Embedding Secure Programming in the Curriculum: Some Lessons Learned

Michael N. Johnstone

***Abstract*—Security is a focus in many systems that are developed today, yet this aspect of systems development is often relegated when the shipping date for a software product looms. This leads to problems post-implementation in terms of patches required to fix security defects or vulnerabilities. One answer is that if code were correct in the first instance, then vulnerabilities would not exist. Security is now seen as an essential part of systems development in several modern methodologies. Unfortunately, the teaching of programming secure software systems is seen as an extra or worse, an impediment to learning programming. This paper presents the case that secure programming should be the norm, rather than the exception and uses a case study to describe the experience of teaching secure programming in an Australian university. It was found that students enjoyed the challenges presented by learning secure programming and expected to use these skills in industry.**

***Index Terms*—Information systems security, secure programming, applications development.**

I. Introduction

Secure programming courses are offered at several universities around the world (notably Purdue University in the USA and the University of Birmingham in the UK) as well as in some form in the commercial environment (for example, the SANS secure coding course). It would be expected, therefore, that the quality of software systems would be increasing. This is especially important in safety-critical domains such as military, aerospace and medical systems. Unfortunately most software, according to [1], is still insecure. To make matters worse, [2] suggest that security requirements are often omitted from requirements specifications altogether, therefore secure coding practices are not likely to be implemented. Johnstone [3] points out that this is due to the tension between functional requirements (as seen by a customer) and security requirements (which often are not immediately visible). Anderson [4, p7] was far more direct when he said "Much has been written on the failure of information security mechanisms to protect end users from privacy violations and fraud. This misses the point. The real driving forces behind security system design usually have nothing to do with such altruistic goals. They are much more likely to be the desire to grab a monopoly, to charge different prices to different users for essentially the same service, and to dump risk. Often this is perfectly rational".

One answer is to educate undergraduate software engineers about the need to treat security requirements with the same level of importance as (other) functional requirements, which should flow on to later stages of the system development life cycle (especially programming). This will have little immediate effect as students are unlikely to have the maturity to see the necessity for security requirements. This is not surprising as students are still learning how to elicit, specify and validate requirements as well as learning the science and craft of programming, so increasing their cognitive load will probably not have the desired effect unless something else in the curriculum is removed.

Davis and Dark [5] suggest that the information assurance community (and presumably by extension the computer and information security communities) can learn from the method used by software engineering educators in terms of taking a holistic approach and moving from broad principles to focussed technical subjects. This is evident in the framing of the Software Engineering Body of Knowledge (SWEBOK), and in fact, Davis and Dark suggest constructing a common body of knowledge for information assurance. The result would be, according to Davis and Dark, which "Students repeatedly internalize knowledge and skills leading to, for example, reusable and safe software. Over time, students adopt best practice models as second nature."

The primary argument of this paper is that the education of undergraduates to use secure coding practices is a long-term, but essential goal. As mentioned previously, some universities are already doing this by delivering subjects that teach the theory and practice of secure coding. Such subjects usually have titles similar to "Programming Secure Software Systems", and thus are readily identifiable in the curriculum. The expected benefit is that, over time, secure coding practices will become normal practice rather than an exception, leading to a significant reduction in vulnerabilities and therefore higher quality systems. For this approach to be effective, students must understand exploits in order to see where vulnerabilities exist. This approach is problematic in that for some this is tantamount to educating the next generation of hackers. Frincke [6] and Rubin and Cheung [7] provide some interesting insights with regard to the merits of this approach as discussed in the next section.

This paper describes the issues involved with embedding secure programming in the curriculum and uses a case study from a specific university setting to illustrate the effectiveness of teaching secure programming.

II. The State of Secure Programming in Universities

Before delving into aspects of teaching secure

Manuscript received January 12, 2013; revised March 12, 2013.

M. N. Johnstone is with the Edith Cowan University Security Research Institute, Perth, WA 6027 Australia (e-mail: m.johnstone@ ecu.edu.au).

programming, it is worthwhile examining current practice in teaching computer security generally. The first issue to tackle is to decide whether there is a problem to be solved. Figure 1 makes a case for the scope of the security (or secure programming) problem. Symantec [8] indicate that since 2005 there has been a significant increase in the number of security threats reported. Whilst it is acknowledged that this figure shows threats reported, not total actual threats or the number of successfully exploited systems, it is an indicator of the size and growth rate of the problem.

An Australian perspective on the problem can be gleaned from a recent Auditor-General's information security audit which found that four out of nine state government agencies were not meeting all of the required industry security standards with respect to credit card data [9]. The same audit report noted that "on average, online credit card fraud in Australia is estimated to cause AUD$150 million worth of losses each year, with more than 662,000 fraudulent transactions reported.", which supports the premise that there is a problem to be solved.

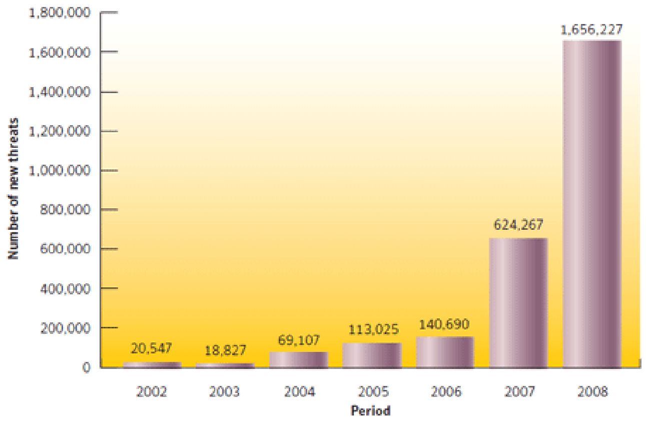

Fig. 1. Malicious code signatures (adapted from [8]).

Integrating computer (or information) security into the curriculum is not a new idea. Irvine et al. [10] point out that "Security insights must be integrated within the existing information systems programs, rather than be treated separately. The technical aspects of security are closely related to computer science and engineering."

Bishop [11] considers that the best undergraduate education serves to enable a student to be educated in broad principles and their application. It deliberately does not focus on any particular situation or system, thus facilitating the learning of general principles that can be abstracted and then applied across many situations or systems. In later work, Bishop [12] extended this view to scholarly research as applied to education. This is not surprising as most universities promote some concept of a "research informing teaching and vice versa" cycle.

Swart and Erbacher [13] point out that "Dealing with epidemic-style attacks will require focused effort in software engineering to develop secure code, but the solution will also require systems that account for the human factors that spread such attacks, including social engineering and end-user psychology".

In considering precisely what to teach in this area, Frincke [6] suggests that there are two distinct perspectives that can be taken: what she calls either "defence assurance" or "attack understanding". Secure programming tends to focus on the former but this is not a clear dichotomy. Frincke notes that most educators fall somewhere between these perspectives.

The ethical and legal minefield that is concomitant with the teaching and dissemination of the output of security research is highlighted by [7]. Academics in Australia face a similar dilemma in dealing with the Defence Trade Controls Act [14]. Regarding the latter, teaching and research activities (and their output that may potentially have a defence application) that are conducted and reported within Australia may not be exportable to other countries. In extreme cases this may mean that a lecture given to students within Australia may not be given to students in the same course at an overseas campus.

Focusing on computer security education and research in Australia, [15] outline the tertiary education landscape in Australia and go on to mention particular universities that are working in the field, viz. The University of South Australia, Queensland University of Technology, Macquarie University, Deakin University and Edith Cowan University (ECU). They remark that each university is noted for particular research areas.

Having identified the need for secure programming and examined some of the relevant curriculum issues, it is now appropriate to discuss the experience of teaching secure programming at an Australian university.

III. Teaching Secure Programming: A Case Study

This section describes some relevant details about the university in the case study, articulates the structure of a specific secure programming unit, provides some vignettes from experience in teaching the unit, describes some student feedback and concludes by analysing how the content or aims of the unit address the security education issues identified by [6].

The School of Computer and Security Science at ECU in Western Australia has a vigorous and industry-respected profile in computer and network security. The School runs several undergraduate programmes that have computer, information and network security as their main focus, namely a bachelor of computer and network security and a bachelor of cyber security. Recently the School created a third-year subject "Programming Secure Software Systems" which is a core (course requirement) subject (unit) in both degrees. The unit is also core to a mobile applications development major aimed at computer science/software engineering students.

The unit covers the major elements of secure programming such as buffer overflows, format string attacks, SQL injections, race conditions and cross-site scripting as well as code obfuscation. For the majority of these elements students were provided with PowerPoint slides, tutorial exercises, readings as well as recorded lectures. Extra support was also provided in the form of language tutorials for C, Java and PHP. This last point is particularly pertinent as will be seen later.

In order to be able to enrol in the unit, students must have already passed an introductory computer security unit as well as a fundamental programming unit. All students had also completed a Java unit and units in C programming or C++

programming. In common with many units at ECU, there are three assessment points, viz: two assignments and a final examination, a pass in the latter being a requirement to pass the unit.

In its first semester of operation last year, the unit had an enrolment of 30 students, with the majority of those being face-to-face enrolments and a small number of on-line mode only students who presumably were unable to attend classes due to schedule clashes or other commitments.

As the examples of insecure code reflected those from real-life (some were drawn from open source systems with well-documented vulnerabilities), students were required to be able to read and write code in a variety of programming and scripting languages, particularly Java, C, SQL and PHP. As all students, regardless of degree or major, are required to complete a common first year where Java and SQL are taught, it was not expected that this would present a difficult conceptual or practical hurdle for the students in the unit. This turned out not to be the case. Many students had problems with the Java and SQL questions in the major assignment. Further, some of those students relied on canned answers obtained from a downloaded "e-book" of questionable provenance and, as it transpired, even more questionable verity. It became apparent, unbeknownst to those students, that there were errors in the e-book. As that particular sub-group never questioned the authority of the e-book or bothered to test the supposed answers for themselves, this did not help with their learning outcomes.

In terms of teaching outcomes, the results varied somewhat. The highest overall mark was 96%, the lowest 2% (the latter being the result of an academic misconduct case) with a mean of 62% and a standard deviation of 20%. 28% of students failed the unit (which includes some students who enrolled but never submitted any assessment). Some 50% of the students were computer science or software engineering majors. This meant that the unit was an elective subject for them and thus they enrolled purely for interest or possibly they perceived some other value to be gained from the knowledge imparted in the unit. What was particularly interesting was the physical separation between the computer security course students and the computer science course students, which they themselves engendered in the lab classes. Without fail, in every lab session, the class would split down the middle, with the security students on one side of the room and the computing students on the other side. The reasons for this behaviour are a mystery-certainly all the students knew one another because of the mixing that occurs in the common first year. What was also observed was that the security course students had trouble with the C language questions in the assignment, despite having completed a unit using that language either concurrently with this one or in the previous semester.

The first assessment required students to find an interesting vulnerability in any system of their choice and then explain and demonstrate it (safely-see below) in-class. What was surprising was the number of hardware-oriented choices. Students took great pride in showing how they were able to exploit devices such as gaming consoles (to be expected perhaps) but also modems, printers, routers and in one case, a motor vehicle electronic control unit.

TABLE I: MATCH BETWEEN SECURITY ISSUES AND SECURE PROGRAMMING.

Security Education Issue	Addressed within ECU Secure Programming by…
1. Where do we draw the line when discussing security systems?	Given that information about the attack surface for many systems is freely available and the tools to exploit those systems are mostly free or open source, artificially making something out of bounds doesn't necessarily have the desired effect. Perhaps better to rely on good sense, a reasonable moral code and act in an ethical manner (but see point 5 below).
2. What perspective do we use when presenting our material?	Secure programming focuses on defence assurance because if secure code was written in the first place, vulnerabilities would not exist and exploits would not occur. Nonetheless, we acknowledge Frincke's [6] point about attack being potentially more attractive to students.
3. Do we describe real flaws in real systems or analyse flawed models?	Flawed models are excellent for explaining and demonstrating principles of vulnerabilities that lead to exploits. As models are simplifications of reality they are good for learning theory, however, authentic assessment is part of the unit structure, so describing and demonstrating real flaws in real systems is key to engagement and anchoring student learning. Thus the answer is both, in the right proportions.
4. Do we concentrate on proper system design, common system failures, or some mixture?	At ECU, a mixture of evaluating common vulnerabilities coupled with secure design appears to work well.
5. Do we include hands-on exercises in our classrooms, and, if so, what supervision and safety measures do we provide?	Hand-on exercises are part of the curriculum. The philosophy of the unit is based on the idea that to understand secure coding a student must first understand vulnerabilities. Students are not permitted to demonstrate anything that would contravene the ECU student charter, nor are they allowed to actively attack any University or commercial system.
6. Should we put any restrictions on who is allowed to participate in our classes and research?	Engaging students in new knowledge via state of the art research is part of the essential nature of a university, thus that the answer is "no". With the passing of the Australian Defence Trade Controls Bill in 2012, the answer is "possibly".
7. What is the pedagogical goal behind our methodology?	This is fairly straightforward and the answer may be well be the same for any university, not just the one mentioned in the case. The pedagogical goal is to enable students to learn by themselves, to engage in critical thinking and to present as useful members of the workforce upon graduation. The comments from the students suggest that the goal has been achieved, although it should be noted that some of the initial impetus for the unit came from industry feedback that said students who could not identify and correct a buffer overflow would not be hired.

Similar to most Australian universities, ECU runs on-line unit and teaching evaluation questionnaires which students may complete voluntarily and anonymously. Feedback via these instruments was, in the main, positive:

"...It was a great experiencing [sic]. I came in with a daunting feeling, however I leave (hopefully) with a thorough understanding of software security."-External student.

"I found this unit gave me a lot of new knowledge and skills which I definitely will use in the future."-On-Campus student.

"Excellent unit, just what I was looking for when finishing off my undergrad comp science course. It deals with Buffer Overflows, Smashing the Stack, SQL Injection, XSS, race conditions, string formatting and much more. All excellent tools for a programmer to be aware of."-On-campus student.

"The tutorials were very thorough and explored the unit modules in depth. The examples were excellent. Very helpful towards learning for the assignments and the exam." -On-campus student.

Some students clearly had difficulties:

"I had to do a crash course on C and PHP, as well as refresh myself in SQL to barely understand the content. The book was barely adequate to cover all topics in the unit."-On-campus student. It appears that this student failed to notice the extra materials on C for Java programmers and the links to C library functions and PHP primer provided in the University's courseware management system.

"…the first assignment was weighted 15% of the unit, but had quite a lot of work that needed to be done, where as the second assignment was 35% and didn't require much work."-External student. Unfortunately it is not possible to determine whether this student was a computer security student or a computer science major (due to the anonymised survey). Perhaps the latter as the second assignment required students to read vulnerable code and then write a secure alternative, for example, replacing a call to gets with fgets. It would be expected (but by no means proven) that the computer science students would be more comfortable with this aspect of the assessment.

Frincke [6, p56] poses some interesting questions about security education generally that provide a useful framework with which to analyse the benefits (or otherwise) of teaching secure programming. These questions (and responses from the case experience) are described in Table I.

IV. Conclusion

This paper described the implementation of a secure programming unit into an existing university curriculum. The current state of play with respect to teaching secure programming was briefly explained and a case study that detailed specific experiences was articulated and discussed.

Specifically, this work used a case study to show how effective secure programming could be when inserted into the curriculum. It was argued that the benefits of providing knowledge about vulnerabilities and how to protect against them using secure programming outweighed the costs of potentially educating the next generation of hackers. It was shown that a secure programming unit could address key issues in security education, at least for the university described in the case study. It is expected that this experience can be generalised.

Further work would involve using graduate destination data to see whether graduates are using knowledge gained in the unit in practice. It might also be possible to interview students completing a final-year work placement which would also gauge the use of the knowledge gained in the unit. Additionally, this work has focused on secure programming (i.e. defence assurance). It would be valuable to compare and contrast this experience with one based on attack understanding (i.e. ethical hacking) to examine how the approaches might complement one another.

References

[1] A. Shostack and A. Stewart, *The New School of Information Security*, Upper Saddle River, NJ: Addison Wesley, 2008.

[2] C. Wysopal, L. Nelson, D. D. Zovi, and E. Dustin, *The Art of Software Security Testing*. Upper Saddle River, NJ: Addison Wesley, 2007.

[3] M. N. Johnstone, "Security Requirements Engineering-The Reluctant Oxymoron," *Proceedings of the 7th Australian Information Security Management Conference*, Edith Cowan University, Perth Western Australia, 1st-3rd December 2009.

[4] R. Anderson, "Why information security is hard: An economic perspective," Cambridge University Technical Report. 2001.

[5] J. Davis and M. Dark, "Teaching Students to Design Secure Systems," *IEEE Security and Privacy*, pp. 56-58, March/April 2003.

[6] D. Frincke, "Who Watches the Security Educators," *IEEE Security and Privacy*, pp. 56-58, May/June 2003.

[7] B. S. Rubin and D. Cheung, "Computer Security Education and Research: Handle with Care," *IEEE Security and Privacy*, pp. 56-59, November/December 2006.

[8] Symantec. Internet security threat report. Volume XIV. Analysis of threat activity January-December 2008. [Online]. Available: http://www.symantec.com/business/theme.jsp?themeid=threatreport

[9] OAG, Information Systems Audit Report, Office of the Auditor General Western Australian Government. 2012.

[10] C. E. Irvine, S. K. Chin, and D. Frincke, "Integrating Security into the Curriculum," *IEEE Computer*, pp. 25-30, December 1998.

[11] M. Bishop, "Education in Information Security," *IEEE Concurrency*, vol. 8, no. 4, pp. 4-8, Oct. 2000.

[12] M. Bishop, "Computer Security Education: Training, Scholarship, and Research," *IEEE Computer*, Part Privacy and Security Supplement, vol. 35, no. 4, pp. 31-33, Apr. 2002.

[13] R. S. Swart and R. F. Erbacher, "Educating Students to Create Trustworthy Systems," *IEEE Security and Privacy*, vol. 5, no. 3, pp. 58-61. 2007.

[14] Commonwealth of Australia, *Defence Trade Controls Act 2012: An Act to regulate dealings in certain goods, services and technologies, and for related purposes*, Australian Government, no. 153, 2012.

[15] J. Slay and B. Turnbull, "Computer Security Education and Research in Australia," *IEEE Security and Privacy*, pp. 64-67, Sept/Oct 2006.

Michael N. Johnstone gained the MSc and PhD degrees from Curtin University in 1992 and 2008 respectively. He is a senior lecturer at Edith Cowan University (ECU) in Western Australia where he teaches secure programming and software engineering. He is a member of the Security Research Institute at ECU. His research interests include secure development methodologies (for mobile applications), wireless sensor networks (for military systems) and cloud security (for e-Health data sharing). He has been a contractor for private industry, government and research organisations and has held various roles including programmer, systems analyst, project manager and network manager before moving to academia. Dr. Johnstone is a member of the Australian Computer society.