Proceedings of the 16th Colloquium for Information Systems Security Education
Orlando, FL June 11-13, 2012

Secure Coding Education: Are We Making Progress?

Kara Nance, Brian Hay, *University of Alaska Fairbanks*, and Matt Bishop, *University of California at Davis*

Abstract – While the necessity of ensuring that secure coding practices are universally taught and adopted is becoming increasingly apparent, there is still debate over whether we are making significant progress in this area. This paper recalls the accomplishments of the first Secure Coding Workshop in 2008 and discusses some of the outcomes, challenges, and findings from that workshop. It then discusses the 2011 Summit on Secure Education, which explored some of the issues raised at the Secure Coding Workshop. It also discusses some of the follow-on activities that the workshop helped to inspire or promote, and some remaining objectives that are still presenting challenges in the ongoing pursuit of secure coding.

Index terms – Secure coding, computer security, computer education

I. Introduction

While current computer science (CS) programs are adept at teaching programming skills including exposing students to languages commonly used in industry, the focus is often on "making programs work". Students are typically given an assignment with a set of functional goals such as to create a program that reads records from a file, and then performs some calculation based on the values retrieved. In such cases little consideration is given to secure programming issues, and as such students do not learn how to write programs that would be resilient to accidentally or maliciously malformed input in real world conditions.

For example, suppose that the number of records in the file is specified in the file header, and that the application must allocate sufficient space to load all records. Such an application is easy to write if programmer makes the assumptions that the header will always be accurate and well-formed. However, in reality these assumptions are probably not valid. Consider the implications for the program if the number of records value in the file header was very large, zero, negative, or any value not equal to the actual number of records in the file. It is far more challenging to write a program that can handle these cases.

If practical lessons such as this are easy to incorporate into a general computer science education; and these skills increase the marketability of programmers; and help protect society against attacks that exploit vulnerabilities in code, why is the practice not included as a part of the required curriculum?

A word about terminology will clarify much of the following discussion. Security per se has no definition; instead, a set of rules known as a security policy defines that term in a particular context. Thus, "secure programming" should refer to a style of programming that produces code satisfying stated security requirements. But in practice, the term is used to refer to programs that avoid generic problems like buffer overflows or failure to validate inputs properly. A more accurate term for this style of programming is robust programming or defensive programming. In this paper, though, we use the term "secure programming" for robust or defensive programming to conform to the more common usage.

II. Background

A group of educators and industry representatives met at the Secure Coding Workshop in April 2008, with the goal of addressing the issue of integrating secure coding practices into current academic curricula. The initial interactions at the workshop demonstrated how important it is to draw the distinction between programming security functionality and secure programming. The former involves activities such as writing cryptographic or auditing functionality, and is typically the domain of a relatively small number of programmers with extensive training and specialized knowledge. The later impacts all programmers and all code, and is the cause of many security vulnerabilities in programs.

The industry representatives felt that it was imperative that all programmers have a knowledge of secure programming techniques, and were concerned about the amount of time and effort they currently spend training new CS graduates in what they considered to be elementary secure programming concepts. These concepts included the ability for programmers to "think like attackers" when writing code, how and why to validate input (and what constitutes "input"), and how to identify and address possible overflows and underflows in various data types (including arrays and numeric types). Some concern was raised that while these needs were being communicated at forums such as this, that little or no emphasis on these skills was evident in the industry job postings, which many students use as a guide for current industry requirements and many academic

ISBN 1-933510-95-1/$15.00 © 2012 CISSE

departments use through Industry Advisory Boards to help shape and guide curriculum.

The group also identified several challenges in the addition or integration of secure programming material into the CS curriculum. While secure programming is of great interest to the workshop participants, this is not necessarily true for all faculty members, nor is it the only topic competing for inclusion in the CS curriculum. Curricula are already extremely full and finding a place to add material is very difficult, even supposing that a faculty consensus can be reached on which material to add.

The addition of secure programming techniques to existing assignments, or the replacement of existing assignments with new exercises which include secure programming requirements is also an area that presents some challenges. Many of these assignments are designed to allow students to focus on a particular concept, and any attempt to integrate other requirements must be undertaken carefully to ensure that the original goals at the core of the assignment are not negatively impacted. As such, there is little point in teaching students to program securely when they fail to grasp the myriad of other important concepts that must also be taught to CS students.

While the workshop participants all had an interest and experience in security and secure programming, this is not necessarily true of the wider body of instructors and faculty members which also presents a challenge to the integration of secure programming practices throughout the current curriculum. Students can often take security specific classes which are taught be faculty with expertise in that field. However, such classes tend to produce students who are suited to programming security functionality, but generally don't address a wide enough population to ensure that all CS students have a baseline level of secure programming knowledge. As such, exercises must be designed in such a way that instructors throughout the curriculum can easily integrate secure programming material in their classes, whether or not they posses extensive security expertise.

III. Goals

As a group, the participants defined the following specific goals:

1. Identify areas in the CS curriculum where secure programming concepts or exercises could be introduced.
2. Develop and exchange exercises that promote secure programming practices.
3. Identify methods for disseminating secure programming materials to the wider academic community, and in particular to those instructors who do not currently have a strong secure programming or security background.
4. Identify areas in which industry can assist academia in educating students in secure coding practices.

IV. Outcomes

Three curricular areas were identified with the goal of creating and sharing exercises appropriate to various levels in the CS curriculum. These categories included introductory classes, web programming, and software engineering. (Operating systems was also debated as a potential topic, but not selected for a focus area.) The potential for teaching secure programming concepts in each of the selected curricular areas is discussed in the following section.

A. Introductory Classes

This section focused on the CS1 and CS2 classes - the first two classes in the CS major and CS0, which equates to the pre-major programming class offered by many institutions. At this level, students are being introduced to the core CS concepts, and much of the time is spent on programming assignments in languages such as C++ or Java. Students in these classes are often expected to have some limited programming experience, either from a CS0 class or high-school. It is in these classes that students are exposed to good programming practices including the use of internal and external documentation, code formatting, testing strategies, and the development of problem solutions prior to programming. The inclusion of secure programming practices at this stage will ensure that students do not have to "unlearn" insecure practices; however, care must be taken to ensure that assignments and lectures remain tightly focused on the primary goal of these classes, which is to introduce students to the fundamental tools for writing computer programs to solve problems.

There are, however, several avenues in which secure programming can be introduced in these classes without overly burdening the instructor or the students. The first approach is to introduce secure programming in the context of program robustness. While many programs written and demonstrated in these classes are trivial, they often incorporate many of the issues that cause problems in more realistic programs, such as user input and mixed data types. For example, an assignment may require students to write a simple phone book application, where the user is asked to enter a name and phone number for several users. A robust version of such a program may check to ensure that a valid phone number and name is entered. Students can then discuss what "valid" means in

ISBN 1-933510-95-1/$15.00 © 2012 CISSE

this context, and write the code to check for and responds appropriately to valid and invalid input.

Another function of these early classes is to help the students develop testing strategies, which often initially involve demonstrations that programs work with valid input. However, as the students develop expertise, their approaches to testing can be directed to include demonstrations that invalid input is also handled correctly, with emphasis on choosing good test cases that provide coverage of a wide range of possible inputs.

Lecture material in these introductory classes can also include demonstrations and discussions of the extent to which common secure programming issues (e.g., failure to properly validate user input) can have catastrophic consequences, such as the ability to run arbitrary code on a system, or bypass authentication at a website as a result of carefully malformed input. While few students are likely to fully grasp the details of such vulnerabilities at this point, the more important point to relay at this stage is that a failure to use secure programming techniques can be more serious than the program just crashing.

B. Web Programming

Many students are exposed to web development as they experiment with programming during their university careers, and often manage web servers and even write web based applications. This is an area where secure programming techniques are critical, as access is not limited to local users but is typically available to remote users from anywhere on the Internet. This may be the first environment they encounter in their programming careers in which the specter of malicious user is very real, and as a result the failure to use secure programming techniques is likely to result in either a compromise of their server or of the clients that connect to it, depending on the attack used.

In many cases the underlying concepts are no different from other environments. For example, validation of input is vital in the web environment, but the much of the problem lies in carefully identifying what input may be problematic. For example, a site which accepts user's comments, which are then displayed to other users must take care that input that could be interpreted by the browser (such as HTML tags or JavaScript) is handled very carefully.

Exercises applicable to this environment range from very short and simple "servers" which have limited functionality but demonstrate some common issues with web programming to advanced fully featured environments such as Web Goat.

C. Software Engineering

While software engineering generally is introduced later in the curriculum, it does provide an excellent opportunity to discuss the "process" of secure coding, and the techniques that can help support secure coding. Three areas that were suggested for introduction of secure coding concepts into software engineering included case studies, code review, and version control.

Arguably, the field of software engineering evolved as the result of failed large projects and these projects provide a rich arena to demonstrate the importance of secure coding. Development of a pool of case studies in this area would provide information on secure coding while allowing students the opportunity to learn from the mistakes of others.

1. Code Review

Code review is an important part of the effort to produce secure code. Manual code reviews can be useful in validating (or invalidating) the assumptions made by the original developer, and such reviews can be conducted within the scope of almost any class in the CS curriculum by swapping assignment solutions between students and having them review each other's code.

Automated tools can also help in this effort, and it can again be a very interesting exercise to expose students to these tools, either on their own code or on code written by others. These tools typically provide extensive reports, but the results generally require careful review by a human to eliminate false positives, and in some cases to understand the reasons for real security problems identified by the tool.

2. Version Control

There are, of course, many reasons to use source code control systems (SCC) in the normal course of business. From the perspective of secure coding, the use of SCC provides some specific advantages:

- The ability to easily determine what changes were made between software versions, and at what point an exploitable section of code may have been inserted or removed. For example, if version 1.53 of a particular software program was found to have an exploitable function, it can be easily determined when the code was inserted (giving a range of versions that may also be exploitable).
- The ability to determine which of a group of programmers was responsible for each line of code. This can be used to determine which team members are in need of training in secure coding

ISBN 1-933510-95-1/$15.00 © 2012 CISSE

practices (e.g., if Alice performs rigorous input validation whereas Bob does not, it is possible to provide appropriate and targeted training to rectify that situation).

D. Summary

The workshop provided examples of one way to improve the state of education in secure programming, and the participants explored several other ideas. After the workshop, several participants examined ways to incorporate secure programming into a curriculum [1, 2, 3, 4, 5, 6] through projects funded by the National Science Foundation.

The NSF Course Curriculum and Laboratory Improvement (CCLI) grants funded creation of a framework for developing materials that emphasize secure coding [1]. The framework includes background incorporating a real-life example, a problem-related security lab, a checklist to assist students in demonstrating mastery, and analysis-discussion questions which provide the opportunity to demonstrate critical thinking skills as well as providing immediate feedback to the instructors [7].

Even when the material is created, environments to support the material must be developed and disseminated. One way to do this is to deploy exercises as virtual machines, so that instructors and students can download pre-configured systems that can be safely used for secure coding exercises. One example of this is the NSF-funded SEED materials available at no-charge [3]. These exercises utilize two virtual machines on which a large number of security-related exercises, including secure coding, can be performed.

While running local VMs is one approach to provide a wide range of students and educators with high-quality materials, another is to deploy the VMs in a remotely accessible environment and make them available to others. The SEED VMs, for example, have been deployed in the NSF-funded Remotely Accessible Virtualized Environment (RAVE) [6], and students can work through those exercises as though they were running the VMs locally. This ensures that any VM activities that might be harmful to the local workstations or networks is completely isolated within the RAVE environment. Other advantages of this approach are:

1. VM performance is equal for all students, no matter what how their local computer is configured or what resources it has.
2. The same VMs can be accessed from any network connected location, so students can start to work through exercises in class, and complete it from their home computer.
3. Instructors and students can simultaneously access the same VM, allowing assistance to be provided whether the student and instructor are in the same room or in different parts of the country.

The workshop made progress in several areas; most notably recognizing that the teaching of secure programming had to be pervasive across the computer science curriculum, and identifying ways in which industry could help improve the state of education in this area. How to do this, though, was not explored.

V. The Summit on Education in Secure Software

In 2010, the NSF funded a Summit on Education in Secure Software (SESS), with the goal of developing a set of "road maps" to help institutions develop methods of teaching secure programming. Its specific objectives - quoting from the summit report [8], were:

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.

The Summit arrived at these objectives by considering 6 groups of students. Students at 4-year academic institutions were divided into computer science and non-computer science students; community college students were a separate group, as were K-12 students, computer science professionals, and non-computer science professionals. Each group needed to learn something about computer security, although what Summit participants thought each group should know varied among the groups. These recommendations were put into "road maps", each of which had a common structure.

First came a description of what the members of the group should know; then followed a set of methods that might educate the students appropriately, and what resources would be needed to achieve the desired educational goals. A key component of the road maps lists expected and possible barriers or hindrances to meeting these goals (the

ISBN 1-933510-95-1/$15.00 © 2012 CISSE

"potholes"). Then, the road map makes specific recommendations for meeting the desired educational goals despite the existence of the "potholes". In essence, the road maps take a vision of what anyone who programs should know about writing secure programs, and views that through the different lenses of the constituent groups. The end result is to teach programmers how to write secure programs, and analyze existing programs for robustness problems; and to provide a foundation for non-programmers, so if they begin to program, they will have the proper mindset to think about secure programming. That foundation will also help them identify poorly written programs even though they may not be able to write any programs themselves.

A key observation emerging from SESS was that the environment in which the system is designed, deployed, operated, and decommissioned plays a critical role in secure programming. For example, the specific measures needed to ensure a robust program to be used on a special-purpose processor (where all inputs are known and constrained) are different than those for a program to be run on a general-purpose processor (where inputs are unconstrained). This leads to several other observations.

First, understanding security requires a holistic approach, and must be an integral part of the design and implementation rather than be added on afterwards. (This is why current "patching" mechanisms are so flawed from the security point of view.) Next, knowing the principles underlying security leads to more secure coding, as does knowing about software development frameworks. Understanding how attacks work, and being able to identify potential points of attack (the so-called "attack surface") also allows the programmer to incorporate countermeasures. This includes realizing that all frameworks have weaknesses that create problems in programs that a knowledgeable attacker could exploit. Part of secure programming is coming up with strategies and tactics to overcome these problems, and using tools to aid in the development and validation of programs.

The SESS participants recommended the following to improve the state of education in secure software [8]:

1. We need more faculty who understand both the importance of secure programming, and who will require students to practice it.
2. These faculty will need support to ensure the students do practice secure programming; this includes additional security content in textbooks or supplements, and labs or clinics to reinforce the practice of secure programming in student programs.
3. Establishing professional enrichment opportunities in this area for all educators will heighten their awareness of the need for better, more robust programming and its principles.
4. Integrating computer security content (including ways to think about security) into existing technical and non-technical courses will reach students in a much wider variety of disciplines.
5. A required computer security course should focus on technical topics for computer science majors, and on raising awareness of basic ideas and issues of computer security for non-majors.
6. The cross-discipline education will require innovative teaching methods to inculcate an understanding of computer security basics to the various constituencies.
7. Because of the lack of resources for training and education in academia, government, and industry, organizations in all three groups should encourage partnerships and collaborate on the development of curricula to meet their needs.
8. Metrics to measure the effectiveness of educational techniques in specific environments will help assess progress towards meeting the educational goals for secure programming of the institution.
9. Finally, the role of computer security professionals in key business and government policy decisions must be highlighted, so they are consulted when appropriate.

The Summit concluded that emphasizing the role software plays in our society will emphasize the importance of writing good, solid code—something that is critical to the development of high assurance systems, and to the successful, effective use of ordinary computers.

VI. Conclusion

Most introductory programming classes focus on good coding style, which—essentially—is all that secure coding is. But after that class, rarely are students required to practice those precepts; the only issue is whether the program works in the general case.

The Workshop on Secure Coding discussed incorporating exercises to demonstrate the need for, and principles of, secure programming in general computer science classes.

The Summit on Education in Secure Software provided a set of more general frameworks for teaching secure programming including identifying potential problems (the "potholes"). Its goal was to provide information and advice to institutions that wished to emphasize secure programming in their curriculum. How to implement its recommendations, and which ones to implement, will depend on the institution's goals, environment, resources, and organizations.

ISBN 1-933510-95-1/$15.00 © 2012 CISSE

Acknowledgements: The authors were supported by awards from the National Science Foundation's Directorate for Computer and Information Science and Engineering and the Directorate for Education and Human Development. Matt Bishop was supported under award CNS-1039564 via George Washington University. Kara Nance and Brian Hay were supported under award DUE-1023135. Any opinions, findings, and conclusions or recommendations expressed in this publication are those of the authors and do not necessarily reflect the views of the National Science Foundation.

VII. References

[1] Security Injections @ Towson University. Available from: http://triton.towson.edu/~cssecinj/secinj/.

[2] Garramone, V. and D. Schweitzer, "PRISM: A Public Repository for Information Security Material," in Colloquium for Information Systems Security Education(CISSE). 2010: Baltimore, MD.

[3] Du, W. and Wang, R.,. "SEED: A Suite of Instructional Laboratories for Computer Security Education (Extended Version),". In The ACM Journal on Educational Resources in Computing (JERIC), Volume 8, Issue 1, March 2008.

[4] Taylor, B. and S. Azadegan, "Moving Beyond Security Tracks: Integrating Security in CS0 and CS1," in Technical Symposium on Computer Science Education (SIGCSE), ACM, Editor. 2008, ACM.

[5] Hislop, G.W., et al., "Ensemble: creating a national digital library for computing education," in Proceedings of the 10th ACM conference on SIG-information technology education. 2009, ACM: Fairfax, Virginia, USA. p. 200.

[6] Hay, B., Dodge, R., Nance, K., "Using Virtualization to Create and Deploy Computer Security Lab Exercises," proceedings of the 23rd International Information Security Conference (SEC 2008), 8 – 10 Sept, 2008, Milan, Italy.

[7] Nance, K., B. Taylor, R. Dodge, and B. Hay. Creating Shareable Security Modules. Proceedings of the 2011 Workshop on Information Security Education. June, 2011. Lucerne, Switzerland.

[8] Burley, D. and M. Bishop, *Summit on Education in Secure Software: Final Report*, Technical Report CSE-2011-15, Dept. of Computer Science, University of California at Davis, Davis, CA 95616-8562, USA (June 2011).

ISBN 1-933510-95-1/$15.00 © 2012 CISSE