Enhancement of an Undergraduate Software Engineering Course by Infusing Security Lecture Modules

Hyunju KIM, Natarajan MEGHANATHAN, and *Loretta A. MOORE
Department of Computer Science, *Division of Research and Federal Relations
Jackson State University
Jackson, MS 39217, USA

ABSTRACT

As software is used everywhere in our daily lives, the importance of developing secure software becomes more apparent. Software security is increasingly considered a software engineering problem, thus traditional software engineering curricula need to be enriched with security components. This paper reports an effort to incorporate security topics in a senior-level undergraduate software engineering course. The course was modified with respect to topics covered, course objectives, and course requirements. This paper also details specific security topics introduced into the course and their associations with traditional software engineering topics. Course assessment data and student feedback show that our enhancements have been efficient in infusing the security considerations required for secure software development.

Keywords: Software Engineering Education, Secure Software Engineering, Security Lecture Modules, Course Enhancement, Software Security Attacks

1. INTRODUCTION

As software is used everywhere in our daily lives, the importance of developing secure software becomes more apparent. The challenge is to enrich the traditional software engineering approach with security aspects, taking into account limited time and resources. Most successful attacks result from targeting and exploiting known, non-patched software vulnerabilities and insecure software configurations, many of which are introduced during design and coding [2]. Thus it is imperative that secure design and coding principles are embedded throughout the whole software development lifecycle.

This paper reports an effort to incorporate security lecture modules in a traditional undergraduate software engineering course. This effort forms part of the NSF (National Science Foundation)-funded TUES (Transforming Undergraduate Education in STEM) program on incorporating security aspects in the undergraduate Computer Science curriculum at Jackson State University.

2. BACKGROUND AND RELATED WORK

The Department of Homeland Security has established the Build Security In (BSI) project and provided useful resources through a website (https://buildsecurityin.us-cert.gov/) for incorporating security into every phase of software development. These resources have been developed according to the principle that software security is a software engineering problem and must be addressed in a systematic way throughout the software development life cycle [6]. In response to this principle, there have been several efforts to develop new software engineering modules and courses to educate computer science majors in secure software development.

A study reported in [13] developed a course module for writing secure code. The module consisted of security and insecure code concepts, safe programming practices, and simple lab exercises. It was taught in an introductory Java programming class and a programming design class for freshman and sophomore, respectively. This study showed that a course module-based approach is an effective method of educating the students about the impacts of insecure code and safe programming practices. Alternatively, the authors of [3] developed a software security learning process that used outcomes of a traditional software engineering course and developed secure software. It was an effort to develop a teaching process for secure software rather than to develop teaching materials.

Efforts reported in [4, 5, 8, 9, 11, 12] developed undergraduate or graduate software engineering courses and/or teaching materials to incorporate security topics and issues into a traditional software engineering education. The studies in [8, 11, 12] involved the development of teaching modules covering major security topics for a graduate secure software engineering course. Although the course developed in [9] was undergraduate-level, it was designed to replace the traditional software engineering course within an undergraduate computer science curriculum.

On the other hand, an effort reported in [4, 5] incorporated security studies into an existing undergraduate software engineering course. This traditional undergraduate software engineering course

was modified in three areas: instructions; lecture materials and exercises; and the semester-long project. The course adopted a combination of both facilitator and formal authority-based teaching. Along with the traditional software engineering topics, security-related topics such as the security development lifecycle model, security risk, and secure design were introduced to the course. A single, semester-long project was executed as well. The challenges identified from this effort were a lack of security teaching materials for undergraduate students and a lack of time for completing the development project within a single semester. Besides developing or enhancing software engineering courses, a study in [10] proposed security topics for secure software engineering according to the Software Engineering Body of Knowledge (SWEBOK) Guide.

These efforts demonstrate an awareness that traditional software engineering education needs to be enhanced with security components. The new Computer Science Curricula 2013 [1] reflects this particular need by introducing the new knowledge area of Information Assurance and Security (IAS). IAS covers Security Concepts, Secure Design, Defensive Programming, Threats and Attacks, Network Security, and Cryptography as its Core-Tier topics.

The traditional knowledge area of Software Engineering (SE) has been also changed: Security Risk has been added to Software Project Management, and Software Construction (secure coding practices, security enhanced programming, security problems in programming, and security considerations) has been added as two hours of Core-Tier2. In addition, the elective topic of Software Reliability from the 2001 computing curricula is now a one hour Core-Tier2 topic.

Addressing the need for secure software engineering, this paper presents security component enhancements made to a traditional undergraduate software engineering course. As part of our TUES project, the team developed two new elective senior-level undergraduate security courses: Systems and Software Security, and Advanced Information Security [7]. Besides these elective courses, we adopted security lecture modules used in these courses into our undergraduate software engineering course required by the Computer Science BS program. Consequently, all computer science majors have been educated in the essentials of secure software engineering for the past three academic years. Section 3 of this paper introduces this enhanced software engineering course. Section 4 discusses findings from our experiences, followed by a conclusion in Section 5.

3. THE UNDERGRADUATE SOFTWARE ENGINEERING COURSE

The BS program in Computer Science at Jackson State University requires students to take CSC 475 Software Engineering followed by a senior capstone project course before their graduation. While students are taking the software engineering course, they form a team of three to four members for their senior project. Each team selects a capstone project, and students are asked to complete requirements engineering and system analysis while in CSC 475. During their capstone project course, they focus more on system design, implementation, and testing.

The CSC 475 Software Engineering course has the course description: "Introduction to software engineering, software design, APIs, software tools and environments, software development processes, software requirements and specifications, software verification and validation, software implementation, software evolution, and software project management". As this description indicates, the course covers the fundamentals of software engineering for senior-level computer science majors. Prior to Fall 2010 semester, however, it did not have appreciable coverage of security-related issues.

As part of our TUES project, since Fall 2010 semester, we have (1) identified security-related issues and topics in developing software; (2) developed security lecture modules; (3) mapped security topics to relevant software engineering topics; and (4) incorporated the security lecture modules into the software engineering teaching materials. In this particular effort, our team did not intend to develop a new software engineering course. Instead, we aimed to efficiently incorporate security components in the traditional software engineering course.

3.1 Course Modification

As security topics were introduced, the course has been modified with respect to topics covered, course objectives, and course requirements. A set of security topics that are closely related to software development was identified as follows:

- Secure Software Development Lifecycle
- Principles and Models for Software Security
  - System dependability: security fundamentals
  - The 10 principles of software security
  - Bell-LaPadula confidentiality model
  - Biba integrity model
- Security Requirements
  - Functional and nonfunctional security requirements
  - Misuse cases and mitigation plans
- Security Risk Analysis
  - Risk assessment process
  - STRIDE threat model
  - Attack trees
- Software Security Attacks
  - Attack patterns
  - Dependency attacks
  - User interface attacks
  - Design attacks
  - Implementation attacks

- Testing for Software Security
    - Security functional testing: a fault model
    - Fuzz testing
    - Mutation testing
    - Run-time fault injection

These topics range from the concepts of software security to security testing methods so that students can learn security considerations and guidelines in every phase of software development. Thus, the course teaches the importance of secure software development lifecycle as a semester progresses. Our approach to infuse these security topics to the existing software engineering topics is presented in Section 3.2. The course objectives were also updated accordingly as follows:

Each student who successfully completes this course should be able to:

- CO1: Explain software process models and their characteristics and principles and models for software security.
- CO2: Understand issues in project management, including planning for software development and specify software evolution processes and issues in software maintenance.
- CO3: Apply key elements and common methods for elicitation and analysis to produce a set of software requirements, including the appropriate security-related aspects, for the chosen Senior Project.
- CO4: Select and apply appropriate design tools and guidelines in developing software.
- CO5: Specify issues in risk assessment for secure software design and explore different software security attacks with respect to dependency, user interface, design, and implementation.
- CO6: Test software, including the security aspects using software verification and validation methods.

The course objective of CO5 was newly added, and the course objectives of CO1, CO3, and CO6 were updated in order to incorporate the security components. Each student was evaluated with respect to the objectives through exams, quizzes, and project assignments. In addition, the following in-class or lab exercises were designed and used to provide students hand-on experiences and/or real-life attack examples:

- TOCTTOU (Time-of-Control-to-Time-of-Use) attack: this exercise asks students to simulate the TOCTTOU vulnerability by granting and revoking the permissions to access a text file in an Ubuntu virtual machine.
- Code injection attack: this exercise shows how code injection attacks such as SQL injection attack and Cross-Site Scripting (XSS) attacks are launched. An online auction site was setup using PHP and XAMPP. Students are asked to develop client-side input validation controls and server-side controls to prevent such injection attacks.
- Cross-Site Request Forgery (XSRF) attack: this exercise shows how the attack exploits a web site's trust in the user.
- Email tracker: this exercise shows a web bug and how it works.

Not all these exercises were required by the course in every semester: they were selectively provided to the class depending on student's background and learning pace. However, all the exercises were usually covered in the two consecutive courses of Software Engineering and Senior Project.

The course requires each student to be in a team for a capstone project. The project teams are directed to identify a project topic that has security components, for instance authentication, authorization, encryption, etc. During the requirements engineering and system analysis, each team is asked to identify and document security requirements, misuse cases, and mitigation plans for their project. The knowledge and tools taught by this course are heavily used in implementing and testing the project during the capstone project course.

3.2 A Mapping of Security Topics in Software Engineering

As mentioned earlier, our TUES project developed two, new security courses (CSC 438 Systems and Software Security and CSC 439 Advanced Information Security) as senior-level electives. During the first year of the course modification, the software engineering course invited the instructor who developed the new security courses. The invited instructor taught the security lecture modules of the topics listed in Section 3.1. However, based on student feedback, the team decided to infuse these security topics to the existing software engineering teaching materials, and a single instructor began to teach all the course topics. This made the class flow more seamless, and a consolidated set of teaching materials became available to students.

In order to infuse the security topics to the teaching materials, we identified software engineering topics that are related to these security topics. We developed a mapping of security topics in software engineering, which also includes the associations with the course objectives. Table 1 summarizes the mapping that has been implemented as teaching materials in the forms of lecture notes, project assignments, and/or lab exercises. All the security modules and teaching materials are available through our project website (http://www.jsums.edu/cms/tues) for the public and possible adoption at other institutions.

Table 1. A mapping of security topics in software engineering

Software Engineering Topics	Security Topics	Course Objectives
Software Process Models	Secure Software Development Lifecycle	CO1
Software Quality Management: System Dependability	Principles and Models for Software Security	CO2, CO5
Project Risk Management	Security Risk Analysis	CO2, CO5
Requirement Engineering: Functional, Nonfunctional, User, System, and Domain Requirements; Object-oriented Analysis: UML Diagrams	Security Functional and Nonfunctional Requirements; Misuse cases and mitigation plans	CO3
Object-oriented Design: Providing Access Control	Authentication and Authorization; Software Security Attacks	CO4
Software Verification and Validation	Testing for Software Security	CO6

Adding the security topics to the course required the course topics and their depth to be adjusted. The incorporation of security topics limited the coverage of certain elective or advanced software engineering topics in the course. Project discussions during lecture were also limited in order to allocate sufficient time for the security coverage.

4. FINDINGS AND STUDENT ASSESSMENT

As reported in the previous studies [4, 5, 8], challenges commonly faced in efforts to incorporate security into the teaching of software engineering, especially at undergraduate level, include a lack of teaching materials and a lack of time for students to complete development projects. Our approach to these problems was to develop portable security lecture modules in one or more security courses and infuse them to the existing teaching materials in the software engineering course/s. This also made it possible to easily adjust levels of coverage and depth of the course depending on the student's background and learning pace.

In relation to the implementation of security features, capstone projects have been used. Because the projects span two consecutive courses, students are able to spend more time working on security aspects and risk analysis in developing system requirements. On the other hand, the course demands that students select projects with security features, which puts a limit on available project topics. It has been challenging to identify student projects that are both reasonable and diverse.

The incorporation of security topics into the course has primarily been evaluated through faculty course assessment and self-assessment feedback surveys from students. The data and feedback indicate that students have gained knowledge on the security topics as intended. On a scale of 1-4 (1 being Poor and 4 being Excellent), students from Fall 2010 through Spring 2012 semesters evaluated their ability to incorporate security-related aspects in software development before taking the software engineering course to be 1.3 on average. After taking the course, they evaluated their ability to be 2.7 on average, which indicates that the course has contributed to their security education.

Students were also able to design and implement security modules in their senior projects. Table 2 summarizes security features that were commonly incorporated into capstone projects since the course was modified.

Unlike the most of the previous studies described in Section 2, our goal in this effort was not to develop a new secure software engineering course, but to enhance the existing software engineering course with security components. Independent guest lectures on security topics were not appreciated by students because the lectures were considered an extra burden. Therefore, there was a need to seamlessly integrate the teaching of security and the teaching of software engineering. Our approach of developing subject mapping and infusing security topics according to mapping in software engineering has been observed to be sufficient to satisfy this need.

5. CONCLUSION

We developed a mapping of security topics in software engineering and infused the security lecture modules to the undergraduate software engineering teaching materials according to the mapping. This approach required modification of the course topics, course objectives, and course requirements. Our approach has been observed to be efficient in delivering a desirable amount of security coverage according to student background and learning pace. Thus, it can be utilized as a method to introduce security components to undergraduate software engineering, with no curriculum-level changes.

In order to evaluate our activities, we collected course assessment data through faculty course assessment and feedback surveys from students. The data has shown that

the students' knowledge of software security and its related issues increased after taking the enhanced software engineering course, and they were able to effectively incorporate security modules in their capstone projects.

Table 2. Security features commonly incorporated in capstone projects

Injection attack prevention module	User input fields are protected from injection attacks with a white-list sanitization approach.
Authentication module	Security questions are asked for registration and used when the user tries to reset the password after failing to provide the correct one in a certain number of tries.
Encryption module	Confidential data such as user IDs and passwords are protected by encryption procedures at the client and server sides.
Denial of Service (DOS) prevention module	* Each user's requests are limited to a certain number during a certain period of time to prevent DOS attacks. * A registration email is sent to an email address during registration to prevent a single user from having multiple user accounts.
Automatic logout module	When the user is inactive for a certain amount of time, the application automatically logs out the user to prevent spoofing identity.
Static testing with Fortify SCA (Static Code Analyzer)	As part of testing, the code is tested with Fortify SCA.

Acknowledgement

This work has been supported through the National Science Foundation CCLI/TUES grant (Grant # DUE-0941959) on "Incorporating Systems Security and Software Security in Senior Projects". The views and conclusions contained in this document are those of the author and should not be interpreted as necessarily representing the official policies, either expressed or implied, of the funding agency.

6. REFERENCES

[1] **Computer Science Curricula 2013**, Ver. 1.0, http://ai.stanford.edu/users/sahami/CS2013/ironman-draft/cs2013-ironman-v1.0.pdf.

[2] K. M. Goertzel, "Introduction to Software Security", https://buildsecurityin.us-cert.gov/introduction-software-security.

[3] A. Hazeyama and H. Shimizu, "A Learning Environment for Software Security Education", In **Proc. of the 5th International Conference on Secure Software Integration and Reliability Improvement - Companion**, 2011, pp. 7-8.

[4] C. Y. Lester and F. Jamerson, "Incorporating Software Security into an Undergraduate Software Engineering Course", In **Proc. of the 3rd International Conference on Emerging Security Information, Systems and Technologies**, 2009, pp. 161-166.

[5] C. Y. Lester, "A Practical Application of Software Security in an Undergraduate Software Engineering Course", **International Journal of Computer Science Issues**, Vol. 7, No. 7, 2010, pp. 1-10.

[6] G. M. McGraw and N. R. Mead, "Engineering Security into the Software Development Life Cycle", **CrossTalk: The Journal of Defense Software Engineering**, Vol. 18, No. 10, 2005, pp. 4.

[7] N. Meghanathan, H. Kim, and L. A. Moore, "Incorporation of Aspects of Systems Security and Software Security in Senior Capstone Projects", In **Proc. of the 9th International Conference on Information Technology - New Generation**, 2012, pp. 319-324.

[8] R. Shumba, J. Walden, S. Ludim C. Taylor, and A. J. A. Wang, "Teaching the Secure Development Lifecycle: Challenges and Experiences", In **Proc. of the 10th Colloquium for Information Systems Security Education**, 2006, pp. 116-123.

[9] M. L. Stamat and J. W. Humphries, "Training≠ Education: Putting Secure Software Engineering Back in the Classroom", In **Proc. of the 14th Western Canadian Conference on Computing Education**, 2009, pp. 116-123.

[10] M. A. Talib, A. Khelifi, and L. Jololian, "Secure Software Engineering: A New Teaching Perspective based on the SWEBOK", **Interdisciplinary Journal of Information, Knowledge, and Management**, Vol. 5, 2010, pp. 83-99.

[11] J. Walden and C. E. Frank, "Secure Software Engineering Teaching Modules", In **Proc. of the 3rd Annual Conference on Information Security Curriculum Development**, 2006, pp. 19-23.

[12] S. S. Yau and Z. Chen, "Software Security: Integrating Secure Software Engineering in Graduate Computer Science Curriculum", In **Proc. of the 10th Colloquium for Information Systems Security Education**, 2006, pp. 124-130.

[13] H. Yu, N. Jones, G. Bullock, and X. Y. Yuan, "Teaching Secure Software Engineering: Writing Secure Code", In **Proc. of the 7th Central and Eastern European Software Engineering Conference in Russia**, 2011, pp. 1-5.