing structure* (a series of separate tests, as in Architecture, or a one-sitting omnibus test as in Law); *scope/subject areas* to be tested, including for each subject area of competency whether to test primarily skills in applied contexts, *e.g.*, applied secure coding skills, theoretical knowledge, or both.

16.3 By Credentials submitted: Individualized review & approval by appointed committee of professionals, which might work for some of the existing software workforce sectors but also might prompt accusations of unfairness and lack of objectivity.

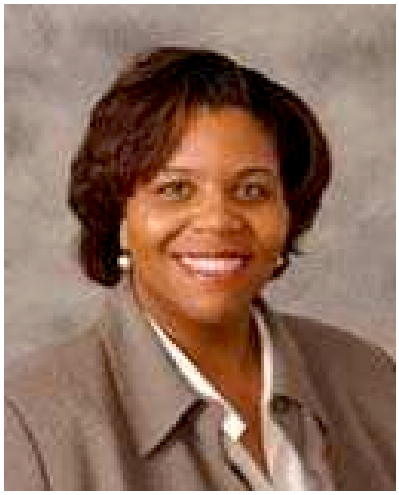
17.0 Phasing in or sequencing to encompass currently active programmers, not only newbies?

18.0 Federal Strategy to Jump-Start: Federal contracting option to jump-start (swift-kick) the process by adapting the strategy used in Exec. Order 11,246. (I'd be happy to work with Federal government representatives to provide more content to this option.)

19.0 *Summary*: Highly valued professions are licensed to facilitate higher standards and quality of work. The creation of a licensure system customized to be conducive to secure, ethical software development is overdue but achievable. If delayed until a digital 9-11 event, crisis management politics will constrain options and the time available for careful, collaborative deliberation on the best programmatic licensure features for software development. Consider licensing ***a mark of having arrived*** as a vital profession as critical as law, architecture, medicine, and airline piloting.

Diana Burley

The George Washington University
Graduate School of Education and Human Development
Department of Human and Organizational Learning
dburley@gwu.edu



Diana L. Burley is an Associate Professor in the Graduate School of Education and Human Development at The George Washington University. Dr. Burley joined GW in 2007 and during her tenure at the university she has served as the inaugural Department Chair of the Human and Organizational Learning Department and as Director of the Executive Leadership Doctoral Program. Prior to joining the GW faculty, she served as Program Officer in the Directorate for Education and Human Resources at the National Science Foundation (NSF). At NSF, she managed multi-million dollar grant programs designed to increase the capacity of the U.S. higher education enterprise to produce professionals in scientific fields. Her area of expertise at NSF was in computer science education. Based on her work, she was honored by the Federal Chief Information Officers Council and the Colloquium on Information Systems Security Education for outstanding efforts towards the development of the federal cyber security workforce. She currently serves as Vice Chair of the Association for Computing Machinery Special Interest Group on Computers and Society. Dr. Burley holds an M.S. in Management and Public Policy, an M.S. in Organization Science, and a Ph.D. in Organization Science and Information Technology from Carnegie Mellon University.

Matt Bishop

The University of California, Davis
College of Engineering
Department of Computer Science
bishop@cs.ucdavis.edu

Matt Bishop received his Ph.D. in computer science from Purdue University, where he specialized in computer security, in 1984. He is on the faculty of the Department of Computer Science at the University of California at Davis. His main research area is the analysis of vulnerabilities in computer systems, including modeling them, building tools to detect vulnerabilities, and ameliorating or eliminating them. He has participated in numerous studies of computer systems, including as one of the co-PIs of the California Top-to-Bottom Review of electronic voting systems certified for use in that state. He is active in information assurance education, and has presented tutorials at SANS, USENIX, and other conferences. His textbook, *Computer Security: Art and Science*, was published in December 2002 by Addison-Wesley Professional. He also teaches software engineering, machine architecture, operating systems, programming, and (of course) computer security.