Proceedings of the 13th Colloquium for Information Systems Security Education
University of Alaska, Fairbanks
Seattle, WA June 1 - 3, 2009

Cross-site Security Integration: Preliminary Experiences across Curricula and Institutions

Blair Taylor, Harry Hochheiser, Shiva Azadegan, and Michael O'Leary, *Towson University*

Abstract – Despite its clear and growing importance, computer security education is often relegated to a secondary role in undergraduate curricula. Exposure to computer security concerns is often limited to specialized courses and tracks that reach only a small percentage of students, often late in their academic careers. Effective security education approaches must engage more students earlier in their education. These techniques must be adaptable to fit the needs of differing educational institutions and student bodies. Our earlier work with checklist-based security lab modules in CS0 and CS1 provides a basis for a model that can be applied throughout the undergraduate curriculum and at a wide range of institutions. We present an agenda for extending this model from introductory to upper-level undergraduate courses at institutions ranging from community colleges to comprehensive universities.

Index terms: Security Injections, Secure Programming, Lab Modules, Security Integration

I. INTRODUCTION

Although security is commonly agreed to be an important topic for computer science and information systems education, this importance is generally not reflected in current curricula and instructional models. While many institutions have added security classes and security tracks, security is not well ingrained throughout undergraduate computing curricula. New approaches aimed at bringing security concerns to all computing students are desperately needed.

Many of the most common security problems – and particularly those which are most amenable to inclusion in undergraduate contexts – are based on failures to apply computing concepts and lessons. From a computer science viewpoint, coding errors that lead to common security flaws such as integer overflow, buffer overflow, and input validation failures reflect a basic failure in coding fundamentals [1,2]. Code with these vulnerabilities is not simply insecure code: it is bad code.

This perspective on computer security education informs the design of our new approach to computer security education. If we are to teach students how to code correctly and securely, we cannot leave these topics to specialized electives taken in the final year of an undergraduate education. Security concerns must be introduced early – before habits are formed – and reinforced from multiple perspectives throughout subsequent courses.

A novel pedagogical approach is necessary for re-conceptualization of computer security education, but it is not sufficient. Two-year institutions, small colleges, and large universities differ widely in their curricula and students. Models that address only one course in one institution may succeed in their narrowly-defined context, but successful generalization to wider audiences is by no means guaranteed. Proper attention to dissemination to a variety of educational contexts and challenges will make these new approaches more generalizable.

Building upon our earlier work [3], we have developed a model of "security injections" – targeted modules that can be integrated into existing courses with minimal overhead. This paper will describe some requirements for both a conceptual approach to computer security education and for the generalization of this approach. Specifically, we will present a model for extending this approach, moving beyond the core computer science courses to include upper-level computing courses in networks, databases, and web programming, with an eye towards meeting the needs of students at institutions ranging from two-year community colleges to four-year universities.

II. RELATED WORK

Security is a relatively new challenge in the dynamic field of computer science. The current state of security education comprises an increasing number of security tracks and specialized security courses, such as cryptography and digital forensics. Towson University established a security track for computer science majors in 2002. While feedback from students, faculty, and industry has been positive [4], the number of students remains small in comparison to the hundreds of students in the computer science and computer information systems majors, and includes few female and minority students.

Recently, there has been increased attention in both industry and education towards secure software and

Authors' affiliation: Department of Computer and Information Sciences, Towson University, 8000 York Road Towson, MD 21252 USA

ISBN 1-933510-96-9

systems development [5]. Since most security threats are the result of system or software vulnerabilities, it is imperative that all current and future software developers are proficient in secure design and programming. While textbooks have been slow to address these problems, some recently-published lab materials specifically address common security flaws such as buffer overflow [6,7,8]. In 2008, the SANS Institute helped sponsor a workshop on Secure Software Development to allow faculty and industry security experts to share ideas and create coding exercises.

Security education in its current state reaches too few students, too late in their curriculum, after they have developed insecure programming habits. Many agree that security should no longer be an afterthought, but instead must be seamlessly integrated or threaded across the entire computing curriculum, beginning with the foundation courses and re-enforced throughout all students' course of study [2,9,10,11,12,13,14,15,16]. Despite this, comprehensive security integration is largely an unrealized ideal.

III. PREVIOUS RESULTS

Two years ago, we began implementing a security integration model. Our primary goal was to reach more students, earlier in their curriculum. Towards this end, we have been developing, piloting, and assessing a series of strategically-placed security-related modules, or security injections, into our core computer science courses: CS0, CS1, and CS2.

Security Injections. Security injections are lab modules that target crucial security issues. As described in detail in our previous paper [3], each security injection module includes a background section, a lab exercise centered around a security checklist, and related discussion questions.

Background information includes a summary of the vulnerability, a description of the problem and risk, a documented example of a real or potential problem in an existing system, and advice on avoiding the vulnerability. For example, a security injection module which targets integer errors describes the Comair program crash in 2004 that occurred when bad weather caused the number of crew changes to overflow a 16-bit integer [17]. When appropriate, the modules also contain short code snippets that demonstrate the vulnerability.

The laboratory component includes exercises designed to give students experience with the particular vulnerability being targeted. The labs are specifically crafted to dovetail with the course concept [3]. Input validation, for instance, is introduced with the topic of selection and looping in CS0 and CS1 and reinforced with functions and classes in CS2.

A core component of the security injection module is the security checklist, which gives students a clear procedure for identifying vulnerabilities in their code [3]. Each checklist targets a particular vulnerability and is specifically designed for the course and experience level, For example, students in CS0 use a checklist to identify arrays with potential buffer overflow. Students in CS2 use checklists to check loop limits and array arguments for possible overflow conditions. Checklists can serve double duty on some exercises, acting as scorecards that instructors can use to grade the security of students' code.

The final component of the security injections is analysis in the form of discussion questions. Asking students to reflect on their work enhances the active-learning approach of the security injection modules.

Results. In spring 2007, we began pilot testing the security injection modules in the first two programming classes in our introductory sequence: an elective introduction to programming designed for students with limited programming background (CS0), and the first course in our required two-course seqeunce (CS1). In fall 2007, we piloted a section in the second required course (CS2). To measure the impact of our model, we created and administered a 25 question pre and post questionnaire. The questionnaire included 5 demographic questions and 20 security questions. In total, we tested 392 students pre and 275 post and found a significant increase in student awareness of security concerns for those students who completed the security injections [3].

These results indicate that the security injections are an effective way to teach security across the curriculum with minimal impact on already-overburdened undergraduate degree programs. These modules also serve as valuable supplemental materials for faculty, augmenting the meager, or non-existent, discussions of security concerns in existing computer science textbooks.

IV. INCREASING BREADTH

Two-year institutions, small colleges, and large universities vary greatly in their curricula, students, and resources. The ever-changing nature of computer science provides unique challenges for educators and computing programs differ widely in terms of requirements, course sequences, and electives. Even introductory computer science courses suffer a lack of uniformity in programming languages, platforms, textbooks, instructors, and requirements; yet, these courses are often expected to transfer seamlessly between two and four year institutions. Despite these differences, educators at

different institutions share similar issues, including overextended curriculum, lack of resources, the challenge of educating diverse populations of students, and the pressure to stay up-to-date in a rapidly evolving field.

To increase the applicability of our model, we are currently working with four partnering institutions, including Anne Arundel Community College, the Community Colleges of Baltimore County, Harford Community College, and Bowie State University, a four-year historically-black university. Our efforts to understand the specifics of teaching computer security led to a pedagogical approach that can be applied to varying types and sizes of institutions. What began as a pilot program in one section of one course has expanded across five institutions and four courses, with additional courses in development.

The collaborative nature of this project led to a cyclical model for developing these materials. Borrowing from the software development lifecycle and NSF's cycle of innovation [18], our approach to implementing curriculum change across multiple institutions is shown in Figure 1.

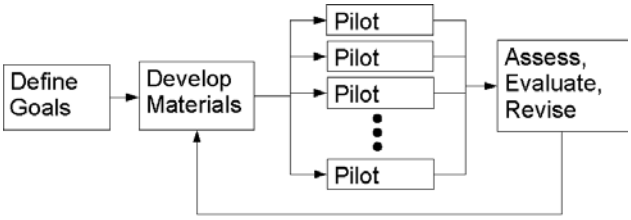

Figure 1: A collaborative model of improving and disseminating materials

Define Goals. Our project began with a need to address deficiencies in security education and our goal was to infuse security throughout the curriculum. The objectives were to increase students' security awareness and ability to apply security principles, increase the number of security-skilled students, and improve faculty awareness. It was advantageous to the success of the project that our goal was a well-established ideal shared among computer science professionals. It was relatively easy, in our case, to "sell" the concept of security amongst our colleagues. Clear, reasonable goals increase motivation towards participation and eventual dissemination and deployment.

Develop Materials. The primary materials for our project include the security lab modules and the instruments used for assessment. The writing and revision of lab modules is an iterative and collaborative process. We place all modules on a project wiki (available from the project web page http://www.towson.edu/cosc/securityinjections/) for downloads, comments, and revisions. The security injections are repeatedly revised and improved based on pilot testing, assessment, and peer reviews. Additionally, to formalize the assessment process, we have developed evaluation instruments including pre and post questionnaires.

Pilot Test. An important element of our model is incremental pilot testing. Pilot testing begins at the start of a semester with a base set of materials, an assessment instrument, and one instructor. We found instructors reacted most favorably to a flexible, voluntary approach, so pilot testing may include multiple instructors who choose to use only a subset of the materials. Feedback from the pilot tests provides guidance for fine-tuning the contents of the injections. Participation in the pilot tests provides instructors with experience that will help them act as mentors to colleagues piloting or deploying modules in subsequent semesters. It is recommended that each phase of curriculum change, new materials, a different course or a different institution, begin as a pilot.

These curriculum changes are intended to be minimally invasive. Therefore, specific details of deployment are largely determined by the individual instructor. It is expected that instructors will modify content as needed. Larger classes (>100 students) may require staged deployment.

Assess, Evaluate, and Revise. Unique to our model and critical to producing effective curriculum materials was a formal assessment process. Formal assessment includes pre-tests and post-tests across all sections, including controls, of targeted courses; scorecards; and faculty surveys to assess student achievement. Independent consultants in both technical content and pedagogical assessment provide additional feedback.

Student awareness of security issues is measured pre- and post-exposure to the modules. Data collected, including overall and content specific answers from pre-tests and post-tests, sampling of security injections, scorecard evaluation, and faculty feedback from survey instruments each semester informs the process of revising content. Subjective responses and comparisons between versions of module material will be particularly important in this regard, providing the project team with insights into what worked and what did not. Content is updated accordingly after each semester.

In addition to validating the security injection modules, formal assessments evaluate the success of a particular approach for integrating an important concept across several undergraduate classes. Comparison of the qualitative and quantitative measurements will be used to improve pre- and post-test questionnaires, with the goal of developing broadly applicable models.

Data from different sections of each class and from each institution will be examined for differences that will

increase our understanding of the factors that determine the success of security injections. Differences between institutions (university vs. community college), classes (introductory vs. upper-level) and student profiles (particularly presence of females and under-represented groups) that are correlated with variances in effectiveness of these approaches would be of particular interest.

Our experience with the initial deployment [3], detailed in the previous section, culminated in the above model. The direct result of the collaborative and iterative nature of the process is a quality product, namely the security injections. By beginning with a base set of materials and operating under an open source framework, we continue to develop a robust and well-tested set of materials that works across many institutions. All partnering institutions have begun pilot testing modules in at least one of the courses.

V. INCREASING DEPTH

While it is important to expose students to security concepts early in their studies, before they learn poor habits, incorporating security injections in the introductory classes is not sufficient. Security in depth incorporates a multilayered approach and comprehensive security integration requires expanding the security injections to other courses beyond the core. To ensure thorough comprehension of key security concepts, repetition and reinforcement is important. Additionally, more advanced level classes allow further and deeper investigation of security issues. Following the security injection model we have developed for CS0 and CS1 [3], and in the spirit of recent efforts that have built upon this model [8], we are expanding our integration to include a variety of courses throughout the undergraduate curriculum. These efforts will include programming-intensive courses from computer science programs and systems-level courses suitable for information systems/computer information systems majors as well as the introductory computer literacy course for non-majors.

Content in these modules is organized around the three main themes of the Common Weakness Enumeration/ SANS top 25 most dangerous programming errors: insecure interaction between components, risky resource management, and porous defenses [19]. Building on materials developed for the first two introductory courses (CS0 & CS1), we are moving on to several subsequent courses in the undergraduate curriculum. The second required course in our sequence (CS2) was piloted during the 2007-2008 academic year, and materials for database, networks, web development, and introductory general education course are under development.

An overview of the mapping between these themes and the specific contents of current and potential modules is given in Table 1.

High Level Category	Security topic	Courses
Risk	Risk analysis and management	*DBase, CIS0*
Insecure Interaction Between Components	Input validation	*CS0, CS1, CS2, Dbase, Networking, CIS0*
	SQL Injection	*Dbase*
	Cross-Site Scripting	*Web*
Risky Resource management	Buffer overflow	*CS0, CS1, CS2*
	File names & search paths	*CS2, Networking, Web*
	Download of code without integrity check	*CIS0*
	Improper initialization	*CS1, CS2*
	Incorrect calculation (integer overflow)	*CS0, CS1, CS2*
	Improper resource shutdown or release	*CS2*
Porous Defenses	Improper authorization	*Networking*
	Hardcoded password	*CS1, CIS0*
	Insecure permission assignment for critical resources	*Networking, Dbase*
	Insufficient randomness	*CS1, CS2*
	Malicious file execution	*CIS0*

Table 1: Mapping CWE/SANS top programming errors [19] to specific courses.

A. CS0, CS1, and CS2

Materials for CS0, CS1, and CS2 focus on the "big three" security vulnerabilities: integer errors, buffer overflow, and input validation [20]. Each vulnerability is introduced at a level that corresponds with the appropriate course topic or primitive, such as selection, looping, or functions [3]. For example, CS1 includes three input validation modules: a gentle introduction to validating input to be used when discussing selection (if-else and switch), a more robust coverage employing loops (while and do-while), and a final module that discusses input validation with functions. See Table 2 for an overview of modules and topics covered in CS0, CS1, and CS2.

Vulnerability	Task	CS0	CS1	CS2
Integer Error	Data	X	X	X
	Operations	X	X	
Input Validation	Selection	X	X	
	Loops	X	X	
	Functions	X	X	X
	Classes			X
Buffer Overflow	Arrays	X	X	X

Table 2: Coverage of vulnerabilities and tasks in introductory courses.

A set of common learning objectives (Table 3) links these materials across the three classes. By making these objectives progressively more challenging as students move from CS0 to CS1 and CS2, we hope to use repeated exposure to reinforce the relevant concepts. Modules for CS0, CS1, and CS2 are provided in both Java and C++, to accommodate varying preferences for languages of instruction. Translation to other languages should be straightforward.

Objective	CS0	CS1	CS2
Describe the vulnerability	X	X	X
Describe problems that may result from the vulnerability	X	X	X
Identify potential vulnerabilities in a simple program written in the language of instruction	X	X	X
Discuss general strategies for mitigating or avoiding potential vulnerabilities		X	X
Write code that uses appropriate techniques to mitigate or avoid potential vulnerabilities		X	X
Revise a program, eliminating potential vulnerabilities			X

Table 3: Learning Objectives for CS0, CS1, and CS2

CS0 and CS1: Security injections for CS0 and CS1 present very basic introductions to the various vulnerabilities via programming assignments that demonstrate software weaknesses and require students to write simple code without vulnerabilities [3]. In CS0, students run C++ code that demonstrates integer errors and buffer overflows. In CS1, they use checklists and appropriate test data to ensure they write programs without these vulnerabilities.

Security checklists are introduced incrementally. Specifically, students in CS0 first learn to identify potential integer errors and progress to recognize improperly validated input data and possible buffer overflows. Repeated exposure to security checklists teaches students to self-check and reinforces key security principles.

CS2: As the second required course in the introductory programming sequence, CS2 generally focuses on object-oriented programming (including inheritance and polymorphism), along with an introduction to basic data structures. CS2 modules present more challenging examples with more realistic code – for example, implementing routines to extract substrings from a character array. CS2 modules also differ explicitly in the added objective of modifying programs to eliminate potential vulnerabilities. Students are asked to revise programs after using the checklists to spot buffer accesses and integer operations that might be associated with security flaws. This additional task presents challenges that go beyond writing code without vulnerabilities, while providing a basic introduction to the realistic task of fixing someone else's code.

The greater sophistication of CS2 students allows for security injection modules that explore these topics in-depth. Once students have completed introductory work in object-oriented programming, they will implement basic classes that provide safer integer operations, implement managed buffers such as vectors, or provide robust input validation. Future modules will also address topics such as entropy, randomness, secrets, and basic encryption.

B. Databases

Security injections for a database course address both programming and design concerns. As perhaps the most familiar and dangerous class of database security vulnerabilities [20], SQL injections are a natural starting point. Input validation for queries and the appropriate use of prepared statements and parameterized queries will be the basis for additional modules.

Database design issues will also focus on data organization and management strategies aimed at reducing risks. Authentication, access controls and least privilege models will be discussed in terms of minimizing unauthorized access or modification of data. More specifically, security injections will be prepared related to mandatory access control, discretionary access control, and role-based access control.

Appropriate use of cryptography for protecting sensitive information will be discussed [23], along with strategies for storing and managing authentication-related information (e.g., storage of hashed passwords instead of originals). Issues related to information leakage and database fingerprinting through error messages and the use of logging facilities as a means of supporting security auditability are other topics of interest.

C. Networks

Security injections for a networking course will focus on theoretical aspects of network protocol design and practical issues of network configuration. Protocol design questions will address flooding, denial of service, and network encryption (link and end-to-end). Network configuration topics include firewalls, DMZs, proxies, authentication, and tunneling.

D. Web Development

With content including page design with HTML and CSS, client-side scripting, and database-driven web applications, web development courses include numerous security-related concerns. The Open Web Application Security Project's (OWASP) top 10 web vulnerabilities [21] fit within the larger framework of the CVE/SANS themes [19] to form a roadmap for addressing security issues in web development. Many of the topics discussed in the afore-mentioned courses - particularly input validation and SQL injection - would be applicable in this course as well. Other topics of interest include cross-site scripting, and cross-site request forgery (insecure interactions); malicious file execution, information leakage, failure to restrict URL access, and insecure direct object reference (risky resource management); and improper session management, session management problems, insecure communication, and insecure cryptographic storage (porous defenses) [21,22].

Security injections for these topics involve a combination of design questions regarding handling of information, construction of URLs and web parameters, and information flow in web applications. Programming examples and exercises may involve best (or worst) practices in managing security issues in web frameworks such as Java EE, PHP, Ruby on Rails, or .NET. Security training tools such as OWASP's WebGoat [22] might provide appropriate exercises.

E. CIS0

To expand the scope of our integration beyond the major classes listed above, we have also targeted the computer literacy course, or "CIS0", as an effective way to reach an even larger number of students. Designed for non-computing majors, CIS0 is an introductory level breadth-first approach to the fundamental terminology, concepts, and applications of computing. Typically housed in the computer science or information systems department to satisfy a general education requirement, CIS0 is generally transferable between two and four-year institutions. Consequently, CIS0 is a popular choice for students looking to gain valuable technology skills; our own institution generally offers over 25 sections per semester. This course presents an excellent opportunity to extend security integration to a large and diverse audience of students, early in their studies.

Brainstorming with our partner institutions, we identified some of the following topics as appropriate for security injection modules in CIS0: phishing, asset and risk management, password management, social engineering, passwords, firewalls, data mining, malware, and physical security. Introduction to important security terminology, awareness, and real life examples is important at this level. Sample exercises might include phishing activities, risk analysis, and researching sites such as CERT for current events including vulnerabilities and malware warnings. Currently, Bowie State University is piloting several security injection modules in its CIS0 course, including topics such as phishing, input validation, passwords, and risk analysis.

VI. ASSESSMENT

Our earlier work in CS0 and CS1 used a questionnaire to assess student awareness of security and secure programming concerns. Surveys were administered at the start– before exposure to the security injections – and at the end of each semester. Comparison of these pre- and post-tests yielded encouraging results [3]. Continued efforts using a refined version of this survey will allow cross-institutional comparison and (through anonymous ID numbers) examination of differences on a per-student basis.

Assessment of mastery of content material will take two forms. Samples of completed lab modules will be collected and independently-graded by multiple instructors. These results will be used to assess the success of the modules relative to objectives such as those given in Table 3.

Additional assessment exercises, in the form of questions and problems, are currently under development. Where possible, these exercises will be included on quizzes and tests, with responses collected and reviewed relative to learning objectives. It is important to frame these questions with an emphasis on best practices rather than language-specific issues. Thus, instead of asking students to write code that avoids integer errors, a test question might ask students to ensure that their code robustly handles a wide range of input values.

VII. DISCUSSION

Five interdependent aspects of our model combine to provide a foundation for sustainable curricular change:

Clearly targeted goals and mechanisms: The combination of a clear project goal - integrating computer security into existing undergraduate computing classes – and mechanisms for meeting that goal – the security modules – provide an easily-understood model that has proven effective for engaging colleagues and educators.

This focus has been particularly important for helping collaborators understand the scope of our project. If we had suggested a wholesale revision of all courses in the undergraduate curricula to include explicit security modules, we probably would have encountered significant resistance from colleagues who would have been (justifiably) concerned about the impact of such an ambitious agenda. Instead, our model proposed a modest, well-defined set of changes to a manageable number of courses. These achievable goals are much easier to sell to colleagues and administrators.

Manageable expectations of colleagues: Even with the small number of courses included in our project, any changes that required substantial time and energy commitment from faculty colleagues would have been unlikely to succeed. We specifically designed our curricular proposals to be minimally-invasive: instead of asking colleagues to take time to develop new materials, we provided them with modules that help them address the constant need for challenging lab exercises. So far, this had been an effective strategy for generating engagement.

Templates and roadmaps: High-level descriptions of goals and techniques may be necessary for contextualizing the rationale behind curriculum changes, but they aren't sufficient for making such change happen. All of the materials that we develop are presented in a common format on the project web site. In addition to providing concrete details that clearly moved the effort out of the realm of abstraction, these documents provide a starting point for future collaboration and contribution, as project participants can see how they could use these models to revise materials or to develop entirely new modules.

Collaboration and feedback from project participants with varied perspectives: Colleagues from our partner institutions were engaged from the early stages in active roles. Meetings that were initially scheduled as training sessions morphed into open workshops that provided invaluable feedback. These workshops, which included open discussions and breakout groups to promote feedback, reflection and sharing of teaching strategies, helped to integrate faculty early into the development process and maximize feedback in the hopes of identifying factors that work well across the different demographic groups.

Although financial support for participation certainly helps to motivate participation, we believe that including participants as true partners helps sustain engaged commitment. Modules developed by the authors were presented to project participants as drafts – initial proposals that served as a starting point for discussion and revision. Feedback, criticisms, and input from the participants was repeatedly encouraged and incorporated into the materials.

Multiplicity of Models: The range of participants in our security project underscores a fundamental challenge in applying curricular changes to multiple, differing institutions: there is no "one-size-fits-all" solution. We have repeatedly encouraged our partners in the project to use our materials as starting points for customization to meet their special needs. These revisions will be made available along with the originals, providing a variety of interpretations and perspectives that will be accessible to a broader audience of instructors. Providing a range of options – particularly to those who are not directly associated with the project – will move us in the direction of a community effort that might be able to provide "something for everyone."

VIII. STATUS AND FUTURE WORK

After extensive pilot testing at Towson University, materials for the first courses – CS0 and CS1 – are currently being deployed at Towson University and piloted across our partnering institutions. Currently, piloting of CS2 is in progress at Towson University and piloting of CIS0 has begun at Bowie State University. Training, piloting, and deployment are ongoing and will continue until all partners are using these materials. Materials for web development are under development; database and networking materials will follow, with materials generally being piloted at the institution of the developing staff before deployment to partner institutions. Assessment and revision will be ongoing processes for all classes.

Dissemination and adoption remain significant challenges. An open model that invites participation while providing instructors with the flexibility needed to adapt materials for their particular needs may remove some obstacles, but systemic curriculum change requires broader commitment. Our experience has been that most instructors understand the need for more pro-active engagement with security issues in undergraduate courses, but time constraints and lack of effective pedagogical materials prevent them from doing so. By deploying our modules in a variety of educational contexts, refining the material based on the results of those experiences, conducting assessments to validate the efficacy of the materials, and providing instructors with clear guides regarding the use of these modules, we hope to build a suite of security tools that instructors will find worthwhile.

Future goals include expansion to additional undergraduate classes, including software engineering, human-computer interaction, and systems analysis and design. Modules with programming or system-specific content can be translated either into additional languages such as Python or Perl for CS0/CS1/CS2, or to different platforms for database and web development classes as needed. Revision of assessment materials is expected to be ongoing. All materials, including modules, objectives statements, and assessment tools, will be available at http://www.towson.edu/cosc/securityinjections.

Acknowledgments: This research was supported by NSF CCLI Grant DUE-0817267. Thanks to AC Chapin, Patricia Gregory, Jack McLaughlin, Claude Turner, and our colleagues for their valuable contributions and feedback.

IX. References

[1] R. Seacord, *Secure Coding in C and C++.* Addison-Wesley, Upper Saddle River, NJ (2006).

[2] J. Viega and G. McGraw, *Building Secure Software.* Addison-Wesley, Boston, MA (2002).

[3] B. Taylor and S. Azadegan, Moving Beyond Security Tracks: Integrating Security in CS0 and CS1. *Proceedings of the 38th SIGCSE technical symposium on Computer Science Education*. Portland, OR, (2008).

[4] S. Azadegan, M. Lavine, M. O'Leary, A. Wijesinha, and M. Zimand, An undergraduate track in computer security. *Proceedings of the 8th Annual Conference on innovation and Technology in Computer Science Education*, Greece (2003).

[5] R. Westervelt, Educators see secure coding training challenges, improvements. Search Security.com http://searchsecurity.techtarget.com/news/article/0,289142,sid14_gci1346086,00.html, Accessed February 12, 2009.

[6] J. Walden and C. Frank, Secure Software Engineering Teaching Modules. In *InfoSecCD Conference '06*, Kennesaw, GA (2006).

[7] J.R. Crandall, S.L. Gerhart, and J.C. Hogle, Driving Home the Buffer Overflow Problem: A Training Module for Programmers and Managers. *National Colloquium for Information Systems Security Education*, NCISSE 2002 Conference, Seattle, WA (2002).

[8] A. Frazier, X. Yuan, Y. Li, and S. Hudson, Course Modules for Software Security. *Proceedings of the 12th Colloquium for Information Systems Security Education*, Dallas, TX (2008).

[9] V. Pothamsetty, Where Security Education is Lacking. *Proceedings of the 2nd Annual conference on Information Security Curriculum Development*, (2005).

[10] W. A. Conklin and G. Dietrich, Secure Software Engineering: A New Paradigm. *Proceedings of the 40th Hawaii International Conference on System Sciences,* Manoa, HI (2007).

[11] J. Davis and M. Dark, Teaching Students to Design Secure Systems. *IEE Security and Privacy*, Vol. 1, Num. 2, (March 2003).

[12] M Graff and K. van Wyck, *Secure Coding: Principles and Practices*. O'Reilly, Sebastopol, CA (2003).

[13] G. Hoglund and G. McGraw, *Exploiting Software: How to Break Code.* Addison-Wesley, Boston, (2004).

[14] M. Howard & D. Leblanc, *Writing Secure Code*. Microsoft Press, Redmond, WA (2003).

[15] C. E. Irvine, S. Chin and D. Frincke, Integrating Security into the Curriculum. *IEEE Computer*, pp. 25-30., (Dec. 1998).

[16] L.F. Perrone, M. Aburdene, and X. Meng, Approaches to undergraduate instruction in computer Security. *Proceedings of the American Society for Engineering Education Annual Conference and Exhibition*, (2005).

[17] Epstein, J. 2004, "Comair cancels all flights 25 December", Comp.RISKS 23(63) http://catless.ncl.ac.uk/Risks/23.63.html#subj2 (Accessed April 8, 2009).

[18] National Science Foundation Research and Evaluation on Education in Science and Engineering (REESE) Program Solicitation NSF 07-595, http://www.nsf.gov/pubs/2007/nsf07595/nsf07595.htm, retrieved February 12, 2009.

[19] S. Christey, 2009 CWE/SANS Top 25 MostDangerous Programming Errors, http://cwe.mitre.org/top25/, Accessed 13 February 2009.

[20] SANS Institute, New Report Identifies the Three Programming Errors Most Frequently Responsible For Critical Security Vulnerabilities and Security Incidents in 2006, retrieved February 12, 2009 from http://www.ssi-sans.org/resources/top_three.pdf.

[21] Open Web Application Security Project, Top 10 2007, http://www.owasp.org/index.php/Top_10_2007, Accessed February 13, 2009.

[22] Open Web Application Security Project, OWASP WebGoat Project, http://www.owasp.org/index.php/Category:OWASP_WebGoat_Project, Accessed February 13, 2009.

[23] K. Keenan, *Cryptography in the Database: The Last line of Defense*. Addison-Wesley, Boston, MA (2005).