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

Teaching Secure Software Development with Vulnerability Assessments

Bei-Tseng Chu, Will Stranathan, James Cody, Jonathan Peterson, Adam Wenner, *University of North Carolina at Charlotte*,
HuiMing Yu, *North Carolina A&T State University*

.

Abstract

Software flaws are a root cause of many security vulnerabilities found today. Empirical evidence suggests that teaching developers techniques for secure software development can significantly reduce security vulnerabilities. Unfortunately, most Computer Science curricula, including popular textbooks, have paid little attention to secure software development. In this paper we discuss possible approaches to teaching secure software development in Computing Curricula. We also share our experiences in teaching secure software development, including laboratory exercises and assignments.

I. Introduction

Software flaws are a root cause of many of today's information security vulnerabilities. Seventy-five percent of the SANS 2007 Top 20 Security Risks [15] can be mitigated by code patches. Fortunately there is empirical evidence that, with proper training in secure software development, vulnerabilities caused by software flaws can be dramatically reduced.

In early 2002, Bill Gates instructed all Microsoft developers to halt all software development and take secure software development training. For six weeks all Microsoft employees associated with software development were required to take full time training in secure coding [5, 7] and threat modeling [7,14]. In addition, Microsoft instituted a Security Development Lifecycle (SDL) [8], which is used to develop all Microsoft products. This process yielded significant reductions in security flaws in Microsoft products as illustrated in Table 1 [10].

Laboratory of Information Integration Security and Privacy, University of North Carolina at Charlotte
Charlotte, NC 28223, Department of Computer Science North Carolina A&T State University, 1601 E. Market St. Greensboro, NC 27411. *This work is supported in part by grants from the National Science Foundation DUE 0830624 and 0516085*

	Security Flaws	
Application	**Before SDL**	**After SDL**
Windows Family	119	66
SQL Server	16	0
Internet Explorer	26	17

Table 1. Security improvements in Major Microsoft products as a result of SDL

There is an emerging consensus within the technology community that the improvement of software assurance processes is critical to elevating the level of information security. Both government and industry efforts have been launched to address software assurance issues. The Department of Homeland Security (DHS) has developed a Common Body of Knowledge (CBK) for software assurance [4]. Recently Safecode.org, a consortium of prominent technology companies, was formed to promote software assurance with two white papers outlining best industry practices [12,13]. While the DHS's CBK for software assurance is comprehensive, it does not provide any guidance for prioritizing efforts. Safecode.org's best practices documents do provide a perspective on priorities. For example, the documents places much more emphasis on secure coding and other implementation techniques [14].

Unfortunately, much more effort is needed on the part of the academic community to incorporate secure software development into computing curricula. Universities train most of the software developers employed in the technology industry. Not only is a majority of faculty not trained in secure coding, textbooks used in classrooms may even, unknowingly, teach students to write code with security flaws [1].

II. Challenges

Effectively incorporating secure software development into computing curricula poses many challenges for the academic community. A National Science Foundation sponsored faculty workshop on secure software

development was held in April 2008. A number of industry representatives from SANS, Oracle, Fortify, and Symantec participated as well. The following are some of the key issues discussed:

Lack of good exercises. There was consensus among workshop participants that many more concrete exercises are needed. Such exercises should be designed to illustrate common software flaws that are vulnerable to malicious exploits and ways to correct such mistakes. It is highly desirable that exercises be demonstrated in realistic application scenarios in order to make the exercises engaging for students.

Keeping up with new attack vectors. New attack vectors are discovered regularly. For example, one prominent industry expert, Jeremiah Grossman, lists 71 significant new attack vectors for 2008 [9]. The use of new technologies and new applications of technologies are two of the top reasons for new attack vectors. This leads to a constant need for new exercises that can teach students to defend against new attack vectors. A partnership between leading industry security experts and academic faculty is a very important step toward achieving this goal.

Lack of education opportunity for faculty. The vast majority of Computer Science faculty has never been trained in secure software development practices. Both educational opportunities and incentives need to be put in place for faculty and textbook companies to take teaching secure software development seriously.

"Hacking" mentality. Industry participants stressed that there is a great shortage of developers with the "hacking" mentality, that is, developers who can think of ways to exploit vulnerabilities in software. Such skills are highly desirable for secure software development.

III. Approaches

Secure software development can be incorporated into university curricula in a number of different ways. We briefly discuss two different approaches: diffusion throughout curricula, and concentrated courses. We believe a combination of both is the best approach.

The diffusion throughout curricula model is based on the idea that many secure software development practices, such as teaching proper input validation [2], are the same practices that lead to robust and high-quality code. Therefore, these practices should be taught throughout the software development curricula, much the same way good software engineering practices should be taught. While there is a great deal of merit to this particular line of reasoning, it does have several problems.

First, vast majorities of computing faculty have never been trained in basic secure software development techniques. Training faculty to teach such techniques will not be easy. Second, it is unrealistic to expect those teaching subject matters like database administration and operating systems to be up to date on latest attack vectors. Third, there simply may not be enough class time to cover both the course topics and secure software design issues during a standard semester or quarter. For example, in a computer security course, the professor may cover principles of authentication and access control, but not have enough time to cover best implementation practices for all functions associated with authentication, such as password resetting.

The concentrated course model features a small number of focused courses teaching secure software development topics. This approach has the advantage of being able to quickly introduce this important subject into the curricula and can cover specialized as well as advanced topics. Problems with this approach include (a) if this were an elective course, students may not be exposed to the subject at all, (b) there may not be enough room in the program of study for students to take this course, and (c) secure software development techniques may not be reinforced throughout the curricula.

To fully address secure software development, a combination of both approaches is needed. First, basic concepts, such as input validation and output filtering, should be taught throughout the computing curricula, particularly in introductory programming classes. Carefully selected topics can be taught in courses focused on secure software development. In this paper, we discuss our experience in teaching secure software development with vulnerability assessment.

IV. Case study

We believe the best way to train students to have a "hacking mentality" is to teach the subject in a concentrated course. We have chosen this method of teaching for several reasons. Vulnerability assessment (aka "ethical hacking") is highly heuristic. Students learn best through hands-on practices with many realistic examples. Teaching the "hacking mentality" requires special knowledge from the instructor that is quite different from other subject-matter topics in the curricula. The subject of vulnerability assessment can be very engaging for students, as the methods taught are very different from other, more traditional computer science courses. We find that it is also very effective to teach vulnerability assessment alongside secure software development concepts because students can easily learn vulnerabilities, how to exploit them, and how to properly

fix these security flaws through secure software design and implementation.

We have been offering a course in secure software development through vulnerability assessment since Spring 2008, in both spring and fall semesters. The course assumes students have taken a three-semester hour course on web-based application development. The course also contains a unit discussion of ethics [6, 11]. Students are required to sign a participation agreement on rules to ensure ethical conduct. The course has been well received by students. It has contributed significantly to the interest in information security, has healthy and sustained course enrollment, and contributed to overall enrollment increase of the department.

It has been our experience that in order for students to become engaged with these exercises, they must look and feel like a normal application. Students are in near constant contact with professionally designed software interfaces. Web pages using default HTML fonts and formatting do not convey the required amount of realism - to the student they feel artificial.

Developing a realistic experience for the student requires some planning and balance. We tend to build pages that have the appearance of functionality but with limited depth. Non-critical links promise to take the user to other application areas but simply refresh the current page. We use application specific style guidelines for fonts, colors, and images. We also employ the "Greeking" method of using Latin-like placeholder words whenever we need the visual effect of having text under headings or in paragraphs. This is a “common technique that allows the designer and viewer to focus on the overall design instead of the actual ‘copy’ material” [16]. Figure 1 illustrates a home page of an online banking application. Extensive “Greeking” text can be found in web sites such as [18].

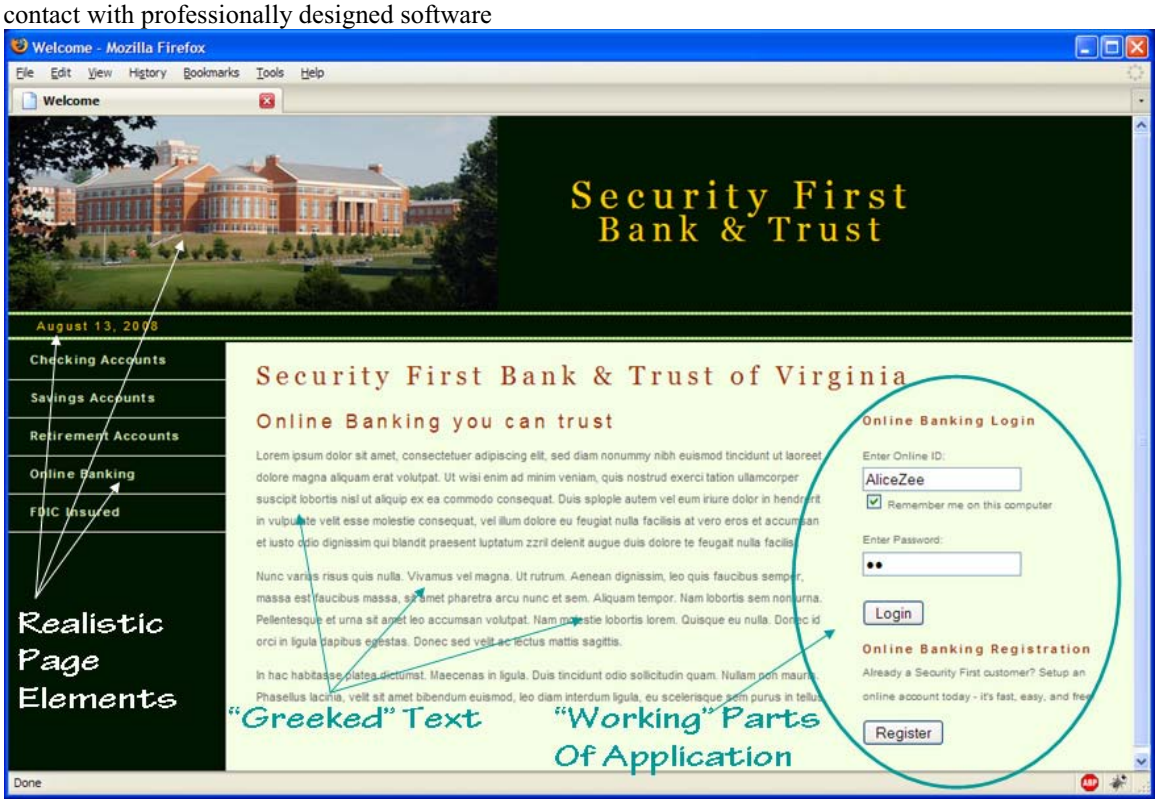

Figure 1. Using the “Greeking” method to create realistic application interfaces.

Realism extends to the actual source code. We believe that it is important for the source code to be professionally crafted, even though it has commonly seen vulnerabilities. We never highlight the flawed code because we feel it is important for the student to learn that vulnerabilities can be very hard to detect, even if you have access to well formed source code.

For the remainder of this paper, we discuss six exercises we developed for teaching vulnerability assessment and secure software development. All of the exercises were developed in Java. Each exercise is distributed with full source code and can be executed in a stand-alone environment under the Eclipse integrated development environment and a standard Java web server (e.g. Apache Tomcat). Some of them require an open source database connection. Students can use these exercises to practice

penetration testing and to learn static source analysis and the secure software development techniques needed to avoid common vulnerabilities [3]. Table 2. Summarizes the six exercises, and vulnerabilities they contain.

	Attacks & Vulnerabilities		Notes & Other Considerations
Applications	**Due to data validation**	**Due to logic error or design flaws**	
Bog: Blog Server	XSS, SQL injection	Does not check password	Used as introduction, with very obvious flaws
Tunestore: Online Music Store	Stored and reflective XSS, DOM attack, SQL injection, stealing money (payment in negative amount)	XSRF (adding friends, change password, send free gifts), broken authorization (down load music for free)	
Tickle~Mail: Web-Based Email		Poor design of password resetting, weak password, improper password challenge questions	
Tickle~Shop: E-Commerce Site		Poor implementation involving CAPTCHA, multistage login, password hints.	
NCCure Health Care Systems: HR Benefit Management		Privilege escalations due to poor design of remote management functions	
Security First: Online Banking	SQL injection, XSS, email injection	XSRF (improper fund transfer), poor error handling	Time-critical patching

Table 2. Summary of exercises and their features.

Security vulnerabilities included in these exercises can be grouped into two categories. The first category contains basic coding mistakes such as improper validation of input and improper filtering of output. Such flaws may be detected via static source code analysis. Prevention of such errors could be covered in introductory to programming classes and re-enforced through curricula. The second category includes logic errors/design flaws. Such flaws typically can only be detected via penetration testing. We believe it is more effective to cover them in the context of vulnerability assessment. Highlights for these exercises include:

Bog [17]. Modeled after standard online blogging sites, Bog looks and feels like a well-crafted blog application. But it performs no input validation whatsoever, which allows students to easily try all forms of attacks and see the result of their attacks in different parts of the application. Students are able to perform SQL injection attacks and drop the application's database, or perform cross-site scripting attacks that will steal a particular user's credentials. We use this application as an introduction to vulnerability assessment and hacking.

Tunestore. Tunestore was created to model an online music purchasing and sharing system. This application performs some input validation, stripping certain characters in a blacklist method. This gives students the opportunity to modify the attacks they learned from Bog and implement them differently to get past input filters. By obfuscating their attacks, students learn how to get past a system that may block certain attacks, but still remains vulnerable to other attacks.

Cross-site Request forgery (XSRF) is a common design flaw in many web applications where a malicious web page can take advantage of a properly authenticated session to submit malicious transactions. We demonstrated this common vulnerability inside several applications. For example, in Tunestore, an attacker can use XSRF to make other users to pay for songs he/she likes and send them to the attacker. We teach solutions to overcome XSRF attacks; however, these techniques can be defeated if the attacker employs clickjacking. We further describe ways how clickjacking may be thwarted.

Tickle. Prompting for user login and password is a commonly used security function. The simplicity of this user interaction can mask a complex set of design decisions that go into the authentication management process. Significant security issues are faced by software engineers as they design ways to create accounts, validate user credentials, and handle forgotten passwords. Authentication management is an area where developing just one student exercise would be insufficient. We developed a number of authentication related applications that all carry the Tickle name, look, and style, but with distinct authentication techniques and vulnerabilities.

Tickle~Mail. Our webmail application, Tickle~Mail, is designed to expose some of the commonly seen design flaws in authentication. Users are allowed to create email accounts with the user name of their choice, but passwords are created by the system using an algorithm that appears to be random. Users that forget their passwords have to answer challenge questions before they are allowed to reset their password.

While Tickle~Mail appears to be secure on the surface, it has a number of vulnerabilities. One of these vulnerabilities is non-technical and is demonstrated by certain "celebrity" user accounts that are pre-established in the account database. Through experimentation and online research, students are able to gain access to the celebrity accounts by finding answers to password challenge questions. At first, some of the challenge questions seem to require specialized knowledge, but with the amount of personal information available online, they pose little challenge to the student. This exercise brings out the issue of careful design of challenge questions.

Tickle~Shop. Software developers often solve a security problem by implementing a known solution or design pattern without fully understanding how it addresses the problem. This is a major issue in rapid-development projects where off-the-shelf libraries are used to piece together a product quickly. Even if a library works as intended, an incorrect implementation may lead to a security hole that will be more detrimental than the absence of the solution in the first place. This may be due to a false sense of security or an entirely new hole. As an example, Tickle~Shop is an online-store using CAPTCHA a implementations as part of its account registration function. However, the CAPTCHA answer is transmitted in a hidden HTML field, rendering the tool useless against attackers. The developer completely failed to understand a common attack vector. This faulty CAPTCHA implementation was found during our research as an incorrectly installed add-on for a well-known, open-source application.

NCCure Health Care Systems. It is common for hackers to focus on the objective of obtaining administrator credentials. Often this quest starts by establishing a user account and looking for ways to escalate these "user" privileges to "administrator". To demonstrate different ways of accomplishing this attack vector, we developed NCCure. It is a human resource management system that demonstrates a number of vulnerabilities that, when discovered and exploited in a specific manner, will allow the student to create a user account and then escalate their role so that they ultimately have complete control of all functions of the application.

V. Conclusions

With technology taking such a prominent role in our world today, demands to train software professionals who are well versed in security are expected to increase dramatically. It is crucial for academia to adopt secure software development standards into their curricula as soon as possible, and begin teaching security in software from the ground up. We have had very positive experiences in teaching secure software development through vulnerability assessment.

VI. References

[1] The 2008 SANS Awards for Finding Coding Books with Secure programming Flaws www. sans-ssi.org, July 2008.
[2] Bishop, M. “Some Exercises for an Introductory Class” in *Faculty Workshop on Secure Software Development* Orlando, FL April 2008.
[3] Chu, B., Stranathan, W., Xu, H., and Xiong, S. “Teaching Secure Web Application Development” in *Faculty Workshop on Secure Software Development* Orlando, FL April 2008.
[4] Homeland Security, “Software Assurance: A Curriculum Guide to the Common Body of Knowledge to Produce, Acquire, and Sustain Secure Software”, U.S. Department of Homeland Security, 2007.
[5] Gallagher, T. Privation communication with Tom Gallagha of Microsoft, Oct. 2008.
[6] Himma, K. *Internet Security: Hacking, Counterhacking, and Society* Jones and Barnett, 2007.
[7] Howard, M. and LeBlanc, D. *Writing Secure Code*. Microsoft Press 2003.
[8] Howard, M. and Lipner, S. *The Security Development Lifecycle*. Microsoft Press 2006.
[9] Grossman, J. http://jeremiahgrossman.blogspot.com/.
[10] Microsoft, “The Microsoft Security Development Lifecycle (SDL): Measurable Improvements for Flagship Microsoft Products” http://msdn.microsoft.com/en-us/security/cc424866.aspx. 2009.
[11] Rainforest Puppy, “Full Disclosure Policy”, http://www.wiretrip.net/rfp/policy-simple.html.
[12] Safecode.org “Software Assurance: an Overview of Current Industry Best Practices”, Software Assurance Forum for Excellence in Code, www.safecode.org, Feb. 2008.
[13] Safecode.org “Fundamental Practices for Secure Software Development”, Software Assurance Forum for Excellence in Code, www.safecode.org, Feb. 2008.
[14] Swiderski, F. and Snyder W. *Threat Modeling*. Microsoft Press 2004.

[15] SANS Institute "SANS To-20 2007 Security Risks (2007 Annual Update)" SANS Institute, www.sans.org/top20, Nov. 2007.
[16] Tullis, T. "A method for evaluating Web page design concepts", in ACM Conference on Computer-Human Interaction (CHI), pp. 323-4, 1998.
[17] Walden, J. "Web Application Security: Exercise Development Approaches", in *Faculty Workshop on Secure Software Development* Orlando, FL April 2008.
[18] Greeking website: http://www.lorem-ipsum-dolor-sit-amet.com/lorem-ipsum-dolor-sit-amet.h