Some Thoughts on Teaching Secure Programming

Matt Bishop
Dept. of Computer Science
University of California at Davis
Davis, CA 95616-8562
email: bishop@cs.ucdavis.edu

February 2, 2012

Teaching students, developers, and others to "program securely" raises many issues, several of which I consider misconceptions or hindering, rather than helping, such efforts. This short note discusses them. It is based on my experience teaching and practicing this material for over 30 years, as a graduate student, company employee, research scientist, and a university faculty member. I teach "secure programming" in my programming and computer security classes, and have presented numerous tutorials to conferences (like SANS and USENIX), several companies, and other forums. I have also written on this topic (see for example [1–9]).

To be precise, my concern is for teaching people 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. Here, I call this "secure programming".[1]

The good reader will note I do not criticize what has, and has not, been done. That would be counterproductive at best. Efforts to blame, or foist problems off on others, generally *hinder* solving a problem. Readers who interpret or use this document to assess blame or "point fingers" misuse this note. The point of this note is to help fix the problem.

The 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. A different approach would be far more effective.

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.[2] That students will write secure programs is assumed, and rarely checked.

When checked, the effects are salutary. In my operating systems class, I had my grader (a student in the Computer Security Lab) check the students' programs. When grading the first assignment, he told the students he would deduct 20% of their score if he saw code that non-secure

[1] It's actually "robust" programming, which deals with more than security.

[2] Similarly, in industry, the *usual* focus is on shipping the product, not on the security of the programing.

again. The students complained to me; I said I thought the grader was not deducting enough. The programs for the second assignment were substantially better.

The NSA's IASP capacity-building program gave us a little funding to try a different approach: a "secure programming clinic".[3] This essentially functions like a writing clinic in law school or an English department; see Bishop and Orvis [9] for details. Our goal was to see if the idea was worth studying further. The results were quite impressive [9, p. 171] and definitely warrant further trials.

Myth #2. If students learn to write secure programs, the state of software and system security will dramatically improve. Several factors make this assertion questionable.

The cliché "a chain is only as strong as its weakest link" applies to security. Programs rely on operating system, library, and other services. Programs can check some results, but not all; so if the services fail, so will the program. In 1999, for example, the buffer overflow in the widely-used RSAREF2 library enabled the compromise of many security-based programs [12,13]. 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, even in areas of national security such as elections [14].

Companies must also provide support for the use of these skills. Writing secure programs takes far more time and care than normal programming, because one must test the programs far more than is current commercial (and governmental) practice. This will increase the cost of developing software, and lengthen the time to market. Whether organizations that develop software and systems will be willing to pay this price *in practice*, despite the long-term benefits, is unclear. Whether customers will be willing to pay higher prices, and endure longer development times, for the higher assurance software and the long-term benefits is equally unclear.

Myth #3. Academic institutions are hierarchical in organization. In no institution I know about can a university president, chancellor, or dean require faculty to teach a particular topic, and how to teach it.

The marketplace of ideas is akin to a purely (idealistic) capitalistic economy: 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. This is because one of the two products of academia, learning, depends on the students—and people have different learning styles.

In computer science and in educational methodology, the other product of academia, knowledge, depends on the scientific method: develop a hypothesis, gather data, test the null hypothesis, and reject or fail to reject it. Those who advocate a particular method of teaching need to support their assertion with data and hypothesis testing, not simply assertions that "this works". As an example, the secure programming clinic test in Myth #1 developed some hypotheses—it ***absolutely*** did not validate a claim that the clinic will work everywhere.[4] Such a claim would require refinement of the hypotheses and much broader testing. This leads us to our final myth.

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.* To date there has been little funding for projects examining this. Contrast this with the funding levels for general

[3]Fortify Software also contributed their software analysis tool. I express my appreciation to both.

[4]For one thing, the sample size was too small.

"cybersecurity awareness", "building secure infrastructure", and "building secure software engineering". None of that funding goes to testing different methods of teaching secure programming, because the sponsors do not target education.

Before we decide the problem of teaching secure programming is not worth funding, or that we know how to do it, we should fund specific projects to test different methods of teaching it, and see what works, and under what circumstances. The National Science Foundation's Summit on Education in Secure Software [10, 11] provides a basis for many different methods that can, and should be tested; it provides roadmaps for developing such pilot projects. These projects must be grounded in science, to ensure meaningful results. From these projects, we will understand how well the different methods work, the support necessary to make the programs effective, and thus can justify the changes and support that a large-scale effort will require.

Conclusion

Improving the state of software requires a concerted effort on the parts of all sectors. The above tries to provide answers to claims that, in my opinion, will hinder success in this endeavor. Ultimately, we must work together to achieve the goal of students programming securely as second nature. Perhaps what that wise old man, Benjamin Franklin, said at the signing of the Declaration of Independence applies here: "We must all hang together, or assuredly we shall all hang separately."

Disclaimer. All opinions expressed herein are those of the author. They are not the opinions of anyone who has read the document, anyone with whom the author has worked, any organization or government agency that has funded any work the author has done, or anyone else.

References

[1] M. Bishop, "How to Write a Setuid Program," *;login:* **12**(1) pp. 5–11 (Jan. 1987).

[2] M. Bishop, "Teaching Computer Security," *Proceedings of the Workshop on Education in Computer Security* pp. 78–82 (Jan. 1997).

[3] M. Bishop, *Computer Security: Art and Science*, Addison-Wesley Professional, Boston, MA, USA (Dec. 2002).

[4] M. Bishop, "Best Practices and Worst Assumptions," *Proceedings of the 2005 Colloquium on Information Systems Security Education (CISSE)* pp. 18–25 (June 2005).

[5] M. Bishop, "Teaching Context in Information Security," *ACM Journal on Educational Resources in Computing* **6**(3) pp. 3:1–3:12 (Sep. 2006).

[6] M. Bishop and M. Dilger, "Checking for Race Conditions in File Accesses," *Computing Systems* **9**(2) pp. 131–152 (Mar. 1996).

[7] M. Bishop and C. Elliott, "Robust Programming by Example," *Proceedings of the Seventh World Conference on Information Security Education* pp. 23–30 (June 2011).

[8] M. Bishop and S. Engle, "The Software Assurance CBK and University Curricula," *Proceedings of the Tenth Colloquium for Information Systems Security Education* pp. 14–21 (June 2006).

[9] M. Bishop and B. J. Orvis, "A Clinic to Teach Good Programming Practices," *Proceedings of the Tenth Colloquium for Information Systems Security Education* pp. 168–174 (June 2006).

[10] D. Burley 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, USA (June 2011).

[11] D. Burley and M. Bishop, *Summit on Education in Secure Software: Final Report*, Technical Report GW-CSPRI-2011-7, Cyber Security Policy and Research Institute, The George Washington University, Washington, DC (June 2011).

[12] CERT, *Buffer Overflows in SSH daemon and RSAREF2 Library*, CERT Advisory CA-1999-15, CERT, Pittsburgh, PA, USA (Dec. 1999).

[13] CoreLabs Research, *Buffer Overflow in RSAREF2*, CoreLabs Advisory CORE-120199, CoreLabs Research (1999).

[14] S. C. Hoke and M. A. Bishop, *Essential Research Needed to Support UOCAVA-MOVE Act: Implementation at the State and Local Levels*, Technical Report 10-197, Cleveland-Marshall College of Law, Cleveland State University, Cleveland, OH, USA (Oct. 2010).