Teaching Secure Coding- The Myths and the Realities

Matt Bishop, U.C. Davis
University of California at Davis
Davis, CA
bishop@cs.ucdavis.edu

Elizabeth Hawthorne
Union County College
Cranfield, NJ
ehawthorne@acm.org

Kara Nance
University of Alaska Fairbanks
Fairbanks,AL
klnance@alaska.edu

Blair Taylor (moderator)
Towson University
Towson, MD
btaylor@towson.edu

1. SUMMARY

Teaching secure coding has never been more important. As attacks on financial, medical, government, and critical infrastructure systems increase in number and severity, there is a need for curriculum that prepares all computer science graduates to design and implement secure software. Accordingly, the CS2013 Ironman draft has added Information Assurance and Security as a new Knowledge Area in undergraduate computer science education and recommended that security content be cross-cutting across all curricula. In 2010, the Summit on Education in Secure Software provided a series of recommendations to facilitate secure coding education, including: 1) increasing the number of faculty who understand the importance of secure programming principles, and will require students to practice them; 2) integrating computer security content into existing technical and non-technical courses to reach students across disciplines; and 3) using innovative teaching methods to strengthen the foundation of computer security knowledge across a variety of student constituencies.

In this panel, we will speak to these recommendations and the new curricular guidelines outlined in CS2013 and discuss the importance and challenges of teaching secure coding. The panelists have been active participants in security education and will share their experiences teaching secure coding.

Categories and Subject Descriptors

D.2.4 [**Software**]: Software/Program Verification - *reliability*

K.3.2 [**Computers and Education**]: Computer and Information Science Education – *computer science education*

General Terms

Reliability, Security

Keywords

Secure Coding

2. MATT BISHOP

Teaching robust programming, or how to write programs that are reliable, in the sense that they perform the tasks they are supposed to, and handle any unexpected inputs or events in a reasonable manner, has several myths.

Myth #1. There is no room in the curriculum for a course on secure programming. This statement assumes that a separate course is required. But introductory and second programming classes teach the basics of secure programming: how to check for bad inputs, to do bounds checking, check return values, catch and handle exceptions, and so forth. When students program for more advanced classes, the focus is on the class material and not on the mechanics of programming. That students will write secure programs is assumed, and rarely checked.

Myth #2. If students learn to write secure programs, the state of software and system security will dramatically improve. Programs rely on operating system, library, and other services. Further, system security relies on systems being set up and configured as required *for the particular environment in which the system is used*; the size of the gap between this and practice is unknown, but probably large. Companies must also provide support for the use of these skills. Whether organizations that develop software and systems will be willing to pay this price *in practice* is unclear, as is whether customers will be willing to pay higher prices, and endure longer development times.

Myth #3. Academic institutions are hierarchical educationally. Few academic institutions allow a university president, chancellor, or dean to require faculty to teach a specific topic, and how to teach it. The best ideas and methods of learning and teaching are developed through trying different ways, and seeing which works best. There may be no "best way" to do something; often, several different ways work equally well.

Myth #4: We know what to do and how to do it. We don't know how to teach secure programming. We have ideas, but we *do not know what will work, and when.*

3. ELIZABETH HAWTHORNE

Software is an intrinsic part of our personal lives. Nearly every facet of modern society depends on complex software systems: business, energy, transportation, education, communication, government, and defense communities. The Strawman release of Computer Science Curricula 2013 (CS2013) recognizes the urgent need to teach secure coding in the undergraduate computer science curriculum [2]. CS2013 includes a new, dedicated Knowledge Area (KA) on *Information Assurance and Security* (IAS). The IAS KA is comprised of several knowledge units with one specifically on secure software design and engineering. Other knowledge units distributed throughout the proposed CS2013 Body of Knowledge (Bok) - especially in the *Software Development Fundamentals* and *Software Engineering* KAs – also address secure coding practices, defensive programming techniques, and software assurance methodologies. The next iteration of CS2013, called Ironman, is scheduled for public release and comment in February 2013. This panel will highlight the places secure coding topics and learning outcomes appear in Ironman and provide the opportunity for community feedback and input.

According to CERT, software assurance is an important discipline to ensure that software systems and services function reliably and securely. In September 2011, the Software Engineering Institute at Carnegie Mellon University published a technical report entitled, "Software Assurance Curriculum Project Volume IV: Community College Education" [3] sponsored by the U.S. Department of Homeland Security (DHS) National Cyber Security Division (NCSD). This report focuses on community college courses for software assurance and includes a review of related curricula, outcomes and body of knowledge, and outlines for six courses.

4. KARA NANCE

Many faculty members are overwhelmed with the thought of preparing original materials that demonstrate and emphasize the importance of secure coding. Further, creating a lab environment to allow students to gain hands-on experience with the potential effects of insecure coding can be challenging and may require resources and isolated environments that are not available at many institutions. The RAVE Project, funded by NSF, provides a remotely accessible virtualized environment that can be leveraged by instructors to demonstrate these effects in a safe and easy-to-use environment. The shareable instructional materials available through the RAVE Project have been developed through an ever-growing cadre of CS educators and facilitate easy inclusion of these important foundational concepts in a wide range of CS courses. Instructors can utilize these resources in a number of ways -- from setting up their students with accounts in the RAVE (if local resources are limited) -- to adapting the materials for use in a local environment. The emphasis of the RAVE Project is on sharing resources to minimize duplication of effort as instructors with common educational objectives work together.

5. BLAIR TAYLOR (moderator)

To adequately prepare computer science graduates for current and future cybersecurity challenges, security and secure coding can no longer be elective or relegated to a track for select students. All computing students should learn secure coding, starting with their first programming course, and security principles should be reiterated throughout the computer science curriculum. The challenges to security integration are significant, and include: faculty lacking security skills; programming textbooks that include insecure coding examples; and courses that leave little room for additional topics.

To help address these issues, the Security Injections @ Towson project (www.towson.edu/securityinjections) includes over 30 security injection modules, available to all instructors. (This work is supported by NSF – 0817267) S*ecurity injections* are security-related modules which address top security concerns, including integer error, input validation, and buffer overflow, that are particularly relevant for introductory programming courses. Modules are available at varying levels for CS0, CS1, and CS2, and in a variety of languages including Java, C++, pseudocode and Python. Each module is self-contained and can be easily inserted into the course with no additional instruction required by the faculty. Modules have been tested across a variety of two and four-year institutions and one HBCU. Additional modules are also available for the Computer Literacy course (for non-majors), Database, Networking, and Web Development.

Security injections allow faculty to easily incorporate important security topics that are often not well-addressed in textbooks. Teaching undergraduates to validate input and avoid integer errors and buffer overflow goes a long way towards producing more secure code, as these errors lead to a large number of the top vulnerabilities. Another important goal of the security injections is to "create a security mindset" amongst faculty and students, which many security experts consider to be one of the most important strategies towards improving security. Each security injection module includes a "Code Responsibly" section which includes guidelines for writing code that is robust and secure.

Through this project, we have integrated secure coding concepts in CS0, CS1, and CS2 to over 1000 students; coached faculty at two and four year institutions to implement the security injections; and piloted a "build-a-lab" workshop to grow the modules library and foster engagement. The next step, recently funded by NSF - 1241738, is to expand the build-a-labs and create a community of "security ambassadors" to expand security integration at more institutions. **Blair will serve as moderator of the session.**

6. REFERENCES

[1] Burley, D and Bishop, M. 2011. Summit on Education in Secure Software Final Report. http://nob.cs.ucdavis.edu/bishop/notes/2010-sess/2010-sess.pdf

[2] ACM/IEEE-CS 2013 Joint Task Force on Computing Curricula. (2012, February). *Computer Science Curricula 2013 Strawman Draft*. Retrieved from www.cs2013.org/strawman-draft/cs2013-strawman.pdf

[3] Mead, N., Hawthorne, E. and Ardis, M. *Software Assurance Curriculum Project Volume IV: Community College Education*. Retrieved from http://www.sei.cmu.edu/library/abstracts/reports/11tr017.cfm