Secure Modules for Undergraduate Software Engineering Courses

Jeong Yang, Akhtar Lodgher, Young Lee
Computing and Cyber Security
Texas A&M University–San Antonio
San Antonio, TX, USA
{jeong.yang, akhtar.lodgher, young.lee}@tamusa.edu

***Abstract*—Security affects every software component in different types of computing systems. Many vulnerabilities and attacks on software systems are due to security weaknesses in the software itself. During the process of software specification, development, or testing, security issues are either taken into consideration insufficiently or not at all. Such software, due to internal weaknesses is prone to new attacks. By teaching secure software engineering techniques for designing and developing software modules, students would learn systematic secure software development techniques, such as defect detecting and security testing. This paper presents a series of modules that are designed to be integrated into undergraduate software engineering courses from a security perspective. The goal of the modules is to teach the building of robust software security requirements, secure software design and development, and secure software verification through a secure software development lifecycle.**

Keywords—secure software engineering, secure design, secure coding, security testing, security assessment

I. INTRODUCTION

The software is being used to control a countlessly large number of systems in everybody's daily life. Security affects every software component in different types of computing systems. Many vulnerabilities and attacks on those systems are due to security weaknesses in the software itself. During the process of software specification, development, or testing, security issues are either taken into consideration insufficiently or perhaps not at all. Such software, due to internal weaknesses is prone to new attacks. By using secure software engineering modules, which cover defect detecting and security testing, students will learn systematic secure software development techniques.

With significant growth in security issues and the increased importance of computer security and networks, a new Knowledge Area (KA), Information Assurance and Security (IAS) was developed and included to a set of KAs in 2013 Computer Science (CS) curricular [1]. The survey results from department chairs and directors of undergraduate studies in CS and related disciplines indicated a strong need for topics of security curricular [1, 2]. In recognizing the importance of cybersecurity, new initiatives were also established to fill the workforce demand for cybersecurity-related positions [3]. The Cybersecurity Curricular Framework developed comprehensive curricular guidance in cyber education and the Software Security (SS) KA is one of the eight core KAs in the Curricular Framework [3]. The SS KA focuses on the development and use of secure software that covers security requirements, fundamental design principles of security, implementation issues of secure coding, static and dynamic analysis and testing, and deployment and maintenance [3].

While industries started integrating security concepts with software development and development lifecycles, academies also implemented secure software engineering education. For example, researchers from the Software Engineering Institute (SEI) at Carnegie Mellon University (CMU) developed curricula that provide the guidelines for teaching secure software engineering courses for both undergraduate and graduate levels. It also provides the secure coding guidelines in programming languages such as C/C++, Java, Android, and Perl [4, 5].

A series of secure modules on various topics are being developed for a National Security Agency (NSA) grant project [6]. The goal of the project is to teach cyber security concepts in CS courses from the first introductory course to senior level courses: CS1, CS2, Secure Application Programming, Computer Security, Computer Network, Software Engineering I and II, and Cryptography. The objectives are to keep the modules complete and independent so that they can be easily integrated into CS courses. Each module package consists of PowerPoint instructions, lab exercises and solutions, and assessment methods. The modules are also designed to incorporate the National Initiative for Cybersecurity Education (NICE) Cybersecurity Workforce Framework (NCWF) topics of Cyber Threat and Vulnerabilities, Risk Management and Software Reverse Engineering [7].

This work in progress focuses on nine modules that can be integrated into undergraduate software engineering I and II courses. The goal of the modules is to teach about building robust software security requirements, secure software design and development, and secure software verification through a secure software development lifecycle. With experiences of defensive programming and secure programming from earlier courses, students will practice secure software engineering with these modules.

978-1-5386-1174-6/18/$31.00 ©2018 IEEE

Table 1. Secure Modules for Undergraduate Software Engineering I and II Courses

Module No.	*Student Learning Outcomes / Objectives*	*Hours*	*Course(s)*	
1	Describe the requirements for integrating security into the software development lifecycle	4	***Secure Software Requirement Specification***	SW I
2	Apply the risk management on a software project plan	4		SW I
3	Specify functional requirements and identifies the expected execution paths.	5		SW I
4	Apply the concepts of the Design Principles for Protection Mechanisms, the Principles for Software Security, and the Principles for Secure Design on a software development project	7	***Secure Coding Practice***	SW I
5	Describe software development best practices for minimizing risks and vulnerabilities in software development.	7		SW I/II
6	Identify security test cases of a software application	4	***Security Testing and Verification***	SW II
7	Conduct a software security testing	4		SW II
8	Conduct a security verification and assessment of a software application	5		SW II
9	Use reverse engineering tools to safely perform static and dynamic analysis of software (or malware) of potentially unknown origin	5	***SW Reverse Engineering***	SW II

Table 2. NCWF Categories, Specialty Areas, Knowledge, Skills, and Abilities for Modules

Module No.	*Categories*	*Specialty Areas*	*Knowledge, Skills, and Abilities*
1	Securely Provision (SP)	System Requirements Planning (RP) Software Development (DEV)	**Knowledge**: Of risk management processes, cybersecurity and privacy principles, cyber, system and application threats and vulnerabilities, computer algorithms, encryption algorithms, information security architecture, application security risks, evaluation and validation requirements, programming structures and logic, security system design tools, methods, and techniques, software debugging principles, software development models, systems testing and evaluation methods, secure coding techniques, **Skills**: In conducting vulnerability scans and recognizing vulnerabilities, assessing the robustness of security systems and designs, secure test plan design, conducting software debugging, determining an appropriate level of test, determining how a security system should work, systems integration testing, writing code in a currently supported programming language, writing and evaluating test plans, using code analysis tools **Abilities**: To identify systemic security issues, to tailor code analysis, to recognize vulnerabilities, to collect, analyze and verify test data, to apply cybersecurity principles, to develop secure SW to secure SW deployment methodologies
2	Securely Provision (SP)	Risk Management (RM) Software Development (DEV)	
3	Securely Provision (SP)	System Requirements Planning (RP) System Architecture (ARC) Software Development (DEV)	
4	Securely Provision (SP)	Systems Development (SYS)	
5	Securely Provision (SP)	Systems Development (SYS) Software Development (DEV) Technology R&D (RD)	
	Protect and Defend (PR)	Vulnerability Assessment & Management (VA)	
6	Securely Provision (SP)	Systems Development (SYS) Test and Evaluation (TE)	
7	Securely Provision (SP)	Systems Development (SYS) Test and Evaluation (TE)	
8	Securely Provision (SP)	Securely Provision (SP) Test and Evaluation (TE) Risk Management (RM)	
	Protect and Defend (PR)	Vulnerability Assessment & Management (VA)	
9	Analyze (AZ)	All Source Analysis (AN) Exploitation Analysis (XA)	
	Operate & Maintain (OM)	System Analysis (AN)	

II. SECURE SOFTWARE ENGINEERING MODULES

Table 1 lists the secure modules along with their student learning outcomes and objectives, and hours taken by a student to complete the modules in Software Engineering (SW) I and II courses. These modules are for the NICE NCWF curricula categories of Vulnerabilities, Risk Management and Software Reverse Engineering [7]. They fall within the listed NCWF Categories, Specialty Areas, Knowledge, Skills, and Abilities as shown in Table 2. The modules are also aligned with the essential areas of Information Assurance and Security (IAS). The IAS is one of the 18 core Knowledge Areas (KAs) that correspond to topic areas of study in computing provided by the 2013 curricula guidelines for undergraduate degree programs in CS [1].

The first five modules are designed to be taught in the SW I course. They are to teach students to build the robust software with security requirements, risk management, and secure design through a secure Software Development Life Cycle (SDLC). In the next four modules for the SW II course, students can practice secure coding with the concepts of the design principles for protection mechanisms and software security testing, using several techniques, as well as reverse engineering using tools such as disassemblers and debuggers.

By using these modules, which cover defect detecting and security testing, students will learn systematic secure software development techniques. For classroom-based teaching, the modules are accompanied by a) pedagogical instructions explaining student learning outcomes and objectives, b) lab instructions for exercising each process, c) detailed test case instructions and scenarios, and d) assessment questions to measure the learning outcomes. The modules are categorized into four main groups as described in the following: Secure Software Requirement Specification (Modules 1, 2, 3), Secure Coding practice (Modules 4, 5), Security Testing and Verification (Modules 6, 7, 8), and Software Reverse Engineering Practice (Module 9).

A. Secure Software Requirement Specification

The first three modules, 1, 2, and 3, are developed to focus on a secure Software Requirement Specification (SRS). In module 1, security at the SRS level is introduced by identifying threats, sources, defects, and security leaks. Based on the specification, detailed secure software risk management and project management plans are discussed in module 2 along with software defects detection exercises and the activities of detecting defects and security leaks in the software specifications in module 3.

The software security requirements focus on what the system should not do while the general software requirements are concerned with what the system should do. The first step for producing secure software is to gather security requirements about the malicious part of the environment and to decide how security breaches can be restricted [8]. The core security services (confidentiality, integrity, availability, authentication, authorization, and auditing) should be incorporated in this security requirements phase. It will guide students on how to gather security requirements along with Secure Functional Requirements (SFR) and later incorporate security during other phases of software development. SFR must specify services to be integrated into each functional requirement describing what should not happen while executing programs. Students will learn the practical skills necessary for performing risk assessments for their project. The ability to perform risk management is important because there could be many threats and potential vulnerabilities in the system that they develop. Through the module, students will learn to perform step-by-step risk management and map the security requirements for their project.

B. Secure Coding Practice

In the next two modules, 4 and 5, the focus is on secure coding practices. The concepts of the Design Principles for Protection Mechanisms, the Principles for Software Security, and the Principles for Secure Design on a software development project are covered in these modules. These concepts are aligned with the risk management framework and project management developed in the earlier modules. A small software system as a team project, using secure coding practice principles, may be developed, thereby covering the topics of code analysis, code review, and program coding in a sequence.

The secure coding practices here are based on the secure coding guidelines and standard rules suggested by the Software Engineering Institute (SEI)'s Computer Emergency Readiness Team (CERT) program in teaching the fundamental concepts of the Information Assurance and Security (IAS) and Defensive Programming [SE2]. From the earlier modules developed in the CS1, CS2, and Secure Application Programming courses, students are familiar with the secure coding for developing reliable and secure programs. They apply those rules and guidelines in their project implementation of the SFR. It could be good practices for minimizing risks and vulnerabilities in software development.

C. Security Testing and Verification

The next three modules, 6, 7, and 8, are developed to conduct the software security testing using different testing methods and techniques. The developed software from the previous modules is verified by using techniques for defects detecting and security testing. This is to ensure that the software satisfies the SFR and should not behave as specified, as well as the defects and security leaks identified in secure SRS. Modules 6 and 7 are developed for setting up a testing environment and building a series of security test cases. Module 8 conducts a secure software verification and assessment by defects/security leaks testing with the test cases.

Software testing methodologies usually involve testing that the final product works via its requirement specification. It

should not have undesirable side effects when used outside of its design parameters. The testing methodologies encompass everything, from unit testing for individual modules and integration testing for an entire system, to specialized testing such as security and performance. As software applications become more complex and intertwined on different platforms and devices, it is important to have a robust testing methodology for making sure that those have been fully tested.

Previously, security was mostly revealed and tested as the results from errors in the software. As the risks associated with software vulnerabilities have been raised, software applications need to be designed at the front and developed with handling security issues. Then, security testing involves software testing for confidentiality, integrity, availability, authentication, authorization, and auditing. The testing makes sure that the developed system should not behave as specified in the SFR. Then the Security Verification and Assessment of the application verifies the list of SFR specified in the SRS. Through the testing practices, students will learn how to apply Security Verification and Assessment by checking the Secure Coding Checklist [9] which is specific to that software system.

D. Reverse Engineering Practice

The last module 9 covers the practice of software reverse engineering in order to understand the obfuscated code and analyze security leaks. It uses reverse engineering tools, such as disassemblers and debuggers, to perform static and dynamic analysis of software (or malware) of potentially unknown origin. It also covers the recovery of software functional specification and ownership using the tools.

Using a disassembler is an example for the static analysis. The disassembler translates a machine language into an assembly language. Disassembly, the output of the disassembler, is formatted for human-readability, making it a reverse-engineering tool for understanding the code in analyzing security leaks. Most of the steps there involve finding information and determining its overall functionality. This module also introduces a debugger for the malware dynamic analysis to detect malware.

With this practice, students can build a strong foundation for reverse-engineering malicious software. It will teach students to learn how to perform static and dynamic analysis, how to carve malicious executables, and how to recognize common malware tactics and disassemble malicious binaries. Students also learn to examine malicious code with the help of the disassembler and debugger in order to understand its key components and execution flow, as well as further identify common malware characteristics by looking at suspicious patterns. The techniques learned in this module can be reinforced with a series of Capture-the-Flag challenges which can provide opportunities to gain practical, hands-on malware analysis skills in a fun setting.

E. Discussion

There have been efforts to frame and put the security concepts to the software engineering courses [2, 10, 11, 12]. While their detail implementations are slightly different, the objectives are the same to teach students the fundamental concepts of software engineering from a security perspective: covering the essential five knowledge areas – (Secure) Software Requirements, (Secure) Software Design, (Secure) Software Construction, (Secure) Software Testing, and Software Maintenance. Talib et al. suggested some additions to the courses: introductory definitions for software security, security threat, insider threat vulnerabilities, secure engineering activities such as the use of static analysis tools, security assurance activities such as verification and validation, risk management, etc [10]. The modules developed for this project accordingly reflect some of those suggested additions: using static analysis tools (module 9), conducting security verifications (module 8), and applying the risk management (module 2). A significant improvement on these modules is the practice of the software reverse engineering. From this practice, students can benefit to obtain the knowledge and skills of detecting malware of potentially unknown origin when analyzing security leaks.

The current curriculum of SW I and II courses will remain unchanged. The primary focus of the developed modules is to teach the core content for software engineering from a cyber security perspective. Also, the modules are stand-alone packages which can be used and easily customized by the instructor as a plug-in to the current curriculum. Thus, students will not lose content coverage. For example, the concepts of the requirements will be taught by integrating security into the software development lifecycle, and the software design concepts will be focused on the principles of software security.

III. CONCLUSION AND FUTURE PLAN

This paper presents secure modules that are designed to be integrated into undergraduate software engineering courses. The goal of the modules is to teach the student how to build robust software with the practice of the secure SRS, secure coding, security testing, and software reverse engineering. Upon completion of all lectures, labs, and assessments, the modules will be available through the NSA public library for use. Authors have a plan to teach these modules in Software Engineering I and II courses which are year-long senior level courses at their University. For each module, student performance will be measured based on the assessment questions formed. To evaluate the educational effectiveness of the modules, experiments with students will be conducted. Pre and post testing can be used to examine students' comprehension level in the practice of secure software engineering. Qualitative survey data will also be collected and analyzed to observe student's perception and awareness of the importance of securing the software.

ACKNOWLEDGEMENT

Partial support for this work was provided by the National Security Agency (NSA)'s grant project entitled "Cyber Security Modules for Core, Major and Elective Courses in the Bachelor of Science (BS) Computer Science Curriculum."

REFERENCES

[1] The Joint Task Force on Computing Curricula (2013) Association for Computing Machinery (ACM) and IEEE-Computer Society, Computer Science Curricula 2013 Curriculum Guidelines for Undergraduate Degree Programs in Computer Science, Dec 2013. Retrieved from https://www.acm.org/binaries/content/assets/education/cs2013_web_final.pdf.

[2] Yuan, Xiaohong, Yang, Li, Jones, Bilan, Yu, Huiming, and Chu, Bei-Tseng (2016), "Secure Software Engineering Education: Knowledge Area, Curriculum and Resources," Journal of Cybersecurity Education, Research and Practice: Vol. 2016 : No. 1, Article 3.

[3] Cybersecurity Curricula 2017, Curriculum Guidelines for Post-Secondary Degree Programs in Cybersecurity, Retrieved from https://cybered.hosting.acm.org/wp-content/uploads/2018/02/newcover_csec2017.pdf.

[4] Software Engineering Institute (SEI) at Carnegie Mellon University. Curricula: Software assurance Materials and Artifacts. Retrieved from https://www.sei.cmu.edu/education-outreach/curricula/software-assurance/.

[5] Software Engineering Institute (SEI) at Carnegie Mellon University. Secure Coding Standards. Retrieved from https://wiki.sei.cmu.edu/confluence/display/seccode/SEI+CERT+Coding+Standards.

[6] Lodgher, Akhtar and Yang, Jeong, "Cyber Security Modules for Core, Major and Elective Courses in the Bachelor of Science (BS) Computer Science Curriculum," National Security Agency (NSA) Grant, Sep 2017-Aug 2018.

[7] William Newhouse, Stephanie Keith, Benjamin Scribner, Greg Witte, "National Initiative for Cybersecurity Education (NICE) Cybersecurity Workforce Framework," National Institute of Standards and Technology (NIST), U.S. Department of Commerce, 2017, https://doi.org/10.6028/NIST.SP.800-181.

[8] H. Schmidt, "Threat- and risk-analysis during early security requirements engineering," In the Proc. of Availability, Reliability, and Security, ARES'10 International Conference, IEEE Computer Society, 2010, pp. 188 – 195.

[9] OWASP, Open Web Application Security Project, Application Security Verificatio Standard 3.0.1. Retrieved from https://www.owasp.org/images/3/33/OWASP_Application_Security_Verification_Standard_3.0.1.pdf.

[10] Talib, Manar Abu, Khelifi, Adel, and Jololian, Leon, "Secure Software Engineering: A New Teaching Perspective Based on the SWEBOK," Interdisciplinary Journal of Information, Knowledge, and Management Volume 5, 2010.

[11] Stamat, Michael L. and Humphries, Jeffrey W., "Training ≠ Education: Putting Secure Software Engineering Back in the Classroom," WCCCE '09 Proceedings of the 14th Western Canadian Conference on Computing Education, pp. 116-123.

[12] Ealden, James, and Frank, Charles E., "Secure Software Engineering Teaching Modules," InfoSecCD '06 Proceedings of the 3rd annual conference on Information security curriculum development, pp. 19-23.